Hierarchical Density-based Clustering of Malware Behaviour

Navein Chanderan, Johari Abdullah
Faculty of Computer Science & Information Technology,
Universiti Malaysia Sarawak, 94300, Kota Samarahan, Sarawak, Malaysia.
15020358@siswa.unimas.my

***Abstract*—The numbers and diversity of malware variants grows exponentially over the years, and there is a need to improve the efficiency of analysing large number of malware samples efficiently. To address this problem, we propose a framework for the automatic analysis of a given malware's dynamic properties using clustering technique. The framework also provides outlier discovery, abnormal behaviour analysis and discrimination of malware variants. We also created a module for normalisation of malware labelling based on the labels we get from VirusTotal, which provides consistency of malware labels for accurate analysis of malware family and types. An evaluation model for the proposed framework is also discussed. Ultimately, the proposed framework will ensure rapid analysis of malware samples and lead to better protection for various parties against malicious malware.**

***Index Terms*—Anomaly Detection; Automated Dynamic Malware Analysis; Clustering; Malware Behaviour.**

I. INTRODUCTION

Malicious software, which is also popularly known as malware, is one of the major cybersecurity threats today. In fact, many cybersecurity incidents are usually caused by malware [1]. It comes in various forms, such as viruses, Trojans, worms, botnets, and rootkits, to name a few. Recent report from AV-Test reveals that it registers over 390,000 samples daily [2]. Due to the exponentially growing numbers of malware over the years, a problem that is faced by analysts is large scale malware analysis. The high number of malware samples posed difficulty for analysis as analysts need to extract meaningful information from the samples. To add to this problem, the complexity of modern malware employing evasion techniques such as polymorphism, code obfuscation and metamorphism makes analysis harder. These techniques are effective against static analysis of malware binaries [3]. In contrast, dynamic analysis of malware binaries does not have this limitation for the most part, as these evasion techniques are hard to conceal during run-time. Due to this, there are many researches which focused on dynamic analysis [5-8].

While dynamic analysis is a good approach for analysing malware samples, it does not scale as it is a time-consuming process. It also does not alleviate the problem of exponential malware sample analysis. Therefore, the ability to efficiently and automatically analyse malware behaviour is needed. This is not a new concept as it has been studied and applied before, either by clustering or by classification, usually by applying different algorithms of clustering or classification, and by applying different behaviour representations. The goal of clustering is to discover patterns of similar behaviour and to discover novel malware classes and variants [9-11]. Meanwhile, the goal of classification enables unknown malware variants to be added to existing classes of behaviours [12, 13].

In this paper, we proposed clustering of malware behaviour using hierarchical and density based algorithm (HDBSCAN) to cluster malware samples, and discover unknown variants of malware in an efficient manner.

II. RELATED WORK

Machine learning is employed to automate analysis as it can analyse large number of samples efficiently for the discovery of novel malware, reduce analysis efforts and provide insights into patterns and trends. There are basically two main approaches to machine learning for malware analysis, classification, and clustering. It can be done based on static analysis or dynamic analysis. Based on previous research [10], it has been shown that machine learning based on dynamic analysis gives better results than machine learning analysis based on static analysis, due to the limitations of static analysis.

Since our work focuses on the clustering of malware behaviour for unknown malware, to aid the discovery of unique samples, reduce manual analysis time, and to discover patterns of malware behaviour, we focus on these line of research works. Various methods have been proposed for this purpose with varying level of success. In [10], the authors modelled malware behaviour as a non-transient state changes. Although their technique achieves good results by abstracting higher level calls, the system fails to recognize the relationship between state changes, and thus does not paint a complete picture of malware behaviour, as compared to fine-grained analysis. In [28], the authors use dynamic analysis and machine learning to estimate malware functions, which is useful in identifying the characteristics and behaviour family of malware. The methods and results of this research looks promising.

Bayer et al. [11] proposed a fine-grained malware behaviour analysis. Their framework utilised local sensitivity hashing (LSH) on features extracted, to reduce the number of comparisons during clustering. However, the variable length feature representation makes their approach less scalable. Rieck et al. [23] on the other hand, uses prototype-based clustering to approximate malware behaviour, which reduces the run-time complexity. However, the n-gram approach that the framework uses is susceptible to behaviour obfuscation. In [29], the authors used hybrid deep learning approach to model malware call sequences for classification by combining recurrent neural networks with convolutional neural networks. Using these techniques, the algorithm gets a

hierarchical feature extraction architecture that combines convolution of n-grams with full sequential modelling. The results are good but we argue that it is impractical to run due by many analysts due to the powerful hardware requirements needed.

Labelling of malware samples and clustering of samples based on behaviour are explored in [9]. The authors argued that anti-malware labels by AV vendors are inconsistent, based on the discrepancy clustering results they achieved using self-organizing map (SOM), and the majority vote of labels from antimalware vendors.

III. Challenges in Analysis of Malware Behaviour

Dynamic malware analysis systems can be evaluated based on three main factors, which are efficiency, quality and stealthiness. Malware behaviour analysis is a type of dynamic analysis technique which overcomes the limitations of static analysis. However, it does come with its own set of downsides and challenges. As a start, it is time consuming and resource intensive, thus the efficiency is low, when compared to static analysis methods. This is an unavoidable situation, but the advantages it brings outweigh the disadvantages, as it is able to disclose the natural behaviour of malware. In certain cases, malware samples might perform differently when the sample detects that it is being executed in a virtual environment, and may show artificial behaviour instead of its real behaviour [24].

This is a problem of stealthiness which needs to be taken care of seriously. For any data analysis work, data is the most important resource, as the quality of the data determines the output of the data analysis. Fortunately, there are countermeasures which remedies this up to a certain extent, such as the techniques used in [8]. To ensure that the analysis environment is safe, we use pafish [25] to check and plug vulnerabilities that might interfere with the analysis from producing good results. With this, the analysis in the sandbox environment can be done by reducing, if not eliminating, the triggering the false or undesired behaviours of malware samples.

There is also an issue of analysis quality, and whether the analysis is coarse-grained or fine-grained. Coarse-grained analysis is usually faster but it gives less valuable data and the opposite is true for fine-grained analysis. This is always a constant issue in malware analysis. Faster methods should be researched which can bring a balance between these two approaches. Besides that, the choice of dynamic analysis methods such as bare-metal based or virtual machine based also influences the quality and results.

A malware sample may also contain more than one behaviour branch, but most dynamic analysis tools only observe a single execution path. This might lead to the analysis not showing the sample's true behaviour, which may only be triggered under specific condition. There have been work done to mitigate this problem [5].

IV. Requirements for An Automated Analysis of Malware Behaviour

The main objective of this research is to design and develop a framework which is capable of automated and fast analysis of malware behaviour. The resulting framework should provide the following features to the end users:

A. Fully automated

Malware samples will be automatically analysed in a malware sandbox which is run in a distributed virtual machine environment. The resulting behaviour logs will be processed to be used for further analysis using machine learning algorithm. The results from the analysis will then be provided to the analysts, all without analysts doing the manual analysis on the large number of samples.

B. Minimal false positive

False positive is always an issue with malware detection due to the nature of malware. Some malware variants mimic benign software behaviour thus avoiding detection. By understanding and exploiting the behaviour of malware samples, this can be minimised. Using machine learning algorithms to perform clustering, analysing abnormal behaviours and anomalies, analysts can further inspect analysis data so that false positives can be reduced.

C. Efficient, fast, and timely

A malware analysis framework should not take a long time to perform analysis, because it can impede response time on addressing malware threats. Thus, efficient method must be explored and implemented for fast analysis time without sacrificing quality and accuracy of analysis.

V. Framework for Automatic Analysis of Malware Behaviour

Behaviour of malware can vary from being simple, or it can be very complex by having diverse behaviour. However, more often than not, malware variants of the same family will share common behavioural patterns. Based on this trait, it is possible to exploit this and perform automatic analysis to cluster malware of similar behaviour together. It is also possible to analyse samples and identify anomalies and abnormal behaviour which does not fit into any shared patterns learned.

A diagram of the malware behaviour analysis framework is shown in Figure 1. The general steps are summarised as the following.

1. The framework will execute and monitor malware binaries in a sandbox environment. Based on the behaviour in terms of actions and operations, a report is generated for each binary.
2. Important information in the reports are then extracted, and features will be selected, in terms of spatial and temporal information of behaviours. Spatial information includes the operation and the arguments of API calls, dynamic imports, mutex, processes, filesystem operations, network operations and registry operations while temporal information is the sequence of the actions. The features are then embedded in a vector space.
3. Clustering technique is then applied to the embedded reports which are in vector space, to cluster similar behaviours together, and identify novel malware samples.
4. Report is then generated in file format and visualisation format to help in analysis.

In the following sections, we discuss the individual steps and technical background in detail by explaining how we plan to develop and execute this framework.

A. Malware Behaviour Monitoring

A framework for runtime behaviour-based analysis is required for efficient monitoring of malware behaviour. There exist many tools towards this end, which monitors malware behaviours by intercepting the system calls and logging the execution sequences into log files [6][7]. This contrasts with static based analysis, in which malware binaries are disassembled or debugged. For this research, we choose Cuckoo Sandbox [8]. It is an open source malware sandbox which is widely used and provides good controlled analysis environment for executing malware binaries. The environment is setup behind a firewall to control inbound and outbound traffic, and using VPN to mask the original IP address for security and privacy reasons.

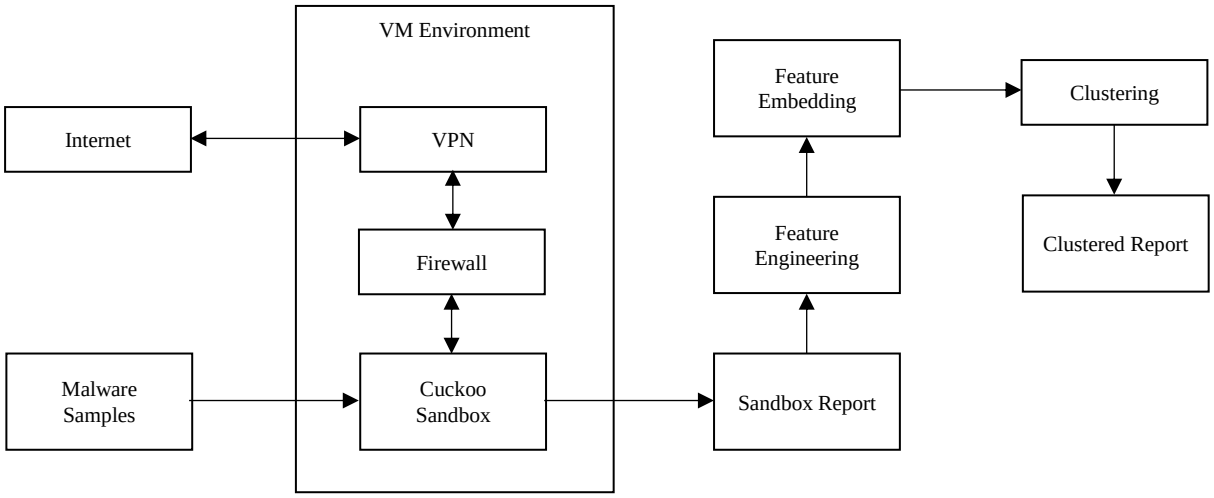

Figure 1: Framework Overview

B. Data set

For testing this framework, we gather our malware samples from three sources: Virus Share [18], Das Malwerk [19], Contagio Dump [20] and Malheur data set [22]. Malheur data set will be used as reference data set. The other data set that are gathered will be used as application data set. These sources are selected for the application data set to provide diverse file types and malware types for our analysis, besides giving us the large number of volume of malware samples collected from various point in time. For the application data set, we have collection of 69,000 unidentified malware binaries.

C. Labelling of malware

Unsupervised learning usually works with unlabelled data. In the case of malware clustering, there is a need to label the data for us to identify malware correctly. However, labelling across Anti-malware systems is not consistent. Due to that, there have been several researches regarding this. [10, 14, 16]. Inspired by these previous works, we have designed a labelling algorithm which reflects the malware more accurately by using CARO naming convention [15] as guideline. This algorithm normalises the prediction of malware labelling provided by VirusTotal [17] by using CARO malware naming scheme:

"Family_Name.Group_Name.Major_Variant.Minor_Variant[:Modifier]"

D. Feature Selection from Behavioural Report

Reports from Cuckoo Sandbox behaviour analysis contain crucial information for analysis. It gives high level information of malware behaviour usually in JSON representation. From our study, we have identified that File system, Windows registry, Process, and Network are critical OS resources, as it represents the chokepoint in a Windows OS. Regardless of whether it is a clean file or a malicious file, every program utilises these resources. We focus on these selected categories of information for this framework.

To fully exploit the information in a behaviour reports based on the resources that we have selected, we look at both spatial and temporal aspect of information. For spatial information, the operation and the arguments of API calls, dynamic imports, mutex, processes, filesystem operations, network operations and registry operations will be extracted. For temporal information, sequence of the actions is also taken into account, as it will generally tell how samples behave. Feature selection is important because it will help keep the dimensionality lower by selecting only relevant information and helps improve clustering performance and results.

Based on the features that were selected, format abstraction is applied. Each API call is map to the value of the call to show the relationship between the operations and its values. The sequence of the operations and its values will also be mapped. Information abstraction is needed as JSON format is not an appropriate format for machine learning, therefore requiring a more suitable representation. Moreover, the complexity of the extracted textual information from JSON reports is high and it will impact the run-time of the algorithm.

E. Feature Embedding

Once the features and its values are selected and abstracted from the reports, the conversion into suitable format for use in vector space model is done. Then, the sequence characterisation of the instructions is performed. It is done by considering the contiguous subsequence of fix-length tokens. The result of this is referred to as w-shingling, an overlapping word-based n-gram. The short behavioural sequence patterns is designed in such a way that it will implicitly capture the program semantic.

The report can then be embedded into a vector space. After embedding the report into vector space, normalisation of the

vector space is performed to remove implicit bias using the technique presented in [4].

F. Comparing Embedded Malware Behaviour

The goal of this framework is to cluster malware with similar behaviour together and give them meaningful label. There are several methods that can be used to group samples together. One way is to measure the similarities between reports, and apply the metric to clustering. The distance metric that is chosen to be used is the Jaccard distance, which is a simple yet powerful technique. The equation of Jaccard distance is;

$$J_d(A,B) = 1 - J_s(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \quad (1)$$

where s is the similarity which is bounded on the interval of [0,1].

G. Clustering of Malware Behaviour using HDBSCAN

Clustering is a basic machine learning technique to group data set into meaningful groups of data, whereby objects which are similar are in the same group while objects which are dissimilar are in different groups. Clustering can be done using the various kinds of clustering algorithms, where different algorithms will vary in its methods to determine the clusters, for example, by calculating the distance or statistical distributions of the cluster members. Generally, clustering algorithms can be grouped into two groups, which is partitional-based clustering or hierarchical-based clustering [26].

Partitional-based clustering algorithm clusters the data by decomposing the data into a set of disjoint clusters. It produces clusters by emphasizing the local structure or the global structure of the by optimising the criterion function. It typically involves minimising the dissimilarity of data within clusters and maximising the dissimilarity of different clusters.

Hierarchical-based clustering algorithm, on the other hand, are usually either agglomerative or divisive. Agglomerative approach starts with each pattern in a singleton, and iteratively merging clusters until conditions are satisfied. Divisive approach starts with a single large cluster and iteratively splitting the data into smaller clusters until conditions are satisfied. While the approach taken to achieve the results are different, they both have similar characteristic. Both produces a dendrogram, a tree-like clustering structure, which shows the nested grouping of patterns, and the similarity levels of the pattern grouping at which grouping level changes.

There have been many techniques and approaches using clustering techniques to classify unknown samples into known malware families, or into unknown malware family, based on the behaviour of malware samples [9-12, 23]. When clustering is applied to malware behavioural report for malware analysis, it allows the learning of malware data structures and the discovery of unknown malware structures. Analysis of code-reuse by comparing malware families can also be done using clustering. Due to this nature of malware, many researchers use hierarchical clustering to exploit this information, as it can capture the level of similarities based on different levels of granularity shown as levels of groups in dendrograms. The dendrogram can also be used to determine the individual clusters of malware families. Another reason many researchers prefer hierarchical clustering is the fact that the number of clusters from the data set is not known and hierarchical clustering algorithm will determine it automatically.

Despite that, hierarchical clustering is sensitive to noise and outliers. It is also unable to handle different sized clusters and convex shaped clusters. On the other hand, density-based algorithm such as DBSCAN does not suffer from noise and outliers as it is robust to it [27]. Another advantage of DBSCAN is that it can find clusters of arbitrary shape, as opposed to many other clustering algorithms. Like hierarchical clustering, DBSCAN does not require the number of clusters to be known a priori. However, DBSCAN is not able to cluster data of different density properly thus making it a limitation.

By utilising the strength of these two types of algorithm, Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) was developed [16]. Its main capabilities are:

1. the number of clusters are calculated automatically
2. ability to handle clusters of different density and shapes
3. ability to handle noise and outliers

We propose that HDBSCAN to be used to solve the problem of having unbalanced and unknown malware dataset. It will be able to handle malware which cannot be added into any existing clusters by treating it as outliers. As such, it lowers the probability that a malware sample will be misclassified. HDBSCAN uses single-linkage clustering technique. To avoid the problem associated with this technique, it transforms the space by arranging the data space such as sparse points are pushed further away and the other points closer. This means that potential noise points are pushed further away in order to keep the results as accurate as possible. It does this by defining the core distance, $core_k(x)$, where (x) is the distance from its k-th nearest neighbour. It then calculates the mutual reachability distance which can be formalised as $\mathrm{d_{mreach\text{-}k}(a,b)=max(core_k(a), core_k(b), d(a,b))}$, where $d(a,b)$ is the metric distance between a and b. From the transformed space, a distance matrix is formed, and a weighted graph can be used to represent the points. Then, the minimum spanning tree (MST) is constructed using Prim's algorithm.

The cluster hierarchy is then calculated from the minimum spanning tree. This is done by sorting the edges by distance from closest to furthest and iterate through. This creates a merged cluster for each edge. The result of this operation is a dendrogram. From this dendrogram, malware family relationship can be viewed. But for large dataset this can be infeasible. The clustering can stop here at this point, but it can still go further by turning the dendrogram into flat clusters automatically, as opposed to DBSCAN which requires the number of clusters to be specified. HDBSCAN first uses the minimum cluster size as a parameter to clean up the dendrogram by condensing it into a smaller tree. Once this is done, the clusters will be extracted by measuring the persistence and stability of the clusters. The persistence of the cluster can be formalised as $\lambda = 1/distance$ where λ_{birth} is defined as the distance at which a cluster's parent split yield a cluster and λ_{death} is defined as the distance at which a cluster splits into sub-clusters. Then the stability of each cluster is calculated for each point within each cluster based on the value defined by λ_p, which is the point at which cluster

splits. The value lies between λ_{birth} and λ_{death}. It can be formalised as $\sum_{p \in cluster}(\lambda_p - \lambda_{birth})$.

The leaf nodes are then declared as selected clusters. The algorithm works in reverse topological sort order to work through the tree to calculate the sum of stability. If the sum of the sub-cluster is greater than the cluster, then the stability of the cluster is set to the sub-cluster's stabilities. On the other hand, if the sum of the cluster stability is more than the sum of the sub-cluster's stability, then the cluster to be a selected cluster. The sub-cluster will then be part of the cluster. Once the operation reaches the root node, then the clusters are returned. Any points which are not selected are considered noise points. The pseudocode for HDBSCAN is shown in Figure 2.

```
1. Transform the space according to the
   density/sparsity.
   a) Calculate core distance
   b) Spread points with differing density,
      calculate mutual reachability
      distance

2. Build the minimum spanning tree of the
   distance weighted graph.
   a) Draw vertices for data points with
      weighted core points as edge

3. Construct a cluster hierarchy of
   connected components.
   a) Convert minimum spanning tree into
      cluster hierarchy (dendrogram)

4. Condense the cluster hierarchy based on
   minimum cluster size.
   a) Calculate minimum cluster size
   b) Determine cluster membership

5. Extract the stable clusters from the
   condensed tree.
   a) Compute cluster stability
   b) Split clusters
   c) Data points which are not part of any
      clusters are considered noise
```

Figure 2: Pseudocode for HDBSCAN

H. Report of Clustered Behaviour

The results of the clustering will be saved in a file, e.g.: CSV format file, for storing the results. The results will contain the number of clusters generated, the properties of each clusters and the anomaly which are detected during the clustering. Anomalies in this case mean samples which cannot be clustered due to the number of similar samples being too low to be clustered. Visualisation techniques will also be used to display the results generated.

VI. EVALUATION

Evaluating the quality of clustering results is an inherently difficult task. Therefore, to evaluate the correctness of the clustering, a reference data set of known malware samples from Malheur is obtained [22]. We will compare the results from the Malheur reference data set using labels by major anti-malware companies and results from our framework.

Although anti-malware labels suffer from inconsistencies in naming, selecting vendors which follows CARO naming convention for samples would produce consistent and accurate results. Once the framework has been evaluated and calibrated, the application data set can then be analysed. The results of the application data set will be discussed after the framework has been completed.

To evaluate our framework based on the reference data set, we use the evaluation metrics of precision and recall. Precision reflects the agreement between malware classes and individual clusters, while recall reflects the extent of classes scattering across clusters. Precision for the set of cluster C can be formally defined as;

$$P = \frac{1}{n} \sum_{c \in C} \#_c \tag{2}$$

where $\#_c$ is the largest number of reports in cluster c sharing the same class. Recall for the set of malware classes M can be formally defined as;

$$R = \frac{1}{n} \sum_{m \in M} \#_m \tag{3}$$

where $\#_m$ is the largest number of reports labelled m within a cluster.

The ideal case would be to have a 100% precision and 100% recall value. However, in most cases, there will be a trade-off between precision and recall, as the threshold value needs to be set to determine the probability of the membership of malware classes. Thus, experiments will be conducted to find the best possible threshold value to get the best possible result based on precision and recall.

A comparison of the analysis results of our framework against the framework created by [23] and [9] using the application data set will be done.

VII. CONCLUSION & FUTURE WORKS

The exponentially increasing malware threats in numbers and complexity means that there needs to be a way to efficiently analyse the large number of samples efficiently. This paper proposed a method to efficiently analyse malware in a scalable manner using clustering technique which is based on hierarchical structure and density. The methods used is designed to provide important analysis results such as common cluster behaviours, and this can assist in the creation of compact signatures and knowledge base. It can also be used for further manual analysis if required.

Currently research is still being done on this framework. After completion, the framework will have additional capability to detect anomaly in the samples analysed and be capable of abnormal behaviour analysis. A comparison will be done with the state of art to compare and verify its effectiveness.

ACKNOWLEDGEMENT

This research is funded by Universiti Malaysia Sarawak's Research and Innovation Management Centre (RIMC) under Geran Penyelidikan: F08/SpFRC/1432/16/6. We would like to thank Universiti Malaysia Sarawak for their support.

REFERENCES

[1] "Internet Security Threat Report," 2016. Available: https://www.symantec.com/content/dam/symantec/docs/ report/istr-21-2016-en.pdf

[2] Av-test.org, "AV-TEST – The Independent IT Security Institute," 2016. Available: http://www.av-test.org/en/statistics/malware

[3] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in Computer Security Applications Conference, 2007, ACSAC 2007, Twenty-third annual, IEEE, 2007, pp. 421-430.

[4] K. Rieck, and P. Laskov, "Linear-time computation of similarity measures for sequential data," Journal of Machine Learning Research 9, Jan 2008, pp. 23-48.

[5] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in Proceedings of the 2007 IEEE Symposium on Security and Privacy, IEEE, 2007, pp. 231-245.

[6] C. Willems, T. Holz, and F. Freiling, "CWSandbox: Towards automated dynamic binary analysis," IEEE Security and Privacy 5, no. 2, 2007, pp. 32-39.

[7] A. Dinaburg, P. Royal, M. Sharif and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in Proceedings of the 15th ACM conference on Computer and communications security, ACM, 2008, pp. 51-62.

[8] C. Guarnieri, A. Tanasi, J. Bremer, and M. Schloesser, The Cuckoo Sandbox, 2012.

[9] R. S. Pirscoveanu, M. Stevanovic and J. M. Pedersen, "Clustering analysis of malware behavior using Self Organizing Map," in 2016 International Conference On Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), London, 2016, pp. 1-6.

[10] M. Bailey, J. Oberheide, J. Andersen and Z. M. Mao, "Automated classification and analysis of internet malware," in International Workshop on Recent Advances in Intrusion Detection, Springer Berlin Heidelberg, 2007, pp. 178-197.

[11] U. Bayer, P. M. Comparetti, C. Hlauschek and C Kruegel, "Scalable, Behavior-Based Malware Clustering," in NDSS, vol. 9, Feb 2009, pp. 8-11.

[12] K. Rieck, T. Holz, C. Willems, P. Düssel and P. Laskov, "Learning and classification of malware behaviour," in International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Springer Berlin Heidelberg, 2008, pp. 108-125.

[13] H. S. Galal, Y. B. Mahdy and M. A. Atiea, "Behavior-based features model for malware detection," in Journal of Computer Virology and Hacking Techniques 12, no. 2, 2016, pp. 59-67.

[14] R. Perdisci, "VAMO: towards a fully automated malware clustering validity analysis," in Proceedings of the 28th Annual Computer Security Applications Conference, ACM, 2012, pp. 329-338.

[15] CARO - Computer Antivirus Research Organization, "A New Virus Naming Convention (1991)". Available: http://www.caro.org/articles/ naming.html

[16] R. J. G. B. Campello, D. Moulavi and J. Sander, "Density-based clustering based on hierarchical density estimates," in Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, 2013, pp. 160-172.

[17] VirusTotal. Available: https://www.virustotal.com/

[18] VirusShare.com, "VirusShare.com," 2016. Available: http://virusshare.com/

[19] Dasmalwerk.eu, "DAS MALWERK," 2016. Available: http://dasmalwerk.eu/

[20] Contagiodump.blogspot.com, "contagion," 2016. Available: http://contagiodump.blogspot.com/

[21] M. Chandramohan, H. B. K. Tan and L. K. Shar, "Scalable malware clustering through coarse-grained behavior modeling," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, ACM, 2012, p. 27.

[22] K. Rieck, "Malheur Dataset". Available: https://www.sec.cs.tu-bs.de/ data/malheur/

[23] K. Rieck, P. Trinius, C. Willems and T. Holz. "Automatic analysis of malware behavior using machine learning," in Journal of Computer Security 19, no. 4, 2011, pp. 639-668.

[24] C. Kolbitsch, E. Kirda, and C. Kruegel. "The power of procrastination: detection and mitigation of execution-stalling malicious code," in Proceedings of the 18th ACM conference on Computer and communications security, ACM, 2011, pp. 285-296.

[25] Pafish, "a0rtega/pafish: Pafish is a demonstration tool that employs several techniques to detect sandboxes and analysis environments in the same way as malware families do," 2017. Available: https://github.com/a0rtega/pafish

[26] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," in ACM computing surveys (CSUR) 31, no. 3, 1999, pp. 264-323.

[27] M. Ester, H. P. Kriegel, J. Sander, and X. W. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in Kdd, vol. 96, no. 34, 1996, pp. 226-231.

[28] N. Kawaguchi, and K. Omote, "Malware function classification using APIs in initial behavior." in 2015 10th Asia Joint Conference on Information Security (AsiaJCIS), IEEE, 2015, pp. 138-144.

[29] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep Learning for Classification of Malware System Call Sequences," in Australasian Joint Conference on Artificial Intelligence, Springer International Publishing, 2016, pp. 137-149.