Analysis Report for: a.vba

Decoded using latin-1...

Overall Functionality

This VBA macro embedded in an Excel file performs several actions related to data processing and manipulation. It primarily focuses on updating and formatting data within an Excel workbook, including manipulating pivot tables. The macro dynamically adjusts the data source for pivot tables to encompass all rows returned by a query, it also copies formulas across rows and formats the data. It then creates a text file based on the processed data and deletes the intermediate data sheet. The macro includes error handling and attempts to handle different Excel versions. The presence of functions like `CreateTextFile`, `CreateObject`, and string obfuscation techniques raises significant security concerns.

Function Summaries

* **`Formatear()`**: This is the main subroutine. It orchestrates the entire data processing workflow, including data source updating for pivot tables, formula dragging, data range definition, table creation (or bordering if Excel version is older), column auto-fitting, and cleanup actions (including deleting a sheet).
* **`Marco(rango As String)`**: Adds borders to a specified range of cells. It takes the range name as input. No return value.
* **`FilaTexto(columna As String, Texto As String, Optional hoja As String, Optional iteracion As Integer) As Integer`**: Searches for a specific text within a given column and optionally sheet, returning the row number of the first occurrence or subsequent occurrences if `iteracion` is specified. Returns the row number as an integer.
* **`ConfigurarPagina(hoja As String, Optional ancho As Integer, Optional apaisado As Boolean, Optional filaTitulos As String)`**: Configures page setup for printing, allowing specification of page width, landscape orientation, and header rows. No return value.
* **`ArrastrarFormula(celda As String, Optional hoja As String)`**: Autofills a formula down a column to a specified row (`filaFin`). Takes the starting cell and optional sheet name as input. No return value.
* **`Desbloquear(hoja As String, area As String)`**: Unlocks specified cells within a sheet, allowing editing. No return value.
* **`ProtegerHoja(hoja As String)`**: Protects a specified sheet, enabling only selection of unlocked cells. No return value.
* **`CrearHoja(nomHoja As String)`**: Creates a new sheet with a given name. No return value.
* **`EliminarHoja(nomHoja As String)`**: Deletes a sheet with a given name. No return value.
* **`Rellenar(campo As String, longitud As Integer, relleno As String, izquierda As Boolean)`**: Pads a string to a specified length with a given character, either on the left or right side. Returns the padded string.
* **`InsertarFila(fila As Integer)`**: Inserts a row at a specified position. No return value.
* **`EliminarFila(fila As Integer, Optional hoja As String)`**: Deletes a row at a specified position (optional sheet name). No return value.
* **`ColorFondo(rango As String, Color As String)`**: Sets the background color of a specified range of cells. No return value.
* **`CopiarCeldas(hojaOrigen, rangoOrigen, hojaDestino, celdaDestino)`**: Copies values from a range of cells in one sheet to another sheet. No return value.
* **`FechaFinMes(fecha As Date)`**: Returns the last day of the month for a given date as a string.
* **`MoverElementoTD(hojaTabla, nomTabla, campo, elemento, Optional final As Boolean)`**: Moves a pivot table item to the first or last position within a field. No return value.
* **`CreacionFichero()`**: Creates a text file based on data from the worksheet. The filename is user-defined. No return value.
* **`OcultarHoja(nomHoja As String)`**: Hides a given sheet. No return value.
* **`EliminarFilasVacias(filaFin As Long)`**: Deletes rows containing only empty cells. No return value.

Control Flow

The `Formatear()` subroutine is the primary control flow point. It follows this general structure:

1. **Initialization**: Sets up global variables and application settings (calculation mode, alerts).
2. **Data Sheet Processing**: Identifies the data sheet, determines the number of rows and columns, hides a specific column, defines the data range, and copies formulas down. This part includes a loop to iterate through columns.
3. **Table Creation or Range Bordering**: Creates an Excel table (or adds borders for older Excel versions) based on the data range.
4. **Pivot Table Update**: Iterates through each sheet and updates the data source for any pivot tables found. This section has nested loops to handle multiple pivot tables and pivot fields. It also includes version checks for Excel compatibility.
5. **Final Formatting**: Performs additional formatting actions on a sheet named "LOTE", including cleaning rows and creating a list object.
6. **Cleanup**: Deletes the original data sheet.
7. **Error Handling**: Includes a basic error handler for error code 1004 (likely related to handling of objects or ranges).

The other functions have more straightforward control flows, typically involving simple loops or conditional statements based on their specific tasks. For example, `ArrastrarFormula` uses a `Select Case` statement to determine the column letter based on the input cell string.

Data Structures

The code primarily uses the built-in data structures of VBA, interacting with Excel's worksheet objects, ranges, and pivot tables. There are no custom user-defined data structures. Global variables are used to store information about the data ranges, sheets, and other processing parameters.

Malware Family Suggestion

While this macro doesn't directly contain malicious code like network connections or file system modifications beyond the creation of a text file, the suspicious functions (`CreateTextFile`, `CreateObject`), use of string manipulation techniques potentially used for obfuscation, along with a seemingly legitimate data manipulation procedure make this strongly suggestive of a **macro virus or a downloader**. The macro's actions could be part of a larger attack, where the generated text file is used to exfiltrate data, or as a command and control mechanism which could then download further malicious payload. The fact the oletools analysis flagged hex and Base64 strings strongly supports this hypothesis. The IOC xvars.js further strengthens the possibility of further malicious activity if this excel file is involved in a larger attack infrastructure. Further static and dynamic analysis is needed to confirm the exact nature of the threat. The obfuscation makes complete analysis difficult without additional tools.