Analysis Report for: a.4c4fca5bb9ed4e4dd01b144dfa5909466bf4c0a58304a8c8db4aba4d64fbb496.au3

Overall Functionality

This C code exhibits strong indicators of being malicious. It performs a series of obfuscated actions, including file installations, DLL manipulations, and numerous function calls with hexadecimal arguments that strongly suggest malicious intent. The code uses AutoIt, a scripting language often used for creating malware, and heavily relies on string concatenation and hexadecimal values to hide its true functionality. The code attempts to install files ("uppishly" and "underbalanced"), manipulates DLL structures, and makes numerous system calls using a function seemingly named `$HBCNBCPFY`. The functions are largely numeric, indicating an attempt to evade static analysis. This strongly suggests an attempt to execute some sort of payload or perform harmful actions on the system. The sheer volume of nested loops and conditional statements further complicates analysis, adding to the obfuscation.

Function Summaries

* **`FileInstall("uppishly", @TempDir & "\uppishly", 0x1)`**: Installs a file named "uppishly" to the temporary directory. `0x1` likely indicates an overwrite flag.

* **`Global $hbcnbcpfy = Execute("C" & "all")`**: Executes a command likely designed to start the payload. The "C" & "all" is a simple obfuscation technique that could be easily resolved as "Call".

* **`Global $wmniahdux = $HBCNBCPFY("File" & "Re" & "ad", $HBCNBCPFY("Fil" & "eO" & "pen", @TempDir & "\uppishly"))`**: This line reads the contents of the "uppishly" file using a custom function `$HBCNBCPFY` and seemingly re-reads or re-processes that file using "FileRead" function.

* **`$wmniahdux = XUPNFYBKC($wmniahdux)`**: Processes the data read from "uppishly" using the `XUPNFYBKC` function.

* **`Global $blpdlrhr = $HBCNBCPFY("DllS" & "truct" & "Create", XUPNFYBKC("103 126 121 106 96") & BinaryLen($wmniahdux) & XUPNFYBKC("98"))`**: Creates a DLL structure using `$HBCNBCPFY`. The arguments are heavily obfuscated using hexadecimal string manipulation through `XUPNFYBKC`. This line manipulates DLL structures, potentially loading a malicious DLL into memory.

* **`$HBCNBCPFY("Dl" & "lStr" & "uctSet" & "Data", $blpdlrhr, 0x1, $wmniahdux)`**: Sets data within the created DLL structure.

* **`Global $rzufbsghe = $HBCNBCPFY("Dll" & "Struct" & "Ge" & "t" & "Ptr", $blpdlrhr)`**: Gets a pointer to the data within the DLL structure.

* **`FileInstall("underbalanced", @TempDir & "\underbalanced", 0x1)`**: Similar to the first file installation, installs another file to the temporary directory.

* **`$HBCNBCPFY("D" & "l" & "lC" & "all", ...)` (multiple calls)**: Makes multiple calls to a function that likely executes code from the loaded DLL. The parameters are heavily obfuscated using hexadecimal strings and likely contain function addresses and arguments.

* **`QZIRZFJQQ()`, `ITJRCVER()`, `BICSJJAGMC()`, `JJWHBZIO()`**: These functions contain numerous AutoIt API calls, many involving file system manipulation, registry access, and process management. They appear to perform various actions, likely distractions or additional steps in the malware's execution.

* **`XUPNFYBKC($mytccqoy)`**: This function takes a string as input, splits it using spaces as delimiters, converts each part to its ASCII decimal equivalent, and concatenates the results. This is a common obfuscation technique.

Control Flow

The control flow is highly complex due to nested loops and conditional statements. The primary execution flow seems to be:

1. Install two files ("uppishly" and "underbalanced").
2. Execute a command ("Call").
3. Read and process the "uppishly" file.
4. Create and manipulate a DLL structure.
5. Execute code from the manipulated DLL via multiple `$HBCNBCPFY("DllCall", ...)` calls.
6. Execute several functions (`QZIRZFJQQ`, `ITJRCVER`, `BICSJJAGMC`, `JJWHBZIO`) that perform various operations, likely acting as decoys or components of the attack.

The nested loops create a large number of execution paths, making reverse engineering difficult. Many of the loops and conditionals seem designed to obfuscate the true actions by creating complex, seemingly arbitrary execution flow.

Data Structures

The code primarily uses simple variables. The most significant data structure is the DLL structure created and manipulated within the code. The exact structure of this DLL is not apparent from the obfuscated code.

Malware Family Suggestion

Given the characteristics of the code—file installation, DLL manipulation, obfuscation techniques (hexadecimal encoding, string splitting), extensive use of AutoIt API calls, and the overall structure designed to hinder analysis—it is highly likely that this code belongs to a type of **downloader/dropper** malware. The code downloads and installs files, then uses those files to load a payload into memory. The extensive number of function calls and the usage of obfuscation techniques indicate a high level of sophistication employed to evade antivirus software and code analysis. The actions within the various functions suggest that further actions beyond simply dropping a payload are likely performed. A more thorough reverse engineering would be needed to definitively identify the exact nature of the payload and the full extent of the malware's capabilities. However, it's crucial to note that based on this analysis, attempting to execute this code is extremely dangerous and should absolutely be avoided.