# Problem Description

Malware categorisation techniques play a critical role in safeguarding existing technical infrastructure. Malware comes in many forms and uses many different attack vectors to gain access to computers in order to cause undesired and potentially harmful behaviours to occur. Attacks can often cause significant financial harm to companies and in some cases individuals.

The symptoms of malware differ, depending on the malware type or family; As the symptoms differ, so does the potential risk to systems infected by the malware.

Categorisation allows us to make an approximation of the potential risk a file poses. Though there are many existing anti-virus scanners available, the results of these scanners can disagree when it comes to the categorisation of a particular piece of malware, and in some rare cases an anti-virus may classify a file as safe, when in fact it is not.

There is likely no perfect way of categorising malicious files, but the goal of many Security Operation Centres is, to quickly and accurately assess the risk associated with a malicious file so that the necessary mitigations and remediation can be put in place.

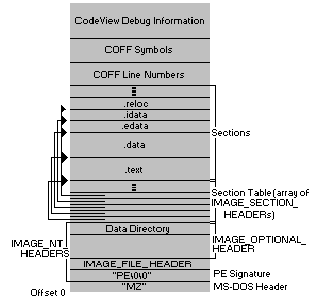
For this project, we will be focusing on static analysis of the malware file structure to determine its category. Our goal is to investigate this new method of malware categorisation, which will data-mine a set of Windows PE files and make use of existing machine learning libraries and clustering algorithms.

**Static Analysis**

Static analysis is a technique used to inspect properties of a piece of malware without loading the executable into memory. The main advantage of this is that the malware does not execute, preventing damage to the host operating system. Static analysis can be carried out on the source code of a piece of malware or on its binary representation (executable). Static analysis also has limitations, such as its inability to fully analyze the behavior of a binary file which uses self-modifying code techniques or relies on data which is not static eg. The current date/time.

For this project, we will only be looking at the binary representations of the malware samples.

There is a large amount of information available from an executable file including its target operating system, the date and time it was compiled and the DLLs it imports. A typical PE32 file consists of several sub-structures shown below.



**Fig1.** The PE File Format [2]

Each of the components of a PE File have a clearly designed specification which can be found at [2].

# Feature Selection

Although there are many features which can be extracted from PE files, not all of these will contribute positively to the accuracy of our solution and the inclusion of some features for clustering may, in fact, result in reduced accuracy. To increase the accuracy of our solution we will be carrying out dimensionality reduction [1] via data mining. This will result in a lower dimensional dataset which will contribute to performance improvements when processing the data, and allow us to visualize our clusters on a scatter graph.

There are several scenarios we will need to cater for with our solution. The first being distinguishing between benign files and malicious files. The experiments carried out by Raman[2] present a useful starting point for this classification, achieving a 98.57% true positive rate with a 5.68% false positive rate. These results were achieved using only 7 independent features of the PE files analysed.

The results achieved by Shafiq et al.[3] are marginally better with respect to true positive rate with an average of 99% and significantly better with respect to false positive rate at 0.5%. This increased accuracy was achieved by mining 189 features from the initial dataset.

**Proposed Machine Learning Features**

**File Entropy**

This is a number between 0 and 8 and represents the randomness of the data within the executable. According to the results proposed by Lyda and Hamrock [3], packed files and encrypted executables usually have a higher value for entropy when compared to non-packed and non-encrypted executables. Since a large proportion of malware uses these techniques while benign files rarely use them we can use this entropy value as an indicator of whether a file has been packed or encrypted and therefore if it is benign or malicious. This indicator alone would obviously be good enough to distinguish between a benign or malicious with 100% accuracy but it is a useful metric nonetheless.

**Ratio of Code to Initialised Data**

This is the ratio of executable code to initialised data within the PEFile. During my initial research, mining data from my dataset of malicious and benign files, I observed that benign files often have a large amount of data but a small amount of executable code in comparison. Conversely I observed that malicious files tend to have little to no data but a larger proportion of executable code. To normalise these observations and make them more meaningful I have chosen to express this ratio as a single machine learning feature as opposed to two distinct features consisting of the size of code and size of initialised data.

**Major Image Version**

This number is the version of the executable. According to Ramen [2] malicious files often have a value of zero and benign files often have a higher value. I observed the same pattern during my initial research and as such this feature has been the best indicator of whether a file is benign or malicious, that I have come across.

**Number Of Sections**

This is the number of section headers in the section header table. As Yonts[4] observed, the number of sections in an executable can be a very good indicator of whether the file is benign or malicious. In general, benign files range from 0-10 sections while malicious files almost always have 3 or 4 sections. I was also able to observe this pattern when data mining my dataset.

**Common Dll Imports**

The dlls imported by an executable paint a very good picture of the functionality and behaviour of the executable. For example, if an executable is importing Wsock32.dll then you can surmise that the executable makes use of networking. Although there will be overlap between the imports of benign files and malicious files, we should be able to classify the malware by family using this information as behaviours differ between families. We will be using the same technique Shafiq et. Al. [3] proposed where we look for a particular import and set a flag to true or false depending on the whether it was present or not.

|  |  |
| --- | --- |
| **DLL Name** | **Function** |
| WS2\_32.dll | Networking |
| Wsock32.dll | Networking |
| Kernel32.dll | Memory Management/ IO operations |
| WININET.DLL | Networking |

## Technologies

#### Languages and libraries

This project will be implemented in Python 3.5. The reasoning for this is that Python is ideal for rapid prototyping; often requiring significantly less code than other object-orientated languages such as Java and C++ due to its dynamic typing. There are of course trade-offs with this choice, namely performance in comparison to C++ and other low level languages.

Due to the specialised requirements of this project, I will be making use of a number of libraries to aide in the rapid development of a working piece of software.

For the machine learning component of the project, I have opted to use scikit-learn. This is an open source machine learning library for python which has implementations of common clustering algorithms such as KMeans and MeanShift. There are also a number of implementations for dimensionality reduction/decomposition available, namely Principle Component Analysis (PCA), Non-Negative Matrix Factorisation (NMF) and Fast Independent Component Analysis (FastICA). I intend to make use of PCA for my decomposition as it is suited to numeric data, while FastICA and NMF are better suited for sound isolation and image data decomposition, respectfully. To cluster the decomposed data, I intend to make use of KMeans as it is widely used for clustering in data-mining applications while MeanShift is better suited to clustering in the context of image processing.

A crucial component of the KMeans algorithm is the number of clusters which the algorithm is instructed to create. Unlike MeanShift the KMeans algorithm does not decide the number of clusters internally based on the data. There are a number of techniques which can be used to determine the optimal number of clusters, one of which is silhouette metric. Scikit-learn provides functionality to calculate a silhouette metric for the data and labels generated by the KMeans algorithm. This metric is expressed as a score between -1 and 1. A score tending towards -1 would indicate that there may be to many or too few clusters and a score tending towards 1 would indicate that the data has been clustered correctly. A score around 0 indicates that the data may fit more than one cluster[6].

The Elbow Method is an alternative technique for determining the optimal number of clusters for a given dataset. This involves calculating the cost function for a given value of K starting with k = 2 and increasing K by 1 until the cost function of K drops drastically [6]. The cost function can be defined as the sum of squared errors, that is, the sum of the squared distance of each point to its nearest centroid. The reason the we see the cost of K decrease as K increases is that, as the number of centroids increases, the distances from each point to its nearest centroid decreases.

For the purposes of this project I will be using both of the previously discussed methods for determining the most appropriate value for K.

I have chosen to use Matplotlib for data visualisation as it provides the ability to plot graphs in 2d and 3d with relatively little code. The data-mined features will initially be too high-dimensional to plot on a graph, so we will use PCA to project the data to a lower dimensional subspace which retains most of the variance from the initial dataset [5]. We will then use the output of this as our input for the KMeans algorithm and then subsequently plot the clustered data using Matplotlib.

To implement a rudimentary GUI which will display the plotted graphs and evaluation of the clustering labels, we will be using PyQt which is a cross platform python binding used for rapid GUI development.

#### Environment

Due to the malicious nature of the Windows PE files I will be analysing, my host operating system for my development environment is Mac OS X. This will ensure that the malware files do not execute, as OS X uses the ELF format for executables, similar to a Linux environment.

I will be using Spyder as my Python IDE which is packaged with the Anaconda Python data-science platform. Anaconda comes with the Pylab package which includes Numpy and Matplotlib. My reasons for choosing the Anaconda Python platform were its ease of set-up and its ability to create multiple python environments with different versions of referenced libraries. It also comes bundled with scikit-learn and PyQt which minimises the time required to set-up a full development environment.

To create my GUI, I will be using the tools contained within QtCreator IDE. QtCreator is a cross platform C++ IDE which comes bundled with QtDesigner. QtDesigner is a WYSIWYG GUI design tool, which outputs a .ui file that can be used to generate a PyQt GUI class, using the pyuic5 command line tool. This solution to designing and creating a GUI minimises time spent creating complex layouts which could be better spent elsewhere in the development of the solution. It also allows for significantly more complicated layouts than those possible with built in Python GUi libraries such as TKinter.

**System Design**

1. Browse to a directory of PE32 files. For ech file create an instance of the PEFile class. This will expose properties of the original PEFile which have been parsed from the file on disk.
2. For each PEFIle object extract our chosen machine learning features to a row in a CSV file.

|  |  |
| --- | --- |
| Feature | Source |
| File Entropy | PEFILE.GetFileEntropy() |
| Code/ Data Ratio | PEFILE.GetCodeDataRatio() |
| Major Image Version | PEFILE.GetMajorImageVersion() |
| Number Of Sections | PEFILE.NumberOfSections() |
| CommonDll Imports | PEFILE.GetImports() - map this list of imports to an array of true or false values for the presence of certain dlls. |

1. Perform Principle Component Analysis on the high dimensionality data in the CSV file and project to a lower dimensional subspace suitable for plotting. Store this in a Numpy array.
2. Cluster data from array using KMeans algorithm and plot data points on a 2d Matplotlib scatter graph.



Example K=2 KMeans clustering for malicious (red) and benign(green) files.

1. Use labels generated by KMeans Algorithm in conjunction with filenames to determine the accuracy of the clustering. Note: for the above clustering malicious files were prefixed with MAL and benign files prefixed with BEN. This allows us to quickly compare the clustering of files.

**Clustering accuracy = Total number of correctly clustered files**

**Total number of files analysed**

## Gannt Chart

# References

[1] S. Gadhiya and K. Bhavsar, “Techniques for Malware Analysis,” *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 3, no. 4, pp. 2277–128, 2013.

[1] M. Pietrek, “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format,” *Microsoft Syst. Journal-US Ed.*, vol. 9, no. 3, pp. 15–38, 1994.

[2] “Microsoft Portable Executable and Common Object File Format Specification,” 2015. [Online]. Available: http://download.microsoft.com/download/9/c/5/9c5b2167-8017-4bae-9fde-d599bac8184a/pecoff\_v83.docx.

[4] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT Press, 2012.

[5] K. Raman, “Selecting Features to Classify Malware,” *InfoSec Southwest 2012*, pp. 1–5, 2012.

[6] R. Lyda and J. Hamrock, “Using Entropy Analysis to Find Encrypted and Packed Malware,” *IEEE Secur. Priv.*, vol. 5, pp. 40–45, 2007.

[7] J. Yonts, “Attributes of Malicious Files,” 2012. [Online]. Available: https://uk.sans.org/reading-room/whitepapers/malicious/attributes-malicious-files-33979. [Accessed: 08-Dec-2016].

[8] M. Zubair Shafiq, S. Momina Tabish, F. Mirza, and M. Farooq, “‘PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime’ in Recent Advances in Intrusion Detection,” Springer Science + Business Media, 2009, pp. 121–141.

[9] T. M. Kodinariya and P. R. Makwana, “Review on determining number of Cluster in K-Means Clustering,” *Int. J. Adv. Res. Comput. Sci. Manag. Stud.*, vol. 1, no. 6, pp. 2321–7782, 2013.