Analysis Report for: a.au3

Overall Functionality

This C code appears to be a script for creating a multi-boot USB drive, specifically designed for UEFI systems and capable of booting both Windows VHD and WIM images. The code extensively uses AutoIt's ability to call Windows APIs, managing low-level system operations like disk partitioning, file system manipulation, and interacting with the Windows Boot Configuration Data (BCD) store to add boot entries for the specified images. It also includes substantial functionality for handling various error conditions and providing user feedback through GUI elements.

Function Summaries

The code contains numerous functions, many of which are wrappers around Windows APIs. Here's a summary of some key functions:

* **`_WinAPI_*` functions:** These functions are wrappers around various Windows APIs, providing error handling and potentially some data type conversions. Examples include `_WinAPI_CreateFile`, `_WinAPI_GetLastError`, `_WinAPI_SendMessage`, etc.

* **`_Security_*` functions:** These functions handle privilege adjustments and token manipulation, enabling the script to perform actions requiring elevated privileges.

* **`_Mem*` functions:** Functions for memory allocation and manipulation, particularly useful for interacting with the Windows API calls which often handle data through pointers.

* **`_GUICtrlStatusBar_*` functions:** A set of functions specifically built for managing a status bar GUI element within the main AutoIt GUI.

* **`_Array*` functions:** A collection of functions for array manipulation; these functions go beyond the basic array functions in AutoIt, offering specialized operations for sorting, searching, and data manipulation.

* **`_Date*` functions:** Date and time manipulation routines. These functions use Windows APIs for higher-precision timekeeping and conversions.

* **`_ProcessGetName` and `_ProcessGetPriority`:** These retrieve process name and priority level, showing interaction with running processes on the system.

* **`_MOUNT_EFI`:** Mounts the EFI system partition (ESP), a crucial step for creating bootable UEFI USB drives.

* **`_IMG_FSEL`:** Allows the user to select the boot image (VHD or WIM) file.

* **`_TARGET_DRIVE` and `_WinDrv_drive`:** Functions to let the user select the target boot and system drives respectively via GUI.

* **`_GO`:** This is the main function triggered by the "GO" button in the GUI. It orchestrates all the steps involved in creating the multi-boot USB.

* **`_WIM_MENU`:** Handles the addition of a WIM boot entry to the BCD store.

* **`_VHD_MENU`:** Handles the addition of a VHD boot entry to the BCD store. This is more complex due to requiring interaction with diskpart.

* **`_BCD_*` functions:** Helper functions for managing the BCD store (adding entries, setting parameters, etc).

* **`_LISTUSB_UEFI`:** A function responsible for identifying USB devices and gathering their properties.

* **`__ARRAYDISPLAY_SHARE`:** A function designed to create and display a detailed GUI representation of array data, useful for debugging and inspecting the contents of arrays within the script.

* **`_WinAPI_WideCharToMultiByte` and `_WinAPI_MultiByteToWideChar` functions:** These functions provide conversion between wide character strings and multi-byte strings; this is critical for proper string handling across different character encodings.

* **`__UDF_GETNEXTGLOBALID` and `__UDF_FREEGLOBALID`:** These provide a custom ID allocation and release mechanism for internal GUI controls and resources.

Control Flow

The main control flow is driven by the GUI events.

* **Main GUI Loop:** The script starts by creating a GUI and then enters a `While 1` loop that continuously checks for GUI events.

* **`_Go` Function:** This function is executed when the user clicks the "GO" button. It handles the main logic: drive checking, creating BCD entries, and then showing results. The process includes calls to other functions.

* **`_IMG_FSEL` Function:** This function is triggered when the user selects a boot image file. It determines the image type, checks the path (validating length and location) and updates GUI labels with the file path and size.

* **`_TARGET_DRIVE` Function:** This function is executed when the user selects the boot drive. It verifies the drive's readiness, file system (must be FAT32 for a UEFI system), size, and partition style.

* **`_WinDrv_drive` Function:** Similar to `_TARGET_DRIVE`, but for selecting the system drive.

* **`__ARRAYDISPLAY_SHARE`:** This creates a sophisticated GUI to display array contents. It features loops to iterate over rows and columns of the array, adding entries to a list view. It also handles column alignment, and user interaction features.

Data Structures

* **Arrays:** The script heavily uses arrays to store and manipulate various data, including lists of drives, partition information, and other system details. Notably, the `$__g_awinlist_winapi` array keeps track of found windows.

* **DllStructs:** `DllStructCreate` is used extensively to create structures for passing data to and from Windows API functions. These structures are crucial for interfacing with the lower-level system calls. Examples include structures for representing RECTs, POINTS, SYSTEMTIME, FILETIME, etc. These data structures are used to interact with various Windows API function calls.

* **Dictionaries:** The `_ArrayUnique` function utilizes a dictionary to efficiently find unique elements in arrays.

Malware Family Suggestion

Given the code's capability to interact with the Windows BCD store, system drives, and execute commands through the `Run` and `RunWait` functions, it has characteristics consistent with a **bootloader infector** or **rootkit**. The ability to add new boot entries to the BCD gives it the ability to make itself the primary boot loader, replacing the legitimate system one. This behavior combined with file system manipulations is a strong indicator of malicious intent. The extensive use of error handling and attempts to clean up after itself don't entirely rule out malware, as sophisticated malware often tries to hide its actions. The code's functionality to copy, modify, and run files on the target system warrants careful review.

Important Note: This analysis is based purely on the code's functionality and structure. Without knowing the context of where this code would be used, and the actual functionality of some custom functions (like the user-defined `__ArrayDisplay_SortCallBack`), it is impossible to say with certainty that it is or is not malicious. The actions performed by this code could easily be abused for malicious purposes. Therefore it should be handled with extreme caution.