Computers & Security 116 (2022) 102660

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

TC 11 Briefing Papers

Image-based malware classification hybrid framework based on space-filling curves

Stephen O'Shaughnessy*, Stephen Sheridan

Department of Informatics, Technological University Dublin, Dublin, Ireland

ARTICLE INFO

Article history:
Received 7 June 2021
Revised 10 February 2022
Accepted 13 February 2022
Available online 16 February 2022

Keywords:
Malware classification
Space-filling curves
Image processing
Computer vision

ABSTRACT

There exists a never-ending "arms race" between malware analysts and adversarial malicious code developers as malevolent programs evolve and countermeasures are developed to detect and eradicate them. Malware has become more complex in its intent and capabilities over time, which has prompted the need for constant improvement in detection and defence methods. Of particular concern are the anti-analysis obfuscation techniques, such as packing and encryption, that are employed by malware developers to evade detection and thwart the analysis process. In such cases, malware is generally impervious to basic analysis methods and so analysts must use more invasive techniques to extract signatures for classification, which are inevitably not scalable due to their complexity. In this article, we present a hybrid framework for malware classification designed to overcome the challenges incurred by current approaches. The framework incorporates novel static and dynamic malware analysis methods, where static malware executables and dynamic process memory dumps are converted to images mapped through space-filling curves, from which visual features are extracted for classification. The framework is less invasive than traditional analysis methods in that there is no reverse engineering required, nor does it suffer from the obfuscation limitations of static analysis. On a dataset of 13,599 obfuscated and non-obfuscated malware samples from 23 families, the framework outperformed both static and dynamic standalone methods with precision, recall and accuracy scores of 97.6%, 97.6% and 97.6% respectively.

1. Introduction

Incidents of malware attacks are continually rising, with antivirus (AV) vendors, such as Kaspersky, reporting an average of 360,000 malware samples received daily (Kaspersky, 2020). Malware's relentless growth can be attributed to the financial rewards that can be gained from malware exploits such as crypto-currency coin miners, banker Trojans and ransomware. According to AV-Test (2020), approximately 87% of malware processed in 2020 were not original programs, but rather variants of existing families. This presents a major problem to AV companies in terms of the processing overheads required to analyse every sample. Automated tools can help alleviate the burden, but generally are configured for detection rather than classification. Analysis is further hampered by malware obfuscation. Obfuscation has legitimate uses in computing to secure applications from copying or to protect intellectual property, but is also widely adopted by malware developers to evade detection from AV scanners and slow down the analysis process. For the most part, static analysis is not robust against obfuscation, due to the upheaval of the binary structure of the malware program caused by these obfuscation methods. Dynamic approaches overcome the obfuscation limitations of static analysis, since the programs must be de-obfuscated and written to main memory prior to execution. However, dynamic analysis can produce a considerable number of false positives, due to similarities in behaviour to benign programs or other families of malware. Hybrid methods, where elements of both static and dynamic approaches are combined, have been presented to overcome these limitations (Anderson et al., 2012).

In this article, we present a novel malware classification hybrid framework based on computer vision techniques. Computer vision has been previously demonstrated as a viable alternative to the more typical methods for malware classification, such as static frequency and sequence-based approaches (Khalilian et al., 2018; Kolter and Maloof, 2004) or dynamic API call analysis (Fukushima et al., 2010). Image-based approaches to malware classification involve the conversion of malware binaries to 2-dimensional images, such that characteristic properties or features

* Corresponding author.
E-mail addresses: stephen.oshaughnessy@tudublin.ie (S. O'Shaughnessy), stephen.sheridan@tudublin.ie (S. Sheridan).

https://doi.org/10.1016/j.cose.2022.102660
0167-4048/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/)

of the original file are retained (Vasan et al., 2020; Wagner et al., 2015; Yajamanam. et al., 2018). Computer vision techniques are then used to extract discriminant features from the images for classification. Malware classification through computer vision can be considered less invasive than the analysis approaches mentioned previously, as no reverse-engineering or domain-specific knowledge is required, so it is less complex and thus can improve scalability for the processing of large malware datasets.

Our framework combines static and dynamic features gathered from malware samples converted to 2-dimensional images, mapped through space-filling curve (SFC) traversals. In mathematical analysis, SFCs are curves whose ranges contain the entire 2-dimensional unit square or more generally, an n-dimensional unit hypercube. However, for the purposes of this research, only the 2-dimensional space is considered, since the outputs are 2-dimensional images. In this case, the 2-dimensional unit square represents an image of $n \times n$ pixels. SFCs pass through every pixel point of a regular spatial region, such that the spatial locality of the data is preserved, i.e., closely located points in the binary file space will also be closely located when mapped to an SFC image. This is of significant importance, as the data structures within the original malware binaries can be retained in the resulting SFC images and thus used to represent the malware for classification purposes. Our hybrid method processes non-obfuscated samples statically and obfuscated samples dynamically. Non-obfuscated samples do not require behavioural dynamic analysis because sufficient features can be captured from the static binary. Dynamic analysis can be considered more invasive since malware samples must be executed in a controlled environment to capture their behavioural data and then written to physical memory on the host. Additionally, the time complexity for a fully dynamic analysis approach would make it infeasible for large scale datasets. Adopting this hybrid approach enabled us to optimise the analysis process. Our hybrid framework is discussed in detail in Section 4. The main contributions of this research are:

- Novel image-based malware classification framework that combines the strengths of static and dynamic analyses to overcome the challenges of obfuscated programs;
- A novel representation of malware behavioural data derived from process memory dumps mapped to SFC image format;
- A study of several feature descriptor and classification approaches for image-based malware classification;
- Three open source malware image datasets in SFC format, made available for further research (ÓShaughnessy, 2019).

2. Related work

In this section, we review related visualisation-based techniques applied to malware analysis. Our focus is on binary-based approaches where malware binary programs are converted into image format and from there, discriminant features are extracted for classification. Such approaches do not require any preliminary pre-processing of the malware binaries to extract characteristic features.

2.1. Visualisation of binary files

The first study on the visualisation of binary files as images was introduced in the work of Conti et al. (2008), which presented a method of reverse engineering binary files into visual images, dubbed *byteplots*, to enhance the capabilities of text-based hex editors. Conti et al. used the byteplot method to create a visual taxonomy of binary file fragments to aid in the identification of common file formats. In (Conti et al., 2010), the authors extended their previous work using a K-nearest neighbours classifier to classify file types by their visual features, trained on a dataset of 14k samples comprising 14 file types, including random data, encryption, compression, machine code, text and bitmap images. The best results reported were from un-encoded and base64 encoded files, with 100% accuracy, with plain text files scoring 98.7%. While the initial work presented by Conti et al. was not in the malware domain, it demonstrated that the internal static structure of binary files could be represented as 2-dimensional images, which paved the way for further research efforts and tools that utilised byteplots for malware classification.

2.2. Byteplot malware visualisation

Nataraj et al. (2011a) first applied byteplots to represent malware binaries for classification. Their method visualized static malware samples as gray-scale images, with the observation that for many malware families, the images belonging to the same family appear very similar in layout and textures. Malware feature vectors were derived using GIST, a global feature descriptor first presented by Oliva and Torralba (2001), that breaks images into sub-band blocks and computes features based on filters tuned to varying scales and orientations. The authors compiled a dataset, dubbed Malimg, comprising 9548 samples from 25 malware families. Using a K-nearest neighbours (KNN) classifier, Nataraj et al. reported a classification accuracy of 98%. Nataraj et al. (2011b) followed on from their original research, presenting a comparison of the byteplot method with dynamic analysis. They used a combination of API hooking and pre-infection and post-infection system snapshots to build dynamic feature vectors. Results showed that overall, image classification was comparable in terms of classification accuracy with dynamic analysis methods but outperformed the latter in terms of reduced time complexity and robustness against obfuscated malware samples. While Nataraj et al. report the byteplot method as resilient to obfuscation, their solution to identifying packed malware was to treat it as a separate class. This approach does not solve the packed malware problem as it is identifying the packed and unpacked samples from the same families as two different species. Furthermore, the authors do not consider the effect of encrypted malware samples on their GIST-based classification method.

A number of subsequent research efforts applied the byteplot format to malware classification. Luo and Lo (2017) extracted visual features for classification using Tensorflow and Local Binary Patterns (LBP) feature descriptors on the Malimg dataset. Results showed that LBP and Tenserflow convolutional neural network produced a classification accuracy of 93.17% as opposed to 82.83% using the method proposed by Nataraj et al. (2011a). Yajamanam et al. (2018) evaluated the GIST-based byteplot method using a subset of features of the original method. Further testing was carried out using deep learning with Tensorflow. Results were comparable with the original research of Nataraj et al. (2011a), while using a smaller feature set of 60 features. Le et al. (2018) used convolutional neural networks on binary files from the Microsoft Malware Classification Challenge dataset (Ronen et al., 2018) converted to byteplot images. Results on the data reported 98.8% accuracy on validation data with a fast processing time-frame of approximately 20ms per sample.

Fu et al. (2018) extracted features from both gray-scale and colour byteplots using local gray-level co-occurrence matrices and global colour moments, which were then passed to Random Forest, K-Nearest Neighbour and Support Vector Machine classifiers. The experiments were carried out on a dataset comprised of 7087 malware samples from 15 distinct malware families showed the combined feature sets performed better than either local or global features alone, scoring 97% precision, recall, accuracy and f-measure with the Random Forest classifier.

2.3. Space-filling curve malware visualisation

In contrast to the byteplot-related works discussed here, limited research has been presented on malware classification using space-filling curves. Baptista (2018) presented a method of classifying malware by type using Hilbert curves and a Self-Organizing Incremental Neural Network. Experiments were conducted on 180 samples, 78 of which were benign. The malware was classified into four classes: Trojan, ransomware, other and unknown. Although the author reported classification accuracies of 89%, the small sample set of 180 samples is insufficient to provide a clear indication of the merits of applying the Hilbert curve to malware classification. Smaller datasets tend to produce overfitting, which means the model will not generalise well to unseen samples. Vu et al. (2019) presented a malware detection scheme which consisted of a hybrid of malware statistical and syntactic features translated to an image format using several layout functions, including space-filling curves. The scheme, dubbed *hybrid image transformation* (HIT) was tested on a dataset of 16,000 benign and malicious samples. HIT was evaluated using four different layout functions, including two SFC implementations, namely Hilbert and Hcurve. The authors report the best results of 93% accuracy using their hybrid method and Hilbert SFC layout function. The scheme presented in this article considers only two classes, benign or malicious, so does not classify malware into family groups. In terms of classification, accuracy was the only performance metric used to gauge classification performance. Accuracy alone may not be sufficient to measure performance; other metrics such as precision, recall and F1-score could have been employed to give a better indication of the scheme's success. Furthermore no testing was performed using a holdout dataset to evaluate how well the model's generalised to unseen data.

2.4. Summary

A review of the related research discussed in this section identifies several limitations. First, previous work focused predominantly on the byteplot image format to represent the malware data. This format is essentially a static view of the malware binary, which is not robust against obfuscation techniques such as encryption or packing. Next, the Malimg dataset collected by Nataraj et al. (2011a), was the test dataset chosen by the majority of researchers to represent malware in image format. This dataset was compiled in 2011 and so could not be considered representative of the current malware threat landscape. Other research efforts utilized the Microsoft Challenge dataset. The file headers have been removed from the malware samples in this dataset for sterility. The incomplete nature of the malware samples in this dataset means that it is not possible to extract a full feature set for classification.

The work presented in this article follows on from previous research in (O'Shaughnessy, 2019), where the efficacy of space-filling curves was evaluated as a means of representing malware variants for classification. Three SFC image datasets were produced by mapping 9235 32-bit executable malware samples from 28 distinct families to images using Z-order, Hilbert and Gray-code curve traversals. Local Binary Patterns (LBP), Histogram of Oriented Gradients (HOG) and Gabor filters were chosen as feature extraction descriptors, based on their suitability for texture analysis. The features were used to train Random Forest, K-Nearest Neighbour and Support Vector Machine classifiers. The best results were obtained from the KNN-HOG model trained on the Z-order dataset, with precision, recall and accuracy metric scores of 94.5%, 87.1% and 91.6% respectively. A comparative assessment with the method presented by Nataraj et al., showed the KNN-HOG Z-order SFC model outperformed the GIST byteplot method against previously unseen samples. This research demonstrated the suitability of SFCs for representing malware binaries in image format. However, as images were generated from static malware executable binaries, the method did not perform optimally against obfuscated samples.

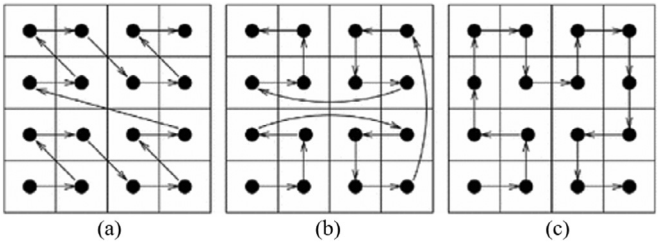


Fig. 1. Space-filling curve traversals. (a) Z-order, (b) Gray-code and (c) Hilbert.

The framework presented in this article overcomes the limitations of the previous research discussed, in the following ways:

- A malware dataset which is representative of current malware variants.
- A hybrid approach to feature selection and extraction, combining the strengths of static and dynamic analysis, with less overheads and limitations than previous methods discussed.
- Space-filling curves used to represent the malware for classification. SFCs have previously demonstrated better classification performance compared to the predominant byteplot method.

3. Space-filling curves

Space filling curves are mathematical constructs, also known as continuous fractal curves, the limits of which contain the entire 2-dimensional unit square. Peano (1890) introduced the first SFC, followed by an improved variant by Hilbert (1891). Other variations include Moore (Moore, 1900), Z-order (Morton, 1966) and Gray-code (Gray, 1953) curves. The Hilbert, Z-order and Gray-code curves were considered for the purposes of this article. The basic traversal paths of these curves are shown in Fig. 1.

3.1. Suitability of space-filling curves for binary data representation

The locality preservation properties of SFC's mean they can be utilised in many areas of computing, such as partitioning or reordering data and computations (Moon et al., 2001). Practical applications of SFC implementations can be found in technologies such as database indexing (Huang, 2013), parallel computing Nivarti et al. and image retrieval (Nguyen et al., 2012). Google's S2 Geometry library uses the Hilbert curve for Geo-spatial indexing in the Google Maps application (S2-Geometry, 2017). In the context of this research, the aim is to capture all relevant characteristic features of malware binaries in an image format so that discriminant features may be extracted for classification purposes. At the core of these techniques is a layout function that inputs a set of values and outputs a partition of space on the resulting mapped image, such that regions represent disjoint data that do not intersect. Previously, tree-maps, have been presented as a solution to the graphical display of data (Johnson and Shneiderman, 1991). However, for high dimensional, dense layout tree maps, aspect (width-to-height) ratios tend to be extremely high and thus elements in the mapped image are difficult to correlate and interpret. Wattenberg (2005) defined the notion of a "perfect" layout function, i.e., a layout method for a space-filling visualization that satisfies the following four desirable criteria:

- *Stability*: a small change in inputs should produce a subsequent small change in outputs;

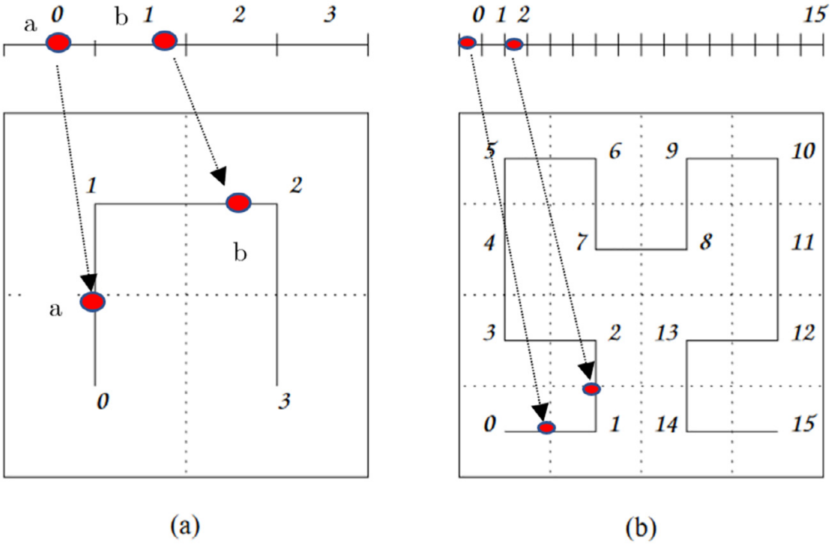


Fig. 2. Mapping data through SFC curve. Points a and b represent data points mapped to the resulting 2-dimensional SFC image.

- *Split neutrality*: any structural changes do not affect the structure of other regions on the image map;
- *Order adjacency*: each similar item should be located adjacent to each other;
- *Locality*: layout regions should be relatively compact. If layout regions are represented loosely, e.g. long and narrow, then the visualisation of regions will not be as clear.

Wattenberg derived a proof that any perfect layout function is actually an instance of a jigsaw map, so-called due to its resemblance of a jigsaw puzzle, thus providing a complete characterization of perfect layouts in terms of screen-filling curves. In particular, SFCs possess these four essential properties, therefore are a practical solution to representing data in a 2-dimensional format.

3.2. Mapping binary files using space-filling curves

A SFC traces a continuous curve through every unit square, i.e. pixel, in the image. To envision this for the purpose of file mapping, the binary code of a malware file can be thought of as a flattened out 1-dimensional line because the code is parsed sequentially, byte-by-byte. The SFC maps each point (bytes) from the 1-dimensional space to the 2-dimensional space (SFC image map) such that closely located points in the binary file space will tend to also be closely located when mapped to the SFC.

Fig. 2 depicts the mapping of a 1-dimensional set of points to a 2-dimensional curve representation, in this case the Hilbert curve, in first and second order iterations. It shows that closely located data points (a and b in the example) map to similar positions on the SFC map. For the purposes of this research, we used `scurve`, a library written in Python, to convert the malware binaries into SFC image format (Cortesi, 2012).

The scurve library uses a colour-coding scheme, based on data type, that maps the malware binary data to the resulting SFC images. The colour scheme classifies bytes (0–255 range) into the following categories: black for 0 × 00 (0), blue for ASCII text (1–126), red for high-value bytes (127–254), and white for 0×`ff` (255). Fig. 3 shows an example of an SFC image in Z-order generated from the Chir family, which is a type of email worm file infector. The image was generated using the static PE executable file and processed with the scurve library. If the data type from the malware binary is distinct, it is displayed as non-overlapping sections of a single colour. Fig. 3(*d*) displays a large region of black, indicating large sections of the file contain zeros or null padding, which is sometimes used as a simple obfuscation method by malware developers. There is also a large region of printable characters, shown in blue Fig. 3(*c*), with some small sections of high-value bytes, denoted by the red regions Fig. 3(*b*). The rest of the file comprises a mixture of multiple data types, which give the resulting image distinct texture regions Fig. 3(*a*). Distinct features that are common to related species of malware can be extracted from the textures produced in the SFC images for classification.

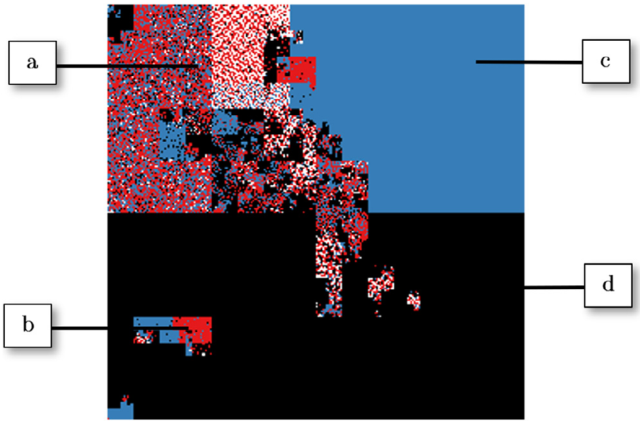


Fig. 3. SFC image mapped from a sample of the Chir worm. *a* denotes a mixture of data types; *b* is a region of high value bytes (127–255 range); *c* is a region of ASCII printable characters; *d* is a region of null bytes (binary zeros).

4. Hybrid framework

This article introduces a novel framework for the classification of malware variants into their respective family classes through computer vision and machine learning. The framework comprises static and dynamic analysis components, i.e. features extracted from malware binaries in both static or non-running and dynamic or running states. A process workflow of the framework is shown in Fig. 4. This section describes the framework, which comprises three main stages: malware conversion, feature extraction and classification. Malware sample data gathering and pre-processing are first discussed.

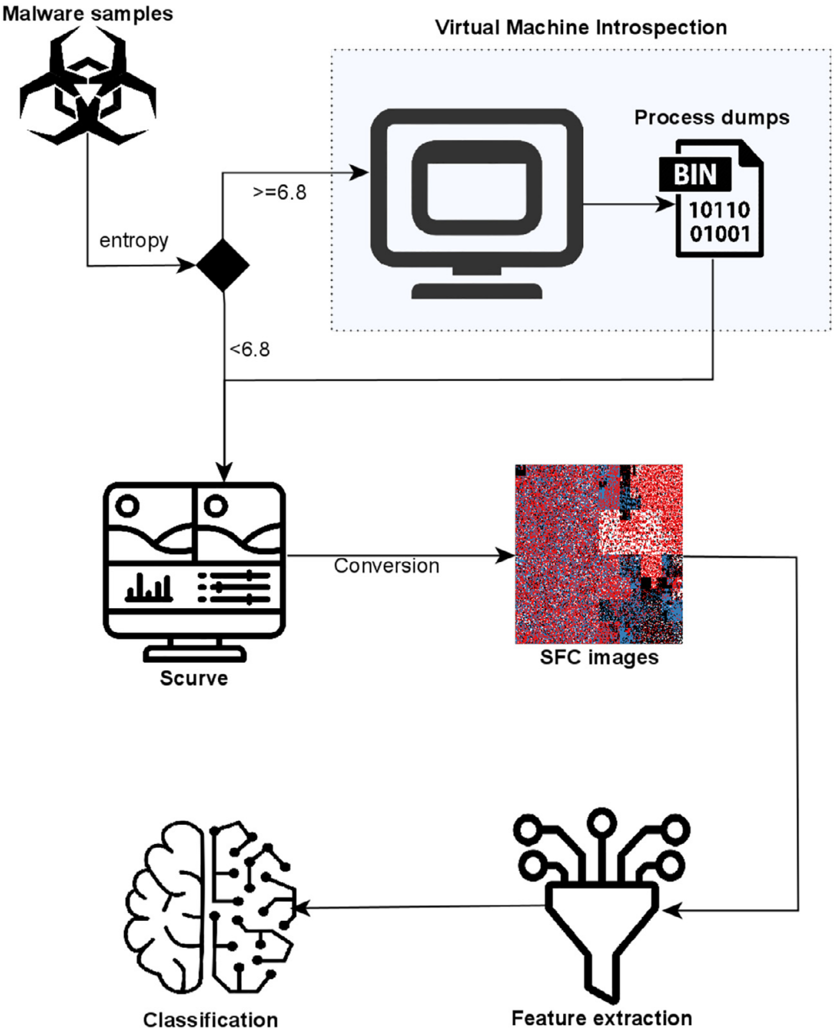


Fig. 4. Process workflow of the proposed SFC hybrid framework.

4.1. Data gathering and pre-processing

A dataset comprising 13,599 32-bit executable portable executable (PE) samples, from 23 distinct malware families was compiled from the VirusTotal Academic share repository. Three image datasets were derived from the samples through the Hilbert, Gray-code and Z-order curve traversals using the scurve library. Table 1 shows the structure of the dataset, including the family and count for each class. Out of the 23 family classes, 5 were considered obfuscated. The dataset contains a more up-to-date set of samples than the Malimg and Kaggle datasets, with some of the most prevalent and current malware species in circulation today. To test the predictive capabilities of the classification models on unseen malware, an evaluation set of 1350 samples or approximately 10% of the data was held out from the original dataset. Samples were chosen using stratified random sampling, which gives a subset structure that is representative of the entire dataset. Following the removal of the data for the evaluation set, the training dataset consisted of 12,249 samples. It should be noted that benign files were not included in the dataset, as the aim of this research was to classify variants into their correct family classes, rather than detect between malicious and benign samples.

Prior to conversion of the malware binaries to SFC image format, the Shannon entropy for each sample was first calculated to determine the level of obfuscation, if any. Shannon entropy

Table 1
Malware dataset comprising samples collected from the VirusTotal academic share repositories.

Family	Obfuscation	Count
InstallMonster	No	575
Agen	Yes	676
Autoit	No	966
Allaple	No	339
Bitman	No	1090
Vilsel	No	752
Berbew	No	1692
Emotet	Yes	492
Hematite	No	562
Dinwod	No	893
Trojan.Agent.BDMJ	No	97
Picsys	No	534
Shifu	No	476
Viking	No	264
Locky	No	648
Cryptowall	Yes	282
Scar	Yes	304
Dridex	No	523
Salgorea	No	607
Socks	No	495
Sytro	No	1140
Dorkbot	No	92
Chir	Yes	306
Total		**13,599**

(Shannon, 1948) measures the degree of randomness for a given set of data, ranging from 0 for orderly or non-random data, to 8 for very random data. For binary files, Shannon entropy can be used as an indicator for malware obfuscation since techniques such as encryption or packing distort binary data and therefore produce a high entropy score. Previous work by Lyda and Hamrock (2007) demonstrated that obfuscation techniques such as encryption and packing increase the entropy value of the original binary file. Using a dataset of 21,576 portable executable malware files, the authors produced a table of entropy ranges to identify the obfuscation type. The average entropy for packed malware binaries was found to be 6.8, whereas the average entropy for encrypted binaries was 7.1. The average entropy for a native executable file was found to be 5.1. For the purposes of this research, we considered the value μ as the Shannon entropy threshold of 6.8, to determine if samples were obfuscated.

4.2. Virtual machine introspection

As stated previously, we optimised the framework by only extracting dynamic or behavioural data from the samples considered to be obfuscated, above the threshold μ. Dynamic analysis can be considered more invasive than static methods since the samples must be executed in a controlled environment to capture their behavioural data, which is then written to the physical host. Consequently, the time complexity to execute every sample in this manner would make it infeasible for large datasets. Furthermore, to implement this method on non-obfuscated malware samples would not yield any greater results, as demonstrated by O'Shaughnessy (2019), where classification performance for low entropy malware executables in SFC format was found to be high. For dynamic analysis, we implemented Virtual Machine Introspection (VMI), a process introduced by Garfinkel and Rosenblum (2003) that refers to the inspection or monitoring of operating systems residing inside a virtual machine from the hypervisor level. In this case, malware is passed to a virtual machine, is executed and a process memory dump is extracted using the Microsoft utility Procdump (Russinovich and Richards, 2020). The dump file is then written back to disk on the host machine. To optimise the analysis process, mini process dumps containing process thread and handle information, were extracted from each malware in execution. While full process dumps contain the full malware process data, it was found they ranged on average from 50 to 70 MB for each dump, which would not scale well for large datasets and so were not considered for the purposes of this research.

4.2.1. Process dump file analysis

In this section we analyse the contents of sample process dump files in order to demonstrate the advantage of using dynamic data over static analysis for obfuscated malware. Static methods are limited to the analysis of structure and code of the malware binary. However, obfuscation methods such as packing, encryption and encoding alter the binary structure of the code to the extent that common salient features cannot be obtained for the purposes of classification. By executing malware in a controlled virtual environment, the binary must be de-obfuscated and written to memory. Thus, we can extract behavioural features characteristic of same-family classes. As stated previously, we use the small or mini dump format for processing efficiency. Mini dumps contain the following data:

- The Stop message and its parameters and other data
- A list of loaded drivers
- The processor context (PRCB) for the processor that stopped
- The process information and kernel context (EPROCESS) for the process that stopped
- The process information and kernel context (ETHREAD) for the thread that stopped
- The Kernel-mode call stack for the thread that stopped

To illustrate the suitability of process dumps for representing malware behavioural features, we focus here on examining the call stack traces taken from two variants of the Chir family. The call stack is made up of the chain of function calls that have led to the current location of the program counter, where the process has been terminated by the Procdump tool. Intuitively, malware binaries that share similar source code will behave the same when executed on the target machine, which results in similar execution traces in the same sequence. These similarities can be used as signatures to group malware variants into their respective family classes. It is infeasible to show a complete stack trace of each program, so we confine our comparison to the process injection routine of the Chir samples, shown in Fig. 5.

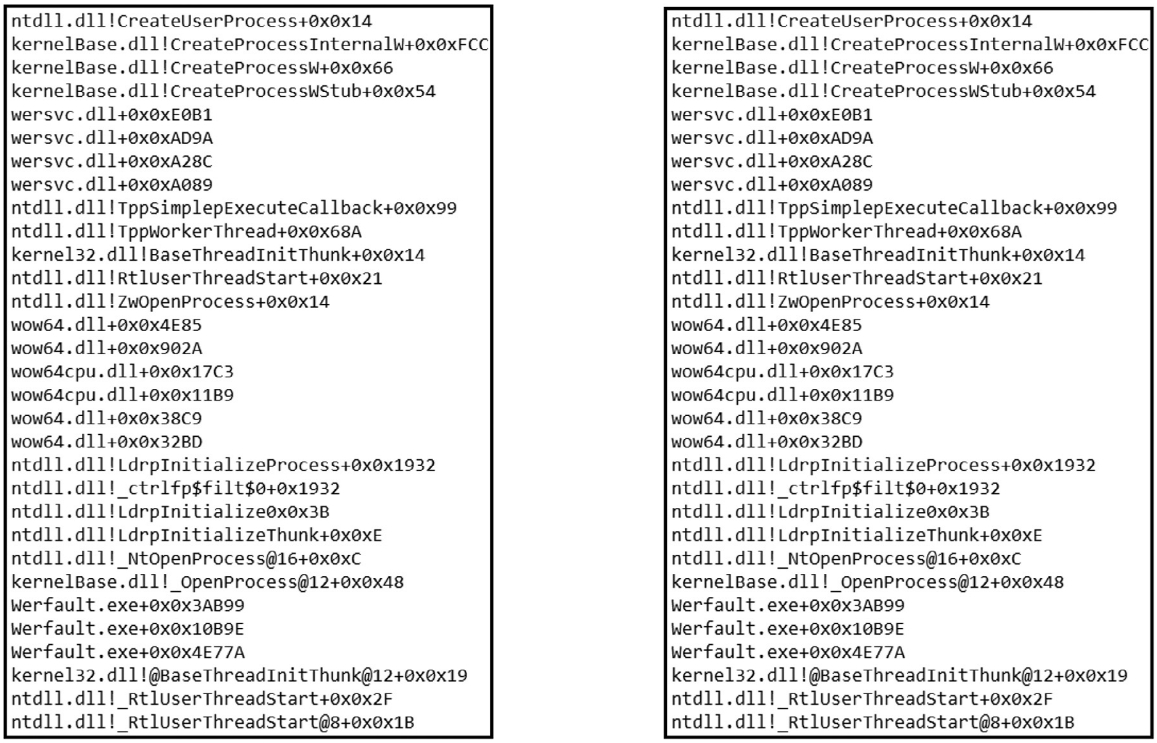
Chir malware implements a process injection routine to obfuscate its presence on the operating system. During this routine, the malware injects a copy of itself, via `svchost.exe`, into `WerFault.exe`, which is a legitimate Windows program for error fault reporting. This specific routine is characteristic of the Chir variant, so could be considered as part of the signature for identification. From Fig. 5, the function calls and call sequences are identical, which indicates the two samples share the same behavioural traits for this code injection routine. The behavioural data captured from the running process dumps allowed us to overcome the obfuscation limitations and improve classification performance over the static standalone method, as discussed in Section 5.

4.3. Malware conversion

Once the malware samples have been processed, the next stage in the framework is to convert the non-obfuscated malware executable samples and the obfuscated sample process dumps to SFC format through the `scurve` library. This section discusses the implementation of the mapping procedures using the `scurve` library outlined in Section3. The Hilbert implementation differs from the Z-order and Gray-code methods, which are inherently similar. The Hilbert implementation first determines the dimensions of input binary. It then iterates through each coordinate point, which is mapped to its data colour class. The implementation of the Z-order and Gray code curves are similar, in that the scurve library considers the bit range for each distinct chunk of binary data as coordinates. In other words, the start offset is the first coordinate and the end offset is the second coordinate. In the case of the Z-order code, the bit interleaving process then takes place. For the Gray code, the coordinates are XORed from left to right. The resulting codes in each case, represent the cell that the binary chunk will occupy, which is mapped to a distinct colour, depending on the data colour class.

Fig. 6($a - c$) illustrates an individual sample taken from the Allaple malware family, transformed into SFC images through the Hilbert, Gray-code and Z-order traversals. The areas of blue (ASCII printable characters) and the smaller areas of red (high value non-printable bytes) are evidently similar across the three SFC images. However, the positions of the texture patterns differ, due to the different traversal patterns of the space-filling curves.

Examples of SFC images converted from the process dump files are shown in Fig. 7. Each image represents a variant of the Cryptowall ransomware family. The Cryptowall variant uses encrypted sections as a means of obfuscation and so static analysis methods produce poor results. This necessitated their execution through VMI to gather behavioural data for classification. It can be seen from the images that the data texture regions are sparse, with large regions of black, which equate to binary zeros. An examina-



```
ntdll.dll!CreateUserProcess+0x0x14
kernelBase.dll!CreateProcessInternalW+0x0xFCC
kernelBase.dll!CreateProcessW+0x0x66
kernelBase.dll!CreateProcessWStub+0x0x54
wersvc.dll+0x0xE0B1
wersvc.dll+0x0xAD9A
wersvc.dll+0x0xA28C
wersvc.dll+0x0xA089
ntdll.dll!TppSimplepExecuteCallback+0x0x99
ntdll.dll!TppWorkerThread+0x0x68A
kernel32.dll!BaseThreadInitThunk+0x0x14
ntdll.dll!RtlUserThreadStart+0x0x21
ntdll.dll!ZwOpenProcess+0x0x14
wow64.dll+0x0x4E85
wow64.dll+0x0x902A
wow64cpu.dll+0x0x17C3
wow64cpu.dll+0x0x11B9
wow64.dll+0x0x38C9
wow64.dll+0x0x32BD
ntdll.dll!LdrpInitializeProcess+0x0x1932
ntdll.dll!_ctrlfp$filt$0+0x1932
ntdll.dll!LdrpInitialize0x0x3B
ntdll.dll!LdrpInitializeThunk+0x0xE
ntdll.dll!_NtOpenProcess@16+0x0xC
kernelBase.dll!_OpenProcess@12+0x0x48
Werfault.exe+0x0x3AB99
Werfault.exe+0x0x10B9E
Werfault.exe+0x0x4E77A
kernel32.dll!@BaseThreadInitThunk@12+0x0x19
ntdll.dll!_RtlUserThreadStart+0x0x2F
ntdll.dll!_RtlUserThreadStart@8+0x0x1B
```

```
ntdll.dll!CreateUserProcess+0x0x14
kernelBase.dll!CreateProcessInternalW+0x0xFCC
kernelBase.dll!CreateProcessW+0x0x66
kernelBase.dll!CreateProcessWStub+0x0x54
wersvc.dll+0x0xE0B1
wersvc.dll+0x0xAD9A
wersvc.dll+0x0xA28C
wersvc.dll+0x0xA089
ntdll.dll!TppSimplepExecuteCallback+0x0x99
ntdll.dll!TppWorkerThread+0x0x68A
kernel32.dll!BaseThreadInitThunk+0x0x14
ntdll.dll!RtlUserThreadStart+0x0x21
ntdll.dll!ZwOpenProcess+0x0x14
wow64.dll+0x0x4E85
wow64.dll+0x0x902A
wow64cpu.dll+0x0x17C3
wow64cpu.dll+0x0x11B9
wow64.dll+0x0x38C9
wow64.dll+0x0x32BD
ntdll.dll!LdrpInitializeProcess+0x0x1932
ntdll.dll!_ctrlfp$filt$0+0x1932
ntdll.dll!LdrpInitialize0x0x3B
ntdll.dll!LdrpInitializeThunk+0x0xE
ntdll.dll!_NtOpenProcess@16+0x0xC
kernelBase.dll!_OpenProcess@12+0x0x48
Werfault.exe+0x0x3AB99
Werfault.exe+0x0x10B9E
Werfault.exe+0x0x4E77A
kernel32.dll!@BaseThreadInitThunk@12+0x0x19
ntdll.dll!_RtlUserThreadStart+0x0x2F
ntdll.dll!_RtlUserThreadStart@8+0x0x1B
```

Fig. 5. Process injection call stack traces from two Chir family variants.

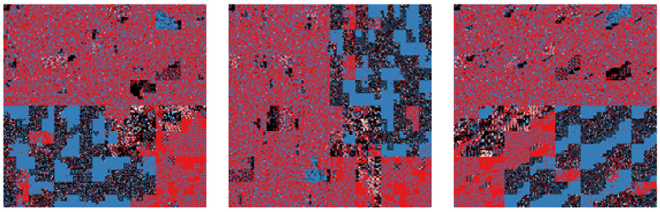


Fig. 6. SFC images of the same Allaple malware sample. (a) Hilbert, (b) Gray-code and (c) Z-order curve implementations.

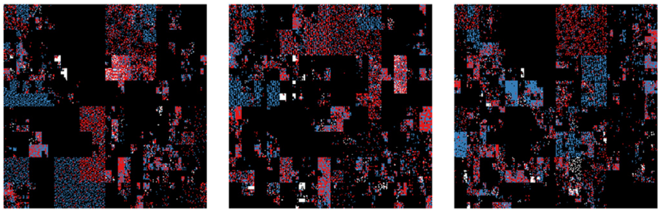


Fig. 7. SFC images converted from process dumps of Cryptowall ransomware samples.

tion of the memory dumps confirmed that the majority of the files comprised of binary zeros.

4.4. Feature extraction

Three feature descriptor algorithms were evaluated, namely Local Binary Patterns (LBP) (Ojala et al., 1996), Gabor filters (Gabor, 1946) and Histogram of Oriented Gradients (HOG) (McConnell, 1986). While the HOG descriptor is primarily used for object detection in images, it has produced favourable results in texture analysis (Demir, 2018; Harb et al., 2017). This section provides an overview of how each descriptor was applied in the context of this work. In the case of each descriptor, a dataset of feature vectors was generated for every image, along with a dataset of corresponding family class labels.

LBPs compute a local representation of texture by comparing and thresholding each pixel with its surrounding neighbourhood of pixels. The original LBP operator forms labels for the image pixels by thresholding the 3 × 3 neighbourhood of each pixel with the centre value and considering the result as a binary number. The histogram of these $2^8 = 256$ different labels are used as a texture descriptor, which can represent the grayscale range of $0 - 255$. In our framework, we used the circular pixel neighbourhood extension to the original LBP operator, proposed by Ojala et al. (2000). The extension operator allows a greater number of neighbourhood points to be considered in the calculation, and thus can capture textures at varying scales of granularity. The main parameters to consider in the LBP descriptor are the radius of the circular neighborhood and the number of data points on the circle's circumference. Values at non-integer pixel coordinates are bi-linearly interpolated which allows any radius and number of pixels in the neighbourhood.

Fig. 8$(a - d)$ shows a schematic of the LBP feature extraction process used in this research. The SFC image in Fig. 8(a) is first converted to grayscale Fig. 8(b). Next, the LBP operator is applied Fig. 8(c), which returns a histogram of features that is constructed from the LBP number values Fig. 8(d). Next, each histogram is added to an array. As each histogram is calculated, it is concatenated to the array. The result is an array of feature histograms, which represent the feature vector for the image.

Gabor filters The Gabor filter is a sinusoidal signal of a given frequency and orientation. Since a single filter is spatially localised, banks of Gabor filters with varying orientations and frequencies are applied to the image as the filter must pass in the same direction as the target texture to return accurate features. In our descriptor we used a Gabor bank of 0–180 at intervals of 45 to ensure

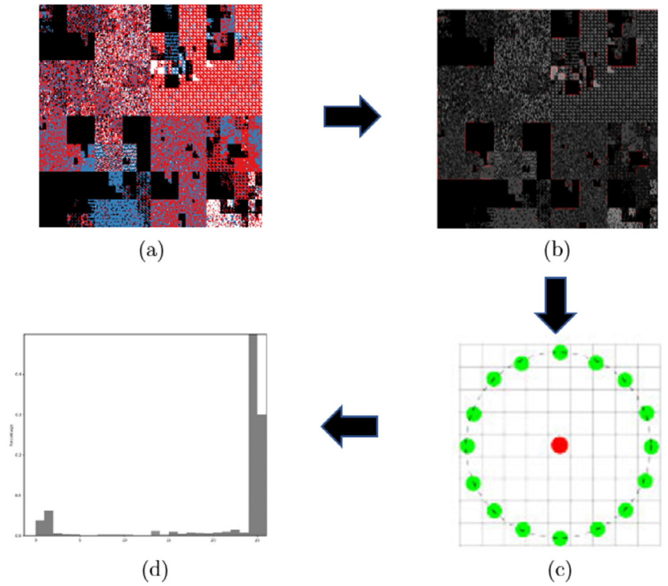


Fig. 8. LBP feature extraction process.

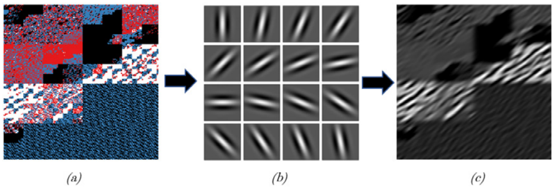


Fig. 9. Gabor filter feature extraction process.

we captured sufficient texture features from varying directions. The main Gabor filter parameters considered in this work were:

- k: is the size in pixels of the Gabor kernel. The value of k is preferably odd and the kernel is a square for the sake of uniformity.
- σ: is the standard deviation of the Gaussian function used in the Gabor filter.
- θ: is the orientation of the normal to the parallel stripes of the Gabor function.
- γ: represents the spatial aspect ratio

Fig. 9(a-c) depicts the Gabor filter process used in this research. The example image is taken from the Emotet banker Trojan family in Hilbert format. The SFC image in 9(a) is input and a series of filters are applied in varying orientations and frequencies 9(b). The resulting filters are then convoluted to produce the final feature vector for the image 9(c), which is appended to the feature array.

Histogram of oriented gradients The Histogram of Oriented Gradients (HOG) feature descriptor counts occurrences of gradient orientation in localized portions of an image. Local object appearance and shape within an image can be characterized by the distribution of intensity gradients or edge directions. Fig. 10 shows the HOG feature extraction process used in our framework. The source SFC image is divided into blocks and each block is sub-divided into cells. The vertical and horizontal gradients are computed for each pixel in a cell. The orientation histogram is then created for each cell. Bins represent the different angles of orientation. The final feature descriptor consists of a concatenation of the cell histograms for each block, which is then flattened into a feature vector. The parameters considered when tuning the HOG descriptor were orientation, pixels-per-cell and cells-per-block. The orientation parameter represents the number of orientation bins in the resulting histograms. Pixels-per-cell determine the size, in pixels, of each cell. Similarly, cells-per-block determine the size of each block.

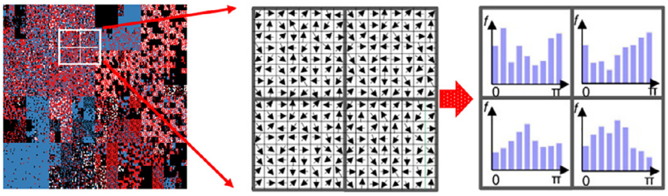


Fig. 10. HOG feature extraction process.

It should be noted here that the optimum parameters for each descriptor were determined through an extensive tuning process, outlined in Section 4.6

4.5. Classification of SFC hybrid features

The next stage in the framework is the classification stage. The aim of classification in this context is to order the malware samples into their taxonomic groups or family classes, based on the textural similarities extracted from the SFC images by the LBP, Gabor and HOG descriptors. Supervised learning algorithms were used exclusively in this research since the class labels, i.e., the malware family names, were already known. The trained classification models were then tested on the 10% holdout validation set, described in previously in Section 4.1, to gauge how well they generalised to previously unseen data. The following classification algorithms were evaluated: Random Forest (RF), Support Vector Machine (SVM) and K-nearest Neighbours (KNN). These classifiers were chosen as previous research by O'Shaughnessy (2019) demonstrated favourable performance results when applied to malware SFC feature datasets. Efforts were made during the classification training phases to reduce the possibility of underfitting and overfitting in the resulting classification models. Overfitting occurs when the model fits the data too well, mainly due to it capturing noise along with the underlying signatures or patterns in the data. A model that is trained to fit slightly inaccurate data can infect it with substantial errors and reduce its predictive power. To reduce the risk of underfitting, a suitably large dataset was compiled. Stratified k-fold cross-validation was used to minimize overfitting. Through experiments, the optimal value for k was found to be 5.

4.6. Parameter tuning

In order to produce optimal classification performance, a complex parameter tuning process was necessary, which involved collectively tuning the feature extraction algorithms with the classifiers. Determining an optimal set of parameters was therefore not a trivial task; there is not a *"one-size-fits-all"* default set of parameters that will optimise performance of the feature descriptors and classifiers combined. A search method was implemented to determine the optimal combination of hyper-parameters, namely GridSearchCV, which is part of the Python Sci-kit Learn library (Sklearn, 2020). With GridSearchCV, all the possible combinations of parameter values are evaluated, and the best combination is retained. Initially, default parameter ranges were chosen and then tested. It was found in some cases, wider parameter ranges were required to produce optimal results. For example, prior research in SVM classifiers tended to use a small value for the penalty parameter C, between 0 and 1. However, it was found that the models

Table 2
Evaluation of image size on processing time and classification performance.

Dims.	Precision	Recall	Accuracy	Processing time (ms)
32	86.2	84.1	84.2	0.5
64	92.8	92.7	92.6	3.8
128	96.8	96.8	96.7	2.28
256	97.7	97.7	97.7	8.1
512	97.9	97.8	97.9	28.3

Table 3
Sample sizes for each dump interval (15 secs to 60 secs).

	Time interval (secs.)			
Family	15	30	45	60
Agen	676	630	520	433
Chir	306	201	132	76
Cryptowall	282	197	99	67
Emotet	492	365	229	164
Scar	304	156	102	83

produced in this work performed better with higher values of 500-10,000.

4.7. Performance metrics

In order to gauge how robust the frameworks classification models were, we tested their performance using precision, recall accuracy. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. In this case, the ratio of correctly predicted samples for a family to the total predicted as that family i.e., *'for all the malware labelled as a particular family, how many were correct?'*. Recall is the ratio of correctly predicted positive observations to the all observations in actual class. In this case, it is the ratio of the correctly predicted malware to the total number for that family, i.e., *'for each malware family, how many that should have been labelled as that family, were labelled correctly?'*. Accuracy is the fraction of correct predictions that the model predicted correctly.

5. Experimental analysis and results

This section presents the experiments devised to investigate the efficacy of the hybrid framework for the classification of malware. The influence of image size on classification was first asserted to produce the optimal dataset for further analysis. The testing of the hybrid framework is discussed next, followed by a comparative analysis with both static and dynamic standalone methods. In the case of standalone static analysis, all samples were converted directly to SFC image format prior to the feature extraction stage. In standalone dynamic analysis, all samples were executed through VMI and their process dumps were converted to SFC image format prior to feature extraction.

5.1. Evaluation of image dimension on classification

The aim of this evaluation was to investigate the influence of varying sizes of SFC images on classification performance and time complexity. The objective was to choose an image size that was feasible in terms of its conversion and processing times, while returning optimum classification performance. The dimensions of an image can influence the number of features produced, which can in turn have an affect on classification performance and processing speed. Our aim was to build a robust framework that was also scalable, so time complexity was an additional consideration. The evaluation was conducted on images ranging from 32 × 32 pixels to 512 × 512 pixels. Processing times for image sizes above 512 became infeasible for both conversion and classification and so are not considered. Table 2 shows the influence of the varying image dimensions on classification performance. From the results, it is evident that as image dimension size increases, performance improves, but at a cost of slower processing times. For the purposes of our further analysis, we chose the 256 × 256 dataset, since it yielded a high classification performance, 97.7% across all three metrics, within a reasonably fast processing time, at 8ms per sample. The 512 × 512 dataset yielded marginally better classification results, but processing time was 3.5 times slower. Furthermore, the conversion process from binary to 512 × 512 image, took approximately 5 seconds per sample, which would not be scalable to large datasets.

5.2. Influence of dynamic runtime on classification performance

The process dump files extracted through VMI represent the behaviour of the malware in execution, i.e., its interaction with the operating system. Therefore, the time interval in which the samples are allowed to execute before the dump is triggered can greatly influence the number of behavioural features returned and consequently, can affect the overall classification performance.

To this end, we determined the optimum running times for each family by executing the obfuscated samples in the virtual machine at intervals ranging from 15 seconds to 60 seconds. Intervals beyond 60 seconds could not be considered scalable to large sample sets and so were not investigated here. It was observed during execution, that many malware samples terminated at different intervals. Closer examination of the dynamic execution revealed the following:

- The Chir, Cryptowall and Scar families each performed code injection as part of their execution routine, which meant the original processes executed by the Procdump tool were terminated. This resulted in the dump being triggered for the interval up until that point and no further behavioural data could be captured.
- In the case of Emotet, it was found that the malware attempted to connect with domains that were no longer in service and the failed connection triggered a kill-switch. Similarly to the previous samples, when the process terminated no further data could be captured.

This resulted in a reduced number of dump samples for each incremented time interval. Table 3 shows the sample size for each family at the varying dump intervals. For example, the original dataset for the Scar variant contained 304 samples, which reduced down to 83 at the 60 second interval.

Table 4 shows the classification results for each of the families tested. For comparative purposes, the results from the static method are also shown. It is evident from Table 4 that, as the sample sizes decrease, so too does classification performance. Smaller sample sizes generally mean there is a narrow range of features for training the classification models. This can negatively influence the generalization capabilities of the models on previously unseen data, which is the case here. The optimum time interval for Chir, Cryptowall, Emotet and Scar was found to be 15 seconds, while Agen performed best at the 30 second interval, shown in bold. The shorter processing intervals mean our method could potentially scale well to larger datasets. Overall, the results demonstrate that the dynamic method provides an improvement in classification performance over the standalone static method.

While a full process trace could be implemented to include the injected process information, it is beyond the scope of this study. For the purposes of further analysis presented in this section, the

Table 4
Influence of process dump time interval on classification performance.

	Static			15			30			45			60		
Family	precision	recall	accuracy	precision	recall	accuracy	precision	recall	accuracy	precision	recall	accuracy	precision	recall	accuracy
Chir	79.2	56.1	65.6	**94.4**	**92.7**	**93.6**	92.5	88.6	90.5	81.8	80.0	80.9	80.9	76.0	78.4
Cryptowall	81.6	57.1	67.2	**90.1**	**91.0**	**90.5**	85.5	75.7	80.3	87.5	46.7	60.9	72.7	64.0	68.1
Emotet	86.9	86.7	85.9	**93.8**	**94.3**	**94.1**	79.1	68.0	73.1	76.2	71.1	73.6	79.1	75.6	77.3
Scar	86.6	76.4	81.2	**93.4**	**89.5**	**91.4**	92.3	68.6	78.7	88.9	53.3	66.7	87.5	35.0	50.0
Agen	82.5	79.1	80.8	99.5	92.6	95.9	**99.1**	**97.7**	**98.4**	98.0	95.2	96.6	97.6	97.1	97.4

Table 5
Hybrid framework classification performance results (training phase).

	Hilbert			Z-order			Graycode		
Model	precision	recall	accuracy	precision	recall	accuracy	precision	recall	accuracy
KNN-LBP	91.9	91.9	91.9	91.2	91.2	91.1	91.2	91.2	91.2
RF-LBP	90.4	90.4	90.3	90.5	90.5	90.4	90.2	90.2	90.2
DT-LBP	86.2	86.2	86.2	85.9	85.9	85.9	85.5	85.5	85.5
KNN-Gabor	92.1	91.4	92.1	92.5	92.5	92.5	93.5	93.2	93.5
RF-Gabor	92.4	92.4	92.3	92.2	88.5	92.2	93.1	93.1	93.0
DT-Gabor	89.0	89.0	88.9	88.0	88.0	88.0	88.3	88.3	88.3
KNN-HOG	94.9	94.9	94.0	**97.6**	**97.6**	**97.6**	93.8	93.2	93.9
RF-HOG	92.0	92.0	92.0	91.5	91.5	91.4	92.0	92.0	92.0
DT-HOG	82.8	82.8	82.7	80.7	80.7	80.7	81.0	81.0	81.0

15 second interval dumps were chosen for Chir, Cryptowall, Emotet and Scar and the 30 second interval dumps for Agen.

5.3. Framework classification training

Experimental analysis was carried out using the Hilbert, Graycode and Z-order image datasets. Each classification model was trained following the same procedure:

- Calculate the Shannon entropy for each executable:
  - If $\mu \geq 6.8$, extract process dump using VMI
- Convert each sample/ process dump to SFC image format.
- Compute the feature vector and label for each image sample.
- Train the classifier on the feature set (perform k-fold cross-validation).
- Compute the average precision, recall and accuracy.

For each of the Hilbert, Graycode and Z-order SFC image datasets, features were extracted using LBP, Gabor and HOG feature descriptors and each feature set was then passed to the SVM, RF and KNN classifiers. In total, 27 classification models were produced, 9 for each SFC dataset. Table 5 shows the training classification performance results for each model. The KNN-HOG model yielded the best training results for all three SFC datasets, with precision, recall and accuracy of 97.6% for each metric. The confusion matrix for the KNN-HOG Z-order model is shown in Fig. 11. It is evident from the confusion matrix that the model yielded favourable results overall on the training data, with true positive rates of 90% or above for 22 out of the 23 classes, including 100% true positive scores in 9 classes. The lowest true positive rate was returned for Locky at 89%.

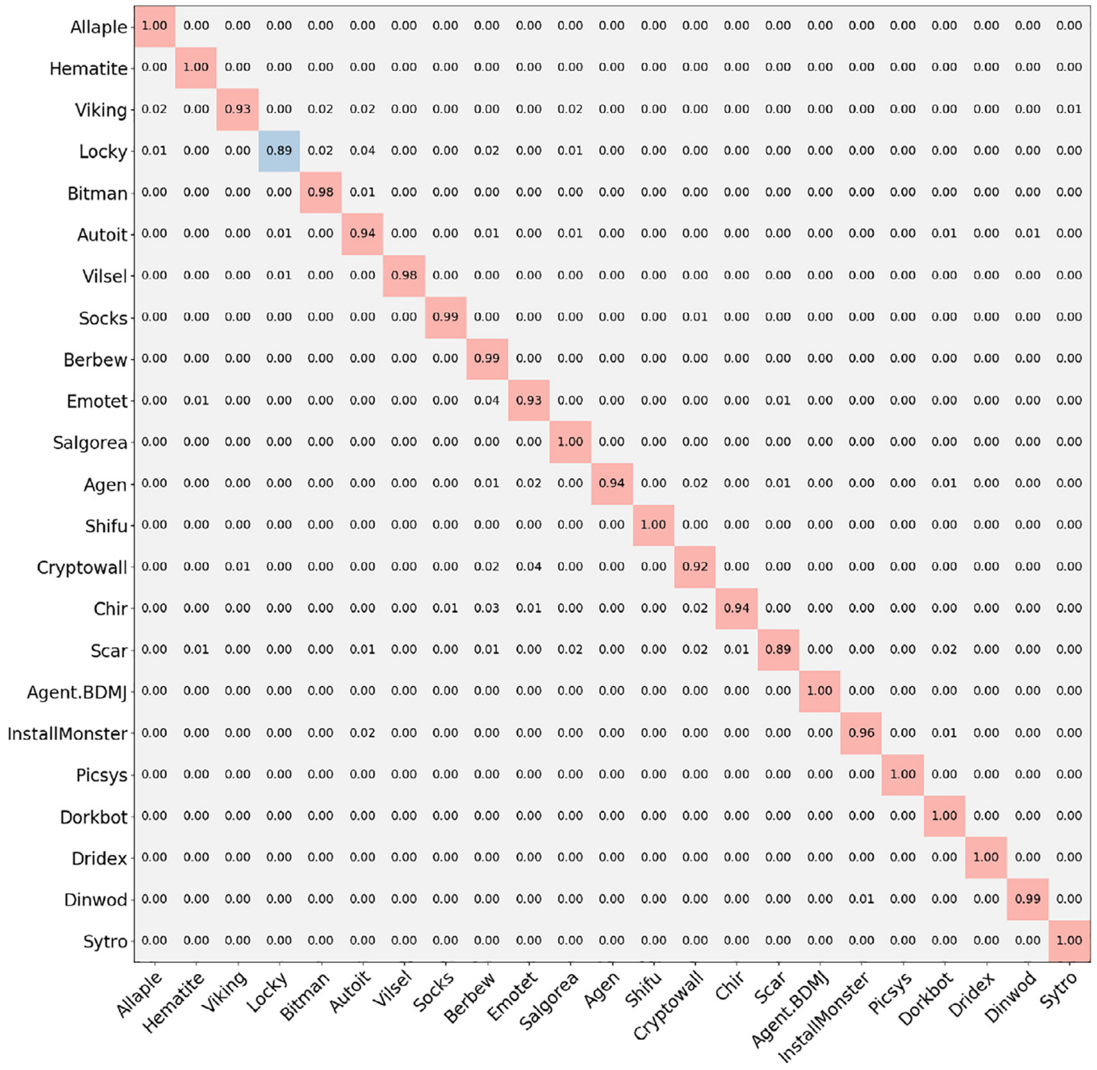
5.4. Hybrid classification model evaluation

In order to build robust classification models, it is crucial to evaluate how well they generalise to new data. Poor models may perform well in the training phase, but when presented with previously unseen data, performance can decrease considerably. As stated previously, 10% of the original data was held for evaluation purposes. As in the training phase, the KNN-HOG Z-order model performed best on the unseen data, with precision, recall and accuracy scores of 97.1%, 96.6% and 97.0% respectively. The performance metrics indicate that the KNN-Model model trained on the Z-order data generalises best to new data, as it can identify variants it has not seen previously with a high degree of classification performance.

5.5. Comparison with static and dynamic standalone approaches

We performed a comparative analysis of the framework against static and dynamic standalone methods to demonstrate the performance advantages of incorporating a hybrid approach. In the case of the static approach, all samples were mapped from non-running executable binaries to SFC image format. For the dynamic method, all samples were executed in a virtual machine and process dumps were collected, which in turn were converted to SFC format. Classification models were trained as before in the case of the hybrid framework. Table 6 shows the results for each approach. For brevity, we only show the best performing models for each of the static, dynamic and hybrid approaches. The KNN-HOG-Graycode model proved to be the best classifier on the training data for the static analysis method. The KNN-HOG-Z-order model performed the best in both dynamic and hybrid methods. Although each of the models performed well in the training phase using cross-validation, the hybrid model outperformed both static and dynamic models, with precision, recall and accuracy scores of 95.1% in each case.

The three models were tested on the 10% holdout evaluation dataset. The KNN-HOG model performed best across all analysis methods. The hybrid model again outperformed both the static and dynamic models, with precision, recall and accuracy scores of 97.1%, 96.6% and 97.0% respectively. In terms of time complexity, the static method provided the fastest processing time, with an average of 7.88ms per sample. The processing overheads are low for the static method, since there is no need to execute the malware in a virtual machine. As expected, the dynamic standalone method was slowest, at an average of 33 seconds per sample, since each malware sample was executed for 30s in the virtual machine to capture sufficient behavioural data in the process dump. The hy-



	Allaple	Hematite	Viking	Locky	Bitman	Autoit	Vilsel	Socks	Berbew	Emotet	Salgorea	Agen	Shifu	Cryptowall	Chir	Scar	Agent.BDMJ	InstallMonster	Picsys	Dorkbot	Dridex	Dinwod	Sytro
Allaple	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Hematite	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Viking	0.02	0.00	0.93	0.00	0.02	0.02	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
Locky	0.01	0.00	0.00	0.89	0.02	0.04	0.00	0.00	0.02	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Bitman	0.00	0.00	0.00	0.00	0.98	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Autoit	0.00	0.00	0.00	0.01	0.00	0.94	0.00	0.00	0.01	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.01	0.00
Vilsel	0.00	0.00	0.00	0.01	0.00	0.00	0.98	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Socks	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Berbew	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Emotet	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.04	0.93	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Salgorea	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Agen	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.00	0.94	0.00	0.02	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00
Shifu	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Cryptowall	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.00	0.00	0.00	0.92	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Chir	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.03	0.01	0.00	0.00	0.00	0.02	0.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Scar	0.00	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.01	0.00	0.02	0.00	0.00	0.02	0.01	0.89	0.00	0.00	0.00	0.02	0.00	0.00	0.00
Agent.BDMJ	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
InstallMonster	0.00	0.00	0.00	0.00	0.00	0.02	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00	0.01	0.00	0.00	0.00
Picsys	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00
Dorkbot	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00
Dridex	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
Dinwod	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.99	0.00
Sytro	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00

Fig. 11. Confusion matrix for KNN-HOG trained on the Z-order SFC dataset.

Table 6
Classification performance and time complexity comparison of the proposed hybrid method against static and dynamic standalone approaches (evaluation phase).

			Training			Evaluation			Time
Method	Model	SFC	precision	recall	accuracy	precision	recall	accuracy	Average (ms)
Static	KNN-HOG	Gray-code	94.5	87.1	91.5	82.3	80.3	83.1	**7.88**
Dynamic	KNN-HOG	Z-order	92.2	90.5	93.2	93.8	92.7	92.2	33 × 10^3
Hybrid	KNN-HOG	Z-order	**95.1**	**95.1**	**95.1**	**97.6**	**97.6**	**97.6**	12.5 × 10^3

brid model averaged 12.5 seconds per sample. The time complexity of the hybrid model is affected by the number of obfuscated or high entropy samples encountered in the dataset.

5.5.1. Resilience to obfuscation

In this section we demonstrate the superiority of the hybrid method against obfuscation by focusing on the performance of the static, dynamic and hybrid analysis on the obfuscated families within the evaluation set. The obfuscated set comprises 205 samples from the Agen (68), Emotet (49), Cryptowall (28), Scar (30) and Chir (30) families. Table 7 shows the results for the three methods against the obfuscated data. It should be noted that these results were based on the performances from the full evaluation set. It is evident that the hybrid model proved to provide the best resilience to the obfuscated samples, outperforming both static and dynamic methods across all families. Although the dynamic method performed comparatively well, the time complexity of the process dump approach makes it infeasible at scale.

5.6. Comparative analysis with benchmark method

It is important to test the efficacy of our method against previous image-based malware classification approaches. The approach chosen for comparative analysis was the KNN-GIST method presented by Nataraj et al. (2011a), described previously in 2.2. This method can be considered a benchmark as it was the first to apply

Table 7
Static, dynamic and hybrid models resilience to obfuscation.

	Static			Hybrid			Dynamic		
Family	precision	recall	accuracy	precision	recall	accuracy	precision	recall	accuracy
Chir	79.2	56.1	65.6	**94.4**	**92.7**	**93.6**	92.5	88.6	90.5
Cryptowall	81.6	57.1	67.2	**90.1**	**91.0**	**90.5**	89.5	88.7	89.3
Emotet	86.9	86.7	85.9	**93.8**	**94.3**	**94.1**	92.3	92.2	93.1
Scar	86.6	76.4	81.2	**93.4**	**89.5**	**91.4**	92.3	88.6	90.7
Agen	82.5	79.1	80.8	**99.1**	**97.7**	**98.4**	93.1	92.7	94.4

Table 8
Classification performance of SFC KNN-HOG vs. byteplot KNN-GIST models.

Method	Prec.	Recall	Acc.
KNN-HOG (train)	97.6	97.6	97.6
KNN-GIST (train)	97.6	97.6	97.5
KNN-HOG (test)	97.1	96.6	97.0
KNN-GIST (test)	96.9	95.9	96.4

the byteplot representation to the classification of malware. Furthermore, several research efforts have based and compared their work with that of Nataraj et al. (Fu et al., 2018; Le et al., 2018; Luo and Lo, 2017; Ronen et al., 2018).

The byteplot and SFC conversion processes differ greatly in their representation of the binary file. The byteplot uses a byte-to-pixel mapping, where each byte value (0–255) is mapped to the equivalent grayscale intensity value (0–255). SFC images are mapped to images through a traversal curve, where data type rather than byte value, determines the mapping. As described previously in Section 3.2, the scurve library uses a colour-coding scheme that classifies bytes (0–255 range) into black for 0 × 00 (0), blue for ASCII text (1–126), red for high-value bytes (127–254), and white for 0×ff (255).

The code for the byteplot KNN-GIST algorithm was obtained from (Nataraj, 2022), which allowed us to reproduce the original method. The VirusTotal dataset was first converted into byteplot representation. Next, GIST features were calculated from the images. Finally, the feature set and labels were passed to a K-nearest neighbor classifier.

Table 8 compares the performance metrics obtained for the KNN-GIST method against our best-performing KNN-HOG method from Section 5.4. The byteplot KNN-GIST model returned training performance scores of 97.6%, 97.6% and 97.5% for precision, recall and accuracy, which is almost identical to our KNN-HOG model's performance. In the testing phase, our KNN-HOG model performed marginally better, with precision, recall and accuracy scores of 97.1%, 96.6% and 97.0% against 96.9%, 95.9% and 96.4% for KNN-GIST. In terms of processing times, the KNN-GIST method produced an average of 6.3ms per sample, which is marginally faster than our KNN-HOG method, which averaged 8.1ms for the 256 × 256 SFC dataset.

5.7. Discussion of results

By varying the image size produced by the scurve library, it was observed that as image size increased, so too did classification performance. However, for the purposes of scalability, which is a requirement of our framework, we chose the 256 × 256 SFC dataset as it provided near optimum classification performance, while being 3.5 times faster than the 512 × 512 samples. Testing the influence of process dump time intervals on classification performance proved the optimum time varied across each family. Process injection and legacy issues with the code prove that the dynamic generation of SFC images has its limitations. Further, more invasive process tracing could be performed, but this would be at the cost of both time and computation complexity.

The results on the evaluation of the static, dynamic and hybrid approaches show that the static method was considerably faster than the dynamic or hybrid methods, yet yielded the poorest classification performance. It was observed that the static model returned lowest prediction results for Cryptowall (61%), Chir (58%) Scar (76%), Emotet (87%) and Agen (87%), which were all obfuscated. The results demonstrate that the static method is not robust against obfuscation. Classification rates were generally higher with the dynamic method, but the slow processing times of 33s per sample make this method infeasible to work at large scale. The hybrid approach processes non-obfuscated samples statically and obfuscated samples dynamically, achieving a per-sample average processing time of 12.5s. Since there is no need to endure the processing burden of executing low entropy samples through VMI, the hybrid framework allows for a considerable decrease in processing time over the dynamic approach alone. For obfuscated samples, the hybrid approach solves the limitation of the static method, increasing classification model performance by approximately 15%. The hybrid framework has shown to be the best solution by combining the strengths of static and dynamic approaches.

In a comparative analysis on the VirusTotal dataset, our SFC KNN-HOG model performed comparably well against the byteplot KNN-GIST model in terms of classification performance and processing time complexity. The results demonstrate our method provides an accurate representation of malware binaries in image format that is comparable to current approaches.

6. Conclusions and future work

In this article, we presented a novel hybrid framework for malware classification, based on computer vision and machine learning, designed to improve the current limitations experienced in static and dynamic malware variant classification methods. The framework is designed to be less invasive than traditional analysis methods in that there is no reverse engineering required, nor does it suffer from the obfuscation limitations of static analysis. The framework incorporates classification of discriminant features extracted from malware binaries in image format, mapped through space-filling curves. Previous research has shown that space-filling curves provide sufficient features to be used for image-based malware classification. Through experimental analysis, using an up-to-date malware sample dataset, it was demonstrated that the hybrid framework outperformed the static and dynamic analyses standalone methods. The methods produced by this research could be used to augment the performance of automated malware analysis tools by increasing the classification rates of variants and thereby reducing the possibility of ambiguity in malware labelling.

Our current research is focused on implementing image segmentation for classification at a more granular level. Segmentation can be used to partition the images into distinct regions containing groups of pixels with similar textures, in this case regions of similar binary. Regions of interest in an executable binary could be extracted in this way, disregarding irrelevant data such as padding (regions of binary zeros) or obfuscated sections, where the entropy would otherwise adversely affect the classification algorithms. Additionally we intend to investigate the efficacy of other SFCs, such as the H-curve (Niedermeier et al., 2002), applied to the malware domain. Finally, deep learning models, such as convolutional neural networks, could be utilised to improve performance. Such models do not require feature extraction and so would simplify the processing phases.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Stephen O'Shaughnessy: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data curation, Writing – original draft, Writing – review & editing, Visualization, Project administration. **Stephen Sheridan:** Writing – original draft, Writing – review & editing, Supervision, Project administration.

References

Anderson, B., Storlie, C., Lane, T., 2012. Improving malware classification: Bridging the static/dynamic gap. Association for Computing Machinery, New York, NY, USA, pp. 3–14. doi:10.1145/2381896.2381900.

AV-Test, 2020. Malware statistics & trends report. https://www.av-test.org/en/statistics/malware/.

Baptista, I., 2018. Binary visualisation for malware detection. The Plymouth Student Scientist 11, 223–237.

Conti, G., Bratus, S., Shubina, A., Lichtenberg, A., Sangster, B., Supan, M., 2010. A visual study of primitive binary fragment types.

Conti, G., Bratus, S., Shubina, A., Sangster, B., Ragsdale, R., Supan, M., Lichtenberg, A., Perez-Alemany, R.,. Automated mapping of large binary objects using primitive fragment type classification 7, S3–S12. doi:10.1016/j.diin.2010.05.002.

Conti, G., Dean, E., Sinda, M., Sangster, B., 2008. Visual reverse engineering of binary and data files. In: Goodall, J.R., Conti, G., Ma, K.-L. (Eds.), Visualization for Computer Security. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–17.

Cortesi, A., 2012. scurve, a space-filling curve library. https://github.com/cortesi/scurve.

Demir, H., 2018. Classification of texture images based on the histogram of oriented gradients using support vector machines. Istanbul University - Journal of Electrical and Electronics Engineering 18, 90–94.

Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C., 2018. Malware visualization for fine-grained classification. IEEE Access 6, 14510–14523. doi:10.1109/ACCESS.2018.2805301.

Fukushima, Y., Sakai, A., Hori, Y., Sakurai, K., 2010. A behavior based malware detection scheme for avoiding false positive. In: 2010 6th IEEE Workshop on Secure Network Protocols, pp. 79–84. doi:10.1109/NPSEC.2010.5634444.

Gabor, D., 1946. Theory of communication. Journal of Institution of Electrical Engineers 93 (3), 429–457. doi:10.1049/ji-3-2.1946.0074.

Garfinkel, T., Rosenblum, M., 2003. A virtual machine introspection based architecture for intrusion detection. In: In Proc. Network and Distributed Systems Security Symposium, pp. 191–206.

Gray, F., 1953. Pulse code communication.

Harb, H.M., Desuky, A.S., Mohammed, A., Jennane, R., 2017. Histogram of oriented gradients and texture features for bone texture characterization. Int J Comput Appl 165 (3), 23–28. doi:10.5120/ijca2017913820.

Hilbert, D., 1891. Über die stetige abbildung einer linie auf ein flächenstück. Mathematische Annalen Zeitschriftenband (1891) 459—-460.

Huang, Y.-K., 2013. Indexing and querying moving objects with uncertain speed and direction in spatiotemporal databases. doi:10.1007/s10109-013-0191-6.

Johnson, B., Shneiderman, B., 1991. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceeding Visualization '91, pp. 284–291. doi:10.1109/VISUAL.1991.175815.

Kaspersky, 2020. The number of new malicious files detected every day increases by 5.2% to 360,000 in 2020. Kaspersky.com. https://www.kaspersky.com/about/press-releases/2020_the-number-of-new-malicious-files-detected-every-day-increases-by-52-to-360000-in-2020.

Khalilian, A., Nourazar, A., Vahidi-Asl, M., Haghighi, H., 2018. G3md: Mining frequent opcode sub-graphs for metamorphic malware detection of existing families. Expert Syst Appl 112, 15–33. doi:10.1016/j.eswa.2018.06.012.

Kolter, J.Z., Maloof, M.A., 2004. Learning to detect malicious executables in the wild. Association for Computing Machinery, New York, NY, USA, pp. 470–478. doi:10.1145/1014052.1014105.

Le, Q., Boydell, O., Namee, B.M., Scanlon, M., 2018. Deep learning at the shallow end: malware classification for non-domain experts 26, S118–S126. doi:10.1016/j.diin.2018.04.024.

Luo, J., Lo, D.C., 2017. Binary malware image classification using machine learning with local binary pattern. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 4664–4667. doi:10.1109/BigData.2017.8258512.

Lyda, R., Hamrock, J., 2007. Using entropy analysis to find encrypted and packed malware. IEEE Security Privacy 5 (2), 40–45. doi:10.1109/MSP.2007.48.

McConnell, R. K., 1986. Method of and apparatus for pattern recognition.

Moon, B., Jagadish, H.V., Faloutsos, C., Saltz, J.H., 2001. Analysis of the clustering properties of the hilbert space-filling curve. IEEE Trans Knowl Data Eng 13 (1), 124–141. doi:10.1109/69.908985.

Moore, E.H., 1900. Errata: "on certain crinkly curves" [trans. amer. math. soc. **1** (1900), no. 1, 72–90; 1500526]. Trans Am Math Soc 1 (4), 507. doi:10.1090/s0002-9947-1900-1500428-3.

Morton, G.M., 1966. A computer oriented geodetic data base; and a new technique in file sequencing. Technical Report (IBM).

Nataraj, L., 2022. Supervised classification with k-fold cross validation on a multi family malware dataset. https://sarvamblog.blogspot.com/2014/08/supervised-classification-with-k-fold.html.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware images: Visualization and automatic classification. In: Proceedings of the 8th International Symposium on Visualization for Cyber Security. Association for Computing Machinery, New York, NY, USA doi:10.1145/2016904.2016908.

Nataraj, L., Yegneswaran, V., Porras, P., Zhang, J., 2011. A comparative assessment of malware classification using binary texture analysis and dynamic analysis. Association for Computing Machinery, New York, NY, USA, pp. 21–30. doi:10.1145/2046684.2046689.

Nguyen, G., Franco, P., Mullot, R., Ogier, J., 2012. Mapping high dimensional features onto hilbert curve: Applying to fast image retrieval. In: Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012), pp. 425–428.

Niedermeier, R., Reinhardt, K., Sanders, P., 2002. Towards optimal locality in mesh-indexings. Discrete Appl. Math. 117 (1–3), 211–237. doi:10.1016/s0166-218x(00)00326-7.

Nivarti, G. V., Salehi, M. M., Bushe, W. K.,. A mesh partitioning algorithm for preserving spatial locality in arbitrary geometries 281, 352–364. doi:10.1016/j.jcp.2014.10.022.

Ojala, T., Pietikäinen, M., Mäenpää, T., 2000. Gray scale and rotation invariant texture classification with local binary patterns. In: Proceedings of the 6th European Conference on Computer Vision-Part I. Springer-Verlag, Berlin, Heidelberg, pp. 404–420.

Ojala, T., Pietikäinen, M., Harwood, D., 1996. A comparative study of texture measures with classification based on featured distributions. Pattern Recognit 29 (1), 51–59. doi:10.1016/0031-3203(95)00067-4.

Oliva, A., Torralba, A., 2001. Modeling the shape of the scene: a holistic representation of the spatial envelope. Int J Comput Vis 42 (3). doi:10.1023/A:1011139631724.

O'Shaughnessy, S., 2019. Image-based malware classification: a space filling curve approach. 2019 IEEE Symposium on Visualization for Cyber Security (VizSec) doi:10.1109/vizsec48167.2019.9161583.

ÓShaughnessy, S., 2019. Sfc datasets. https://github.com/stephen-oshaughnessy/SFC-datasets.

Peano, G., 1890. Sur une courbe, qui remplit toute une aire plane. Mathematische Annalen 36 (1), 157–160. doi:10.1007/bf01199438.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M., 2018. Microsoft malware classification challenge. arXiv:1802.10135.

Russinovich, M., Richards, A., 2020. Procdump - windows sysinternals. https://docs.microsoft.com/en-us/sysinternals/downloads/procdump.

S2-Geometry, G., 2017. S2 cells. https://s2geometry.io/devguide/s2cell_hierarchy.html.

Shannon, C.E., 1948. A mathematical theory of communication. Bell System Technical Journal 27 (3), 379–423. doi:10.1145/584091.584093.

Sklearn, 2020. Sklearn gridsearchcv. https://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html.

Vasan, D., Alazab, M., Wassan, S., Safaei, B., Zheng, Q., 2020. Image-based malware classification using ensemble of cnn architectures (imcec). Computers & Security 92, 101748. doi:10.1016/j.cose.2020.101748.

Vu, D., Nguyen, T., Nguyen, T.V., Nguyen, T.N., Massacci, F., Phung, P.H., 2019. Hit4mal: hybrid image transformation for malware classification. Transactions on Emerging Telecommunications Technologies 31 (11). doi:10.1002/ett.3789.

Wagner, M., Fischer, F., Luh, R., Haberson, A., Rind, A., Keim, D.A., Aigner, W., 2015. A survey of visualization systems for malware analysis. In: Borgo, R. (Ed.), Eurographics Conference on Visualization (EuroVis) ; STARs - State of The Art Reports. The Eurographics Association, pp. 105–125. doi:10.2312/eurovisstar.20151114.

Wattenberg, M., 2005. A note on space-filling visualizations and space-filling curves. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005, pp. 181–186. doi:10.1109/INFVIS.2005.1532145.

Yajamanam, S., Selvin, V., Di Troia, F., Stamp, M., 2018. Deep learning versus gist descriptors for image-based malware classification. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy (2018), pp. 553–561. doi:10.5220/0006685805530561.

Yajamanam, S., Selvin, V.R.S., Di Troia, F., Stamp, M., 2018. Deep learning versus gist descriptors for image-based malware classification. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy vol. 1, ForSE, pp. 553–561. INSTICC. SciTePress. doi: 10.5220/0006685805530561.

S. O'Shaughnessy is a lecturer in Cyber Security at the Technological University Dublin, Ireland, where he delivers modules in Secure Coding, Application Security and Malware Forensics. He is also programme coordinator for the BSc. (Hons.) in Digital Forensics and Computer Security degree programme. His research interests lie in Cyber Security, with a particular focus on applications of machine learning to malware detection and classification.

Stephen Sheridan is a full-time Lecturer in Computing Science at Technological University Dublin (TU Dublin) and is a member of the Security Research group on the Blanchardstown campus. Stephen has been involved in the design and delivery of a wide range of Computer Science programmes at TU Dublin and supervises student projects in Computer Science and Cyber Security at undergraduate and postgraduate levels. His current research interests are in the area of data exfiltration and in particular the use of DNS as a covert channel. The recent focus of Stephen's research work in data Exfiltration over DNS has been in the application of machine learning techniques to detect temporal patterns in DNS traffic that are often a sign of command-and-control activity and the analysis of DNS query strings using statistical techniques in order to identify useful features that can help detect data exfiltration.