Building a Machine Learning Classifier for Malware Detection

Zane Markel and Michael Bilzor
Computer Science Department
U.S. Naval Academy
Annapolis, MD
m154500@usna.edu, bilzor@usna.edu

***Abstract*—Current signature-based antivirus software is ineffective against many modern malicious software threats. Machine learning methods can be used to create more effective antimalware software, capable of detecting even zero-day attacks. Some studies have investigated the plausibility of applying machine learning to malware detection, primarily using features from n-grams of an executables file's byte code.**

We propose an approach that primarily learns from metadata, mostly contained in the headers of executable files, specifically the Windows Portable Executable 32-bit (PE32) file format. Our experiments indicate that executable file metadata is highly discriminative between malware and benign software. We also employ various machine learning methods, finding that Decision Tree classifiers outperform Logistic Regression and Naive Bayes in this setting. We analyze various features of the PE32 header and identify those most suitable for machine learning classifiers. Finally, we evaluate changes in classifier performance when the malware prevalence (fraction of malware versus benign software) is varied.

I. Introduction

Modern antivirus software is effective at detecting known threats, but can be evaded by novel malware. A study by the Australian computer security team AusCERT found that 80 percent of new malware was not detected by the latest antivirus software [6]. In 2013, the security firm iMPERVA gathered over 80 new virus samples and ran them through 40 of the best antivirus products available; only 5 percent of the new samples were correctly classified as malware [4].

A. Limitations of Traditional Antivirus

Traditional signature-based antivirus products are reactive by nature. Malware analysts manually generate a *signature*, usually a hash, to detect a specific piece of malware and add the signature to a malware database. Antivirus software consults the database of signatures during each new scan. This method was effective when viruses, trojans, and worms were first written, but malware analysts can no longer keep up with manual analysis, thanks to automated malware polymorphism and obfuscation. Signature-based detection may not identify zero-day attacks — malicious files targeting vulnerabilities that are previously undisclosed. In order to create a more robust and reliable antivirus product, we need to develop alternatives that complement traditional signature-based detection.

B. Machine Learning in Malware Classification

Supervised machine learning classification can be used to address some of the limitations of signature-based malware detection. To use machine learning to classify files as malicious or benign, one must first build labeled datasets for training. The *features* of each file are derived from some specific characteristics of the file, and each file is *labeled* as either benign or malicious. Learning algorithms analyze training records to generate a model that maps the relationship of file features and labels. That model — the classifier — is used to predict the class of each record in the *test* set, which contain file records without labels.

C. Related Results

Previous attempts to build a machine learning malware classifier have had mixed results. Much research to date has focused on using n-gram features derived from a file's binary code. For instance, Kolter and Maloof applied machine learning to n-grams of malicious and benign software, and their model detected 98 percent of malware while only incorrectly guessing 5 percent of benign software [5]. Santos et al. correctly identified 74 percent of 1,000 malware samples as malicious while correctly identifying all of 1,000 benign software samples [8]. To be useful in an operational system, though, such a classifier must be even more accurate, while maintaining a very low rate of false positives.

D. Some Header Data Differ in Malware

32-bit Portable Executable (PE32) is the executable file format for 32-bit Windows machines. The PE32 header data contains many fields which describe the structure of the executable file. In 2012, Yonts published statistical comparisons between the header data of clean and malicious PE32 files [12], showing that malicious and benign programs frequently differ in certain components of the header. Yonts manually designed single-feature detection rules that individually detected many malicious files while falsely alerting on relatively few benign files [12]. The Yonts analysis did not attempt to combine multiple features or use machine learning to build a general classifier, however.

Yan et al. reinforced this finding by comparing how well various feature types discriminate among different malware families. They examined n-grams of a file's binary contents and disassembled x86 code, its dynamic execution traces, and

the file's PE32 header data. Of these feature types, PE32 header data yielded the best classification performance [11]. While the study did classify between malware families, it did not classify between malicious and benign files. No single file feature has been shown to effectively discriminate between malicious and benign files with high accuracy, but the findings of both Yonts and Yan, et al. suggest that a well-designed machine learning algorithm may be able to discriminate between clean and malicious files by examining a collection of header features and other metadata.

PE32 header data is attractive for malware detection because many header features are intrinsic to a program's structure and might therefore be difficult for an attacker to manipulate without affecting program function. Thus, malware authors will not be able to easily evade the detection rules for machine learning based malware detection simply by modifying the header data alone. Also, some zero-day attacks might have a structure similar to known malware, so a machine learning based antivirus program might be able to detect previously unseen malware via similarities in the PE data. Finally, an operational malware detector could quickly gather a file's PE data without having to run the program or perform any complicated analysis.

II. Goals and Hypothesis

Our goal in this research is to build a machine learning classifier that can discriminate between malware and benign software. We hypothesize that machine learning classifiers can successfully discriminate between malware and benign software by learning from intrinsic file characteristics and metadata, such as PE32 file header data. In pursuit of this general hypothesis, we explore several sub-goals:

- Construct and compare malware classifiers using three commonly-used machine learning algorithms, Naive Bayes, Logistic Regression, and Decision Trees.
- Explore the effect of using different proportions of malware and benign software in training and test data.
- Identify PE file headers that can be easily modified by an attacker without effecting file execution.
- Determine which PE file header features individually discriminate best between malware and benign software.

III. Experimental Setup

A. Data

To train the classifiers, we constructed a database of known clean and malicious PE32 files. The malicious files were randomly selected from a collection we obtained from Open Malware, a group dedicated to safely distributing malware for research purposes [10]. There were 122,799 total malicious files used in our trials. To obtain a representative sample of benign files, we scanned all the PE32 files in the 'C:\Windows' and 'C:\Program Files' directories from the following clean installations of Microsoft Windows:

- Windows Vista Enterprise
- Windows 7 Professional
- Windows Server 2008 R2 Standard
- Windows 8.1 Professional

Additionally, we scanned the PE32 files from a diverse set of 46 new installations of the most popular Windows software applications. Some examples included Chrome, Firefox, QuickTime, Microsoft Office, Python, Java, VLC, etc. After removing duplicate files (mostly from common Windows executables), there were 42,003 benign files, for a collection of 164,802 files total.

We used a Python module called `pefile` [1] to extract 44 different features from the PE32 headers, plus 2 boolean features derived from the section entropies of a PE32 file, to build a database. Each file in the database had a record containing the features and the label, benign or malicious, of that file.

B. Method

Each experiment was broken into a set of *trials* in which we sampled the database for training and test data, trained a classifier, and measured classifier performance on test data.

Our sampling algorithm generates non-intersecting training and test samples from a given dataset. We specify the desired number of records as well as the malware prevalence, or *malprev*, which is the percent of records in the sample that correspond to malicious files. We varied *malprev* to see how learning algorithm performance varied when it has plenty of examples of both malware and benign software (50% prevalence), versus a more realistic *malprev* (0.1%). We also provided seeds to randomly generate samples in a reproducible manner. For each trial, we used ten seeds for ten different sub-trials.

After generating training and test samples, we trained a new classifier for each sub-trial using one of several standard, popular machine learning algorithms. Specifically, we used the `scikit-learn` Python module [2] to implement:

- Naive Bayes assuming Gaussian feature distributions
- Logistic Regression using L_1 regularization
- Classification and Regression Tree (CART) Decision Trees with splits computed based on *Entropy*

After being trained, the classifier was used to predict the labels of each record in the test sample. For each sub-trial, the predicted labels were compared with the known labels associated with each record to compute the overall precision, recall, and F-score. For every trial, we found the average and standard deviation F-scores of the sub-trials.

IV. Results

A. Learning Algorithm Comparison

Our first experiment was designed to determine which learning algorithm would most effectively learn at various *malprev* levels. Our trials used 22,500 training samples and tested on 2,500 test samples. The results are shown in Table I.

malprev	Naive Bayes	Decision Tree (CART)	Logistic Regression
(0.5, 0.5)	0.5127	0.9792	0.9456
(0.1, 0.1)	0.4640	0.9270	0.7941
(0.5, 0.1)	0.4804	0.9150	0.7905
(0.01, 0.01)	0.4250	0.7581	0.3023
(0.5, 0.01)	0.3342	0.5247	0.2873
(0.001, 0.001)	0.0493	0.4157	0.03124
(0.5, 0.001)	0.0697	0.1193	0.04857

TABLE I. F-SCORES OF TRIALS WITH VARYING *malprev* AND LEARNING ALGORITHMS. *Malprev* IS EXPRESSED AS A TUPLE CONTAINING THE TRAINING SAMPLE *malprev* AND TEST SAMPLE *malprev*, RESPECTIVELY.

malprev (train, test)	F-Score
(0.1, 0.1)	0.9270
(0.01, 0.01)	0.7581
(0.001, 0.01)	0.4157
(0.5, 0.1)	0.9150
(0.5, 0.01)	0.5247
(0.5, 0.001)	0.1193

TABLE II. F-SCORES OF TRIALS PERFORMED WHEN TRAINING *malprev* WAS KEPT THE SAME AS THE TEST *malprev* (FIRST THREE ROWS) AND WHEN TRAINING *malprev* WAS KEPT AT 0.5 (HIGH *malprev*). DATA SHOWN IS FOR TRIALS WITH CART DECISION TREES.

malprev (train,test)	Pruned Database	All Features
(0.5, 0.5)	0.9792	0.9848
(0.1, 0.1)	0.9270	0.9452
(0.5, 0.1)	0.9150	0.9379
(0.01, 0.01)	0.7581	0.7845
(0.5, 0.01)	0.5247	0.6011
(0.001, 0.001)	0.4157	0.4550
(0.5, 0.001)	0.1193	0.1529

TABLE III. F-SCORES OF TRIALS WITH AND WITHOUT EASILY MODIFIABLE FEATURES. DATA SHOWN IS FOR TRIALS WITH CART DECISION TREES.

Feature	F-score
MajorOperatingSystemVersion	0.8846
Local_Syms_Stripped	0.8597
Line_Num_Stripped	0.8576
Relocs_Stripped	0.8156
MinorOperatingSystemVersion	0.7870
SizeOfStackReserve	0.7752
BaseOfData	0.7471
SizeOfCode	0.7266
SizeOfInitializedData	0.7259
HighEntropy	0.7246

TABLE IV. THE TEN MOST DISCRIMINATORY FEATURES, MEASURED BY F-SCORE WHEN USED AS A SINGLE-FEATURE CLASSIFIER.

Similar to the finding of Yan et al. that Decision Trees are more effective for classifying malware into similar families [11], our trials indicated that, regardless of *malprev*, Decision Trees (using CART) were the best performing learning algorithm for discriminating between malicious and benign files. Naive Bayes outperformed Logistic Regression when test *malprev* was smaller.

B. Malware Prevalence

Operational machine learning based malware detection software would need to be able to detect malware on systems with a very low *malprev*. We hypothesized that test data with lower *malprev* would have both a higher false positive and false negative rate. We also hypothesized that, by keeping *training malprev* at 0.5, performance would be better at lower *test malprev*, since the classifier would have a richer training set to learn from. Table II summarizes the trials we ran to test these hypotheses. These trials used the same sample sizes as those performed for our learning algorithm comparisons trials.

The results confirmed our first hypothesis; classification performance is poorer with lower *malprev*. However, we found that the performance decrease was actually more pronounced when *training malprev* was kept high, as opposed to decreasing it in parallel with the *test maplrev*.

These findings indicate that before machine learning can be employed effectively in an operational setting, further research needs to be done on how to address the problem of "class imbalance" [3] between malicious and benign files, since the malicious ones will be few in a real-world system.

C. Easily Modifiable Features

Of the features we recorded for each file, we identified six that can be easilly modified without effecting file execution: MajorLinkerVersion, MinorLinkerVersion, MajorImageVersion, MinorImageVersion, Year of creation, and NumberOfRvaAndSizes. Because a malware author could easily modify these PE32 header fields to make malware appear more benign to a classifier, we pruned these features for subsequent trials. To test the effect on classification performance of pruning these features, we compared the results of training and testing on databases with and without these features using CART decision trees at varying *malprev*. Table III shows our results.

Classifier performance decreased, but not significantly. A malware author could modify PE32 fields like these to make malware appear more benign to a classifier. At the same time, the classifiers were almost as effective with the easily-modifiable features removed.

D. Feature Isolation

This experiment tested to see which features, in isolation, were most discriminative between malware and benign software. To do this, we split our original database into a database for each feature that only contained feature-label pairs for each record in the original database. For each database, we ran a trial using using $malprev = 0.5$ for both the training and test samples and CART learning. Table IV lists the 10 features whose classifiers achieved the highest average F-score.

The last feature listed in the table, *HighEntropy*, is not a PE32 header field, but a boolean feature we added. *HighEntropy* is true if any section in the corresponding PE32 file has an information entropy greater than 7. Features like entropy could be useful in real-world malware detection because they are easy to measure and are individually highly discriminative between malware and benign software. Additionally, they cannot be easily modified without changing the executable structure. The other features in the table refer to the PE32 header fields of the same name.

V. CONCLUSIONS

The results generally supported our hypothesis that PE32 header data can successfully be used to detect malware through machine learning classification. Decision Trees, in particular, achieved a 0.97 F-score on a balanced sample of malicious and

benign files. Additionally, the results show that many PE32 header fields are indeed individually discriminative between malware and benign software.

While promising, a useful anti-malware system requires a lower error rate. Further research will be necessary to optimize the learning process to achieve an acceptable false positive and false negative rate. Particular attention needs to be paid to the class imbalance problem. We are not aware of other research studies on the effect of varying malware prevalence when using machine learning based malware detection.

VI. Recommendations for Future Work

There are many possibilities for future research in this area. Suggested research includes:

- *Class Imbalance*. Many studies have proposed methods to mitigate the class imbalance problem in general. Because malware is often greatly outnumbered by benign software in the real world, some of these methods could improve performance.
- *Other Algorithms*. Several other learning algorithm families, such as random forests, could potentially perform well. Each algorithm family has a variety of implementations that could be tried. Additionally, some feature selection and preprocessing techniques could potentially increase performance.
- *Combinations*. It should be possible to integrate this methodology with complementary malware detection techniques that examine the machine instructions, imported functions, or other data.
- *More Data*. Classification performance almost always increases with more data. Thus, it would be useful to gather more malware and benign software.
- *Operational System*. Construct a working runtime malware detection system using machine learning-based classifiers.
- *Obfuscated Malware*. Test this methodology against malware that has been sufficiently obfuscated to avoid all traditional virus detection.
- *Defenses*. Explore techniques that an attacker could use to modify PE32 header data to evade detection by classifiers that use it.

Acknowledgments

The authors would like to thank the U.S. Naval Academy Research Office and Trident Scholar program for their generous support.

References

[1] E. Carrera. `pefile` python module, 2014.

[2] F. Pedregosa, et al. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830, Nov. 2011.

[3] X. Guo, Y. Yin, C. Dong, G. Yang, and G. Zhou. On the Class Imbalance Problem. *2008 Fourth International Conference on Natural Computation*, pages 192–201, 2008.

[4] iMPERVA. Assessing the Effectiveness of Antivirus Solutions. *iMPERVA Hacker Intelligence Initiative, Monthly Trend Report*, 2012(14), 2012.

[5] J. Z. Kolter and M. a. Maloof. Learning to Detect and Classify Malicious Executables in the Wild. *Journal of Machine Learning Research*, 7:2721–2744, 2006.

[6] Kotadia. Eighty Percent of New Malware Defeats Antivirus, July 2006.

[7] A. Ng and M. Jordan. On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes. In *Proceedings of Advances in Neural Information Processing Systems*, pages 841–848, 2001.

[8] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas. N-grams-based file signatures for malware detection. In *ICEIS (2)'09*, pages 317–320, 2009.

[9] T. Stibor. A Study of Detecting Computer Viruses in Real-infected Files in the N-gram Representation with Machine Learning Methods. In *Proceedings of the 23rd International Conference on Industrial Engineering and Other Applications of Applied Intelligent Systems - Volume Part I*, IEA/AIE'10, pages 509–519, Berlin, Heidelberg, 2010. Springer-Verlag.

[10] Various. Open malware repository, 2014.

[11] G. Yan, N. Brown, and D. Kong. Exploring discriminatory features for automated malware classification. In *Proceedings of the 10th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'13, pages 41–61, Berlin, Heidelberg, 2013. Springer-Verlag.

[12] J. Yonts and A. Atlasis. Attributes of Malicious Files, 2012.