MAALIGN: Explainable Static Raw-byte Based Malware Family Classification using Sequence Alignment

Shoumik Saha\textsuperscript{a,b,*}, Sadia Afroz\textsuperscript{c,d}, Atif Hasan Rahman\textsuperscript{b,*}

\textsuperscript{a}University of Maryland, College Park, Maryland, United States
\textsuperscript{b}Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh
\textsuperscript{c}ICSI, University of California, Berkeley, United States
\textsuperscript{d}Avast, United States

Abstract

For a long time, malware classification and analysis have been an arm's-race between antivirus systems and malware authors. Though static analysis is vulnerable to evasion techniques, it is still popular as the first line of defense in antivirus systems. But most of the static analyzers failed to gain the trust of practitioners due to their black-box nature. We propose MAALIGN, a novel static malware family classification approach inspired by genome sequence alignment that can not only classify malware families but can also provide explanations for its decision. MAALIGN encodes raw bytes using nucleotides and adopts genome sequence alignment approaches to create a signature of a malware family based on the conserved code segments in that family, without any human labor or expertise. We evaluate MAALIGN on two malware datasets, and it outperforms other state-of-the-art machine learning-based malware classifiers (by 4.49\%~0.07\%), especially on small datasets (by 19.48\%~1.2\%). Furthermore, we explain the generated signatures by MAALIGN on different malware families illustrating the kinds of insights it can provide to analysts, and show its efficacy as an analysis tool. Additionally, we evaluate its theoretical and empirical robustness against some common attacks. In this paper, we approach static malware analysis from a unique perspective, aiming to strike a delicate balance among performance, interpretability, and robustness.

Keywords: malware, sequence alignment, explainability, machine learning, adversarial

1. Introduction

Over the past decade, malware has been growing and spreading exponentially. For example, the AV-Test institute registers more than 450K new malware and potentially unwanted programs (PUP) every day (AV-Test, 2023). They found 16 million new malware in just the first 3 months of 2023.\textsuperscript{1}

To detect this rising number of malware at scale, static analysis is used as the first line of defense by all commercial antivirus (AV) systems, because it requires less computational power and time than dynamic analysis. However, dynamic analysis allows security analysts to perform more in-depth analysis by executing the malware in a sandbox or cloud, and capturing its behavior and activity (Or-Meir et al., 2019; Anderson et al., 2011; Avllazagaj et al., 2021). Additionally, static analysis is comparatively easier to evade than dynamic ones.

Like all other fields, machine learning (ML) is getting adapted to security too, including malware detection and family classification (Schultz et al., 2000; Nataraj et al., 2011; Pascanu et al., 2015; Kalash et al., 2018; Shahzad et al., 2011; Lu, 2019; Yan et al., 2018; Ahmadi et al., 2016; Raiff et al., 2018).

\textsuperscript{1}https://portal.av-atlas.org/malware

ML has proved helpful in detecting the rising number of malware at scale. In fact, most commercial malware detectors use machine learning for their static analysis nowadays. However, these deep-learning based black-box models lack interpretability, and conventional explanation methods for machine learning (Zhang et al., 2019) are not suitable in security (Guo et al., 2018a). Consequently, security practitioners cannot fully trust these models, especially in safety-critical applications where the ability to explain a classification error is crucial.

Unfortunately, while a large body of work has gone into designing highly accurate static malware classifiers by outperforming each other over the past few years, very little work has been done considering the explainability aspect. Existing works (Arp et al., 2014; Kinkead et al., 2021; Melis et al., 2018; Liu et al., 2022; Kumar et al., 2018) on explainable malware classifier are done on extracted features, e.g., requested and used permissions, API calls, network addresses, etc., rather than analyzing the executable code directly. Thus, they can only provide explainability at a higher-level semantics, while the potential insights and new knowledge that could be extracted from low-level input features remain unexplored. This untapped information could include valuable findings such as common practices among malware authors, prevalent obfuscation techniques, and other critical insights essential for analysts.

Another shortcoming of current static malware classifiers is that they are vulnerable to different adversarial attacks (Song...
et al., 2020; Suciu et al., 2019; Demetrio et al., 2021; Ceschin et al., 2019; Kolosnjai et al., 2018; Huang et al., 2018; Kreuk et al., 2018a). Though these adversarial attacks use different techniques and work in different settings, fundamentally, they fall under one umbrella – adding or modifying bytes (from benign files or generated using gradient approaches) without altering the malware semantics. Consequently, there have been works on proposing defenses for such attacks, e.g., monotonic classifier (Fleshman et al., 2018a), adversarial training (Lucas et al., 2018), etc. Unfortunately, most of these defenses suffer from low standard accuracy and can provide robustness to a limited number of attacks; which introduces a trade-off between robustness and accuracy.

Moreover, end-to-end deep-learning based models are data hungry (Marcus, 2018). The more data they get, the better they learn the features. However, this data hunger poses a concerning challenge in the context of malware family classification, particularly in the detection of a new variant. When a new malware variant or family emerges, it takes time for AV vendors to collect enough samples from the wild. Unfortunately, the prevalence of such new variants is on a steady rise. For example, in 2020, there was a 62% rise in detected malware variants. In March 2022, there was a new record of discovering 60,000 new malware variants. As a result, these deep-learning based approaches fail to perform at their fullest for such new malware variants due to their insufficient training set size.

All of these issues raise an important research question: "Can we design a novel static analysis technique that can effectively mitigate the shortcomings (non-interpretability, vulnerability, data-hungry) mentioned above?"

In this study, we focus on a raw-byte-based static malware family classification technique as a potential answer to this question. By exploring this approach, we aim to enhance the explainability and robustness of static analysis in combating the ever-evolving landscape of malware threats.

**Our Design.** We propose **MAALIGN**, a novel static malware family detection method that operates directly on the raw bytes from executables. Unlike conventional static analyzers, our approach leverages the sequence alignment mechanism from bioinformatics and incorporates it with a machine learning model. The rationale behind adopting such an alignment mechanism is the indistinct similarity between genome sequences and malware executables. Genomes contain critical regions for the survival of the organism, such as protein-coding genes where mutations (or changes in genes) can be lethal. Similarly, malware families contain malicious code blocks associated with the malicious functionalities of that family, and altering them can change their semantics or corrupt the files. Embracing the analogy between these two distinct domains, we capitalize on the advanced alignment methods developed in bioinformatics. Our rationale is that – this alignment technique will capture code blocks that are common in one malware family the way it does for genomes in species.

The alignment technique identifies the common blocks, i.e., consensus sequences, per family, and estimates the degree of conservation at each location by processing the generated alignments. Then we generate a conservation score for each of these blocks depending on their conservation and frequency in that family. In this context, these aligned blocks along with their scores can be considered as the features for that corresponding malware family. However, there can be aligned common blocks that are shared in all malware families (for example, common PE header), and hence, they will not represent any family-specific functionalities. Therefore, we train a classifier on these generated conservation scores from these aligned blocks, including different families, so that the classifier learns the importance of these generated blocks for a specific family.

During inference, to classify whether a new malware belongs to a family, we generate the alignment of this new malware with the previously generated common blocks of that family, and use it to compute a set of alignment scores. These scores are then fed to the classifier model to classify the new malware to its family.

Additionally, our adopted alignment approach can effectively handle addition, subtraction, substitution, and re-ordering in sequences, which provides better robustness to perturbation than conventional techniques. Furthermore, the use of such a back-trackable alignment approach with an interpretable ML model enables us to map its classification decision to the responsible suspicious code blocks. As a result, MAALIGN is trustworthy for its decision which is highly crucial for malware applications.

**Evaluation.** We evaluate MAALIGN for detecting malware families on two datasets: (i) Kaggle Microsoft Malware Classification Challenge (Big 2015) (Microsoft, 2015), and (ii) Microsoft Machine Learning Security Evasion Competition (2020) (MLSEC, 2020). In comparison to the state-of-the-art methods such as MalConv (Raff et al., 2018), CNN-based model (Kalash et al., 2018), and Feature-Fusion model (Ahmadi et al., 2016), our approach has higher accuracy, and the difference in performance is more noticeable when datasets are small. Using our auto-generated aligned blocks, i.e., consensus sequences, for each family, we backtracked to the exact code blocks that were responsible for our model decision. Thus, we were able to discover suspicious functionalities from malware samples, making our model explainable. Moreover, we theoretically prove that MAALIGN is robust by design to gradient attacks. We also evaluate the empirical robustness by generating adversarial malware using a gradient-based patch-append attack.

In summary, our main contributions are:

1. **Sequence alignment based approach.** We propose MAALIGN by adopting sequence alignment concepts from bioinformatics into the static malware family classifier. We used a multiple sequence alignment-based tool to find conserved regions in raw binary executables (Section 4).

2. **High accuracy.** We evaluated MAALIGN on two datasets, and it outperforms other state-of-the-art static detection methods. Besides, the performance gap gets higher with a lower amount of training data (Section 5).
3. Explainable by-design. Due to our design choice, MA\textit{lign} is backtrackable and explainable for its decision. We explore this in-depth and identify responsible code-blocks for multiple malware families. We share some of the interesting findings and insights that can help the security community (Section 6).

4. Robust by-design. We mathematically prove that our design choice makes MA\textit{lign} robust against gradient attacks. We also evaluate its empirical robustness against gradient-based patch attacks (Section 7).

2. Related Work

Machine Learning based classifiers. 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. (2017) and Zakeri et al. (2015) proposed a signature-based approach using static analysis. Later, a dynamic approach - malware detection by analyzing the malware behavior, was proposed by Martignoni et al. (2008) and Willems et al. (2007). Each time, machine learning based techniques are also being used to classify malware. Schultz et al. (2000) first proposed a data mining technique for malware detection using three different types of static features. Subsequently, Nataraj et al. (2011) proposed a malware classification approach based on image processing techniques by converting the bytes files to image files. Later, Kalash et al. (2018) improved on (Nataraj et al., 2011) by developing M-CNN using convolutional neural networks (CNN). Besides CNN, RNN has also been used for malware analysis. Shahzad et al. (2011) and Lu (2019) proposed techniques with LSTM using opcode sequences of malware. Santos et al. (2013) proposed a hybrid technique by integrating both static and dynamic analysis. Subsequently, Yan et al. (2018) developed MalNet using an ensemble of CNN, LSTM and by extracting metadata features, while Ahmadi et al. Ahmadi et al. (2016) extracted and selected features of malware depending on the importance and applied feature fusion on them. Recently, Raff et al. (2018) developed a state-of-the-art technique MalConv using only the raw byte sequence as the input to a CNN-based deep-learning model.

Sequence Alignment based Classifiers. In the past, sequence alignment based approaches have been used for malware analysis by a number of researchers (Chen et al., 2012; Narayanan et al., 2012; Naidu and Narayanan, 2014; Kirat and Vigna, 2015; Naidu and Narayanan, 2016; Cho et al., 2016; Kim et al., 2019). Chen et al. (2012) used multiple sequence alignment to align computer viral and worm codes of variable lengths to identify invariant regions. This approach was subsequently enhanced later (Narayanan et al., 2012; Naidu and Narayanan, 2014, 2016). However, these approaches were not scalable enough to train on large malware sets and did not evaluate the robustness of their models, unlike our work.

Sequence alignment has also been applied on system call sequences of malware to extract evasion signatures and to cluster samples by Kirat and Vigna (2015), classification of malware families by Cho et al. (2016), and for malware detection, classification by D’Angelo et al. (2021), and visualization by Kim et al. (2019). Along with this line of work, there are existing works that proposed ensemble detection leveraging sequence alignment on API sequences (Ficco, 2021). All these approaches require access to API call sequences which require dynamic analysis and execution in a sandbox, virtual machine, or host. However, our proposed method is based on static analysis and only takes the raw bytes of a file which does not require any execution in a secondary machine.

Drew et al. (2016) utilized another approach developed by the bioinformatics and computational biology community and used gene or sequence classification methods for malware classification. The method is based on extracting short words i.e. k-mers from sequences and calculating the similarity between sequences based on the set of words present in them. Although the method is efficient, it does not fully utilize the information provided by stretches of conserved regions in malware and is not suitable for identifying critical code blocks in malware. In contrast, our method takes the whole raw-byte executable into consideration and so, can explain its decision by backtracking to the responsible code blocks.

Explainability of Models. Though there has been a large body of work on classifying and detecting malware, not much work has been done on explaining and interpreting the model decision. However, there are a few works that explored explainability, e.g., DREBIN (Arp et al., 2014), Melis et al. (2018, 2022), Backes and Nauman (2017), Kumar et al. (2018), Kinkead et al. (2021), MalDAE (Han et al., 2019), XMAL (Wu et al., 2021), etc. Unfortunately, all of these works were on Android malware using higher-level features like API calls, requested permissions, opcode sequences, etc., from manifest and disassembled code. In contrast, MA\textit{lign} takes raw-bytes from executables (Windows) and can explain the decision from that.

On the other hand, recent works in the machine learning community like LIME (Zhang et al., 2019), SHAP (Lundberg and Lee, 2017) propose strategies to interpret a decision of a blackbox model. However, these strategies assume that the local area of the classification boundary is linear. Later Guo et al. (2018a) proposed LEMNA showing that – when the decision boundary is non-linear, previous approaches would produce errors which makes them unsuitable for security applications. So, they solved this by using a mixture regression model enhanced by fused lasso. Though such models have played a great role in the explainability domain and inspired us, they have to be implemented and trained separately on a classifier model, which requires human engineering (like hyperparameter tuning) and adds computational overhead. However, our proposed method, MA\textit{lign} has a pipeline that is interpretable from top to bottom, i.e., easy to backtrack from decision output to problem space. Thus, it does not require any other model on top of it to explain its decision. Moreover, we generate signatures for each family that are easily interpretable by practitioners (see Section 6).

Adversarial Robustness. Continuous circumvention from malware authors and defense against such evasion by AV systems have turned into a never-ending cat-and-mouse game. For example, in the whitebox setting, Suciu et al. (2019), Kolos-
njaji et al. (2018), Kreuk et al. (2018b), Grosse et al. (2016), Al-Dujaili et al. (2018a), and in the blackbox setting, Demetrio et al. (2021), Ceschin et al. (2019), Fleshman et al. (2018b), Hu and Tan (2018), Rosenberg et al. (2018), Hu and Tan (2017), showed that adversarial attacks are effective against machine learning (ML) models in multiple ways. Also, Anderson et al. (2018), Song et al. (2020) proposed a reinforcement based approach to rewrite a binary for evading ML models.

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 most commonly used approaches to improve adversarial robustness (Bai et al., 2021; Grosse et al., 2017; Zhang et al., 2021; Al-Dujaili et al., 2018b). However, such approaches hamper the classification accuracy, add computational overhead, and rely on the availability of enough adversarial samples, which may not always be the case in the fast-changing malware world (Lucas et al., 2023). So, in this work, we focus on the latter one – robustness by design.

Robustness by design approaches build classifiers to eliminate a certain class of adversarial attacks. For example, Chen et al. (2020) proposed learning PDF malware detectors with verifiable robustness properties, Saha et al. (2023) proposed a detector adopting the de-randomized smoothing scheme with certified robustness, İncé Romeo et al. (2018) trained an XGBoost based malware detector with the monotonicity property, Fleshman et al. (2018a) trained a non-negative network to ensure that an adversary cannot decrease the classification score by adding extra content, etc. However, these approaches had to compromise the detection accuracy to some extent. For example, the non-negative MalConv model’s detection accuracy drops from 94.1% to 89.4%, while MalAlign can hold its performance as well as its robustness.

Moreover, recent works (Lucas et al., 2021, 2023) demonstrate the binary-diversification technique against new and old ML defenses. It is fundamental that – no ML model is fully robust against all attacks, and so it is better to not totally rely on ML models. So, we propose MalAlign that incorporates sequence alignment in front of ML models which makes it non-invertible and non-differentiable. We evaluate our model against a gradient-based attack and theoretically prove that it is robust-by-design.

3. Background

Sequence alignment is a widely studied problem in bioinformatics to find similarities 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. For example, an alignment of the sequences ATTGACCTG-A and AT---CGTGTA is shown below where the regions denoted in black, characterized by identical characters, are matched, whereas the red and blue regions denote substitutions i.e. point mutations, and insertions or deletions during the evolutionary process, respectively.

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 (Needleman and Wunsch, 1970) and the Smith-Waterman (Smith and Waterman, 1981) 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 (Thompson et al., 1994; Katoh et al., 2002; Edgar, 2004), that use heuristics to solve the multiple sequence alignment (MSA) problem (wiki).

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), the shuffling of the 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 (Delcher et al., 1999) can perform alignment of two sequences in the presence of rearrangements, whereas Mauve (Darling et al., 2004), Cactus (Paten et al., 2011), etc. can handle multiple sequences.

Recently, Armstrong et al. (2020) and Minkin and Medvedev (2020) have developed Progressive Cactus and SibeliaZ respectively, that can align hundreds to thousands of whole-genome sequences that may include rearrangements. The tools identify similar sub-sequences in the sequences from different species to create blocks of rearrangement-free sequences, and then perform a multiple sequence alignment of the sequences in each block.

Since adversaries can modify malware relatively easily by changing orders of blocks of codes without changing malware semantics, 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 family detection and classification (see Section 4). 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 a small number of instructions within the blocks, making it robust to adversarial attacks to some extent (see Section 7).

4. Methods

4.1. Overview

To classify known malware family and their variants, here we propose a malware family classification system based on
Figure 1: Overview of MALGEN. (1) The malware bytes files (executables) from malware 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 malware to form blocks. Highly similar sequences (colored sequences) can be in different orders 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-malware or malware 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.

4.2. 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 the existing whole-genome alignment tool can take the files as input. The conversion from the byte code to nucleotide sequence is done by converting each pair of bits to a nucleotide according to the following table:

| Byte   | Nucleotide |
|--------|------------|
| 00     | A          |
| 01     | C          |
| 10     | G          |
| 11     | T          |

Note that, similar conversions have been used by previous works, e.g. Naidu and Narayanan (2014, 2016); Drew et al. (2016), etc., and were successful in showing that such nucleotide sequences still can preserve necessary functionalities from the byte files.

In some malware datasets such as the Kaggle Microsoft Malware Classification Challenge (Big 2015) dataset (Microsoft, 2015), the provided bytes files contain “??” and long stretches of “00” in some cases which do not preserve any significant value or meaning. These are removed before the conversion to nucleotide sequences. Stretches of “00” longer than a threshold value of 32 were removed.
The multiple alignment format (MAF) is a format for describing multiple sequence alignment of locally co-linear blocks. The sequences share blocks of similar sequences shown in dashed lines of the same color. They may also contain sequence logos, where the height of the corresponding letter is at that particular coordinate of the sequence logo. The height of the individual letters in the 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 the highest frequency for each coordinate. Similarly, the consensus sequences for all blocks are generated and are stored in a file in FASTA (wiki, 2023a) 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. These files are later used to classify malware families.

In addition to the consensus sequence, we calculate conservation scores for the blocks. In bioinformatics, conservation score is used during the evaluation of sites in a multiple sequence alignment, in order to identify residues critical for structure or function. This is calculated per base, indicating how many species in a given multiple alignment match at each locus. We adopted this concept in the malware domain where the conservation score can indicate the significance or importance of a code segment in a malware family. The responsible code segments of malware will have high conservation scores or common pattern of that family. These files are later used to classify malware families.

In Figure 2, the heights of the bars corresponding to conservation scores indicate the degree of conservation at the corresponding positions. For each coordinate, we store the score for each of the four nucleotides which is given by

$$C_{B,i} = \arg \max_{N \in \{A,C,G,T\}} k_{B,i,N}$$

for 1 ≤ i ≤ l_B and B ∈ B, where k_{B,i,N} is the count of the character N (= A, C, G, T) at the i-th position in block B. The height of the individual letters in the sequence logo represents how common the corresponding letter is at that particular coordinate of the alignment.

During the block construction process; the order of the shared blocks may differ in different sequences and the blocks may not be fully shared across all sequences. This helps MALIGN to be robust against evasion attempts, such as reordering of instructions, injection of benign or random code at places, etc. In addition, the shared blocks may not be fully conserved, i.e., there can be mismatches of characters to some extent which means minor alteration or modification to the code will not prevent the detection of blocks. These properties of multiple whole-genome alignment improves the robustness of MALIGN against many evasion attacks.

SibeliaZ first identifies the shared linear blocks and then performs multiple sequence alignment of locally co-linear blocks. The block coordinates are output in the GFF format (wiki, 2023b) and the alignment is in the MAF format (Biopython). 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 malware. We generate such an MAF file for each malware family using the training samples and identify the blocks of codes that are highly conserved across the malware family.

4.4. Consensus Sequence and Score Generation

We process all sequences of the alignment blocks of the MAF file from the previous step and generate a new sequence for each block, which we will call the consensus sequence — motivated by Kececioglu and Myers (1995). 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 the highest occurrence for each coordinate. We put the most frequently occurring nucleotide at the corresponding index of the consensus sequence, i.e., the characters of the consensus sequences of the blocks are given by:

$$C_{B,i} = \arg \max_{N \in \{A,C,G,T\}} k_{B,i,N}$$

for 1 ≤ i ≤ l_B and B ∈ B, where k_{B,i,N} is the count of the character N (= A, C, G, T) at the i-th position in block B. The length of block B.

In Figure 2, consensus sequence generation of an alignment block (Block-1) is shown in detail. Below the alignment block, the corresponding sequence logo is shown. The height of the individual letters in the 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 the highest frequency for each coordinate. Similarly, the consensus sequences for all blocks are generated and are stored in a file in FASTA (wiki, 2023a) 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. These files are later used to classify malware families.

### Algorithm 1: MALIGN

**Input:** Training set, \( X = (X_1, X_2) \), where \( X_i \) : byte files from malware of a family, \( X_2 \) : byte files from non-malware or malware from other families, and Test set, \( Y \)

**Output:** Labels for \( Y \)

1. \( (S_i, S_f) \) ← Convert to nucleotide sequence files \( (X_i, X_f) \)
2. \( B \) ← Perform multiple sequence alignment and identify conserved blocks (\( S_i \))
   
   for all blocks \( B \) ∈ \( B \)
   
   \[ C_B \rightarrow \text{Get consensus sequence \( (B) \)} \]
   
   for \( i = 1 \rightarrow \text{length}(B) \)
   
   \[ \text{Calculate ConservationScore}(B, i, N) \text{ where } N = A, C, G, T \]

   for all sequences \( Z \in (S_i, S_f) \) do
   
   \[ F_Z := \text{Feature vector of } Z \]
   
   for all blocks \( B \) ∈ \( B \)
   
   \[ S \rightarrow \text{Get alignments (Z, C_B)} \]
   
   \[ \text{AlignmentCount} \leftarrow \text{Calculate alignment score (S, B)} \]
   
   \[ \text{AlignmentCount} \leftarrow \text{Get alignment count (S)} \]
   
   \[ F_Z \leftarrow F_Z \cup (\text{AlignmentScore, AlignmentCount}) \]

3. \( M \leftarrow \text{Learn classification model \( (F, X) \)} \text{ where } F: \text{ feature matrix} \)
4. \( T \rightarrow \text{Convert to nucleotide sequence files \( (Y) \)} \)
5. \( \text{Labels } \ell \) : labels
6. \( \text{for all sequences } T \in T \) do
7. \( \text{for all blocks } B \in B \) do
8. \( S \leftarrow \text{Get alignments \( (T, C_B) \)} \)
9. \[ \text{AlignmentCount} \leftarrow \text{Calculate alignment score (S, B)} \]
10. \[ \text{AlignmentCount} \leftarrow \text{Get alignment count (S)} \]
11. \[ F_T \leftarrow F_T \cup (\text{AlignmentScore, AlignmentCount}) \]
12. \( L \leftarrow \text{Predict } (M, F_T) \)
13. \( L \leftarrow L \cup L \)
14. return \( L \)
score at the $i^{th}$ index of the alignment block $B$ for nucleotide $N$ is given by
\[ \gamma_{B,i,N} = \frac{f_{B,i,N}}{n_B} \]
where $n_B$ is the number of sequences that constructed block $B$.

For example, in Figure 2, Block-1 has 3 sequences in total. Since at the 1$^{st}$ index, the block contains 3 $G$s,
\[ \gamma_{1,1,G} = \frac{3}{3} = 1.00 \]

Again at the 6$^{th}$ index, the block contains 2 $T$s and 1 $A$. Therefore,
\[ \gamma_{1,6,T} = \frac{2}{3} = 0.66 \] and \[ \gamma_{1,6,A} = \frac{1}{3} = 0.33 \]

4.5. Alignment with Consensus Sequences

Once the consensus sequences and the conservation scores are generated, 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 malware 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 scores, we calculate new scores called alignment scores for each block for all samples.

An example of alignment score calculation is shown in Figure 2. Malware X-1 and X-2 are positive and negative samples respectively. X-1 has three aligned sequences with the consensus sequence (shown in purple) whereas X-2 has only one. The sum of scores for all aligned sequences will be the score for the corresponding block of that sample. As an example, for the total score of sample X-1, we sum the scores of 3 aligned sequences, where 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 malwares from that family. In Figure 2, adding the score for all coordinates of the first aligned sequence of the sample X-1, 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 × 3). Finally, the total alignment score for the block was calculated by adding the scores of all 3 aligned sequences.

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

$$\alpha_{Z,B} = \sum_{s \in S} \left( \sum_{i=1}^{\text{length}(s)} \gamma_{B,i,s} \right) \times n_B$$

where, $S$ is the set of sequences from sample $Z$ that got aligned to $C_B$, $s_i$ is the $i$-th nucleotide of the sequence $s$, and $j$ is the index of $C_B$ where $s_j$ 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, i.e., alignment count $\beta_{Z,B} = |S|$. Both the number of occurrences and the total alignment score for the consensus sequence of each block for a malware family are used as input for the subsequent classification, resulting in $2m$ features if a malware family has $m$ aligned blocks.

4.6. Classification

Finally, we train machine learning models for each malware family to classify malware. The scores and the 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 trees, and a simple deep learning model. Since the results did not vary significantly across models (see Table 2 in Results), we use logistic regression as our primary model for its better interpretability. We are going to refer this model as MALIGN (Logistic Regression) in the rest of this paper. We also experimented with a simple deep-learning model to find if it gives any better results, which we are going to refer to as MALIGN (Deep Learning). However, such deep-learning models are not directly interpretable, and so, whenever we mention MALIGN in this paper, we indicate the MALIGN (Logistic Regression) version.

After the training phase, we get classifiers that can be used to classify new samples as shown in Figure 1. The scores of the training and the test samples are calculated in the same way. The scores for new samples are passed to the classifier for binary classification.

5. Evaluation

5.1. Datasets

5.1.1. The Kaggle Microsoft Malware Classification Challenge (Big 2015)

The Kaggle Microsoft Malware Classification Challenge (Big 2015) (Microsoft, 2015) aimed to organize a competition on malware family classification with a dataset consisting of 9 malware families (see Table 1).

This challenge simulates the data processed on over 160 million computers by Microsoft’s real-time anti-malware detection products inspecting over 700 million computers per month. 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 binary content of the malware.

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 an 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 an 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 was set aside as the test sets while the remaining 80% was used as the training sets. For the deep learning approaches that require hyper-parameter selection, the 80% was further split into training (60%) and validation sets (20%).

5.1.2. 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 new malware families only have a few samples - especially during their initial stages. Moreover, they might not always accurately represent real-world scenarios and in-the-wild practice. Therefore to assess the performance of our method on in-the-wild datasets with a small number of instances per family, we used the Microsoft Machine Learning Security Evasion Competition (2020) (MLSEC, 2020) dataset too. 4

In this competition, the defenders’ challenge was to create a solution model that can defend against evasive variants created by the attackers. The defenders were provided a ‘Defender Challenge’ dataset with 49 original malware and their evasive variants created by real malware attackers. Each malware contains a different number of evasive variants varying from 5 to 20. On average, malware has 12 variants in this dataset. We chose this dataset because – i) it mirrors different types of real-life attacks, and ii) it has a limited number of samples which can be a good representation of new emerging malware.

Similarly to the Kaggle Microsoft Malware dataset, 49 different datasets were created for each family, which were then split into training, validation, and test sets. During the split of 4The dataset may not remain public by Microsoft by the time of publication, but we can provide the dataset upon request.
this dataset, we always kept the original sample in the train set and the variants in the test set for each family, so that the test set can be considered as an evolution of the training set.

5.2. 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 labeled using ‘Training Labels’ data provided by Microsoft. Then we generated common alignment blocks using SibeliaZ and constructed the consensus sequences as discussed in Section 4. We generated the conservation scores for each consensus sequences using the frequency of nucleotides which were then used as features for the machine learning models.

We experimented with logistic regression, decision trees, and support vector machines (SVM). Table 2 shows the test accuracy for 80%-20% train-test split on the Kaggle Microsoft Malware Classification Challenge (Big 2015) Dataset. We observe that the models show similar performances in terms of accuracy. So, we selected MAAlign (Logistic Regression) for future experiments because of its simplicity and interpretability. For hyper-parameters details and results on the train-set, readers can refer to Appendix A and Appendix B.

5.3. Comparison with Existing Approaches

Next, we compare the performance of MAAlign with some of the state-of-the-art approaches that can take raw-byte, MalConv (Raff et al., 2018) (a deep-learning based approach using raw byte sequence), Feature-Fusion (Ahmadi et al., 2016) (a feature extraction, selection and fusion based approach using byte and assembly files), and M-CNN (Kalash et al., 2018) (a CNN based approach relying on conversion to images) 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 specific dataset. However, we compare with MalConv and M-CNN, as they have been successfully applied to many different datasets, and to our knowledge, the Feature-Fusion method has even better accuracy (multiclass logloss = 0.00128) than the winning team of the Kaggle competition. Besides simple ML models, we also implement a deep-learning based model on top of the alignment scores calculated by MAAlign, which we refer to as MAAlign (Deep Learning). The architectures of the MAAlign (Deep Learning), MalConv and M-CNN are shown in Figure A.9 (see appendix).

The test and training accuracy of MAAlign with logistic regression and deep learning along with those of Feature-Fusion, MalConv, and M-CNN are shown in Tables 3 and B.11 (appendix). Table 3 shows that MAAlign (Deep Learning) has the best accuracy on the test set than other approaches. MAAlign (Logistic Regression) and (Deep Learning) have 97.99% and 98.59% accuracy which is close to the Kaggle winning team’s performance.

We compared MAAlign with other models with respect to recall, specificity, precision, and F1 score. Table 4 represents the overall performance of models taking the weighted average for all types (detailed result for each family in Table B.12 in the appendix). It shows that MAAlign has a more balanced performance than others. Moreover, it has higher precision which indicates that it has a comparatively low false positive rate – an indicator of a good malware detection model in commercial AV systems.

5.4. Applicability with Limited Amount of Data and Features

Although deep learning based approaches have been widely applied for malware classification and detection, they require extensive amounts of data for training and tend to overfit in the absence of that. Table 3 shows that Feature-Fusion, MalConv, and M-CNN perform arguably well for most malware families. However, we observe that, for Type 5 (Simda), which has only 42 samples, the test accuracies of Feature-Fusion, MalConv, and M-CNN are respectively 76.47%, 52.94% and 62.5% whereas the typical methods like Feature-Fusion need other information or features, e.g., assembly file, address for byte sequence, section information from PE, etc. In addition, MAAlign can take any arbitrary length of input sequence whereas E2E deep learning based approach like MalConv, M-CNN suffers from limited input-length constraint.

We also evaluated the performances of the methods on the MLSEC dataset (Microsoft Machine Learning Security Evasion

| 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 1: Malware Families in the Kaggle Microsoft Malware Classification Challenge Dataset
Family Name | Test Accuracy | Logistic regression | Decision tree | SVM
---|---|---|---|---
Ramnit | 99.64 | 99.82 | 99.64 |
Kelihos_ver3 | 99.27 | 99.27 | 99.27 |
Vundo | 97.4 | 97.4 | 97.4 |
Simda | 84.62 | 84.62 | 84.62 |
Tracur | 97.3 | 94.6 | 96.3 |
Kelihos_ver1 | 96.7 | 98.9 | 98.9 |
Obfuscator.ACY | 95.9 | 92.7 | 96 |
Gatak | 96.2 | 96.2 | 96.2 |
Overall | 97.99 | 97.42 | 98.02 |

Table 2: Test Accuracy on the Kaggle Dataset for different Machine Learning Models

| Family Name | MALIGN (Logistic Regression) | MALIGN (Deep Learning) | Feature-Fusion | MalConv | M-CNN |
|---|---|---|---|---|---|
| Ramnit | 99.64 | 99.46 | 98.7 | 95.66 | 88.44 |
| Kelihos_ver3 | 99.27 | 99.82 | 99.18 | 100 | 99.72 |
| Vundo | 97.4 | 98.70 | 95.79 | 94.89 | 97.21 |
| Simda | 84.62 | 84.62 | 76.47 | 52.94 | 62.5 |
| Tracur | 97.3 | 98.2 | 98.34 | 93.91 | 94.55 |
| Kelihos_ver1 | 96.7 | 95.7 | 98.75 | 96.08 | 94.67 |
| Obfuscator.ACY | 95.9 | 96.39 | 98.98 | 94.42 | 91.74 |
| Gatak | 96.2 | 98.37 | 98.03 | 98.67 | 88.68 |
| Overall | 97.99 | **98.59** | 98.52 | 96.95 | 94.10 |

Table 3: Test Accuracy of all Models on the Kaggle Dataset

Figure 3: Radar Chart showing Accuracy of all Models along the perimeter on each of the individual 49 malware types in the MLSec Dataset

Competition (2020) (MLSEC, 2020). Since it contains a limited number of variants created in almost real-time, this can be used to identify how our method works on new variants of a malware 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. Table 5 shows the weighted result of all models for an 80%-20% train-test split on all 49 malware families. Additionally, we evaluated another set of results for 60%-20%-20% train-validation-test split just for deep learning models (see Figure B.13 in appendix). We observe that MALIGN outperforms the other models on the MLSEC dataset regardless of the splitting.

Furthermore, Figures 3a and 3b provide radar charts that visually illustrate the training and test accuracy for all 49 types individually. From these figures, we observe that on the train set, all approaches have consistent performance, but on the test set M-CNN and MalConv are relatively inconsistent\(^ 5 \). For example, types 29 and 43 have 10 and 5 available variants respectively, and M-CNN’s test accuracy is 0 on both, indicating a
The more circular the perimeter line is, the more consistent the performance is.

(iii) Processing the multiple alignment file (MAF) to find out the sequences and their indices that constructed the blocks.

(iv) Locating the code fragments in binary and assembly files (using a disassembler) corresponding to the sequences found in step (iii).

We evaluated the interpretability of our MALIGN by following the above-mentioned backtracking process on multiple families from the Microsoft Kaggle Dataset. We discuss some of the interesting findings below (more in Appendix C).

6.1. Kelihos_ver1 Backdoor

The Kelihos backdoor allows the affected system to be part of the Kelihos botnet. It is then typically used to send spam email messages, steal information, etc. (fsecure, 2023a).

In our analysis of this specific backdoor malware, interestingly, one of our consensus sequences got mapped to 119 samples. While looking into the assembly code of this signature, we found that all of them are calling GetUserDefaultLangID function (see Figure 4). Usually, this function is called by malware to track geological position and trigger a regional attack (malware function list).

Furthermore, we found more code blocks having suspicious Windows function calls from IDENTpack.dll which gets abused by malware authors very often. From Figure C.13 in Appendix, we want to emphasize that – the order of these function calls is different in malwares, e.g., IsNTAdmin function is declared before RebootCheckOnInstall in C.13a and C.13b, but not in C.13c. However, MALIGN can successfully detect them due to its flexibility in alignment. The common pattern in these malwares from Figure C.13 is that – they are checking for admin privileges by IsNTAdmin, calling suspicious function RebootCheckOnInstall (also found in previous reports from Falcon Sandbox of CrowdStrike (2017, 2015)), and deleting files sometimes using DelNodeRunDLL32.

6.2. Tracur Trojan-Downloader

Tracur Downloader: W32/Tracur.J is a malware that is known to identify a malicious DLL file that installs a plug-in for Internet Explorer and/or Mozilla Firefox web browsers and then redirects searches to an unsolicited website (fsecure, 2023c).

Consequently, malware authors write malware from Tracur family in such a way that it can get connected to the internet and download or upload the desired file (close to banking trojan (IAActive, 2012)). Our consensus sequence successfully detected code segments for such activities. Figure 5 is an example of such malware which is calling functions like InternetReadFile, InternetCloseHandle, etc. The clearly reveals that MALIGN is capable of capturing such sequential function calling that is suspicious.

Table 6: Test Accuracy Drop for dropping the train-set size of Tracur family to half

| Model          | Recall | Specificity | Precision | F1 Score |
|----------------|--------|-------------|-----------|----------|
| MALIGN (Logistic Regression) | 96.66  | 98.60       | 98.67     | 97.61    |
| MALIGN (Deep Learning)         | 99.00  | 98.18       | 98.19     | 98.59    |
| MalConv                  | 97.37  | 96.48       | 97.36     | 97.3     |
| M-CNN                    | 93.44  | 94.85       | 94.59     | 93.94    |

Table 4: Performance of Models on the Kaggle Dataset for other Evaluation Metrics

| Model                          | Train Accuracy | Test Accuracy |
|--------------------------------|----------------|---------------|
| MALIGN (Logistic Regression)   | 98.18          | 80.42         |
| MALIGN (Deep Learning)         | 97.72          | 80.00         |
| Feature-Fusion                 | 98.02          | 60.94         |
| MalConv                        | 98.24          | 79.22         |
| M-CNN                          | 96.99          | 71.09         |

Table 5: Performance of Models on the MLSEC dataset for train-test split

| Model                          | Accuracy Drop | Accuracy Change (%) ↓  |
|--------------------------------|---------------|------------------------|
| MALIGN (Logistic Regression)   | 97.3% – 92.56%| 4.87%                  |
| MALIGN (Deep Learning)         | 98.2% – 92.67%| 5.53%                  |
| MalConv                        | 93.91% – 85.67%| 8.77%                  |
| M-CNN                          | 94.55% – 83.22%| 11.98%                 |

6. Explainability

An important feature of MALIGN which makes it different from other classifiers is its interpretability, and its capability of deriving suspicious code blocks in malware families through a simple backtracking process. Upon analysis, the ‘maliciousness’ of these suspicious code blocks can be determined. The backtracking process is as follows:

(i) Finding the blocks that are assigned high weights by the logistic regression model.

(ii) Selecting the blocks (found from step (i)) that are highly conserved.

For further analysis, we experimented with dropping the train-set size of the Tracur family of the Kaggle dataset to half and evaluated all models. We observed that MALIGN has the least accuracy drop compared to others (see Table 6). Again, this proves the advantage of our method when a malware family does not have a considerably large sample size, which is very common in the case of emerging malware families. This was one of the motivations for adopting sequence alignment since its capability of explicit identification of critical code blocks still sustains the scarcity of data.
6.3. Obfuscator.ACY

Malware with vastly different purposes often “obfuscate” i.e. hide its purpose so that security software does not detect it (Microsoft, 2017). While different obfuscator tools can obfuscate or protect the malicious code in different ways, some packers like Themida keep one function – TlsSetValue (Borges, 2020) so that it can unpack it later. This function is used to set the value of a pointer allocated for a TLS variable and has been heavily used by malware authors for many different attacks (Hellal and Romdhane, 2016; Poudyal and Dasgupta, 2021), e.g., stack-smashing attack (Nebenzahl et al., 2006).

In our analysis, we found consensus sequences getting matched with code blocks that contain TlsSetValue. Figure 6 shows the code block from one of such consensus sequences which was present in 39 samples from Obfuscator.ACY family.

6.4. Vundo Trojan

Vundo is malware that downloads and displays pop-up advertisements of rogue software. It can download and install other malware and there are variants that collect information such as IP address, MAC address, Windows and browser versions, etc. and send it to attackers (FireEye, 2008-2014).

While analyzing random malware executables from Vundo family, surprisingly, we found some consensus sequences that were mapped to binary sequences from different sections. To evaluate whether they are just some spurious features or not, we examined their assembly code and interestingly found that there is obfuscated code (from DllEntryPoint function) put into the ‘data’ section instead of ‘text’ section. It might be the case that – one malware is loading another in its data section. However, such patterns got identified by our MAAlign since their binary representations are similar (see Figures C.11 and C.10 in Appendix).

6.5. Simda Backdoor

Backdoor:W32/Simda is a large malware family that allows attackers to remotely control machines they are installed in to steal personal or system data, take screenshots, download additional files onto the system, etc. (Secure, 2023b)

Among the MAAlign generated consensus sequences for Simda family, we randomly selected some of them and found matches to some suspicious function calls. For example, two samples had exactly the same code where they were declaring suspicious functions like – LookUpPrivilegeValueW, CreateProcessAsUserW, SetTokenInformation, etc.
Then, they are using ADVAPI32.DLL through which they can access core windows components such as the service manager and registry (see Figure 7). It might be the case that the backdoor malware is checking for the privilege to perform privileged operations, or checking if the privilege escalation exploit should be executed later.

7. Robustness

A major issue with static malware detection approaches is they can be evaded by malware authors in multiple ways, such as adding content without altering malware functionalities. There are several ways to find or generate this content, such as gradient-based patch (Kreuk et al., 2018b; Kolosnjaji et al., 2018; Suciu et al., 2019). Since MALign relies on finding conserved blocks critical to the malware families for classification through sequence alignment and score calculation, and an interpretable logistic regression model, it should in principle to be inherently robust to such attacks.

7.1. Theoretical Robustness

To formally show the difficulty of such attacks, as in Pierazzi et al. (2020); Guo et al. (2018b), we mathematically prove that inverting our whole genome alignment process is an NP-Hard problem.

7.1.1. Proof

We consider problem space $Z$ that contains malware files $z$ each having a ground truth label $y \in Y$. The feature mapping function $\varphi$, i.e., the alignment tool, score generation algorithm, etc, maps the input $z$ to a feature vector $x$; $\varphi(z) = x$. The machine learning classifier predicts a label $y$ for $x$; $g(x) = y$.

| Symbol | Description |
|--------|-------------|
| $Z$ | Problem space (i.e., input space) |
| $X$ | Feature Space |
| $Y$ | Label Space |
| $\varphi$ | Feature mapping function, $\varphi : Z \rightarrow X$ |
| $g$ | Classifier, $g : X \rightarrow Y$ |

Table 7: Symbols with Description

Figure 8: Flow of our method and the backward flow for attacks

Let’s consider that all training and classifying algorithm is disclosed to the attacker. The attacker’s goal would be to perturb the input $z$ to $z'$ in such a way that our model misclassifies it as label $t$, where $t \neq y$. Since the machine learning classifier is a differentiable model, it is possible to find perturbed feature vector $x'$ through the gradient attack, such that $g^{-1}(t) = x'$. The next job of the attacker would be to find an inverse feature mapping function, such that $\varphi^{-1}(x') = z'$. Now to block this backward flow, the feature mapping function $\varphi$ needs to be non-invertible and non-differentiable.

**Hardness of Invertibility**

Let, $B = \{B_1, B_2, B_3, ..., B_m\}$ denote the set of $m$ alignment blocks of a malware family where each element $B_i$ is a string
of nucleotides. The $i^{th}$ alignment block $B_i$ is represented by a conservation score matrix $A_i \in [0,1]^{5\times n}$ which stores the conservation scores for each nucleotide, where $n$ is the length of the block. So, each row of the score matrix $A$ will store the scores for A, C, G, T and gap, respectively.

Let $S_i = \{S_i^1, S_i^2, S_i^3, ..., S_i^e\}$ denote the set of sequences from perturbed malware file $\varepsilon'$ that got aligned with alignment block $B_i$ for $e$ times. Let’s assume, without loss of generality, that only one sequence from input $\varepsilon'$ is aligned with alignment block $B_i$ and so only one alignment score $a_{\alpha,B}$ is generated.

So, to prove the hardness of invertibility, it suffices to prove that reconstructing the alignment sequence $S_i$ from the conservation score matrix $A_i$ and the alignment score $a_{\alpha,B}$ is hard i.e. the following problem is NP-hard.

**Alignment Inversion Problem:** Given a conservation score matrix $A \in [0,1]^{5\times n}$ and a target alignment score $\alpha$, the alignment inversion problem is to find $y_{i,j} \in [0,1]^{5\times n}$ such that

$$\sum_{j=1}^{5} \sum_{i=1}^{n} y_{i,j} A_{i,j} = \alpha$$

Here the conservation scores for each column in matrix $A$ is normalized, i.e., $\sum_{i=1}^{n} A_{i,j} = 1$ for all $j$.

In order to show that the Alignment Inversion Problem is NP-hard, we reduce a well-known NP-complete problem, the subset sum problem, to our problem.

**Subset Sum Problem:** The subset sum problem (SSP) is, given a set of integers and a target sum $T$, to decide whether any subset of the integers sum to $T$. The problem is NP-complete even when the numbers are restricted to positive integers. So, SSP refers to finding variables $(x_1, x_2, x_3, ..., x_n) \in \{0, 1\}^n$, given positive integers $(a_1, a_2, a_3, ..., a_n) \in \mathbb{N}$ and $T$, such that

$$\sum_{i=1}^{n} x_i a_i = T$$

Now, given an SSP instance, we generate an instance of the alignment inversion problem by setting

$$A_{1,j} = \frac{a_j}{N}, \quad A_{S,j} = 1 - \frac{a_j}{N}, \quad A_{i,j} = 0 \text{ for } i \in \{2, 3, 4\}$$

and $\alpha = \frac{T}{N}$

Here $N$ is a positive integer chosen to make sure that i) $0 \leq A_{i,j} \leq 1$ for all $i,j$, and ii) $A_{S,j} = 1 - \frac{a_j}{N} \geq \alpha$ for all $j$. This can be done by setting

$$N > a_{\text{max}} + T$$

where $a_{\text{max}} = \max(a_1, a_2, \ldots, a_n)$.

Since $A_{S,j} \geq \alpha$ for all $j$, none of the scores for gaps can be included in the target alignment score $\alpha$, and $A_{i,j}$ for $i = 2, 3, 4$ cannot contribute towards $\alpha$ as they are set to 0. Therefore, given the solution to the instance of one problem, we can obtain the solution to the other by setting $x_j = A_{1,j}$ for all $j$, because of the following:

$$\sum_{j=1}^{n} x_j a_j = T \iff \frac{1}{N} \sum_{j=1}^{n} x_j a_j = \frac{T}{N} \iff \sum_{j=1}^{n} y_{i,j} A_{1,j} = \alpha$$

Since SSP is a NP-hard problem and polynomial-time reducible to our problem, our problem is a NP-hard problem too. In our original problem, we may have to consider multiple nucleotides in most cases which is a harder version of this instance. So, it is trivial to prove that, our original problem is a NP-hard problem too.

While we acknowledge that – theoretical robustness does not ensure practical robustness, this property of MALIGN shows that the model is robust to certain kinds of attack.

7.2. Empirical Robustness

We also investigated the empirical robustness of our method compared to the other conventional malware detection techniques. We generated adversarial malwares using gradient-based patch attack of Kolosnjaji et al. (2018). It creates adversarial samples just by modifying (or padding) approximately 1.25% of the total size of malware which can successfully evade the MalConv model with a high percentage. The evasion rate increases with the percentage of modification on malware samples. We experimented with their implementation (github link) on some of the types in the MLSEC dataset.

| Type  | MalConv Evaded/Total | MalConv Evasion Rate | MALIGN Evaded/Total | MALIGN Evasion Rate |
|-------|----------------------|----------------------|---------------------|---------------------|
|       | Train set | Test set |                  |                     |                     |
| 1     | 5/15     | 2/4       | 36.84%             | 0.00%               |
| 11    | 1/9      | 1/2       | 18.18%             | 0.00%               |
| 12    | 5/13     | 2/4       | 41.18%             | 0.00%               |
| 28    | 5/8      | 2/2       | 70%                | 0.00%               |
| 45    | 4/5      | 2/2       | 85.71%             | 0.00%               |

Table 8: Evasion Rate on MalConv and MALIGN using Gradient-based patch attack for some types in the MLSEC Dataset.

We then applied these evasive samples on MALIGN and all of them were successfully detected with almost 100% prediction confidence (see Table 8). The probable reason for such a difference in the evasion rate is – gradient-based patch attack (Kolosnjaji et al., 2018) appends random bytes at the end of a malware file to evade the classifier, and since the alignment mechanism of MALIGN is robust to insertions, its decision remains unaltered. Moreover, due to the non-invertible property of MALIGN (proved in subsection 7.1.1), the attacker cannot generate any such random content with gradient approach to evade it.

It is worth mentioning that – no static analyzer can provide robustness to all type of attacks (similarly, no attack can evade all defense models). So, as security researchers, our goal is to make the bar higher for attackers. Since MALIGN finds conserved code blocks in the presence of substitutions, insertions, and deletions of instructions, modifying those code segments sufficiently while preserving the malware semantics will be more challenging than other used models. However, we have discussed more about evasion techniques for MALIGN in the Appendix D.1.
8. Running Time

For our experiments, we used a CPU of 40GB memory and GeForce RTX 3090. MALIGN only requires the CPU, whereas we had to use the GPU for the deep learning based methods.

8.1. Mean-Time-To-Detect (MTTD)

| Method     | MTTD   | Training Time |
|------------|--------|---------------|
| MalConv    | 0.0112584 s | 18hr 4min    |
| M-CNN      | 1.2461524 s  | 18hr 13min   |
| M-CNN      | 0.8706633 s  | 14hr 26min   |

Table 9: MTTD and training time for different methods on the MLSEC dataset

We evaluated the MTTD for all methods by considering the time they take in their inference stage. We ran all methods on the test set of the MLSEC dataset and measured the total time for the classifier to make the decisions. Then we divided this time by the total number of samples in the test set to find the MTTD.

Table 9 shows the MTTD for each method. Since MalConv and M-CNN are deep learning models unlike MALIGN (Logistic Regression), they take comparatively more time to classify a sample. Notably, we did not include the feature extraction time here, for which MALIGN takes 15.9498 seconds on average. In this feature extraction step, the graph construction step by TwoPaCo (Minkin et al., 2017) takes up to 12 seconds which is 75.24% of the whole process. Since we do not have any control over the implementation of TwoPaCo, we cannot speed this process up anymore. However, once the signatures are found in MALIGN, they can be trimmed if needed (based on how conserved they are) and then the alignment of the sequence of the new instance and the signature will be faster.

8.2. Training Time

We also evaluated the total training time for each method on the MLSEC dataset. From Table 9, we can see that the training time is not vastly different from other methods. Here, we have included the time for alignment and feature extraction too.

Ahmadi et al. (2016)’s Feature-Fusion method has not been included in Table 9 because for the MLSEC dataset, partial features (only features from byte code) were considered. But we can get an estimation of its running time on the Kaggle Microsoft dataset from the original paper (Ahmadi et al., 2016). For example, it takes almost 17 hours and 15 minutes to extract only the ‘REG’ feature from all samples, and it has to extract 14 features in total.

9. Discussion

In this work, we proposed MALIGN for malware family classification from raw executables – distinct from the prior works in this domain for its interpretability and better robustness. To show that MALIGN can actually help the AV security practitioners in real-life, we analyzed a vast number of potentially suspicious code blocks found from our consensus sequences, and upon analysis, we evaluated their maliciousness depending on the context. Some of them have been shared in Section 6 and Appendix C (after getting double-checked by a security analyst from a reputed AV company). Besides, we also generated some statistics on our consensus sequence analysis (see Appendix E).

9.1. Limitations

The approach used in MALIGN has some limitations too. Firstly, though with recent and efficient multiple sequence alignment tools like SibeliaZ (Minkin and Medvedev, 2020) we can align malware files very fast, the time can grow with the number of samples in a family. Secondly, any static analyzer is fundamentally vulnerable since it cannot execute and analyze the behavior of the malware. Though MALIGN provides robustness against some specific attacks, such as gradient-based patch attacks, it might not be the case for all attacks (discussed in Appendix D.1). Thirdly, like most other static analyzers, it cannot perform on packed and encrypted malware without unpacking or decrypting it.

10. Conclusion

In this paper, we presented a malware family classification tool MALIGN adopting a recently developed multiple whole-genome alignment tool SibeliaZ. Sequence alignment based approaches have been used for malware analysis in the past, but the use of a whole-genome alignment tool makes MALIGN scalable to long malware sequences and protects against trivial adversarial attacks. The method is also interpretable and can be used to derive insights on malware such as the identification of critical code blocks and code obfuscation. To our knowledge, we are the first to explore the interpretability on a raw-byte based static malware classifier.

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 state-of-the-art static classifier methods such as the MalConv, Feature-Fusion, and M-CNN method. However, outperforming other models is not the main goal of this paper, rather we wanted to explore static analysis from a different perspective to balance accuracy, interpretability, and robustness.

We discovered malicious code-blocks from our consensus sequences that can provide insights to malware analysts. We welcome security practitioners to utilize MALIGN for not only classifying but also analyzing the malware. MALIGN can be used in the future to measure the variance among malware samples which might provide insights about common practices by malware authors. We believe it might open up a new paradigm in the malware analysis domain.

Code Availability

Our code can be found here - https://github.com/ShoumikSaha/Malign
CRediT authorship contribution statement

- **Shounik Saha**:Conceptualization, Data curation, Formal Analysis, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing
- **Sadia Afroz**: Resources, Supervision, Writing – review & editing
- **Atif Hasan Rahman**: Conceptualization, Formal Analysis, Investigation, Resources, Supervision, Writing – review & editing

Acknowledgements

We would like to express our sincere gratitude to Erin Avlazagaj for his review and feedback. He helped us to improve the paper by analyzing some signatures.

References

Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G., 2016. Novel feature extraction, selection and fusion for effective malware family classification, in: Proceedings of the sixth ACM conference on data and application security and privacy, pp. 183–194.

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

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

Anderson, B., Quist, D., Neil, J., Stormlic, C., Lane, T., 2011. Graph-based malware detection using dynamic analysis. Journal in computer Virology 7, 247–258.

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

Armstrong, J., Hickey, G., Diekhans, M., Fudde, I.T., Novak, A.M., Deran, A., Fang, Q., Xie, D., Feng, S., Stiller, J., et al., 2020. Progressive Cactus is a multiple-genome aligner for the thousand-genome era. Nature 587, 246–251.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and explainable detection of android malware in your pocket., in: Ndos, pp. 23–26.

AV-Test, 2023. Av-test malware statistics. URL: https://www.av-test.org/en/statistics/malware/.

Avlazagaj, E., Zhu, Z., Bilge, L., Balzarotti, D., Dumitras, T., 2021. When malware changed its mind: An empirical study of program behaviors in the real world, in: 30th USENIX Security Symposium (USENIX Security 21), pp. 3487–3504.

Backes, M., Nauman, M., 2017. Luna: quantifying and leveraging uncertainty in android malware analysis through bayesian machine learning, in: 2017 IEEE European symposium on security and privacy (eurosp), IEEE, pp. 204–217.

Bai, T., Luo, J., Zhao, J., Wen, B., Wang, Q., 2021. Recent advances in adversarial training for adversarial robustness. arXiv preprint arXiv:2102.01356

Biopython. Multiple alignment format. https://biopython.org/wiki/Multiple_Alignment_Format.

Borges, A., 2020. Handling advanced threats. URL: http://tinyurl.com/bhdz5x.

Ceschin, F., Botacin, M., Gomes, H.M., Oliveira, L.S., Grégio, A., 2019. Shallow security: On the creation of adversarial variants to evade machine learning-based malware detectors, in: Proceedings of the 3rd Reversing and Offensive-oriented Trends Symposium, pp. 1–9.

Chen, Y., Narayanan, A., Pang, S., Yao, R., 2012. Multiple sequence alignment and artificial neural networks for malicious software detection, in: 2012 8th International Conference on Natural Computation, IEEE, pp. 261–265.

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

Cho, I.K., Kim, T.G., Shim, Y.J., Ryu, M., Im, E.G., 2016. Malware analysis and classification using sequence alignments. Intelligent Automation & Soft Computing 22, 371–377.

CrowdStrike, 2015. Falcon sandbox report on 927.exe. URL: https://tinyurl.com/3h8px5b9.

CrowdStrike, 2017. Falcon sandbox report on freepdf412.exe. URL: https://tinyurl.com/5xmx2tzk.

CrowdStrike, 2018-2019. Falcon sandbox report on backdoor. https://tinyurl.com/3h8px5b9.

Darling, A.C., Mau, B., Blattnier, F.R., Perna, N.T., 2004. Mauve: multiple alignment of conserved genomic sequence with rearrangements. Genome Research 14, 1394–1403.

Delcher, A.L., Kasif, S., Fleischmann, R.D., Peterson, J., White, O., Salzberg, S.L., 1999. Alignment of whole genomes. Nucleic Acids Research 27, 2369–2376.

Demetro, L., Biggio, B., Lagorso, G., Roli, F., Armando, A., 2021. Functionality-preserving black-box optimization of adversarial windows malware. IEEE Transactions on Information Forensics and Security 16, 3469–3478.

Drew, J., Moore, T., Hahsler, M., 2016. Polyomorphic malware detection using sequence classification methods, in: 2016 IEEE Security and Privacy Workshops (SPW), IEEE, pp. 81–87.

D’Angelo, G., Cicco, M., Palmieri, F., 2021. Association rule-based malware classification using common subsequences of api calls. Applied Soft Computing 105, 107234. URL: https://doi.org/10.1016/j.asoc.2021.107234.

Edgar, R.C., 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research 32, 1792–1797.

Ficco, M., 2021. Malware analysis by combining multiple detectors and observation windows. IEEE Transactions on Computers 71, 1276–1290.

FireEye, 2008-2014. Fireeye vundo trojan. URL: https://mi1.fireeye.com/edp.php?name=Trojan.Vundo.

Fleshman, W., Ruff, E., Sylvester, J., Forsyth, S., McLean, M., 2018a. Non-negative networks against adversarial attacks. arXiv preprint arXiv:1806.06108.

Fleshman, W., Ruff, E., Zak, R., McLean, M., Nicholas, C., 2018b. Static malware detection & subterfuge: Quantifying the robustness of machine learning and current anti-virus., in: 2018 13th International Conference on Malicious and Unwanted Software (MALWARE), IEEE, pp. 1–10.

f-secure, 2023a. Backdoor/keilhos by f-secure. URL: https://www.f-secure.com/v-descs/keilhos.shtml.

f-secure, 2023b. Backdoor/w32/simda by f-secure. URL: https://www.f-secure.com/v-descs/keilhos.shtml.

f-secure, 2023c. Trojan-downloader/w32/tracur.j by f-secure. URL: https://www.f-secure.com/v-descs/trojan-downloader/w32_tracur.j.shtml.

Grosse, K., Papenrot, N., Manoharan, P., Backes, M., McDaniel, P., 2016. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435.

Grosse, K., Papenrot, N., Manoharan, P., Backes, M., McDaniel, P., 2017. Adversarial examples for malware detection, in: European symposium on research in computer security, Springer, pp. 62–79.

Guo, W., Ma, D., Xu, J., Qu, S., Wang, G., Xing, X., 2018a. Lemna: Explaining deep learning based security applications, in: proceedings of the 2018 ACM SIGSAC conference on computer and communications security, pp. 364–379.

Guo, W., Wang, Q., Zhang, K., Oortviba, A.G., Huang, S., Liu, X., Giles, C.L., Lin, J., Xing, X., 2018b. Defending against adversarial samples without security through obscurity, in: 2018 IEEE International Conference on Data Mining (ICDM), IEEE, pp. 137–146.

Han, W., Xue, J., Wang, Y., Huang, L., Kong, Z., Mao, L., 2019. Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics. computers & security 83, 208–233.
Kumar, R., Xiaosong, Z., Khan, R.U., Kumar, J., Ahad, I., 2018. E
Kirat, D., Vigna, G., 2015. Malgene: Automatic extraction of malware analysis
Kinkead, M., Millar, S., McLaughlin, N., O’Kane, P., 2021. Towards explain-
Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D., Wang, Y., Iqbal, F.,
https://www.jesobox.com/analysis/821041/0/html#.
Kalash, M., Rochan, M., Mohammed, N., Bruce, N.D., Wang, Y., Iqbal, F., 2018. Malware classification with deep convolutional neural networks, in:
2019 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS), IEEE, pp. 1–5.
Katoh, K., Misawa, K., Kuma, K.i., Miyata, T., 2002. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nu-
Kecicioglu, J.D., Myers, E.W., 1995. Combinatorial algorithms for dna se-
sequence assembly. Algorithmica 13, 7–51.
Kim, H., Kim, J., Kim, Y, Kim, I., Kim, K.J., Kim, H., 2019. Improvement of malware detection and classification using api call sequence alignment and visualization. Cluster Computing 22, 921–929.
Kinked, M., Millar, S., McLaughlin, N., O’Kane, P., 2021. Towards explain-
able cnns for android malware detection. Procedia Computer Science 184, 959–965.
Kirat, D., Vigna, G., 2015. Malgene: Automatic extraction of malware analysis evasion signature, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 769–780.
Kolonosajji, B., Demontis, A., Biggio, B., Maiorca, D., Gacicnto, G., Eckert, C., Roli, F., 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables, in: 2018 26th European signal processing conference (EUSIPCO), IEEE, pp. 533–537.
Kreuf, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B., Keshet, J., 2018a. Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv preprint arXiv:1802.04528.
Kreuf, F., Barak, A., Aviv-Reuven, S., Baruch, M., Pinkas, B., Keshet, J., 2018b. Deceiving end-to-end deep learning malware detectors using adversarial examples. arXiv preprint arXiv:1802.04528.
Kumar, R., Xiaosong, Z., Khan, R.U., Kumar, J., Ahad, I., 2018. Effective and explainable detection of android malware based on machine learning algo-
rithms, in: Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, pp. 35–40.
github link, .https://github.com/yuxiaorun/MalConv-Adversarial.
MalConv, function list, . Windows functions in malware analy-
sis. URL: https://gist.github.com/404NetworkError/a81591849f5b6b5e095f517efc189cd#\waintadmin-top.
Liu, Y., Tanthithamthavorn, C., Li, L, Liu, Y., 2022. Explainable ai for android malware detection: Towards understanding why the models perform so well?, in: 2022 IEEE 33rd International Symposium on Software Relia-
bility Engineering (ISSRE), IEEE, pp. 169–180.
Lu, R., 2019. Malware detection with lstm using opcode language. arXiv preprint arXiv:1906.04593.
Lucas, K., Pai, S., Lin, W., Bauer, L., Reiter, M.K., Sharif, M., 2023. Adversar-
ial training for raw-binary malware classifiers.
Lucas, K., Sharif, M., Bauer, L., Reiter, M.K., Shintre, S., 2021. Malware makeover: Breaking ml-based static analysis by modifying executable bytes, in: Proceedings of the 2021 ACM Asia Conference on Computer and Commu-
nications Security, pp. 744–758.
Lundberg, S.M., Lee, S.I., 2017. A unified approach to interpreting model predictions. Advances in neural information processing systems 30.
Marcus, G., 2018. Deep learning: A critical appraisal. arXiv preprint arXiv:1801.00631.
Martignoni, L., Stinson, E., Fedriksen, M., Jha, S., Mitchell, J.C., 2008. A lay-
ered architecture for detecting malicious behaviors, in: International Work-
shop on Recent Advances in Intrusion Detection, Springer, pp. 78–97.
Melis, M., Maiorca, D., Biggio, B., Giacinto, G., Roli, F., 2018. Explaining black-box android malware detection, in: 2018 26th european signal pro-
cessing conference (EUSIPCO), IEEE, pp. 524–528.
Melis, M., Scalas, M., Demontis, A., Maiorca, D., Biggio, B., Giacinto, G., Roli, F., 2022. Do gradient-based explanations tell anything about adversar-
ial robustness to android malware? International journal of machine learning and cybernetics, 1–16.
Microsoft, , Microsoft PE format. URL: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.
Microsoft, 2015. Microsoft malware classification challenge (big 2015). https://www.hnaggle.com/c/malwareclassification/.
Microsoft, 2017. Virtualwin32/obfuscator.acy by microsoft. URL: https://www.microsoft.com/en-us/teams/attacks/malware-encyclopedia-description?Name=VirTool:Win32/ Obfuscator ACPI.
Minkin, I., Medvedev, P., 2020. Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ. Nature communications 11, 1–11.
Minkin, I., Pham, S., Medvedev, P., 2017. Twopaco: an efficient algorithm to build the compacted de bruijn graph from many complete genomes. Bioin-
fomatics 33, 4024–4032.
MLSEC, 2020. Microsoft machine learning security evasion competition. http://www.micro-
cosm.com/en-us/security/blog/2021/07/29/ attack-ai-systems-in-machine-learning-evasion-competition/.
Naidu, V., Narayanan, A., 2014. Further experiments in biocomputational structural analysis of malware, in: 2014 10th International Conference on Natural Computation (ICNC), IEEE, pp. 605–610.
Naidu, V., Narayanan, A., 2016. Needlemann-wunsch and smith-waterman al-
gorithms for identifying viral polyomorphic 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/PSCom/DataCom/CyberSciTech), IEEE, pp. 326–333.
Narayanan, A., Chen, Y., Pang, S., Tao, B., 2012. The effects of different rep-
resentations on malware motif identification, in: 2012 Eighth International Conference on Computational Intelligence and Security, IEEE, pp. 86–90.
Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S., 2011. Malware im-
age: visualization and automatic classification, in: Proceedings of the 8th international symposium on visualization for cyber security, pp. 1–7.
Nebenzahl, D., Sagiv, M., Wool, A., 2006. Install-time vaccination of windows executables to defend against stack smashing attacks. IEEE Transactions on Dependable and Secure Computing 3, 78–90.
Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48, 443–453.
O’Meir, O., Nissim, N., Elovici, Y., Rokach, L., 2019. Dynamic malware analysis in the modern era—a state of the art survey. ACM Computing Surveys (CSUR) 52, 1–48.
Pascunu, R., Stokes, J.W., Sanossian, H., Marinescu, M., Thomas, A., 2015. Malware classification with recurrent networks, in: 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, pp. 1916–1920.
Paten, B., Earl, D., Nguyen, N., Diekhans, M., Zerbino, D., Haussler, D., 2011. Cactus: Algorithms for genome multiple sequence alignment. Genome Res-
earch 21, 1512–1528.
Petti, S., Bhattacharya, N., Rao, R., Dauparas, J., Thomas, N., Zhou, J., Rush, A.M., Koo, P.K., Ovchinnikov, S., 2021. End-to-end learning of multiple sequence alignments with differentiable smith-waterman. BioRxiv, Pierazzi, F., Pendlebury, F., Cortellazzi, L., Cavallaro, L., 2020. Intriguing effects of adversarial ml attacks in the problem space, in: 2020 IEEE Symposium on Security and Privacy (SP), pp. 1332–1349. doi: 10.1109/ SP40000.2020.00073.
Poudyal, D., Dasgupta, D., 2021. Analysis of crypto-ransomware using ml-
based multi-level profiling. IEEE Access 9, 122532–122547.
Raff, E., Barker, J., Sylvester, J., Brandon, R., Catanzaro, B., Nicholas, C.K., 2018. Malware detection by eating a whole exe, in: Workshops at the Thirty-
Second AAAI Conference on Artificial Intelligence, pp. 268–276.
Rosenberg, I., Shabtai, A., Rokach, L., Elovici, Y., 2018. Generic black-box end-to-end attack against state of the art api call based malware classifiers,
Appendix A. Models

Appendix A.1. Hyperparameters for Machine Learning Models

We experimented with the hyper-parameters for our machine learning models. The finding for each model are given below –

Appendix A.1.1. Logistic Regression

We got the best results for ‘elasticnet’ penalty, C=0.05 (regularization factor), ‘saga’ solver and l1_ratio=0.5.

Appendix A.1.2. Decision Tree

We got the best result with ‘gini’ criterion, ‘best’ splitter, ‘none’ max_depth keeping the min_samples_split=5.

Appendix A.1.3. Support Vector Machine (SVM)

We found that the SVM classifier performed best for C=0.005 (regularization factor) and ‘linear’ kernel.

Appendix A.2. Hyperparameters for Deep Learning Models

Appendix A.2.1. MalConv Model

For the MalConv model, we followed the similar implementation of the original paper (Raff et al., 2018). We used SGD with Nesterov momentum with 0.01 learning rate and 0.9 momentum. We set the max input length to 2MB and 8 samples per batch. We trained the model for 20 epochs.

Appendix A.2.2. M-CNN model

For the M-CNN model, we used the same backbone and hyperparameters mentioned in the original paper (Kalash et al., 2018). We used VGG-16 as the backbone setting the learning rate to 0.001 with 0.9 momentum. We trained the model for 20 epochs.

Appendix A.2.3. Feature-Fusion Model

For this model, we used the implementation that was made available on github by Ahmadi et al. (2016) in their paper. We used ‘random forest’ as the classifier.

Appendix A.3. Deep Learning Model Architectures

Figure A.9: Architectures of (a) MALIGN (Deep Learning) model, (b) MalConv model Raff et al. (2018), and (c) M-CNN model Kalash et al. (2018).
### Appendix B. Result Tables

#### Table B.10: Training Accuracy on Kaggle Microsoft Malware Dataset for Different Machine Learning Models

| Family Name    | Logistic regression | Decision tree | SVM          |
|----------------|---------------------|---------------|--------------|
| Ramnit         | 99.91               | 99.91         | 99.91        |
| Kelihos_ver3   | 99.83               | 99.94         | 99.81        |
| Vundo          | 100                 | 100           | 100          |
| Simda          | 100                 | 100           | 97.92        |
| Tracur         | 100                 | 100           | 99.5         |
| Kelihos_ver1   | 100                 | 100           | 99.5         |
| Obfuscator.ACY | 100                 | 100           | 100          |
| Gatak          | 99.17               | 99.17         | 99.17        |
| Overall        | 99.82               | 99.86         | 99.74        |

#### Table B.11: Training Accuracy of different models on Kaggle Microsoft Malware Classification Challenge Dataset

| Family Name    | MALIGN (Logistic Regression) | MALIGN (Deep Learning) | Feature-Fusion | MalConv | M-CNN |
|----------------|------------------------------|-------------------------|----------------|---------|-------|
| Ramnit         | 99.91                        | 99.58                   | 100            | 98.39   | 97.64 |
| Kelihos_ver3   | 99.83                        | 99.86                   | 100            | 99.89   | 99.91 |
| Vundo          | 100                          | 98.26                   | 100            | 99.4    | 99.26 |
| Simda          | 100                          | 94.44                   | 100            | 100     | 100   |
| Tracur         | 100                          | 98.18                   | 100            | 98.74   | 98.43 |
| Kelihos_ver1   | 100                          | 98.92                   | 100            | 98.54   | 99.52 |
| Obfuscator.ACY | 100                          | 98.66                   | 100            | 97.33   | 99.70 |
| Gatak          | 99.17                        | 99.2                    | 100            | 99.15   | 92.69 |
| Overall        | 99.82                        | 99.24                   | 100            | 98.96   | 98.4  |

#### Table B.12: Performance of different models on Test-set of Kaggle Microsoft Malware Classification Challenge Dataset for other evaluation metrics

| Family Name    | MALIGN (Logistic Regression) | MalConv | M-CNN |
|----------------|------------------------------|---------|-------|
|                | Recall | Specificity | Precision | Recall | Specificity | Precision | Recall | Specificity | Precision |
| Ramnit         | 99.65  | 99.63       | 99.65     | 96.75  | 94.81       | 94.9      | 90.26  | 86.36       | 86.88     |
| Kelihos_ver3   | 98.98  | 99.32       | 99.32     | 100    | 100         | 100       | 100    | 99.32       | 99.32     |
| Vundo          | 91.67  | 100         | 100       | 96.15  | 93.06       | 95.24     | 98.55  | 96.36       | 94.44     |
| Simda          | 100    | 71.43       | 75        | 52.94  | 0           | 100       | 54.54  | 80          | 85.71     |
| Tracur         | 89.29  | 100         | 100       | 91.62  | 98          | 98.79     | 94.49  | 94.57       | 91.96     |
| Kelihos_ver1   | 98.04  | 95.24       | 96.15     | 96.29  | 95.83       | 96.29     | 90.79  | 98.65       | 98.57     |
| Obfuscator.ACY | 94.53  | 97.52       | 97.58     | 97.5   | 91.15       | 92.12     | 89.03  | 94.47       | 94.19     |
| Gatak          | 95.05  | 97.03       | 96.97     | 97.61  | 100         | 100       | 84.08  | 93.85       | 93.89     |
| Overall        | 96.66  | 98.60       | 98.67     | 97.37  | 96.48       | 97.36     | 93.44  | 94.85       | 94.59     |
## Appendix C. Explainability

### Appendix C.1. Vundo Trojan

We found multiple consensus sequences, i.e. signatures of **MALIGN**, for Vundo Trojan family where code obfuscation was being used by malware authors. Here, we show two such examples in Figures C.10 and C.11. In both of them, we found the same binary sequence being used in **text** and **data** section of PE files. We suspect that – one malware sample is loading another by keeping the code in its **data** section.

#### Figure C.10: Code obfuscation in Vundo Trojan. (a) and (b) have the same code in **.text** and **.data** section, respectively

```plaintext
(a) Code in **.text** segment (MD5 hash = jEzkrOP7GayNmiXdget5)

```

```plaintext
(b) Code in **.text** segment (MD5 hash = kRUx3TuodJsgpOsquadZG)
```

---

| Models          | Train Accuracy | Validation Accuracy | Test Accuracy |
|-----------------|----------------|---------------------|---------------|
| **MALIGN** (Deep Learning) | 97.53          | 81.67               | 79.58         |
| Feature-Fusion  | 96.83          | 58.39               | 62.50         |
| MalConv         | 95.39          | 81.22               | 64.45         |
| **M-CNN**       | **98.4**       | 74.29               | 73.83         |

Table B.13: Performance of Deep Learning Models on MLSEC Dataset for train-validation-test split
(a) Code in .text segment (MD5 hash = g7vEfrR3s49CH8AVuESu)

(b) Code in .text segment (MD5 hash = Gpd5s3Y7HKwXf4NaOOpZk)

(c) Obfuscated Code in .data segment (MD5 hash = 6WbENDkcC750euPGqApQ)

Figure C.11: Code obfuscation in Vundo Trojan. (a) and (b) have the same code in .text section whereas (c) obfuscated that in its .data section
Appendix C.2. Simda Backdoor

We also found that most of the malware from this family is getting matched with consensus sequence where they are calling the `GetCurrentThreadId` function which is one of the most important prerequisites for a backdoor (see Figure C.12), and thus, we found this function in the report by various sandboxes for previously found backdoors, e.g., JoeSandbox JoeSandbox (2023), Falcon Sandbox CrowdStrike (2018-2019), etc.

![GetCurrentThreadId function call by Simda Backdoor matched with consensus sequence](image)

```assembly
; Appendix C.2. Simda Backdoor

We also found that most of the malware from this family is getting matched with consensus sequence where they are calling the `GetCurrentThreadId` function which is one of the most important prerequisites for a backdoor (see Figure C.12), and thus, we found this function in the report by various sandboxes for previously found backdoors, e.g., JoeSandbox JoeSandbox (2023), Falcon Sandbox CrowdStrike (2018-2019), etc.

![GetCurrentThreadId function call by Simda Backdoor matched with consensus sequence](image)
```
Appendix C.3. Kelihos_ver1 Backdoor

Figure C.13 is one of the examples where our consensus sequence got matched with byte sequence that might be related to a malicious function. But here an important observation is – the order of the suspicious calls is different in different samples. All of them are checking for admin privileges and reboot requirement. While the malware in Figures C.13a and C.13b declares IsNTAdmin before RebootCheckInstall, malware in Figure C.13c does not. But still, they get detected by MALign.

(a) First checks for admin privilege, then calls suspicious functions from IEDavpack.dll and deletes file (OYJRabHQ1reDxN3yjUCd)

(b) First checks for admin privilege, then calls suspicious functions from IEDavpack.dll (7cx4QdBWg9wMa2pvOLuo)

(c) First calls suspicious functions from IEDavpack.dll, then checks for admin privilege (d5ki60fcbqeH3SvZzpoB)

Figure C.13: Suspicious Windows function calls by Kelihos (version 1) got detected by MALign
Appendix D. Robustness

Appendix D.1. Plausible Techniques to Evade MA\textsc{align}

To evade our model, some possible attacks and the robustness of our model against them are discussed below:

- **Including pieces of other malware to confuse family detection:** The inclusion of pieces from other malware will be detected but code-pieces of the original family will also be detected at the same time, and consequently, the score for the original family should be higher since it contains more code-pieces than the other family. However, if an attacker can include enough code-segments from other families that can outnumber the score for the real family, MA\textsc{align} might predict a wrong family. One defense against this attack can be to incorporate monotonic classifier İncir Romeo et al. (2018), or non-negative network Fleshman et al. (2018a) with MA\textsc{align}.

- **Code Randomization:** Attackers can try to replace instructions with semantically similar ones to evade MA\textsc{align} Song et al. (2020). However, the attacker will have to change a lot of instructions so that it does not get aligned with enough consensus sequences which is not a trivial task, nor impossible at the same time. To defend against such attacks, we can generate artificial adversarial malware using code randomization and train MA\textsc{align} using them to capture the signature for such code randomizations.

- **Register Reassignment:** An attacker can add multiple instructions that reassign registers and create an adversarial malware that has different sequences than our signature Lucas et al. (2021). One way to defend against this attack with MA\textsc{align} can be – changes in the alignment algorithm (or apply heuristic) so that it becomes flexible against register names and instructions.

- **Instruction Reordering:** Attackers can try changing the order of instructions without harming the functionality of malware. MA\textsc{align} is robust to such attack to some extent, and we discovered that in one of our consensus sequences (see Figure C.13). However, if the order is jumbled to an extreme, possibly through the use of jump instructions, it might get evaded.

Appendix D.2. Direct Attack on MA\textsc{align}

Besides the above mentioned approaches, an attacker can try generating adversarial malware specifically for MA\textsc{align} in a whitebox setting. Since MA\textsc{align} is hard to invert and is non-differentiable, the attacker cannot compute gradient or apply reverse engineering unlike other end-to-end deep learning models. They will have to replace the feature mapping function, or especially the MSA tool with the best approximated model Petti et al. (2021) to create an adversarial sample. But replacing an MSA with approximated DNN is another challenging task. Moreover, when the attacker has to preserve the semantics, creating such an approximated model becomes even harder.
Appendix E. Consensus Sequence Analysis

We analyzed the conserved sequences to estimate the number of informative code blocks for each malware family. Table E.14 (in Appendix) represents the number of consensus sequences, i.e., conserved blocks and the number of informative code blocks for each family in the Kaggle Microsoft dataset. We term a code block ‘informative’ if the value of the corresponding weight in our regression model is positive. The rationale behind this is that - the higher the weight of the consensus sequence, the higher impact it has on the classification.

We can observe from the table that, the number of consensus sequences found by MALIGN is proportional to the number of samples in a malware family. However, the percentage of the number of informative consensus sequences varies across malware families. The percentage values indicate that large fractions of blocks identified through the sequence alignment process are useful for the subsequent classification task.

| Family Name     | Number of Consensus Sequence | Number of Informative Consensus Sequence | Percentage of Informative Consensus Sequence |
|-----------------|------------------------------|-----------------------------------------|--------------------------------------------|
| Ramnit          | 962129                       | 177905                                  | 18.49%                                      |
| Kelihos_ver3    | 331432                       | 105223                                  | 31.75%                                      |
| Vundo           | 27210                        | 17601                                   | 64.69%                                      |
| Simda           | 3098                         | 1194                                    | 38.54%                                      |
| Tracur          | 37967                        | 20548                                   | 54.12%                                      |
| Kelihos_ver1    | 27459                        | 13467                                   | 49.04%                                      |
| Obfuscator.ACY  | 94172                        | 42863                                   | 45.51%                                      |
| Gatak           | 275984                       | 96554                                   | 34.99%                                      |

Table E.14: Number of Consensus Sequence in Malware Families