Analysis Report for: script.bin

Decoded using latin-1...

The provided code is heavily obfuscated, likely through the use of a packer or similar tool. The majority of the code consists of seemingly random characters and control characters (like ÿ and numerous null characters) interspersed with snippets of what appear to be function calls and strings. A meaningful analysis is impossible without deobfuscation. However, we can make some observations based on the visible fragments.

Overall Functionality

The deobfuscated code likely implements a program uninstaller. This conclusion is drawn from the presence of strings related to uninstalling, deleting files and folders, registry manipulation, and handling user interaction (abort, retry, ignore prompts). The language used in the strings suggests multiple language support. The presence of OLE32 function calls further suggests it may interact with the Windows registry and possibly other system components. The overall structure appears to be a series of calls that perform uninstallation steps.

Function Summaries

Due to the obfuscation, it's impossible to provide accurate function summaries. We can only speculate about the function calls based on the visible parts of the strings:

* **`Kernel32::GetCurrentProcess()`**: Likely retrieves a handle to the current running process. (Standard Windows API call).
* **`Kernel32::GetProcessId()`**: Likely retrieves the process ID of the specified process. (Standard Windows API call).
* **`Kernel32::CreateToolhelp32Snapshot()`**: Creates a snapshot of the specified processes. (Standard Windows API call).
* **`Kernel32::Process32FirstW()`**: Retrieves information about the first process in a snapshot. (Standard Windows API call).
* **`Kernel32::Process32NextW()`**: Retrieves information about the next process in a snapshot. (Standard Windows API call).
* **`Kernel32::CloseToolhelp32Snapshot()`**: Closes the snapshot handle. (Standard Windows API call).
* **`Kernel32::OpenProcess()`**: Opens a handle to a specified process. (Standard Windows API call).
* **`psapi::GetModuleFileNameExW()`**: Retrieves the full path of a module for a given process. (Standard Windows API call).
* **`Kernel32::GetVersionEx()`**: Retrieves version information about the Windows operating system. (Standard Windows API call).
* **`SHELL32::SHGetKnownFolderPath()`**: Retrieves a known folder path, like "Program Files". (Standard Windows API call).
* **`OLE32::CoTaskMemFree()`**: Frees memory allocated using `CoTaskMemAlloc`. (Standard Windows API call).
* **`SHELL32::SHChangeNotify()`**: Notifies the shell of changes to the file system, which could be used to update the system after the uninstallation. (Standard Windows API call).

Control Flow

Again, due to the extensive obfuscation, a detailed control flow analysis is not feasible. The visible code suggests a sequence of operations:

1. Get information about the current process.
2. Create a snapshot of running processes.
3. Iterate through the processes (likely searching for a specific process to uninstall).
4. Get module information (file paths).
5. Perform file and folder deletion operations.
6. Perform registry key modifications.
7. Notify the shell of changes.

Data Structures

No explicit data structures are visible in the obfuscated code. However, the use of Windows API calls suggests the use of standard Windows data structures (like `PROCESSENTRY32` and structures used for file and registry manipulation).

Malware Family Suggestion

While the obfuscation prevents definitive classification, the functionality strongly suggests a **backdoor trojan or a potentially unwanted program (PUP)**. The ability to manipulate processes, files, and the registry, coupled with a uninstaller structure, could be abused. A legitimate uninstaller would not typically require this level of system access or obfuscation. The extensive obfuscation is a major red flag, hiding malicious behavior like persistence mechanisms, data exfiltration, or other harmful actions. Further analysis after deobfuscation is crucial for proper malware family identification. It could also be part of a larger malware infection, acting as an uninstaller for other components.