Analysis Report for: a.txt

Overall Functionality

This C code is a sophisticated obfuscated malware sample. Its primary goal is to execute malicious payload. The code heavily relies on obfuscation techniques to hide its true intentions and make analysis difficult. The obfuscation involves heavily encoded strings, custom functions for encoding and decoding, and the use of many seemingly unrelated functions. The core functionality is encapsulated in the `RUNPE` function which appears to execute a Portable Executable file, likely the final payload. The code also exhibits characteristics of persistence, attempting to establish itself by creating directories and possibly registry entries. Other functions seem designed to interact with the system, possibly for reconnaissance or data exfiltration.

Function Summaries

Function Name	Purpose	Parameters	Return Value
`QWCOXOBPSC`	Performs an obfuscated check, potentially related to system checks or anti-analysis.	None	None (side effects)
`_Timer_Diff`	Calculates the difference between two timestamps.	`$itimestamp` (performance counter timestamp)	Time difference in milliseconds
`_Timer_GetIdleTime`	Gets the system's idle time.	None	Idle time in milliseconds or error code
`_Timer_GetTimerID`	Retrieves a timer ID from an internal array.	`$wparam` (timer ID)	Timer ID or 0x0
`_Timer_Init`	Initializes a performance counter timestamp.	None	Performance counter timestamp
`_Timer_KillAllTimers`	Kills all timers associated with a window handle.	`$hwnd` (window handle)	`True` on success, `False` on failure
`_Timer_KillTimer`	Kills a specific timer associated with a window handle.	`$hwnd` (window handle), `$itimerid` (timer ID)	`True` on success, `False` on failure
`__TIMER_QUERYPERFORMANCECOUNTER`	Gets the current performance counter value.	None	Performance counter value as extended
`__TIMER_QUERYPERFORMANCEFREQUENCY`	Gets the performance counter frequency.	None	Performance counter frequency as extended
`_Timer_SetTimer`	Sets a Windows timer.	`$hwnd` (window handle), `$ielapse` (interval), `$stimerfunc` (callback function), `$itimerid` (optional ID)	Timer ID or error code
`KKDVBGVTQI`	Performs an obfuscated action, potentially related to message boxes or logging.	`$title`, `$body`, `$type`	None (side effects)
`READRESOURCES`	Reads data from a resource.	`$resname` (resource name), `$restype` (resource type)	Read data
`EFBOSMAKIJ`	Executes obfuscated code, possibly for additional actions.	None	None (side effects)
`EKMHCBDXKA`	Obfuscated loop potentially related to persistence.	`$pid` (process ID)	None (side effects)
`ACL`	Performs actions related to Access Control Lists (ACLs).	`$handle` (handle)	None (side effects)
`VEPAMNJXCR`	Executes obfuscated code, possibly for additional actions.	None	None (side effects)
`NFMAREIAQT`	Performs an obfuscated cryptographic or encoding operation.	`$vdata`, `$vcryptkey`, `$rt`	Obfuscated key
`KDCDDSOPEY`	Executes a loop a specific number of times, possibly for delay or obfuscation.	`$loop`, `$time`	None (side effects)
`FSPDSTIVNY`	Performs an obfuscated operation, potentially related to file handling.	`$soccurrencename` (string)	None (side effects)
`QKQQMVQIDV`	Executes obfuscated code in a loop, possibly for additional actions.	None	None (side effects)
`YKGKWUVAJN`	Performs an obfuscated operation, potentially related to file handling and execution.	`$file`, `$startup`, `$res`	None (side effects)
`EDMELFDYNO`	An empty function, likely a placeholder or remnant of development.	None	None
`WETSZLRSRC`	Performs obfuscated operation, potentially related to file handling.	`$name`, `$filename`	None (side effects)
`GLOBALDATA`	Performs an obfuscated operation that handles data.	`$data`, `$rt`	Obfuscated data
`HOJTTVMGDVIY`	A custom function to decode strings.	`$nqtfqsvfcawd`, `$aownlhrcygbr`	Decoded string
`VKDNHDMYHSQJNHC`	A custom function that decodes a string representation of an index into a character from a predefined alphabet.	`$str` (comma-separated string of numbers)	A character from the alphabet
`RUNPE`	This function appears to be the core malicious function. It processes shellcode and potentially runs a PE file.	`$wpath`, `$lpfile`, `$protect`, `$persist`	None (side effects)
`_WinAPI_Beep`	Wraps the Windows API `Beep` function.	`$ifreq` (frequency), `$iduration` (duration)	`True` on success, `False` on failure
`_WinAPI_FormatMessage`	Wraps the Windows API `FormatMessageW` function.	`$iflags`, `$psource`, `$imessageid`, `$ilanguageid`, `$pbuffer`, `$isize`, `$varguments`	Formatted message length
`_WinAPI_GetErrorMessage`	Wraps the Windows API `FormatMessageW` to retrieve error messages.	`$icode`, `$ilanguage`	Error message string
`_WinAPI_GetLastError`	Gets the last Windows error code.	None	Last error code
`_WinAPI_GetLastErrorMessage`	Gets the last Windows error message.	None	Last error message string
`_WinAPI_MessageBeep`	Wraps the Windows API `MessageBeep` function.	`$itype` (type)	`True` on success, `False` on failure
`_WinAPI_MsgBox`	Wraps the Windows API `MsgBox` function with input blocking.	`$iflags`, `$stitle`, `$stext`	None
`_WinAPI_SetLastError`	Sets the last Windows error code.	`$ierrorcode`	NULL
`_WinAPI_ShowError`	Displays an error message box and optionally exits the script.	`$stext`, `$bexit`	None
`_WinAPI_ShowLastError`	Displays an error message box with the last error information.	`$stext`, `$babort`, `$ilanguage`	0x1 on success or error code
`_WinAPI_ShowMsg`	Displays an informational message box.	`$stext`	None
`__COMERRORFORMATING`	Formats a COM error object into a user-friendly string.	`$ocomerror`, `$sprefix`	Formatted COM error string

Control Flow (Significant Functions)

* **`RUNPE`**: This function is crucial. It first defines a large chunk of encoded shellcode (`$bin_shellcode`). It then uses previously defined obfuscated functions (`HOJTTVMGDVIY`, `VKDNHDMYHSQJNHC`, `$REKTJQJGQK`, `$DYKACAYBNA`, `$sxgudebaxdni`) to decode and manipulate data. The decoded shellcode is likely the main payload. The function includes conditional logic based on `$protect` and `$persist`

parameters determining whether to perform further actions related to access control lists (ACLs) and persistence (via the `EKMHCBDXKA` function).

* **`HOJTTVMGDVIY`**: This function takes two comma-separated strings as input. It uses `$SXGUDEBAXDNI` (likely a string-to-binary conversion function) to convert the input strings into arrays of characters. It iterates through the first array, performing XOR operations using the length of the second array and additional calculations to create a final decoded string.

* **`NFMAREIAQT`**: This function performs a complex operation. It uses global data (`$__g_acryptinternaldata`), which is initialized and manipulated within the function. It heavily uses the obfuscated decoding functions, implying a cryptographic or encoding operation is being performed on the input data (`$vdata`). The result is stored in `$vcryptkey`.

* **`GLOBALDATA`**: This function takes obfuscated data (`$data`) and a runtime type (`$rt`) as input. It uses control flow to obfuscate the logic. The function calls `QWCOXOBPSC` after processing the input data, suggesting an intricate process that depends on system conditions.

Data Structures

* **`$__g_atimers_atimerids`**: A global array used to manage timer IDs, likely a custom timer mechanism for the malware's internal operations.

* **`$__g_acryptinternaldata`**: A global array used to hold intermediary data in the `NFMAREIAQT` function, appearing to manage internal state for encryption or encoding.

* Many other global arrays and constants are used throughout the code but they mostly seem to be to store obfuscated data.

Malware Family Suggestion

Given the code's heavy reliance on obfuscation, its use of shellcode injection (`RUNPE`), and attempts at persistence (`EKMHCBDXKA`), this sample strongly suggests a **downloader or dropper** that's designed to deliver and install a more harmful payload. The complex encoding and cryptographic routines indicate an attempt to avoid static analysis and signature-based detection. The exact family is impossible to determine without executing and further analyzing the final payload.

Note: Analyzing obfuscated malware can be extremely dangerous. This analysis is based solely on static analysis of the provided code. Executing this code is highly discouraged due to the substantial risk of infecting your system. Further dynamic analysis would be required to determine the complete functionality and the specific malware family involved. Proper tools and a sandboxed environment are crucial when handling such code.