electronics

MDPI

Article

Malware Variant Identification Using Incremental Clustering

Paul Black *, Iqbal Gondal, Adil Bagirov and Md Moniruzzaman

Internet Commerce Security Laboratory (ICSL), Federation University, P.O. Box 663, Ballarat 3353, Australia; iqbal.gondal@federation.edu.au (I.G.); a.bagirov@federation.edu.au (A.B.); m.moniruzzaman@federation.edu.au (M.M.)

* Correspondence: p.black@federation.edu.au

Abstract: Dynamic analysis and pattern matching techniques are widely used in industry, and they provide a straightforward method for the identification of malware samples. Yara is a pattern matching technique that can use sandbox memory dumps for the identification of malware families. However, pattern matching techniques fail silently due to minor code variations, leading to unidentified malware samples. This paper presents a two-layered Malware Variant Identification using Incremental Clustering (MVIIC) process and proposes clustering of unidentified malware samples to enable the identification of malware variants and new malware families. The novel incremental clustering algorithm is used in the identification of new malware variants from the unidentified malware samples. This research shows that clustering can provide a higher level of performance than Yara rules, and that clustering is resistant to small changes introduced by malware variants. This paper proposes a hybrid approach, using Yara scanning to eliminate known malware, followed by clustering, acting in concert, to allow the identification of new malware variants. F_1 score and V-Measure clustering metrics are used to evaluate our results.

Keywords: malware clustering; Yara rules; malware variant identification; incremental clustering

Citation: Black, P.; Gondal, I.; Bagirov, A.; Moniruzzaman, M. Malware Variant Identification Using Incremental Clustering. *Electronics* **2021**, *10*, 1628. https://doi.org/10.3390/electronics10141628

Academic Editor: Taeshik Shon

Received: 4 June 2021
Accepted: 5 July 2021
Published: 8 July 2021

1. Introduction

This paper provides a technique called Malware Variant Identification, using Incremental Clustering (MVIIC). A sandbox is an instrumented virtual machine that executes malware samples and other programs, and gathers dynamic analysis features resulting from program execution. These features include filesystem activity, registry activity, network traffic, program execution, and code injection. Two common sandboxes are Cuckoo [1] and CWSandbox [2]. Yara is a pattern matching technique that can use sandbox memory dumps for the identification of malware families. Yara rules contain regular expressions and strings [3]. Yara rules may fail to identify new malware variants when software development modifies code corresponding to the Yara regular expressions, or when program strings are changed. These unidentified malware samples provide a valuable source of unknown malware families and new malware variants.

While the clustering of dynamic analysis features has previously been used for malware detection [4,5], this paper proposes a hybrid scheme using Yara rules, and a novel incremental clustering algorithm [6] to enable the identification of new malware families and malware variants.

In this research, malware identification is performed in two layers. In the first layer, the previously developed Yara rules are used to reject samples of known malware families. A novel incremental clustering algorithm is applied in the second layer. Unlike traditional clustering algorithms that are sensitive to the choice of starting cluster centers, incremental clustering is able to find or approximate the best cluster distribution and detect small size clusters. This is important in malware clustering when new data (unidentified malware samples) become available, as such data generate separate and small clusters. Therefore, this incremental clustering algorithm is well suited for use in a malware analysis system,

where new samples are reviewed on a periodic basis, and new malware variants can initially become visible in small numbers due to initial testing before becoming available in large numbers, due to wide-scale malware deployment.

The usage scenario for this research is a malware analysis system, where new malware samples are received on a periodic (daily) basis and processed in a sandbox.

Yara rules identifying known malware variants are used to scan the sandbox memory dumps. These Yara scans are used to identify known malware variants for further analysis. However, the unidentified malware samples represent a valuable source of new malware families and new malware variants. Random sampling from the clusters can then be used for the identification of new malware families and variants. As an example of this use case, consider a situation where a malware author is testing a new malware product and a few samples of this new malware are received in the analysis system; however, the clusters are small and may not be selected for analysis. As this new malware product becomes used in the wild, the number of samples received increases, larger clusters are generated, and they are finally selected for analysis.

Yara rules employ regular expressions for the identification of malware families. These rules are widely used in industry for malware identification, and have the following characteristics:

- They are readily created.
- Testing allows the elimination of obvious false positives.
- They are excellent for the matching of machine code, program headers, metadata, or strings in programs.
- Detection may fail, due to code changes resulting from malware development, or recompilation.
- When Yara rules fail, no warning is given.

MVIIC provides a key component of the intelligent malware analysis process shown in Figure 1. In this scenario, dynamic analysis of samples from a malware feed (1) is performed, using a Cuckoo sandbox. Yara rules are used to scan the memory dump (2) from the sandbox. Clustering is performed using the dynamic analysis features (4) of each unidentified malware sample (3,5). New malware families (6) and malware variants (6) contained in a random selection of malware samples from each cluster are identified, and new Yara rules (8) are created, using the memory dumps (7) of the clustered malware samples.

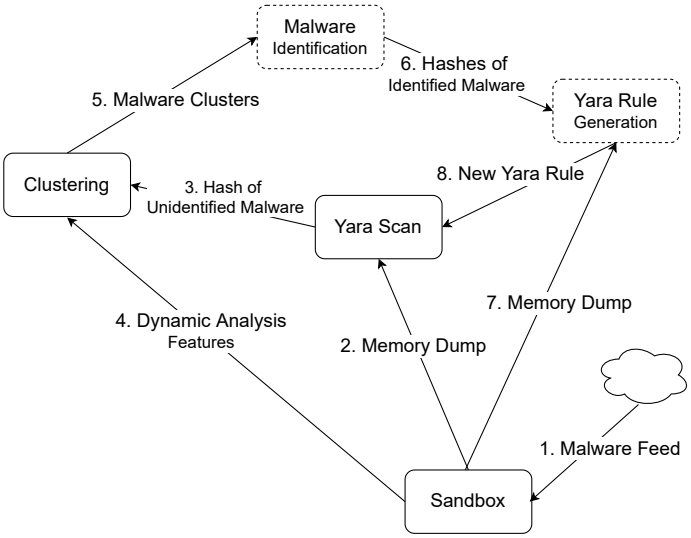

Figure 1. Analysis concept.

This paper makes the following contributions:

- Provides a use case for a novel incremental clustering algorithm.
- Provides feature engineering of features derived from the dynamic analysis logs of malware samples captured from the live feed.
- Develops a two-layered dynamic analysis system that uses Yara rules to reject known malware, and uses a novel clustering algorithm and feature engineering to perform clustering of unidentified samples to enable the identification of new malware families, variants, and deficiencies in the Yara rules.

The structure of this paper is as follows: Section 2 presents related work; Section 3 provides the problem definition and overview; Section 4 describes our approach; Section 5 provides an empirical evaluation of the new approach; Section 6 provides discussion and limitations; and Section 7 presents the conclusion.

2. Related Work

2.1. Malware Classification Using Machine Learning

Machine learning is widely used in malware detection, classification, and clustering. Common feature extraction techniques are used in each of these activities [7]. Malware feature extraction can be performed using dynamic or static analysis. Feature extraction using dynamic analysis is often performed by running a malware sample in a sandbox. The sandbox extracts behavioral features of the malware execution. These behavioral features include API call traces, filesystem and registry activity, network traffic, memory, instruction, and register traces. A further type of feature may be created by representing malware bytes as gray-scale images. The main limitations of behavioral analysis are malware detection of the analysis environment and the limited program path that a malware sample executes without command and control inputs. Static analysis provides features that are extracted from the malware sample, and includes the function call graph, control flow graphs, API function calls, instruction statistics, graphical representations, strings, byte representations, entropy measurements, and hashes. Malware obfuscation, including packing (compression or encryption of the original malware binary), is used to hinder static analysis [7].

Traditional machine learning techniques, such as support vector machine (SVM) or random forest, require time-consuming feature engineering for the design of the machine learning model. Deep learning architectures do not require feature engineering, and provide a trainable system that automatically identifies features from the provided input data. Deep learning algorithms include the convolutional neural network, residual network, autoencoder, recurrent neural network, short-term memory network, gated recurrent unit network, and neural network [7].

2.2. Malware Clustering

A malware clustering and classification system using dynamic analysis was developed by Rieck et al. [8]. In this research, API call names and call parameters are extracted using CWSandbox. The API call names and parameters are encoded into a multi-level representation called the Malware Instruction Set (MIST) [8]. Malware samples of a specific family frequently contain variants with a high degree of behavioral similarity, providing a dense vector space representation. This research study used prototype clustering to represent the dense groups. The use of a prototype representation accelerated clustering times by reducing the machine learning computation. The clustering experiments used hierarchical clustering with a Euclidean distance measure; features were taken from 33,698 malware samples, and provided a maximum F_1 score of 0.95.

A study of the clustering of malware samples using dynamic analysis is provided by Faridi et al. [9]. This research used a data set of 5673 Windows malware samples. The ground truth was determined by using the Suricata Intrusion Detection System (IDS) alerts [10] to classify the malware samples based on the network traffic generated from the execution of each malware sample. This led to a manually verified identification of 94 malware clusters. The clustering features were extracted from the Cuckoo sandbox file,

registry keys, services, commands, API names, and mutex reports. Features present in less than two samples were dropped to reduce the outliers. The cluster number was estimated, using the gap statistic [11]. Clustering was performed, using the following algorithms: DBSCAN, K-Means++, Spectral, Affinity propagation. The features were derived from the Term Frequency Inverse Document Frequency (TF/IDF) matrix of the behavioral data extracted from the Cuckoo sandbox reports. The following pairwise distance functions were used in calculating the TF/IDF matrix: Cosine7, Cityblock, Euclidean, Hamming, Braycurtis, Canberra, Minkowski, and Chebyshev. The performance of density-based, hierarchical, and prototype clustering was evaluated. The metrics that were used to evaluate the clustering performance are as follows: adjusted mutual information score, adjusted Rand score, homogeneity, completeness, V-measure, G-means (geometric mean of precision and recall), and silhouette coefficient. This paper notes that precision and recall are, in general, sensitive to the number of clusters calculated, while homogeneity and completeness are not. Hierarchical clustering with a Bray–Curtis distance measure provided the best performance with 107 clusters and a V-measure of 0.921 in 195.71 s [9].

2.3. Incremental Clustering

Internet security organizations often maintain a knowledge base of malware family behaviors. When a new malware sample is received, the malware knowledge base is used for malware classification, which identifies whether the malware sample is a variant of an existing family, or if it is a new malware family that requires reverse engineering to understand its behaviors. Clustering of malware features can be used in the identification of malware families. However, samples in a malware feed are subject to an ongoing concept drift, due to the software development in existing malware families and the creation of new malware families. Traditional clustering algorithms are designed to operate with static ground truth. To deal with concept drift, traditional clustering algorithms require that the whole data set be periodically re-clustered to incorporate the updated ground truth. The execution time of traditional clustering algorithms increases in proportion to the data set size. The problems of concept drift and data set size can be addressed by a two-layered solution, where clustering is used to identify new malware families and a classifier is then trained with the updated malware families. Difficulties with this approach lie in determining an optimal batch size and the retraining interval. MalFamAware [12] performs both the identification of new malware families and malware family classification through the use of online clustering, and removes the cost of periodic classifier retraining. MalfamAware makes use of the BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies) incremental clustering algorithm. MalFamAware extracts static and dynamic features, using the Cuckoo sandbox. MalFamAware uses the following two phases: tuning, and family identification and classification. In the first phase, BIRCH is tuned using the existing ground truth; in the second phase, BIRCH is used to perform malware classification and new family identification. MalFamAware testing was performed, using a data set of 18 malware families with a total of 5351 malware samples. In the evaluation of MalFamAware, the BIRCH algorithm provided the best performance with an F1 score of 0.923 [12].

2.4. Yara

Yara rules were developed by Victor Alvarez of VirusTotal [13] for the identification and classification of malware samples [14]. They consist of strings and regular expressions [15] that are used to match the program data. They are relatively simple and can be easily created. Yara rules represent patterns and other data in programs, including malware.

These patterns are dependent on the toolchain used to build the program, and can change whenever the program is rebuilt or when software development is performed. Yara rules will fail to identify new malware variants where software development or recompilation have resulted in modifications to machine code corresponding to previous Yara rules. This may result in a failure to identify new malware variants or new malware families.

3. Malware Variant Identification Using Incremental Clustering

There are two classes of incremental algorithms in clustering. The first class contains algorithms that add data points incrementally and update clusters accordingly (e.g., BIRCH [16]). Algorithms from the second class construct clusters incrementally. In order to compute the $k > 1$ clusters, these algorithms start by calculating one cluster, which is the whole data set, and gradually add one cluster at each iteration. Our algorithm belongs to the second class of incremental algorithms. The following section provides a summary of this incremental clustering algorithm.

Incremental Non-Smooth Clustering Algorithm

Clustering is an unsupervised partitioning of a collection of patterns into clusters based on similarity. Most clustering algorithms can be classified as hierarchical or partitional. We consider partitional clustering algorithms. These algorithms find the partition that optimizes a clustering criterion [6]. Next, we briefly describe the non-smooth optimization formulation of the clustering problem used in this paper.

Assume that A is a finite set of points in the $n-$dimensional space $\mathbb{R}^n$, that is $A = \{a^1, \ldots, a^m\}$, where $a^i \in \mathbb{R}^n, i = 1, \ldots, m$. The hard clustering problem is the distribution of the points of the set A into a given number k of pairwise disjoint and nonempty subsets $A^j, j = 1, \ldots, k$ such that

$$A = \bigcup_{j=1}^{k} A^j \tag{1}$$

The sets $A^j, j = 1, \ldots, k$ are called clusters, and each cluster A^j can be identified by its center $x^j \in \mathbb{R}^n, j = 1, \ldots, k$. The problem of finding these centers is called the k-clustering (or k-partition) problem. *The similarity measure* is defined using the squared L_2 norm:

$$d_2(x, a) = \sum_{i=1}^{n} (x_i - a_i)^2 \tag{2}$$

In this case, the clustering problem is also known as *the minimum sum-of-squares clustering problem*. The non-smooth optimization formulation of this problem is [6,17] as follows:

$$\text{minimize } f_k(x) \text{ subject to } x = (x^1, \ldots, x^k) \in \mathbb{R}^{nk} \tag{3}$$

where

$$f_k(x^1, \ldots, x^k) = \frac{1}{m} \sum_{a \in A} \min_{j=1,\ldots,k} d_2(x^j, a) \tag{4}$$

One of the well-known algorithms for solving the clustering problem is the k-means algorithm. However, it is sensitive to the choice of starting cluster centers and can find only local solutions to the clustering problems. Such solutions in large data sets may significantly differ from the global solution to the clustering problem.

To enable an effective clustering process, we apply the incremental non-smooth optimization clustering algorithm (INCA), introduced in [18] (see also [6]). This algorithm computes clusters gradually, starting from one cluster. It involves the procedure to generate starting cluster centers; the procedure is described in [6]. This algorithm is accurate and efficient in data sets with a large number of features, and can find either global or near-global solutions to the clustering problem (3). In INCA this problem is solved using the nonsmooth optimization method, the discrete gradient method [19]. In the implementation of the INCA algorithm, we apply the following stopping criteria:

1. $\overline{f} < \varepsilon$, where $\overline{f} = \frac{f_{i-1} - f_i}{f_{i-1}}$, $i \geq 2$, f_i is the value of the clustering function obtained at the i-th iteration of an incremental algorithm and ε is a user defined tolerance.
2. $n_c > C$, where n_c is the number of clusters and C stands for the maximum number of clusters.

The parameter ε is user defined. This parameter is defined using the cluster function value at the first iteration and the number of records: $\varepsilon = f_1/m$. The incremental non-smooth clustering algorithm is shown in Algorithm 1.

Algorithm 1 Incremental non-smooth clustering algorithm

Require: Cluster count k > 0
Require: Finite point set $A \subset \mathbb{R}_n$
Calculate set A center $X_1 \in \mathbb{R}_n$
$L = 1$
while $L < k$ or $f_{L-1} - f_L < \varepsilon$ **do**
$L = L + 1$
Find starting point $\bar{y} \in \mathbb{R}_n$ by solving the auxilarity clustering problem for L^{th} cluster
Find $\tilde{y} \in \mathbb{R}_{nL}$ by solving clustering problem starting from $(x_1, \ldots, x_{L-1}, \bar{y})$
Set solution of the L-partition problem $x_j = \tilde{y}_j, j = 1, \ldots, L$ and find the value f_L of the cluster function f at this solution.
end while

4. Experimental Methodology

The use of the incremental clustering algorithm allows the computation of accurate solutions to the clustering problem to consider at once cluster distributions for different numbers of clusters, and to cluster unidentified malware samples.

4.1. Feature Engineering

The following features were extracted from the analysis reports following the dynamic analysis in a Cuckoo sandbox [1] running on a Windows 7 virtual machine. Common operating system related hostnames were excluded.

- API Call names.
- API call frequencies.
- DNS A record lookup requests.

Previous machine learning research achieved good performance, using API call frequency histograms [20,21], while other research indicated good performance associated with network and HTTP features [9], including hostnames. As a result, it was decided to use API call frequency and DNS hostname features. Two types of feature encoding were performed: histogram encoding, and TF/IDF encoding. The histogram was built, using call counts for each unique API call, and DNS lookup requests for each unique hostname. The TF/IDF encoding created vectors from the unique API call names and unique DNS hostnames.

4.2. Clustering

Clustering was performed, using INCA [6]. The evaluation of feature encoding was performed, using both histogram features and TF/IDF encoding. Feature engineering experiments were performed to identify the configuration that provided the highest accuracy.

4.3. Clustering Metrics

The metrics applied for the clustering results presented from this research are the F_1 score, V-measure, and cluster purity. The F_1 score is calculated using precision and recall, which are calculated from the *TP*, *TN*, *FP*, and *FN* counts. *Precision* and *recall* [22] are defined in Equations (5) and (6), respectively.

$$Precision = TP/(TP + FP) \tag{5}$$

$$Recall = TP/(TP + FN) \tag{6}$$

The F_1 score (F_1) [22] defined in Equation (7) is used to assess the quality of the results in terms of *precision* and *recall*.

$$F_1 = 2 \times (Precision \times Recall)/(Precision + Recall) \tag{7}$$

The V-measure is an entropy-based external cluster evaluation measure. This measure is based on two criteria: homogeneity and completeness. It determines the degree of similarity between a clustering distribution and a class distribution. Homogeneity of a set of clusters means that each cluster from this set contains data points only from a single class. Completeness of a set of clusters means that data points from a given class are elements of the same cluster. Given a set of classes $C = \{C_i : \ i = 1, \ldots, p\}$ and a set of clusters $K = \{A_j : \ j = 1, \ldots, k\}$ the V-measure is defined as follows:

$$V_\beta = \frac{(1+\beta)hc}{\beta h + c} \tag{8}$$

Here, homogeneity h is defined as follows:

$$h = 1 - \frac{H(C|K)}{H(C)} \tag{9}$$

and,

$$H(C|K) = -\sum_{j=1}^{k}\sum_{i=1}^{p}\frac{n_{ij}}{|A|}\log\frac{n_{ij}}{|A_j|} \tag{10}$$

$$H(C) = -\sum_{i=1}^{p}\frac{|C_i|}{|A|}\log\frac{|C_i|}{|A|} \tag{11}$$

Completeness c is defined as follows:

$$c = 1 - \frac{H(K|C)}{H(K)} \tag{12}$$

and,

$$H(K|C) = -\sum_{i=1}^{p}\sum_{j=1}^{k}\frac{n_{ij}}{|A|}\log\frac{n_{ij}}{|C_i|} \tag{13}$$

$$H(K) = -\sum_{j=1}^{k}\frac{|A_j|}{|A|}\log\frac{|A_j|}{|A|} \tag{14}$$

In the above formulas n_{ij} is the number of data points belonging to the i-th class and the j-th cluster and $|\cdot|$ is the cardinality of a set. In this paper, we take $\beta = 1$.

Purity shows how well the cluster distribution obtained by a clustering algorithm reflects the existing class structure of a data set [6]. Let $\bar{A} = \{A^1, \ldots, A^k\}$, $k \geq 2$ be the cluster distribution of the set A and $C^1, \ldots, C^l$ be the true classes of A. Denote by n_{tj} the number of points from the t-th class belonging to the j-th cluster. Compute the following:

$$\bar{m}_j = \max_{t=1,\ldots,l} n_{tj}, \ j = 1, \ldots, k. \tag{15}$$

The purity for the cluster distribution $\bar{A}$ [6] is given in Equation (16).

$$PR(\bar{A}) = \left(\frac{1}{m}\sum_{j=1}^{k}\bar{m}_j\right) \times 100\%. \tag{16}$$

4.4. MVIIC Algorithm

Algorithm 2 illustrates the process of building a malware data set from existing samples and then submitting the malware data set for processing in a Cuckoo sandbox. When the sandbox processing is completed, a script is used to extract the raw features from the Cuckoo logfiles. The raw features are encoded as either an API histogram or as API name and hostname data that are encoded using TF/IDF. The incremental clustering program is then run to cluster the features; when the clustering is finished, the clustering metrics are extracted.

Algorithm 2 MVIIC Algorithm

Require: Malware data set
Submit malware data set to Cuckoo sandbox
while Sandbox processing samples **do**
Wait
end while
Extract features from Cuckoo report
Encode features to vector format
Run incremental clustering program
while Clustering in progress **do**
Wait
end while
Read clustering metrics

5. Experiments and Results

5.1. Malware Feed

The malware feed used in this work was provided by the abuse.ch research project on [23]. A common feature of machine learning research is the need for accurate identification of the feature labels. In the case of malware clustering, the feature labels identify the malware families. The automatic identification of malware families is an open research problem. While prior research used anti-virus labels [24] or network traffic [9] to estimate the ground truth, the filenames of the samples used in this research contain the malware family name and the sample hash. The malware family name was used to provide the ground truth for this research. The malware families in this feed are a mixture of banking malware and Remote Access Trojans (RATs).

5.2. Research Environment

Feature extraction was run on a laptop with an Intel i5-5300U CPU 2.30 GHz, with 8 GB of RAM, using Cuckoo 2.0.7, Virtualbox 5.2.42, and a Windows 7 64-bit VM. Clustering was performed on a workstation with an Intel i7-3770 CPU 3.40 GHz, and 32 GB of RAM. The Cuckoo sandbox [1] was used for dynamic analysis in this research. The malware sample data sets were submitted to Cuckoo, and a script was written to extract the features from the Cuckoo reports.

5.3. Data Sets

A prior study [25] examined research that clustered malware samples selected based on their anti-virus classification. This research reported a high degree of clustering accuracy. However, reduced accuracy was observed using the clustering algorithm on a different malware data set. This study concluded that the method used for the selection of malware samples may bias sample selection toward easily classified malware samples.

Data Set 1 contains samples of 6 malware families; the details of this data set are provided in Table 1. Prolonging the sandbox execution timeout allows the malware samples to execute more API calls, and this may improve clustering performance. To investigate whether clustering performance is influenced by the sandbox timeout, three sets of feature

extraction were performed with Data Set 1, using sandbox timeouts of 60, 120, and 180 s. These features are referred to as Data Set 1A, 1B, and 1C, respectively.

Table 1. Composition of Data Set 1.

Malware Family	Count
Agent Tesla	365
Heodo	151
Njrat	407
Quasar Rat	580
Tinba	374
Trickbot	263
Total	2140

5.4. Experiments

The experiments listed below were performed to examine the effectiveness of the clustering algorithm. These experiments were designed to investigate feature engineering for dynamic analysis features, to investigate the potential of sandbox hardening measures, to demonstrate the effectiveness of the incremental clustering algorithm in this application, and to provide a comparison of our clustering approach against Yara rule-based malware detection.

- Experiment 1: Feature frequency threshold.
- Experiment 2: API histogram compression.
- Experiment 3: TF/IDF encoding.
- Experiment 4: Sandbox hardening.
- Experiment 5: Execution time.
- Experiment 6: Comparison against Yara rules.

The following abbreviations are used in reporting the results of Experiments 1–5; these experiments use the filtering of infrequent or high-frequency API calls from the features in order to improve the clustering performance.

- **Min API Count** is the threshold that is used to exclude infrequent API features.
- **Max API Count** is the threshold that is used to exclude high-frequency API features.
- **API Count Truncation** is the maximum value of the API histogram.
- **Feat Count** is the number of features that were present following filtering.
- **Cl** is the number of clusters that were identified.
- **Purity** is the cluster purity.
- F_1 is the average F_1 score of all clusters.
- **V-M** is the average V-measure of all clusters.

Experiment 1: Feature Frequency Threshold. In this paper, it is hypothesized that features that occur infrequently or very frequently do not improve malware clustering accuracy. To investigate this empirically, a histogram of the API call counts for all malware samples was constructed. The API call count histogram is a sparse representation. Fujino et al. used TF/IDF to eliminate the infrequent or high-frequency API calls from the data set [26]. Our research uses a similar approach to remove infrequent or high-frequency API features, using minimum and maximum thresholds.

The effects of using API feature frequency thresholds are shown in Tables 2 and 3. Referring to Table 2, it can be seen that the clustering performance improves when features with fewer than 1000 nonzero API counts are excluded, giving a maximum F_1 score of 0.68. Referring to Table 3, it can be seen that the clustering performance is improved by excluding features with an API count of more than 2120, giving an F_1 score of 0.47.

Table 2. Data Set 1C exclusion of infrequent features.

Min API Count	Feat Count	Cl	Purity	F_1	V-M
0	388	13	32.8	0.32	0.10
600	213	17	63.9	0.49	0.51
1000	181	26	82.09	0.68	0.65
1200	175	25	82.09	0.68	0.65
1400	170	29	82.23	0.68	0.64
1600	168	29	82.23	0.68	0.64
1800	164	34	77.95	0.67	0.63
2000	163	28	77.66	0.66	0.64
2140	155	29	49.43	0.38	0.39

Table 3. Data Set 1C exclusion of high-frequency features.

Max API Count	Feat Count	Cl	Purity	F_1	V-M
2140	388	13	32.80	0.32	0.10
2120	387	14	49.61	0.47	0.41
2100	385	27	49.01	0.46	0.45
2080	383	14	37.37	0.34	0.20
2000	379	26	37.89	0.33	0.16
1600	375	14	32.89	0.32	0.11
1200	368	28	33.1	0.32	0.11

Experiment 2: API Histogram Compression. Some malware samples make a large number (multiple 100,000) of calls to specific APIs. This may be an attempt to hinder analysis by causing sandbox timeouts. CBM [27] improves performance, using a logarithmic method to compress the value of API histogram counts.

In this experiment, we use a simpler method to compress the API histogram. Our method truncates the API histogram value to a specified maximum value called the maximum API count. The results in Table 4 use a minimum API count of 1000 and a maximum API count of 2120 and varies the API count truncation value. The best clustering performance occurs with an API count truncation value of between 5 and 25.

Table 4. Data Set 1C API count truncation.

API Count Truncation	Feat Count	Cl	Purity	F_1	V-M
1	180	12	87.56	0.76	0.74
3	180	7	86.33	0.80	0.77
5	180	7	87.42	0.81	0.78
7	180	10	91.8	0.82	0.78
10	180	7	85.39	0.81	0.76
15	180	12	90.66	0.80	0.76
25	180	11	89.73	0.80	0.76
50	180	11	89.87	0.81	0.76
100	180	15	93.50	0.81	0.76
300	180	29	95.05	0.80	0.76
500	180	19	85.96	0.71	0.69

Experiment 3: TF/IDF Encoding. TF/IDF was used to encode the API names and DNS addresses. Duplicates were removed from the API names and DNS addresses prior to TF/IDF encoding. The results of clustering using TF/IDF feature encoding are shown in Table 5. The columns labeled Min DF and Max DF contain the values passed to the SciKit-Learn TFIDF CountVectorizer that are used to ignore terms that occur below a low-frequency threshold or above a high-frequency threshold. These results show that the performance of the histogram and TF/IDF encoding in these tests is equivalent.

Table 5. Data Set 1C TF/IDF performance.

Min DF	Max DF	Feat Count	Cl	Purity	F_1	V-M
0.000	1.00	386	7	88.50	0.81	0.78
0.001	1.00	248	9	90.25	0.81	0.77
0.002	1.00	223	7	88.60	0.81	0.78
0.003	1.00	213	8	88.60	0.80	0.77
0.004	1.00	203	7	88.60	0.80	0.78
0.010	1.00	186	7	88.60	0.81	0.78
0.020	1.00	172	7	88.60	0.81	0.78
0.030	1.00	162	8	89.44	0.81	0.77
0.040	1.00	153	8	88.55	0.76	0.75
0.050	1.00	141	8	88.55	0.76	0.75
0.001	0.99	244	8	88.60	0.80	0.77
0.001	0.95	240	7	88.60	0.81	0.78
0.001	0.80	229	7	88.60	0.81	0.78
0.001	0.60	229	8	90.95	0.82	0.79
0.001	0.40	216	8	88.60	0.80	0.77

Experiment 4: Sandbox Hardening. Some malware families contain anti-analysis code that terminates the malware or performs decoy actions when execution in an analysis environment is detected [28,29]. In this research, the following anti-analysis techniques were mitigated:

- VM integration software detection.
- Processor core count checking.
- Malware slow startup.

VirtualBox and other VM environments provide optional software to improve the integration between the host computer and the VM. While this software improves the integration of the VM, it is readily detected by malware [29]. The first VM hardening technique used in this research was the removal of the VirtualBox Guest Additions software.

Some malware, e.g., Dyre [30], counts the number of processor cores. If only one processor core is present, the malware terminates. This test provides a mechanism for detecting virtualized analysis environments. The VM used in this research was configured with two processor cores.

Some malware programs start slowly, and this may cause a sandbox timeout before malware behaviors are revealed. Setting a sandbox analysis timeout to a low value allows a higher malware sample throughput but may not allow sufficient time to capture malware behaviors. Setting a longer analysis timeout may allow identification of malware behaviors, but does so at the cost of reduced throughput. In this research, an experiment was performed to determine if clustering performance varies as a result of setting different analysis timeout values. The results of this experiment, shown in Table 6, indicate that increasing sandbox execution times results in improved clustering metrics (cluster purity, F1 score, and V-Measure). This indicates that longer sandbox timeouts can be used to capture more details of malware behaviors.

Table 6. VM timeout versus maximum cluster purity.

Analysis Timeout (s)	Feat Sel	Cl	Purity	F_1	V-M
Data Set 1A 60 s	75	11	89.29	0.79	0.73
Data Set 1B 120 s	109	10	90.11	0.81	0.75
Data Set 1C 180 s	180	10	91.80	0.82	0.78

In addition, the following changes were also made to harden the VM [31]: disable Windows Defender, disable Windows Update and deactivate Address Space Layout Randomization (ASLR) [32].

Experiment 5: Execution Time. The execution times for feature extraction and clustering are shown for the three versions of Data Set 1 in Table 7. The experiments in this research were conducted on a workstation using an Intel i7-3770 3.40 GHz CPU with 32 GB of RAM. Feature extraction was performed, using a Python script. The clustering program was written in Fortran and was compiled with *gfortran*. These results show that the clustering operation accounts for the majority of the run time, the malware clustering times are suitable for research purposes, and that the clustering time is proportional to the feature count.

Table 7. Execution time.

Data Set	Feat Sel	Extraction(s)	Clustering(s)
Data Set 1A (60 s)	209	0.7	113.5
Data Set 1B (120 s)	256	0.8	164.1
Data Set 1C (180 s)	327	1.0	275.5

5.5. Yara Rules

Yara rules for the six malware families in Data Set 1 were obtained from Malpedia [33], and these rules are summarized in Table 8. While these Yara rules were not written specifically for this research, they were used to conduct a study that highlights the shortcomings of Yara rule-based malware detection. The best performing Yara rules were Agent Tesla, Njrat, and Tinba rules. These rules were used in experiments that compared the performance of Yara rules and clustering.

Table 8. Yara rule counts.

Malware Family	Yara Rule Count
Agent Tesla	2
Heodo	4
Njrat	2
Quasar Rat	1
Tinba	4
Trickbot	7

Experiment 6: Comparison Against Yara Rules. The Yara rules from Malpedia [33] were run against process memory dumps from Data Set 1. The malware identification results for the Yara rules are shown in Table 9. These Yara rules were not optimized for this data set. However, it can be seen that the detection rate using these Yara rules varied from 18.90 percent to 94.12 percent. Yara rules are known to be sensitive to minor malware changes resulting from software development. Referring to Table 6, it can be seen that the cluster purity was approximately 90%. If the dynamic analysis features from the malware samples that were not identified by the Yara rules had been clustered, then clusters with a purity of approximately 90% would have allowed identification of the unidentified variants, allowing an opportunity for updating the Yara rules. MVIIC uses a Cuckoo sandbox-based dynamic analysis platform to provide the first step in the identification of new malware variants. The runtime performance of the Cuckoo sandbox is heavily dependent on the hardware platform, sandbox configuration, and the configured Yara rules. This leads to the conclusion that, while the clustering performance is an important parameter of this research, the Cuckoo sandbox execution time (including Yara scanning) is not relevant.

Table 9. Data Set 1 Yara performance.

Family	Sample Count	% Detection	Yara Hits
Agent Tesla	365	18.90	69
Heodo	365		
Njrat	407	67.81	276
Quasar Rat	580		
Tinba	374	94.12	352
Trickbot	263		

6. Discussion and Limitations

This research is set in the context of a two-layered malware analysis system that uses Yara rules for the identification of malware families and attempts to address the problem of minor changes in the malware code, causing a detection failure. In this use case, this paper proposes a hybrid approach, using Yara scanning and clustering, these act in concert to identify new malware variants. In this scenario, those malware samples that are not detected by Yara rules are collected for clustering, and the results of this clustering process can be used for the identification of new malware variants. While previous malware clustering research focused on the identification of the feature combinations and clustering algorithms that provide the highest performance, this research investigates the application of a novel incremental clustering method.

The results of the experiments in this paper show that this novel clustering algorithm can be used for the task of clustering malware, using features obtained by dynamic analysis. In the data set used in this research, the filenames of each malware sample contain the malware family name. This family name was used to provide the clustering ground truth. Any errors in the malware labeling will reduce the reported clustering accuracy. This highlights a need for the accurate labeling of malware data sets. The experiments in this paper showed that clustering performance was improved by filtering infrequent or high-frequency features. The performance of API frequency histogram encoding and TF/IDF encoding of features gave similar results. Increasing the sandbox malware sample execution timeout improved the clustering performance. The performance of malware family detection using Yara rules was compared with clustering performance, this experiment illustrated that the clustering approach has a more consistent and higher level of performance that is resistant to small changes introduced by malware variants.

While there is a need to compare MVIIC with other related algorithms, care needs to be taken to ensure that these comparisons are valid and to evaluate MVIIC using data sets from previous research. MVIIC research has not yet progressed to the point of being evaluated against data sets from other research.

7. Conclusions

This paper proposes a two-layered MVIIC technique that makes use of a novel incremental clustering algorithm to support Yara-based malware family identification by the clustering of unidentified malware variants. These unidentified malware samples represent a valuable source of unknown malware families and new malware variants.

While some previous malware clustering research has focused on the identification of the feature combinations and clustering algorithms that provide the highest performance, this research investigates malware clustering using a novel incremental clustering method. Using a data set of in-the-wild malware, this research has shown that this clustering algorithm can be used to cluster malware features obtained by dynamic analysis. The clustering performance was improved by excluding infrequent and high-frequency features. A comparison of feature encoding using an API frequency histogram and TF-IDF encoding of API names was performed. These feature encoding methods were shown to provide equivalent performance. Increasing the malware sample sandbox execution time was shown to improve clustering performance.

An investigation of the detection of the malware samples by a set of existing Yara rules was performed. Clustering provided cluster purity of 90%, while Yara detection varied from 18.90% to 94.12%. This illustrates that the pattern matching approach used in Yara rules may fail, due to minor code variations, while the performance of a clustering approach is not impacted by minor code variations.

This research has shown that this novel incremental clustering algorithm is able to cluster malware dynamic analysis features.

Future Work

The work presented in this paper can be extended as follows:

- Investigate techniques to improve the clustering performance of dynamic malware analysis features.
- Investigate techniques to improve the run time performance of the clustering of dynamic malware analysis features.
- Make use of the incremental operation of the non-smooth optimization clustering technique to support the identification of new variants in a continuous malware analysis operation.
- Investigate techniques for the automatic generation of Yara rules to support the analysis concept shown in Figure 1.

Author Contributions: Conceptualization, P.B.; Formal analysis, A.B.; Funding acquisition, I.G.; Investigation, P.B.; Methodology, P.B.; Software, P.B., A.B. and M.M.; Supervision, I.G.; Writing—original draft, P.B.; Writing—review and editing, I.G. and A.B. All authors have read and agreed to the published version of the manuscript.

Funding: The research by Adil Bagirov is supported by the Australian Government through the Australian Research Council's Discovery Projects funding scheme (Project No. DP190100580).

Data Availability Statement: The authors can be contacted for the availability of the datasets, and requests will be processed case by case basis.

Acknowledgments: The research was conducted at the Internet Commerce Security Lab (ICSL), where Westpac, IBM and Federation University are partners.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cuckoo Authors. Cuckoo Sandbox. 2021. Available online: https://cuckoo.sh/blog (accessed on 7 July 2021).
2. Willems, C.; Holz, T.; Freiling, F. Toward automated dynamic malware analysis using cwsandbox. *IEEE Secur. Priv.* **2007**, *5*, 32–39. [CrossRef]
3. Brengel, M.; Rossow, C. YARIX: Scalable YARA-based Malware Intelligence. In Proceedings of the USENIX Security Symposium, Virtual Event, 11–13 August 2021.
4. Shan, Z.; Wang, X. Growing grapes in your computer to defend against malware. *IEEE Trans. Inf. Forensics Secur.* **2013**, *9*, 196–207. [CrossRef]
5. Bayer, U.; Comparetti, P.M.; Hlauschek, C.; Kruegel, C.; Kirda, E. Scalable, behavior-based malware clustering. In Proceedings of the NDSS 2009, San Diego, CA, USA, 8–11 February 2009; Volume 9, pp. 8–11.
6. Bagirov, A.; Karmitsa, N.; Taheri, S. *Partitional Clustering via Nonsmooth Optimization*; Springer: Chem, Switzerland, 2020.
7. Gibert, D.; Mateu, C.; Planes, J. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Appl.* **2020**, *153*, 102526. [CrossRef]
8. Rieck, K.; Trinius, P.; Willems, C.; Holz, T. Automatic analysis of malware behavior using machine learning. *J. Comput. Secur.* **2011**, *19*, 639–668. [CrossRef]
9. Faridi, H.; Srinivasagopalan, S.; Verma, R. Performance Evaluation of Features and Clustering Algorithms for Malware. In Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW), Singapore, 17–20 November 2018; pp. 13–22.
10. Wong, K.; Dillabaugh, C.; Seddigh, N.; Nandy, B. Enhancing Suricata intrusion detection system for cyber security in SCADA networks. In Proceedings of the 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, Canada, 30 April–3 May 2017; pp. 1–5.

11. Tibshirani, R.; Walther, G.; Hastie, T. Estimating the number of clusters in a data set via the gap statistic. *J. R. Stat. Soc. Ser. B (Stat. Methodol.)* **2001**, *63*, 411–423. [CrossRef]
12. Pitolli, G.; Laurenza, G.; Aniello, L.; Querzoni, L.; Baldoni, R. MalFamAware: Automatic family identification and malware classification through online clustering. *Int. J. Inf. Secur.* **2021**, *20*, 371–386. [CrossRef]
13. Arntz, P. Explained: YARA Rules. 2017. Available online: https://virustotal.github.io/yara (accessed on 7 July 2021).
14. Yara. Yara: The Pattern Matching Swiss Knife for Malware Researchers. 2017. Available online: https://blog.malwarebytes.com/security-world/technology/2017/09/explained-yara-rules (accessed on 7 July 2021).
15. Infosec Institute. YARA: Simple and Effective Way of Dissecting Malware. 2015. Available online: http://resources.infosecinstitute.com/yara-simple-effective-way-dissecting-malware/#gref (accessed on 7 July 2021).
16. Zhang, T.; Ramakrishnan, R.; Livny, M. BIRCH: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov.* **1997**, *1*, 141–182. [CrossRef]
17. Bagirov, A.M.; Yearwood, J. A new nonsmooth optimization algorithm for minimum sum-of-squares clustering problems. *Eur. J. Oper. Res.* **2006**, *170*, 578–596. [CrossRef]
18. Bagirov, A.; Mohebi, E. Nonsmooth optimization based algorithms in cluster analysis. In *Partitional Clustering Algorithms*; Celebi, E., Ed.; Springer: Chem, Switzerland, 2015.
19. Bagirov, A.; Karmitsa, N.; Mäkelä, M. *Introduction to Nonsmooth Optimization*; Springer: Chem, Switzerland, 2014.
20. Black, P.; Sohail, A.; Gondal, I.; Kamruzzaman, J.; Vamplew, P.; Watters, P. API Based Discrimination of Ransomware and Benign Cryptographic Programs. In Proceedings of the International Conference on Neural Information Processing, Bangkok, Thailand, 18–22 November 2020; Springer: Berlin/Heidelberg, Germany, 2020; pp. 177–188.
21. Takeuchi, Y.; Sakai, K.; Fukumoto, S. Detecting ransomware using support vector machines. In Proceedings of the 47th International Conference on Parallel Processing Companion, Eugene, OR, USA, 13–16 August 2018; p. 1.
22. Lipton, Z.C.; Elkan, C.; Naryanaswamy, B. Optimal thresholding of classifiers to maximize F1 measure. In Proceedings of the Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Nancy, France, 15–19 September 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 225–239.
23. Abuse.ch. Fighting Malware and Botnets-Abuse.ch. 2021. Available online: https://abuse.ch (accessed on 7 July 2021).
24. Zhang, Y.; Sui, Y.; Pan, S.; Zheng, Z.; Ning, B.; Tsang, I.; Zhou, W. Familial clustering for weakly-labeled android malware using hybrid representation learning. *IEEE Trans. Inf. Forensics Secur.* **2019**, *15*, 3401–3414. [CrossRef]
25. Li, P.; Liu, L.; Gao, D.; Reiter, M.K. On challenges in evaluating malware clustering. In Proceedings of the International Workshop on Recent Advances in Intrusion Detection, Ottawa, ON, Canada, 15–17 September 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 238–255.
26. Fujino, A.; Murakami, J.; Mori, T. Discovering similar malware samples using api call topics. In Proceedings of the 2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC), Las Vegas, NV, USA, 9–12 January 2015.
27. Qiao, Y.; Yang, Y.; He, J.; Tang, C.; Liu, Z. CBM: Free, automatic malware analysis framework using API call sequences. In *Knowledge Engineering and Management*; Springer: Berlin/Heidelberg, Germany, 2014; pp. 225–236.
28. Black, P.; Opacki, J. Anti-analysis trends in banking malware. In Proceedings of the 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, PR, USA, 18–21 October 2016; pp. 1–7.
29. Ferrand, O. How to detect the cuckoo sandbox and to strengthen it? *J. Comput. Virol. Hacking Tech.* **2015**, *11*, 51–58. [CrossRef]
30. Kovacs, E. Dyre Banking Trojan Counts Processor Cores to Detect Sandboxes. 2015. Available online: http://www.securityweek.com/dyre-banking-trojan-counts-processor-cores-detect-sandboxes (accessed on 7 July 2021).
31. Byte Atlas. Knowledge Fragment: Hardening Win7 x64 on VirtualBox for Malware Analysis. 2020. Available online: http://byte-atlas.blogspot.com/2017/02/hardening-vbox-win7x64.html?m=1 (accessed on 7 July 2021).
32. Russinovich, M.E.; Solomon, D.A.; Ionescu, A. *Windows Internals*; Pearson Education: New York, NY, USA, 2012.
33. Plohmann, D.; Clauss, M.; Enders, S.; Padilla, E. Malpedia: A Collaborative Effort to Inventorize the Malware Landscape. *J. Cybercrime Digit. Investig.* **2018**, *3*. [CrossRef]