Dynamic Malware Detection

I. Introduction

Traditional malware detection methods based on signature matching are becoming less effective due to advanced malware generation and obfuscation techniques. To counter this, the authors present a method that employs dynamic analysis to extract sequences of API calls. They then use a color mapping technique based on API category and occurrence to convert these sequences into feature images that visually represent malware behavior. These feature images serve as inputs to train a Convolutional Neural Network (CNN), which can classify the images into nine different malware families, each with 1000 variants.

Results show that the visualization and deep learning model effectively classifies malware, providing an innovative technique compared to traditional methods.

II. Set Environment

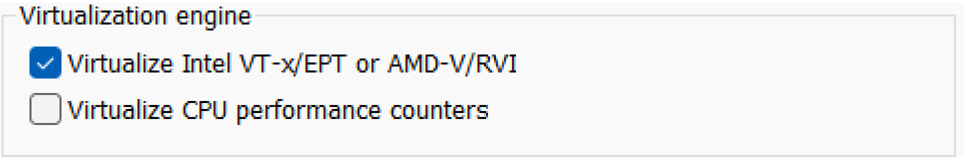
A. Download VMware (https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html.html)

B. Download Ubuntu ISO file (https://docs.ossii.com.tw/books/ubuntu-serve-2004/page/ubuntu-server-2004-iso)

C. Download Win7_ultimate.iso file

D. Download cuckoo_setup_virtualenv.sh file

1. Difficulty: it will be incompatible with Hyper-V, so VMware cannot be opened.

   Virtualization engine

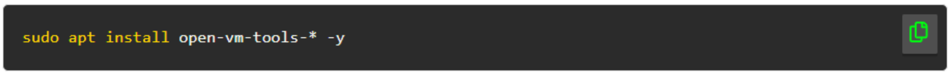
   ☑ Virtualize Intel VT-x/EPT or AMD-V/RVI

   □ Virtualize CPU performance counters

   Solution: close Hyper-V (https://www.vmware.com/products/workstation-player/workstatio n-player-evaluation.html.html)

2. Difficulty: cannot directly paste text in Linux.

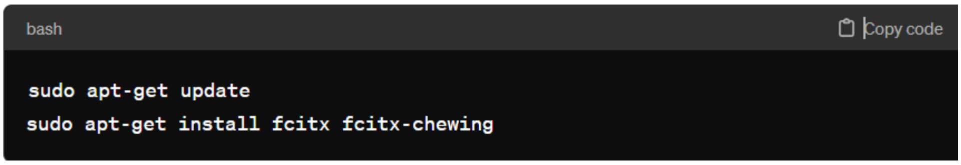
   Solution: install the open-vm-tools package

```
sudo apt install open-vm-tools-* -y
```

3. Difficulty: cannot directly type in Chinese in Linux

   Solution: download Fcitx5



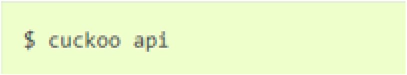
```
sudo apt-get update
sudo apt-get install fcitx fcitx-chewing
```

4. Difficulty: unable to control the mouse in the virtual machine

   Solution: ctrl + alt

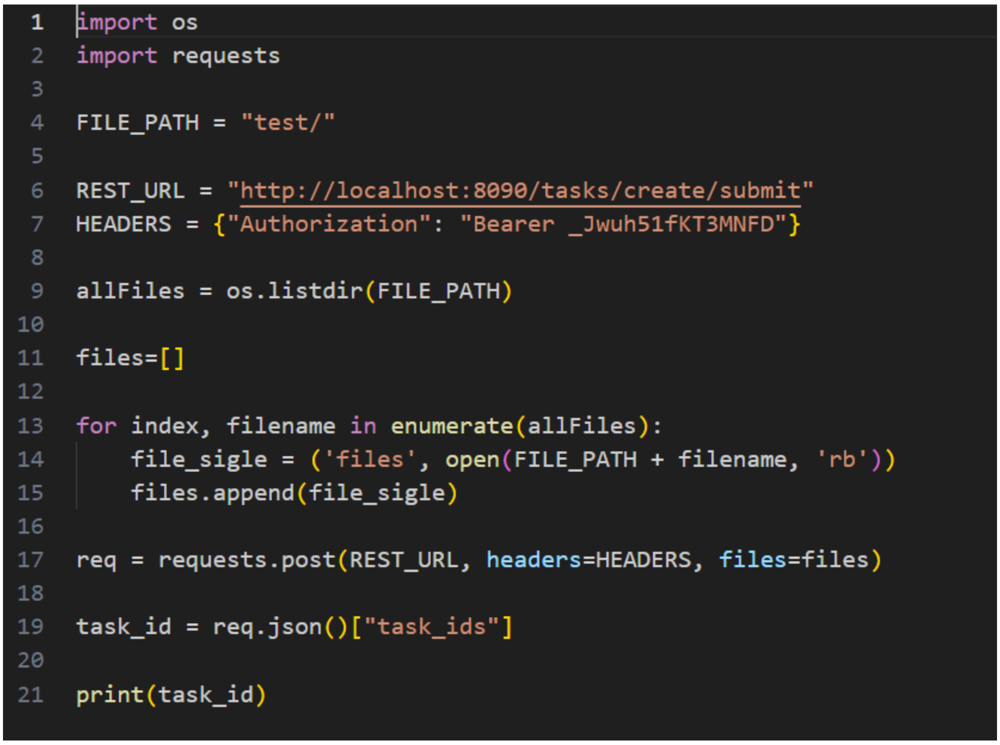
III. Collect the Information of API Call Sequences

A. Use cuckoo api



```
$ cuckoo api
```

B. Upload malware



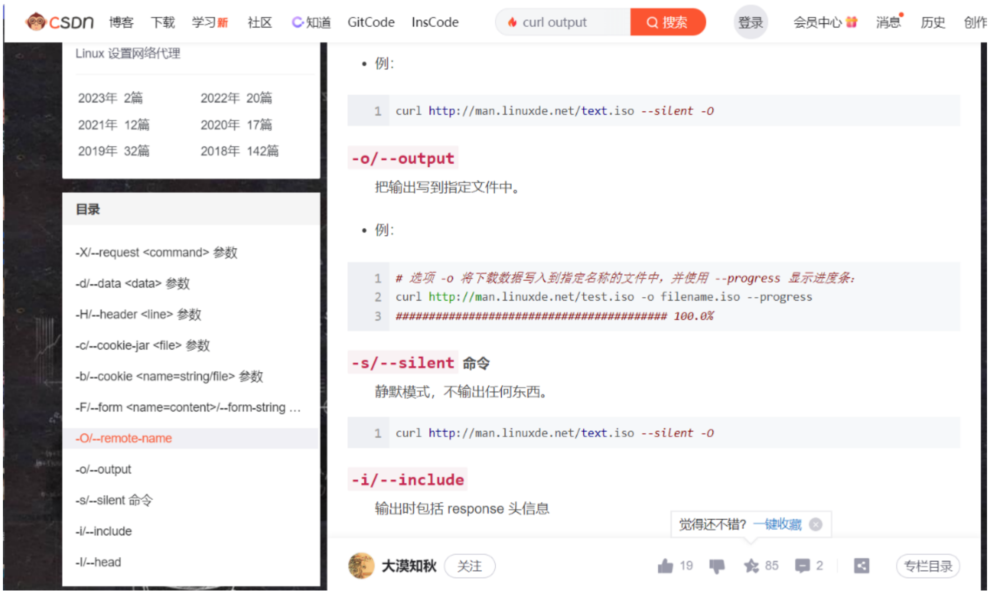
```
1   import os
2   import requests
3
4   FILE_PATH = "test/"
5
6   REST_URL = "http://localhost:8090/tasks/create/submit"
7   HEADERS = {"Authorization": "Bearer _Jwuh51fKT3MNFD"}
8
9   allFiles = os.listdir(FILE_PATH)
10
11  files=[]
12
13  for index, filename in enumerate(allFiles):
14      file_sigle = ('files', open(FILE_PATH + filename, 'rb'))
15      files.append(file_sigle)
16
17  req = requests.post(REST_URL, headers=HEADERS, files=files)
18
19  task_id = req.json()["task_ids"]
20
21  print(task_id)
```

1. Line 13: Loop through all the files in the directory
2. Line 14: Open each file in binary read mode and create a tuple
3. Line 17: Send a POST request to the REST_URL with the headers and the list of file tuples
4. Line 19: Extract the 'task_ids' from the JSON response, because many files so we need to add 's' at the end

C. Download json file

```
curl -H "Authorization: Bearer S4MPL3" http://localhost:8090/tasks/report/1
```

It will only appear at the command line, so we need to add '**-o/--output**' at the end so the file will download to our Linux



Linux 设置网络代理

2023年 2篇　2022年 20篇
2021年 12篇　2020年 17篇
2019年 32篇　2018年 142篇

目录

-X/--request <command> 参数
-d/--data <data> 参数
-H/--header <line> 参数
-c/--cookie-jar <file> 参数
-b/--cookie <name=string/file> 参数
-F/--form <name=content>/--form-string ...
-O/--remote-name
-o/--output
-s/--silent 命令
-i/--include
-I/--head

- 例:

```
curl http://man.linuxde.net/text.iso --silent -O
```

-o/--output

把输出写到指定文件中。

- 例:

```
# 选项 -o 将下载数据写入到指定名称的文件中，并使用 --progress 显示进度条:
curl http://man.linuxde.net/test.iso -o filename.iso --progress
######################################### 100.0%
```

-s/--silent 命令

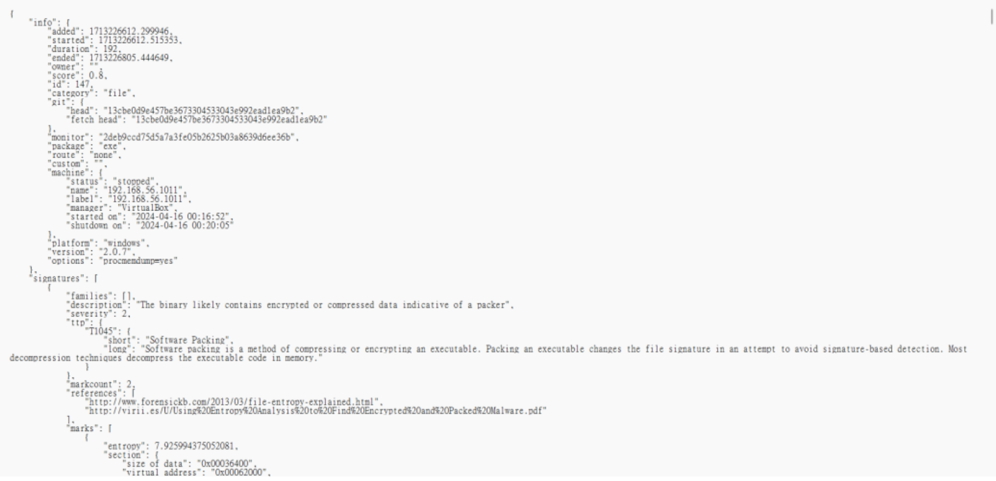
静默模式，不输出任何东西。

```
curl http://man.linuxde.net/text.iso --silent -O
```

-i/--include

输出时包括 response 头信息

D. Part of the outcome

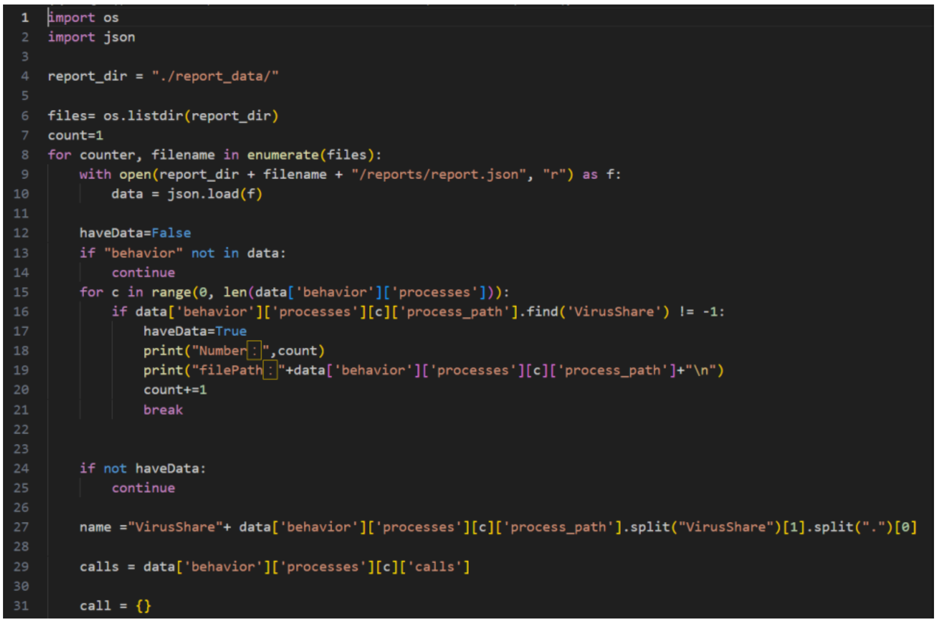


```
{
    "info": {
        "added": 1713226612.299946,
        "started": 1713226612.515353,
        "duration": 192,
        "ended": 1713226805.444649,
        "owner": "",
        "score": 0.8,
        "id": 147,
        "category": "file",
        "git": {
            "head": "13cbe0d9e457be3673304533043e992ead1ea9b2",
            "fetch_head": "13cbe0d9e457be3673304533043e992ead1ea9b2"
        },
        "monitor": "2deb9ccd75d5a7a3fe05b2625b03a8639d6ee36b",
        "package": "exe",
        "route": "none",
        "custom": "",
        "machine": {
            "status": "stopped",
            "name": "192.168.56.1011",
            "label": "192.168.56.1011",
            "manager": "VirtualBox",
            "started_on": "2024-04-16 00:16:52",
            "shutdown_on": "2024-04-16 00:20:05"
        },
        "platform": "windows",
        "version": "2.0.7",
        "options": "procmemdump=yes"
    },
    "signatures": [
        {
            "families": [],
            "description": "The binary likely contains encrypted or compressed data indicative of a packer",
            "severity": 2,
            "ttp": {
                "T1045": {
                    "short": "Software Packing",
                    "long": "Software packing is a method of compressing or encrypting an executable. Packing an executable changes the file signature in an attempt to avoid signature-based detection. Most
decompression techniques decompress the executable code in memory."
                }
            },
            "markcount": 2,
            "references": [
                "http://www.forensickb.com/2013/03/file-entropy-explained.html",
                "http://virii.es/U/Using%20Entropy%20Analysis%20to%20Find%20Encrypted%20and%20Packed%20Malware.pdf"
            ],
            "marks": [
                {
                    "entropy": 7.925994375052081,
                    "section": {
                        "size_of_data": "0x00036400",
                        "virtual_address": "0x00062000",
```

IV. Transformation

A. Transform report to json

1. Process JSON reports related to malware behavior, specifically looking for reports that contain certain characteristics



```python
1  import os
2  import json
3
4  report_dir = "./report_data/"
5
6  files= os.listdir(report_dir)
7  count=1
8  for counter, filename in enumerate(files):
9      with open(report_dir + filename + "/reports/report.json", "r") as f:
10         data = json.load(f)
11
12     haveData=False
13     if "behavior" not in data:
14         continue
15     for c in range(0, len(data['behavior']['processes'])):
16         if data['behavior']['processes'][c]['process_path'].find('VirusShare') != -1:
17             haveData=True
18             print("Number:",count)
19             print("filePath:"+data['behavior']['processes'][c]['process_path']+"\n")
20             count+=1
21             break
22
23
24     if not haveData:
25         continue
26
27     name ="VirusShare"+ data['behavior']['processes'][c]['process_path'].split("VirusShare")[1].split(".")[0]
28
29     calls = data['behavior']['processes'][c]['calls']
30
31     call = {}
```

a. Line 7: Initialize a counter to keep track of the number of reports processed

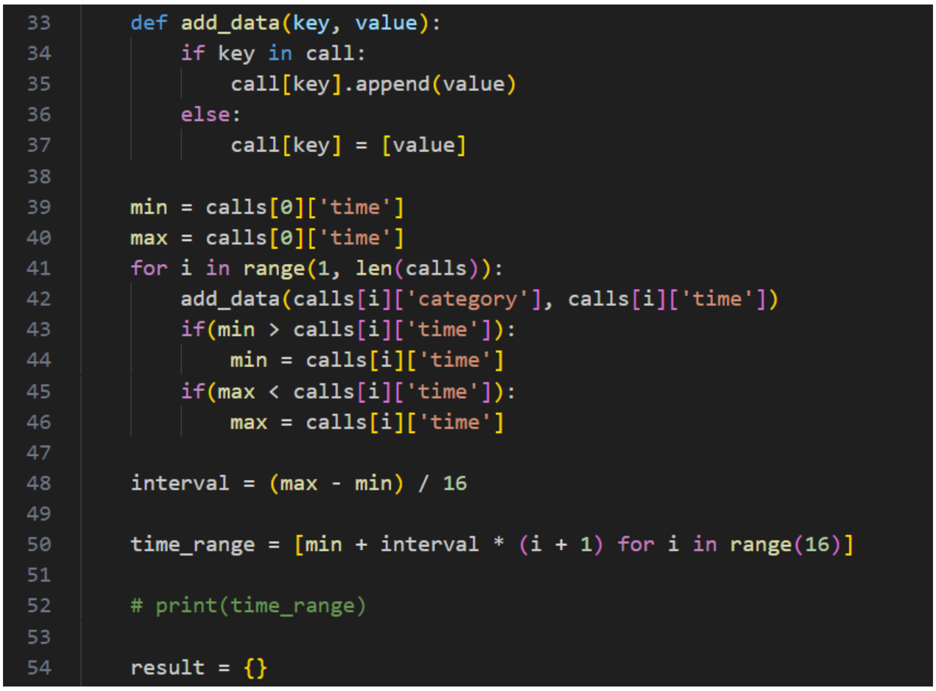
b. Line 12: Flag to check if relevant data is present

c. Line 13: Continue to next file if 'behavior' key is not in the JSON data

d. Line 16: Check if the 'process_path' contains 'VirusShare'

e. Line 17: Set flag as true since relevant data is found

2. Aggregates API calls by category and timestamps, then calculates the time intervals to prepare for temporal analysis of malware activity



```python
    def add_data(key, value):
        if key in call:
            call[key].append(value)
        else:
            call[key] = [value]

    min = calls[0]['time']
    max = calls[0]['time']
    for i in range(1, len(calls)):
        add_data(calls[i]['category'], calls[i]['time'])
        if(min > calls[i]['time']):
            min = calls[i]['time']
        if(max < calls[i]['time']):
            max = calls[i]['time']

    interval = (max - min) / 16

    time_range = [min + interval * (i + 1) for i in range(16)]

    # print(time_range)

    result = {}
```

a. Line 34: If the key exists, append the value to its list

b. Line 36: If the key does not exist, create a new entry with the value in a list

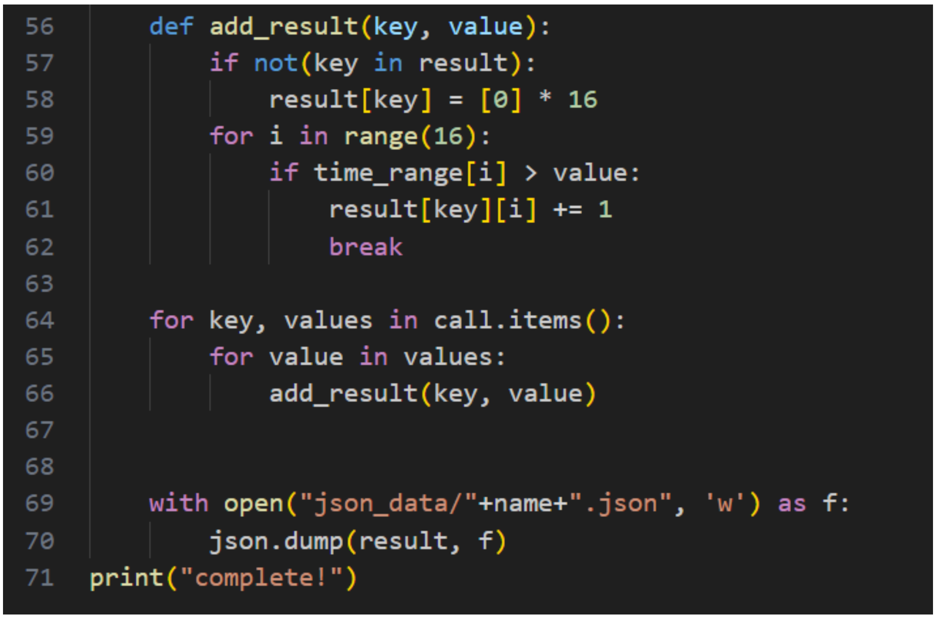
c. Line 38 to 39: Initialize 'min' and 'max' to the first call time

d. Line 41: Loop over the calls to find the minimum and maximum times

e. Line 48: Calculate the interval for dividing time into 16 parts

f. Line 50: Create a list of time ranges

3. This part of the script populates a data structure with categorized API call frequencies over time and then writes this summarized information to a JSON file for each malware sample



```
56      def add_result(key, value):
57          if not(key in result):
58              result[key] = [0] * 16
59          for i in range(16):
60              if time_range[i] > value:
61                  result[key][i] += 1
62                  break
63
64      for key, values in call.items():
65          for value in values:
66              add_result(key, value)
67
68
69      with open("json_data/"+name+".json", 'w') as f:
70          json.dump(result, f)
71  print("complete!")
```

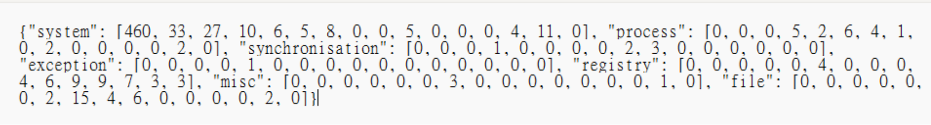
a. Line 57: If the key does not exist, create a new entry with a list of 16 zeros

b. Line 59: Loop over the time ranges and increment the corresponding time slot

c. Line 64: Loop over each category and add the result based on the time of the call

d. Line 64: Open a new JSON file to write the results

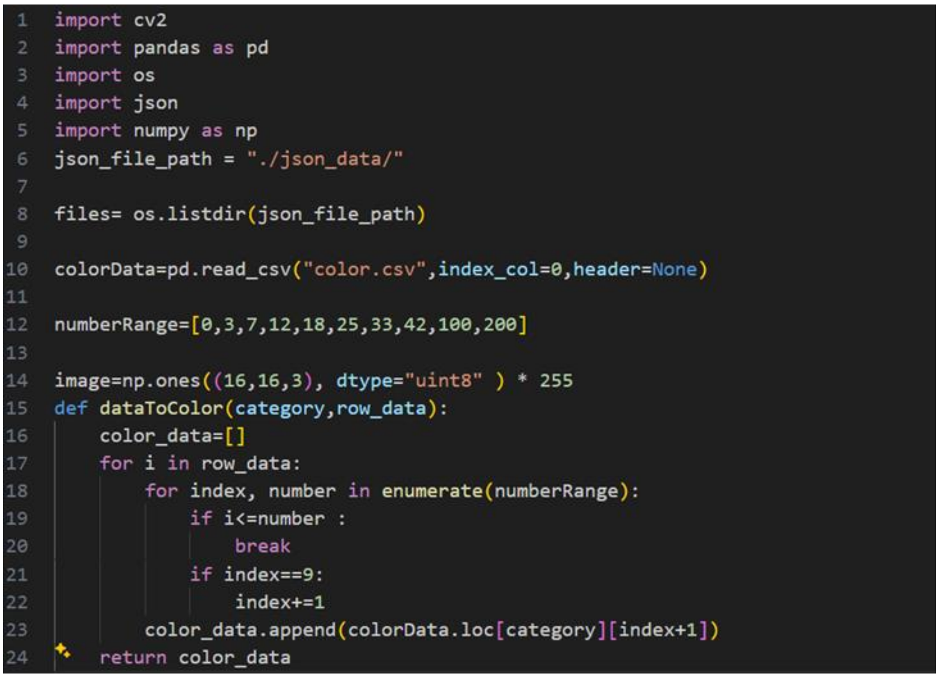
4. Example of the outcome



```
{"system": [460, 33, 27, 10, 6, 5, 8, 0, 0, 5, 0, 0, 0, 4, 11, 0], "process": [0, 0, 0, 5, 2, 6, 4, 1,
0, 2, 0, 0, 0, 0, 2, 0], "synchronisation": [0, 0, 0, 1, 0, 0, 0, 0, 2, 3, 0, 0, 0, 0, 0, 0],
"exception": [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "registry": [0, 0, 0, 0, 0, 4, 0, 0, 0,
4, 6, 9, 9, 7, 3, 3], "misc": [0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 1, 0], "file": [0, 0, 0, 0, 0,
0, 2, 15, 4, 6, 0, 0, 0, 0, 2, 0]}
```

B. Transform json to image

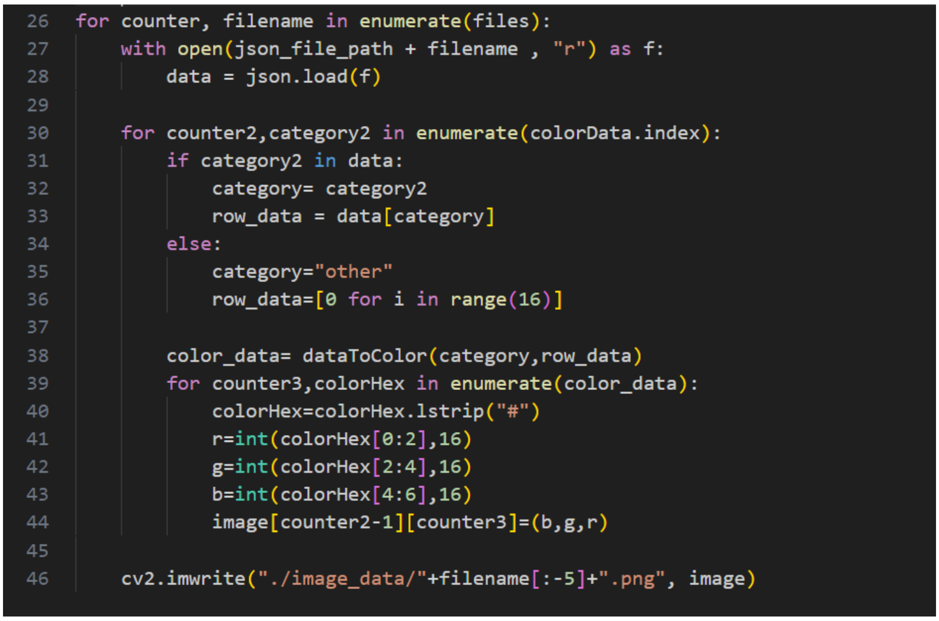
1. process JSON files that contain behavioral data, presumably for malware analysis. It is preparing a color mapping system to translate numerical data into color information that will later be used to generate visualization



```
1  import cv2
2  import pandas as pd
3  import os
4  import json
5  import numpy as np
6  json_file_path = "./json_data/"
7
8  files= os.listdir(json_file_path)
9
10 colorData=pd.read_csv("color.csv",index_col=0,header=None)
11
12 numberRange=[0,3,7,12,18,25,33,42,100,200]
13
14 image=np.ones((16,16,3), dtype="uint8" ) * 255
15 def dataToColor(category,row_data):
16     color_data=[]
17     for i in row_data:
18         for index, number in enumerate(numberRange):
19             if i<=number :
20                 break
21             if index==9:
22                 index+=1
23         color_data.append(colorData.loc[category][index+1])
24     return color_data
```

   a. Line 12: Define the boundaries for different color ranges as a list
   b. Line 14: Create a 16x16 white image with three color channels (RGB), using 8-bit unsigned integers
   c. Line 15: Define a function that maps row data to specific colors based on a category
   d. Line 17: Check which range the data point falls into and break the loop once found
   e. Line 21: Adjust the index if it's the last specified range
   f. Line 23: Append the corresponding color for the category and range to the list

2. processes each JSON file to extract behavioral data, categorizes the data, translates categories into specific colors based on predefined ranges, constructs an image from these colors, and then saves the image. Each pixel in the resulting image corresponds to a specific category's occurrence intensity, visualized through color coding based on the frequency of that category's events



```
26  for counter, filename in enumerate(files):
27      with open(json_file_path + filename , "r") as f:
28          data = json.load(f)
29  
30      for counter2,category2 in enumerate(colorData.index):
31          if category2 in data:
32              category= category2
33              row_data = data[category]
34          else:
35              category="other"
36              row_data=[0 for i in range(16)]
37  
38          color_data= dataToColor(category,row_data)
39          for counter3,colorHex in enumerate(color_data):
40              colorHex=colorHex.lstrip("#")
41              r=int(colorHex[0:2],16)
42              g=int(colorHex[2:4],16)
43              b=int(colorHex[4:6],16)
44              image[counter2-1][counter3]=(b,g,r)
45  
46      cv2.imwrite("./image_data/"+filename[:-5]+".png", image)
```

a. Line 31: If the category is present in the data, use it

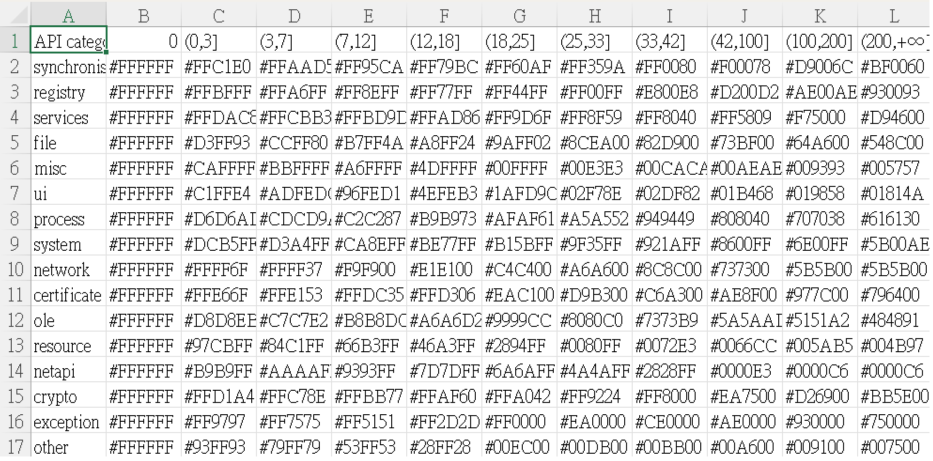
b. Line 34: If not, use the 'other' category

c. Line 38: Convert the row data to color data

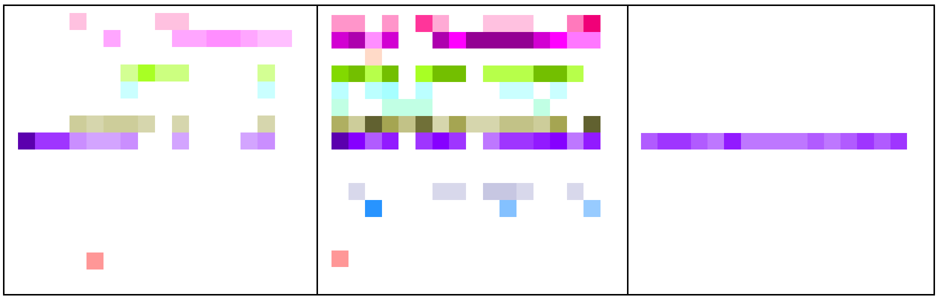
d. Line 34: For each color in the color data, convert it to RGB format

e. Line 44: Assign the color to the pixel in the image

3. Outcome



	A	B	C	D	E	F	G	H	I	J	K	L
1	API categ	0	(0,3]	(3,7]	(7,12]	(12,18]	(18,25]	(25,33]	(33,42]	(42,100]	(100,200]	(200,+∞]
2	synchronis	#FFFFFF	#FFC1E0	#FFAAD5	#FF95CA	#FF79BC	#FF60AF	#FF359A	#FF0080	#F00078	#D9006C	#BF0060
3	registry	#FFFFFF	#FFBFFF	#FFA6FF	#FF8EFF	#FF77FF	#FF44FF	#FF00FF	#E800E8	#D200D2	#AE00AE	#930093
4	services	#FFFFFF	#FFDAC8	#FFCBB3	#FFBD9D	#FFAD86	#FF9D6F	#FF8F59	#FF8040	#FF5809	#F75000	#D94600
5	file	#FFFFFF	#D3FF93	#CCFF80	#B7FF4A	#A8FF24	#9AFF02	#8CEA00	#82D900	#73BF00	#64A600	#548C00
6	misc	#FFFFFF	#CAFFFF	#BBFFFF	#A6FFFF	#4DFFFF	#00FFFF	#00E3E3	#00CACA	#00AEAE	#009393	#005757
7	ui	#FFFFFF	#C1FFE4	#ADFED(	#96FED1	#4EFEB3	#1AFD9C	#02F78E	#02DF82	#01B468	#019858	#01814A
8	process	#FFFFFF	#D6D6AI	#CDCD9.	#C2C287	#B9B973	#AFAF61	#A5A552	#949449	#808040	#707038	#616130
9	system	#FFFFFF	#DCB5FF	#D3A4FF	#CA8EFF	#BE77FF	#B15BFF	#9F35FF	#921AFF	#8600FF	#6E00FF	#5B00AE
10	network	#FFFFFF	#FFFF6F	#FFFF37	#F9F900	#E1E100	#C4C400	#A6A600	#8C8C00	#737300	#5B5B00	#5B5B00
11	certificate	#FFFFFF	#FFE66F	#FFE153	#FFDC35	#FFD306	#EAC100	#D9B300	#C6A300	#AE8F00	#977C00	#796400
12	ole	#FFFFFF	#D8D8EE	#C7C7E2	#B8B8DC	#A6A6D2	#9999CC	#8080C0	#7373B9	#5A5AAI	#5151A2	#484891
13	resource	#FFFFFF	#97CBFF	#84C1FF	#66B3FF	#46A3FF	#2894FF	#0080FF	#0072E3	#0066CC	#005AB5	#004B97
14	netapi	#FFFFFF	#B9B9FF	#AAAAF.	#9393FF	#7D7DFF	#6A6AFF	#4A4AFF	#2828FF	#0000E3	#0000C6	#0000C6
15	crypto	#FFFFFF	#FFD1A4	#FFC78E	#FFBB77	#FFAF60	#FFA042	#FF9224	#FF8000	#EA7500	#D26900	#BB5E00
16	exception	#FFFFFF	#FF9797	#FF7575	#FF5151	#FF2D2D	#FF0000	#EA0000	#CE0000	#AE0000	#930000	#750000
17	other	#FFFFFF	#93FF93	#79FF79	#53FF53	#28FF28	#00EC00	#00DB00	#00BB00	#00A600	#009100	#007500



V. Conclusion

We use an innovative method for malware classification using dynamic API call visualization and convolutional neural networks (CNNs). By dynamically analyzing malware to extract API call sequences and converting these into color-coded images, the method allows for visual and automatic analysis. The visual approach helps in understanding the behavioral patterns of malware, which are then classified into distinct families by a CNN. High classification accuracy achieved in experiments demonstrates the method's effectiveness, indicating a significant advancement over traditional signature-based detection systems.

VI. Difficult

Because we are preparing for the midterm; hence, we don’t have enough time to train the data. Next time we will start it earlier.