Experiments on Recognition of Malware based on Static Opcode Occurrence Distribution

University of Cincinnati

Jacob Carlson and Anca Ralescu

April 28, 2023

Abstract

This paper discusses a static method for recognizing malicious code samples by comparing opcode distributions created through a novel approach. Distributions are created by aggregating the number of operations between consecutive calls of an opcode. Creating these distributions for each file and comparing them to ground truth distributions representing benign and malicious code samples offers an accurate method of malicious code detection. This paper also provides a dataset of distributions, as well as describes the methods used to create these distributions.

Keywords

Opcode Distributions, Opcode Dataset

1 Proposed Plan

Static malware analysis is an approach in which an executable is inspected for malware without being executed. The most common methods include matching common patterns or deriving a series of parameters to learn from each file, such as the Ember Data set[7]. This paper proposes a method of static malware analysis in which an executable will be translated into a set of distributions that will be compared against ground truth distributions from both benign and malicious code samples. These comparisons will then be used to determine whether a file is malicious or benign.

All compiled code exists as a sequence of op codes, instructing the machine on what to do at the most basic level. While the sequence in which the opcodes are written is not the exact same order in which they will be executed, it is still a very strong representation of the program at hand. To create a set of distributions for an executable, the number of operations between occurrences of the same opcode will be obtained for each common opcode. This data is then placed into bins and divided by the total number occurrences to create a distribution of the occurrences of each tracked opcode.

This distribution will then be compared to ground truth distributions using the Kullback-Leibler divergence to assess the difference between distributions. These KL divergence values will then be used as input data into a machine learning model to classify a file as benign or malicious.

2 Data Acquisition

To train a network capable of detecting malicious code, a large dataset was needed. Most online datasets for malware detection do not provide raw files in their datasets, especially at scale; however, Practical Security Analytics, an online security blog, has released a dataset of $200,000+$ benign and malicious executables for machine learning purposes[8].

This analysis focuses on the instructions that a machine will execute, so Unix command $objdump\ -d$ was used to disassemble the executable into a human readable form that lists the instructions as they would be received by the operating system executing them. This form provides more than the opcode being executed, so extra information such as the line number and the parameters of the instruction must be removed. Each instruction follows a common form, so python was used to filter each entry down to its opcode. To remove the risk of the executable being used and simplify data access, opcode lists will then be stored and used for future operations instead of the executable form.

3 Distribution Construction

3.1 Single Executable Distribution Construction

In this method, the final goal is to have a set of distributions that can be compared to ground truth distributions to determine whether or not the file contains malicious code. Distributions are created by taking all the jumps between occurrences of a given op code, *op*, and aggregating it in *b* bins using the bin assignment function ϕ to discretize the data. A jump is the number of instructions invoked between two opcodes of the same type. Given the executable file has been converted to an opcode sequence, the indices of each opcode in the list can be used to determine the jumps between successive op codes.

$$bin\ assignment:\ \phi(jump) = \lfloor \frac{jump}{1000} * bin_{size} \rfloor$$

Since executables are of variable length, and it is possible for a jump to be close to the length of the entire executable if occurrences are sparse, jump values are capped at 1000 to reduce extreme outliers. These large jumps are infrequent, so keeping them would skew the distributions and subsequent comparisons to such distributions. Any jump value whose bin assignment falls outside the number of bins is omitted.

Algorithm 1 Single Executable Distribution Algorithm

```
for $op$ in opcode set do
    $N_{op}$ ← number of op occurrences
    $distribution_{op}$ ← 0 vector the size of the number of bins
    for each occurrence of $op$ in sequence = 1 to $(N_{op} - 1)$ do
        $jump_{op\_k} = op_{k+1} - op_k$
        $distribution_{op}[\ \phi(jump_{op\_k})\ ]$ += 1
    end for
    for each bin $b$ in $distribution_{op}$ = 1 to $B$ do
        $distribution_{op}[b]$ \= $(N_{op} - 1)$
    end for
end for
```

3.2 Multiple Executable Distribution Construction

The process to create aggregated distributions is similar to the algorithm for individual files, with the exception that the distribution is aggregated over a large number of files of the same class. This creates a distribution set representative of what a typical jump distribution from a single executable of either class should look like.

The values obtained from comparing a code sample to a ground truth distribution of each class over all opcodes in an opcode set will generate a vector of values twice the size of the opcode set, each distribution is compared to a benign and malicious ground truth. This will be used as the input to machine learning models for each sample.

Algorithm 2 Multiple Executable Distribution Algorithm

```
for $op$ in opcode set do
    $distribution_{op}$ ← 0 vector the size of the number of bins
end for
for executable $l = 1$ to $L$ do
    for $op$ in opcode set do
        $N_{l\_op}$ ← number of op occurrences in $l$
        for each occurrence of $op$ in sequence = 1 to $(N_{l\_op} - 1)$ do
            $jump_{l\_op\_k} = op_{l\_k+1} - op_{l_k}$
            $distribution_{op}[\ \phi(jump_{l\_op\_k})\ ]$ += 1
        end for
    end for
end for
for $op$ in opcode set do
    $total\_jumps_{op}$ ← $\sum_{b=1}^{B} distribution_{op}[b]$
    for each bin $b$ in $distribution_{op}$ = 1 to $B$ do
        $distribution_{op}[b]$ \= $total\_jumps_{op}$
    end for
end for
```

Ten ground truth samples were created for each class, each sample is aggregated over 500 total files and no file was in used in multiple ground truth distributions. To show that there was some form of similarity between distributions of the same class, heatmaps were created to show comparisons between all ground truth samples. Each cell displays the sum of the Kullback-Leibler divergence between two ground truth sets over all the distributions in the set. This does not completely prove that the distributions are similar, but if the value of the sum is low, then that should show that the set of distributions are more similar than if the value is high. Figure 1 shows heatmap examples of two different opcode sets. Both examples show two distinct sets of low KL Divergence, one grouping of benign ground truth distributions and another grouping of malicious ground truth sets.

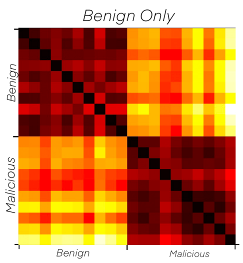

Figure 1: Benign and Malicious Heatmaps
Heatmaps for 20 ground truth samples, 10 of each class. Cells of a darker color indicated lower Kullback-Leibler Divergence values, while cells of a lighter color indicate higher KL Divergence values.

4 Opcode Set Collection

The executables in the training set are all from machines running the Windows operating system, primarily Windows 7[8], but their opcode sets vary and a total of 1,457 different op codes were found in the training set. Only 1.9% of those op codes occur in more than 25% of the training samples, this means that most of the opcodes will not make a difference when being used to make comparisons between files. When an opcode does not occur, the distribution would just be uniform since there were no occurrences to derive jumps from. Comparing uniform distributions to ground truth distributions will not provide any meaningful information, so an effort was made to find an optimal and commonly occurring set of opcodes.

4.1 Benign, Malicious

Simple opcode sets were derived by taking the 50 most occurring opcodes from different sample sets of benign and malicious files only. This ensured that when creating distributions for individual executables, there was a good chance that most of the op codes had enough occurrences to yield a meaningful distribution. 86% of the opcodes overlapped between the two sets, so the distribution sets share most of the same data. This reduces the chances of an opcode from one set being infrequent when compared against either opcode set. The heatmaps for both samples, Figure 2, show a consistent pattern of similarity between ground truth samples of the same class over both opcode sets. The heatmaps look similar due to the number of overlapping opcodes in each set.

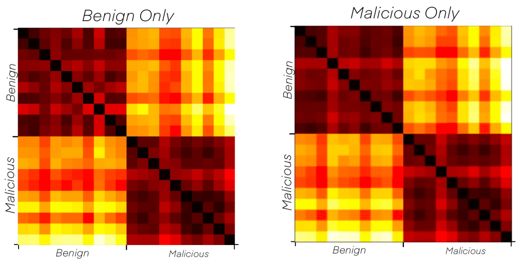

Figure 2: Benign and Malicious Heatmaps
Heatmaps for 20 ground truth samples, 10 of each class. Cells of a darker color indicated lower Kullback-Leibler Divergence values, while cells of a lighter color indicate higher KL Divergence values.

Benign	Malicious
88.1% ± 1.1%	88.7% ± 1.7%

Table 1: Average Accuracy of Benign and Malicious Opcode Sets
Average test accuracies and standard deviations are obtained using ten different sample ground truth sample pairs, tested 4 times each using a different random seed each time to differ the data. Resulting in 40 different test cases.
$Bins: 100$, $KL\ Divergence\ Method: KL(X||dist)$, $Model: MLPScaled$

Using a Signed Wilcoxon Rank Test, where the input data is the difference between the accuracies under the same conditions ($KLDivergenceMethod: KL(X||dist)$), the malicious set was shown to be the more accurate model ($Z = 3.34, p = 1.00$).

4.2 Union, Intersction, Disjoint

Without using more in-depth approaches, the union, intersection, and disjoint of the benign and malicious opcode sets were tested as opcode sets. The union and intersection of the benign and malicious op code sets only differ slightly from the original sets, since those sets were so similar. The benign and malicious sets each have seven opcodes that do not belong to the other set, so the benign and malicious union set has 57 opcodes while the intersection set has 43 opcodes. These sets both provide a valuable insight since the union set has more opcodes, which are frequent to at least one set, while the intersection set has fewer opcodes, but they are all frequent to both opcode sets.

Benign - Intersection:
$[abcb, cmpw, jns, lock, movzwl, sbbb, sete]$
Malicious - Intersection:
$[call, jmpl, leave, popl, retl, sarl, shll]$

When using the previously constructed sets the goal was to find opcodes that occurred frequently enough to create a strong distribution, but the idea behind the disjoint set was to potentially use the lack of distribution to illustrate difference, as was found in Bilar[1]. Opcodes that are common to malicious files but not benign files, and vice versa, were combined in the disjoint set. This potential for increasingly different KL Divergence values when using infrequent opcodes is illustrated in the disjoint heatmap (Figure 3), which does not follow as clear of a pattern as the previous sets heatmaps. There is a lower more consistent summed KL divergence between the malicious ground truth samples, while the Benign samples seems to be much more erratic.

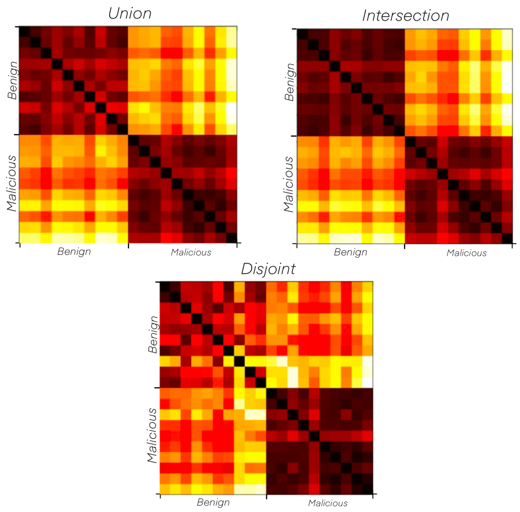

Figure 3: Union, Intersection, Disjoint Heatmaps
Heatmaps for 20 ground truth samples, 10 of each class. Cells of a darker color indicated lower Kullback-Leibler Divergence values, while cells of a lighter color indicate higher KL Divergence values.

Union	Intersection	Disjoint
88.6% ± 1.6%	88.0% ± 1.3%	84.9% ± 1.6%

Table 2: Average Accuracy of Union, Intersection, Disjoint Opcode Sets
Average test accuracies and standard deviations are obtained using ten different sample ground truth sample pairs, tested 4 times each using a different random seed each time to differ the data. Resulting in 40 different test cases.
$KL\ Divergence\ Method : KL(X||dist)$

Opcode Set	Z	p
Union	-0.35	0.37
Intersection	-3.28	0.00
Disjoint	-5.28	0.00

Table 3: Probability of Opcode set yielding higher accuracy than Malicious Opcode Set
Z scores and Probabilities of opcode sets being more accurate than the malicious opcode set when tested with a Wilcoxon Signed Rank test, using the differences between the accuracies of samples tested on the same data under the same conditions.
$KL\ Divergence\ Method : KL(X||dist)$

Out of the newly tested opcode sets, the union of the benign and malicious opcodes was the most accurate, but it still failed to achieve a higher accuracy than the malicious opcode set. The malicious opcode set is a subset of the union opcode set, meaning that the seven opcodes that are a part of the benign set and not the malicious set do not provide any meaningful information and even decrease the accuracy when compared to the malicious only opcode set.

While the intersection opcode set does not have the benign opcodes that seemingly decreased the accuracy of the union set, it has a lower accuracy than both the malicious and union opcode sets. Even though the differences in accuracy are minimal, the opcodes that belong to the malicious opcode set seem to provide enough meaningful information to boost accuracy in a significant way.

The disjoint opcode set is much smaller than any of the other opcode sets that have been tested. Disjoint had an interesting heatmap, showing strong similarity in the malicious opcode set, but had the lowest accuracy by far.

The malicious opcode set group is the most accurate of the opcode set groups tested. The malicious set being more accurate than the union opcode set, deters any further tests into opcode set size since increasing the set size by 14% failed to increase accuracy and any further size increases will come with increased chance of infrequent opcodes. The significantly worse accuracy, under $a = 0.05$, of the intersection and disjoint sets also leads to the conclusion that no further steps need to be taken to investigate opcodes that occur frequently in both sets (intersection) or opcodes that only occur strongly in one set (disjoint).

5 Kullback-Leibler Divergence Method Selection

The Kullback-Leibler Divergence, $D_{KL}(p||q)$, measures the difference of distribution p against distribution q and will be used as the basis of comparison between distributions in this paper. In the context of differentiating a set of distributions against a base standard, it made sense to consider the distribution, or set of distributions, from the given file that we want to classify as p and the ground truth distribution sets as q. This was how all previous accuracies were determined.

5.1 Symmetric KL Divergence — Exploring Flipped Divergence

D_{KL} is asymmetric, and flipped D_{KL} can be used to provide different insights than the standard method. Flipping the KL Divergence, $D_{KL}(q||p)$, as well as taking the average of both options, $(D_{KL}(p||q) + D_{KL}(q||p))/2$, provides alternative values that could provide different and possible more valuable information. Both options have a high probability of being more accurate than $D_{KL}(x||dist)$, but neither are conclusively better since they don't meet the level of significance laid out in this test ($a = 0.05$).

KL Method	Z	p
$D_{KL}(dist\|\|x)$	0.98	0.84
Symmetric	1.06	0.86

Table 4: Probability of KL Divergence method yielding higher accuracy than $D_{KL}(x||dist)$
Z scores and Probabilities of a more accurate model when using a different KL Divergence method compared to $D_{KL}(x||dist)$, using the differences between the accuracies of samples tested on the same data under the same conditions.
$OpcodeSet : MaliciousOnly,$

5.2 Logarithm Mapping KL Divergence Values

KL Divergence values range from zero, when two distributions are exactly alike, to infinity, when distributions share no similarity. While the KL divergence values derived from these distributions fall in a much smaller range upper bound near 10. Machine learning algorithms typically perform better when input data is normalized to a standard range[2]. To normalize and more evenly distribute the data, the KL Divergence value was mapped using log_{10} to put almost all the data in between -1 and $+1$.

On the lower bound, no samples were similar enough to a ground truth distribution to result in a KL Divergence of less than .1, so no samples passed the lower bound of -1. Any KL Divergence value that was greater than 10 will pass the soft boundary of $+1$ after mapped. The number of upper bound outliers is minimal and these values do not range very far above 10. Any outliers will be very close to $+1$ so none of these outliers are significant. Most data ends up within a range of -1 and $+1$ so outliers were not a very large concern.

KL Method	Z	p
$D_{KL}(x\|\|dist)$	-0.75	0.23
$D_{KL}(dist\|\|x)$	0.79	0.79
Symmetric	0.61	0.73

Table 5: Probability that taking the log_{10} of the KL Divergence result in higher Accuracy
Z scores and Probabilities of a higher accuracy when taking the log_{10} of the KL Divergence before being used as input for the model, using the differences between the accuracies of samples tested on the same data under the same conditions.
Opcode Set : *Malicious Only*

KL Method	Z	p
$log_{10}(D_{KL}(x\|\|dist))$	-0.75	0.23
$log_{10}(D_{KL}(dist\|\|x))$	2.18	0.99
$log_{10}(Symmetric)$	0.78	0.78

Table 6: Probability of KL transformations against $D_{KL}(x||dist)$
Z scores and Probabilities of different KL divergence methods being more accurate than $D_{KL}(x||dist)$, using the differences between the accuracies of samples tested on the same data under the same conditions.
Opcode Set : *Malicious Only*

Taking the log_{10} of KL Divergence values did not provide a significant enough increase in accuracy when compared to the same KL Divergence with no transformation. However, when comparing the current leading KL Divergence method, $D_{KL}(x||dist)$, to $log_{10}(D_{KL}(dist||x))$, the latter proved to be more accurate with a high level of significance. $log_{10}(D_{KL}(dist||x))$ will now be the primary KL Divergence method.

This did not change any findings made in the previous sections. While the accuracies of the opcode tests increased they did so consistently across all opcodes sets and did not change how they compare.

6 Test Results and Conclusion

The final model was tested against multiple datasets which were not used to train any models. A set of benign files was obtained from analyzing all files with a *.exe* extension on a clean install of a Windows 10 virtual machine. Three malicious file sets were downloaded from VirusShare[3, 4, 5], an online malware repository for security researchers. Each file set had over $130,000$ malware samples, but only $15,000$ executables were used from each set due to processing time constraints. The first two datasets from VirusShare ($VirusShare\_00000$ and $VirusShare\_00005$) were written in 2012, while $VirusShare\_00451$ was written more recently. These malicious files are curated from user submissions, so there is reason to believe that most of the files in $VirusShare\_00451$ are from current systems.

A final benign set was obtained from my personal computer, running macOS Big Sir, by analyzing all executable files that had *.bundle*, *.dylib*, or *.so* extensions. All the models from the testing portion were saved, leaving 40 total models for the data to be tested against. 250 random selected samples from each test set were tested on every model.

Final Model Parameters
Opcode Set - Malicious Only
KL Divergence - $log_{10}(D_{KL}(dist||x))$
Bins - 100
Model - Multi-Layer Perceptron,
hidden layers - [100, 200, 50]

Sample Set	**Class**	**Accuracy**
Windows10 Virtual	Benign	79.3% ± 5.8%
VirusShare_00000	Malicious	98.4% ± 0.9%
VirusShare_00005	Malicious	97.5% ± 1.3%
VirusShare_00451	Malicious	94.7% ± 3.5%
MacOS Big Sur	Benign	35.8% ± 4.4%

Table 7: Average Accuracy of models on purely Test Datasets

Test accuracies and standard deviations are obtained using ten different samples ground truth sample pairs, tested 4 times each using a different random seed each time to differ the data. Resulting in 40 different test cases.
$Opcode\ Set : MaliciousOnly$
$KL\ Divergence\ Method : log_{10}(D_{KL}(dist||x))$

All three of the malicious data sets were more accurate than the accuracies from validation sets during training, while the benign samples faltered below that accuracy. The weighted average of the accuracies is very similar to that of the tests conducted on these models during training, ignoring the MacOS files. There was no reason to believe that the MacOS files would be accurately classified when using a model trained on executable obtained from a Window's 7 operating system[8], but the fact that it had an accuracy less than a random decision maker would suggest that there is data to be learned from the MacOS samples it just does not correlate with the windows data.

The increase in accuracy over the Malicious data sets could be due to the difference in samples under the training set and test sets. VirusShare may have stricter guidelines for what is considered malware. The training set classifies malicious files based of whether multiple antivirus software identifies it as malicious[8], while VirusShare is cultivated by a digital forensics team[3, 4, 5]. This more hands on approach could mean that the malicious samples in the test set are more malicious and differ more from benign files than the malicious samples in training did.

The more recent VirusShare sample was less accurate than the older samples. The models were trained on Windows 7 files[8], which was the most popular windows operating system at the time the older sets were curated[6]. The models most likely performed very strongly on malicious code from Windows 7 Operating Systems since that what it was trained on, it is even possible that there is overlap between samples but there is no way to verify that. Similar logic could be used to attribute the low accuracy of the Windows 10 samples.

References

[1] Daniel Bilar. "Opcodes as predictor for malware". In: *Int. J. Electronic Security and Digital Forensics* (2017).

[2] Birmohan Sign Dalwinder Singh. "Investigating the Impact of Data Normalization on Classification Performance". In: *Applied Soft Computing Journal,* (2019).

[3] Corvus Forensics. "VirusShare_00001". In: *Virus Share* (2012).

[4] Corvus Forensics. "VirusShare_00005". In: *Virus Share* (2012).

[5] Corvus Forensics. "VirusShare_00451". In: *Virus Share* (2022).

[6] statcounter GlobalStats. "Desktop Windows Version Market Share Worldwide". In: (2012).

[7] Phil Roth Hyrum S. Anderson. "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models". In: (2018).

[8] Micheal Lester. *PE Malware Machine Learning Dataset*. 2021.