Analysis Report for: a.txt

Overall Functionality

This C code, written using AutoIt's AutoItScript syntax (despite the `DllCall` suggesting a DLL interaction which is unusual in pure AutoIt), exhibits strong characteristics of malicious behavior. It appears to be a malware sample designed to perform various actions on the compromised system, including file manipulation, system calls, and potentially network communication (implied by `InetRead` and `InetClose`). The core functionality is obfuscated through extensive use of hexadecimal numbers, meaningless variable names, and a custom XOR-based encryption/decryption function (`BNHAIIZGAO`). The code's actions are spread across nested loops and conditional statements, making analysis difficult. The ultimate goal is unclear without further reverse engineering and understanding the encrypted strings and DLL calls.

Function Summaries

* **`BNHAIIZGAO($pkclgfxt)`:** This function acts as a simple XOR cipher. It takes a string (`$pkclgfxt`) as input and returns an XOR-encoded version of that string. The encryption key is dynamically generated using a modulo operation.

Control Flow

The code's main logic is a series of nested loops and conditional statements. Let's analyze the `BNHAIIZGAO` function and the main execution flow:

* **`BNHAIIZGAO($pkclgfxt)`:**
1. Initializes an empty string `$wgjoeutigv`.
2. Sets an initial XOR key `$hejnpatxl` to 0x37 (55 decimal).
3. Iterates through each character of the input string `$pkclgfxt`.
4. Gets the ASCII value of each character.
5. Performs a bitwise XOR operation between the ASCII value and the current key.
6. Appends the character representation of the XOR result to `$wgjoeutigv`.
7. Updates the XOR key using a modulo operation: `$hejnpatxl = Mod($hejnpatxl + 0xd, 0x100)`. This ensures the key changes for each character.
8. Returns the encrypted string `$wgjoeutigv`.

* **Main Execution Flow:**
1. Installs two files ("brawlys" and "pyogenesis") to the temporary directory. This strongly suggests the dropping of malicious payloads.
2. Reads the content of "brawlys," decrypts it using `BNHAIIZGAO`, and stores it in `$zcjjirh`.
3. Creates a DLL structure (`$xnftjrgu`) whose size and content are derived from more XOR-encrypted strings and the length of the decrypted "brawlys" content. The structure's data is populated with `$zcjjirh`.
4. Obtains a pointer (`$lwvjjryj`) to the created DLL structure.
5. Makes several `DllCall`s, passing in a mix of XOR-encrypted strings and the pointer `$lwvjjryj`. These calls are highly suspicious, most likely interacting with a loaded DLL to execute malicious commands.
6. The rest of the code consists of a massive sequence of nested loops and conditional statements involving numerous AutoIt functions related to GUI manipulation (`GUICtrl...`), file system operations (`File...`), process management (`Process...`), network operations (`Inet...`), registry manipulation (`Reg...`), and timer functions (`Timer...`). This section heavily obfuscates the actual actions performed. Many of these operations are repeated with slightly different parameters, possibly for anti-analysis techniques.

Data Structures

The primary data structure is the DLL structure created using `DllStructCreate`. Its purpose is to hold the decrypted content of the "brawlys" file, likely providing data to the malicious DLL functions called later. The structure's size is dynamically calculated, adding to the obfuscation.

Malware Family Suggestion

Given the file installation, XOR encryption, DLL calls (possibly for code injection or other advanced techniques), extensive file system and registry manipulation, and the obfuscation techniques used, this code is highly suggestive of a **backdoor or advanced persistent threat (APT)**. The specific family cannot be definitively determined without further analysis of the decrypted strings and the behavior of the DLLs invoked. The heavy obfuscation is a common characteristic of sophisticated malware designed to evade detection. The nature of the network operations (if any) would provide further clues about the malware's intended actions (e.g., data exfiltration, command-and-control communication).