Analysis Report for: 22.txt

Overall Functionality

This VBA macro code, found within a potentially malicious executable, contains four main subroutines (`Arial_to_ArialMon`, `ArialMon_to_Arial`, `Danzan_to_ArialMon`, `Montimes_to_ArialMon`, and `dos2arial`). These subroutines appear designed to perform character substitutions within a Microsoft Word document. Each subroutine systematically iterates through the characters of the active document, replacing certain characters with others based on predefined mappings. The mappings are hardcoded within the `Select Case` statements. The presence of multiple, similar subroutines suggests a possible obfuscation technique. The nature of the character substitutions points towards a possible encoding or decoding scheme designed to hide text or alter the appearance of the document.

Function Summaries

* **`Arial_to_ArialMon()`**: This subroutine iterates through the characters of the active Word document. If a character's Unicode value matches specific values (e.g., 1025, 1028, 1256, etc.), it's replaced with a corresponding ASCII character (defined by `ascii_code`). A range of Unicode characters (1040 to 1103) undergoes a specific transformation (`ascii_code = uni_code - 848`) before replacement. Otherwise, the code simply moves to the next character. There is no explicit return value.

* **`ArialMon_to_Arial()`**: This subroutine is the inverse of `Arial_to_ArialMon()`. It takes ASCII characters (e.g., 168, 170, 175, etc.) and replaces them with their corresponding Unicode counterparts. Similar to the previous function, there's a range transformation (`uni_code = acsii_code + 848`). No explicit return value.

* **`Danzan_to_ArialMon()`**: This subroutine works similarly to the previous two but operates on ASCII characters directly using `Asc()` instead of Unicode (`AscW()`). It replaces specific ASCII characters (e.g., 61, 45, 47, etc.) with other ASCII characters. No explicit return value.

* **`Montimes_to_ArialMon()`** and **`ArialMon_to_Montimes()`**: These subroutines perform bidirectional character mappings within a limited ASCII range (186-255). They essentially swap characters within this range, creating a simple substitution cipher. No explicit return value.

* **`dos2arial()`**: This function is very similar to `Montimes_to_ArialMon()` and `ArialMon_to_Montimes()`, performing a large scale character substitution from a range (128-251) to another (192-255) with some exceptions. No explicit return value.

Control Flow

The control flow in each subroutine is largely the same:

1. **Initialization:** The maximum character count (`Max`) is obtained, and the selection is moved to the beginning of the document. An index `i` is initialized to 1.
2. **Loop:** A `While` loop iterates until the index `i` exceeds `Max`.
3. **Character Retrieval:** The current character (`Char`) is obtained using `Selection.Text`.
4. **Character Code Extraction:** The Unicode or ASCII value (`uni_code` or `asc_code`) of `Char` is extracted using `AscW()` or `Asc()`.
5. **Conditional Replacement:** A `Select Case` statement checks `uni_code` or `asc_code` against various predefined values. If a match is found, the character is replaced using `Selection.TypeText` with a specified replacement character.
6. **Iteration:** The index `i` is incremented, and the selection is moved to the next character using `Selection.MoveRight`.
7. **Loop Continuation:** Steps 2-6 are repeated.

Data Structures

The primary data structures used are implicit:

* **Word Document:** The VBA code interacts directly with the currently open Microsoft Word document's text. The document's characters are treated as a sequence.
* **Variables:** Simple variables like `Max`, `i`, `r`, `Char`, `uni_code`, `ascii_code` are used to hold temporary values and control the loop.

Malware Family Suggestion

Given the functionality of the code, it's most likely a component of a **polymorphic virus** or a more general **macro virus**. The code's primary purpose is character substitution, which serves as a simple but effective obfuscation technique. This obfuscation makes reverse-engineering and detection more difficult. The multiple similar functions further enhance obfuscation. The mappings themselves could encode commands or data, making analysis even harder without decoding them. The presence in a `.exe` file suggests it's likely part of a larger malware payload. The repeated similar functions are consistent with attempts to evade signature-based antivirus.