# Malware Visualization

This project will help in analysing about a Malware behavior and its functionality more easily with visualization than analysing the system log files manually.

Jay Patel (jmp840@nyu.edu)
Rahul Kala (rsk430@nyu.edu)
Shailesh Chaudhary (ssc496@nyu.edu)

**Project page (on Github):** https://github.com/nyu-cs6313-fall2015/Group-10
**Video:** http://tinyurl.com/zcy6t3w
**Working demo:** http://nyu-cs6313-fall2015.github.io/Group-10/

## What is the problem you want to solve and who has this problem?

Malwares are infections of computer system. When it enters the system it abuse the system which prevents use of resources of the system. The abusive use of resources by a malware is called its behaviour. A malware can create many unnecessary files, initiate many programs, send secret data of user over network to hacker, etc. In order to understand the working/behaviour of a malware, two approaches can be used: Static and Dynamic.

Static Approach: Examining the executable files of the malware without running them. It can basically identify the functionality of the malware but can miss important behaviours against sophisticated malwares.

Dynamic Approach: This approach executes the malware file in a controlled system and then study its functionality/behaviour by logging the activities. The log reports generated during analysis of malware consist of large number of API and system process calls. To analyse such a large file is very difficulty as to keep track of all process activities and it is time consuming. Whenever any new malware is detected, the antivirus companies needs to find a cure for it as soon as possible because the more time they take the more damage the malware does.

So, in-order to speedup the process of analysing and understanding the malware, we need some tool which make this study easy to understand and reduce the time to analyse.

## What questions do you want to be able to answer with your visualization?

➔ What is the behavior of malware in system?

To understand what a malware does and what areas in the system it is going to affect, what kind of system calls it makes, what steps it keeps on repeating, etc. This in one word can be called as Behaviour of the Malware.

➔ If this malware is a member of any existing family?

Malware are nothing but some malicious code which perform some activity when enter any system. Whenever a new malware is created or any new version of existing malware is created it will be done by doing some little changes in the existing malware activities. So, it will show almost same kind of behaviour except for the new changes done to it.

Hence, the malware can be categorized based on the behaviour it shows or the maximum matching behaviour it shows to an already existing malware. These categorization is what called as Malware Family. Identifying malware family is important as it helps researchers and antivirus companies in finding solution to the malware.

**What is your data about? Where does it come from? What attributes are you going to use? What is their meaning? What are their attribute types (data abstraction)? Do you plan to generate derived attributes? If yes, which and why?**

Our raw data for Malware Visualization comes by logging the system activities before and after the malware was inserted in the system. The system log files looks like shown below:

```
.
.
.
pc: 2001563828
instr: 674079
new_pid {
  pid: 828
  name: "svchost.exe"
}
pc: 2001563826
instr: 823988
nt_create_file {
  proc {
    pid: 828
    name: "svchost.exe"
  }
                                                                         filename:
"\\??\\IDE#CdRomQEMU_QEMU_DVD-ROM_______________________1.0_____#5&3a2a
5854&0&1.0.0#{53f5630d-b6bf-11d0-94f2-00a0c91efb8b}"
}
.
.
.
```

This log file was then converted into a CSV file in order to understand it for make a Visualization. The CSV files has below mentioned attributes:

| Attribute Name | Attribute Type | Description | Type / Value | Derived? |
|---|---|---|---|---|
| instr | Ordinal | An instruction number based on the time stamp. | Long integer<br>E.g. 4850081937 | No |
| call_name | Categorical | Type of system call | String<br>E.g. nt_create_file | No |
| pid | Categorical | Unique identification number for a process | Integer<br>E.g. 828 | No |
| name | Categorical | Name of the process | String<br>E.g. cmd.exe | No |
| new_pid | Categorical | Process id for newly created process | Integer<br>E.g. 901 | No |
| new_name | Categorical | Name of the newly created process | String<br>E.g. paint.exe | No |

| | | | | |
|---|---|---|---|---|
| file_name | Categorical | Name of the file to be created, read or written | String<br>E.g. test.txt | No |
| keyname | Categorical | Name of the registry key to be read or written | String<br>E.g. PCIIDE\IDECHANNEL\4&3084357F&0&1\Device Parameters | No |
| target_pid | Categorical | The process id to which other process id will write data. | Integer<br>E.g. 901 | No |
| target_name | Categorical | Name of the above mentioned process | Integer<br>E.g. 364 | No |
| section_id | Categorical | Id for the section(a memory location shared by multiple processes) | String<br>E.g. csrss.exe | No |
| section_name | Categorical | Name for the section | String<br>E.g. \Windows\SharedSection | No |
| port_id | Categorical | Unique id created to be used as port | Integer<br>E.g. 2212816928 | No |
| port_name | Categorical | Name of the port | String<br>E.g. \ThemeApiPort | No |
| client_pid | Categorical | Client process id which initiates Inter Process Communication(IPS) | Integer<br>E.g. 901 | No |
| client_name | Categorical | Name of the client process | String<br>E.g. cmd.exe | No |
| server_pid | Categorical | Server process id which acts as a server for IPC | Integer<br>E.g. 1901 | No |
| server_name | Categorical | Name of the server process | String<br>E.g. conhost.exe | No |
| call_category | Categorical | Name of API Call<br>derived from call_name | String<br>E.g. Process, File, etc. | Yes |

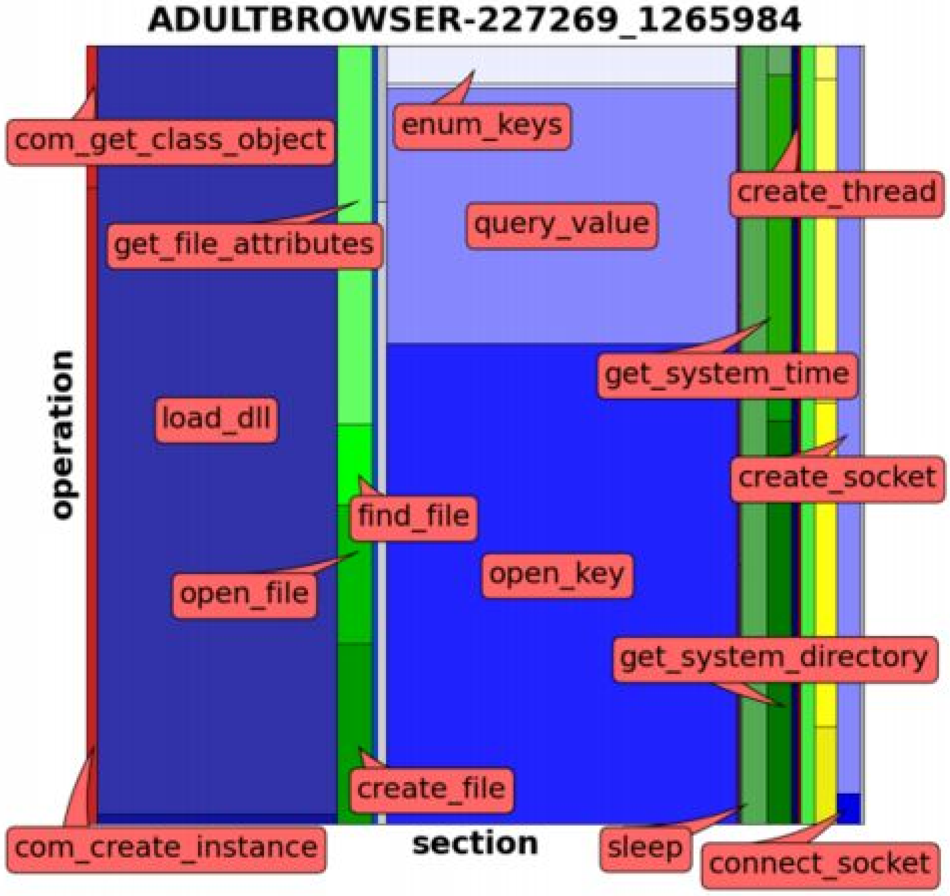
### What have others done to solve this or related problems?

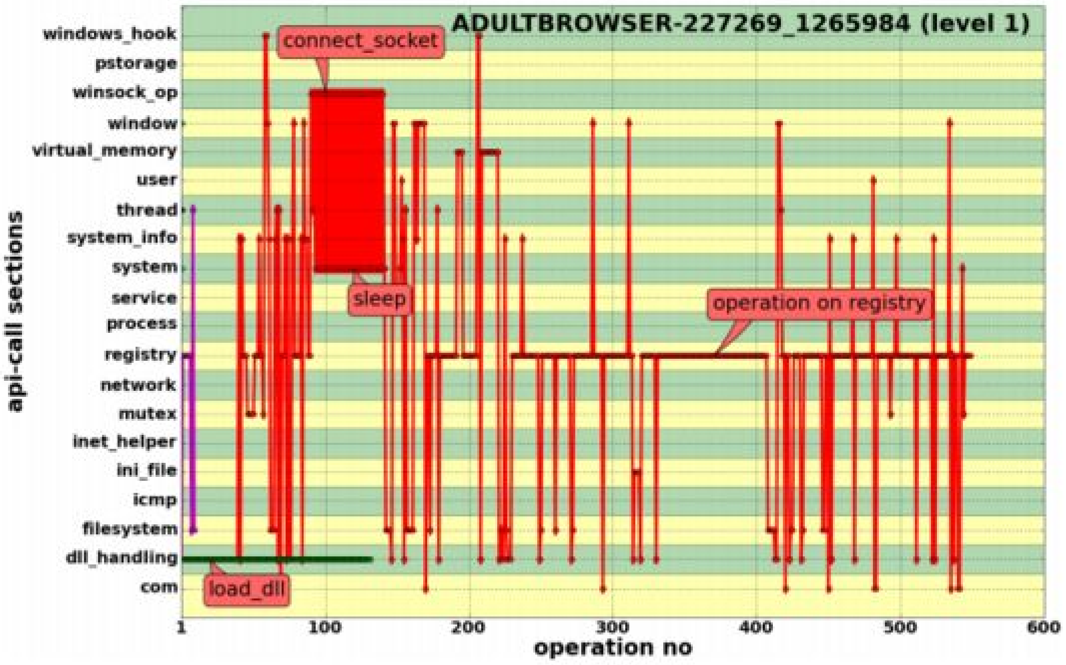
Some researchers have done static analysis in which they used decompiler to gain insight of malware infected code and predict the behaviour of the code. While many are doing dynamic analysis to see the behaviour of malware by running the code  and plot the same using different visualization technique. The most popular techniques are Behavioural Image, API Calls Density cluster, Malware Tree map and Malware tree thread graph.

We took reference from the following research papers:

**Visual Analysis of Malware Behavior Using Treemaps and Thread Graphs**
Philipp Trinius, Thorsten Holz, Jan Gobel and Felix C. Freiling

- This paper aims to help human analysts to quickly assess and classify the nature of a new malware sample.
- Here dynamic analysis is done on a malware sample by executing it in a sandbox environment to analyze the behaviour of malware.
- A sandbox executes a malware sample in a controlled environment and records all system-level behavior such as modifications of the filesystem or the registry. As a result, the sandbox generates an analysis report summarizing the observed behavior of the sample.
- The paper introduces two visualization techniques: treemap and thread graphs. Treemap displays the distribution of the individual operations performed by a sample. The resulting treemap presents this information as a set of nested rectangles and provides a quick overview of the main overall behavior of the sample, e.g., whether the main task of the sample lies in the area of network interaction, changes to the file system, or interaction with other processes. Since tree maps display nothing about the sequence of operations, we use Thread Graphs to visualize the temporal behavior of the individual threads of a sample. A thread graph can be regarded as a behavioral fingerprint of the sample in which operations are recorded with time as log entries and plotted on graph. An analyst can then study this behavior graph to quickly learn more about the actions of each individual thread.


ADULTBROWSER-227269_1265984
operation
section
com_get_class_object
enum_keys
create_thread
get_file_attributes
query_value
get_system_time
load_dll
create_socket
find_file
open_file
open_key
get_system_directory
create_file
com_create_instance
sleep
connect_socket


ADULTBROWSER-227269_1265984 (level 1)
api-call sections
windows_hook
pstorage
winsock_op
window
virtual_memory
user
thread
system_info
system
service
process
registry
network
mutex
inet_helper
ini_file
icmp
filesystem
dll_handling
com
connect_socket
sleep
operation on registry
load_dll
1
100
200
300
400
500
600
operation no

**Visualization Techniques for Malware Behavior Analysis**

André R. A. Grégio and Rafael D. C. Santosc

- In this paper, authors did analysis based on the behavior of malwares to understand them. They used dynamic analysis in order to record its behavior.
- To visualize the data obtained by analysing various malware, clustering technique was used where the vertices are the individual malware and the edges are based on the associated weight which is calculated based on the degree of agreement among antiviruses.
- They also used Thread Graph technique to represent the activities of malware and to track what new process do they run and create.
- Based on the information obtained from the graphs, analyst can understand what actually happened in the compromised system and can compare these studies with those of new malware to see whether they behave in a similar fashion or not.

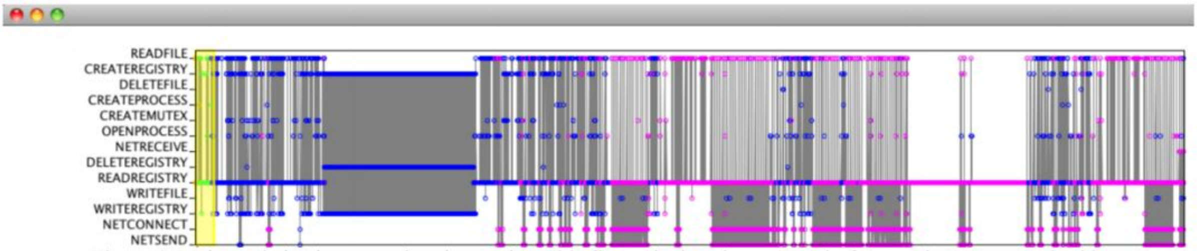
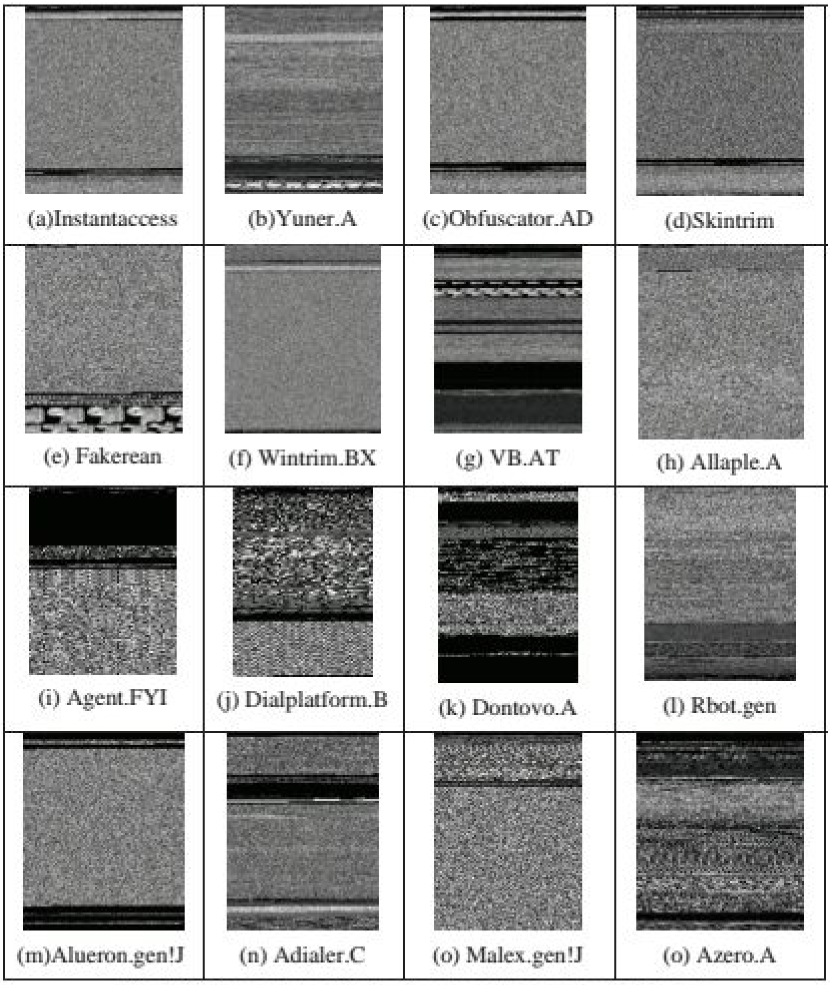


Figure 4. Malware behavior execution time series. This figure depicts the top panel of our tool, were a user is able to walk through it using the yellow magnifier.

**Malware Images: Visualization and Automatic Classification**

L. Nataraj, S. Karthikeyan, G. Jacob and B. S. Manjunath

- This paper uses the static analysis method i.e., without running the malware in virtual machine but studying malware binaries.
- They used image processing technique to visualize the malware binaries as gray-scale images.
- The gray-scale images produced for different malware remains almost similar if malwares are from same family.
- They tested about 9000 samples with 25 different malware families and had classification accuracy of 98%.



(a)Instantaccess (b)Yuner.A (c)Obfuscator.AD (d)Skintrim

(e) Fakerean (f) Wintrim.BX (g) VB.AT (h) Allaple.A

(i) Agent.FYI (j) Dialplatform.B (k) Dontovo.A (l) Rbot.gen

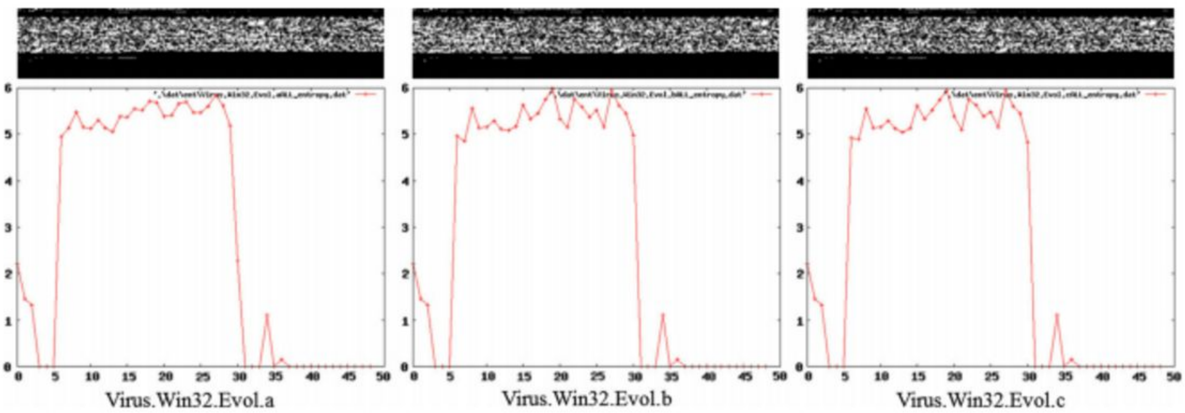
(m)Alueron.gen!J (n) Adialer.C (o) Malex.gen!J (o) Azero.A

Fig. 7 Malware Images belonging to various malware families

**Malware analysis using visualized images and entropy graphs**

Kyoung Soo Han, Jae Hyun Lim, Boojoong Kang and Eul Gyu Im

- Malware are created using some automated tools and methods may reuse some modules to develop malware variants which can be used to classify malware and identify malware families. This paper classifies malwares based on converting binary files into images and entropy graphs. They allow malware researchers to understand the structures of malware binary files without disassembling and to make a decision for applying of unpacking.
- They uses Bitmap Image Converter which receives Windows Portable Executable binary files as input and converts binary files into bitmap images. The Entropy Graph Generator calculates the entropy value of each line of bitmap image and generates entropy graph based on these values. These entropy values and graphs are stored in database and used to classify and detect malware based on similarities.
- They had a limitation when no particular pattern can be identified in gray-scale images and entropy value of binaries can be very high.

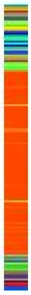
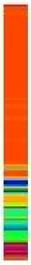
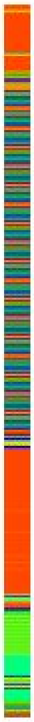
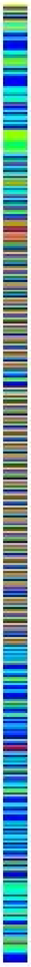
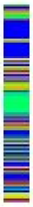
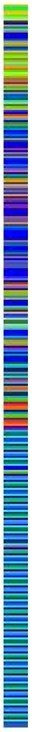


Virus.Win32.Evol.a Virus.Win32.Evol.b Virus.Win32.Evol.c

**Malware Behavior Image for Malware Variant Identification**

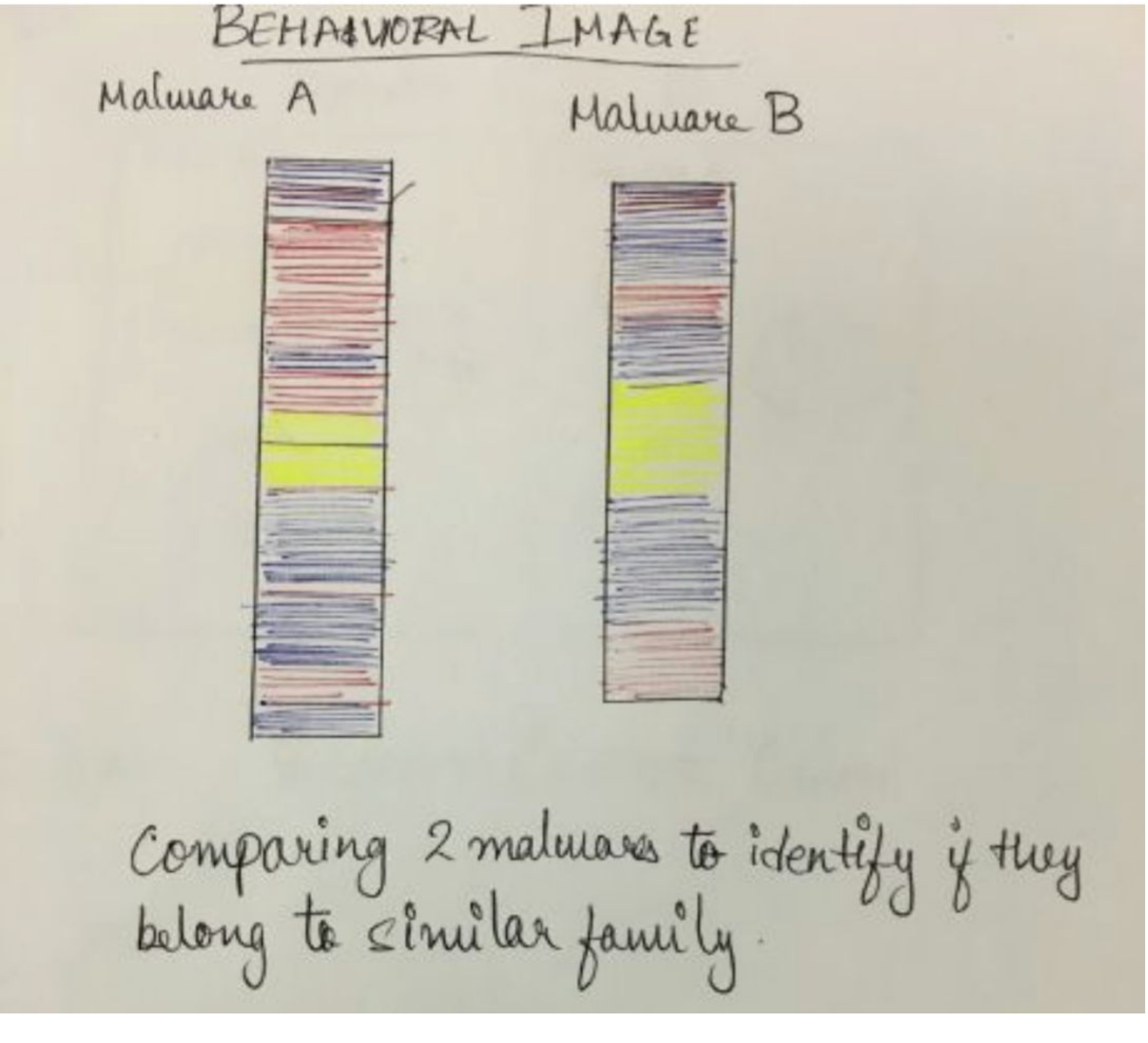
Syed Zainudeen Mohd Shaid, Mohd Aizaini Maarof

- This paper talks about mapping malware behaviour using color in behaviour image.
- A color map is a map which ranges from red(hot) to blue(cold). Hot colors are used to represent malicious API calls while cold colors are used to represent non malicious API calls. Calls are grouped together as malicious, less malicious , non malicious, etc. Each API call is represented by 64*4 pixel colored rectangle and the color is determined by the type of call.
- Finally we get a Behaviour Image for the malware. When we plot these images for malwares of the same family, we can see that they are nearly similar.

| Malicious sample | | | Benign sample | | |
|---|---|---|---|---|---|
|  |  |  |  |  |  |
| SdBot 4 | Allaple B | Korgo B | Paint | Explorer | Internet Explorer |

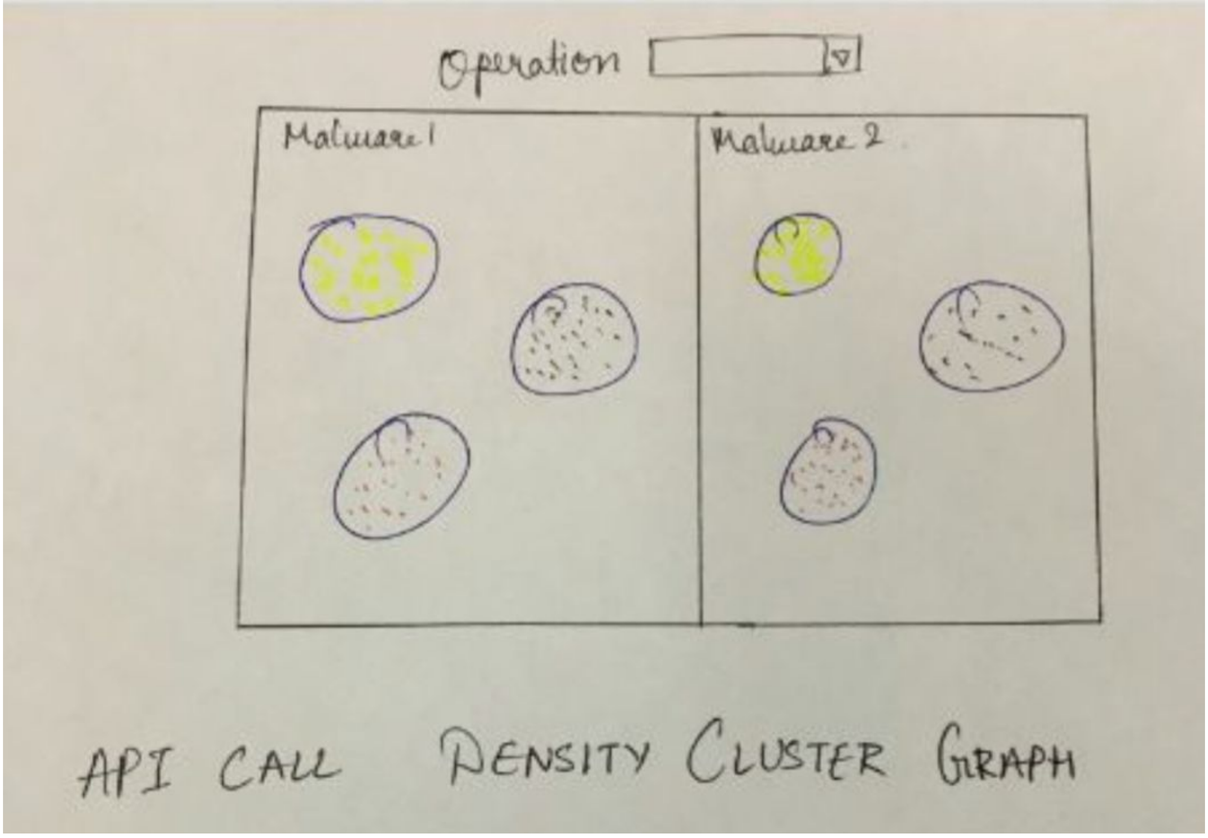
**Initial Mockup**

Behavioural Image: This image shows the what kind of system API calls were made, we can narrow it down for malware to see its sequence. This image needs to be read from top to down i.e., the number of instructions increases from top to down and each of the lines(rectangles) are System API Calls.
For all 6 kind of API calls, Process, File, Registry, Memory Section, Virtual Memory and IPc there is separate color representing.

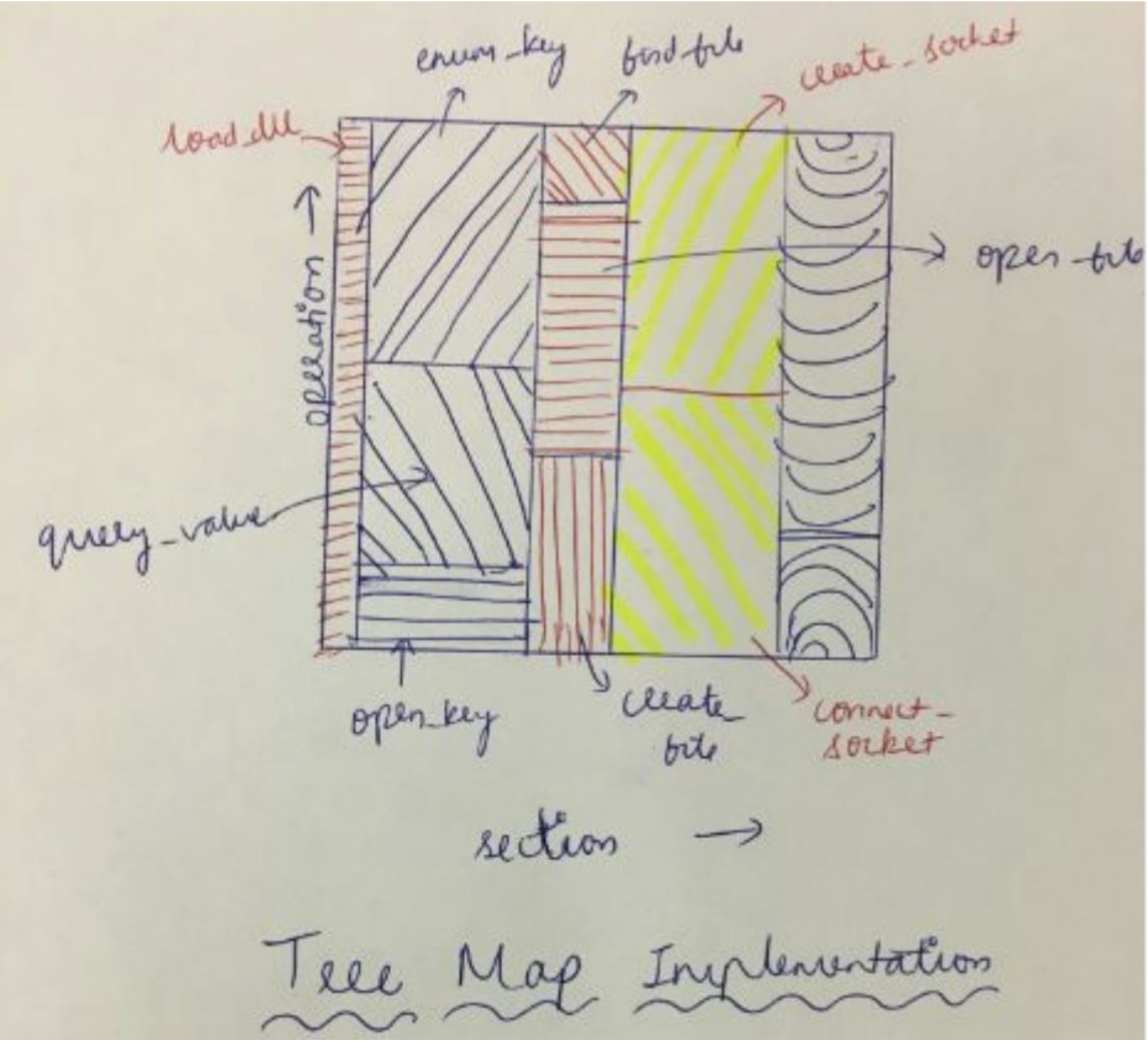


Comparing 2 malwares to identify if they belong to similar family.

Density Cluster: API Call Density cluster used to represent the overall behaviour of the malware and we can compare two malware in it. The malwares can be compared at different levels of operations or can be compared using all the operations. If the clusters formed are exactly similar implies that the two malwares belong to the same family even with with little variations,  a new version of the existing malware is found.



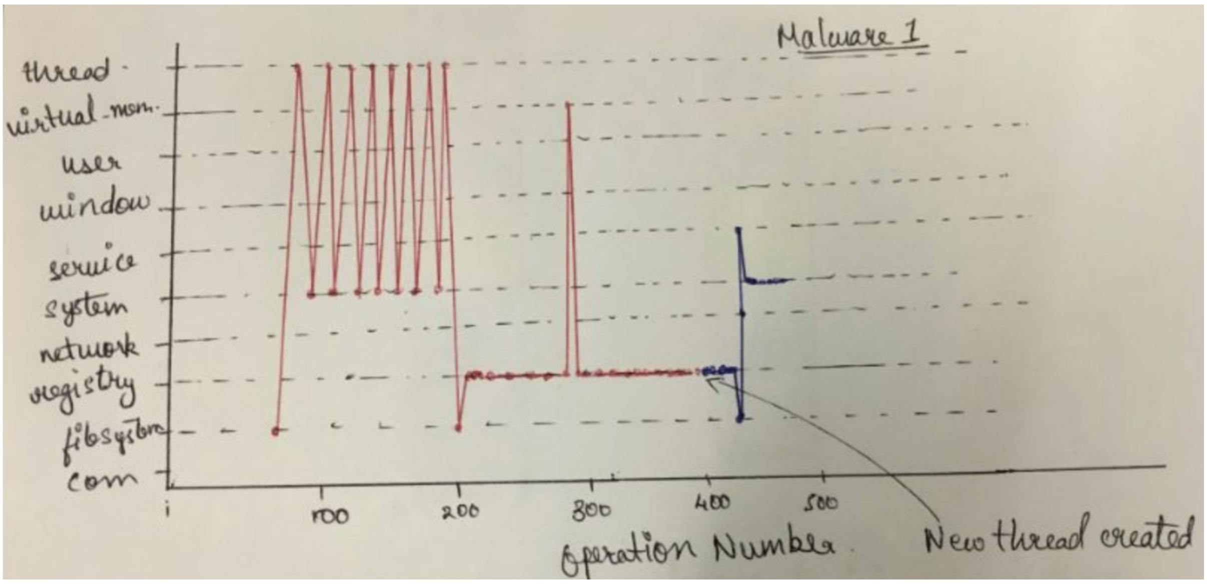
Tree Map: In this visualization, a map is divided into horizontal sections. Each section represent a specific group of API call such as file, socket or key operation calls. Inside each section there are vertical sections created which represents specific call for that group like file_create, file_delete or file_read. If dynamic analysis of two malware produces same kind of tree map then they belong to same family or one malware can be the new version of other malware.



Tree Map Implementation

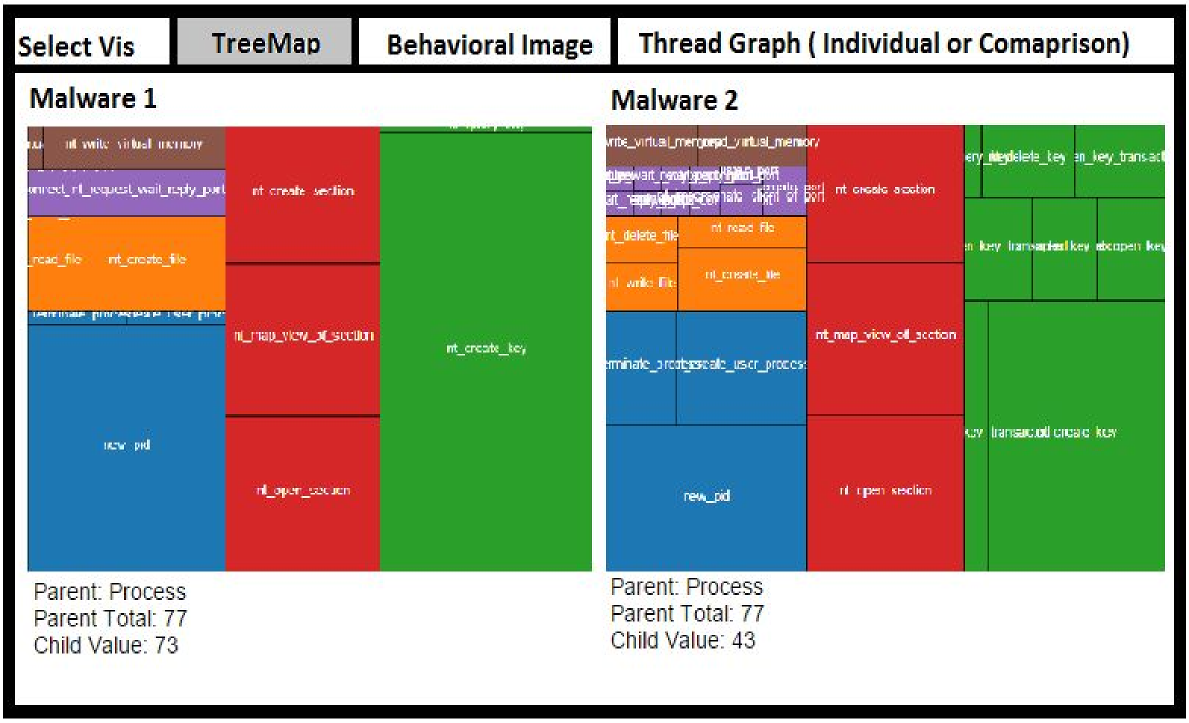
Thread Graph: The Threaded graph represents all the activities done by the malware during its execution in the virtual system. The graph also displays the new processes created by the one we are monitoring.

From left to right, instruction number increases and y-axis show which system API call was made during the instruction.



**Project Update**

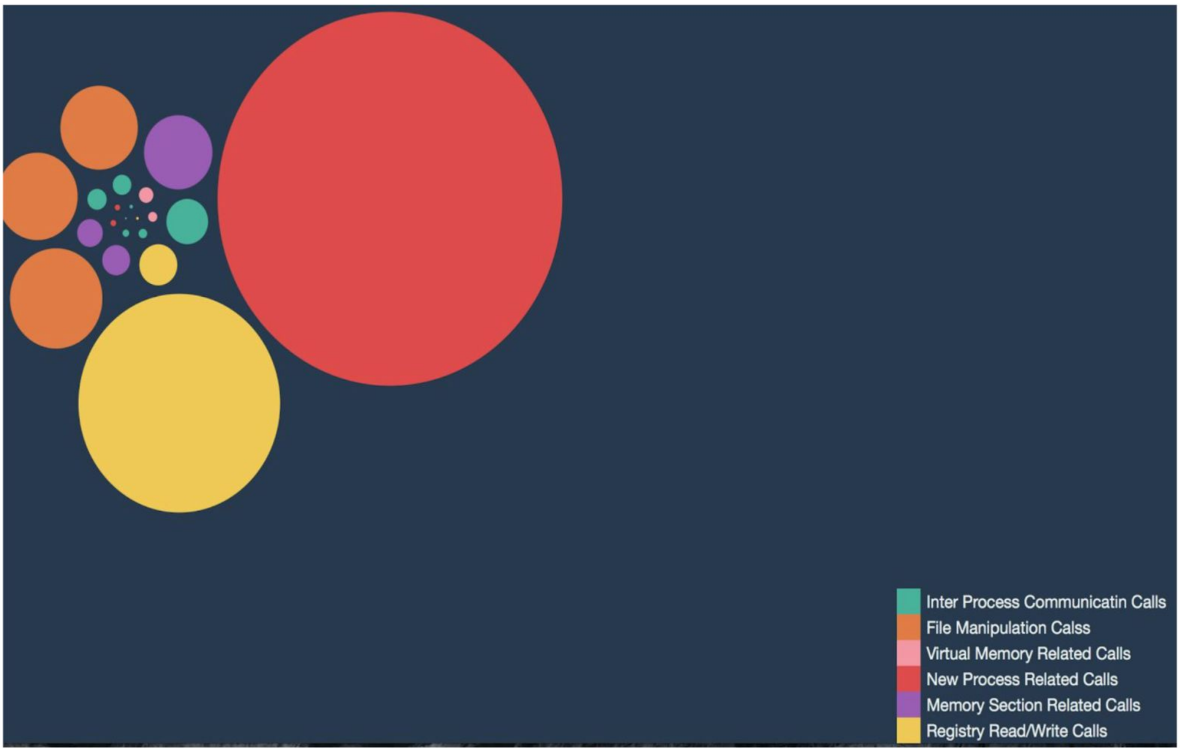
Our first visualization created are shown below,
Treemap showing 2 malware side by side, helps in examining the ratio of the overall calls made in each malware.



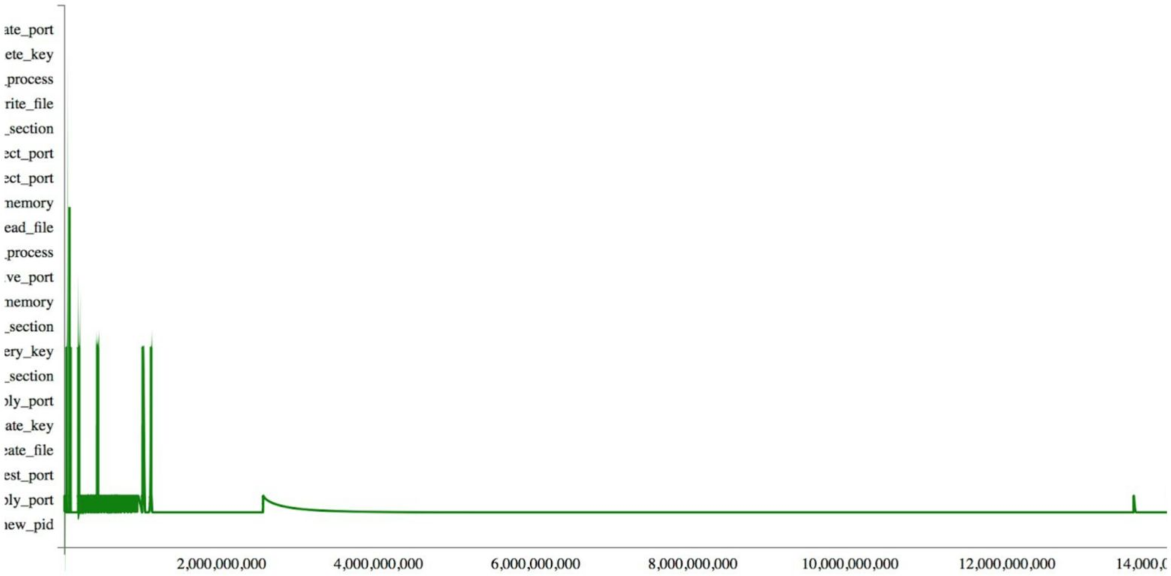
Parent: Process
Parent Total: 77
Child Value: 73

Parent: Process
Parent Total: 77
Child Value: 43

Next is the density cluster which shows which kind of calls were made and the size of each cluster represents the number of times the call was made.



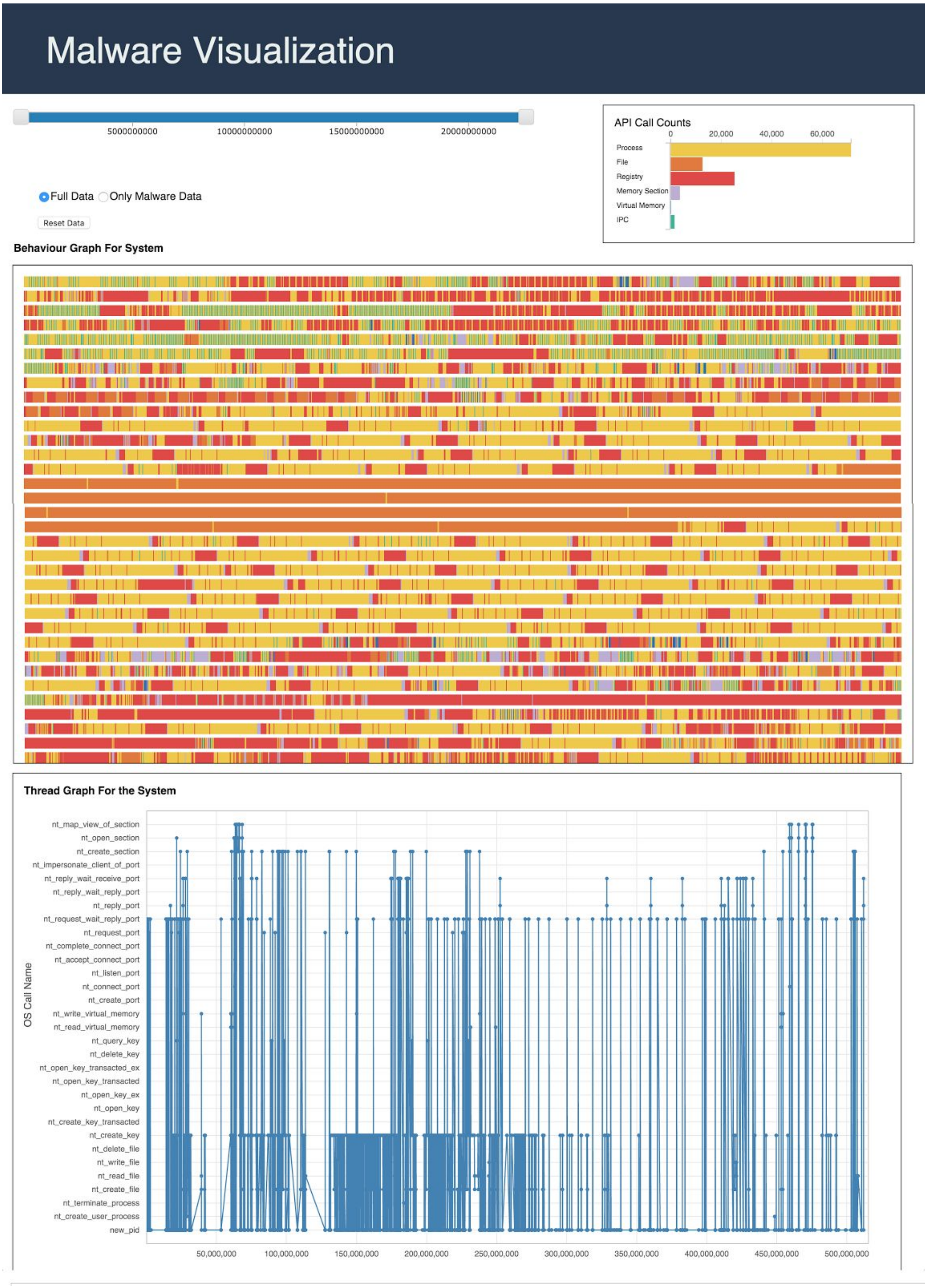
Last is the Thread Graph showing the calls made at what particular instruction. From left to right it shows the growing instruction number and on y-axis System API calls.



**Final Visualization**

Our final visualization consists of Bar Graph, Behavior Image and a Thread Graph. We changed our design from Treemap and Density cluster because when we started to connect all the visualization, those two graphs were not helping much in answering our problem questions. Now we have connected our designs to interact with each other which helps in analysing and identifying the behavior of malware. The thread graph is showing patterns of activities in it which the malware is performing.

The color of the Bar graph and the Behavior graph are same and help in relating to each other.



● Full Data ○ Only Malware Data

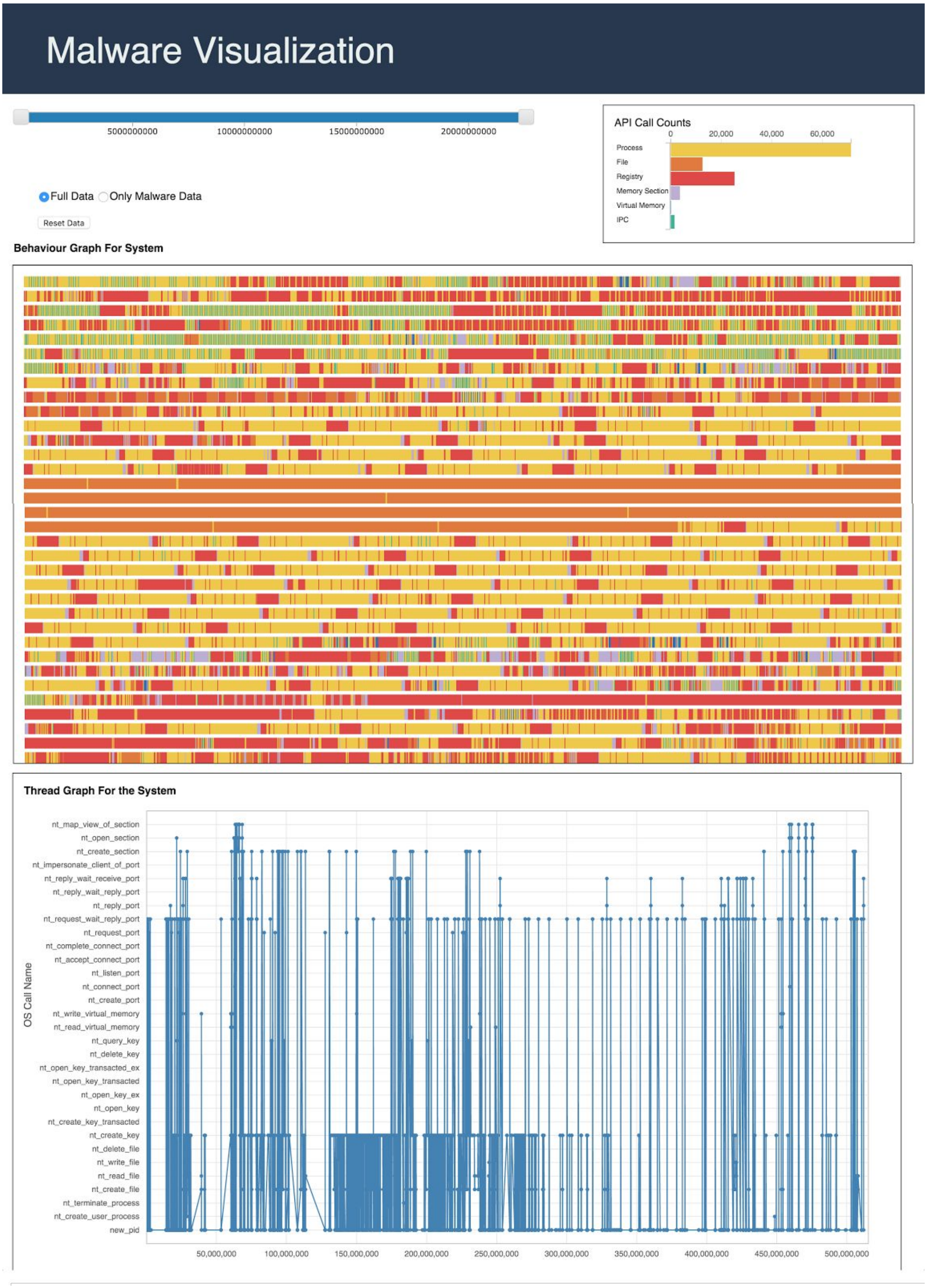
To read the graph, it starts initially at Full data mode during which it shows all the activities going on in the system. On switching to Only malware data, all visualizations shows data related to malware as shown below. The Bar graph show the number of count i.e., number of times a specific kind of system API call was made. The behavior graph grows from left to right and top to bottom and the thread graph shows the data as per the instructions from left to right on x-axis and API calls on y-axis. The thread graph shows system call at very granular level.

All these graphs are controlled by a single slider at the top of the visualization, it basically controls the amount of data to be shown in a visualization and from instruction A to instruction B.

Also, the bar graph controls the amount of data that to be seen in the Behavior graph and Thread graph. Right click on the bar removes the related API calls and Left click brings it back in both the graphs.

**Data Analysis**

Initially when our Visualization application starts it shows for the over all data as shown below. Currently it is representing all the 3 graphs for the over all system activity. This shows the complete behavior of the system when running. It includes both malware as well as the normal system API calls and some other applications those are running in the system.

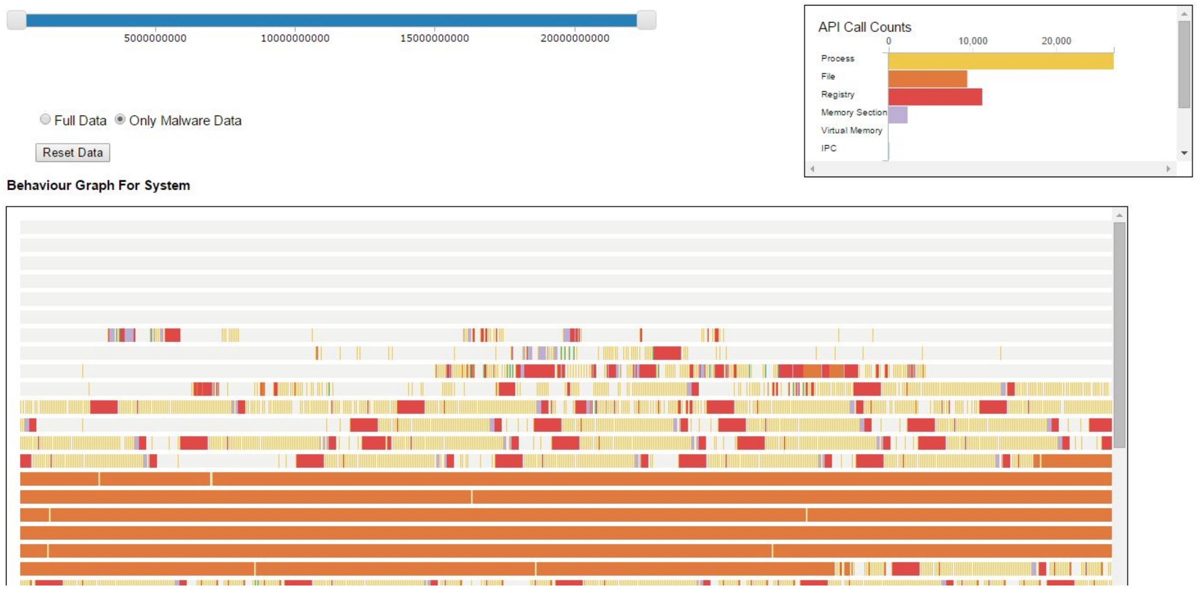


Malware Visualization

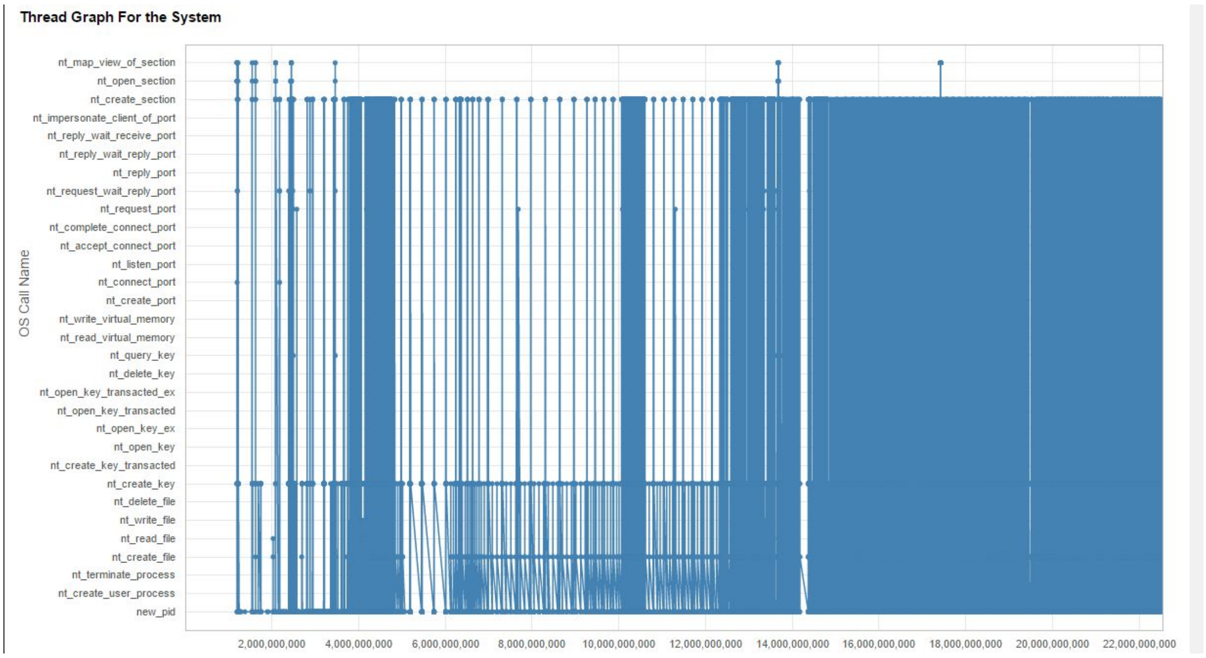
● Full Data ○ Only Malware Data

Reset Data

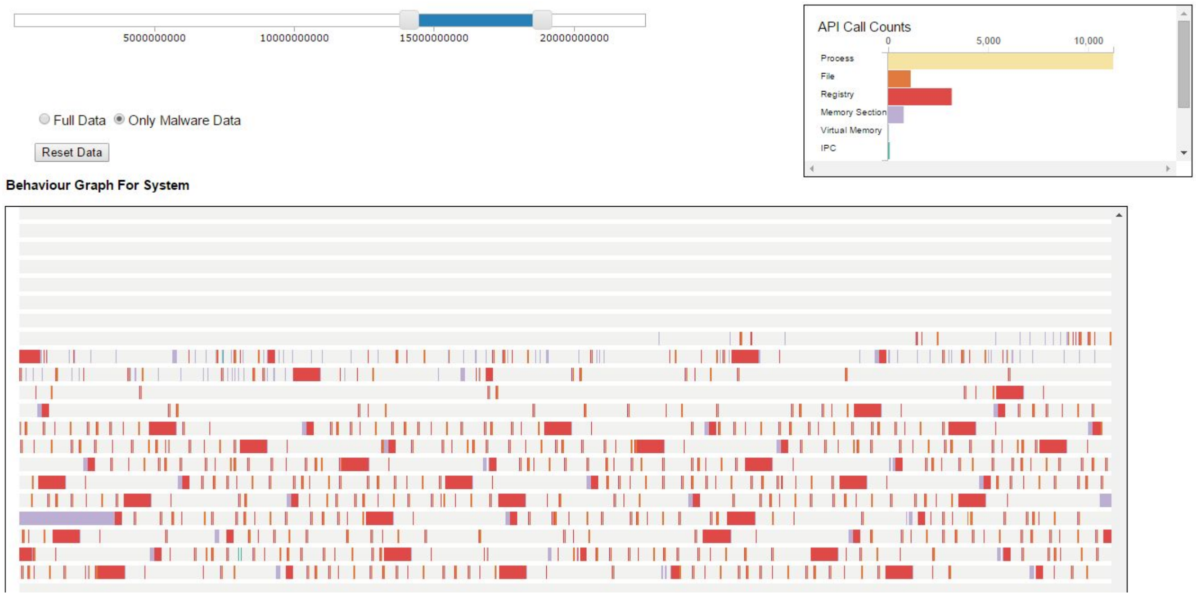
But when we switch to Only Malware Data mode, we can see in Behavior map from when the malware started in our system and in Behavior map it shows the activities of the malware only which can be seen below.



○ Full Data ● Only Malware Data

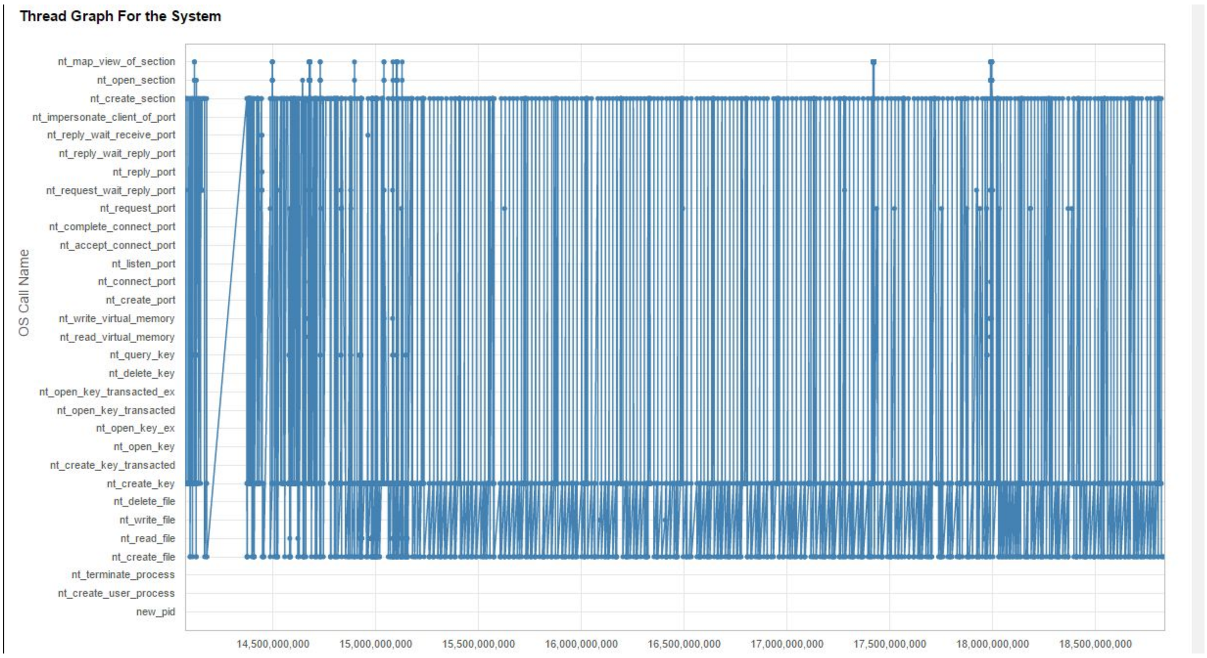


The Bar graphs helps in getting the count of System API calls as well as to limit the Behavior and Thread graph to some specific API calls.

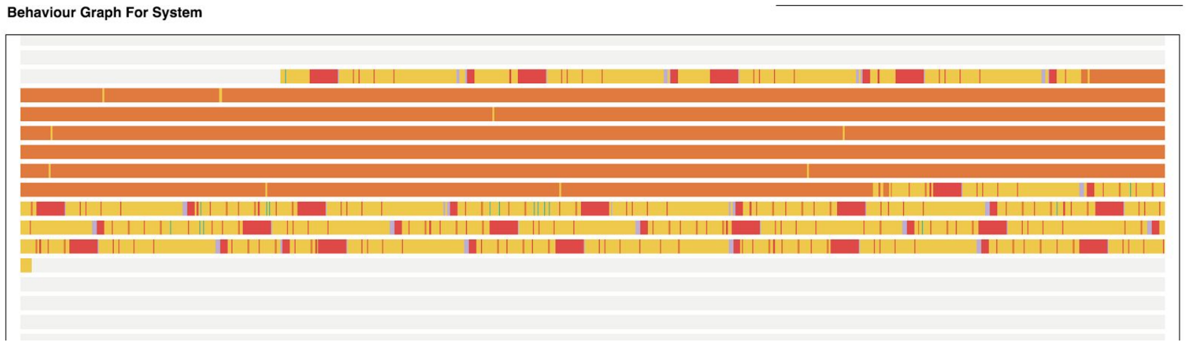


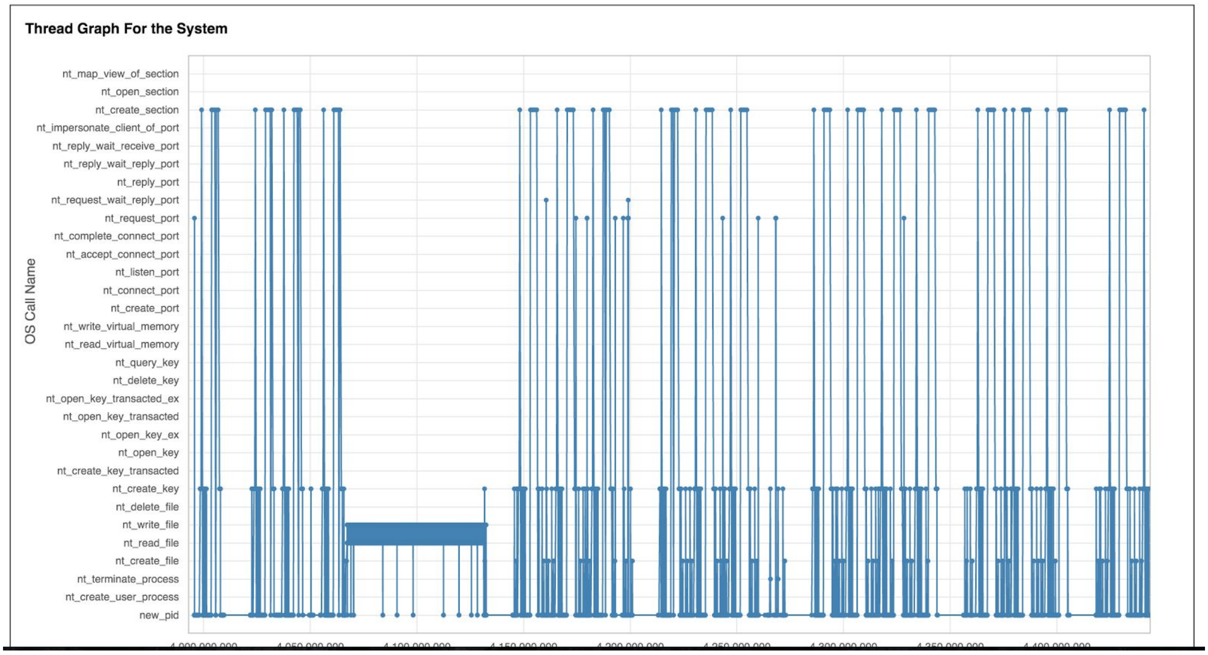
○ Full Data ● Only Malware Data

In the above Image we can see that we have currently removed the Process type API calls and now we can see a pattern for this malware i.e., Repeated Registry, Section and File operations which can be seen in more details in Thread Graph shown below.



In another example in which we can see that our malware is make File related system calls for a large number of instructions. This shows some kind of behavior which can be helpful in confirming the family of the malware.





For comparing any two malware, we need to open this same application on two screens or on two different tabs.

## Limitations and Future Works

Currently, our application is static and does not load any CSV that we want. For this we need to to update some code which can enable such activity.

Furthermore, the application is loading data from CSV file which is very large and hence takes time to load data every time any interaction is done with it.

To identify the correct time slice, we need to do some trial and error and estimate the approximate value to find the perfect location.

Finally, our main question was to compare a new malware to find its family which can be done using our current application but would require two applications to be stated in two different application.

If we had some more time, we could have overcome the issue of time slice. We had a plan to provide a tooltip over the behavior map which provides the instruction number and hence will help in moving the slider to accurate position instead of trial and error.

What are the major limitations of your project at this stage (it’s fine to talk openly about what does not work well and what you have not been able to realize)? If you had more time to develop your application further, what would be the next steps?