Gino Placella
4/23/21
CSEC 759

Volatility CSV Plugin Support

For the final project, the final decision made was to implement an output format for the memory analysis tool, Volatility. After working with the tool the data output was somewhat difficult to interpret and hard to work with. The main issue with the output from Volatility was, the output came from standard out on the command line. This means that when the python program is run the output is placed on the command line which makes it hard for a user to navigate through the data as well as exporting the data to work with it. This led me to implement a data format that is easily comprehensible and simple to work with. The format chosen was a Comma Separated Values format.

Volatility as an open-source tool for volatile memory analysis. It consists of a single, cohesive framework that analyzes Random Access Memory (RAM) dumps for 32/64 bit Windows, Linux, Mac, and even android systems [1]. The framework is implemented in Python and used to analyze crash dumps, raw dumps, VMware, and in our case Sandbox dumps. Since this tool analyzes the dumps that result from malicious activity on a system, it is generalized as a dynamic analysis tool. This is because it is used after the malware is executed in the testing environment. The output from the tool depends on the use of the several plugins that can be used in association with the tool. The capabilities that Volatility provides are vast, allowing the researcher to see commands that were executed, processes on the system, and more.

The architecture of Volatility consists of several python files and libraries that are all interconnected either by import statements or pass by references. When adding additional formatting options for Volatility the author must ensure that there are plugins configured to allow the rendering of the output. As stated in the GitHub documentation for the tool, there are no plugins with alternate output formats pre-configured for use, so you'll need to add a function named `render_FORMAT` for each plugin before using the `--output=FORMAT` plugin for the tool [1]. This flag renders the output that by default, uses text renderers to standard output. Therefore, for a CSV format, the function needed to be added is `render_csv` for the `--output=csv` plugin. This function also handles output redirection to a file. These redirections use the `--output-file=FILE` plugin when using the Volatility tool. This function declaration should exist in the “commands.py” python file for Volatility. This file executes each plugin command used within the tool such as the format.

After declaring the execution of the plugin command (see Figure 4), a file needs to be created that contains the instructions that are being executed. This file is contained within the renderers directory within Volatility that contains all of the different formats that the output can be rendered in. For the solution presented in this paper, the file created was called "csvFormat.py". This file contains the main code to be executed that renders the output into the corresponding format as well as taking care of the data writing process. Once this file is created, the author must navigate back to the "commands.py" file to import the main class of the renderer file. In this case, the import contained the main class of "csvFormat.py", which happened to be `CSVRenderer` as this is the specified style for creating renderer files.

With the vast capabilities granted by Volatility, the tool has very few issues in terms of use. However, I noticed that most problems came when attempting to interpret and interact with the results. As compared with Cuckoo Sandbox, Cuckoo provides the user with two types of reports, an HTML and a JSON report. The HTML report is pretty straightforward and simple, providing the user with an organized and clean output of the information gathered during the analysis. The JSON report is more in-depth and requires more user interaction to gather the necessary information that the user is looking for. With Volatility, the tool does have a plugin to allow for the generation of an HTML report, however, a CSV file provides the user with an organized structure that can be accessed via spreadsheets. Spreadsheet reports give users the ability to filter the data contained in the cells, organize the data, format cells based on specific data entries, and more. In many situations, spreadsheets are a frequently used report type for businesses and corporations, especially those that would produce massive amounts of data.

The solution presented in this paper is a general solution to the formatting problem provided by Volatility. As tested, this solution provides simple and clean output for the functionality of Volatility. The malware being tested using Volatility should have no impact on executing the CSV plugins created for the tool as the analysis is completed before the execution of the plugins. The addition to the tool uses the render check algorithm that is already implemented within Volatility to check if the format needs to be rendered in a different format. If that is true and the specified format via the `--output=` plugin is 'csv', then a "try" statement is executed to attempt to render the output via the `CSVRenderer` class that exists in the "csvFormat.py" file. This file uses several pre-defined python libraries to assist in parsing the

data, formatting the data, and writing the data to the CSV file specified in the `--output-file=` plugin.

```
def render_csv(self, outfd, data):
    try:
        self._render(outfd, CSVRenderer(), data)
    except NotImplementedError, why:
        debug.error(why)
    except TypeError, why:
        debug.error(why)
```

Figure 1: render_csv

The basic overview of the file shows the use of two functions, the `render_row` function and the `render` function. The `render` function is the main function of the file. This function does most of the work in parsing, formatting and writing the data. The issue that came about when parsing the data was contained within a Tree Structure used by the file. This structure was very difficult to navigate and most of the data, once parsed, was outputted in a JSON format. In order to parse through the tree, the render_row function was created. This function appends the values contained in the node of a tree that has been visited to an empty list. This list, once full, contains both the columns of the tree which are not needed, and the rows. The rows are the values that we need as the analyst. Since, the list is a JSON structure the values being stored at 'rows' were simple to ascertain. These values were then appended to the another empty list. The reason for having a list of lists is because the CSV module in python writes a list to the CSV file. This means that if we were to use a single list, the data would be written incorrectly (each character would be separated by a comma instead of the groups of data). Once the list of lists is created, the `render` function proceeds to write the values into their own separate rows.

```
import StringIO
import csv
import json
from volatility.renderers.basic import Renderer

__author__ = 'Gino Placella'

class CSVRenderer(Renderer):

    def __init__(self):
        pass

    def render_row(self, node, accumulator):
        return accumulator + [node.values]

    def render(self, outfd, data):
        json = StringIO.StringIO()
        json_input = {"columns": [column.name for column in data.columns], "rows": data.visit(None, self.render_row, [])}
        info = []
        for i in range(0, len(json_input['rows'])):
            info.append([])
        for i in range(0, len(json_input['rows'])):
            temp = str(json_input['rows'][i])
            info[i].append(temp.split("RowStructure", 1)[1])
        writer = csv.writer(outfd, delimiter=",")
        writer.writerows(info)
```

Figure 2: csvFormat.py

As seen in Figure 3, the LibreOffice application on Ubuntu contains the output of the Process List provided by Volatility. Each process is written to its own separate row. The addition provided in this project only requires that the JSON, CSV, and StringIO python modules are installed on the system Volatility is being run on. This is to ensure proper function of the rendering plugins. It is also highly recommended that the CSV files are opened within a spreadsheet application to see the writing process as reading the file in the terminal will not show how the output was written. The main purpose of this addition is to help future researchers and analysts with interpreting and filtering results administered by the Volatility tool. As seen in the Google Rapid Response tool, CSV files provide effective ways for interpreting large amounts of results. This was something that seemed missing with the Volatility tool as many of the formats and general output lacked in-depth analysis tools and filtering.

Gino Placella
4/23/21
CSEC 759

Appendix

A
'offset_v_=2174535728L, name='System', pid=4, ppid=0, thds=51, hnds=254, sess=-1, wow64=0, start='', exit='')
'offset_v_=2172536432L, name='smss.exe', pid=360, ppid=4, thds=3, hnds=19, sess=-1, wow64=0, start='2008-11-26 07:38:11 UTC+0000', exit='')
'offset_v_=2172763512L, name='csrss.exe', pid=596, ppid=360, thds=10, hnds=322, sess=0, wow64=0, start='2008-11-26 07:38:13 UTC+0000', exit='')
'offset_v_=2172825856L, name='winlogon.exe', pid=620, ppid=360, thds=16, hnds=503, sess=0, wow64=0, start='2008-11-26 07:38:14 UTC+0000', exit='')
'offset_v_=2172893816L, name='services.exe', pid=672, ppid=620, thds=15, hnds=245, sess=0, wow64=0, start='2008-11-26 07:38:15 UTC+0000', exit='')
'offset_v_=2172501040L, name='lsass.exe', pid=684, ppid=620, thds=21, hnds=347, sess=0, wow64=0, start='2008-11-26 07:38:15 UTC+0000', exit='')
'offset_v_=2173017456L, name='svchost.exe', pid=844, ppid=672, thds=19, hnds=198, sess=0, wow64=0, start='2008-11-26 07:38:18 UTC+0000', exit='')
'offset_v_=2172900192L, name='svchost.exe', pid=932, ppid=672, thds=10, hnds=229, sess=0, wow64=0, start='2008-11-26 07:38:18 UTC+0000', exit='')
'offset_v_=2173313792L, name='svchost.exe', pid=1064, ppid=672, thds=63, hnds=1308, sess=0, wow64=0, start='2008-11-26 07:38:20 UTC+0000', exit='')
'offset_v_=2172616096L, name='svchost.exe', pid=1164, ppid=672, thds=5, hnds=77, sess=0, wow64=0, start='2008-11-26 07:38:23 UTC+0000', exit='')
'offset_v_=2172708592L, name='svchost.exe', pid=1264, ppid=672, thds=14, hnds=209, sess=0, wow64=0, start='2008-11-26 07:38:25 UTC+0000', exit='')
'offset_v_=2172429432L, name='explorer.exe', pid=1516, ppid=1452, thds=12, hnds=362, sess=0, wow64=0, start='2008-11-26 07:38:27 UTC+0000', exit='')
'offset_v_=2171500008L, name='spoolsv.exe', pid=1648, ppid=672, thds=12, hnds=112, sess=0, wow64=0, start='2008-11-26 07:38:28 UTC+0000', exit='')
'offset_v_=2171271264L, name='VMwareTray.exe', pid=1896, ppid=1516, thds=1, hnds=26, sess=0, wow64=0, start='2008-11-26 07:38:31 UTC+0000', exit='')
'offset_v_=2171270216L, name='VMwareUser.exe', pid=1904, ppid=1516, thds=1, hnds=28, sess=0, wow64=0, start='2008-11-26 07:38:31 UTC+0000', exit='')
'offset_v_=2171212736L, name='VMwareService.e', pid=1756, ppid=672, thds=3, hnds=45, sess=0, wow64=0, start='2008-11-26 07:38:45 UTC+0000', exit='')
'offset_v_=2170902496L, name='alg.exe', pid=512, ppid=672, thds=6, hnds=105, sess=0, wow64=0, start='2008-11-26 07:38:53 UTC+0000', exit='')
'offset_v_=2170829608L, name='wuauclt.exe', pid=1372, ppid=1064, thds=8, hnds=225, sess=0, wow64=0, start='2008-11-26 07:39:38 UTC+0000', exit='')
'offset_v_=2170872744L, name='wscntfy.exe', pid=560, ppid=1064, thds=1, hnds=31, sess=0, wow64=0, start='2008-11-26 07:44:57 UTC+0000', exit='')

Figure 3: Process List CSV

```
$ volatility -f "Windows XP SP3 x86.img" profile=WinXPSP3x86 pslist --output=csv --output-file=ps.csv
```

Figure 4: Usage with Plugins

```
Volatility Foundation Volatility Framework 2.6
Outputting to: ps.csv
```

Figure 5: Command Output

Gino Placella
4/23/21
CSEC 759

References

[1] Case, Andrew. “Volatilityfoundation/Volatility.” GitHub, VolatilityFoundation, github.com/volatilityfoundation/volatility/wiki.