Analysis Report for: 3C2570AF17BC799E6779D2C0A899AECF.cs

Overall Functionality

This C# code implements a ransomware program disguised as a legitimate PowerShell script executor. The program uses a PowerShell runspace to execute a Base64-encoded PowerShell script that encrypts files in a specified directory (or the C:\Users directory by default). It features command-line arguments for specifying the target directory for encryption, enabling/disabling certain features (like a "debug" mode), and a way to directly deploy the encryption script to a file. The program cleverly hides its malicious intent by presenting a seemingly normal PowerShell script execution interface.

Function Summaries

* **`Console_Info.GetStdHandle(Console_Info.STDHandle stdHandle)`**: A P/Invoke wrapper for the Windows API function `GetStdHandle`. It takes a `STDHandle` enum value (representing stdin, stdout, or stderr) and returns a handle to the corresponding standard stream.

* **`Console_Info.GetFileType(UIntPtr hFile)`**: A P/Invoke wrapper for the Windows API function `GetFileType`. It takes a file handle (`UIntPtr`) and returns a `FileType` enum indicating the type of the file (e.g., character device, disk file).

* **`Console_Info.IsInputRedirected()`**: Checks if standard input is redirected. It does this by getting the standard input handle, determining its file type, and returning `true` if it's not a character device or unknown.

* **`Console_Info.IsOutputRedirected()`**: Checks if standard output is redirected (similar logic to `IsInputRedirected`).

* **`Console_Info.IsErrorRedirected()`**: Checks if standard error is redirected (similar logic to `IsInputRedirected`).

* **`MainApp.Main(string[] args)`**: The main entry point of the program. It parses command-line arguments, sets up a PowerShell runspace, executes a Base64-encoded ransomware script, and handles potential errors.

* **`MainApp.CurrentDomain_UnhandledException(object sender, UnhandledExceptionEventArgs e)`**: An event handler for unhandled exceptions in the application domain. It throws a generic exception indicating an unhandled error.

* **`MainModule.MainModule(MainAppInterface app, MainModuleUI ui)`**: Constructor for the `MainModule` class, which implements a custom `PSHost`. It initializes the `MainAppInterface` and `MainModuleUI` instances for interaction.

* **`MainModule.PrivateData`**: Getter for private data, provides access to a `ConsoleColorProxy` for controlling console colors.

* **`MainModule.CurrentCulture`, `MainModule.CurrentUICulture`, `MainModule.InstanceId`, `MainModule.Name`, `MainModule.UI`, `MainModule.Version`**: Implementations of the `PSHost` interface, providing basic information about the host.

* **`MainModule.EnterNestedPrompt()`, `MainModule.ExitNestedPrompt()`, `MainModule.NotifyBeginApplication()`, `MainModule.NotifyEndApplication()`**: Empty implementations of methods from the `PSHost` interface.

* **`MainModule.SetShouldExit(int exitCode)`**: Sets the exit code and signals the application to exit.

* **`MainModule.ConsoleColorProxy`**: A nested class that acts as a proxy to change console colors in the UI, allowing for colored output (error, warning, etc.).

* **`MainModuleRawUI.ReadConsoleOutput()`, `MainModuleRawUI.WriteConsoleOutput()`, `MainModuleRawUI.ScrollConsoleScreenBuffer()`, `MainModuleRawUI.GetStdHandle()`**: P/Invoke methods for interacting with the Windows console at a low level, allowing for manipulating the console's buffer.

* **`MainModuleRawUI.BackgroundColor`, `MainModuleRawUI.BufferSize`, `MainModuleRawUI.CursorPosition`, `MainModuleRawUI.CursorSize`, `MainModuleRawUI.FlushInputBuffer()`, `MainModuleRawUI.ForegroundColor`, `MainModuleRawUI.GetBufferContents()`, `MainModuleRawUI.KeyAvailable`, `MainModuleRawUI.MaxPhysicalWindowSize`, `MainModuleRawUI.MaxWindowSize`, `MainModuleRawUI.ReadKey()`, `MainModuleRawUI.ScrollBufferContents()`, `MainModuleRawUI.SetBufferContents()`, `MainModuleRawUI.WindowPosition`, `MainModuleRawUI.WindowSize`, `MainModuleRawUI.WindowTitle`**: Implementations of the `PSHostRawUserInterface` interface to control the console.

* **`MainModuleUI.Prompt()`, `MainModuleUI.PromptForChoice()`, `MainModuleUI.PromptForCredential()`, `MainModuleUI.RawUI`, `MainModuleUI.ReadLine()`, `MainModuleUI.ReadLineAsSecureString()`, `MainModuleUI.Write()`, `MainModuleUI.WriteDebugLine()`, `MainModuleUI.WriteErrorLine()`, `MainModuleUI.WriteLine()`, `MainModuleUI.WriteProgress()`, `MainModuleUI.WriteVerboseLine()`, `MainModuleUI.WriteWarningLine()`**: Implementations of the `PSHostUserInterface` interface. These handle user input and output. Notably, the `PromptForCredential()` function is used, despite the password being obtained directly, suggesting an attempt to obfuscate the actual malicious actions.

Control Flow

* **`MainApp.Main()`**: This is the core of the program's execution. It follows this path:
1. Parse command-line arguments.
2. Initialize UI and main module.

3. Create a PowerShell runspace.
4. Setup event handlers for console cancellation and PowerShell error streams.
5. If input is redirected, read input from stdin and pass it to the PowerShell script.
6. Decode and execute the Base64-encoded ransomware script via PowerShell.
7. Parse the remaining command-line arguments and add them as parameters to the PowerShell script.
8. Wait for the PowerShell script to complete.
9. Handle potential errors.
10. If the `-wrant` flag is set, pause before exiting.
11. Return the exit code.

* **`MainModuleUI.Prompt()`**: This function iterates through the provided `FieldDescription` collection. If it's an array type, it continually prompts for input until an empty line is entered. Otherwise, it prompts for input based on the field's type (string, `SecureString`, `PSCredential`). The error handling is minimal, potentially leading to unexpected behavior if invalid input is provided.

Data Structures

* **`Console_Info.FileType`**: An enum representing different file types.
* **`Console_Info.STDHandle`**: An enum representing standard handles (stdin, stdout, stderr).
* **`MainModuleRawUI.CHAR_INFO`**: A struct representing a character cell in the console buffer.
* **`MainModuleRawUI.COORD`**: A struct representing coordinates in the console buffer.
* **`MainModuleRawUI.SMALL_RECT`**: A struct representing a rectangle in the console buffer.

Malware Family Suggestion

This code strongly suggests a **ransomware** program. The Base64-encoded script is the primary indicator; the code's structure and the use of obfuscation techniques (Base64 encoding, misleading command-line arguments like "-wrant" which is a variation of "-wait" making it hard for the average user to detect), further confirm this. The script's purpose is to encrypt files, a defining characteristic of ransomware. The `-extdimmt` (a variation of "-extract") option is a deceptive addition, pretending to offer a file extraction feature while potentially serving as a means to deploy the malware or retrieve encrypted files (depending on the script's functionality). The extensive error handling is weak and likely used to hide the true error during execution. The inclusion of a `-debug` option could even be a backdoor for the malware authors.