MALIGN: Adversarially Robust Malware Family Detection using Sequence Alignment

1st Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

2nd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

3rd Given Name Surname
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—We propose MALIGN, a novel malware family detection approach inspired by genome sequence alignment. MALIGN encodes malware using four nucleotides and then uses genome sequence alignment approaches to create a signature of a malware family. Unlike previous approaches based on sequence alignment, our method uses a multiple whole-genome alignment tool that protects against trivial adversarial attacks such as shuffling of code blocks. Our approach outperforms state-of-the-art machine learning based malware detectors and demonstrates robustness against trivial adversarial attacks. MALIGN also helps identify the techniques malware authors use to evade detection.**

***Index Terms*—malware, adversarial, sequence alignment**

I. INTRODUCTION

To detect the rising number of malwares at scale, machine learning is necessary. Indeed, currently all commercial malware detectors use machine learning. However, one shortcoming of current static malware detectors is that they can be easily evaded by changing a malware trivially without changing the core of the malware [1]–[10]. Fundamentally, the adversarial attacks use one simple technique: add random content to the malware.

One simple way to design an adversarially robust malware detector is to make it rely on the core functionalities of a malware and ignore randomly added content. This turns out to be surprisingly hard, especially for static detectors, without increasing the false positive rate. This happens because malware authors often take a similar approach to evade detection—hide malicious content in the overlay and data section instead of the code section.

Taking inspiration from bioinformatics, we model a malware like a DNA sequence or a genome. Just as DNA sequences are made of only four types of nucleotides, malwares are sequences of bits, and modifications of malwares mirrors accumulation of mutations in genomes during evolution. Genomes contain critical regions for the survival of the organism, such as, protein coding genes where mutations may be lethal. Similarly, malwares contain code blocks that are difficult to modify without altering its functionality. If we can translate a malware in terms of the basic building blocks, our detector will be robust by design that cannot be evaded without fundamentally changing the malware.

We propose MALIGN, a novel malware family detection approach inspired by genome sequence alignment. Our approach at first converts a family of malware files into malware nucleotide sequence files i.e. sequences of A, C, G and T. Then we use a multiple whole-genome alignment tool to identify common alignment blocks per family. These alignment blocks work as a signature of the malware family. We train a classifier using the features that represent how well a block identifies with a particular family. To classify whether a new malware belongs to the family, we first compute the alignment of the new malware with the sequences representing the blocks i.e. signature of the family, and use it to classify the malware.

Our robustness properties come from the use a recent multiple whole-genome alignment method that can find conserved blocks of sequences even in the presence of sequence reordering and minor modifications, and through estimation of the degree of conservation at each location by processing the generated alignment. It prevents certain types of adversarial manipulation, such as, adding extra content, changing code order, and minor changes to the code. To evade detection, an attacker needs to change the code significantly.

We evaluate MALIGN on two datasets: Kaggle Microsoft Malware Classification Challenge (Big 2015) and Microsoft Machine Learning Security Evasion Competition (2020) (MLSec). In comparison with MalConv and CNN-based malware classifiers, our approach has higher accuracy and robustness. Moreover, sequence alignment helps reveal the common practices of malware families, such as hiding code in non-code sections.

In summary, our main contributions are:

- **Scalable and Explainable**: Our approach is simple, scalable and easily explainable.
- **High accuracy with low train data**: Our approach achieves high accuracy without large training data.
- **Robustness to adversarial attacks**: MALIGN finds conserved code blocks and assigns high scores to those, critical code blocks may need to be modified to evade detection.

II. BACKGROUND

Sequence alignment is a widely studied problem in bioinformatics to find similarity among DNA, RNA or protein sequences, and to study evolutionary relationships among diverse species. It is the process of arranging sequences in such a way that regions of similarity are *aligned*, with gaps (denoted by '-') inserted to represent insertions and deletions in sequences. An alignment of the sequences `ATTGACCTGA` and `ATCGTGTA` is shown below where the regions denoted in black characterized by matched characters are *conserved* whereas the red and blue regions denote substitutions i.e. point mutations, and insertions or deletions during the evolutionary process respectively.

```
ATTGACCTG-A
AT---CGTGTA
```

In sequence alignment, matches, mismatches and insertions-deletions (in-dels) are assigned scores based on their frequencies during evolution and the goal is to find an alignment with the maximum score. The problem of finding an optimal alignment of the entire sequences (global alignment) and that of finding an optimal alignment of their sub-sequences (local alignment) can be solved by dynamic programming using the Needleman-Wunsch [11] and Smith-Waterman [12] algorithms respectively. While the algorithms can be used to align more than two sequences, the running time is exponential in the number of sequences. To address the tractability issue, a number of tools have been developed [13]–[15], that use heuristics to solve the multiple sequence alignment (MSA) problem.

However, in addition to point mutations and short insertions-deletions, large scale genome rearrangement events take place during evolution. Such genome rearrangement events include reversal of a genomic segment (inversion), shuffling of order of genomic segments (transposition or translocation), duplication and deletion of segments, etc. Although the aforementioned tools are unable to deal with genome rearrangements, methods such as MUMmer [16] can perform alignment of two sequences in presence of rearrangements whereas Mauve [17], Cactus [18], etc. can handle multiple sequences.

Recently, Armstrong *et al.* have developed Progressive Cactus [19], and Minkin & Medvedev have developed SibeliaZ [20] that can align hundreds to thousands of whole-genome sequences in presence of rearrangements. The tools identify similar sub-sequences in the sequences from different species to create blocks of rearrangement-free sequences, and then performs a multiple sequence alignment of the sequences in each block.

Since adversaries can modify malwares relatively easily by changing orders of blocks of codes without altering functionality of the malware, it is important that the tool used to align malware sequences is robust to such rearrangements in code. Here, we use SibeliaZ to align malware sequences to identify conserved blocks of codes and calculate a conservation score of the blocks for malware detection and classification. It is worth noting that the blocks of codes identified need not be fully conserved, i.e. there can be modifications, insertions and deletions of small number of instructions within the blocks, making it robust to adversarial attacks.

III. Related Work

To counter the increasing amount of malware and detect them, several methods and techniques have been developed over the years. In the early days, Wressnegger et. al. [21] and Zakeri et. al. [22] proposed a signature based approach using static analysis. Later, a dynamic approach - malware detection by analysing the malware behavior, was proposed by Martignoni et. al. [23] and Willems et. al. [24]. In recent times, machine learning based techniques are mostly being used to classify malwares. Schultz et. al. [25] first proposed a data mining technique for malware detection using three different types of static features. Nataraj et. al. [26] proposed a malware classification approach based on image processing techniques by converting the bytes files to image files. Later, Kalash et. al. [27] improved on [26] by developing M-CNN using malware images. [28] and [29] proposed techniques with LSTM using opcode sequences of malware. Santos et. al. [30] proposed a hybrid technique by integrating both static and dynamic analysis. Yan et. al. [31] developed MalNet using an ensemble method on CNN, LSTM and extracting metadata features. Recently, Raff et. al. [32] developed a state-of-the-art technique, MalConv using the raw byte sequence as the input to a neural network.

Prior work proposed two main ways to improve the adversarial robustness of malware detectors: adversarial training, and robustness by design. Adversarial training, where a malware detector is trained with adversarial examples, is one of the mostly used approaches to improve adversarial robustness [33]. In the malware domain, several work demonstrated that adversarial training can improve the robustness significantly without reducing the accuracy on the original sample [34]–[36]. Robustness by design approaches build classifiers to eliminate a certain classes of adversarial attacks. Certified or provable robustness is a robustness by design approach that trains classifiers with local robustness properties that can provably eliminate classes of evasion attacks [37]. In the malware domain, Chen et al. [38] proposed learning PDF malware detectors with verifiable robustness properties. Íñigo et al. [39] trained a XGBoost based malware detector with the monotonicity property that ensures that an adversary cannot decrease the classification score by adding extra content. Our approach falls under the robustness by design category. Although our approach does not provide any provable robustness guarantees, but it increases the cost of an adversary by eliminating the possibility of trivial attacks.

In the past, sequence alignment based approaches have been used for malware analysis by a number of researchers [40]–[46]. Chen et al. used multiple sequence alignment to align computer viral and worm codes of variable lengths to identify invariant regions [40]. This approach was subsequently enhanced in [41], [42], [44]. However, none of these methods address the issue that blocks of code can be shuffled without affecting malware behaviour.

Sequence alignment has also been applied on API call sequences of malwares to extract evasion signatures and cluster samples [43], classify malware families [45], and for malware detection, classification and visualization [46]. While it is more difficult for malware developers to shuffle API calls without changing the behaviour of malwares, these approaches require access to API call sequences of malwares and are not suitable in all circumstances.

Drew et al. utilized another approach developed by the bioinformatics and computational biology community for malware classification - that for gene or sequence classification. The method is based on extracting short words i.e. k-mers from sequences and calculating similarity between sequences based the set of words present in them. Although the method is efficient, it does fully utilize the information provided by long stretches of conserved regions in malwares, and is not suitable for identifying critical code blocks in malwares.

IV. Methods

Overview

To classify or detect known/unknown malwares and its variants, in this paper, we propose a malware classification or detection system based on multiple whole-genome alignment. The basic building block of the method is a binary classification system that can predict whether an instance belongs to a particular malware family or not. The input to this binary classifier is a training set consisting of positive examples i.e. malwares from a particular family, and negative examples which may be non-malwares or malwares from other families. The system can be extended to malware detection by creating a binary classifier for all malware families. The instances that are predicted to be negative by all these classifiers can then be treated as benign.

The main steps of our proposed method are shown in Algorithm 1.

The method is illustrated in Figure 1. We start with the given *malware bytes files* i.e. executable files and convert them into *malware nucleotide sequence files* i.e. sequences of A, C, G and T. Then these nucleotide sequence files are aligned with a multiple whole-genome alignment tool (SibeliaZ) which outputs alignment blocks that are common among these files. These alignment blocks are merged and *consensus sequence* is constructed. In this step, *conservation score* for each coordinate of the consensus sequence is also generated. This consensus sequence is aligned with each sample from a balanced train set with positive and negative samples with respect to the malware family of interest, and an *alignment score* is calculated for every sample for each conserved block. These scores are then used as input to the machine learning model which learns a classifier to distinguish between malwares belonging to the family, and malwares from other families as well as non-malwares. To classify new samples, the sequences are aligned to the consensus sequence and alignment score for the new instance is generated similarly and passed into our classifier which then classifies the new samples. Each of these steps is described in more details below.

Algorithm 1: MAlign

```
Input: Training set, $\mathbb{X} = (\mathbb{X}_+, \mathbb{X}_-)$, where $\mathbb{X}_+$: byte files from
       malwares of a family, $\mathbb{X}_-$: byte files from non-malwares or
       malwares from other families, and Test set, $\mathbb{Y}$
Output: Labels for $\mathbb{Y}$
$(\mathbb{S}_+, \mathbb{S}_-) \leftarrow$ Convert to nucleotide sequence files $(\mathbb{X}_+, \mathbb{X}_-)$
$\mathbb{B} \leftarrow$ Perform multiple sequence alignment and identify conserved
  blocks $(\mathbb{S}_+)$
forall blocks $B \in \mathbb{B}$ do
    $C_B \leftarrow$ Get consensus sequence $(B)$
    for $i = 1 \rightarrow length(B)$ do
        Calculate $ConservationScore(B, i, N)$ where
          $N = A, C, G, T$
forall sequences $Z \in (\mathbb{S}_+ \cup \mathbb{S}_-)$ do
    $F_Z$:= Feature vector of $Z$
    forall blocks $B \in \mathbb{B}$ do
        $S \leftarrow$ Get alignments $(Z, C_B)$
        $AlignmentScore \leftarrow$ Calculate alignment score $(S, B)$
        $AlignmentCount \leftarrow$ Get alignment count $(S)$
        $F_Z \leftarrow F_Z \cup (AlignmentScore, AlignmentCount)$
$\mathcal{M} \leftarrow$ Learn classification model $(\mathbb{F}, \mathbb{X})$ where $\mathbb{F}$: feature matrix
$\mathbb{T} \leftarrow$ Convert to nucleotide sequence files $(\mathbb{Y})$
$\mathbb{L}$ : labels
forall sequences $T \in \mathbb{T}$ do
    $F_T$:= Feature vector of $T$
    forall blocks $B \in \mathbb{B}$ do
        $S \leftarrow$ Get alignments $(T, C_B)$
        $AlignmentScore \leftarrow$ Calculate alignment score $(S, B)$
        $AlignmentCount \leftarrow$ Get alignment count $(S)$
        $F_T \leftarrow F_T \cup (AlignmentScore, AlignmentCount)$
    $L \leftarrow$ Predict $(\mathcal{M}, F_T)$
    $\mathbb{L} \leftarrow \mathbb{L} \cup L$
return $\mathbb{L}$
```

Bytes file to nucleotide sequence file conversion

First the binary executable or bytes files are converted to nucleotide sequence files containing sequences of A, C, G and T. The conversion is performed so that existing whole-genome alignment tools can be used. The conversion from the byte code to nucleotide sequence is done by converting each pair of bits to a nucleotide according to Table I.

Byte Character	Nucleotide
00	A
01	C
10	G
11	T

TABLE I
Bytes to Nucleotide Mapping

In some malware datasets such as the Kaggle Microsoft Malware Classification Challenge (Big 2015) dataset [47], the provided bytes files contain "`??`" and long stretches of "`00`" which do not preserve any significant value or meaning. These are removed before conversion to nucleotide sequences.

Multiple alignment of malware nucleotide sequences

The next step is to align the malware nucleotide sequences. In this paper, we use the multiple whole-genome alignment tool SibeliaZ [20]. SibeliaZ performs whole-genome alignment of multiple sequences and constructs locally co-linear blocks.

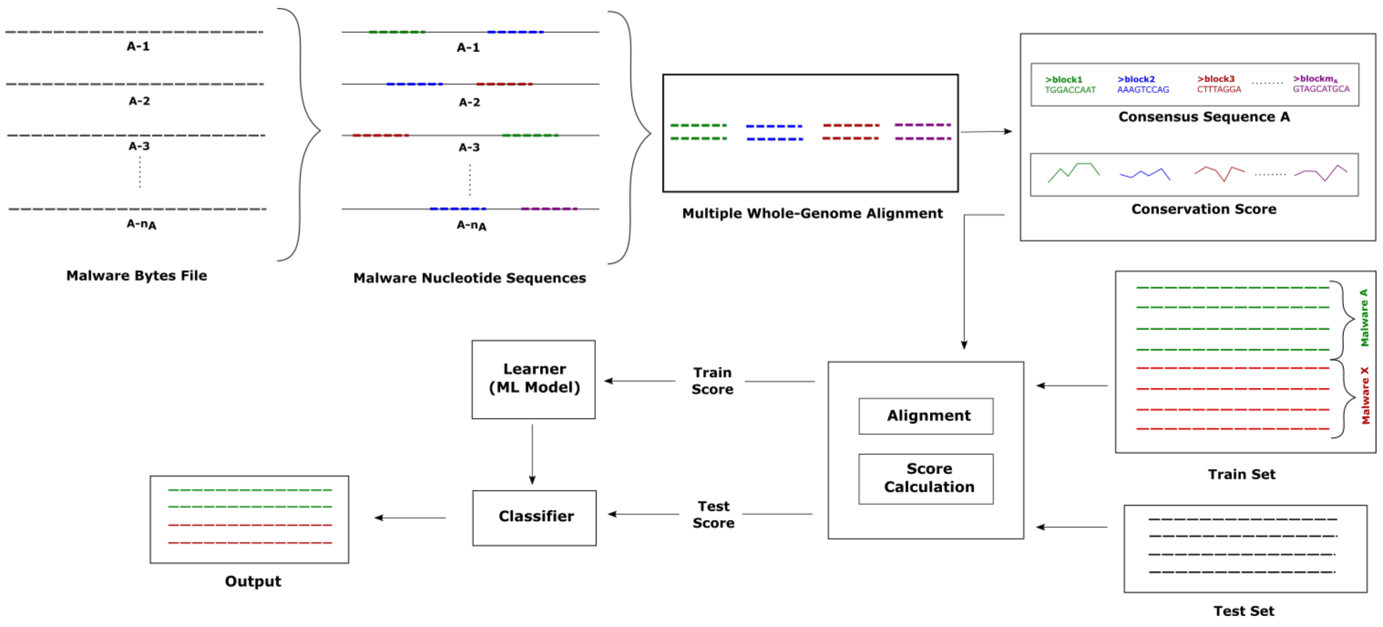


Fig. 1. Overview of MALIGN. (1) The malware bytes files (executables) from malwares of a particular family are first converted to nucleotide sequence files. (2) Then the malware nucleotide sequences are aligned using a multiple sequence alignment tool SibeliaZ. It first identifies similar sequences in different malwares to form blocks. Highly similar sequences (colored sequences) can be in different order in different files. The sequences in each block are then aligned. (3) The aligned sequences in each block are used to construct consensus sequences and conservation scores are calculated for each conserved block. (4) Then two sets of sequences - one corresponding to the malware family of interest and the other corresponding to non-malwares or malwares from other families are aligned to the consensus sequences and the degrees of conservation of each conserved block in the training sequences are estimated. (5) Finally a machine learning model is learnt to classify sequences based on the alignment scores of the sequences with the blocks. To classify new instances, sequences are aligned to the consensus sequences of the blocks and alignments are scored. The scores are then used as features for the class prediction.

Figure 2 illustrates alignment of three different malware nucleotide sequence files from the same family. The sequences share blocks of similar sequences showed in dashed lines of same color. They may also contain sequences unique to each sequence indicated by lines with different colors.

During the block construction process:

- The order of the shared blocks may differ in different sequences and the blocks may not be shared across all sequences. This makes it robust to evasion attempts by shuffling blocks of code.
- The shared blocks may not be fully conserved i.e. there may be mismatches of characters to some extent which means minor alteration to the code will not prevent detection of blocks.

SibeliaZ first identifies the shared linear blocks and then performs multiple sequence alignment of locally co-linear blocks. The block coordinates are output in GFF format and the alignment is in MAF format. The GFF format is a file format used for describing genes and other features of DNA, RNA and protein sequences. The multiple alignment format (MAF) is a format for describing multiple alignments in a way that is easy to parse and read. In our case, this format stores multiple alignment blocks at the byte code level among malwares. We generate such an MAF file for each malware family using training samples and identify the blocks of codes that are highly conserved across the malware family.

Consensus sequence and score generation

We process all sequences of the alignment blocks of MAF file from the previous step and generate a new sequence for each block, which is known as *consensus sequence* [48]. At first, we scan the length of all sequences and find the maximum one that will be the length of our consensus sequence. Then we traverse through the coordinates of every sequence and find the nucleotide of highest occurrence for each coordinate. We put the most frequently occurring nucleotide in corresponding index of the consensus sequence.

In Figure 2, consensus sequence generation of alignment block for Block-1 is shown in detail. Below the alignment block, the corresponding sequence logo is shown. The height of the individual letters in sequence logo represents how common the corresponding letter is at that particular coordinate of the alignment.

We thus construct the consensus sequence by taking the letter (nucleotide) with highest frequency for each coordinate. Similarly, the consensus sequences for all blocks are generated and are stored in a file in FASTA format with a unique id. These consensus sequences are the conserved part of the malware family which can be considered as the signature or common pattern of that family. The files are used in subsequent steps to classify malwares.

In addition to the consensus sequence, we calculate conservation scores for the blocks. In bioinformatics, conservation score is used when evaluating sites in a multiple sequence alignment, in order to identify residues critical for structure

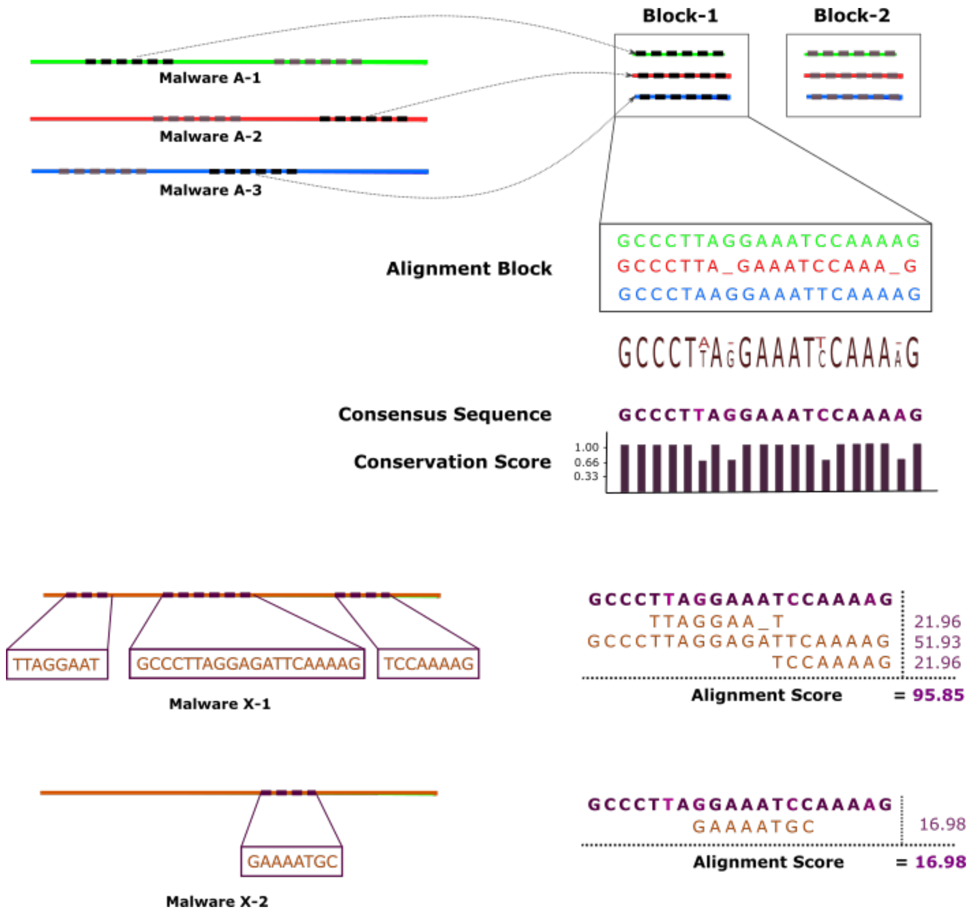


Fig. 2. Details of consensus sequence, conservation score, and alignment score generation from alignment blocks.

or function. This is calculated per base, indicating how many species in a given multiple alignment match at each locus. In malware world, conservation score can indicate the significance or importance of a code segment, or function call, or API call in a malware family. The responsible code segments of a malware will have high conservation scores compared to the segments those are not frequent, or conserved in malware files.

In Figure 2, the height of the bars of conservation score indicates the degree of conservation at the corresponding position. For each coordinate, we store the score for each of the four nucleotides which is given by the occurrence percentage of that nucleotide at that coordinate. So, at the i^{th} index of the alignment block B, the score for nucleotide $N(= A, C, G, T)$ is given by:

$$ConservationScore_{B,\ i,\ N} = \frac{\#\ of\ occurences\ of\ N\ at\ the\ i^{th}\ index\ in\ block\ B}{\#\ of\ sequences\ in\ alignment\ block\ B}$$

For example, in Figure 2, Block-1 has 3 sequences in total. Since at the 1^{st} index the block contains 3 Gs,

$$Score_{1,1,G} = 3/3 = 1.00$$

Again at the 6^{th} index the block contains 2 Ts and 1 A. So

$$Score_{1,6,T} = 2/3 = 0.66$$

$$Score_{1,6,A} = 1/3 = 0.33$$

Alignment with consensus sequences

Once the consensus sequence and the conservation scores are obtained, we take a training set for each malware family. In the training set, the positive examples are samples from that malware family and the negative examples are non-malwares or malwares from other families. All samples from the training set are aligned to the consensus sequences of the corresponding family to get the aligned blocks for each sample. Using the previously generated conservation score, we calculate new scores called alignment scores for each block for all samples which will be used as features.

In Figure 1 the green and red lines indicate the positive and negative samples in the training set respectively. These samples are then aligned with the consensus sequences using the alignment tool, SibeliaZ which outputs alignments for each sample. An example of alignment score calculation is shown in Figure 2. Malware X-1 and X-2 are positive and negative sample respectively. X-1 has three aligned sequences with the consensus sequence (shown in purple) whereas X-2 has only one. Sum of scores for all aligned sequences will be the score for that sample for the corresponding block. As an example, for total score of sample X-1, we sum the scores of 3 aligned sequences. Each aligned sequence's score is the sum of the score of all coordinates.

The aligned sequence score is then multiplied by the number of sequences that constructed the corresponding block since the higher the number of sequences that generated the block, the more conserved the sequence is across the instances from that family. In Figure 2, adding all coordinate's score of sample X-1's first aligned sequence, we get 7.32. Since the corresponding consensus sequence was generated from 3 sequences, the final score for first aligned sequence will be $21.96 (= 7.32 \times 3)$. Finally, the total alignment score for the block was calculated by adding the scores of all 3 aligned sequences.

In general, the total alignmment score of a sample Z for consensus sequence C_B from the alignment block B is given by

$$AlignmentScore_{Z,\ C_B} = \sum_{s \in S} \left(\sum_{i=1}^{length(s)} ConservationScore_{B,\ j,\ s_i} \right) \times number\ of\ sequences\ in\ block\ B$$

where, S is the set of sequences from sample Z that got aligned with C_B, s_i is the i-th nucleotide of the sequence s and j is the index of C_B where s_i was aligned.

Along with this score, we also store the total number of times the consensus sequence of a block gets aligned with the sample. In Figure 2, the consensus sequence was aligned with sample X-1 three times. Both the number of occurrences and the total alignment score for the consensus sequence of each block for a malware family are used as features for the subsequent classification, resulting in $2m$ features if multiple sequence alignment of a malware family has m blocks.

Classification

Finally, we learn machine learning models for each malware family to classify malwares. The scores and number of alignments calculated as mentioned above are used as the features in our classifiers.

We experimented with a number of machine learning models including logistic regression, support vector machines (SVM), decision tree, deep learning. Since the results did not vary significantly across models (see III in Results), we use logistic regression as our primary model because of its simplicity and interpretability.

After the training phase, we get classifiers that can be used detect or classify new instances as shown in Figure 1. The scores of the train and the test examples are calculated in the same way. The scores for new instances are passed to the classifiers to classify them into positive and negative instances. If a new sample is classified as negative by classifiers for all families, it can be considered benign.

V. Results

In the following sections, we first discuss the datasets used in this paper and subsequently present the results on these datasets.

Datasets

The Kaggle Microsoft Malware Classification Challenge (Big 2015): The Kaggle Microsoft Malware Classification Challenge (Big 2015) [47] aimed to organize polymorphic malwares into 9 separate classes of malicious programs at a high level (see Table II). This challenge simulates the file input data processed on over 160 million computers by Microsoft's real-time anti-malware detection products inspecting over 700 million computers per month.

Family Name	No of Train Samples	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos _ver1	398	Backdoor
Obfuscator.ACY	1228	Any kind of obfuscated malware
Gatak	1013	Backdoor

TABLE II
Malware Families in the Kaggle Dataset

Microsoft provided almost half a terabyte of input training and classification input data when uncompressed. They included:

1) *Binary Files:* 10,868 training files containing the raw hexadecimal representation of the file's binary content.
2) *Assembly Files:* 10,868 training files containing data extracted by the Interactive Disassembler (IDA) Tool. This information includes assembly command sequences, function calls and more.
3) *Training Labels:* Each training file name is a MD5 hash of the actual program. Each MD5 hash and the malware class it maps to are stored in the training label file.

From this, we constructed 9 balanced datasets for binary classification consisting of equal number of positive and negative samples for each of the 9 malware families. In each dataset, the positive examples are all the samples from the corresponding family and the negative examples were chosen by randomly sampling from the 8 other families in the dataset. Then 20% of each dataset is set aside as the test sets while the remaining 80% were used as the training sets. For the machine learning approaches that require hyper-parameter selection, the 80% was further split into training (60%) and validation sets (20%).

Microsoft Machine Learning Security Evasion Competition (2020) (MLSec) Dataset: While the Kaggle Microsoft Malware Classification Challenge (Big 2015) dataset is a large and widely studied one, often malwares families only have a few samples - especially when they emerge initially. Therefore it is important to assess the performance of the methods on datasets

Family Name	Train Accuracy			Test Accuracy		
	Logistic regression	Decision tree	SVM	Logistic regression	Decision tree	SVM
Ramnit	99.91	99.91	99.91	99.64	99.82	99.64
Kelihos_ver3	99.83	99.94	99.81	99.27	99.27	99.27
Vundo	100	100	100	97.4	97.4	97.4
Simda	100	100	97.92	84.62	84.62	84.62
Tracur	100	100	99.5	97.3	94.6	96.3
Kelihos _ver1	100	100	99.5	96.7	98.9	98.9
Obfuscator.ACY	100	100	100	95.9	92.7	96
Gatak	99.17	99.17	99.17	96.2	96.2	96.2
Overall	99.82	99.86	99.74	97.99	97.42	98.02

TABLE III
CLASSIFICATION ACCURACY ON KAGGLE MICROSOFT MALWARE (BIG 2015) DATASET FOR DIFFERENT MACHINE LEARNING MODELS

with small number of instances per family. So, we applied our method on the Microsoft Machine Learning Security Evasion Competition (2020) [49] (MLSec) dataset, too. Here we used the dataset from 'Defender Challenge' which consisted of malware bytes code and their variants. Defenders' challenge was to create a solution model that can defend against evasive variants created by the attackers. 49 malwares along with their evasive variants (submitted by the attackers) were found in this 'Defender Challenge' dataset. This dataset contains 49 original malwares with unique id from '01' to '49'. Each malware contains a different number of evasive variants varying from 5 to 20. On average, a malware has 12 variants in this dataset. Similarly to the Kaggle Microsoft Malware dataset, 49 datasets were created which were then split into training, validation and test sets.

Evaluation of machine learning algorithms

First we assess the performance of various machine learning approaches on the Kaggle Microsoft Malware (Big 2015) dataset. The binary files of this dataset were converted to nucleotide sequence files and labelled using 'Training Labels' data. Then we generated common alignment blocks using SibeliaZ and constructed the consensus sequences as discussed in Methods. We generated the conservation scores for each consensus sequences using frequency of nucleotides which were then used as features in the machine learning models.

We experimented with logistic regression, decision trees and support vector machines (SVM). Table III shows the train and test accuracy for 80%-20% train-test split on Kaggle Microsoft Malware Classification Challenge (Big 2015) Dataset. We were unable to align instances of the 'Lollipop' family by SibeliaZ possibly due to the limitation of computational resources. Hence, the family was removed from our analysis. We observe that the algorithms show similar performances in terms of accuracy. So, we selected logistic regression for future experiments because of its simplicity and interpretability. We experimented with the hyper-parameters of logistic regression and found that it gave the best results for 'elasticnet' penalty, C=0.05 (regularization factor), 'saga' solver and l1_ratio=0.5.

Comparison with existing approaches

Next we compare the performance of MALIGN with that of state of the art approaches, MalConv [32] and M-CNN, a convolutional neural network (CNN) based approach relying on conversion to images [27] on the Kaggle Microsoft Malware (Big 2015) dataset. It is worth noting that models with multiclass loss as low as 0.00283 have been reported for this dataset. However, we compare with MalConv and M-CNN, as they have been successfully applied to many different datasets.

We also implement a deep learning based approach that classifies malwares using the alignment scores calculated by MALIGN. The architectures of the deep leaning based approach on alignment scores as well as architectures of MalConv and M-CNN are shown in Figure 3.

The training and test accuracy of MALIGN with logistic regression and deep learning along with those of MalConv and M-CNN are shown in Tables IV and V. Table V shows that MALIGN has better accuracy on the test set than other approaches. MALIGN (Deep Learning) has the best accuracy of 98.59% followed by 97.99% accuracy of MALIGN (Logistic Regression).

Applicability with limited amount of data

Although deep learning based approaches have been widely applied for malware classification and detection, they require extensive amount of data for training and tend to overfit in absence of that. Tables IV and V show that MalConv and M-CNN perform well for most malware families. However, we observe that, for Type 5 (Sinda), which has only 42 samples, the test accuracy of MalConv and M-CNN are 52.94% and 62.5% respectively only whereas MALIGN has 84.62% test accuracy. Both the other methods have training accuracy of 100% indicating overfitting.

We also compare performances of the methods on the MLSec dataset (Microsoft Machine Learning Security Evasion Competition (2020) [49]). Since this dataset contains limited number of variants created in almost real-time, this can be used to identify how our method works on zero-day malwares when only a limited number of samples are available.

Because of the limited number of instances in this dataset, the validation set is very small for some types. So we run the deep learning models on a 80%-20% train-test split as well as 60%-20%-20% train-validation-test split of the data. Tables VI and VII summarizes the performances of the models on all 49 types for train-test and train-validation-test split respectively whereas Figures 4 and 5 provide radar charts showing training and test accuracy in all 49 types individually.

Family Name	MAlign (Logistic Regression)	MAlign (Deep Learning)	MalConv	M-CNN
Ramnit	99.91	99.58	98.39	97.64
Kelihos_ver3	99.83	99.86	99.89	99.91
Vundo	100	98.26	99.4	99.26
Simda	100	94.44	100	100
Tracur	100	98.18	98.74	98.43
Kelihos _ver1	100	98.92	98.54	99.52
Obfuscator.ACY	100	98.66	97.33	99.70
Gatak	99.17	99.2	99.15	92.69
Overall	**99.82**	99.24	98.96	98.4

TABLE IV
PERFORMANCE OF DIFFERENT MODELS ON TRAIN-SET OF KAGGLE MICROSOFT MALWARE CLASSIFICATION CHALLENGE DATASET (BIG 2015)

Family Name	MAlign (Logistic Regression)	MAlign (Deep Learning)	MalConv	M-CNN
Ramnit	99.64	99.46	95.66	88.44
Kelihos_ver3	99.27	99.82	100	99.72
Vundo	97.4	98.70	94.89	97.21
Simda	84.62	84.62	52.94	62.5
Tracur	97.3	98.2	93.91	94.55
Kelihos _ver1	96.7	95.7	96.08	94.67
Obfuscator.ACY	95.9	96.39	94.42	91.74
Gatak	96.2	98.37	98.67	88.68
Overall	97.99	**98.59**	96.95	94.10

TABLE V
PERFORMANCE OF DIFFERENT MODELS ON TEST-SET OF KAGGLE MICROSOFT MALWARE CLASSIFICATION CHALLENGE DATASET (BIG 2015)

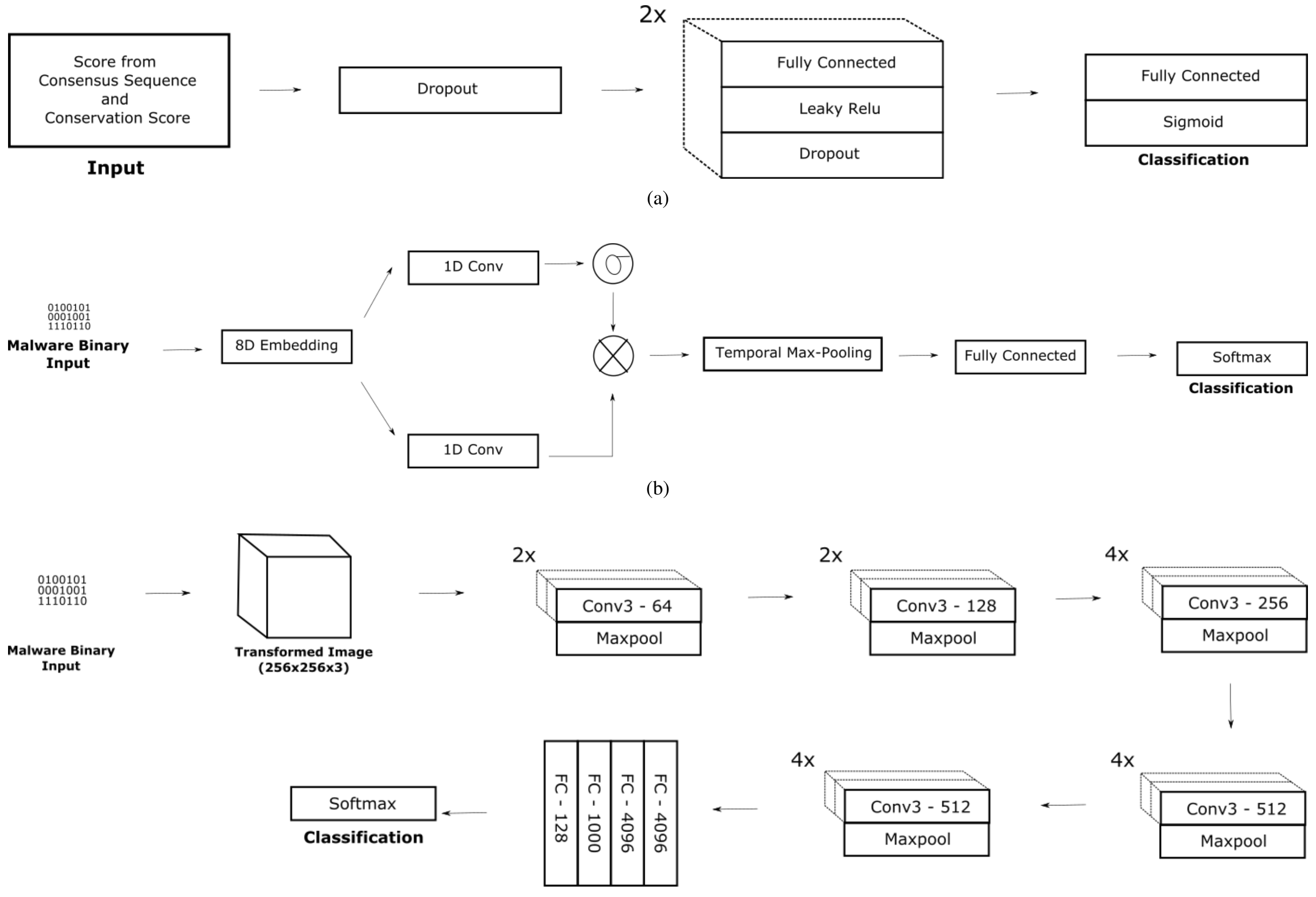


Fig. 3. Architectures of (a) deep learning model on alignment scores, (b) the MalConv model [32], and (c) the M-CNN model [27].

	Train Accuracy	Test Accuracy
MALIGN(Logistic Regression)	98.18	**80.42**
MALIGN(Deep Learning)	97.72	80.00
MalConv	**98.24**	79.22
M-CNN	96.99	71.09

TABLE VI
PERFORMANCE OF MODELS ON MLSEC DATASET FOR TRAIN-TEST SPLIT

	Train Accuracy	Validation Accuracy	Test Accuracy
MALIGN (Deep Learning)	97.53	**81.67**	**79.58**
MalConv	95.39	81.22	64.45
M-CNN	**98.4**	74.29	73.83

TABLE VII
PERFORMANCE OF MODELS ON MLSEC DATASET FOR TRAIN-VALIDATION-TEST SPLIT

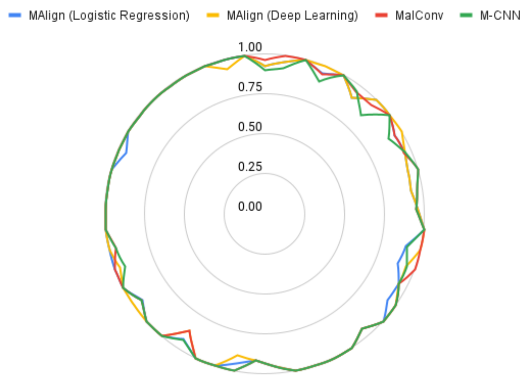


Fig. 4. Radar chart showing accuracy of different models on each of the 49 malware types along the perimeter on MLSec-Train Dataset

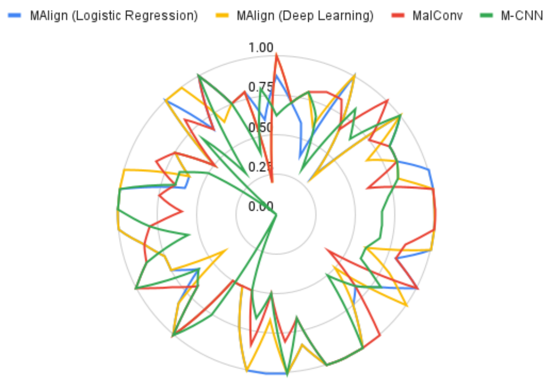


Fig. 5. Radar chart showing accuracy of different models on each of the 49 malware types along the perimeter on MLSec-Test Dataset

We observe that MALIGN (Logistic Regression) outperforms the three deep learning based models overall on this dataset regardless of the splitting of the MLSec dataset. This highlights the advantage of explicit identification of critical code blocks when data is limited. From Figures 4 and 5, we observe that on the train set, all approaches have consistent performance, but on the test set M-CNN and MalConv is relatively inconsistent. For example, type 29 and 43 have 10 and 5 available variants respectively, and M-CNN's test accuracy is 0 on both.

Robustness to adversarial attacks

A major issue with deep learning based malware detection approaches is - they can often be evaded by adding random content to it. Since MALIGN relies on finding conserved blocks in the malware families for classification, it should in principle be inherently robust to such attacks.

We tried to investigate the robustness of our method compared to conventional malware detection techniques. We used the evasion technique on MalConv model that was proposed by Kolosnjaji et al. [1]. It creates adversarial samples just by modifying (or padding) approximately 1.25% of the total size of malwares which can successfully evade the MalConv model with high probability. The evasion rate increases with the percentage of modification on malware samples. We experimented with their implementation [50] on some of the types in the MLSec dataset and the results are shown in Table VIII.

Type	Evaded/Total Train set	Evaded/Total Test set	Evasion Rate
1	5/15	2/4	36.84%
11	1/9	1/2	18.18%
12	5/13	2/4	41.18%
28	5/8	2/2	70%
45	4/5	2/2	85.71%

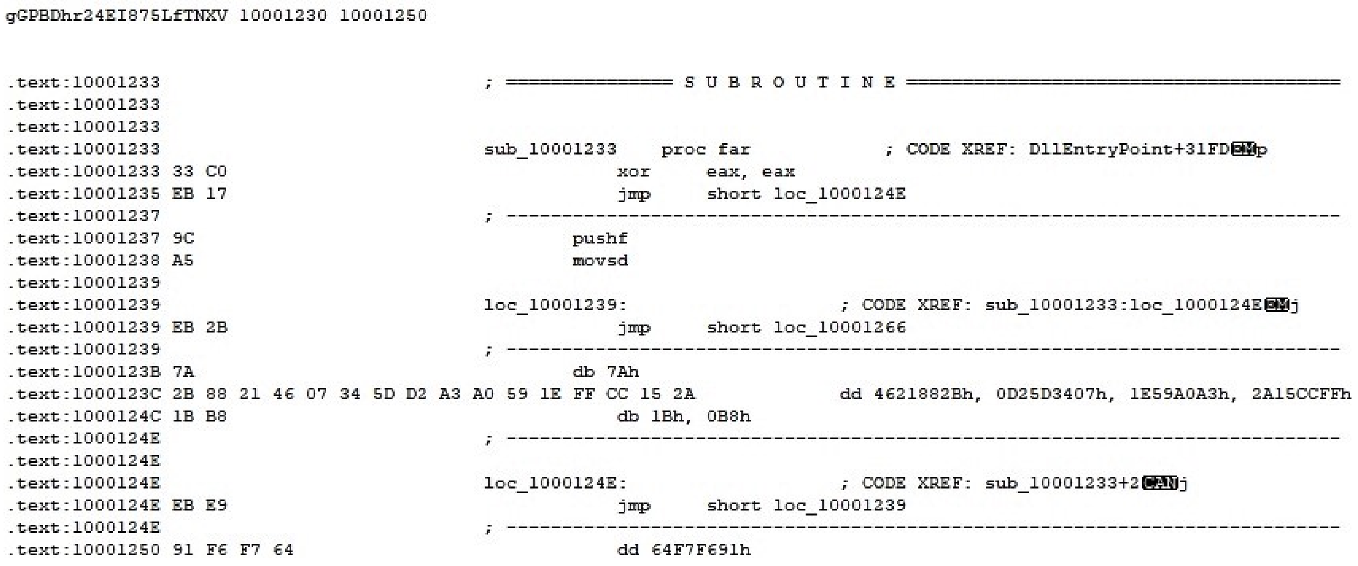
TABLE VIII
GRADIENT ATTACK RESULTS ON MALCONV ON SOME TYPES IN THE MLSEC DATASET

We then applied MALIGN on these evasive samples and all of them were successfully detected with almost 100% prediction probability. It is worth mentioning that since the evasion technique did not have access to the MALIGN model, the experiment is not rigorous. However, since MALIGN finds the blocks critical for function in a malware family and classifies them based on those, to evade MALIGN, the malware attackers will have to go through an incommodious process of changing those blocks without changing its intended features. Moreover, gradient attack based evasion techniques cannot directly be applied on MALIGN because we are not feeding the malware file directly to our model unlike many other techniques. There are plausible approaches to attack the present implementation which can be addressed using classifiers with only non-negative weights and other techniques in the future.

VI. CASE STUDIES

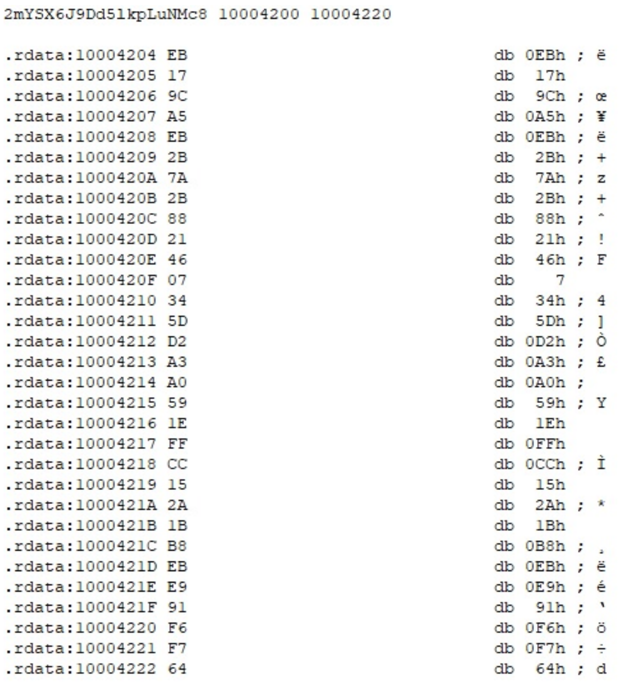
An important advantage of MALIGN is - it is interpretable and insights can be derived about malware families through a *backtracking* process. The process is as follows:

(i) Find the blocks that are assigned high weights by the logistic regression model
(ii) Select the blocks that are highly conserved from the above list

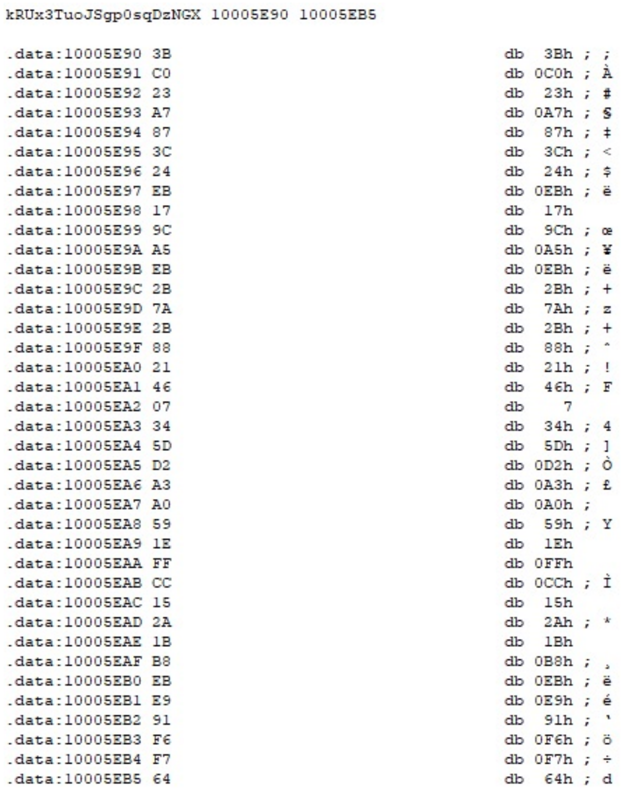


```
gGPBDhr24EI875LfTNXV 10001230 10001250

.text:10001233                          ; ============== S U B R O U T I N E =======================================
.text:10001233
.text:10001233
.text:10001233                          sub_10001233    proc far            ; CODE XREF: DllEntryPoint+31FDEMp
.text:10001233 33 C0                                    xor     eax, eax
.text:10001235 EB 17                                    jmp     short loc_1000124E
.text:10001237                          ; ---------------------------------------------------------------------------
.text:10001237 9C                                   pushf
.text:10001238 A5                                   movsd
.text:10001239
.text:10001239                          loc_10001239:                   ; CODE XREF: sub_10001233:loc_1000124EEMj
.text:10001239 EB 2B                                    jmp     short loc_10001266
.text:10001239                          ; ---------------------------------------------------------------------------
.text:1000123B 7A                                   db 7Ah
.text:1000123C 2B 88 21 46 07 34 5D D2 A3 A0 59 1E FF CC 15 2A          dd 4621882Bh, 0D25D3407h, 1E59A0A3h, 2A15CCFFh
.text:1000124C 1B B8                                    db 1Bh, 0B8h
.text:1000124E                          ; ---------------------------------------------------------------------------
.text:1000124E
.text:1000124E                          loc_1000124E:                   ; CODE XREF: sub_10001233+2CANj
.text:1000124E EB E9                                    jmp     short loc_10001239
.text:1000124E                          ; ---------------------------------------------------------------------------
.text:10001250 91 F6 F7 64                              dd 64F7F691h
```

(a) Code in .text segment



```
2mYSX6J9Dd5lkpLuNMc8 10004200 10004220

.rdata:10004204 EB                        db 0EBh ; ë
.rdata:10004205 17                        db  17h
.rdata:10004206 9C                        db  9Ch ; œ
.rdata:10004207 A5                        db 0A5h ; ¥
.rdata:10004208 EB                        db 0EBh ; ë
.rdata:10004209 2B                        db  2Bh ; +
.rdata:1000420A 7A                        db  7Ah ; z
.rdata:1000420B 2B                        db  2Bh ; +
.rdata:1000420C 88                        db  88h ; ˆ
.rdata:1000420D 21                        db  21h ; !
.rdata:1000420E 46                        db  46h ; F
.rdata:1000420F 07                        db    7
.rdata:10004210 34                        db  34h ; 4
.rdata:10004211 5D                        db  5Dh ; ]
.rdata:10004212 D2                        db 0D2h ; Ò
.rdata:10004213 A3                        db 0A3h ; £
.rdata:10004214 A0                        db 0A0h ;
.rdata:10004215 59                        db  59h ; Y
.rdata:10004216 1E                        db  1Eh
.rdata:10004217 FF                        db 0FFh
.rdata:10004218 CC                        db 0CCh ; Ì
.rdata:10004219 15                        db  15h
.rdata:1000421A 2A                        db  2Ah ; *
.rdata:1000421B 1B                        db  1Bh
.rdata:1000421C B8                        db 0B8h ; ¸
.rdata:1000421D EB                        db 0EBh ; ë
.rdata:1000421E E9                        db 0E9h ; é
.rdata:1000421F 91                        db  91h ; '
.rdata:10004220 F6                        db 0F6h ; ö
.rdata:10004221 F7                        db 0F7h ; ÷
.rdata:10004222 64                        db  64h ; d
```



```
kRUx3TuoJSgp0sqDzNGX 10005E90 10005EB5

.data:10005E90 3B                         db  3Bh ; ;
.data:10005E91 C0                         db 0C0h ; À
.data:10005E92 23                         db  23h ; #
.data:10005E93 A7                         db 0A7h ; §
.data:10005E94 87                         db  87h ; ‡
.data:10005E95 3C                         db  3Ch ; <
.data:10005E96 24                         db  24h ; $
.data:10005E97 EB                         db 0EBh ; ë
.data:10005E98 17                         db  17h
.data:10005E99 9C                         db  9Ch ; œ
.data:10005E9A A5                         db 0A5h ; ¥
.data:10005E9B EB                         db 0EBh ; ë
.data:10005E9C 2B                         db  2Bh ; +
.data:10005E9D 7A                         db  7Ah ; z
.data:10005E9E 2B                         db  2Bh ; +
.data:10005E9F 88                         db  88h ; ˆ
.data:10005EA0 21                         db  21h ; !
.data:10005EA1 46                         db  46h ; F
.data:10005EA2 07                         db    7
.data:10005EA3 34                         db  34h ; 4
.data:10005EA4 5D                         db  5Dh ; ]
.data:10005EA5 D2                         db 0D2h ; Ò
.data:10005EA6 A3                         db 0A3h ; £
.data:10005EA7 A0                         db 0A0h ;
.data:10005EA8 59                         db  59h ; Y
.data:10005EA9 1E                         db  1Eh
.data:10005EAA FF                         db 0FFh
.data:10005EAB CC                         db 0CCh ; Ì
.data:10005EAC 15                         db  15h
.data:10005EAD 2A                         db  2Ah ; *
.data:10005EAE 1B                         db  1Bh
.data:10005EAF B8                         db 0B8h ; ¸
.data:10005EB0 EB                         db 0EBh ; ë
.data:10005EB1 E9                         db 0E9h ; é
.data:10005EB2 91                         db  91h ; '
.data:10005EB3 F6                         db 0F6h ; ö
.data:10005EB4 F7                         db 0F7h ; ÷
.data:10005EB5 64                         db  64h ; d
```

(b) Code in .rdata segment

(c) Code in .data segment

Fig. 6. Code obfuscation in data segment (a) A code fragment in a malware sample, (b) and (c) Same code obfuscated in `.rdata` and `.data` segments indicated by the hex-codes

(iii) Process the alignment file (MAF) to determine the sequences and their indices that constructed the blocks
(iv) Locate the code fragments corresponding to the sequences found in Step (iii)

This process can be used to uncover potential malicious code as discussed next.

Detection of obfuscated malicious code

Different techniques and methods are used by attackers or malware creators to obfuscate malicious, harmful code segments to evade anti-malware tools. MAlign, our proposed method gives us the ability to find responsible malicious code segments from malware files by the backtracking process sketched above and analyze the blocks of code that are highly conserved.

We backtracked from the aligned blocks to the assembly code on some randomly selected samples from the Kaggle Microsoft Malware (Big 2015) dataset. In some cases, we found evidence of malware obfuscation. Figure 6 is an example of such case. Figure 6a, 6b and 6c are snapshots of three different malware files from 'Vundo' malware family of 'Trojan' type. All samples contain the same hex-code but in different segments. Figure 6b and 6c conceal the same code string of Figure 6a in its `.rdata` and `.data` segment, respectively possibly to evade anti-malware tools.

VII. Conclusions

In this paper, we presented a malware detection tool MAlign based on a recently developed multiple whole-genome alignment tool, SibeliaZ [20]. Although sequence alignment based approaches have been used for malware analysis in the past, the use of a whole-genome alignment tool makes MAlign scalable to long malware sequences and protects against trivial adversarial attacks such as shuffling of code blocks. The method is also interpretable and can be used to derive insights on malwares such as identification of critical code blocks and code obfuscation.

We have applied MAlign on the Kaggle Microsoft Malware Classification Challenge (Big 2015) and the Microsoft Machine Learning Security Evasion Competition (2020) (MLSec) datasets, and observed that it outperforms widely used deep learning based methods MalConv and M-CNN.

Preliminary experiments show MAlign is robust to common adversarial attacks such as padding and modification of sequences. In future, the method may be tested against other possible types evasion techniques and the machine learning algorithms can be adjusted accordingly. In addition, other whole-genome alignment tools such as Progressive Cactus [19] may be experimented with.

References

[1] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[2] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial perturbations against deep neural networks for malware classification," *arXiv preprint arXiv:1606.04435*, 2016.

[3] A. Huang, A. Al-Dujaili, E. Hemberg, and U.-M. O'Reilly, "On visual hallmarks of robustness to adversarial malware," *arXiv preprint arXiv:1805.03553*, 2018.

[4] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[5] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv preprint arXiv:1801.08917*, 2018.

[6] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[7] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, "Attack and defense of dynamic analysis-based, adversarial neural malware detection models," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 1–8.

[8] W. Hu and Y. Tan, "Black-box attacks against rnn based malware detection algorithms," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art api call based malware classifiers," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 490–510.

[10] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[11] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of Molecular Biology*, vol. 48, no. 3, pp. 443–453, 1970.

[12] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[13] J. D. Thompson, D. G. Higgins, and T. J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Research*, vol. 22, no. 22, pp. 4673–4680, 1994.

[14] K. Katoh, K. Misawa, K.-i. Kuma, and T. Miyata, "MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform," *Nucleic Acids Research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[15] R. C. Edgar, "MUSCLE: multiple sequence alignment with high accuracy and high throughput," *Nucleic Acids Research*, vol. 32, no. 5, pp. 1792–1797, 2004.

[16] A. L. Delcher, S. Kasif, R. D. Fleischmann, J. Peterson, O. White, and S. L. Salzberg, "Alignment of whole genomes," *Nucleic Acids Research*, vol. 27, no. 11, pp. 2369–2376, 1999.

[17] A. C. Darling, B. Mau, F. R. Blattner, and N. T. Perna, "Mauve: multiple alignment of conserved genomic sequence with rearrangements," *Genome Research*, vol. 14, no. 7, pp. 1394–1403, 2004.

[18] B. Paten, D. Earl, N. Nguyen, M. Diekhans, D. Zerbino, and D. Haussler, "Cactus: Algorithms for genome multiple sequence alignment," *Genome Research*, vol. 21, no. 9, pp. 1512–1528, 2011.

[19] J. Armstrong, G. Hickey, M. Diekhans, I. T. Fiddes, A. M. Novak, A. Deran, Q. Fang, D. Xie, S. Feng, J. Stiller *et al.*, "Progressive Cactus is a multiple-genome aligner for the thousand-genome era," *Nature*, vol. 587, no. 7833, pp. 246–251, 2020.

[20] I. Minkin and P. Medvedev, "Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ," *Nature communications*, vol. 11, no. 1, pp. 1–11, 2020.

[21] C. Wressnegger, K. Freeman, F. Yamaguchi, and K. Rieck, "Automatically inferring malware signatures for anti-virus assisted attacks," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, 2017, pp. 587–598.

[22] M. Zakeri, F. Faraji Daneshgar, and M. Abbaspour, "A static heuristic approach to detecting malware targets," *Security and Communication Networks*, vol. 8, no. 17, pp. 3015–3027, 2015.

[23] L. Martignoni, E. Stinson, M. Fredrikson, S. Jha, and J. C. Mitchell, "A layered architecture for detecting malicious behaviors," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2008, pp. 78–97.

[24] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using cwsandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[25] M. G. Schultz, E. Eskin, F. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*. IEEE, 2000, pp. 38–49.

[26] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, "Malware images: visualization and automatic classification," in *Proceedings of the 8th international symposium on visualization for cyber security*, 2011, pp. 1–7.

[27] M. Kalash, M. Rochan, N. Mohammed, N. D. Bruce, Y. Wang, and F. Iqbal, "Malware classification with deep convolutional neural networks," in *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE, 2018, pp. 1–5.

[28] R. K. Shahzad, N. Lavesson, and H. Johnson, "Accurate adware detection using opcode sequence extraction," in *2011 Sixth International Conference on Availability, Reliability and Security*. IEEE, 2011, pp. 189–195.

[29] R. Lu, "Malware detection with lstm using opcode language," *arXiv preprint arXiv:1906.04593*, 2019.

[30] I. Santos, J. Devesa, F. Brezo, J. Nieves, and P. G. Bringas, "Opem: A static-dynamic approach for machine-learning-based malware detection," in *International joint conference CISIS'12-ICEUTE´ 12-SOCO´ 12 special sessions*. Springer, 2013, pp. 271–280.

[31] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, 2018.

[32] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018, pp. 268–276.

[33] T. Bai, J. Luo, J. Zhao, B. Wen, and Q. Wang, "Recent advances in adversarial training for adversarial robustness," *arXiv preprint arXiv:2102.01356*, 2021.

[34] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *European symposium on research in computer security*. Springer, 2017, pp. 62–79.

[35] Y. Zhang, H. Li, Y. Zheng, S. Yao, and J. Jiang, "Enhanced dnns for malware classification with gan-based adversarial training," *Journal of Computer Virology and Hacking Techniques*, vol. 17, no. 2, pp. 153–163, 2021.

[36] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," in *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 2018, pp. 76–82.

[37] J. Cohen, E. Rosenfeld, and Z. Kolter, "Certified adversarial robustness via randomized smoothing," in *International Conference on Machine Learning*. PMLR, 2019, pp. 1310–1320.

[38] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust {PDF} malware classifiers," in *29th {USENIX} Security Symposium ({USENIX} Security 20)*, 2020, pp. 2343–2360.

[39] Í. Íncer Romeo, M. Theodorides, S. Afroz, and D. Wagner, "Adversarially robust malware detection using monotonic classification," in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, 2018, pp. 54–63.

[40] Y. Chen, A. Narayanan, S. Pang, and B. Tao, "Multiple sequence alignment and artificial neural networks for malicious software detection," in *2012 8th International Conference on Natural Computation*. IEEE, 2012, pp. 261–265.

[41] A. Narayanan, Y. Chen, S. Pang, and B. Tao, "The effects of different representations on malware motif identification," in *2012 Eighth International Conference on Computational Intelligence and Security*. IEEE, 2012, pp. 86–90.

[42] V. Naidu and A. Narayanan, "Further experiments in biocomputational structural analysis of malware," in *2014 10th International Conference on Natural Computation (ICNC)*. IEEE, 2014, pp. 605–610.

[43] D. Kirat and G. Vigna, "Malgene: Automatic extraction of malware analysis evasion signature," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 769–780.

[44] V. Naidu and A. Narayanan, "Needleman-wunsch and smith-waterman algorithms for identifying viral polymorphic malware variants," in *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech)*. IEEE, 2016, pp. 326–333.

[45] I. K. Cho, T. G. Kim, Y. J. Shim, M. Ryu, and E. G. Im, "Malware analysis and classification using sequence alignments," *Intelligent Automation & Soft Computing*, vol. 22, no. 3, pp. 371–377, 2016.

[46] H. Kim, J. Kim, Y. Kim, I. Kim, K. J. Kim, and H. Kim, "Improvement of malware detection and classification using api call sequence alignment and visualization," *Cluster Computing*, vol. 22, no. 1, pp. 921–929, 2019.

[47] "Microsoft malware classification challenge (big 2015)," accessed July 7, 2019 =https://www.kaggle.com/c/malwareclassification/.

[48] J. D. Kececioglu and E. W. Myers, "Combinatorial algorithms for dna sequence assembly," *Algorithmica*, vol. 13, no. 1, pp. 7–51, 1995.

[49] "Microsoft mmachine learning security evasion competition (2020)," https://mlsec.io/.

[50] https://github.com/yuxiaorun/MalConv-Adversarial.