

MAEC Malware Visualization

**Pauline Finlay**

**B00107945**

***Department of Informatics***

***School of Informatics & Engineering***

***Technological University Dublin, Blanchardstown***

***Dublin 15.***

**Page Count: 105 Word Count: 13,600**

**BSc (Hons) in Computing in Digital Forensics & Cyber Security**

**DFCS H4018 Individual Project**

**14th May 2021**

**Plagiarism Declaration**

I hereby certify that this material, which I am submitting for assessment for the course title of BSc. (Hons) In Computing in Digital Forensics and Cyber Security is my own work and that I have:

* Clearly referenced/listed all sources as appropriate
* Given the sources of all pictures, data etc. that are not my own
* Any work of any other student(s) either past or present is clearly referenced
* Not submitted for assessment work previously submitted for any other course, degree, or qualification
* Not incorporated any text acquired from external agencies other than extracts from attributed sources (including online facilities)
* Acknowledged in appropriate places any help that I have received from others (e.g., fellow students, technicians, statisticians, external sources)
* My work may be electronically checked for plagiarism, including, but not exclusively, by the use of plagiarism detection software and stored for future comparison
* Any false claim for this work will be penalised in accordance with the College’s regulations

**Signature :** Pauline Finlay **Date :** 14/05/2021

**Abstract**

The Malware Attribute Enumeration and Characterisation (MAEC) framework is a method used to characterise malware and while it was envisioned that it would be utilised to perform malware visualization no formal tools exist. By extracting actions from the standardised MAEC malware analysis reports this paper investigates the viability of generating effective visualizations. In achieving this aim a dataset of 722 malware samples from 8 malware families was acquired, for which MAEC-format dynamic analysis reports have been generated by Cuckoo sandbox. Action features were then extracted from these reports and visualized using area graphs, line graphs, bar graphs, heatmaps, and 3D surface graphs using the python graphing libraries matplotlib, seaborn and plotly. To evaluate the performance of these visualizations in capturing and summarising malware actions and displaying similarities and correlation amongst samples they were critiqued against formalised criteria for evaluating visualizations. The results illustrate the ability of bar graphs to effectively analyse individual samples, while heatmaps successfully identified correlation and similarities amongst samples but did not identify action categories or individual actions. The area graphs and line plots performed well in smaller sample sets with a similar range of values but struggled with vast ranges and larger sample sets. The 3D visualizations effectively demonstrated the difference in action categories amongst samples and families utilising indexing for categories and samples to avoid cluttering visualizations.

**Acknowledgements**

I would like to thank all my lecturers throughout my time firstly as part of ITB and now TU Dublin. I would like to mention Stephen O’Shaughnessy for instilling the interest in malware analysis and his presentation on project skills.

Special thanks however must go to my project supervisor, Aurelia Power, for her support, encouragement, and guidance throughout the course of this project. Her advice and feedback were greatly appreciated and welcomed.

Lastly, I cannot forget my wonderful family, thank you for your continued support throughout the past 4 years and for making all this possible.

Table of Contents

[1 Introduction 1](#_Toc71899824)

[1.1 Background and Motivation 1](#_Toc71899825)

[1.2 Research Aims 3](#_Toc71899826)

[1.3 Research Objectives 3](#_Toc71899827)

[1.4 Research Methodology Overview 3](#_Toc71899828)

[1.5 Thesis Overview 4](#_Toc71899829)

[1.6 Summary 4](#_Toc71899830)

[2 Literature Review 5](#_Toc71899831)

[2.1 Malware Types & Behaviours 5](#_Toc71899832)

[2.1.1 Adware 5](#_Toc71899833)

[2.1.2 Backdoor Trojan 5](#_Toc71899834)

[2.1.3 Bot 6](#_Toc71899835)

[2.1.4 Fileless Malware 6](#_Toc71899836)

[2.1.5 Keyloggers 6](#_Toc71899837)

[2.1.6 Mobile Malware 6](#_Toc71899838)

[2.1.7 Ransomware 7](#_Toc71899839)

[2.1.8 Rootkits 7](#_Toc71899840)

[2.1.9 Spyware 7](#_Toc71899841)

[2.1.10 Viruses 7](#_Toc71899842)

[2.1.11 Worms 8](#_Toc71899843)

[2.2 Common Malware Distribution Methods 9](#_Toc71899844)

[2.2.1 Cut and Paste Exploit 9](#_Toc71899845)

[2.2.2 False ‘Download Now’ Links 9](#_Toc71899846)

[2.2.3 Email Spear Phishing 9](#_Toc71899847)

[2.2.4 Messages and Posts from Trusted Sources 9](#_Toc71899848)

[2.2.5 USB Thumb Drives 9](#_Toc71899849)

[2.3 Safeguarding your system from Malware 10](#_Toc71899850)

[2.4 Malware Attribute Enumeration and Characterization (MAEC) 11](#_Toc71899851)

[2.4.1 Language Overview 11](#_Toc71899852)

[2.4.2 MAEC Data Model 15](#_Toc71899853)

[2.4.3 Advantages of MAEC 16](#_Toc71899854)

[2.4.4 Associated Language 17](#_Toc71899855)

[2.5 Visualization 18](#_Toc71899856)

[2.5.1 Graph Based Visualization 18](#_Toc71899857)

[2.5.2 Bar Charts and Histograms 20](#_Toc71899858)

[2.5.3 Heatmaps 22](#_Toc71899859)

[2.5.4 Thread Graphs 23](#_Toc71899860)

[2.5.5 3D Malware Visualization 24](#_Toc71899861)

[2.5.6 Summary 25](#_Toc71899862)

[3 Methodology 26](#_Toc71899863)

[3.1 Overview 26](#_Toc71899864)

[3.2 Proposed Method 26](#_Toc71899865)

[3.2.1 Software Used 26](#_Toc71899866)

[3.3 Malware Sample Collection 27](#_Toc71899867)

[3.4 Cuckoo Sandbox and MAEC Classification 28](#_Toc71899868)

[3.5 Feature Extraction 29](#_Toc71899869)

[3.6 Data Pre-processing 30](#_Toc71899870)

[3.7 Visualization 30](#_Toc71899871)

[3.7.1 Data Visualised 31](#_Toc71899872)

[3.7.2 Visualization Libraries 31](#_Toc71899873)

[3.7.3 Visualization Evaluation 31](#_Toc71899874)

[3.7.4 Types of Visualisation 33](#_Toc71899875)

[3.8 Summary 33](#_Toc71899876)

[4 Implementation and Experimentation 34](#_Toc71899877)

[4.1 Overview 34](#_Toc71899878)

[4.2 Acquire Dataset 34](#_Toc71899879)

[4.3 Feature Extraction 36](#_Toc71899880)

[4.4 Data Pre-processing 39](#_Toc71899881)

[4.4.1 Create DataFrame 39](#_Toc71899882)

[4.4.2 Transpose DataFrame 41](#_Toc71899883)

[4.5 Generating Visualizations 42](#_Toc71899884)

[4.6 Summary 47](#_Toc71899885)

[5 Results, Analysis, and Evaluation 48](#_Toc71899886)

[5.1 Files with large number of calls 48](#_Toc71899887)

[5.1.1 Ghostrat sample 48](#_Toc71899888)

[5.1.2 Swizzor sample 51](#_Toc71899889)

[5.1.3 Hamweq Samples 53](#_Toc71899890)

[5.1.4 Zeus samples 56](#_Toc71899891)

[5.1.5 CryptoLocker samples 60](#_Toc71899892)

[5.1.6 Spiked samples 69](#_Toc71899893)

[5.2 Malware Families Analysed 72](#_Toc71899894)

[5.2.1 Select Malware Samples 72](#_Toc71899895)

[5.2.2 Analysis of Samples 72](#_Toc71899896)

[5.3 Full Dataset 90](#_Toc71899897)

[5.4 3D Visualization 91](#_Toc71899898)

[5.5 Evaluation 101](#_Toc71899899)

[5.5.1 Evaluation of Area Graphs 101](#_Toc71899900)

[5.5.2 Evaluation of Bar Charts 101](#_Toc71899901)

[5.5.3 Evaluation of Line Graphs 102](#_Toc71899902)

[5.5.4 Evaluation of Heatmaps 102](#_Toc71899903)

[5.5.5 Evaluation of 3D Visualization 102](#_Toc71899904)

[6 Conclusion 104](#_Toc71899905)

[6.1 Overview 104](#_Toc71899906)

[6.2 Conclusions 104](#_Toc71899907)

[6.3 Further Work 105](#_Toc71899908)

[7 References 106](#_Toc71899909)

[8 Appendix 117](#_Toc71899910)

[8.1 Malware Dataset Structure 117](#_Toc71899911)

[8.2 Action Category Dataset 118](#_Toc71899912)

[8.3 Python Programs 120](#_Toc71899913)

[8.3.1 dataset\_download.py 120](#_Toc71899914)

[8.3.2 Feature Extraction 121](#_Toc71899915)

[8.3.3 Create DataFrame 122](#_Toc71899916)

[8.3.4 Export DataFrame to Excel 123](#_Toc71899917)

[8.3.5 Create DataFrame from .csv file 123](#_Toc71899918)

[8.3.6 Create Interactive Line Plot with Plotly 123](#_Toc71899919)

[8.3.7 Create Summated Bar Graphs with Plotly 123](#_Toc71899920)

[8.3.8 Create Area Graph 123](#_Toc71899921)

[8.3.9 Create Heatmap 123](#_Toc71899922)

[8.3.10 Create 3D Surface Graph 124](#_Toc71899923)

**Table of Tables**

[Table 1: GitHub MAEC Dataset Table 27](#_Toc71899924)

[Table 2: MAEC Namespace Categories 28](#_Toc71899925)

[Table 3: MAEC Unique Actions Table 29](#_Toc71899926)

[Table 4: Visualization Type Table 33](#_Toc71899927)

[Table 5: Ghostrat Action Calls 49](#_Toc71899928)

[Table 6: Swizzor Action Calls 51](#_Toc71899929)

[Table 7: Hamweq Action Calls 53](#_Toc71899930)

[Table 8: Zeus Action Calls 56](#_Toc71899931)

[Table 9: CryptoLocker Action Calls 60](#_Toc71899932)

[Table 10: CryptoLocker Samples Category Index 93](#_Toc71899933)

[Table 11: Ghostrat Samples Category Index 94](#_Toc71899934)

[Table 12: Hamweq Samples Category Index 95](#_Toc71899935)

[Table 13: Swizzor Samples Category Index 96](#_Toc71899936)

[Table 14: Vanti Samples Category Index 97](#_Toc71899937)

[Table 15: Anubis Samples Category Index 98](#_Toc71899938)

[Table 16: Zeus Samples Category Index 99](#_Toc71899939)

[Table 17: Reveton Samples Category Index 100](#_Toc71899940)

[Table 18: Action Category Dataset Totals 118](#_Toc71899941)

**Table of Figures**

[Figure 1: Top Malware Detection by Consumers 2020 (Malwarebytes, 2021) 2](#_Toc71899942)

[Figure 2: Top Malware Detection by Businesses 2020 (Malwarebytes, 2021) 2](#_Toc71899943)

[Figure 3: MAEC Core Components (MITRE Corporation, n.d.) 11](#_Toc71899944)

[Figure 4: MAEC - MSM Standards Relationship (Kirillov, et al., 2011) 12](#_Toc71899945)

[Figure 5: MAEC Bundle Overview (Beck, et al., 2014) 13](#_Toc71899946)

[Figure 6: MAEC Data Model (MITRE, 2020) 15](#_Toc71899947)

[Figure 7: Visualised SDO Relationship between Malware and Malware Analysis (OASIS Open, 2021) 17](#_Toc71899948)

[Figure 8: Attack graph construction from multiple sources (The Honeynet Project, 2012) 19](#_Toc71899949)

[Figure 9: Linked Malware Attack Graph (The Honeynet Project, 2012) 19](#_Toc71899950)

[Figure 10: Similarities based on AV labelling (Grégio & Santos, 2011) 20](#_Toc71899951)

[Figure 11: Bar Chart detailing classifier performance (Nari & Ghorbani, 2013) 21](#_Toc71899952)

[Figure 12:Histogram of File Size Distribution (Shalaginov, et al., 2018) 21](#_Toc71899953)

[Figure 13: Heatmap visualizing benign and malware sample similarities (Anderson, et al., 2012) 22](#_Toc71899954)

[Figure 14: Malware analysed using a thread graph (Trinius, et al., 2009) 23](#_Toc71899955)

[Figure 15: 'ScamFraud4198' (Dragulescu, 2008) 24](#_Toc71899956)

[Figure 16: Skyrails Dragulescu Visualization (Swanson, 2008) 24](#_Toc71899957)

[Figure 17: Visual Representation Criteria (Freitas, et al., 2002) 32](#_Toc71899958)

[Figure 18: MAECProject / datasets (GitHub, Inc., 2014) 34](#_Toc71899959)

[Figure 19: ptrac3 /MAEC-Dataset (GitHub Inc., 2015) 35](#_Toc71899960)

[Figure 20: Sample web .xml file (GitHub, Inc., 2014) 35](#_Toc71899961)

[Figure 21: Python program to download web .xml MAEC reports 36](#_Toc71899962)

[Figure 22: Files read line-by-line extracting MAEC ActionNameVocab-1.0 37](#_Toc71899963)

[Figure 23: ‘ActionNameVocab-1.0’ line format (GitHub Inc., 2015) 37](#_Toc71899964)

[Figure 24: REGEX to extract malware action 37](#_Toc71899965)

[Figure 25: Action’s dictionary of the individual actions and their count 38](#_Toc71899966)

[Figure 26:File sample action dictionaries added to single list 38](#_Toc71899967)

[Figure 27: Extract from printed ‘Data’ list 38](#_Toc71899968)

[Figure 28: Parse list 'Data' to Pandas DataFrame 39](#_Toc71899969)

[Figure 29: Extract from print DataFrame 39](#_Toc71899970)

[Figure 30: Export DataFrame to Excel 39](#_Toc71899971)

[Figure 31: Extract from DataFrame(df) exported to Excel 40](#_Toc71899972)

[Figure 32: Create multi-index columns in DataFrame 41](#_Toc71899973)

[Figure 33: Transposed DataFrame Extract 41](#_Toc71899974)

[Figure 34: Create DataFrame by reading csv file 42](#_Toc71899975)

[Figure 35: Code used to generate area plot for all malware Samples 42](#_Toc71899976)

[Figure 36: Area plot for all malware samples 43](#_Toc71899977)

[Figure 37: Code to generate area plot 43](#_Toc71899978)

[Figure 38: Area plot for full dataset 44](#_Toc71899979)

[Figure 39: Code to generate plotly line graph 44](#_Toc71899980)

[Figure 40: Plotly Line Graph for full Dataset 45](#_Toc71899981)

[Figure 41: Hover Function in Plotly 46](#_Toc71899982)

[Figure 42: Area graph for modified dataset 47](#_Toc71899983)

[Figure 43: Bar graph for Ghostrat sample 50](#_Toc71899984)

[Figure 44: Bar graph for Swizzor Sample 52](#_Toc71899985)

[Figure 45: Bar graph for Hamweq Sample 1 54](#_Toc71899986)

[Figure 46: Bar graph for Hamweq Sample 2 55](#_Toc71899987)

[Figure 47: Bar Graph for Zeus Sample 1 57](#_Toc71899988)

[Figure 48: Bar Graph for Zeus Sample 2 58](#_Toc71899989)

[Figure 49: Bar Graph for Zeus Sample 3 59](#_Toc71899990)

[Figure 50: Bar Graph for CryptoLocker Sample 1 61](#_Toc71899991)

[Figure 51: Bar Graph for CryptoLocker Sample 2 62](#_Toc71899992)

[Figure 52: Bar Graph for CryptoLocker Sample 3 63](#_Toc71899993)

[Figure 53: Bar Graph for CryptoLocker Sample 4 64](#_Toc71899994)

[Figure 54: Bar Graph for CryptoLocker Sample 5 65](#_Toc71899995)

[Figure 55: Bar Graph for CryptoLocker Sample 6 66](#_Toc71899996)

[Figure 56: Reorder DataFrame Columns making Mult-index columns 67](#_Toc71899997)

[Figure 57: Transpose DataFrame 67](#_Toc71899998)

[Figure 58: Code to generate heatmap 68](#_Toc71899999)

[Figure 59: Heatmap for CryptoLocker Spiked Samples 68](#_Toc71900000)

[Figure 60: Plotly Line Graph for Spiked Samples 69](#_Toc71900001)

[Figure 61: Area Graph for Spiked Samples 70](#_Toc71900002)

[Figure 62: Heatmap for Spiked Samples 71](#_Toc71900003)

[Figure 63: Code to select file path to malware folder 72](#_Toc71900004)

[Figure 64: Create DataFrame & Extract of output 72](#_Toc71900005)

[Figure 65: Interactive Plotly Line Graph 73](#_Toc71900006)

[Figure 66: CryptoLocker Plotly Line Graph 74](#_Toc71900007)

[Figure 67: CryptoLocker Heatmap 75](#_Toc71900008)

[Figure 68: Zeus Line Graph 76](#_Toc71900009)

[Figure 69: Zeus Heatmap 77](#_Toc71900010)

[Figure 70: Reveton Samples Line Graph 78](#_Toc71900011)

[Figure 71: Reveton Heatmap 79](#_Toc71900012)

[Figure 72: Swizzor Samples Line Graph 80](#_Toc71900013)

[Figure 73: Swizzor Heatmap 81](#_Toc71900014)

[Figure 74: Hamweq Line graph 82](#_Toc71900015)

[Figure 75: Hamweq Heatmap 83](#_Toc71900016)

[Figure 76: Ghostrat Line Graph 84](#_Toc71900017)

[Figure 77: Ghostrat Heatmap 85](#_Toc71900018)

[Figure 78: Vanti Line Graph 86](#_Toc71900019)

[Figure 79: Vanti Heatmap 87](#_Toc71900020)

[Figure 80: Anubis Samples Line Graph 88](#_Toc71900021)

[Figure 81: Anubis Heatmap 89](#_Toc71900022)

[Figure 82: Heatmap for Full Dataset 90](#_Toc71900023)

[Figure 83: Category DataFrame for Full Dataset 91](#_Toc71900024)

[Figure 84: Code to generate 3D Surface Graph 92](#_Toc71900025)

[Figure 85: 3D Surface Graph for Full Dataset by Category 92](#_Toc71900026)

[Figure 86: CryptoLocker 3D Surface Graph 93](#_Toc71900027)

[Figure 87: Ghostrat 3D Surface Graph 94](#_Toc71900028)

[Figure 88: Hamweq 3D Surface Graph 95](#_Toc71900029)

[Figure 89: Swizzor 3D Surface Graph 96](#_Toc71900030)

[Figure 90: Vanti 3D Surface Graph 97](#_Toc71900031)

[Figure 91: Anubis 3D Surface Graph 98](#_Toc71900032)

[Figure 92: Zeus 3D Surface Graph 99](#_Toc71900033)

[Figure 93: Reveton 3D Surface Graph 100](#_Toc71900034)

# Introduction

## Background and Motivation

The fundamental core of information security is Confidentiality, Integrity and Availability known as the CIA triad (National Institute of Standards and Technology, 2020) and all devices and systems connected to the internet strive to prevent unauthorized access, disclosure, disruption, use, modification, or destruction of their data (National Institute of Standards and Technology, 2017). Malicious Software known as Malware is software which damages or steals data (AVG, 2020) and is a major threat to these devices and systems and the potential damage can be categorised under each pillar of the CIA triad.

While the trends in malware are constantly changing the threat is ever consistent. Amid the changing climate created by the global pandemic over the past year and a half, Malwarebytes (2021) identified the 4 primary goals of cybercriminals in 2020 to be:

* **Exploit Fear** – By playing on the fears of the masses tailored phishing campaigns were executed posing as healthcare advisories, Personal Protective Equipment (PPE) offers or charity requests to illicit a response from the victim (Malwarebytes, 2021).
* **Gather Intel** – Integrated with these phishing campaigns spyware and information stealers were injected into systems to better understand the access and security methods being used by all those now forced to work from home (Malwarebytes, 2021).
* **Upgrade** – While malware is constantly being modified to bypass security measures and improve performance, 2020 saw the adaptation of existing malware variants increasing the emphasis on brute-force attacks against Remote Desktop Protocols (RDP) in line with the increased number of clients signing in remotely to corporate servers (Malwarebytes, 2021).
* **Attack** – In 2020 the attack landscape moved away from the encryption of files and subsequent ransom demands to decrypt those files, to simply demanding a payment to prevent the posting online of sensitive data acquired during an intrusion, which would damage the reputation of the victim (Malwarebytes, 2021).

The Malwarebytes, State of Malware Report (2021) also identifies the top malware category detections by both Consumers and Businesses in 2020, as indicated in Figure 1 and Figure 2.

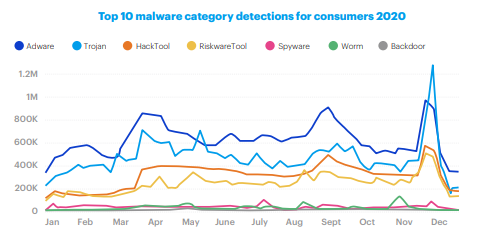


Figure : Top Malware Detection by Consumers 2020 (Malwarebytes, 2021)

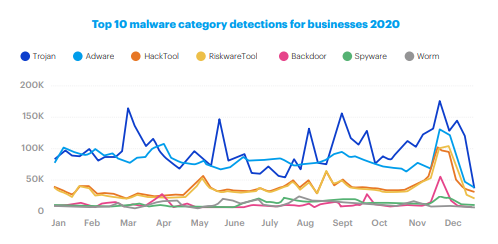


Figure : Top Malware Detection by Businesses 2020 (Malwarebytes, 2021)

For both consumers and businesses, we can see that adware and trojans were the most detected malware families in 2020 and the almost uniform numbers for hack and ransomware tools detected by businesses. Seeing the spike in December 2020 of approximately 1.2 million trojan detections amongst consumers gives an indication of the volume of malware in existence.

The vast number of malware samples and variants and the data generated by their analysis makes it impractical for human analysis, therefore we need to utilise more effective forms of analysis such as Visualization (Grégio & Santos, 2011).

Malware Attribute Enumeration and Characterisation (MAEC) is a structured language used for encoding and sharing information relating to malware based upon attributes like behaviours, artifacts, and relationships amongst malware samples (MITRE, 2020). It was introduced to bring about a standardised method of characterizing malware (Kirillov, et al., 2011) and though it was envisioned that it would be used for malware visualization no formal tools exist (MITRE, 2020). Using this standardised language this project will apply visualization techniques to malware samples based on their actions.

## Research Aims

The aim of this research is to visualize malware actions utilising MAEC-format dynamic analysis reports to extract and plot the relevant features. The MAEC-format reports are created by performing a dynamic analysis of the malware samples using Cuckoo Sandbox, these reports contain the actions observed in a structured format labelled under the ‘ActionNameVocab-1.0’ tags which will be extracted and visualized using the python graphing libraries, finally the visualizations will be evaluated to determine their success and effectiveness.

## Research Objectives

In achieving these aims the research objectives of this project are:

* To determine the differences between malware families and how they perform when they have infected a system.
* To analyse the MAEC language and its malware classification schema.
* To analyse malware samples and generate MAEC-format reports.
* Extract the malware actions from these structured reports.
* To identify the role visualization plays in understanding malware characteristics and behaviours.
* To illustrate malware behaviours using python graphing modules.
* Evaluate the performance of the visualization methods and their effectiveness in identifying malicious activity.

## Research Methodology Overview

To conduct this research the following steps were taken:

* Acquire a representative dataset of malware samples.
* Malware Classification using the MAEC reporting structure.
* Feature Extraction using a python program to parse the report files extracting action call tags.
* Data Pre-processing of the extracted features using Pandas.
* Visualisation of data using the python graphing libraries.
* Evaluation of Visualization techniques.

## Thesis Overview

This paper will introduce the reader to malware and some of the main family types, the MAEC language and its structure, as well as some Visualization techniques in the Literature Review.

The Methodology will outline the acquisition process of the malware sample MAEC analysis reports generated from dynamic analysis by Cuckoo Sandbox. The actions identified in these reports will then be used to generate a visual representation of the malware samples.

In the Implementation and Experimentation chapter the processes taken to execute the steps outlined in the methodology are expanded upon outlining the successes and failures throughout the process.

In the Findings chapter the behavioural visualizations generated for individual samples, malware families and the entire dataset are illustrated, discussed, and evaluated.

## Summary

This research takes the reader through the background and motivation for analysing malware and the reasoning behind a visualization-based approach. It examines the steps and the processes taken to implement each step, finally creating a visual representation of the findings. Finally, these visualizations are evaluated to assess their performance in capturing, summarising, and identifying similarities amongst samples.

# Literature Review

This project aims to provide a visual means of classifying and categorising malware. For this to be successful, it is important to understand what is meant by the term malware and how the different malware families perform once they have infected a system. MAEC provides a consistent framework to characterise malware and its behaviours, so a thorough understanding of the language and reporting structure was examined. In applying a visualization method to the extracted components MAEC reporting lends itself to a graph-based approach and some of the various graph-based methods available are examined.

## Malware Types & Behaviours

Malware is the broad term used to describe software specifically designed to cause damage, disrupt, or illegally access an individual device, server, or network (McAfee, 2021). It can be further broken down into different families categorised by how they spread or the actions they perform. Sophos provides a ‘Threatsaurus’ designed to give the user an understanding of these different families and the actions they perform (Sophos Ltd., 2013).

### Adware

McAfee define Adware as programs designed to display unwanted advertising on a computer in the form of banners or popups when a given action is performed, they are typically installed in exchange for another service such as accessing another program free of charge (McAfee, 2021).

Adware is not always a bad thing but becomes a threat when:

* It installs itself without consent.
* It installs itself in applications other than the one it came with.
* It hijacks the web browser.
* It gathers browsing data without consent.
* It becomes difficult to uninstall (Sophos Ltd., 2013).

### Backdoor Trojan

This is malicious software which comes disguised as a legitimate application allowing an attacker to take control of a system without authorisation, installing itself in the start-up procedures of the device for persistence (Sophos Ltd., 2013). Once executed they can steal data, monitor activities, initiate an attack, or crash a device (McAfee, 2021). Some of the signs that identify the presence of a Trojan is that the system starts to malfunction, changes may appear on the desktop, the taskbar may disappear, or antivirus systems may become disabled (Nord Pass, 2021). Examples include ‘Ghostrat’ (Infosec, 2015), ‘Zeus’ (Kaspersky, 2021), ‘Swizzor’ (Microsoft, 2007), ‘Emotet’, ‘KeyBoy’ and ‘Back Orifice’ (Infosec, 2019).

### Bot

Bots are malware that infect a device allowing them to be remotely controlled by an attacker to carry out an attack, the infected device is then referred to as a Zombie. When an attacker has a cluster of these devices at their disposal, we refer to them collectively as a botnet (Sophos Ltd., 2013). An attacker uses these hijacked devices to carry out other attacks such as Distributed Denial of Service (DDoS) attacks to bring down a website, generating fake traffic on a 3rd party site for financial reward, or emailing spam to countless internet users (Norton Life Lock, 2021). Examples include the ‘Zeus’ trojan which is considered one of the most successful botnet software samples (Kaspersky, 2021).

### Fileless Malware

Fileless malware can infect a system without installing software or writing to disk. It uses operating system tools like Microsoft PowerShell to make an attack appear like a genuine running process leaving no trace on the hard drive. It spreads by tricking users using phishing emails or malicious advertising (‘malvertising’) delivering the malware to memory through scripts (Abraham, 2019). Typically used for lateral movement between devices they obscure themselves in whitelisted or trusted applications abusing the trust model of security applications (McAfee, 2021). The fact that no trace is left behind makes it difficult to remove or even detect (McAfee, 2021).

### Keyloggers

This is malware installed on a device to record keystrokes it can be used to steal usernames, passwords, or other sensitive data (Sophos Ltd., 2013). While legitimate uses do exist for keyloggers, to a hacker they can provide a wealth of information such as PIN codes, passwords, email details or bank account numbers allowing them access to valuable financial or personal information used for identity theft. They are difficult to detect but browsing performance may be impacted, error screens may appear loading graphics or mouse/keystrokes pause or do not appear onscreen (McAfee, 2013).

### Mobile Malware

This is malware specifically designed to operate on mobile devices. Android devices allow download of apps from file sharing sites and popular programs can be infected with malware, but this is not the only risk as even legitimate apps can pose a privacy risk to user data (Sophos Ltd., 2013). This threat continues to escalate with companies allowing employees to use personal devices on their corporate networks (Crowdstrike, 2021). Distribution methods for this type of malware include:

* Mobile Phishing - where the user is tricked into providing their information.
* Spoofing - disguising trusted websites with fake entities to steal information.
* Rooting or Jailbreaking – is the bypassing of internal protections allowing unrestricted access (Crowdstrike, 2021).

Examples of mobile malware include ‘Anubis’ (Zero Day, 2019), ‘AceDeceiver’ and ‘XcodeGhost’ for iOS, as well as ‘Ghost Push’ and ‘Gooligan’ on Android (IDG Communications, 2021).

### Ransomware

This is malware which encrypts the device file system preventing access until payment of a ransom. It may use a symmetric key, where the password is hidden in the trojan code or asymmetric keys, where the key used to encrypt is not the same as the one used to decrypt making to impossible to recover (Sophos Ltd., 2013). Due to its profitable nature, it has become one of the most significant forms of malware (McAfee, 2021) and payment of the ransom does not always ensure that access to the system can be regained (Crowdstrike, 2021). The main methods for distribution are email phishing or the use of an exploit kit to manipulate a security flaw in a system (Crowdstrike, 2021). Examples include ‘Reveton’ (FBI, 2012), ‘CryptoLocker’, ‘WannaCry’, ‘Locky’ and ‘Bad Rabbit’ (Kaspersky , 2021).

### Rootkits

This is an application designed to conceal the presence of other malware like adware or spyware on a device (AVG, 2020). The presence of these tools allows the hacker to steal valuable information or disable security software (Sophos Ltd., 2013). The presence of a rootkit is difficult to detect but some common signs may include error messages in white text on a blue screen ‘Kiss of Death’, web browser redirection, slower performance, or Windows Setting changes. Distribution methods include phishing emails with malicious links or vulnerability exploits. Examples include ‘Flame’ and ‘Machiavellli’ (AVG, 2020)

### Spyware

This is malware which permits the gathering of sensitive information without permission (Sophos Ltd., 2013). This information may include usernames and passwords for sites you visit, payment information or emails you send or receive. It attaches itself to the operating system and may be installed as part of a legitimate program. Distribution methods may include security vulnerabilities, phishing, spoofing, trojans or mobile device spyware (Malwarebytes, 2021). Examples of Spyware include ‘CoolWebSearch’, ‘Gator’, ‘TIBS Dialer’ and ‘Zlob’ (Software Lab, 2021).

### Viruses

A Virus attaches itself to clean files which in turn spread to other clean files, their effects can range from annoying messages, to stealing data, to allowing an attacker access to the device and are usually packaged as an executable, .exe file (AVG, 2020). They are distributed by attaching themselves to other programs or automatically running on execution of certain files (Sophos Ltd., 2013). A virus may also be disguised as an attachment to an email, and once opened the device is infected (McAfee, 2021). Examples include ‘MyDoom’, ‘ILOVEYOU’ and ‘Slammer’ (Norton Life Lock, 2021).

### Worms

Worms are classified as malware that have the ability to self-replicate and spread throughout a system from device to device exploiting vulnerabilities (Sophos Ltd., 2013). They do not require user interaction to allow them to function (McAfee, 2021). The worm takes advantage of vulnerabilities in the software on the system modifying, deleting, or even injecting other forms of malicious software onto the system (Norton Life Lock, 2021). Tell-tale signs of the presence of a worm include reduced performance and speed, the presence of new files or deletion of existing files, worms will also have a huge effect on available hard drive space (Norton Life Lock, 2021). The first internet worm was called the ‘Morris Worm’ released by Robert Morris in 1988 and infected various machines with multiple exploits including password sniffing, buffer overflows and debugging routines in mail components (Spam Laws, 2021). Other examples include ‘Stuxnet’ (Holloway, 2015), ‘Hamweq’ (Microsoft, 2017), ‘The Storm Worm’ and ‘SQL Slammer’ (Software Lab , 2021).

## Common Malware Distribution Methods

Hackers are constantly updating and modifying the methods used to deceive the user into downloading, installing, or running malware on their devices such as email attachments, malicious links, infected storage devices or malvertising, but perhaps the most effective is negligence in not updating and patching software or exploiting human behaviour (University of Delaware, 2015). Some common distribution methods include:

### Cut and Paste Exploit

Browsing for specific commands for Windows Command Prompt, a website will display the required text but the hacker will hijack the user’s copy-paste command using JavaScript triggering a ‘keydown event’ causing a delay in the function overwriting the text actually copied with the malicious script and perhaps adding an auto-run suffix allowing the user no time to realise their mistake (Batt, 2016).

### False ‘Download Now’ Links

Searching the internet for a desired software program can lead the user to malicious links promising the desired program but a well-placed ad with a ‘download now’ button could trick the user into hastily downloading a completely different program (Batt, 2016).

### Email Spear Phishing

This is a targeted spoofing fraud attempt attempting to gain unauthorised access to confidential data or to infect a system with malware while appearing to come from a trusted source (Swinhoe, 2019).

### Messages and Posts from Trusted Sources

Exploiting established trust relationships tricking the user into clicking on malicious links, they often are initiated by hacking a Social Media or Email account and sending messages to all its contacts (Batt, 2016).

### USB Thumb Drives

USB drives come with an autorun feature meaning that once the storage device is mounted by the computer Operating System (OS) malware contained on the device is launched (Aru & Chiaghana Chukwunonso, 2018). In 2010 the Stuxnet worm attacked Iranian Nuclear facilities and is believed to have infiltrated the system through a workers USB Stick (Holloway, 2015).

## Safeguarding your system from Malware

Good internet security can prevent or at least minimise the damage caused by most malware variants and, at a minimum, the steps taken should include:

* The installation of a Security Suite such as McAfee with the ability to detect the presence of malware on a device, remove and sanitise the device and prevent further attacks (Windows Central, 2017).
* Perform regular updates on the computer operating system allowing the installation of patches for known vulnerabilities or set updates to run automatically (Windows Central, 2017).
* Similarly perform regular updates on any applications on the device to patch known vulnerabilities.
* Do not download or open attachments or click on links from email addresses you do not recognise (Windows Central, 2017).
* Utilise a Firewall for internet browsing (Windows Central, 2017).
* Do not visit websites known to be distributors of malware, a full-service Internet Security suite will alert the user to such sites (Norton Life Lock, 2021).
* Never insert an unknown USB Storage Device into your system (Cyware Labs, 2019).
* Educate and Train users (Swinhoe, 2019).
* Backup, Backup, Backup – always maintain an up-to-date offline backup of your system (Windows Central, 2017).

## Malware Attribute Enumeration and Characterization (MAEC)

MAEC was introduced to provide a standardised language to represent all known types, variants, and manifestations of malware (Kirillov, et al., 2011). Sponsored by the US Department of Homeland Security (DHS) and overseen by the MITRE Corporation the first version was released in 2011 and has evolved to the current version - MAEC 5.0 released in October 2017 (Cyware, 2019) .

### Language Overview

The MAEC language consists of three main parts a vocabulary, grammar, and standardised output, as shown in Figure 3.



Figure 3: MAEC Core Components (MITRE Corporation, n.d.)

#### Vocabulary

The vocabulary uses a series of dictionaries to define the three levels of malware attributes.

* **Low-Level Attributes –** Thesefocus on ***WHAT the malware does***. They typically represent the actions carried out by the malware like network activity, hardware accesses or system state changes, for example inserting a registry key of file creation (Beck, et al., 2014).
* **Mid-Level Behaviours –** These are concerned with defining the purpose of low-level actions organisingthem in clusters. Essentially, they define ***HOW the malware operates*** which can be useful in the triage, analysis, and detection process. For example, a low-level registry entry creation or modification does not reveal the intent of a malware instance, but the behaviours provide more information such as the registry key may ensure the malware executes during the boot process (Beck, et al., 2014).
* **High-Level Capabilities –** this is the grouping of mid-level behaviours depending on the function of the malware. They describe ***WHAT the malware can do***, for example when malware executes during bootup this is generally concerned with ‘*Persistence’* capabilities. Other examples of capabilities include *‘Data Theft’, ‘Self-Defence’*, and ‘*Propagation’* (Beck, et al., 2014).

#### Grammar

The grammar provides a baseline for malware repositories and contains at a minimum the following elements:

* **Namespaces** – this is the grouping of behaviours, mechanisms, and other enumerated attributes into defined classes. Where possible namespaces follow the MITRE Making Security Measurable (MSM) standards to characterise malware more accurately (Kirillov, et al., 2011).

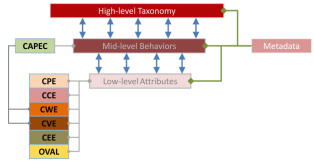


Figure : MAEC - MSM Standards Relationship (Kirillov, et al., 2011)

* + ***Common Attack Pattern Enumeration and Classification (CAPEC)*** to describe attack patterns through an industry standard dictionary of known patterns (MITRE, 2021).
  + ***Common Configuration Enumeration (CCE)*** describing vulnerabilities exploited by configuration issues using unique identifiers (National Institute of Standards and Technology, 2020).
  + ***Common Event Expression (CEE)*** describing logged events associated with malware activity (Kirillov, et al., 2011).
  + ***Common Platform Enumeration (CPE)*** provides a structured naming system for Information Technology Systems, packages and software including a formal name format, a method for name checking and description format (National Institute of Standards and Technology, 2021).
  + ***Common Vulnerabilities and Exposures (CVE)*** to link vulnerabilities exploited and their automated patch through CVE-compatible tools (MITRE, 2021).
  + ***Common Weakness Enumeration (CWE)*** to compare to a list of software and hardware weaknesses searching by keyword or CWE-ID number (MITRE, 2021)
  + ***Open Vulnerability Assessment Language (OVAL)*** to report on the machine state of computer systems encoding system details in a standardised language(MITRE, 2016)
* **Relationships** – set of rules to define relationships between namespaces. Members of the mid-level behaviours can be composed of multiple low-level attributes such as ‘related-to’ or ‘downloaded-by’.
* **Properties** – general set of properties which apply to attributes and namespaces, with a specific set for behaviours like number of occurrences, child/parent relationship to other behaviours (Kirillov, et al., 2011).
* **Metadata** – MAEC characterises all the malware metadata describing common actions and the reasoning behind them ranging from metadata associated with behaviours such as insertion mechanism, to malware artifacts like file hashes (Kirillov, et al., 2011)

#### Output Formats

Output formats like the XML schema act like a container to store and transport MAEC-encoded information about malware. The basic components of a MAEC bundle are outlined in Figure 5 below (MITRE, 2020).

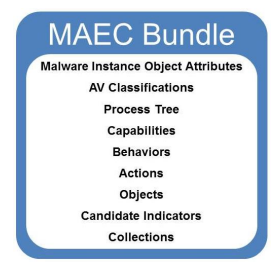


Figure : MAEC Bundle Overview (Beck, et al., 2014)

* **Malware Instance Object Attributes** – provide details about the instance that the bundle characterises such as file object with some attributes like name, size etc (MITRE, 2020).
* **AV Classifications** – refer to Anti-Virus Scanner classifications assigned to the malware instance such as the AV Vendor Name, Hash integrity value or classification Name assigned by the vendor (MITRE, 2020).
* **Process Tree** – illustrates the order of execution of malware processes, fields include the parent process, spawned processes, and any initiated actions (MITRE, 2020).
* **Capabilities** – constitute the properties and objectives of a malware instance such as ‘data-theft’, ‘persistence’ or ‘privilege-escalation’ (MITRE, 2020).
* **Behaviours** – are textual description of the behaviour and any related actions or relationships such as ‘crack-passwords’, ‘erase-data’ or ‘encrypt-files’ (MITRE, 2020).
* **Actions** – refer to the type of action such as ‘create file’, ‘copy-file’, discovery method and relationship to other actions (MITRE, 2020).
* **Objects** – describe the type of object such as ‘file’ and properties such as ‘file name’ and relationships to other objects (MITRE, 2020).
* **Candidate Indicators** – represent components that identify the existence of a malware instance for example the presence of a specific registry key object or examination of a certain ‘send http get request’ action (MITRE, 2020).
* **Collections** – are collective representations of Behaviour, Action, Object and Candidate Indicators with information such as text description, how elements are related and list of all items (MITRE, 2020).

### MAEC Data Model

The MAEC data model can be defined as a connected series of nodes and edges. Top level objects such as Behaviours, Malware Actions, Malware Families and Malware Instances define the nodes while the links describing the relationship between objects define the edges (MITRE, 2020).

Figure 6 details a sample MAEC data model, the following are the characteristics depicted in this image:

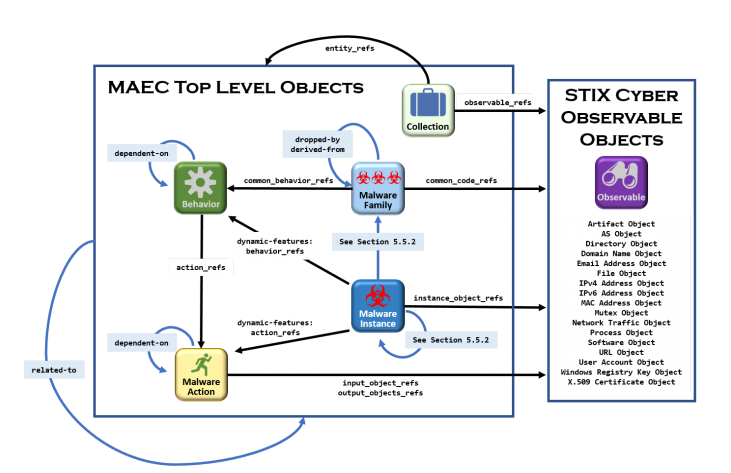


Figure : MAEC Data Model (MITRE, 2020)

* **Top level objects** – The top-level objects such as *Behavior, Collection, Malware Family, Malware Instance* and *Malware Action* are represented in the coloured boxes.
* **Embedded Relationships –** such as *common\_behavior\_refs, common\_code\_refs, observable\_refs, entity\_refs, action\_refs, instance\_object\_refs, dynamic-features: behavior\_refs, dynamic-features: action\_refs, input\_object\_refs, output\_objects\_refs* are represented by the black direction arrows*.* The names correspond to property names.
* **Direct Relationships** – such as *Dependent-on, related-to, dropped-by, derived-from, downloaded-by, variant-of, ancestor-of* are highlighted in a blue box with a blue direction arrow. The names correspond to the literal values for the relationship type.

### Advantages of MAEC

There are several benefits to using MAEC:

* **Elimination of ambiguity and inaccuracy in malware descriptions –** improved communication relating to anti-malware information between human-to-human, human-to-tool, tool-to-tool and tool-to-human (MITRE, 2020).
* **Reduced duplication of malware analysis efforts –** common characterisation methods and standardised reporting methods making it easier to determine if malware has already been analysed (MITRE, 2020).
* **Improved general awareness of malware –** standardised characterisation allows for increased public awareness of threats and activity of malware (MITRE, 2020).
* **Decreased overall response time to malware threats –** faster mitigation and response as countermeasures for previously identified samples can be leveraged (MITRE, 2020).

### Associated Language

MAEC 5.0 streamlines previous versions simplifying and refactoring complicated components and aligning the design principles with Structured Threat Information Expression(STIX) 2 (MITRE, 2017).

#### Structured Threat Information Expression (STIX)

STIX is a structured language that allows users to share, store and analyse Cyber Threat Intelligence (CTI) consistently in a machine-readable manner. The current version is STIX 2.1 which defines eighteen STIX Domain Objects (SDO’s) and two STIX Relationship Objects (SRO’s) (OASIS Open, 2021). STIX Cyber-observable Objects (SCO’s) document what happened on a network or host and when associated with the SDO’s give a better understating of the threat landscape (OASIS Open, 2021).

SDO’s are items which can be defined using STIX and include ***Attack Pattern, Campaign, Course of Action, Grouping, Identity, Indicator, Infrastructure, Intrusion Set, Location, Malware, Malware Analysis, Note, Observed Data, Opinion, Report, Threat Actor, Tool*** and ***Vulnerability*** (OASIS Open, 2021).

The two SRO’s are ***Relationship*** (used to link two SDO’s or SCO’s describing the relationship between them) and ***Sighting*** (the belief an element in CTI was seen).

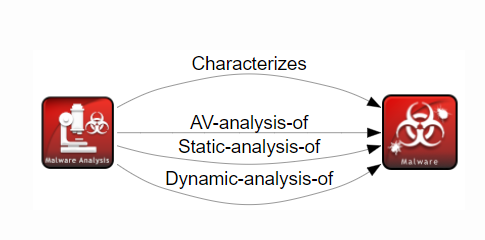


Figure : Visualised SDO Relationship between Malware and Malware Analysis (OASIS Open, 2021)

#### Cyber Observable eXpression (CybOX)

Cyber Observable eXpression (CybOX) is a multi-use language used in many aspects of cybersecurity including Threat intelligence, Security operations, Incident response, Digital forensics as well as Malware characterization (MITRE, 2020). It was introduced to provide a uniform approach to specify, characterize, capture, and communicate information relating to cyber observable objects such as file deletion or registry key modification and was integrated into STIX with Version 2.0 (MITRE, 2020). MAEC utilizes CybOX to characterize system and network events and behaviours (MITRE, 2020).

## Visualization

The use of visualization is a method used to hasten the malware analysis process utilising either static or dynamic analysis techniques, or a combination of both to gather information on potentially malicious programs (Efe & Hussin, 2020).

### Graph Based Visualization

Graph based malware visualization is based on relationships known as edges or links, between objects, known as vertices or nodes. It illustrates patterns and values in relationships such as the interaction, juxtaposition, and magnitude (Grégio & Santos, 2011). This is similar in format to the MAEC generated output and, thus, it appears to be the natural selection for MAEC malware visualization.

Linked graphs visualize data based on hierarchical relationships displaying the different dimensions of data either by the colour, position, or size of a node (Efe & Hussin, 2020). The Honeynet Project (2012) generate linked graphs from multiple attack sources as detailed in Figure 8.

Diagram

Description automatically generated

Figure 8: Attack graph construction from multiple sources (The Honeynet Project, 2012)

Figure 9 details a sample attack graph from the Honeynet Project (2012) with the Malware MD5 values coloured in green, the malicious hostnames extracted from Cuckoo reports coloured in black and the resolved IPs or fast-flux Ips in red. From this graph the ‘weak point’ or domains connected to lots of malware are easily identified (The Honeynet Project, 2012).

A picture containing text, watch

Description automatically generated

Figure 9: Linked Malware Attack Graph (The Honeynet Project, 2012)

To perform visualization based on malware behaviour it is necessary to analyse its execution. Grégio and Santos (2011) in their research monitored the behaviour of malware samples using a modified version of the Behaviour Evaluation from Malware Observation Tool (BEhEMOT), which automates the dynamic analysis process (Grégio, et al., 2011), to identify high level activities such as file write and delete and the system calls issued while undertaking these actions. To identify the relationships they selected a subset of their malware samples sending them to VirusTotal and retaining the positively identified samples. The resultant data was processed mapping vertices to each malware descriptor, applying a weight to the edges among descriptors based on the level of agreement among AV labeling between pairs of malware and the count of being in the same category. The data was visualised using the Java Universal Network/Graph (JUNG) Framework, a software library which allows for analysis, modeling and visualization of data (SourceForge, 2021). The resultant data was then grapgically represented. In order to identify correctly grouped samples they used the Tsunami Backdoor Family by colouring their vertices in red. The following graph shows a strong result with all but 2 samples of Tsunami being clustered together, represented by the red dots, while the remaining malware samples were represented by the green dots (Grégio & Santos, 2011).

A picture containing chart

Description automatically generated

Figure 10: Similarities based on AV labelling (Grégio & Santos, 2011)

### Bar Charts and Histograms

Bar Charts plot data using rectangular shaped columns which can be displayed vertically, horizontally, comparatively or stacked, to display information relating to the category on the x-axis measured by the height of the columns on the y-axis (Investopedia, 2021).

A comparative bar chart is depicted in Figure 11 containing bars grouped in 3’s, one to represent each of the legend items on the right and comparing their performance measured on the y-axis for each of the categories listed on the x-axis.

Chart, bar chart

Description automatically generated

Figure 11: Bar Chart detailing classifier performance (Nari & Ghorbani, 2013)

Nari and Ghorbani (2013) used this bar chart, Figure 11, to evaluate the accuracy of their malware classifier against 11 popular anti-virus programs. The results show how their approach outperforms AVG, Avast, Norman, Panda and TrendM and compares favourably with other programs.

Histograms organise data into specified ranges indicating the number of data points within that range. Organized in classes or bins the frequency of data is depicted by the height of the bar (ThoughtCo, 2019).

The histogram detailed in Figure 12 details the distribution of the file size measured in bytes on the x-axis for three classes, represented in the colours red, green, and blue, displaying their frequency on the y-axis.

Chart, bar chart, histogram

Description automatically generated

Figure :Histogram of File Size Distribution (Shalaginov, et al., 2018)

Shalaginov, *et al.* (2018) used this histogram in their research into Machine Learning Aided Static Malware Analysis, where they classified Portable Executable (PE32) Windows files by static characteristics, using different machine learning methods.

### Heatmaps

Heatmaps are a 2D representation of data using colour (Techopedia, 2021) each square shows the correlation between the x-axis and y-axis variable that intersects and that point. Ranging from +1 to -1, readings closer to 1 show a positive correlation, meaning as one increases so too does the other, values nearer -1 show a negative correlation meaning that as one increases the other decreases and vice-versa, while readings closer to 0 show no linear correlation (Stack Exchange, 2019). The colour scale is used to illustrate these values allowing them to be read immediately.

Anderson *et al.* (2012) in their research used heatmaps to detail the similarities identified between benign and malware samples as depicted in Figure 13. The top left block of the heatmap is the 780 malware samples and the bottom right block are the 776 benign samples. The similarities between samples are represented in the off-diagonal blocks.



Figure : Heatmap visualizing benign and malware sample similarities (Anderson, et al., 2012)

### Thread Graphs

Thread graphs are used for the chronological analysis of malware behaviour displaying executed system commands and threads spawned from them (Efe & Hussin, 2020).

Figure 14 details a thread graph analysis of Adultbrowser malware detailing the api-call actions on the y-axis and the sequence of the performed actions on the x-axis of (Trinius, et al., 2009).

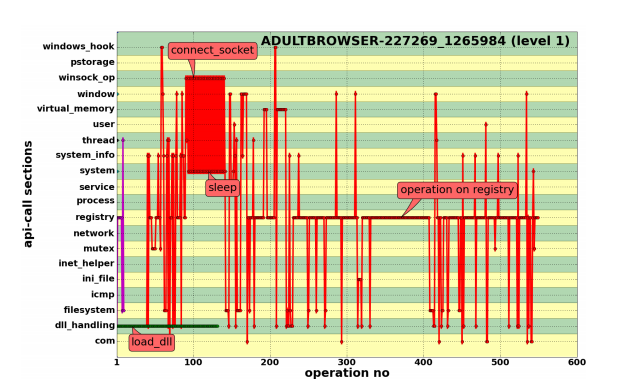


Figure : Malware analysed using a thread graph (Trinius, et al., 2009)

In this sample Tinius *et al.* (2009) identified that one thread was responsible for most actions.

### 3D Malware Visualization

Alex Dragulescu visualized a series of worms, trojans, viruses and spyware code in a collection of limited-edition prints. Although works of art they were created by disassembling the code tracking and analysing API calls, memory addresses and sub routines and mapping their density, frequency, and groupings to an algorithm to produce the 3D entities (Dragulescu, 2008) an example of which is shown in Figure 15.

A firework in the sky

Description automatically generated with medium confidence

Figure : 'ScamFraud4198' (Dragulescu, 2008)

Skyrails a tool to visualize traffic on a network, highlighting suspected anomalies, can extract data and utilizing the Dragulescu algorithm create an image (Swanson, 2008). Figure 16 details an image rendered from code extracted from Skyrails by this algorithm.

A picture containing text

Description automatically generated

Figure : Skyrails Dragulescu Visualization (Swanson, 2008)

### Summary

This section examines malware types and behaviours for families including Adware, Backdoor Trojans, Bots, Fileless Malware, Keyloggers, Mobile Malware, Ransomware, Rootkits, Spyware, Viruses, and Worms. Some common malware distribution methods are also investigated together with methods of safeguarding your system from malware. The MAEC language, its components and structure are also examined together with the associated languages of STIX and CybOX and the advantages of using MAEC. The necessity for visualization techniques in analysing malware is also discussed while some methods of visualization are examined including Graph based methods, Bar Charts and Histograms, Heatmaps, Thread Graphs and 3D visualization.

# Methodology

## Overview

There are currently no MAEC malware visualization tools in existence. By examining the dynamic analysis MAEC output .xml files generated from running malware samples from different families through Cuckoo Sandbox the malware actions are extracted to perform the visualization.

## Proposed Method

* Acquire Dataset.
* Classify malware based on actions.
* Feature Extraction.
* Data Pre-processing.
* Data Analysis and Visualisation.
* Evaluation of Visualization Techniques.

### Software Used

#### Python

Python 3.8.3 was used to write the programs used to carry out the dataset acquisition, data pre-processing, feature extraction and visualizations performed throughout the course of this research.

#### Python Libraries

The Python libraries used in conducting this research were:

* NumPy – essential in performing scientific computing operations supporting large multidimensional arrays and matrices and includes mathematical functions to perform on these arrays (2U, Inc, 2020).
* Pandas – built on top of NumPy this utilizes data structures to manipulate numerical tables (2U, Inc, 2020).
* Matplotlib – this is a 2D plotting library used to create data visualizations such as scatter plots and histograms (2U, Inc, 2020).
* Seaborn – this builds upon matplotlib using enhanced graphics to create dynamic visualizations such as heatmaps (2U, Inc, 2020).
* Plotly – this is used to produce high quality, interactive graphs and can incorporate multiple axes to produce 3D representations (Plotly, 2021).

#### PyCharm

PyCharm Community Edition 2020.3.3 was the Integrated Development Environment (IDE) used to write and edit the python code used in conducting this research.

#### Jupyter Notebook

In addition to PyCharm the interactive web based Jupyter Notebook 6.1.4 was used as it allowed more freedom in executing small chunks of code without altering the previous code chunks. It also allowed different visualizations to be run without re-running the entire program each time or affecting the data used to generate the visualizations.

## Malware Sample Collection

The MAEC classified dataset was created from dynamically analysed malware samples in Cuckoo Sandbox and the output reported using Cuckoo’s MAEC reporting feature. There were existing datasets available on GitHub analysed in this fashion (GitHub Inc., 2015), (GitHub, Inc., 2014) which were utilized in performing the analysis in this paper.

Table : GitHub MAEC Dataset Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Source** | **Malware Family** | **Type** | **Sample Count** |
| MAEC Project/datasets | Anubis | Android Malware | 37 |
| ptrac3/MAEC-Dataset | CryptoLocker | Ransomware | 96 |
| ptrac3/MAEC-Dataset | Ghostrat | Backdoor | 97 |
| ptrac3/MAEC-Dataset | Hamweq | Worm | 73 |
| ptrac3/MAEC-Dataset | Reveton | Ransomware | 55 |
| ptrac3/MAEC-Dataset | Swizzor | Trojan | 160 |
| ptrac3/MAEC-Dataset | Vanti | Packer for Password Stealing | 104 |
| MAEC Project/datasets | Zeus | Trojan | 100 |

## Cuckoo Sandbox and MAEC Classification

Cuckoo Sandbox is an open-source malware analysis system (Cuckoo, 2019). All the malware samples used in this research were dynamically analysed in Cuckoo and the output reported using the MAEC reporting module to create a .xml file for each sample. These files contained information relating to the malware and its execution such as:

* **File Metadata** – the File Size in Bytes, File Format and MD5, SHA-1 and SHA-256 Hash Values.
* **Process Tree** – The Process ID’s generated during execution of the malware and its relevant-timestamp.
* **Actions** – The actions the malware performs classified under the category headings such as Library, Network, Registry, File, HTTP, Directory and DNS actions.

The action category MAEC namespaces observed and analysed in this research are listed in Table 2.

|  |  |
| --- | --- |
| ***MAEC Namespace*** | |
| *DNSActionNameVocab-1.0*  *DebuggingActionNameVocab-1.0*  *DeviceDriverActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HTTPActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *IPCActionNameVocab-1.0* | *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Table : MAEC Namespace Categories

## Feature Extraction

Using a python program, each malware .xml file was read and all the individual actions under the Category Headings listed in Table 2 were extracted from each sample. Each unique action under each category was summated for each sample and a total count calculated. A list of all the unique actions observed in the dataset and the categories they belong too are listed in Table 3.

Table : MAEC Unique Actions Table

|  |  |
| --- | --- |
| **Category** | **Action** |
| **DNSActionNameVocab-1.0** | send dns query |
| **DebuggingActionNameVocab-1.0** | check for remote debugger |
| **DeviceDriverActionNameVocab-1.0** | load driver |
| **DirectoryActionNameVocab-1.0** | create directory |
| **FileActionNameVocab-1.0** | copy file |
| create file |
| delete file |
| find file |
| get file attributes |
| modify file |
| move file |
| open file |
| read from file |
| rename file |
| send control code to file |
| set file attributes |
| write to file |
| **GUIActionNameVocab-1.0** | find window |
| **HTTPActionNameVocab-1.0** | send http get request |
| **HookingActionNameVocab-1.0** | add windows hook |
| **IPCActionNameVocab-1.0** | create named pipe |
| read from named pipe |
| write to named pipe |
| **LibraryActionNameVocab-1.0** | get function address |
| load library |
| **NetworkActionNameVocab-1.0** | connect to socket address |
| download file |
| **ProcessActionNameVocab-1.0** | create process |
| kill process |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory |
| modify process virtual memory protection |
| read from process memory |
| write to process memory |
| **ProcessThreadActionNameVocab-1.0** | create remote thread in process |
| create thread |
| get thread context |
| kill thread |
| set thread context |

|  |  |
| --- | --- |
| **RegistryActionNameVocab-1.0** | close registry key |
| create registry key |
| delete registry key |
| delete registry key value |
| enumerate registry key subkeys |
| enumerate registry key values |
| get registry key attributes |
| modify registry key value |
| monitor registry key |
| open registry key |
| read registry key value |
| **ServiceActionNameVocab-1.0** | create service |
| delete service |
| modify service configuration |
| open service |
| send control code to service |
| start service |
| **SocketActionNameVocab-1.0** | bind address to socket |
| close socket |
| connect to socket |
| create socket |
| get host by name |
| listen on socket |
| receive data on socket |
| send data on socket |
| send data to address on socket |
| **SynchronizationActionNameVocab-1.0** | create mutex |
| open mutex |

## Data Pre-processing

During this stage of the process the data filtered from the .xml malware files was sorted to obtain the proper format for the Pandas Data Frame. Each action is represented by a column with the count value for each sample being the value under each heading. Once the Data Frame has been prepared, the next step was to find a suitable visualization method to display the information.

## Visualization

Data visualization transforms numerical data into visual representations making them more aesthetically pleasing (IntelliPaat, 2020). Regarding malware, visualization techniques are used in the detection, classification, and behaviour discovery and the methods used include area, pie, lines, dots, and 3D volume slicing (Efe & Hussin, 2020).

### Data Visualised

The data extracted to model the visualisations in this research are actions under the 18 categories identified in Table 2. For example, under Library Actions we see individual actions of ‘get function address’ and ‘load library’ are visible in the malware samples analysed. Table 3 details all the individual actions identified in the malware dataset and the categories they belong to.

### Visualization Libraries

#### Matplotlib Visualization

Matplotlib is a python library used to create visualizations from simple line plots to scatter plots and more complex 3D visualizations. For example, the area graphs were generated using matplotlib.

#### Seaborn Visualization

Seaborn is part of the pyData Stack and so accepts Pandas DataFrame objects (Medium, 2018) and can be considered an extension of matplotlib. For example, the heatmaps visualized throughout were generated using seaborn.

#### Plotly Visualization

Plotly is an open-source library which renders interactive graphs including line plots, scatter plots, area plots and histograms (Analytics Vidhya, 2020). For example, the interactive line plots and 3D surface graphs were generated using plotly.

### Visualization Evaluation

In response to the large amount of data being generated and the different visualization techniques implemented to display this information, Freitas *et al.* (2002) proposed a system for the evaluation of information visualization techniques based on the criteria detailed in Figure 17.

Diagram

Description automatically generated

Figure : Visual Representation Criteria (Freitas, et al., 2002)

The criteria used to evaluate visualizations under this proposal are:

* **Limitations**: these refer to constraints which affect the quality of the visualization such as the maximum number of data points or the size of display (Freitas, et al., 2002).
* **Cognitive Complexity**: this refers to the characteristics such as the relevance of the displayed information, the volume or extent of the data (Freitas, et al., 2002).
* **Spatial Organization**: refers to the layout of the graph, the ease of identifying elements or overall distribution. It is concerned with whether the data is hidden or occluded and well organised or logical as well as the distribution of elements (Freitas, et al., 2002).
* **Information Coding**: the addition of symbols or characteristics to the mapping of data and visual elements can be used to better interpret data points examples include clustering elements (Freitas, et al., 2002).
* **State Transition:** this refers to the remodelling of a visualization following a user action , it is quantified by how long it takes to rebuild the visualization and how this effects the spatial organization (Freitas, et al., 2002).

The visualizations generated during this research are evaluated using this rubric, see section 5.5.

### Types of Visualisation

The visualizations generated in this research are listed in Table 4:

Table : Visualization Type Table

|  |  |  |  |
| --- | --- | --- | --- |
| **Visualization Library** | **Graphs Type** | **Data** | **Location** |
| Matplotlib | Area Graph | Details action call count for each sample | Figure 36, Figure 38  Figure 42, Figure 61 |
| Seaborn | Heatmap | Used to correlate data and identify similarities between samples | Figure 59, Figure 62  Figure 67, Figure 69  Figure 71, Figure 73  Figure 75, Figure 77  Figure 79, Figure 81  Figure 82 |
| Plotly | Line plots | Interactive line graphs which detail individual counts for all samples by hovering over sample data point | Figure 40, Figure 41  Figure 60, Figure 65  Figure 66, Figure 68  Figure 70, Figure 72  Figure 74, Figure 76  Figure 78, Figure 80 |
| Bar Graphs | Used to display the total count for each individual action represented by bars with height dependent of the count | Figure 43, Figure 44  Figure 45, Figure 46  Figure 47, Figure 48  Figure 49, Figure 50  Figure 51, Figure 52  Figure 53, Figure 54  Figure 55 |
| 3D Surface Graph | Used to represent data in the form of a three-dimensional graph | Figure 85, Figure 86  Figure 87, Figure 88  Figure 89, Figure 90  Figure 91, Figure 92  Figure 93 |

## Summary

This chapter outlines the proposed method of creating a MAEC Malware Visualization tool and the software used to perform the tasks from acquiring the dataset, to extracting the key malware actions and generating visualizations of this data and the graphing libraries used to create them.

# Implementation and Experimentation

## Overview

This chapter details the processes performed in conducting this research from acquiring the dataset to extracting the malware actions performed from the MAEC reports, data pre-processing and generating visualizations.

## Acquire Dataset

There were existing datasets available on GitHub, **MAECProject / dataset** and **ptrac3 / MAEC-Dataset**, both of which were compiled by running various malware samples through dynamic analysis using Cuckoo Sandbox producing a MAEC report.

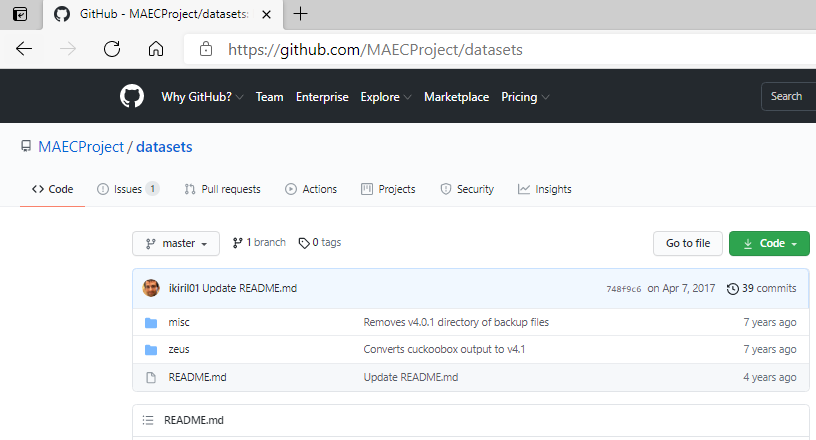


Figure : MAECProject / datasets (GitHub, Inc., 2014)

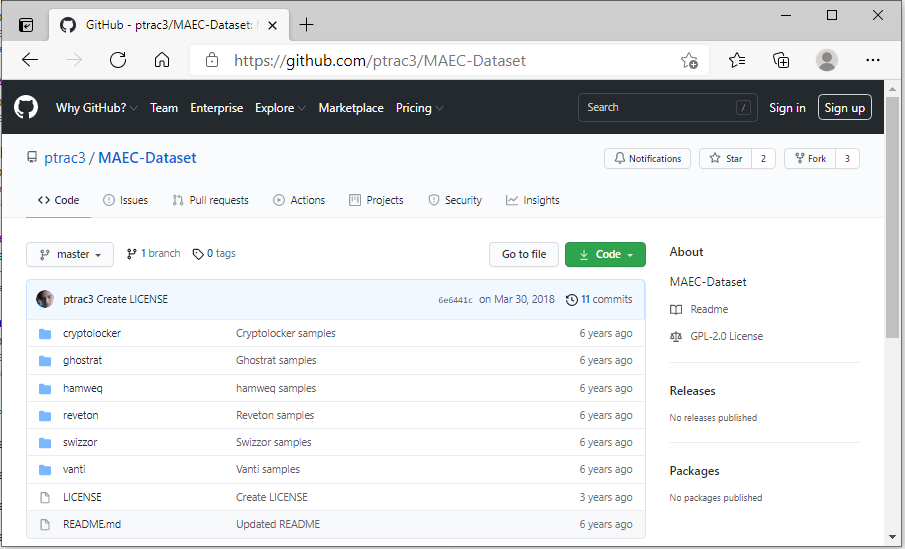


Figure : ptrac3 /MAEC-Dataset (GitHub Inc., 2015)

A separate web .xml file was generated for each sample, an example of which is shown in Figure 20.



Figure : Sample web .xml file (GitHub, Inc., 2014)

Then using a python program, the content of the web xml was downloaded to the system, writing the output as a new .xml file and keeping the GitHub naming format.

Text

Description automatically generated

Figure : Python program to download web .xml MAEC reports

In total 722 samples were acquired from this existing dataset as detailed in Table 1: GitHub MAEC Dataset Table.

## Feature Extraction

The existing GitHub MAEC datasets were all saved setting the **‘include\_namespaces = False’** which meant that the outputted .xml files were easier to read but unfortunately could not be parsed using the element tree xml parsing function in Python (MITRE, 2015). To overcome this, each .xml file was read line-by-line using a Python program, extracting actions performed by the malware under each of the ‘ActionNameVocab-1.0’ elements listed in the MAEC schema documentation (MITRE, 2020). There are 24 ‘ActionNameVocab-1.0’ categories listed in this documentation, in total 18 of these action categories were observed in the dataset utilised in this research. Lines containing information not relevant to the behavioural modelling were disregarded.

Text

Description automatically generated

Figure : Files read line-by-line extracting MAEC ActionNameVocab-1.0

The MAEC ‘ActionNameVocab-1.0’ lines all appear in the following format as shown in Figure 23, to extract each individual action under each of the Action Category headings, a regular expression (REGEX) was used when reading the relevant line, adding the action to a category list Figure 24 .



Figure : ‘ActionNameVocab-1.0’ line format (GitHub Inc., 2015)

The action categories appear in the MAEC report in a CybOX format which supports regular expression modifiers (MITRE, 2020). A reluctant quantifier was used to match 1 or more instances, matching the smaller part of the input string excluding zero matches.



Figure : REGEX to extract malware action

The REGEX removed everything in the line enclosed in ‘<>’ brackets leaving just the action item, for example in the case of Figure 23, ‘open registry key’. The MAEC naming format for the action category matched from the ‘keep\_text’ is added to the list as a key, followed by ‘:’ and then the action stripping the leading white space from before the action and the ‘\n’ from after the action. Once the complete file has been read all actions matching the 18 action category types are extracted to a single dictionary and a count calculated for each unique action in the sample.



Figure : Action’s dictionary of the individual actions and their count

The dictionary for each malware sample is added to an overall list for all the malware samples in the folder.



Figure :File sample action dictionaries added to single list

Printing the data list, we can see the opening ‘{‘ for the sample dictionary, the MAEC naming format, ‘LibraryActionNameVocab-1.0’, followed by ‘:’, the action ‘get function address’ followed by ‘:’ and the unique count ‘32’ for the action observed in the file, all the remaining actions observed in the sample list are listed in a similar fashion and the dictionary closing ‘}’ before the next sample dictionary. This is detailed in Figure 27.

Text

Description automatically generated

Figure : Extract from printed ‘Data’ list

## Data Pre-processing

### Create DataFrame

To prepare the extracted data for visualization the list ‘data’ was parsed to a Pandas DataFrame.

Text

Description automatically generated

Figure : Parse list 'Data' to Pandas DataFrame

This resulted in a dataset of 722 rows (one for each malware sample as expected) and 66 columns each representing an individual action.

Graphical user interface, text, application, email

Description automatically generated

Figure : Extract from print DataFrame

Due to the size of this dataset, it was not possible to view in full using the Jupyter Notebook so the DataFrame was exported to Excel for closer analysis.

Graphical user interface, text, application

Description automatically generated

Figure : Export DataFrame to Excel

Once the file was opened in excel it was possible to view each individual action under the 18 categories they were extracted from.

Table

Description automatically generated

Figure : Extract from DataFrame(df) exported to Excel

### Transpose DataFrame

The DataFrame was modified making the column names multi-index by splitting them on the delimiter as illustrated in Figure 32.

Text

Description automatically generated

Figure : Create multi-index columns in DataFrame

Once the columns were in multi-index format it was possible to transpose the DataFrame using the ‘*df.T’* command and adding the column headings ‘Category’ and ‘Action’.

Graphical user interface, text, application, email

Description automatically generatedText

Description automatically generated

Figure : Transposed DataFrame Extract

## Generating Visualizations

To get an overview of the actions under each of the 18 categories a DataFrame was created using the newly constructed malware action category file, ‘malcat.csv’, modified from the extracted DataFrame excel file to display only 4 columns the ‘*Category’*, individual ‘*Action’*, a total count for each action *’Action Total’* and category ‘*Category Total’*. The content of this .csv file is listed in Appendix 8.2.

A picture containing text

Description automatically generated

Figure : Create DataFrame by reading csv file

This DataFrame was then used to plot the area graph Figure 36 using Pandas built-in visualization and matplotlib styling.

Text

Description automatically generated

Figure : Code used to generate area plot for all malware Samples

Chart, histogram

Description automatically generated

Figure : Area plot for all malware samples

This graph details how each individual action’s total in purple compares to the category totals for each of the 18 action categories in yellow. While this gives us an overview for the full dataset, we do not see how individual samples perform, to visualize this we use the original DataFrame created and generate another area plot.

Text

Description automatically generated

Figure : Code to generate area plot

Chart, histogram

Description automatically generated

Figure : Area plot for full dataset

Due to the large number and obvious disparity between samples this is quite difficult to interpret so an interactive plotly line graph was used to visualize the same data.



Figure : Code to generate plotly line graph

Chart, histogram

Description automatically generated

Figure : Plotly Line Graph for full Dataset

This created a nice feature as the individual action calls and their count for each sample were visible simply by moving the cursor each sample. The graph was created with the samples on the x-axis and the total action calls on the y-axis.

A picture containing text, screenshot, computer, desk

Description automatically generated

Figure : Hover Function in Plotly

Examining the data using the interactive Plotly graph and the excel spreadsheet extracted from the DataFrame identified 13 of the 722 samples as having more than 13,500 action calls recorded during their execution with most samples having under 1000 calls. Due to the vast range of these spiking samples, it made the data quite difficult to decipher in the lower ranges, to better display the remaining information these samples were excluded from the dataset and examined individually, see section 5.1. With these samples excluded a new DataFrame was constructed from the amended dataset and a new area plot generated, Figure 42.

Chart, histogram

Description automatically generated

Figure : Area graph for modified dataset

This too, while being somewhat easier to read, still showed vast spikes in the data so alternative methods of visualising the data were investigated, see section 5.

## Summary

This chapter documents the actual procedures implemented, the difficulties encountered during the process and how they were overcome. Code snippets are used to explain the actions performed and the resultant output.

# Results, Analysis, and Evaluation

## Files with large number of calls

The 13 files with the large number of action calls were identified as being 1 Ghostrat, 1 Swizzor, 2 Hamweq, 3 Zeus, and 6 CryptoLocker samples. Each of these samples were analysed individually.

### Ghostrat sample

The Ghostrat sample registered 31,214 action calls with 10,035 of those calls being ‘open registry key’ the full list of calls is listed in Table 5.

Table : Ghostrat Action Calls

|  |  |  |
| --- | --- | --- |
| **Category** | **Action** | **Calls Count** |
| **DNSActionNameVocab-1.0** | send dns query | 2 |
| **DirectoryActionNameVocab-1.0** | create directory | 7 |
| **FileActionNameVocab-1.0** | create file | 337 |
| delete file | 4 |
| find file | 162 |
| get file attributes | 141 |
| open file | 70 |
| read from file | 87 |
| send control code to file | 59 |
| set file attributes | 370 |
| write to file | 406 |
| **GUIActionNameVocab-1.0** | find window | 1 |
| **HTTPActionNameVocab-1.0** | send http get request | 2 |
| **LibraryActionNameVocab-1.0** | get function address | 5164 |
| load library | 235 |
| **NetworkActionNameVocab-1.0** | connect to socket address | 23 |
| **ProcessActionNameVocab-1.0** | create process | 4 |
| kill process | 2 |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory | 118 |
| modify process virtual memory protection | 11 |
| **ProcessThreadActionNameVocab-1.0** | create thread | 29 |
| **RegistryActionNameVocab-1.0** | close registry key | 371 |
| create registry key | 42 |
| enumerate registry key subkeys | 5272 |
| enumerate registry key values | 365 |
| get registry key attributes | 1737 |
| modify registry key value | 212 |
| open registry key | 10035 |
| read registry key value | 5614 |
| **ServiceActionNameVocab-1.0** | open service | 6 |
| start service | 1 |
| **SynchronizationActionNameVocab-1.0** | create mutex | 322 |
| open mutex | 3 |
|  | | **31,214** |

Using plotly this data was visualized using a bar graph as detailed in Figure 43, with the unique action calls are listed on the x-axis with the number of actions calls listed on the y-axis.

A picture containing chart

Description automatically generated

Figure : Bar graph for Ghostrat sample

Ghostrat is a backdoor trojan and as expected there is a large volume of Registry Action calls where it establishes persistence on the system enabling automatic execution on every restart (Trend Micro, 2012).

### Swizzor sample

The swizzor sample registered 67,745 action calls with almost all, 67,141 of those calls being ‘read from file’ the full list of calls is listed in Table 6.

Table : Swizzor Action Calls

|  |  |  |
| --- | --- | --- |
| **Category** | **Action** | **Calls Count** |
| **DNSActionNameVocab-1.0** | send dns query | 6 |
| **FileActionNameVocab-1.0** | create file | 7 |
| delete file | 1 |
| get file attributes | 2 |
| open file | 3 |
| read from file | 67141 |
| send control code to file | 3 |
| set file attributes | 1 |
| write to file | 2 |
| **LibraryActionNameVocab-1.0** | get function address | 55 |
| load library | 17 |
| **NetworkActionNameVocab-1.0** | connect to socket address | 28 |
| **ProcessActionNameVocab-1.0** | create process | 1 |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory | 2 |
| **RegistryActionNameVocab-1.0** | close registry key | 11 |
| create registry key | 5 |
| enumerate registry key subkeys | 209 |
| enumerate registry key values | 41 |
| get registry key attributes | 2 |
| modify registry key value | 24 |
| open registry key | 62 |
| read registry key value | 64 |
| **SocketActionNameVocab-1.0** | bind address to socket | 3 |
| create socket | 3 |
| get host by name | 7 |
| send data to address on socket | 42 |
| **SynchronizationActionNameVocab-1.0** | create mutex | 1 |
| open mutex | 2 |
|  | | **67,745** |

Using plotly this data was visualized using a bar graph as detailed in Figure 44 with the unique action calls listed on the x-axis and the number of action calls on the y-axis.

A picture containing calendar

Description automatically generated

Figure : Bar graph for Swizzor Sample

Swizzor is a trojan and primarily downloads other malware files by redirecting browser traffic to malicious web pages (Trend Micro, 2012). In this sample we do see DNS queries, where the client queries the DNS serer searching for the IP address for a Fully Qualified Domain Name (FQDN) potentially indicating this activity (OmniSecu.com, 2021). The large number of ‘read from file’ actions could potentially indicate a modification made to original malware sample.

### Hamweq Samples

The Hamweq samples registered 49,477 and 37,699 action calls respectively, both samples show most of their calls in Registry Actions, but the second sample shows a greater number of File Actions and a greater number of Library Actions ‘get function address’ calls. All calls are listed in Table 7.

Table : Hamweq Action Calls

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Action** | **S1** | **S2** |
| **DirectoryActionNameVocab-1.0** | create directory | 5 | 7 |
| **FileActionNameVocab-1.0** | copy file | 1 |  |
| create file | 4 | 432 |
| delete file | 1 | 4 |
| find file |  | 198 |
| get file attributes |  | 234 |
| open file | 4 | 135 |
| read from file |  | 142 |
| send control code to file |  | 92 |
| set file attributes | 2 | 370 |
| write to file | 1 | 406 |
| **GUIActionNameVocab-1.0** | find window |  | 1 |
| **LibraryActionNameVocab-1.0** | get function address | 126 | 7314 |
| load library | 16 | 236 |
| **NetworkActionNameVocab-1.0** | connect to socket address | 30 | 9 |
| **ProcessActionNameVocab-1.0** | create process |  | 4 |
| kill process | 1 | 2 |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory | 10 | 557 |
| modify process virtual memory protection | 7 | 27 |
| write to process memory | 71 |  |
| **ProcessThreadActionNameVocab-1.0** | create remote thread in process | 1 |  |
| create thread | 5 | 29 |
| get thread context |  | 1 |
| set thread context |  | 1 |
| **RegistryActionNameVocab-1.0** | close registry key | 16390 | 310 |
| create registry key | 16390 | 44 |
| delete registry key | 8195 |  |
| enumerate registry key subkeys |  | 5422 |
| enumerate registry key values |  | 373 |
| get registry key attributes |  | 1855 |
| modify registry key value | 8195 | 303 |
| open registry key | 9 | 10947 |
| read registry key value | 7 | 7907 |
| **ServiceActionNameVocab-1.0** | open service | 4 | 6 |
| start service |  | 1 |
| **SocketActionNameVocab-1.0** | get host by name | 1 |  |
| **SynchronizationActionNameVocab-1.0** | create mutex | 1 | 327 |
| open mutex |  | 3 |
|  |  | **49,477** | **37,699** |

Using plotly this data was visualized using a bar graph as detailed in Figure 45 and Figure 46 with the number of action calls on the y-axis and unique calls for each sample on the x-axis.

Chart

Description automatically generated with medium confidence

Figure : Bar graph for Hamweq Sample 1

A picture containing chart

Description automatically generated

Figure : Bar graph for Hamweq Sample 2

Hamweq is a worm and in both samples, we can see the focus on Registry Actions where is establishes an auto start technique and file actions as it propagates through the system (Trend Micro, 2015).

### Zeus samples

The Zeus samples registered 58289, 46870 and 14393 action calls, respectively. The individual actions and their counts for each sample are listed in Table 8.

Table : Zeus Action Calls

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Category** | **Action** | **S1** | **S2** | **S3** |
| **DNSActionNameVocab-1.0** | send dns query | 2 |  |  |
| **DirectoryActionNameVocab-1.0** | create directory |  |  | 1 |
| **FileActionNameVocab-1.0** | copy file |  | 1 | 1 |
| create file | 1292 | 59 | 29 |
| delete file | 1 | 8 | 145 |
| find file | 18 | 57 | 7 |
| get file attributes | 5721 | 83 | 264 |
| move file |  | 1 |  |
| open file | 1274 | 31 | 19 |
| read from file | 8268 | 108 | 17 |
| set file attributes | 7002 | 71 | 11 |
| write to file | 11 | 16 | 4 |
| **GUIActionNameVocab-1.0** | find window | 12258 |  |  |
| **LibraryActionNameVocab-1.0** | get function address | 2365 | 816 | 486 |
| load library | 144 | 106 | 69 |
| **NetworkActionNameVocab-1.0** | connect to socket address | 296 | 42 | 36 |
| **ProcessActionNameVocab-1.0** | create process | 12 | 3 | 1 |
| kill process | 12 | 2 | 2 |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory | 276 | 36 | 45 |
| modify process virtual memory protection |  | 39074 | 99 |
| read from process memory |  | 3197 | 5860 |
| write to process memory | 8 | 69 | 5875 |
| **ProcessThreadActionNameVocab-1.0** | create remote thread in process |  | 1 | 15 |
| create thread | 19 | 7 | 10 |
| get thread context | 2 |  |  |
| set thread context | 2 |  |  |
| **RegistryActionNameVocab-1.0** | close registry key | 5474 | 781 | 126 |
| create registry key | 22 | 42 | 35 |
| delete registry key value |  | 3 | 3 |
| enumerate registry key subkeys | 1911 | 84 | 18 |
| enumerate registry key values | 24 | 154 | 48 |
| get registry key attributes | 2 | 31 | 3 |
| modify registry key value | 15 | 47 | 39 |
| open registry key | 8244 | 1283 | 154 |
| read registry key value | 3479 | 614 | 165 |
| **ServiceActionNameVocab-1.0** | open service | 6 | 6 | 6 |
| send control code to service | 2 |  |  |
| **SocketActionNameVocab-1.0** | get host by name | 125 | 1 |  |
| **SynchronizationActionNameVocab-1.0** | create mutex | 2 | 9 | 10 |
| open mutex |  | 27 | 790 |
|  |  | **58,289** | **46,870** | **14,393** |

Using plotly this data was visualized using a bar graph as detailed in Figure 47, Figure 48 and Figure 49 with the number of action calls on the y-axis and the individual action calls listed on the x-axis.

Chart

Description automatically generated

Figure : Bar Graph for Zeus Sample 1

A picture containing graphical user interface

Description automatically generated

Figure : Bar Graph for Zeus Sample 2

A picture containing chart

Description automatically generated

Figure : Bar Graph for Zeus Sample 3

### CryptoLocker samples

The CryptoLocker samples registered 24784, 43539, 24333, 16107, 31236 and 13,872 action calls, respectively. Samples 1, 3, 4 and 6 reveal a very similar pattern in terms of the action calls performed just variation in the number of calls. Samples 2 and 5 differ greatly with sample 2 activity concentrated in file actions and sample 5 concentrated in registry actions. All calls are listed in Table 9.

Table : CryptoLocker Action Calls

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Category** | **Action** | **S1** | **S2** | **S3** | **S4** | **S5** | **S6** |
| **DNSActionNameVocab-1.0** | send dns query | 2 | 2 | 4 | 4 | 2 | 4 |
| **DirectoryActionNameVocab-1.0** | create directory |  |  |  |  | 7 |  |
| **FileActionNameVocab-1.0** | copy file |  | 1 |  |  |  |  |
| create file |  | 83 |  |  | 337 |  |
| delete file |  |  |  |  | 4 |  |
| find file |  | 34384 |  |  | 162 |  |
| get file attributes |  | 84 |  |  | 141 |  |
| open file |  | 40 |  | 4 | 70 |  |
| read from file |  | 2020 |  |  | 87 |  |
| send control code to file |  |  |  |  | 59 |  |
| set file attributes |  | 4039 |  |  | 370 |  |
| write to file |  | 2600 |  |  | 406 |  |
| **GUIActionNameVocab-1.0** | find window |  |  |  |  | 1 |  |
| **HTTPActionNameVocab-1.0** | send http get request | 1 | 1 | 1 | 2 | 1 | 1 |
| **LibraryActionNameVocab-1.0** | get function address | 19808 | 119 | 19444 | 12850 | 5172 | 11072 |
| load library | 4955 | 5 | 4864 | 3217 | 235 | 2771 |
| **NetworkActionNameVocab-1.0** | connect to socket address | 14 | 17 | 16 | 18 | 16 | 20 |
| **ProcessActionNameVocab-1.0** | create process |  |  |  |  | 4 |  |
| kill process |  |  |  |  | 2 |  |
| **ProcessMemoryActionNameVocab-1.0** | free process virtual memory |  | 17 |  |  | 125 |  |
| modify process virtual memory protection | 4 |  | 4 | 4 | 11 | 4 |
| **ProcessThreadActionNameVocab-1.0** | create thread |  | 11 |  |  | 29 |  |
| **RegistryActionNameVocab-1.0** | close registry key |  | 7 |  |  | 371 |  |
| create registry key |  |  |  |  | 42 |  |
| enumerate registry key subkeys |  | 4 |  |  | 5272 |  |
| enumerate registry key values |  | 42 |  |  | 365 |  |
| get registry key attributes |  | 2 |  | 2 | 1737 |  |
| modify registry key value |  |  |  |  | 212 |  |
| open registry key |  | 31 |  | 6 | 10048 |  |
| read registry key value |  | 30 |  |  | 5615 |  |
| **ServiceActionNameVocab-1.0** | open service |  |  |  |  | 6 |  |
| start service |  |  |  |  | 1 |  |
| **SynchronizationActionNameVocab-1.0** | create mutex |  |  |  |  | 323 |  |
| open mutex |  |  |  |  | 3 |  |
|  | | **24,784** | **43,539** | **24,333** | **16,107** | **31,236** | **13,872** |

Using plotly this data was visualized using a bar graphs as detailed in Figure 50, Figure 51, Figure 52, Figure 53, Figure 54 and Figure 55 with the number of action calls on the y-axis and unique calls for each sample on the x-axis.

Chart, waterfall chart

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 1

A picture containing graphical user interface

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 2

Chart, waterfall chart

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 3

Chart, waterfall chart

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 4

A picture containing chart

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 5

Chart, waterfall chart

Description automatically generated

Figure : Bar Graph for CryptoLocker Sample 6

While the bar graphs make it visually more appealing to read the information from the table is does not clearly show the similarities or differences between the individual samples, to display this information a heatmap was generated using seaborn.

To perform this task the Transposed DataFrame was utilized – firstly the column names were split and cast to multi-index columns.

Table

Description automatically generated with medium confidence

Figure : Reorder DataFrame Columns making Mult-index columns

Next the DataFrame was transposed making the new multi-index columns the index for the DataFrame.

Graphical user interface, text

Description automatically generated

Figure : Transpose DataFrame

Using this transposed DataFrame the data was correlated and a heatmap generated using seaborn to view the similarities or not between samples.

Graphical user interface, text

Description automatically generated

Figure : Code to generate heatmap

Chart, treemap chart

Description automatically generated

Figure : Heatmap for CryptoLocker Spiked Samples

By transposing the DataFrame we are comparing the full sample to each other sample. We read the graph by comparing the sample number from the x-axis with a sample number from the y-axis, the colour of the square at their intersection represents the similarity or not between samples. In this example the yellow square on the diagonal is the perfect match as it compares the sample to itself. The scale of this heatmap runs from +1 to -.2, with a value of +1 indicated by the yellow colour indicating a positive correlation between 3 of the samples, 1 light green coloured sample indicating a reading of approximately .8 also indicating a strong positive correlation and 2 samples indicated in teal and purple with reading of approximately +.2 and -.2 which indicates a reading close to 0 showing almost no linear trend between 2 of the samples. This is in keeping with the figures identified in Table 9 by the dispersion of action calls.

### Spiked samples

Taking the 13 spiked samples collectively and generating a new DataFrame for this information an interactive plotly line graph was generated to display the action calls of each individual sample as shown in Figure 60.

Chart, line chart

Description automatically generated

Figure : Plotly Line Graph for Spiked Samples

This when compared to the area graph generated to display the same information which is shown in Figure 61, is quite difficult to interrupt, in this instance due to the smaller sample size and smaller range in values the area graph is much easier to read.

Chart

Description automatically generated

Figure : Area Graph for Spiked Samples

Correlating the data in these samples by transposing the newly created DataFrame enabled the generation of a heatmap Figure 62, which unlike the CryptoLocker Samples in Figure 59 shows a greater disparity among the 13 samples representing the difference in malware sample families. The scale on this heatmap ranges from +1 to –.4 The teal colour represents a reading of 0 in this graph showing no linear trend between these samples and the purple colour a reading of -.4 indicating these samples are starting to show a negative correlation meaning that as the calls in other samples start to increase these start to decrease, and vice versa.

A picture containing square

Description automatically generated

Figure : Heatmap for Spiked Samples

## Malware Families Analysed

There were variants from 8 different malware sample families included in the dataset, each was examined independently to identify commonalties in actions among the variants. These samples have previously been identified in Table 1.

### Select Malware Samples

To perform the analysis all the variants from the respective samples were placed in separate folders and selected in a similar manner to the full dataset. In Figure 63 below the folder with ‘CryptoLocker’ samples was selected and files counted detailing 96 samples. Each of these files were read line by line and the action calls extracted, counted, and added to the overall ‘data’ list for all samples as for the full dataset.

A picture containing text

Description automatically generated

Figure : Code to select file path to malware folder

### Analysis of Samples

#### CryptoLocker Samples

With the relevant data extracted this list was used to create the Pandas DataFrame as before.

Text, email

Description automatically generated

Figure : Create DataFrame & Extract of output

To visualise the action calls for each sample plotly was used, as it provided an interactive graph and data points for the sample can be compared by hovering over a sample as shown in Figure 65.

Graphical user interface, application, Word

Description automatically generated

Figure : Interactive Plotly Line Graph

Chart

Description automatically generated

Figure : CryptoLocker Plotly Line Graph

To see the correlation between all the CryptoLocker Samples the Seaborn heatmap, used with the spiked CryptoLocker samples and 13 spiked samples was again utilized. This was done in the same fashion making the columns multi-index before transposing the dataset. The resultant heatmap is shown in Figure 67.

Schematic

Description automatically generated

Figure : CryptoLocker Heatmap

From this visualization we can see that the majority of the heatmap is coloured in yellow and a small number of the samples showing the dark green and purple colours. This is a much faster way of comparing samples than trying to read data from a table or DataFrame. The yellow colour again represents a positive correlation between samples and the teal colour, a 0-reading indicating no linear trend and purple colour a move towards a negative correlation.

The same process was completed for each of the sample families

#### Zeus Samples

The Zeus line graph shows a small number of spiking samples with most of the samples registering small counts for their action calls is it quite difficult to compare samples.

Chart

Description automatically generated

Figure : Zeus Line Graph

On the other hand, the use of the heatmap makes the comparison process much more effective simply identifying colour variation among the samples. This map scale runs from +1 to -1 the yellow and light green colours show positive correlation between samples and the dark green to real no linear trend and a few dark purple samples indicating a negative correlation.

Chart

Description automatically generated

Figure : Zeus Heatmap

#### Reveton Samples

The Reveton Samples reveal an almost uniform spiking pattern in the line graph, but they too reveal an unreadable section clustered along the x-axis.

Chart

Description automatically generated

Figure : Reveton Samples Line Graph

The heatmap for the Reveton Samples shows a greater concentration of the green colour than seen for the Zeus samples, in this map the scale goes from +1 to -.2, indicating that the majority of samples demonstrate positive correlation, the teal and purple colours in this case indicate no linear trend.

Chart, schematic

Description automatically generated

Figure : Reveton Heatmap

#### Swizzor Samples

The large spiking sample obscures the readings for the Swizzor samples from the line graph and so comparison of these samples can only be made through the heatmap.

Table

Description automatically generated with medium confidence

Figure : Swizzor Samples Line Graph

The heatmap again shows a deep concentration of yellow identifying the positive correlation among many samples. We do see the presence of deep purple to an extent identifying a move toward a negative correlation in some samples as the scale for this map reads from +1 to -.4.

Diagram, schematic

Description automatically generated

Figure : Swizzor Heatmap

#### Hamweq Samples

As with the previous Swizzor Samples the spiking variations in the Hamweq samples make comparison of samples through the heatmap alone.

Chart

Description automatically generated

Figure : Hamweq Line graph

The heatmap shows a very deep concentration of green all the way through to the deep purple revealing the greatest variation among samples so far. The scale of this map read from +1 to -1 the yellow and green colours in this instance a positive correlation between samples, the teal no linear trend and dark blue and purple a negative correlation. The white cells indicate samples which do not have enough data to correlate.

Chart

Description automatically generated

Figure : Hamweq Heatmap

#### Ghostrat Samples

The Ghostrat Sample line graph shows a few small spikes ranging to larger spikes in the action calls, which this would indicate a possible disparity among samples.

Chart

Description automatically generated with medium confidence

Figure : Ghostrat Line Graph

The heatmap scale ranges from +1 to -.2 most samples have a yellow or light green colour indicating a positive correlation, while the small number of purple indicates no linear trend moving to a slightly negative correlation.

A picture containing text, building

Description automatically generated

Figure : Ghostrat Heatmap

#### Vanti Samples

The Vanti samples reveal an interesting pattern in the line graph completely different to the other samples. While this shows noticeable differences between the individual action calls in each individual sample the heatmap displays an almost identical pattern identified for each of the samples showing positive correlation between almost all samples.

Chart

Description automatically generated

Figure : Vanti Line Graph

Chart

Description automatically generated

Figure : Vanti Heatmap

#### Anubis Samples

The Anubis line plot reveals a small number of spiking actions with the vast number of individual calls per sample clustered in the lower numbers along the x-axis. Gaps are also identified in the samples indicating the presence of a very low number of action calls in samples making them appear as almost nil.

Chart

Description automatically generated

Figure : Anubis Samples Line Graph

The low numbers are also identified in the heatmap by the presence of many white squares indicating non-comparable samples. The scale goes from +1 to -1 so the yellow and green colours represent positive correlation between samples.

Chart, diagram

Description automatically generated

Figure : Anubis Heatmap

## Full Dataset

While the line graphs proved to be non-decipherable for the full dataset, the heatmaps have proved to be a better indicator for comparing the similarities and differences among samples. The DataFrame for the full dataset was again utilised, this time to generate a heatmap.

A picture containing chart

Description automatically generated

Figure : Heatmap for Full Dataset

While the squares in this heatmap are too small to compare individual samples, the overall effect can be described as an almost green square punctuated by strands of yellow, dark green and purple. The scale of this map reads from +1 to -1, therefore the yellow and light green shades indicate most samples show a positive correlation. This would prove difficult to compare new samples against so best results would be achieved by comparing any new samples to the individual malware sample families as in 5.2.

## 3D Visualization

3D visualization of the malware samples in this research allows us to view the action calls in 3 planes, the parameters modelled are the samples, the action categories observed and the total count for all the individual actions belonging to each category per sample. The full dataset was modelled by calculating the total for each category by summating all the individual actions under each of the category headings. Using the earlier ‘malcat.csv’ file the individual actions were removed and in place just a total per sample for the category. A DataFrame was created by reading the data from this new .csv file, ‘catfile.csv’, with categories as the index, and each sample being represented by a column and values for each sample under each category heading as shown in Figure 83.

Text

Description automatically generated

Figure 83: Category DataFrame for Full Dataset

Using this DataFrame and the Plotly Graph Objects, the z-axis was assigned the values of the DataFrame, the samples were represented by index number on the x-axis and categories by index number of the y-axis.

Text

Description automatically generated

Figure 84: Code to generate 3D Surface Graph

The following is the 3D visualization of the full dataset.

A picture containing text, stationary, pencil

Description automatically generated

Figure 85: 3D Surface Graph for Full Dataset by Category

Subsequent 3D visualizations were generated for each of the Sample families in the same fashion as for the full dataset and the resultant visualizations are outlined in the following images.

Table 10: CryptoLocker Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *DNSActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HTTPActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated with medium confidence

Figure 86: CryptoLocker 3D Surface Graph

Table 11: Ghostrat Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *DNSActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HTTPActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated

Figure 87: Ghostrat 3D Surface Graph

Table 12: Hamweq Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12 | *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated

Figure 88: Hamweq 3D Surface Graph

Table 13: Swizzor Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12 | *DNSActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Graphical user interface, chart

Description automatically generated

Figure 89: Swizzor 3D Surface Graph

Table 14: Vanti Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9 | *DNSActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *HTTPActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated

Figure 90: Vanti 3D Surface Graph

Table 15: Anubis Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9 | *DeviceDriverActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *IPCActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated

Figure 91: Anubis 3D Surface Graph

Table 16: Zeus Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *DNSActionNameVocab-1.0*  *DebuggingActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated

Figure 92: Zeus 3D Surface Graph

Table 17: Reveton Samples Category Index

|  |  |
| --- | --- |
| **Category Index** | **Category** |
| 0  1  2  3  4  5  6  7  8  9  10  11  12  13  14 | *DNSActionNameVocab-1.0*  *DirectoryActionNameVocab-1.0*  *FileActionNameVocab-1.0*  *GUIActionNameVocab-1.0*  *HTTPActionNameVocab-1.0*  *HookingActionNameVocab-1.0*  *LibraryActionNameVocab-1.0*  *NetworkActionNameVocab-1.0*  *ProcessActionNameVocab-1.0*  *ProcessMemoryActionNameVocab-1.0*  *ProcessThreadActionNameVocab-1.0*  *RegistryActionNameVocab-1.0*  *ServiceActionNameVocab-1.0*  *SocketActionNameVocab-1.0*  *SynchronizationActionNameVocab-1.0* |

Chart

Description automatically generated with medium confidence

Figure 93: Reveton 3D Surface Graph

There are vast differences between the spiking patterns observed for each of the 8 sample families and indeed among the individual samples in each group. The 3D visualizations allow rotation of the graph on screen and the ability to zoom in on a particular aspect of each graph. This method allows the user more control than interactive line plots and a 3D rendering aids the visualization process of comparing individual samples and families.

## Evaluation

To evaluate the performance of the visualizations generated each type was examined individually and quantified against the criteria proposed by Freitas *et al.* (2002), namely cognitive complexity, spatial organisation, information coding, and limitations as well as the presence of basic elements. State Transition is evaluated generally as no action was taken to modify any of the individual graphs, but graphing techniques were applied with different parameters. The time taken to generate the visualizations was almost instantaneous in all cases with visualizations using the full dataset taking just a fraction longer. The effect on spatial organization is linked to the volume of data being modelled as opposed to the technique.

### Evaluation of Area Graphs

* **Basic Elements:** the basic elements are present, sample numbers are represented on the x-axis, and the number of action calls on the y-axis, which have both been labelled.
* **Cognitive Complexity:** while the data was relevant, the volume of data points and the number of action categories modelled hindered clear identification except for the area graph generated on the modified dataset which excluded the spiking samples.
* **Spatial Organization:** due to the volume and variation of data in the graph, it was almost impossible to identify data from individual samples especially where there was a low number of action calls.
* **Information Coding:** the mapping of action categories using colour was not clear due to the volume of samples
* **Limitations:** the size of the display was not adequate for the number of data elements.

### Evaluation of Bar Charts

* **Basic Elements:** the individual actions are clearly labelled on the x-axis and number of action calls on the y-axis.
* **Cognitive Complexity:** while the data was relevant, the number of data points varied between graphs but overall was not impacted.
* **Spatial Organization:** the spacing of bars allows for identification of individual actions easily, action calls in the upper regions are easily identifiable while in the lower regions are estimable.
* **Information Coding:** as the bar graphs only demonstrate the calls in a single sample this was not relevant**.**
* **Limitations:** the size of the display was adequate for the number of data elements.

### Evaluation of Line Graphs

* **Basic Elements:** the basic elements are present, sample numbers are represented on the x-axis, and the number of action calls on the y-axis, which have both been labelled.
* **Cognitive Complexity:** while the data was relevant, the volume of data points and the number of action categories modelled hindered clear identification.
* **Spatial Organization:** due to the volume and variation of data in the graph, it was difficult to identify data from individual samples especially where there was a low number of action calls.
* **Information Coding:** as plotly graphs are interactive the incorporation of the hover function facilitates the identification of individual samples online but not in the printed graphs.
* **Limitations:** the size of the display was not adequate for the number of data elements.

### Evaluation of Heatmaps

* **Basic Elements:** the basic elements are present, sample numbers are represented on the x-axis and y-axis, which have both been labelled.
* **Cognitive Complexity:** data points are not modelled individually just the correlation between samples, the smaller groupings of samples work better than the denser groupings
* **Spatial Organization:** the smaller sample groupings allow individual samples to be compared to each other, the larger groupings do not allow individual analysis, however it is still possible to establish the overall result for the family by analysing the general colouring of the map
* **Information Coding:** by displaying the difference between the positive and negative correlation of samples using colour it is easy to establish the results
* **Limitations:** while there were some limitations due to the size of the display for the larger sample groupings the overall trend amongst the samples is still decipherable.

### Evaluation of 3D Visualization

* **Basic Elements:** the basic elements are clearly labelled on 3 axes, Samples represented by number of the x-axis, the action categories again identified by number taken from DataFrame index on the y-axis and the count for the action category on the z-axis.
* **Cognitive Complexity:** while the data points are dense the 3D by modelling the action categories rather than the individual actions makes the visualization clearer.
* **Spatial Organization:** the volume and variation of data in the graph was easier to identify due to the 3D aspect.
* **Information Coding:** summating the total for each action category for each of the malware samples made for clearer referencing without obscuring the results by visualizing each individual action. The interactive, rotating and zoom functions available online allow closer inspection of data, while the integration of colour identifies the action call count clearly.
* **Limitations:** the size of the display was adequate for the number of data elements, but the online zoom function makes the interactive graphing more comprehensible.

# Conclusion

## Overview

On concluding this research, it was possible to extract action calls from the dynamic malware analysis MAEC reports generated by Cuckoo Sandbox using the MAEC language components. These actions were subsequently analysed and used to generate visualizations in the form of area graphs, interactive line plots, bar graphs, heatmaps, and 3D surface graphs to illustrate the actions performed by the malware and the comparison between samples.

The visualizations generated illustrated the differences and similarities between malware samples and variants, and the actions they performed once they infected the system. The MAEC schema provides the malware action open vocabulary which was used to extract the actions modelled in this research. From implementing the visualizations using the python graphing libraries the process of comparing samples and analysing the calls is both hastened and aesthetically more pleasing than trawling through report files which can be considerably lengthy.

## Conclusions

The built-in Pandas visualization area graphs are quite hard to read due to the size of the dataset and the variance in action calls between samples, by utilising the Plotly interactive line graphs this made it easier to isolate individual samples and read the data using the hover function.

The bar graphs summate the total for each of the individual actions in a single sample and so perform quite well in identifying the actions performed however the count is clearly visible in the upper ranges the lower regions are estimable.

Heatmaps proved a very informative way of comparing the correlation between samples simply by viewing the colour changes, however it does not visualize the individual actions performed or the categories they belong to.

The 3D surface graphs improved upon the interactive line graphs providing a rendering of the samples in 3 planes which could be rotated and zoomed in on to more closely inspect samples. By concentrating on the action categories rather than the individual actions and overall picture for each sample both in the full dataset and in each of the respective sample family’s is decipherable even amongst the denser datasets.

Due to the complexity and volume of data involved in the malware investigations a more complex visualisation method proved to be the better performing option. While investigating single samples a simple bar graph performs quite well, the area graphs underperform when evaluating data with large volumes and disparity. The interactive nature of the plotly line graphs compensates for failings while dealing with the large data volumes but this does not translate to the printed medium.

## Further Work

During this research Cuckoo Sandbox was setup and configured on an Ubuntu 18.04.5 virtual machine, however integrating a Windows 10 Virtual machine to execute current malware samples proved difficult. Due to time restrictions this had to be cast to one side, however it would be nice to execute current malware samples for analysis.

Online Sandboxing options would be a more accessible option for most users, providing a .JSON or .xml report of the dynamic analysis by uploading a file or file hash. Joe Sandbox Cloud Pro, an online sandboxing option, offer MAEC formatted analysis reports, but it is only part of the Windows or Ultimate paid packages (Joe Security LLC, 2021). A more realistic option for further work would be to develop a script to convert the online .JSON or .xml analysis reports to MAEC format.

Regarding visualization techniques this research focused purely on the actions performed by malware not on the order of execution. In the future an investigation creating a visualized timeline of the execution of malware using the MAEC-format reports would be considered.

# References

2U, Inc, 2020. *How is Python Used for Data Science?.* [Online]   
Available at: https://www.mastersindatascience.org/data-scientist-skills/python/  
[Accessed 27 April 2021].

Abraham, S., 2019. *What is Fileless Malware and How to Protect Yourself?.* [Online]   
Available at: https://www.malwarefox.com/fileless-malware/  
[Accessed 27 February 2021].

Analytics Vidhya, 2020. *Creating Interactive Visualizations using Plotly in Python.* [Online]   
Available at: https://www.analyticsvidhya.com/blog/2021/04/creating-interactive-visualizations-using-plotly-in-python/#:~:text=Plotly%20is%20an%20open-source%20data%20visualization%20library%20to,area%20plot%2C%20histogram%2C%20box%20plot%2C%20bar%20plot%2C%20etc.  
[Accessed 25 April 2021].

Anderson, B., Storlie, C. & Lane, T., 2012. *Improving Malware Classification: Bridging the Static/Dynamic Gap.* New York, Proceedings of the 5th ACM workshop on Security and artificial intelligence (AISec '12), pp. 3-14.

Aru, O. E. & Chiaghana Chukwunonso, E., 2018. Malware Analysis and Mitigation in Information Preservation. *IOSR Journal of Applied Physics (IOSR-JAP),* 10(4 Ver. II), pp. 7-16.

AVG, 2020. *Everything You Need to Know About Rootkits and How to Protect Yourself.* [Online]   
Available at: https://www.avg.com/en/signal/what-is-rootkit#topic-4  
[Accessed 31 March 2021].

AVG, 2020. *What Is Malware? How Malware Works & How to Prevent It.* [Online]   
Available at: https://www.avg.com/en/signal/what-is-malware  
[Accessed 28 March 2021].

AVG, 2020. *What Is Malware? How Malware Works & How to Prevent It.* [Online]   
Available at: https://www.avg.com/en/signal/what-is-malware#:~:text=The%20following%20is%20a%20list%20of%20common%20types,to%20spy%20on%20you.%20...%20More%20items...%20  
[Accessed 31 March 2021].

Batt, S., 2016. *3 Malware Distribution Methods You Really Need to Beware Of.* [Online]   
Available at: https://www.maketecheasier.com/malware-distribution-methods-you-to-know/  
[Accessed 03 April 2021].

Beck, D., Kirillov, I., Chase, P. & MITRE, 2014. *THE MAEC™ LANGUAGE.* [Online]   
Available at: https://maecproject.github.io/documentation/overview/MAEC\_Overview.pdf  
[Accessed 27 February 2021].

Crowdstrike, 2021. *What is Mobile Malware?.* [Online]   
Available at: https://www.crowdstrike.com/cybersecurity-101/malware/mobile-malware/  
[Accessed 31 March 2021].

Crowdstrike, 2021. *What is Ransomware?.* [Online]   
Available at: ttps://www.crowdstrike.com/cybersecurity-101/ransomware/  
[Accessed 31 March 2021].

Cuckoo, 2019. *Cuckoo Automated Malware Analysis.* [Online]   
Available at: https://cuckoosandbox.org/  
[Accessed 17 March 2021].

Cyware Labs, 2019. *Understanding the threats and risk of USB Flash Drive Malware.* [Online]   
Available at: https://cyware.com/news/understanding-the-threats-and-risk-of-usb-flash-drive-malware-024ef150  
[Accessed 03 April 2021].

Cyware, 2019. *What is Malware Attribute Enumeration and Characterization (MAEC)?.* [Online]   
Available at: https://cyware.com/educational-guides/cyber-threat-intelligence/what-is-malware-attribute-enumeration-and-characterization-maec-81e2  
[Accessed 27 February 2021].

Dragulescu, A., 2008. *Alex Dragulescu.* [Online]   
Available at: https://sq.ro/malwarez.htm  
[Accessed 4 May 2021].

Efe, A. & Hussin, S., 2020. Malware Visualization Techniques. *International Journal of Applied Mathematics, Electronics and Computers,* 8(1).

FBI, 2012. *FBI, This Week: Reveton Ransomware.* [Online]   
Available at: https://www.fbi.gov/audio-repository/news-podcasts-thisweek-reveton-ransomware/view  
[Accessed 03 April 2021].

Freitas, C. M. S. S. et al., 2002. *Evaluating Usability of Information Visualization,* s.l.: Brazilian Symposium on Human Factors in Computing Systems.

GitHub Inc., 2015. *ptrac3/MAEC-Dataset.* [Online]   
Available at: https://github.com/ptrac3/MAEC-Dataset  
[Accessed 8 March 2021].

GitHub, Inc., 2014. *MAEC Project/datasets.* [Online]   
Available at: https://github.com/MAECProject/datasets  
[Accessed 8 March 2021].

Grégio, A. R. A. & Santos, R. D. C., 2011. Visualization Techniques for Malware Behavior Analysis. *Sensors, and Command, Control, Communications and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense X,* 8019(International Society for Optics and Photonics), p. 801905.

Grégio, A. R. et al., 2011. *Behavioral analysis of malicious code through network traffic and system call monitoring.* [Online]   
Available at: https://core.ac.uk/download/pdf/296636308.pdf  
[Accessed 6 March 2021].

Holloway, M., 2015. *Stuxnet Worm Attack on Iranian Nuclear Facilities,* Stanford: Stanford University.

IDG Communications, 2021. *Five new threats to your mobile security.* [Online]   
Available at: https://www.csoonline.com/article/2157785/five-new-threats-to-your-mobile-security.html  
[Accessed 31 March 2021].

Infosec, 2015. *Gh0st RAT: Complete Malware Analysis – Part 1.* [Online]   
Available at: https://resources.infosecinstitute.com/topic/gh0st-rat-complete-malware-analysis-part-1/  
[Accessed 03 April 2021].

Infosec, 2019. *Malware spotlight: What are backdoors?.* [Online]   
Available at: https://resources.infosecinstitute.com/topic/malware-spotlight-what-are-backdoors/#:~:text=%20Real-world%20examples%20of%20backdoors%20%201%20Back,worm-like%20Trojan%20eventually%20became%20a%20full...%20More%20  
[Accessed 31 March 2021].

IntelliPaat, 2020. *What is Seaborn in Python.* [Online]   
Available at: https://intellipaat.com/blog/what-is-seaborn-in-python/#:~:text=Seaborn%20is%20one%20of%20the%20world%E2%80%99s%20most%20regarded,and%20this%20is%20how%20the%20workflow%20looks%20like%3A  
[Accessed 27 April 2021].

Investopedia, 2021. *Bar Graph.* [Online]   
Available at: https://www.investopedia.com/terms/b/bar-graph.asp  
[Accessed 13 May 2021].

Joe Security LLC, 2021. *Joe Sandbox Cloud.* [Online]   
Available at: https://joesecurity.org/joe-sandbox-cloud#accounts  
[Accessed 10 April 2021].

Kaspersky , 2021. *What are the different types of ransomware?.* [Online]   
Available at: https://www.kaspersky.com/resource-center/threats/ransomware-examples  
[Accessed 31 March 2021].

Kaspersky, 2021. *Zeus Virus.* [Online]   
Available at: https://usa.kaspersky.com/resource-center/threats/zeus-virus  
[Accessed 03 April 2021].

Kirillov, I., Chase, P., Beck, D. & Martin, R., 2011. *Malware Attribute Enumeration and Characterization.* [Online]   
Available at: https://www.researchgate.net/publication/267691330\_Malware\_Attribute\_Enumeration\_and\_Characterization  
[Accessed 27 February 2021].

Malwarebytes, 2021. *Spyware.* [Online]   
Available at: https://www.malwarebytes.com/spyware/  
[Accessed 31 March 2021].

Malwarebytes, 2021. *State of Malware Report.* [Online]   
Available at: https://resources.malwarebytes.com/files/2021/02/MWB\_StateOfMalwareReport2021.pdf  
[Accessed 19 April 2021].

McAfee, 2013. *What is a Keylogger?.* [Online]   
Available at: https://www.mcafee.com/blogs/consumer/family-safety/what-is-a-keylogger/  
[Accessed 31 March 2021].

McAfee, 2021. *What Is Fileless Malware?.* [Online]   
Available at: https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/what-is-fileless-malware.html  
[Accessed 31 March 2021].

McAfee, 2021. *What is malware?.* [Online]   
Available at: https://www.mcafee.com/en-us/antivirus/malware.html  
[Accessed 04 March 2021].

Medium, 2018. *Data Visualization with Python and Seaborn — Part 1: Loading Datasets.* [Online]   
Available at: https://medium.com/@neuralnets/data-visualization-with-python-and-seaborn-part-1-29c9478a8700  
[Accessed 27 April 2021].

Microsoft, 2007. *TrojanDownloader:Win32/Swizzor.gen.* [Online]   
Available at: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=TrojanDownloader:Win32/Swizzor.gen  
[Accessed 03 April 2021].

Microsoft, 2017. *Worm:Win32/Hamweq.DQ.* [Online]   
Available at: https://www.microsoft.com/en-us/wdsi/threats/malware-encyclopedia-description?Name=Worm:Win32/Hamweq.DQ  
[Accessed 03 April 2021].

MITRE Corporation, n.d. *MAEC A Structured Language for Attribute-Based Malware Characterization.* [Online]   
Available at: https://makingsecuritymeasurable.mitre.org/docs/maec-intro-handout.pdf  
[Accessed 27 February 2021].

MITRE, 2015. *python-cybox.* [Online]   
Available at: https://cybox.readthedocs.io/en/stable/examples.html  
[Accessed 10 February 2021].

MITRE, 2016. *OVAL Open Vulnerability and Assessment Language.* [Online]   
Available at: https://oval.mitre.org/  
[Accessed 03 April 2021].

MITRE, 2017. *MAEC 5.0 Specification Vocabularies.* [Online]   
Available at: https://maecproject.github.io/releases/5.0/MAEC\_Vocabularies\_Specification.pdf  
[Accessed 28 February 2021].

MITRE, 2020. *About CybOX (Archive).* [Online]   
Available at: https://cyboxproject.github.io/about/  
[Accessed 8 May 2021].

MITRE, 2020. *About MAEC.* [Online]   
Available at: https://maecproject.github.io/about-maec/  
[Accessed 28 March 2021].

MITRE, 2020. *AV Classification Type.* [Online]   
Available at: http://maecproject.github.io/data-model/4.1/maecBundle/AVClassificationType/  
[Accessed 03 April 2021].

MITRE, 2020. *MAEC Overview.* [Online]   
Available at: http://maec.mitre.org/about/docs/Introduction to MAEC white paper.pdf  
[Accessed 27 February 2021].

MITRE, 2020. *Main schema maec\_default\_vocabularies.xsd.* [Online]   
Available at: https://maecproject.github.io/documentation/maec4.1-xsddocs/default\_vocabularies-1.html  
[Accessed 18 March 2021].

MITRE, 2020. *Malware Visualization.* [Online]   
Available at: https://maecproject.github.io/documentation/use\_cases/malware\_analysis/malware\_visualization/  
[Accessed 28 April 2021].

MITRE, 2020. *Overview of the MAEC Bundle Data Model for MAEC Version 4.1.* [Online]   
Available at: http://maecproject.github.io/documentation/data\_model\_overview/bundle/  
[Accessed 03 April 2021].

MITRE, 2020. *Regular Expression Support.* [Online]   
Available at: https://cyboxproject.github.io/documentation/regex-support/  
[Accessed 6 April 2021].

MITRE, 2021. *CAPEC Common Attack Pattern Enumeration and Classification.* [Online]   
Available at: https://capec.mitre.org/  
[Accessed 03 April 2021].

MITRE, 2021. *CVE.* [Online]   
Available at: https://cve.mitre.org/  
[Accessed 03 April 2021].

MITRE, 2021. *CWE Common Weakness Enumeration.* [Online]   
Available at: https://cwe.mitre.org/  
[Accessed 03 April 2021].

Nari, S. & Ghorbani, A. A., 2013. Automated malware classification based on network behavior. *International Conference on Computing, Networking and Communications (ICNC),* January.pp. 642-647.

National Institute of Standards and Technology, 2017. *NIST Special Publication 800-12 Revision 1 An Introduction to Information Security.* [Online]   
Available at: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-12r1.pdf  
[Accessed 28 March 2021].

National Institute of Standards and Technology, 2020. *Common Configuration Enumeration (CCE).* [Online]   
Available at: https://nvd.nist.gov/config/cce  
[Accessed 03 April 2021].

National Institute of Standards and Technology, 2020. *NIST SPECIAL PUBLICATION 1800-25A, Data Integrity Identifying and Protecting Assets Against Ransomware and Other Destructive Events.* [Online]   
Available at: https://www.nccoe.nist.gov/sites/default/files/library/sp1800/di-identity-protect-nist-sp1800-25a-draft.pdf  
[Accessed 28 March 2021].

National Institute of Standards and Technology, 2021. *Official Common Platform Enumeration (CPE) Dictionary.* [Online]   
Available at: https://nvd.nist.gov/products/cpe  
[Accessed 03 April 2021].

Nord Pass, 2021. *What Is a Trojan Horse Virus? Detection, Examples, and Remedies.* [Online]   
Available at: https://nordpass.com/blog/what-is-trojan-horse-virus/  
[Accessed 31 March 2021].

Norton Life Lock, 2021. *How to Protect Yourself From Botnets.* [Online]   
Available at: https://us.norton.com/internetsecurity-malware-what-is-a-botnet.html  
[Accessed 31 March 2021].

Norton Life Lock, 2021. *The 8 Most Famous Computer Viruses of All Time.* [Online]   
Available at: https://uk.norton.com/norton-blog/2016/02/the\_8\_most\_famousco.html  
[Accessed 31 March 2021].

Norton Life Lock, 2021. *What is a computer worm, and how does it work?.* [Online]   
Available at: https://us.norton.com/internetsecurity-malware-what-is-a-computer-worm.html#:~:text=A%20computer%20worm%20is%20a%20type%20of%20malware,damage.%20Worms%20can%20be%20transmitted%20via%20software%20vulnerabilities.  
[Accessed 31 March 2021].

OASIS Open, 2021. *Introduction to STIX.* [Online]   
Available at: https://oasis-open.github.io/cti-documentation/stix/intro  
[Accessed 28 February 2021].

OASIS Open, 2021. *STIX Version 2.1.* [Online]   
Available at: https://docs.oasis-open.org/cti/stix/v2.1/cs02/stix-v2.1-cs02.docx  
[Accessed 04 May 2021].

OASIS Open, 2021. *Visualized SDO Relationships.* [Online]   
Available at: https://oasis-open.github.io/cti-documentation/examples/visualized-sdo-relationships  
[Accessed 28 February 2021].

OmniSecu.com, 2021. *What is a DNS Query, How DNS Query Works.* [Online]   
Available at: https://www.omnisecu.com/tcpip/what-is-a-dns-query-how-dns-query-works.php#:~:text=A%20DNS%20Query%20is%20a%20request%20for%20information,related%20with%20a%20Fully%20Qualified%20Domain%20Name%20%28FQDN%29.  
[Accessed 5 May 2021].

Plotly, 2021. *Plotly Python Open Source Graphing Library.* [Online]   
Available at: https://plotly.com/python/  
[Accessed 27 April 2021].

Shalaginov, A., Banin, S., Dehghantanha, A. & Franke, K., 2018. Machine Learning Aided Static Malware Analysis: A Survey and Tutorial. In: A. Dehghantananha, M. Conti & T. Dargahi, eds. *Cyber Threat Intelligence.* s.l.:Springer, Cham, pp. 7-45.

Software Lab , 2021. *What is a Computer Worm?.* [Online]   
Available at: https://softwarelab.org/what-is-a-computer-worm/  
[Accessed 31 March 2021].

Software Lab, 2021. *What is Spyware?.* [Online]   
Available at: https://softwarelab.org/what-is-spyware/#:~:text=Some%20of%20the%20best-known%20examples%20of%20spyware%20include,and%20login%20pages%20with%20advertisements.%20More%20items...%20  
[Accessed 31 March 2021].

Sophos Ltd., 2013. *Threatsaurus The A-Z of computer and data security threats.* [Online]   
Available at: https://www.sophos.com/en-us/medialibrary/PDFs/other/sophosthreatsaurusaz.pdf?la=en  
[Accessed 27 February 2021].

SourceForge, 2021. *JUNG Java Universal Network/Graph Framework.* [Online]   
Available at: http://jung.sourceforge.net/  
[Accessed 6 March 2021].

Spam Laws, 2021. *The History of Worms.* [Online]   
Available at: https://spamlaws.com/history-of-worms.html  
[Accessed 31 March 2021].

Stack Exchange, 2019. *How can one interpret a heat map plot.* [Online]   
Available at: https://stats.stackexchange.com/questions/392517/how-can-one-interpret-a-heat-map-plot  
[Accessed 1 May 2021].

Swanson, I., 2008. *Malware, Viruses and Log Visualization.* Perth, Edith Cowan University.

Swinhoe, D., 2019. *What is spear phishing? Why targeted email attacks are so difficult to stop.* [Online]   
Available at: https://www.csoonline.com/article/3334617/what-is-spear-phishing-why-targeted-email-attacks-are-so-difficult-to-stop.html  
[Accessed 03 April 2021].

Techopedia, 2021. *What Does Heat Map Mean?.* [Online]   
Available at: https://www.techopedia.com/definition/32150/heat-map#:~:text=What%20Does%20Heat%20Map%20Mean%3F%20A%20heat%20map,such%20as%20defense%2C%20marketing%20and%20understanding%20consumer%20behavior.  
[Accessed 28 April 2021].

The Honeynet Project, 2012. *HPFEEDSHONEYGRAPH – AUTOMATED ATTACK GRAPH CONSTRUCTION FOR HPFEEDS LOGS.* [Online]   
Available at: https://www.honeynet.org/2012/09/11/hpfeedshoneygraph-automated-attack-graph-construction-for-hpfeeds-logs/  
[Accessed 28 April 2021].

ThoughtCo, 2019. *What Is a Histogram?.* [Online]   
Available at: https://www.thoughtco.com/what-is-a-histogram-3126359  
[Accessed 4 May 2021].

Trend Micro, 2012. *Ghostrat.* [Online]   
Available at: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/GHOSTRAT  
[Accessed 5 May 2021].

Trend Micro, 2012. *Troj\_Swizzor.* [Online]   
Available at: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/TROJ\_SWIZZOR/  
[Accessed 05 May 2021].

Trend Micro, 2015. *Worm\_Hamweq.aij.* [Online]   
Available at: https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/WORM\_HAMWEQ.AIJ/  
[Accessed 5 May 2021].

Trinius, P., Holz, T., Gobel, J. & Freiling, F. C., 2009. *Visual Analysis of Malware Behavior Using Treemaps and Thread Graphs.* s.l., 6th International Workshop on Visualization for Cyber Security.

University of Delaware, 2015. *How is Malware Distributed?.* [Online]   
Available at: https://sites.udel.edu/infosecnews/2015/05/18/how-is-malware-distributed/  
[Accessed 03 April 2021].

Windows Central, 2017. *7 tips to keep your Windows PC protected against malware.* [Online]   
Available at: https://www.windowscentral.com/how-to-keep-your-windows-pc-protected-against-malware#:~:text=Your%20recovery%20plan%20must%20include%20a%20full%20backup,any%20malware%2C%20hardware%20failure%2C%20errors%2C%20and%20natural%20accidents.  
[Accessed 04 May 2021].

Zero Day, 2019. *Anubis Android banking malware returns with extensive financial app hit list.* [Online]   
Available at: https://www.zdnet.com/article/anubis-android-banking-malware-returns-with-a-bang/  
[Accessed 03 April 2021].

# Appendix

## Malware Dataset Structure

|  |  |  |
| --- | --- | --- |
| **Malware Family** | **Malware Type** | **Sample Count** |
| Anubis | Android Malware | 37 |
| CryptoLocker | Ransomware | 96 |
| Ghostrat | Backdoor | 97 |
| Hamweq | Worm | 73 |
| Reveton | Ransomware | 55 |
| Swizzor | Trojan | 160 |
| Vanti | Packer for Password Stealing | 104 |
| Zeus | Trojan | 100 |

## Action Category Dataset

Table : Action Category Dataset Totals

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Action** | **Action Total** | **Category Total** |
| **DNSActionNameVocab-1.0** | **send dns query** | 2086 | 2086 |
| **DebuggingActionNameVocab-1.0** | **check for remote debugger** | 2 | 2 |
| **DeviceDriverActionNameVocab-1.0** | **load driver** | 4 | 4 |
| **DirectoryActionNameVocab-1.0** | **create directory** | 675 | 675 |
| **FileActionNameVocab-1.0** | **copy file** | 96 | 213302 |
| **create file** | 19557 |  |
| **delete file** | 1421 |  |
| **find file** | 39406 |  |
| **get file attributes** | 33173 |  |
| **modify file** | 206 |  |
| **move file** | 4 |  |
| **open file** | 5570 |  |
| **read from file** | 84126 |  |
| **rename file** | 4 |  |
| **send control code to file** | 10709 |  |
| **set file attributes** | 14323 |  |
| **write to file** | 4707 |  |
| **GUIActionNameVocab-1.0** | **find window** | 12379 | 12379 |
| **HTTPActionNameVocab-1.0** | **send http get request** | 565 | 565 |
| **HookingActionNameVocab-1.0** | **add windows hook** | 42 | 42 |
| **IPCActionNameVocab-1.0** | **create named pipe** | 8 | 144 |
| **read from named pipe** | 66 |  |
| **write to named pipe** | 70 |  |
| **LibraryActionNameVocab-1.0** | **get function address** | 173832 | 205943 |
| **load library** | 32111 |  |
| **NetworkActionNameVocab-1.0** | **connect to socket address** | 15332 | 15333 |
| **download file** | 1 |  |
| **ProcessActionNameVocab-1.0** | **create process** | 735 | 1231 |
| **kill process** | 496 |  |
| **ProcessMemoryActionNameVocab-1.0** | **free process virtual memory** | 10775 | 134912 |
| **modify process virtual memory protection** | 104360 |  |
| **read from process memory** | 11356 |  |
| **write to process memory** | 8421 |  |
| **ProcessThreadActionNameVocab-1.0** | **create remote thread in process** | 326 | 1150 |
| **create thread** | 670 |  |
| **get thread context** | 59 |  |
| **kill thread** | 36 |  |
| **set thread context** | 59 |  |
| **RegistryActionNameVocab-1.0** | **close registry key** | 46519 | 445230 |
| **create registry key** | 21553 |  |
| **delete registry key** | 9527 |  |
| **delete registry key value** | 964 |  |
| **enumerate registry key subkeys** | 34007 |  |
| **enumerate registry key values** | 26451 |  |
| **get registry key attributes** | 34035 |  |
| **modify registry key value** | 18327 |  |
| **monitor registry key** | 160 |  |
| **open registry key** | 140356 |  |
| **read registry key value** | 113331 |  |
| **ServiceActionNameVocab-1.0** | **create service** | 27 | 1359 |
| **delete service** | 5 |  |
| **modify service configuration** | 8 |  |
| **open service** | 1135 |  |
| **send control code to service** | 62 |  |
| **start service** | 122 |  |
| **SocketActionNameVocab-1.0** | **bind address to socket** | 78 | 1343 |
| **close socket** | 381 |  |
| **connect to socket** | 13 |  |
| **create socket** | 312 |  |
| **get host by name** | 462 |  |
| **listen on socket** | 2 |  |
| **receive data on socket** | 9 |  |
| **send data on socket** | 9 |  |
| **send data to address on socket** | 77 |  |
| **SynchronizationActionNameVocab-1.0** | **create mutex** | 3077 | 11818 |
| **open mutex** | 8741 |  |

## Python Programs

### dataset\_download.py

# B00107945 Pauline Finlay  
# TUD Blanchardstown 2021  
# Download .xml files from GitHub  
  
# Import Statements  
import requests  
  
  
# enter the web url to the .xml file  
url = "https://raw.githubusercontent.com/MAECProject/datasets/master/" \  
 "zeus/" "cuckoobox" \  
 "/fcf4a1169ed19d277e42c48815a02905\_cuckoobox\_maec.xml "  
  
# download the web xml content and output to new file naming keeping GitHub  
# naming format  
response = requests.get(url)  
with open('fcf4a1169ed19d277e42c48815a02905\_cuckoobox\_maec.xml', 'wb') as \  
 file:  
 file.write(response.content)

### Feature Extraction

# B00107945 Pauline Finlay  
# TUD Blanchardstown 2021  
# Read Malware Files and extract action calls  
  
  
# import statements to include libraries  
import re  
import pandas as pd  
import matplotlib.pyplot as plt  
from pathlib import Path  
import seaborn as sns  
import chart\_studio.plotly as py  
  
colorscale = 'ylorbr'  
projection = {'type': 'mercator'}  
from plotly import \_\_version\_\_  
  
print(\_\_version\_\_)  
import cufflinks as cf  
from plotly.offline import download\_plotlyjs, init\_notebook\_mode, plot, iplot  
  
init\_notebook\_mode(connected=True)  
cf.go\_offline()  
  
# variable malfolder with path to specified malware report folder choosing only .xml files  
malfolder = Path('C:\\\\Users\\\\IEUser\\\\PycharmProjects\\\\MAEC\\\\malware').rglob('\*.xml')  
# Variable for each file in folder  
malfiles = [x for x in malfolder]  
# print statement to count number of files in folder  
print("count={}".format(len(malfiles)))  
  
# create empty list  
data = []  
# read each file in folder  
for file in malfiles:  
 # open file and read line by line  
 with open(file) as f:  
 f = f.readlines()  
 # create 2 empty lists one for actions and one for irrelevant  
 actions = []  
 irrelevant = []  
 # set the MAEC naming format for actions we wish to keep  
 keep\_text = ['FileActionNameVocab-1.0', 'DirectoryActionNameVocab-1.0', 'LibraryActionNameVocab-1.0',  
 'NetworkActionNameVocab-1.0', 'HTTPActionNameVocab-1.0', 'DNSActionNameVocab-1.0',  
 'RegistryActionNameVocab-1.0', 'DiskActionNameVocab-1.0', 'IRCActionNameVocab-1.0',  
 'FTPActionNameVocab-1.0', 'UserActionNameVocab-1.0', 'SocketActionNameVocab-1.0',  
 'ProcessActionNameVocab-1.0', 'SystemActionNameVocab-1.0', 'ServiceActionNameVocab-1.0',  
 'GUIActionNameVocab-1.0', 'SynchronizationActionNameVocab-1.0', 'NetworkShareActionNameVocab-1.0',  
 'IPCActionNameVocab-1.0', 'ProcessMemoryActionNameVocab-1.0', 'ProcessThreadActionNameVocab-1.0',  
 'HookingActionNameVocab-1.0', 'DeviceDriverActionNameVocab-1.0', 'DebuggingActionNameVocab-1.0']  
  
 for line in f:  
 for text in keep\_text:  
 # check each line in the file to search for matching MAEC format actions  
 if text in line:  
 # add the MAEC naming format and use regular expression to file the action,  
 # remove white space from before action adn \n from end of action   
 # and add reformatted line to actions list  
 actions.append(text + ': ' + re.sub(r"<.+?>", "", line.lstrip().rstrip('\n')))  
 else:  
 # add irrelevant lines to other list  
 irrelevant.append(line)  
  
 # create a count of each unique action and add to a dictionary for each file  
 malactdict = dict(pd.value\_counts(actions))  
  
 # add the dictionary for each sample to the data list  
 data.append(malactdict)

### Create DataFrame

# create DataFrame from list data  
df = pd.DataFrame(data)  
print(df)  
  
# split the column names by delimiter, making multilevel columns, & print DataFrame  
df.columns = df.columns.str.split(':', expand=True)  
df.sort\_index(axis=1, inplace=True)  
df = df.reorder\_levels([0, 1], axis=1)  
print(df)  
  
# Transpose DataFrame making columns index with headings 'Category' & 'Action'  
dft = df.T  
dft.index.names = ['Category', 'Action']  
print(dft)

### Export DataFrame to Excel

# Export DataFrame to Excel  
import openpyxl  
import xlsxwriter  
  
dft.to\_excel("final1.xlsx")

### Create DataFrame from .csv file

# create dataframe from csv file  
dfmod = pd.read\_csv('modds.csv')  
print(dfmod)

### Create Interactive Line Plot with Plotly

# create plotly interactive line plot  
df.iplot()

### Create Summated Bar Graphs with Plotly

# Create summated bar graphs  
df.sum().iplot(kind='bar')

### Create Area Graph

# generate area graph using pandas visualization  
plt.figure()  
dfcat.plot.area(stacked=False, figsize=(15, 15), cmap='viridis')  
plt.xlabel('Individual Actions')  
plt.ylabel('Number of Actions Calls')  
plt.title('Action Calls for Full Dataset')

### Create Heatmap

# find the correlation between columns in DataFrame  
cc = dft.corr()  
  
# generate heatmap based on correlation between columns using seaborn  
plt.figure(figsize=(15, 15))  
sns.heatmap(cc, cmap='viridis')  
  
# label axes & title  
plt.xlabel('Sample Number')  
plt.ylabel('Sample Number')  
plt.title('Spiked Samples Comparision')

### Create 3D Surface Graph

# import plotly graph objects  
import plotly.graph\_objects as go  
  
# assign the values to the z-axis  
fig = go.Figure(data=[go.Surface(z=dfcat.values)])  
  
# create graph assign labels and set styling  
fig.update\_layout(title='Full Dataset by Category', autosize=False,  
 width=800, height=800,  
 scene = dict(  
 xaxis = dict(title='Samples'),  
 yaxis = dict(title='Category'),  
 zaxis = dict(title='Count')),  
 margin=dict(l=65, r=50, b=65, t=90))  
  
fig.show()