Analysis Report for: 5BDAB9FDBCF1AFDD009BA3AF8B5FAE22.vbs

Decoded using latin-1...

Overall Functionality

This C code snippet appears to be designed to convert a Base64 encoded JPEG image string into a JPEG image file, then convert that JPEG to a PNG, and finally output the Base64 encoding of the JPEG. The temporary files are created in the `C:\Temp` directory using a randomly generated GUID. This strongly suggests an attempt to obfuscate malicious activity. The code uses placeholder names like `[Convert]`, `[IO.File]`, `[system.drawing.image]`, `[system.drawing.bitmap]`, `[system.drawing.color]`, and `[system.drawing.graphics]` which are suggestive of a .NET framework or similar environment being mimicked. It is highly suspicious and likely part of a larger malware program.

Function Summaries

The code doesn't explicitly define any functions. Instead, it uses what appear to be function calls from an external library (likely a .NET wrapper or similar). Let's break down these apparent function calls:

* **`[Convert]::FromBase64String("...")`:** This presumed function takes a Base64 encoded string as input and converts it into a byte array. The "..." represents the Base64 encoded JPEG data. The return value is a byte array (`$bytes`).

* **`[IO.File]::WriteAllBytes("...", $bytes)`:** This function writes the entire contents of a byte array to a specified file. The first parameter is the file path ("C:\Temp\$($NewGuid)-PNG.png"), and the second parameter is the byte array (`$bytes`) containing the decoded JPEG data. The return value is likely void or an indication of success/failure.

* **`[drawing.image]::FromFile(...)`:** This likely reads a image from the specified file path ("C:\Temp\$($NewGuid)-PNG.png"). Returns a `drawing.image` object (`$image`).

* **`[System.Drawing.Bitmap]::new($image.Width, $image.Height)`:** This creates a new Bitmap object with dimensions taken from the loaded JPEG image. The return value is a new `System.Drawing.Bitmap` object (`$NewImage`).

* **`$NewImage.SetResolution($image.HorizontalResolution, $image.VerticalResolution)`:** Sets the resolution of the newly created Bitmap to match the original JPEG's resolution. The return value is likely void.

* **`[System.Drawing.Graphics]::FromImage($NewImage)`:** Creates a Graphics object from the Bitmap. Returns a `System.Drawing.Graphics` object (`$Graphics`).

* **`$Graphics.Clear([System.Drawing.Color]::White)`:** Clears the Bitmap to white. The return value is likely void.

* **`$Graphics.DrawImageUnscaled($image, 0, 0)`:** Draws the JPEG image onto the Bitmap, effectively copying it. The return value is likely void.

* **`$NewImage.Save("...", [imageFormat]::Jpeg)`:** Saves the Bitmap as a JPEG file. The first parameter is the file path ("C:\Temp\$($NewGuid)-JPG.jpg"), and the second parameter specifies the JPEG image format. The return value is likely void or an indication of success/failure.

* **`$image.Dispose()`:** Releases resources associated with the `$image` object. The return value is likely void.

* **`[Convert]::ToBase64String((get-content "...") -Encoding Byte)`:** Reads the JPEG file ("C:\Temp\$($NewGuid)-JPG.jpg") content as a byte array and converts it to a Base64 string. The return value is the Base64 encoded JPEG string (`$jpg64`).

Control Flow

The code executes sequentially. There are no loops. The control flow is determined by the success or failure of the file operations (which are assumed, not explicitly checked for errors). Failure is handled silently via `-ErrorAction SilentlyContinue`.

Data Structures

The primary data structures are implied, not explicitly declared:

* **Byte array:** Used to store the decoded Base64 string and the JPEG file contents.
* **`drawing.image` object:** Represents the image loaded from the file.
* **`System.Drawing.Bitmap` object:** Represents the new Bitmap created to hold the image.
* **`System.Drawing.Graphics` object:** Provides methods for drawing on the Bitmap.

Malware Family Suggestion

Based on the functionality, this code snippet exhibits characteristics consistent with several malware families, primarily those employing:

* **Dropper/Installer behavior:** The code downloads a JPEG image (via a Base64 encoded string), saves it to a temporary location, processes it, and then outputs a Base64-encoded version of the processed image. This temporary file creation and manipulation is a common technique used to install or drop other payloads. The seemingly innocuous image processing acts as camouflage.

* **Obfuscation techniques:** The use of a GUID for temporary file names, combined with the placeholder function names (mimicking .NET), is a clear attempt at obfuscation, making analysis and reverse engineering more difficult.

* **Information stealing (potentially):** While not explicitly shown here, the Base64 encoded string could contain anything, including stolen data. This code is the processing stage, after the data has been obtained.

It's difficult to pinpoint a specific malware family without the full context of the surrounding code. However, the techniques used align with many information stealers, downloaders, and droppers. The code's purpose is to handle a file and output a base64 encoded version, indicative of exfiltration rather than typical virus activity. More information about the source of the base64 string is needed to make a more definitive identification.