Analysis Report for: 22.vba

Overall Functionality

The provided VBA code consists of five macros (`Arial_to_ArialMon`, `ArialMon_to_Arial`, `Danzan_to_ArialMon`, `Montimes_to_ArialMon`, `ArialMon_to_Montimes`, and `dos2arial`) designed to manipulate text within a Microsoft Word document. Each macro performs a character-by-character substitution, effectively translating text from one encoding or character set to another. The transformations appear to be designed to convert between different Cyrillic alphabets and a standard Latin alphabet (Arial) and its monospaced equivalent (ArialMon), possibly also involving a custom "Danzan" and "Montimes" encoding. The `dos2arial` macro seems to handle a specific mapping related to DOS characters and the Arial encoding. The presence of multiple nearly identical functions (`Arial_to_ArialMon` and `ArialMon_to_Arial` in both `NewMacros11.bas` and `NewMacros111.bas`) suggests redundancy and obfuscation.

Function Summaries

* **`Arial_to_ArialMon()`**: Translates text from a specific Cyrillic-like encoding (likely based on Unicode code points) to Arial Monospaced. It iterates through each character, checking its Unicode value against a mapping table, and replacing it with its equivalent Arial Monospaced character.

* **`ArialMon_to_Arial()`**: The reverse translation of `Arial_to_ArialMon()`, converting from Arial Monospaced back to the original Cyrillic-like encoding.

* **`Danzan_to_ArialMon()`**: Translates text from a custom "Danzan" encoding (represented by ASCII code points) to Arial Monospaced. This uses ASCII codes instead of Unicode codes.

* **`Montimes_to_ArialMon()`**: Translates text from a custom "Montimes" encoding (represented by ASCII code points) to Arial Monospaced. This involves a large lookup table to convert characters.

* **`ArialMon_to_Montimes()`**: The reverse of `Montimes_to_ArialMon()`, converting Arial Monospaced characters to the "Montimes" encoding.

* **`dos2arial()`**: Translates text from a character set possibly related to extended DOS characters to Arial, using a large mapping table. The name suggests a conversion from a DOS code page to Arial.

Control Flow

All six functions share a similar control flow:

1. **Initialization**: They obtain the total character count of the active document (`Max`) and position the selection to the beginning of the document. They initialize a counter `i`.

2. **Iteration**: A `While` loop iterates through each character in the document (`i <= Max`).

3. **Character Retrieval**: Inside the loop, the current character (`Char`) is obtained using `Selection.MoveRight` and `Selection.Text`.

4. **Character Mapping**: A `Select Case` statement checks the ASCII or Unicode value of the current character (`Asc`, `AscW`, or `asc_code`) against a predefined set of cases. Each case specifies a corresponding replacement character (using `Chr` or `ChrW`). For ranges of characters, calculations are performed to get the `ascii_code` or `uni_code`.

5. **TypeText**: If a match is found, the replacement character is inserted using `Selection.TypeText`. If no match is found (`Case Else`), the selection simply moves to the next character.

6. **Increment and Repeat**: The counter `i` is incremented, and the selection moves to the next character, repeating the process until all characters have been processed.

Data Structures

The primary data structures are implicit: The mapping between input and output characters is embedded within the `Select Case` statements. These could be considered large, sparsely populated lookup tables. The Word document itself serves as an external data structure.

Malware Family Suggestion

The functionality of this VBA code strongly suggests a **polymorphic obfuscator** or a component of a more complex malware. While the code itself doesn't directly perform malicious actions (like data theft or system modification), its purpose is to transform text in a way that makes reverse engineering and analysis more difficult. The multiple near-identical functions and the custom encoding schemes are clear signs of obfuscation. The macros could be used to:

* **Obfuscate Command and Control (C&C) communication:** The character substitution could encode strings used in communication with a C&C server, making network traffic harder to analyze.
* **Hide malicious payloads:** The custom encodings could disguise embedded malicious code within seemingly innocuous text.
* **Evolve malware variants:** By altering the character mapping tables, the malware can generate numerous variants, evading signature-based detection.

In summary, this VBA code is not inherently malicious but acts as a tool to hide or obfuscate other malicious code or actions, typical of advanced malware techniques. The `oletools` output correctly highlights the use of `Chr` and `ChrW` as suspicious indicators of string obfuscation. Further analysis would be required to determine if the transformed text reveals any hidden malicious content or functionality.