Analysis Report for: 0B918A234D1717D66F0C077CE1495235.js

Overall Functionality

This C code is obfuscated malware designed to download and execute additional code from remote servers. It uses several techniques to hide its malicious intent, including base64 encoding, code obfuscation through hexadecimal representation of strings and variable names, and the use of Python's `codecs` and `json` libraries (indicated by the imports, though the actual use is obfuscated). The code leverages the `urllib` library for HTTP requests, suggesting a network-based attack. The overall goal is to download and execute further malicious payloads, making it a downloader or dropper type of malware.

Function Summaries

* **`NoRedirection(urllib_error.HTTPError)`:** A custom exception handler that seems to simply return the HTTP response regardless of errors. It doesn't handle the error itself.

* **`add_addon_log(string, level=xbmc.LOGDEBUG)`:** Logs a message to the addon log, using different log levels based on a 'debug' setting.

* **`makeRequest(url, headers=None)`:** Makes an HTTP request to a given URL. It handles headers, checks for charset encoding in the response, and attempts to decode the response based on the charset. It uses `re.search` to potentially find specific patterns in the response.

* **`getSources()`:** This function attempts to read source information from either a local file ('sources_file') or a default URL ('origin'). It parses JSON data and processes the results, potentially adding items to a list of sources. It also conditionally adds 'Community Files' and 'Search Other Plugins' based on addon settings.

* **`index()`:** This appears to be a stub function that may have been intended to add more functionality. Currently, it checks for the existence of a 'favorites' file and if it exists it adds items to it. It then calls `getData(origin, FANART)`.

* **`addSource(url=None)`:** Adds a new source to a list (potentially persistent storage). It checks for settings ('new_file_source', 'new_url_source') to determine the source URL. It then retrieves metadata from this source using the `getSoup()` function and adds the source data (title, url, thumbnail, etc.) to a source file (`sources_file`). If the source is a YouTube link, it triggers a different processing path.

* **`rmSource(name)`:** Removes a source from the list based on its name.

* **`getSoup(url, data=None)`:** Retrieves data from a URL, handling potential HTTP errors. This function performs some basic checks including extracting a view mode, potentially checking for an encryption key (`k`), and using `gzip` or `AES` decryption as needed. It uses regular expressions for parsing the HTML content.

* **`processPyFunction(data)`:** Processes data containing Python functions, extracting and evaluating them.

* **`getData(url, fanart, data=None)`:** Retrieves and parses data from a URL. It seems designed to handle XML data (`ElementTree`) from sources, potentially extracting information like channels, items and their associated metadata such as URLs and images.

* **`getAddonItems(name, url, fanart)`:** Retrieves and processes items data from a given channel URL.

* **`getSubChannelItems(name, url, fanart)`:** Retrieves and processes sub-channel items data.

* **`getItems(items, fanart, dontLink=False)`:** Processes a list of items, extracting metadata and potentially adding them to a playlist. It has conditional logic based on addon settings (`add_playlist`, `ask_playlist_items`, `parentalblock`). Crucially, this function also constructs and uses YouTube and Dailymotion URLs within its processing.

* **`getGoogleRecapchaResponse(captchakey, cj, type=1)`:** Solves a reCAPTCHA by using a URL request to Google's reCAPTCHA API, possibly for authentication.

* **`getURL(url, cookieJar=None, post=None, timeout=20, headers=None, noredir=False)`:** Fetches a URL using `urllib`, handling cookies and headers. It also handles charset encoding and decoding, similar to `makeRequest`.

* **`get_decode(str, reg=None)`:** Decodes a string using regular expressions, this function seems to be a custom decoder potentially for URL parameters.

* **`javascryptUnescape(str)`:** Unescapes a JavaScript encoded string.

* **`askCaptcha(m, html_page, cookieJar)`:** Handles CAPTCHA challenges during the process. It constructs URL to obtain the required CAPTCHA answer.

* **`askCaptchaNew(imageregex, html_page, cookieJar, m)`:** Alternative CAPTCHA handling function, uses a different URL construction method

than `askCaptcha`.

* **`takeInput(name, headname)`:** Takes user input (likely a password).

* **`InputWindow(xbmcgui.WindowDialog)`:** Custom class that manages a dialog window, probably for displaying CAPTCHA or other user interactions.

* **`getEpochTime()`:** Returns the current epoch time (seconds since epoch).

* **`getEpochTime2()`:** Returns the current epoch time (no multiplication by 1000).

* **`get_params()`:** Parses URL parameters.

* **`getFavorites()`:** Reads favorites data from a JSON file.

* **`addFavorite(name, url, iconimage, fanart, mode, playlist=None, regexs=None)`:** Adds a favorite to a JSON file.

* `rmFavorite(name)`: Removes a favorite based on its name.

* `urlsolver(url)`: Resolves a URL, potentially to check for validity or handle redirects.

* `tryplay(url, listitem, dialog=None)`: Plays a video URL using `xbmc.Player`. It seems to handle YouTube links directly.

* `play_playlist(name, mu_playlist, queueVideo=None)`: Plays a playlist of videos. Handles displaying progress updates during playback. Also handles both regular and regex-based playlist URLs.

* `download_file(name, url)`: Attempts to download a file based on URL and name. It seems to lack robust error handling, suggesting potential vulnerabilities.

* `_search(url, name)`: Performs a search for video sources on different platforms (YouTube, Dailymotion, Vimeo).

* `addDir(name, url, mode, iconimage, fanart, description, genre, date, credits, showcontext=False, regexs=None, reg_url=None, allinfo={})` : This is the core function for adding items to the addon's directory. It constructs the URL for adding the item, handles different modes, and includes metadata for each item.

Control Flow

The main control flow is driven by calls to `getSources()` and related functions that process source data. The functions that retrieve data from URLs (`makeRequest`, `getSoup`, `getURL`) all follow a similar pattern: make the request, check for errors, handle charset encoding and decoding, and then possibly process the response using regular expressions or other means. The `getItems` function iterates through a list of items, adds them to a playlist or directory using `addDir`. Error handling is minimal or non-existent in several places (a common trait in malware to increase execution speed).

Data Structures

The primary data structures are lists and dictionaries. Lists are used to store arrays of strings, sources, and items. Dictionaries store key-value pairs for metadata (such as titles, URLs, thumbnails, and descriptions). JSON is used for serializing and deserializing data to and from files.

Malware Family Suggestion

Given its functionality as a downloader/dropper with network communication and minimal error handling, this code is strongly suggestive of a **downloader/dropper malware**. It might belong to a broader family of video streaming-related malware which often use similar techniques for obtaining and delivering content, or a more generalized info-stealing malware family if the target is user data rather than video content. The heavy obfuscation points to an intent to evade detection.