Developing a Binary Similarity Model for Malware Classification Using Random Forest Machine Learning

Krystian Bista
Arizona State University
Phoenix, Arizona
kbista1@asu.edu

Karanpreet Singh
Arizona State University
Phoenix, Arizona
bbedi@asu.edu

Darci Vincent
Arizona State University
Phoenix, Arizona
dvincen8@asu.edu

Riley Hall
Arizona State University
Phoenix, Arizona
rkhall3@asu.edu

Gursharan Singh
Arizona State University
Scottsdale, Arizona
gsingh72@asu.edu

Rishik Kolli
Arizona State University
Phoenix, Arizona
rkolli1@asu.edu

Matthew Keeley
ProDefense
Phoenix, Arizona
mkeeley@prodefense.io

Nathan Smith
ProDefense
Phoenix, Arizona
nsmith@prodefense.io

Abstract

Several extensive research studies have produced machine learning models that can classify malicious files according to the family in which they belong. Many studies also explore how to reverse engineer malware binaries for feature extraction, and there exist open-source malware datasets consisting of millions of processed benign and malware samples classified by family. We work to improve the effort against malware attacks; to further the study of SmokeLoader and ZeusBot malware by reverse engineering the samples, extracting features, and training and testing a machine learning model on the binary samples similarity for malware classification by family. Here we draw ideas from previous efforts to reverse engineer binaries and create an original feature model to train a machine learning model to achieve a highly accurate model that can classify binary samples as benign, as malware belonging to the SmokeLoader family, or as malware belonging to the ZeusBot family. Increasing the speed in which malware can be identified through a machine learning model will provide a quicker response to cyber attacks. It also allows for new samples to be quickly detected and allows cybersecurity efforts to combat malware before it becomes widely distributed.
Keywords: Malware classification, Machine learning, Random forest, Reverse engineering, Feature extraction, Binary similarity

1 Introduction

Cybercrime remains an ever-changing and dynamic threat to computer systems and networks today. An assortment of malware exists, differing in design, function, and specific aim, rendering the task of protecting against such attacks quite difficult. Malicious code and operations range from malware groups such as spyware, viruses, rootkits, trojans, adware, ransomware, and worms. There exists a major challenge in confronting these attacks and activities in large part because of how many different types of malware exist, and this is further com-

pounded and exacerbated by their sub-variants. An example of such is the many different types of trojan malware that exist, which, differing in their design and function, makes the task of trying to protect just against trojan malware as being considerably difficult in its own right.

To combat this, the correct classification of malware in terms of overall malware group (i.e. spyware, viruses, rootkits, and such), as well as their sub-variants, which can be considered as malicious families of code, is essential. By being able to quickly analyze and classify a given sample of code as belonging to a specific malware family, and by extension, malware group, enhances understanding of current and new threats, and may serve to improve the pace of response to these threats.

Malware today incorporates many obfuscation techniques and methods to change their code signature, but these do not alter their behavior and or functionality. Abilities do exist to reverse engineer the malware binaries, despite these obfuscations, and many binaries have been fully reverse engineered and stored in massive repositories such as those in Malware Bazaar, to enable the study of these samples [1]. As such, analysis of these samples can yield many considerable similarities for particular malware sub-variants or families.

In this project, we take advantage of these similarities as features upon which several sub-variants of malware can be defined and classified. These features include a mix of generic fields as well as features such as functions used that are very common to specific malware families and can be used to hone in on malware types. Since many malware samples function by way of abusing common built-in operating system functions and libraries, we can take advantage of the specific functions used by given samples as ways to indicate what the sample is trying to do, and therefore classify it accordingly.

2 Dataset

For this project we chose to work on classifying using machine learning the sub-variants of the trojan malware family. Trojan malware consists of many different sub-variants that differ in what they specifically target and how they target their intended recipients. Of course, these sub-variants share similarities, ones that this project attempts to exploit as features that can be used for their classification. Since these sub-variants are all trojans as well, these features can be used across several sub-variants, since similarities exist between these malware families as well.

This project involved reverse-engineering and analyzing samples of two initial families belonging to the trojan malware group: SmokeLoader, and ZeusBot. Additional samples of benign software were also acquired to be labeled as being non-malicious ("Benign") as well. Table **??** shows a portion of these reverse engineered samples, including their corresponding features as well as their known classification. The total number of sample binaries acquired included 150 with a roughly balanced mix of SmokeLoader, ZeusBot, and Benign samples.

2.1 Data Acquisition

These binaries were acquired primarily from the online repository Malware Bazaar, which had classified binaries into their respective malware family groups, and allowed us to analyze them and observe what similarities existed between different samples of SmokeLoader as well as between different samples of ZeusBot. In addition, benign samples of code were also included as part of the effort to be able to distinguish between potentially-harmful and harmless sample code. These benign samples were acquired from the Windows-Classic-Samples [2] and Windows-Universal-Samples [3] repositories on GitHub, and used the same features.

As can be seen in Table **??**, observations and analysis of samples of SmokeLoader and ZeusBot, between samples of the same type and between the malware families yielded several features. These features can be viewed in greater detail in Table 1, along with their respective definitions.

2.2 Data Preparation

Once the malware samples were acquired from the repositories and reverse-engineered to acquire the respective fields, the data was then cleaned, formatted, and went into an encoding script to prepare it for training and testing in a machine learning model.

Several fields above were not numerical values, and therefore required an encoding scheme. One-hot encoding is a common scheme used to encode string data, where a new feature column is generated for an instance of a given string [5]. As a re-

Table 1: Feature Definitions

Feature	Definition
MD5 Hash	An alphanumeric string that can be used to identify a malware file.
SHA256 Hash	An alphanumeric string that can be used to identify a malware file.
Image Size	Numerical value representing file size.
Num Imports	The number of functions imported by the given sample. This corresponds to the listed functions under the "IMPORTS" feature column.
Imports	A list of modules and functions belonging to a given operating system that were noticeably leveraged by the given sample. Each module and function name is a string value. If a file was packed, or no matching imports found, this was noted in the data.
Target	The name of the malware family the given sample belongs to, which included: "Benign", "SmokeLoader", and "ZeusBot".

sult, features such as "MD5 HASH" and "SHA256 HASH" have new features developed where the name of the new feature is the previous feature prefix (i.e. "MD5 HASH" or "SHA256 HASH") followed by the value of that feature for a given sample.

Similarly, features with list data types, such as the "IMPORTS" have values separated out by spaces. Ensuring that such values are space-separated was done as part of the cleaning phase of the data, and once done, the encoding phase took each value, and like the string values for the hash features, one-hot encoded each list value. This resulted in a new feature column for each function or module name, which again followed the naming convention of using the previous feature name prefix and the name of the sample value (i.e. the module or function name).

Since many samples used the same modules or functions, this method did not produce an overabundance of features with respect to the sample size. However, it should be noted that this method of encoding can produce such an overabundance and such needs to be reviewed as one follows this process.

The original features for the hash values and listed functions, post-encoding, were dropped from the processed data set file to remove all non-numeric fields and data.

3 Techniques and Tools

3.1 Machine Learning Model Selection

Many models have been used and are actively being used and researched for applications pertaining to malware detection and classification. Such include encoders and decoders, transformers, and random forests. This latter-most model seemed of particular interest due to its considerable flexibility as a model. Random forest models essentially aggregate decision trees, where each decision tree is different in how it determines classifications, thereby allowing for increased variability and flexibility in analysis.

As a consequence, several strengths emerge in using this model, notably that they are able to learn non-linear decision boundaries, can reduce overfitting due to a number of different decision trees produced, can provide information with respect to which features are actually used to split the data and reach decisions as well as which features minimally help towards that end, and also enable considerable scalability and high accuracy, even for small sample sets. Given these strengths, this model seemed optimal towards our use-cases and end goal of developing a quick and robust method for malware classification.

Random forests require minimal hyperparameter tuning and as such provided high-level accuracies for our sample set right out-of-the-box. However, some tuning was done, particularly including changing the number of samples needed for an internal node to split and the randomness of the bootstrapping. The parameter `min_samples_split` was set to 10 from its default value of 2 and the parameter `random_state` was set to 5 from its default value of None. The rest of the parameters were left unchanged.

3.2 Training and Testing

In addition to tuning the model slightly, several training and testing splits were made, but given the size of our small sample set, we resorted to using a split of 60–40, where 60% of our data samples were used for training and 40% for testing. We did not have a validation data set.

4 Results

For the model created that was trained using only the data that we collected, we tested two different datasets. The first dataset contained all the features like hashes, functions, and number of imports, whereas the second dataset contained all of the features except for the hashes. When running both of these datasets through the model without changing any of the hyperparameters, the accuracy for the first dataset, shown in Table 2, was 95% and the accuracy for the second dataset, shown in Table 3, was about 98%. Precision, recall, and F1 score were also calculated for both datasets and are shown in the tables below. Table 4 displays the statistics for the model using the first dataset that contained all features concatenated with the filtered BODMAS dataset. See Conclusion and Future Work section for details.

Table 2: Performance Metrics: Dataset with All Features

Statistic	**Result**
Accuracy	0.95
Precision	≈ 0.9541
Recall	0.95
F1	≈ 0.9494

Table 3: Performance Metrics: Dataset without Hash Features

Statistic	**Result**
Accuracy	≈ 0.9833
Precision	≈ 0.9843
Recall	≈ 0.9833
F1	≈ 0.9834

Table 4: Performance Metrics: Combined Dataset with BODMAS

Statistic	**Result**
Accuracy	≈ 0.9966
Precision	≈ 0.9967
Recall	≈ 0.9966
F1	≈ 0.9966

5 Analysis and Findings

The two tests for this project indicate several positive points about our initial hypothesis in using common operating system functions and modules, along with a Random Forest model to classify malware. Due to the important nature of malware classification, and the catastrophic consequences that can result from failing to correctly classify malware, the most important optimization was for recall. Recall effectively evaluates how well the model is able to classify all positive instances of a given target class with respect to the actual total number of positive instances of that target class. Mathematically this is represented as the number of true positive instances identified by the model divided by the total number of positive instances for a given class. As such, both tests indicated high scores for recall, with Table 2 results showing a recall of 95% and Table 3 that of 98.3%.

Our initial benchmark accuracy with respect to recall was at 90%; that is, we were hoping to achieve a 90% minimum in recall accuracy. Since these two scores exceed that, several points come into light. First, the use of common operating system functions and modules as features can be used for malware detection with considerable accuracy. The creation of so many new features due to the encoding technique was not detrimental to the performance of the model, for which the choice of the model was likely a contributing factor in offsetting this potential issue. Second, our initial feature model included an over-abundance of features that actually negatively affected the performance of the model and that a reduction in the features, namely the hashes, improved the score considerably by about 3.3%.

This second finding is of particular interest and importance here. Namely, the hash functions are not particularly telling about what a given sample does or how it functions. Rather, the list of functions and modules do that. Therefore, though hashes are used in other malware analysis tools as features [4], their impact here is negative, and the use of function and module names is both logical and experimentally valid for use as a feature model. Any other features added that serve to complement the function and module names, such as our use of the number of functions and modules used by each sample, provides additional context to the function and module names, which aids in overall accuracy.

It seems that using features that do not provide context of functionality may not be bad, but the hash features are seemingly semantically null insofar as no significant distinguishing information could be gained from them (every sample has a unique hash value). Our disposition is that careful analysis of any feature should be based on at least two things: one, the feature explains to some significant degree how a given sample functions; two, the feature should be a rather dynamic quality of a given set of samples, that is, the feature should be a necessary property about samples. In the case of the function and module names, this set of features explains what operating system functions a sample uses, and essentially serves as a property of all samples since every program uses and needs to use operating system functionality.

5.1 Alternative Approach

Other options were explored to increase the accuracy of our model. We explored "merging" our dataset with an open-source dataset composed of processed benign and malignant samples. Ultimately, we chose to work with the BODMAS Malware Dataset because the dataset was large enough to increase our sample size significantly without being too large to be inconvenient to work with [4]. The BODMAS dataset contains 134,435 total processed samples while other datasets such as the EMBER dataset contain over 1 million total samples [11].

Both EMBER and BODMAS datasets use the same feature extraction code from the LIEF project [12], which is also open-source. After analyzing the feature engineering code, we concluded that the main categories of features were extracted: histogram, byte entropy, strings, general, header, section, imports, exports, and data directories. Each feature is defined as a class in the feature engineering code, and further examination must be used to identify exactly what features are being extracted from an arbitrary feature, such as the number of imports.

Given time constraints, the datasets were concatenated in order to have one dataset to train and test the model on. This was achieved by filling in the missing features in our dataset with null values and concatenating the datasets (our dataset contains 348 features per sample, while BODMAS has 2,381 features per sample). This is not ideal as it diverges from our original feature model. Ideally, we would have run the samples collected by our team through the feature extraction code used by the BODMAS and EMBER datasets to create one uniform dataset that has over 2,381 features extracted.

This alternative approach was eventually abandoned because it deviates too much from our original goal and feature model used in the final iteration of our machine learning model. Despite its abandonment, the sample size was increased from 149 samples to 1,490 samples and achieved an accuracy rate of 99.6%. Further work should include a uniform dataset with a comparable amount of features to the BODMAS dataset.

6 Conclusion and Future Work

In this project, we examined the efficacy of using the Random Forest machine learning model to classify sub-variants of the trojan malware group. Our work is clearly limited in scope and size, but the work shown here does show potential and can be expanded upon.

Namely, it is of interest of how many malware families may be added as part of a dataset. In our project, we only used SmokeLoader, ZeusBot, and Benign samples, but many other trojans subcategories could be added using the same feature model. Expanding the number of samples for each family we had, as well as adding new families in proper proportion to existing families, would yield a more capable model of classifying more sub-variants. However, it would also be interesting to note if limits exist on the number of families that might be classified without performance suffering.

It is worth noting furthermore that achieving a balance of various malware families may not be easy to do. In the case of this project, samples of SmokeLoader and ZeusBot were in abundance, such that creating a dataset that balances the number of families out evenly was an easy task. However, some malware families have very few, if any, available samples for training and testing, rendering this a worthwhile consideration. In such a case, the selection of the Random Forest model does help in protecting against this, since the model's design allows for finding ways to classify one family from another based on feature values, which does not depend too heavily on the number of samples available. How-

ever, there is of course a logical limit to this, since if there are too few samples, and such samples have very similar feature values to others, accuracy may come into question, and dealing with this problem may require other techniques not mentioned in this paper.

One solution to this as well as to improving overall performance involves expanding the feature model. The feature model is rather robust and dynamic insofar as it builds features according to functions used by the malware. Inevitably, malware will need to use some operating system modules and functions, and so such features can be used. But, other features are also promising and just as dynamic, including the use of visualizing malware binaries as image files. This method has been proposed and used by others with considerable accuracies being reached for classification [5].

In all, we find that the use of functions and some simple sample metrics such as file size and the number of imports show promise in analyzing and classifying malware. In further pursuance of this work we believe that the next step is to increase the number of different malware variants that are included in the data set for training and testing. The number added would depend on how many could be added to the data set without overwhelming the Random Forest model and causing the performance to suffer. Just as this was mentioned earlier, this would require further study, but doing so is obviously necessary for determining how many Random Forest models would be needed for a practical application. Other steps after this would involve expanding and or modifying the feature model to include pixel images or other dynamic features, as they were defined here in this paper. Ultimately, the build-up of these steps mentioned here would shed great light on the accuracies and constraints of the dynamic feature approach using Random Forest models, and elaborate on their possible and promising utility for practical applications for malware classification.

References

[1] "Malware Sample Exchange," MalwareBazaar. [Online]. Available: https://bazaar.abuse.ch/. Accessed: Apr. 1, 2024.

[2] Microsoft, "Microsoft/Windows-Classic-samples: This repo contains samples that demonstrate the API used in windows classic desktop applications.," GitHub. [Online]. Available: https://github.com/microsoft/Windows-classic-samples/tree/main. Accessed: Apr. 1, 2024.

[3] Microsoft, "Microsoft/Windows-Universal-samples: API samples for the universal windows platform.," GitHub. [Online]. Available: https://github.com/Microsoft/Windows-universal-samples. Accessed: Apr. 1, 2024.

[4] L. Yang, A. Ciptadi, I. Laziuk, A. Ahmadzadeh, and G. Wang, "BODMAS: An Open Dataset for Learning based Temporal Analysis of PE Malware," BODMAS Malware Dataset, 2021. [Online]. Available: https://whyisyoung.github.io/BODMAS/. Accessed: Mar. 2024.

[5] "Data Science in 5 minutes: What is one hot encoding?," Educative. [Online]. Available: https://www.educative.io/blog/one-hot-encoding. Accessed: Apr. 1, 2024.

[6] F. C. Garcia and F. Mugga, "Malware Classification Using Machine Learning," arXiv preprint, 2020.

[7] Malwarebytes Labs, "The life and death of the ZeuS Trojan," Malwarebytes, Jul. 21, 2021. [Online]. Available: https://www.malwarebytes.com/blog/news/2021/07/the-life-and-death-of-the-zeus-trojan. Accessed: Apr. 1, 2024.

[8] "SOFTWARE ASSURANCE Technical White Paper Reversal and Analysis of Zeus and SpyEye Banking Trojans," IOActive, Inc., Seattle, WA, 2012. [Online]. Available: https://ioactive.com/pdfs/ZeusSpyEyeBankingTrojanAnalysis.pdf. Accessed: Apr. 1, 2024.

[9] A. Elshinbary, "SmokeLoader Malware Analysis," n1ghtw0lf, Jun. 21, 2020. [Online]. Available: https://n1ght-w0lf.github.io/malware%20analysis/smokeloader/. Accessed: Apr. 10, 2024.

[10] "Smoke Loader — Malware Trends Tracker," ANY.RUN, Mar. 06, 2023. [Online]. Avail-

able: https://any.run/malware-trends/smoke. Accessed: Apr. 1, 2024.

[11] H. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," arXiv preprint arXiv:1804.04637, 2018. [Online]. Available: https://arxiv.org/pdf/1804.04637v2.pdf. Accessed: Apr. 17, 2024.

[12] R. Thomas, "LIEF - Library to Instrument Executable Formats," GitHub, Apr. 2017. [Online]. Available: https://lief.quarkslab.com/. Accessed: Apr. 17, 2024.