Analysis Report for: ab.txt

Overall Functionality

This C code exhibits strong characteristics of malware. It's heavily obfuscated, using a custom encoding scheme (`USVVUWMKBKLL`) to hide strings and function calls, and engages in several actions indicative of malicious behavior: file manipulation, process creation, and registry interaction. The core functionality appears to involve injecting and executing malicious code (`RUNPE` function) likely through a .NET assembly (`RegAsm.exe`). The code decrypts shellcode and then uses it to execute a payload. Various other functions seem to be utility functions supporting this core objective, such as string manipulation, file operations, and potentially, interacting with the system's registry to achieve persistence. The heavily obfuscated nature makes precise determination of the exact malicious payload difficult without significant deobfuscation efforts.

Function Summaries

* **`KWHDSTJFBLGEW($STEXT, $SYMBOL)`:** This function replaces all occurrences of `$SYMBOL` within `$STEXT` with an empty string. It uses a custom string replacement method. Returns the modified string.

* **`XEMNESXDAW($SSTRING)`:** Converts a string to its hexadecimal representation after converting it to binary. Returns a hexadecimal string.

* **`NWLJKWAUMZPQZ($SFILEPATH)`:** Checks if a file exists and if it's not a directory. Returns 1 if it's a regular file; otherwise, 0.

* **`KRAVMKWCVIZX($SSTR, $SCHR, $ICASE = 0x0)`:** Replaces occurrences of `$SCHR` in `$SSTR`. `$ICASE` controls case sensitivity (0x0 for case-sensitive, 0x1 for case-insensitive). Returns an AutoIT's @Extended which is a result of StringReplace (not a clearly defined return value in C).

* **`YCPMZJHEJLRVOME($HICON1, $HICON2)`:** Performs a DLL call, likely to a Windows API function, using parameters passed and processed through `KWHDSTJFBLGEW`. The purpose is unclear without knowing the target DLL and function, but it likely involves interacting with icons or handles to windows objects. Returns the first element of the result array from DLLCall.

* **`APAOLELNIMLE($IVALUE, $VTRUE, $VFALSE)`:** Returns either `$VTRUE` or `$VFALSE` based on whether `$IVALUE` is greater than 0.

* **`AYQRUMTCWLW($SSTRING)`:** Extracts IP addresses from a string using regular expressions. It filters out invalid IPs. Returns an array containing valid IPs and their count.

* **`QNZSNJONDH($SSTRING)`:** Uses regular expressions to validate if the input string is a number (integer or floating-point). Returns 1 if valid, 0 otherwise.

* **`WUOIWTSTFMC($ILENGTH)`:** Converts a length (presumably in miles) to kilometers. Returns the kilometer equivalent.

* **`ZAIQZQSLIF($NJOKER = 0x0)`:** Generates a random playing card (or Joker). `$NJOKER` determines whether the Joker is included. Returns a string representing the card.

* **`HADSULCICSNH($BIN)`:** Converts a binary string to its decimal equivalent. Returns the decimal value.

* **`ITQBADDMSPJTS($N_0, $N_1, $N)`:** Calculates the Nth Fibonacci number. Returns the Fibonacci number.

* **`LDTATTYQYVSUNA($ICOLOR)`:** Performs bitwise operations on a color value. The exact purpose is unclear without more context. Returns a modified color value.

* **`YEJTHFAATYLDIZ($NUMBER, $BASE)`:** Calculates the logarithm of `$NUMBER` with `$BASE`. Returns the logarithm.

* **`VPVWRUOCGQNKL($IMGFULLPATH)`:** Gets the dimensions of an image. It creates a temporary GUI window to obtain the size. Returns an array containing the width and height or an error code.

* **`DMCRDLFTOMF($ILENGTH)`:** Generates a random alphanumeric string of length `$ILENGTH`. Returns the random string.

* **`CMHPRPIALVP($SSTRING, $INUMCHARS)`:** Extracts a substring of length `$INUMCHARS` from `$SSTRING`. Returns an array containing the substring and its length.

* **`VVAYSAJMHT($SDATA)`:** Processes a string, converting lowercase letters to their corresponding ASCII values. Returns a sum of ASCII values.

* **`VLCHHLPBQI($ICOLOUR, $ITOLERANCE = 0x1e)`:** Performs conditional bitwise operations on a color value, based on tolerance. Returns a modified color value.

* **`NJSJGXPLBWK($ICOLOR)`:** Performs bit shifting and bitwise AND operations on a color value. Returns a modified color value.

* **`SJTULAROJQURN()`:** Extracts the filename (without extension) from the script's name. Returns the filename.

* **`JNGXMCYXDPUYXXM($ISTARTYEAR, $SDELIMETER = "-")`:** Creates a string showing the `$ISTARTYEAR` and the current year. Returns a formatted string.

* **`XYIQMRIFHV($SSTRING, $IREPEATCOUNT)`:** Repeats a string multiple times. Returns a repeated string.

* **`GYDXNMVNHJMJCER($SDEFAULT)`:** Checks if a string matches specific keywords using regular expressions. Returns a result indicating the match.

* **`GUISJOSHVZ($SSTRING)`:** Randomly shuffles the characters in a string. Returns the shuffled string.

* **`USVVUWMKBKLL($OZVKZIGWRBQK, $BMWQDBSBKQBX)`:** This is a custom encoding/decoding function. It takes two hexadecimal strings as input, converts them to binary, and then performs XOR operations based on the length of the second string to decode the first. The function is crucial for decoding obfuscated strings used throughout the script. Returns a decoded string.

* **`KKMQSMZPWA($NAME, $FILENAME)`:** This function appears to write data to a file. The file path is constructed using `$STARTUPDIR`, several obfuscated strings processed by `USVVUWMKBKLL`, `$FILENAME`, and more obfuscated strings. It uses other functions to prepare the data and check file existence. The core function likely writes a payload to disk. The conditionals and multiple obfuscated calls suggest it might fail silently under certain conditions.

* **`CNAAAXQUFE($RESNAME, $RESTYPE)`:** This is another function that heavily relies on the custom encoding and decoding. It seems to load and extract data from a resource embedded within an executable or DLL. It uses the obfuscated `$YTZDXJRVTBQU` function for various tasks. This likely loads the malicious .NET assembly which will be run by RUNPE.

* **`SCWHRUBWBM($PID)`:** This function seems to be a loop that continuously checks for a process with a specific ID (`$PID`). The purpose is unclear without knowing the context of `LIASFADCUS()`.

* **`DKZPOKMAMM($FILE, $STARTUP, $RES)`:** Writes data to a file (`$FILE`). The data comes from `CNAAAXQUFE`, implying it writes a file possibly using a resource from a DLL or EXE.

* **`VLDDAJEUMO($TITLE, $BODY, $TYPE)`:** This function's logic depends on a seemingly arbitrary boolean variable `$BOOL`. If true, it calls an obfuscated function (`$OZUBKJYQMQBZ`) with the provided parameters. This function might be related to message boxes or logging.

* **`JMTMLGHHJS($URL, $PATH)`:** This function potentially downloads data from a URL and writes it to a file. The conditionals are again obfuscated.

* **`ICILGBOWJA()`:** This function calls an obfuscated function conditionally, based on a condition involving an obfuscated call.

* **`FVHIUZGIQJ()`:** Iterates through an array and makes obfuscated checks before potentially calling a function (`$LDIBTBPKQSVJ`).

* **`HEDIQPTOPT()`:** An empty function.

* **`ACL($HANDLE)`:** This function appears to perform actions that involve access control lists (ACLs), possibly manipulating file permissions.

* **`QIKMWSCSXZ()`:** This is a function responsible for conditional calls to other functions.The conditions and calls are obfuscated to check and run functions related to the malware payload.

* **`UILMIOFLRL()`:** Performs several actions, including calling obfuscated functions potentially related to file operations.

* **`ZYRALEZAVQ()`:** Similar to `UILMIOFLRL()`, this function likely executes multiple actions, possibly involving file system operations.

* **`RUNPE($WPATH, $LPFILE, $PROTECT, $PERSIST)`:** This is the core malicious function. It takes the path of `RegAsm.exe`, the decrypted shellcode, and boolean flags for protection and persistence. It appears to inject and run malicious shellcode. The shellcode is encoded in `$BIN_SHELLCODE`.

* **`LIASFADCUS()`:** This function's purpose is unknown but is called by `SCWHRUBWBM` and at the end of the main script.

* **Functions `LERTKVRKCKRW()`, `JIKIPMVKZSVI()`, `ZFOBKWVMQVKC()`, `OBWFPPJEQDIK()`, `YUJFIPJVOEMJ()`, `HTEOMVQMKBHV()`, `MRJNQITNDNYP()`, `JKONPWOQPXBV()`, `EBGHSFXKWCDF()`, `DWPPJCACVQIN()`, `VLSYHBDQTVDV()`, `JPODPZKMPNUX()`, `PEGZGEELWAPS()`, `ERPFQUPFUWNK()`, `RZJMWTDMUHUO()`, `JVSFCAMBLCOQ()`, `OZYDFPAAXJCB()`, `AWWLAQJRZIQC()`, `RWPWOJZKJSCI()`, `XGJWLPFTWAUE()`, `OLBQGJPJEDZI()`, `HLHAOIFKCVIZ()`, `HYXXAMTSFNIU()`, `WZFJUIANUATQ()`:** These are all simple functions that seem to return the result of a `USVVUWMKBKLL` call with different hardcoded hexadecimal strings.These functions are likely used to decode strings needed for different malicious activities.

Control Flow

The control flow is complex and heavily obfuscated, especially within functions that use the custom string encoding and multiple nested function calls. The main script executes functions sequentially, with the `RUNPE` function being critical. The conditional checks throughout (especially the numerous `IF 0x...` statements which are likely just to confuse analysis) make it difficult to determine precisely which execution path will be taken. These conditionals are often nonsensical mathematical checks probably designed as obfuscation and to confuse analysis. Looping is present in functions like `SCWHRUBWBM`, `HAGPKWKUMC`, and `GUISJOSHVZ`, but the exact iterations and conditions are obfuscated.

Data Structures

The primary data structure is the array. Arrays are used extensively to store data, such as the results of string splitting, lists of strings, and function return values. There's no use of more complex data structures like linked lists or trees.

Malware Family Suggestion

Given the code's characteristics – obfuscation through custom encoding, file writing, process injection via shellcode, and potential registry manipulation – it strongly suggests a **downloader/dropper** type of malware, possibly belonging to a family known for its use of .NET assemblies for payload delivery. The heavy use of AutoIT-like syntax suggests this could be from the AutoIT malware scene (however, this is not a true AutoIT script due to use of DLL calls). Further deobfuscation is required to determine the specific payload and family. The complex structure suggests the authors had a significant level of experience in obfuscation techniques.