Analysis Report for: 97.txt

Overall Functionality

This VBA code, likely embedded within a Microsoft Word document or similar application, contains several functions designed to perform character substitutions within a document's text. The substitutions are implemented using `Select Case` statements which map specific character codes (ASCII or Unicode) to different character codes. Essentially, these functions act as simple ciphers, converting text from one encoding or representation to another. The presence of multiple functions ("Arial_to_ArialMon", "ArialMon_to_Arial", "Danzan_to_ArialMon", "Montimes_to_ArialMon", "ArialMon_to_Montimes", "dos2arial") suggests a system for encoding and decoding text using different "fonts" or character sets as a disguise mechanism.

Function Summaries

Each function follows a similar pattern: It iterates through each character in the active document, checks its ASCII or Unicode code using `Asc` or `AscW`, and replaces it with a different character according to a predefined mapping in the `Select Case` statement.

* **`Arial_to_ArialMon()`**: Converts characters from one set (likely Arial) to another (ArialMon), which seems to be a slightly different character set encoding. It uses `AscW` (wide character) suggesting Unicode input.
* **`ArialMon_to_Arial()`**: The reverse of `Arial_to_ArialMon()`, converting from ArialMon back to Arial. It also uses `AscW`. Note the typo "acsii_code".
* **`Danzan_to_ArialMon()`**: Converts characters from a set referred to as "Danzan" to ArialMon. This uses `Asc` (single-byte character), suggesting ASCII input.
* **`Montimes_to_ArialMon()`**: Converts characters from a set called "Montimes" to ArialMon. It uses `Asc`.
* **`ArialMon_to_Montimes()`**: The reverse of `Montimes_to_ArialMon()`. Uses `Asc`.
* **`dos2arial()`**: Converts characters, seemingly from a DOS-like encoding (indicated by the ASCII codes used), to Arial. Uses `Asc`.

All functions have no parameters and no explicit return value. They modify the active document directly.

Control Flow

The core logic of each function is the same:

1. **Initialization**: Get the total character count (`Max`) of the active document. Set the selection to the beginning of the document. Initialize a counter `i` and extend the selection to the first character.
2. **Iteration**: A `While` loop iterates through each character in the document (`i <= Max`).
3. **Character Processing**: Inside the loop, the ASCII or Unicode code of the selected character (`Char`) is obtained using `Asc` or `AscW`.
4. **Conditional Substitution**: A `Select Case` statement checks the character code against a series of predefined cases. If a match is found, the character is replaced with a new character specified in the `Case`. Ranges are used for some cases to handle blocks of characters. If there's no match (`Case Else`), the selection moves to the next character without replacement.
5. **Movement and Update**: The selection moves to the next character (`Selection.MoveRight`). `i` is incremented, and the loop continues.

Data Structures

The primary data structures used are implicit:

* **ActiveDocument**: The currently opened Word document (or equivalent) is the main data structure being manipulated. Its characters are accessed and modified directly through the `Selection` object.

Malware Family Suggestion

The code's functionality strongly suggests a **macro virus** or a component thereof. The functions effectively act as an encoding/decoding mechanism to obfuscate potentially malicious commands or data within a Word document. This is a common tactic to evade antivirus detection. The multiple substitution functions are also indicative of some form of polymorphic behavior, making detection more difficult as the code will appear slightly different across different runs.

The primary concern here is not the code itself but its context. If this code is found in a malicious document, its purpose is to hide the actual malicious payload through character substitution, rendering simple string-based analysis ineffective. Further analysis would be needed to understand what text is being encoded and decoded – the substitutions themselves reveal nothing malicious. But the techniques used are classic malware obfuscation.