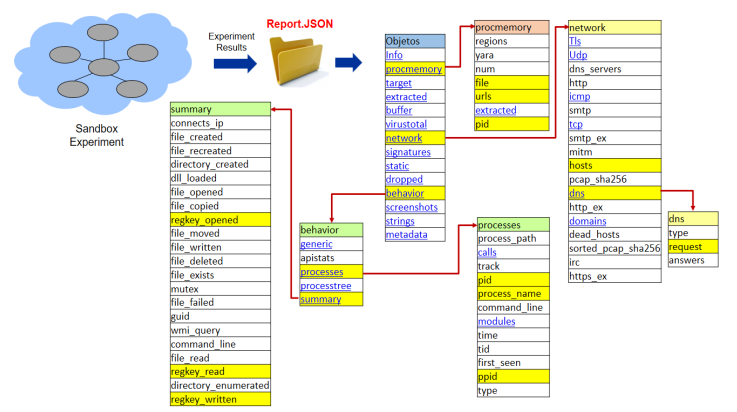
**Cuckoo Feature Extraction**

**Written by:** Jaramillo Ronny

This document describes the feature extraction process that was used to extract data from various JSON reports generated with the Cuckoo Sandbox tool.

It is important to first understand the general structure of the JSON reports that are generated by the tool. The following image shows the general structure contained in any given JSON report. It can be found in [1].



As we can see in the image, a report has a tree-based structure. In this first stage, the features that are marked with yellow were selected.

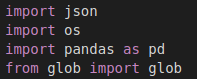
The first level of a contains several categories such as ‘Info’, ‘procmemory’, etc. So it was necessary to first visualize the type of data that is stored in each category to begin the extraction process.

In this case, we used the python programming language since its community provides a lot of libraries that are very useful when it comes to data analysis. The first thing that we needed to do is set up an environment with Python ver. 3.8.5 or higher and the following libraries:

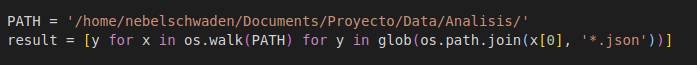
* pandas (ver. 1.1.1 or higher)
* numpy (ver. 1.8.5 or higher)

The ‘json’, ‘os’ and ‘glob’ libraries are also required but they are included in the python standard library so there is no need to explicitly install them.

Once we had our environment up and running, we start off by including the libraries in a new python file.



Then we load all the JSON Cuckoo Sandbox reports that were located in a certain directory. We have to do this recursively since there were nested directories and we have to look up for every json file contained in a given directory. To do this, we used the ‘os’ library method called ‘walk’ which allow us to “walk” through a directory to search for different things. We also used the ‘glob’ method from the “glob” library to extract the path.



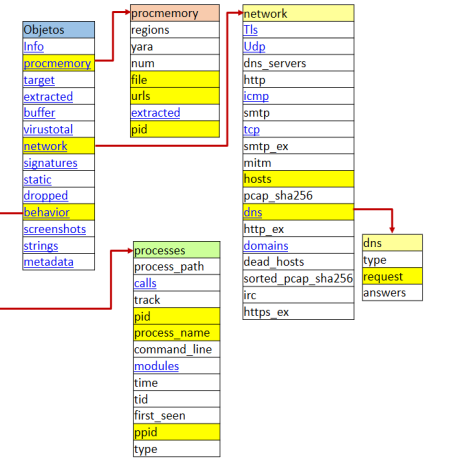
The ‘PATH’ variable contains a string which specifies the directory in which we are going to search for the JSON reports. We then use a comprehension list to loop through all the directory findings and to only retrieve all the absolute paths of every JSON file that is found in the given directory. These paths will be saved in an array in the ‘result’ variable.

Now, to extract the features that were mentioned earlier, we need to know which type of data we’re working with. To do this, we just load a single report into one variable.



The ‘data’ variable loads every single piece of information contained in the specified JSON report as a dictionary. This way, we can obtain all the information contained in any feature from the JSON report. However, to build the dataset, we need to know the type of data that a given feature has. We need to inspect the JSON sample manually.

For example, let’s take the ‘network’ category.

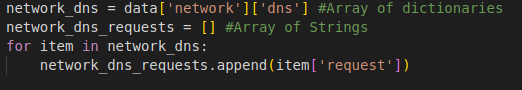


The ‘network’ category contains features like ‘hosts’ and ‘dns’. ‘dns’ contains the ‘request’ feature. We need to extract all data contained in these features but to do this, it is necessary to know the type of data that is contained in them. If we were required to obtain all the ‘request’ feature data, we need to check if the data type refers to an integer, string, array, list, dictionary, etc. So that we can loop through this feature properly.



As we can see, to obtain the ‘request’ feature, we need to first obtain the ‘network’ category from the ‘data’ variable. We then need to obtain the ‘dns’ feature from the ‘network’ category. Since this ‘dns’ extraction will give us an array of dictionaries, we will need to loop through each dictionary so that we can obtain the ‘request’ attribute from every dictionary.

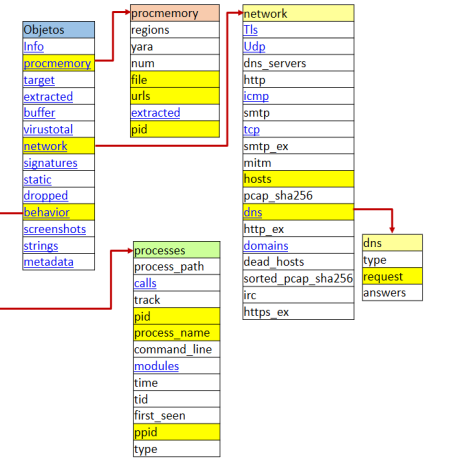
This lead us to the following piece of code.



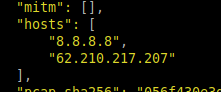
The ‘data’ variable contained all the information of the JSON report. We then access to the ‘network’ category by specifying this category inside the first pair of brackets in the ‘network\_dns’ variable. We then specify the ‘dns’ feature inside another pair of brackets to select all the ‘dns’ registries.

This ‘network\_dns’ variable contains the list of dictionaries shown in a previous image. Now we need to loop through each of these items to obtain only the ‘request’ feature. This is accomplished with a ‘for’ loop which then adds each ‘request’ to the ‘network\_dns\_requests’ which results in all of the requests contained in that JSON report.

Let’s look at another example.

****

Say we want to extract the ‘hosts’ feature from the ‘network’ category. To do this, we first need to check manually the type of data that is stored in this feature.

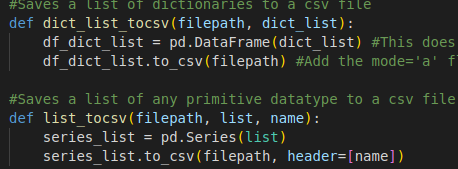
****

As we can see, this is a simple array of strings with all the hosts. In this case, is it much more simple since we can access this particular feature directly as follows.

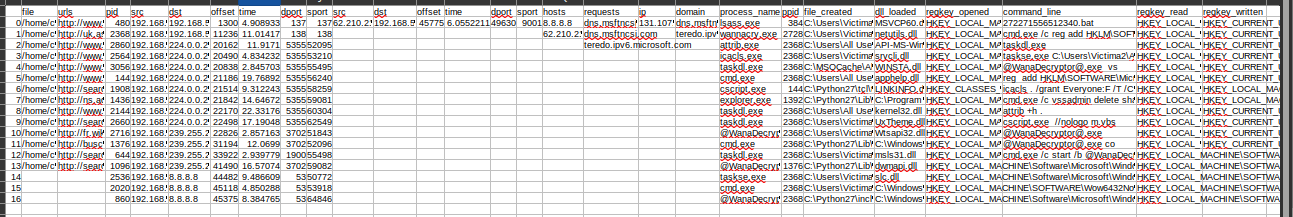


The ‘data’ variable contained all of the JSON report data. We then can access the ‘network’ category by including its name in a pair of brackets. We then can access the hosts lists by doing the same thing with the ‘hosts’ feature. This process is then repeated for all the selected features which will make us end up with the proper code for each category with certain exceptions in certain features.

Now we have all the data stored in various variables and we need to write all the extracted data in a certain file. To do this, we use the following functions.

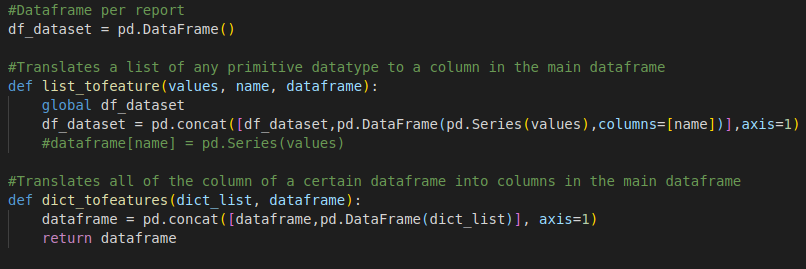


These two functions can write the data contained in a list of any primitive data type or contained in a list of dictionaries to a CSV extension file.

****

This approach was not suitable because the ‘tcp’ and ‘udp’ features contained a list of dictionaries which needed to be written as a group of data. This is because it was needed to assign an ID to each group of data in these two categories. Also there was two instances of the ‘pid’ feature: one contained in the ‘behavior’ category and another in the ‘procmemory’ category and this method caused the ‘pid’ feature from the ‘procmemory’ category to be overwritten by the ‘pid’ feature from the ‘behavior’ category.

This is why we needed to work with Dataframes and Series. A dataframe is like a matrix and series are like arrays. These two datatypes can be encountered when working with the Pandas library. This methods allowed us to obtain and write data properly.



What we do now is that we create one dataframe per report. This dataframe will contain all of the extracted features as columns. This is achieved by using the two methods in the image. Every time we retrieve a certain feature, we call one of these two methods and they map them to one column in the dataframe.

Once we looped through all of the selected features, we need to store the new dataframe somewhere. It is being stored in a list of dataframes which contains each dataframe generated with each processed report.

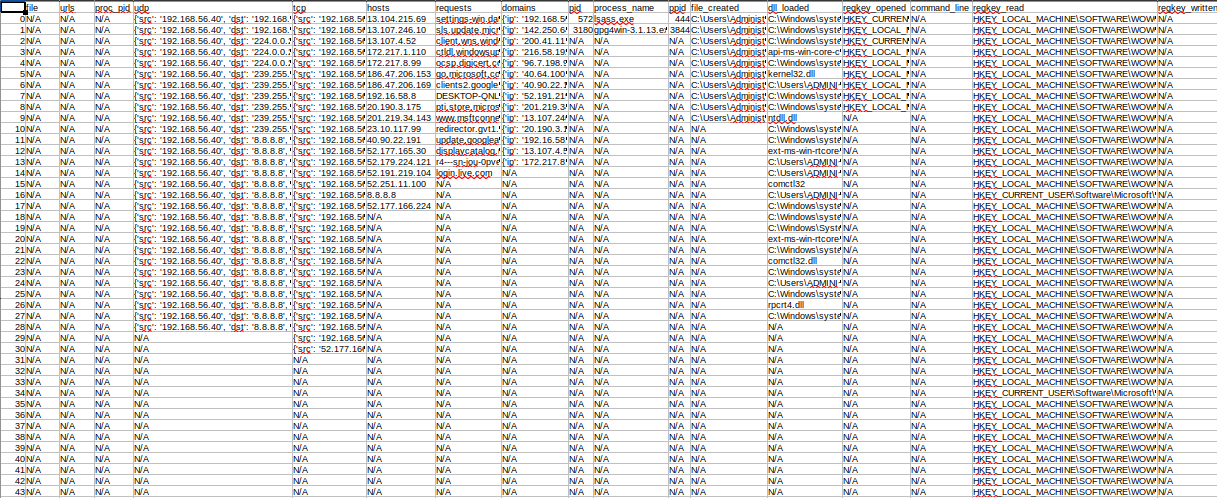




Once we looped through all of the reports, we just concatenate all of the dataframes in one and write it in a CSV file.



This results in a final CSV file that contains all data contained in the extracted features from all of the given JSON reports.

****

**References:**

[1] Silva, Juan A Herrera; Veloz, Freddy Daniel Bazante; López, Lorena Isabel Barona; Caraguay, Ángel Leonardo Valdivieso; Hernández-Álvarez, Myriam. “Dataset de Ransomware basado en análisis dinámico”, risti, 2019.