Analysis Report for: 5937741A59A1F8AF57990CDC246CC200.exe.vbs

Overall Functionality

This C code, written using AutoIt scripting language syntax (indicated by `$` variable prefixes, `RegRead`, `FileWrite`, etc.), acts as an installer for a software suite. It performs several actions:

1. **Preparation:** It extracts necessary files, creates directories, and deletes existing WPI registry keys.
2. **Configuration File Creation:** It generates configuration files (`useroptions.Rider`, `WFix.Rider`, etc.) containing settings for the WPI (WayGame Installer, likely) application, including window positions, audio settings, and a list of software to install. The location of the installer icon is dynamically adjusted based on the presence of `win7Theme.Rider`.
3. **Registry Manipulation:** It writes various configuration values to the Windows Registry, registering file associations and WPI settings. It also specifically attempts to overwrite media player extensions.
4. **Audio File Management:** It creates an M3U playlist file for audio files, dynamically choosing files based on the `win7Theme.Rider` file's existence.
5. **HTML File Modification:** It modifies the `WPI.hta` and `Installer.hta` (likely HTML Applications) files, setting the application title and icon path based on the presence of `win7Theme.Rider`.
6. **Software Installation:** It uses a separate executable (`Fourze.Rider` or `Fourze64.Rider`) to install a series of applications. The selection and installation of these applications are driven by the contents of the `WFix.Rider` file. This installer appears to install various freeware and open-source applications (Adobe Flash, 7-Zip, VLC, etc.). The `Fourze` executables are called with parameters indicating the software to install.
7. **Execution of WPI:** Finally, it launches the main WPI application.

Function Summaries

The code doesn't define any functions in the traditional C sense. Instead, it utilizes AutoIt's built-in functions and commands. The core actions are implemented via sequential AutoIt commands and conditional statements. We can consider blocks of code as functional units for the purpose of this analysis:

* **File Extraction:** `RunWait` calls are used to extract files from archives, acting as a self-contained extraction function.
* **Configuration File Writing:** A series of `FileWrite` calls builds the `useroptions.Rider` and `WFix.Rider` files. This can be considered as a single functional unit.
* **Registry Writing:** Multiple `RegWrite` and `RegDelete` function calls manipulate the registry. Again, this is a single functional unit.
* **Audio Playlist Creation:** `_FileListToArray` and a `For` loop build the audio playlist. This is a clearly defined function-like block.
* **HTML File Modification:** `_FileWriteToLine` calls modify the WPI HTML Application files.
* **Software Installation:** `RunWait` calls invoke the `Fourze` executables.

Control Flow

The control flow is primarily linear, punctuated by conditional statements (`If`, `ElseIf`, `EndIf`) based on:

* Existence of files (e.g., `win7Theme.Rider`).
* Windows version (`$WINVersion`).
* System architecture (`@OSArch`).

Loops are used primarily for writing multiple lines to files (creating configuration files and the audio playlist). The conditional statements determine the specific software installed and configuration settings.

Data Structures

* **Arrays:** The `$ListFolder` array holds the list of audio files obtained using `_FileListToArray`.
* **Strings:** Numerous string variables store file paths, registry keys, and other configuration data. These are manipulated using string functions like `StringReplace`.
* **Configuration Files:** The `useroptions.Rider` and `WFix.Rider` files act as simple configuration files, storing key-value pairs separated by semicolons. `WFix.Rider` uses a more complex data structure where various settings are assigned to arrays using an index `pn`.

Malware Family Suggestion

The code exhibits several characteristics strongly suggestive of malware, specifically an **installer for a potentially unwanted program (PUP)** or a **software bundler**.

* **Stealthy Installation:** The use of `@SW_HIDE` hides the extraction processes from the user.
* **Registry Manipulation:** Extensive registry changes beyond what is necessary for a legitimate application's configuration.
* **Bundled Software:** The installation of numerous programs without explicit user consent. This is a classic hallmark of PUPs. The installer does not clearly communicate what applications will be installed.
* **Arbitrary File Creation:** Creation of files in the `@TempDir`, which is a common location for malware to operate.
* **Overwriting Media Player Extensions:** The attempt to register the ".Rider" extension with MediaPlayer might be designed to hijack file associations.

* **External Executable:** The reliance on an external executable (`Fourze.Rider`/`Fourze64.Rider`) whose behavior is not clearly defined within the script makes it difficult to evaluate its benignity.

The fact that many of the bundled programs are legitimate does not negate the potential maliciousness of this installer; a common tactic is to bundle legitimate software alongside unwanted programs. The extensive registry manipulation and the use of a seemingly separate installer executable (Fourze) are very suspicious.

Therefore, while some functionality *could* be legitimate, the overall design and behavior strongly indicate this code is part of a potentially harmful software distribution mechanism, likely a PUP installer or a software bundler. Further analysis of the `Fourze.Rider` executable is crucial for a definitive determination.