Selecting Features to Classify Malware

Karthik Raman [#1]

[#] *Product Security Incident Response Team (PSIRT), Adobe Systems, Inc.*
601 Townsend Street, San Francisco, CA 94103
[1] kraman@adobe.com

***Abstract*—Malware is a menace to computing. The lag between malware landing on a user's system and the development of signatures to detect the same malware can prove catastrophic for users. Using data mining, we identify seven key features within the Microsoft PE file format that can be fed to machine-learning algorithms to classify malware. The resulting models classify malware with results comparable to existing research that uses many more features.**

I. Introduction

Malicious computer programs, or malware, are a pressing business and personal computing problem. Malware evolves rapidly. Detecting new malware, which may just be variants of existing malware, is a strain on antivirus engines. If antivirus signatures are too general, then there is greater risk of false positives; if the signatures are too specific, then many variants of the same malware can be undetected. Theoretical results imply that no antivirus (AV) program can detect all computer viruses [1][2]. While the task of an AV program is to detect as many computer viruses as possible, before it can detect a heretofore unknown program as malware, it first needs to classify the unknown program as possibly malicious or clean.

Computer antivirus expert Peter Szor defines a computer virus as "a program that recursively and explicitly copies a possibly evolved copy of itself"[3]. Viruses can be further categorized into worms, Trojans, spyware, rootkits, and logic bombs[4]. In this research, we are concerned with the classification, using static file features, of viruses that are themselves not products of a virus infection. The problem of the correct classification, detection, and repair of programs that are benign to begin and become infected is out of scope. We shall use the term "malware" to refer to the category of viruses we are interested in detecting. The proliferation of malware variants means that classifying malware is one of the biggest challenges in computer security[5][6]. We focus on approaching a solution to this problem using machine learning.

A. Scope

Microsoft Windows operating systems (OSes) are the most widely used OSes in the world today. The prevalence of Windows makes it an attractive environment for virus writers to write malware for. The chief executable format for Windows is the Portable Executable (PE), a Microsoft standard[7][8]. In this research, we focus on malware written in the PE format for 32-bit Windows operating systems.

II. Machine Learning for Malware Detection

Siddiqui et al. used data-mining techniques to detect Trojans[9]. They mined n-grams from the body of Trojans and used these as features. Their dataset consisted of 4722 PE files, of which 3000 were Trojans and 1722 were clean. They used Random Forest and Principal Component Analysis algorithms for feature selection, and the Random Forest algorithm and Support Vector Machines for classification. This method resulted in a 94% detection rate for new Trojans.

Schultz et. al presented a data-mining framework to detect new executables[10]. They used 4266 programs of which 3265 were malicious and 1001 were clean. Three kinds of algorithms were applied: an inductive rule-based learner, a probabilistic predictor, and a multi-classifier. The classifiers were ported into a signature-based detection algorithm which had a detection rate of 97.76%.

Shafiq et al. presented the "PE-Miner" framework[11]. They extracted structural features from executables to detect new malware in real time. To guide feature-selection, they used an analysis of outliers of aggregate values of fields extracted from the PE header of files; they settled on 189 features. They used Redundant Feature Removal, Principal Component Analysis, and Haar Wavelet Transform for feature selection. They used various algorithms for detection. They evaluated their system using the VX Heavens and Malfease datasets and obtained a detection rate of 99% and a false positive rate of less than 0.5%.

Ye et al. presented a system that used objective-oriented association mining to detect malware with a 92% detection rate[12]. Ye et al. presented another system that used analysis of strings in a program's body to detect malware[13]. This system had a 93.8% detection rate.

III. Related Work

Shafiq et al. asked, "Which PE format specific features can be statically extracted from PE files to distinguish between benign and malicious files"[11]? We attempt to answer this question by analyzing each PE format feature we extracted, combining them intelligently, and reducing the set of features for classification to a small set.

Unlike the research of Siddiqui et al., we analyze not only Trojans but also other types of malware. Further, our feature set is the metadata in the PE file, instead of n-grams from the PE file body. They used 4722 files in their experiments[9]. Schultz et. al indicate that in future work they would evaluate their model over a larger data set than used, 4266 files[10]. Shafiq et al. used approximately 15,000 files[11]. We used a larger data set of approximately 116,000 programs, of which 100,000 are malicious and 16,000 are clean. Our data also differs from Shafiq et al.'s work in the number of initial features being analyzed. We look at 100 per file, compared to 189 by Shafiq et al. Further, they had used three feature-selection algorithms to preprocess the data. They obtained the best classification results with the RFR preprocessing. However, the number and list of features obtained from RFR is not published.

Like the work done by Khan et al., we analyze the metadata in a PE file including the DOS header, COFF header, PE optional header, data directories, import table, section table, and resources[14]. We used Microsoft's *pedump* utility to extract features from our data set. We wrote a parser to extract the features of interest and aggregate them where possible. Like Shafiq et al., we do not attempt to unpack binaries before extracting their features. The problem of unpacking compressed or obfuscated binaries is out of scope.

IV. Experiments

Our clean files come from the base installations of Windows XP and Windows 7. Our dirty files come from a subset of the VX Heavens archive at vx.netlux.org. We conduct experiments for attribute (feature) selection and classification using WEKA, a machine-learning workbench with implementations of machine-learning algorithms. It is written for domain experts who "need an environment in which they can easily manipulate data and run experiments themselves"[15].

According to Witten et al., irrelevant features have negative effects on machine learning. Removing them improves the performance of algorithms, speeds them up, represents the problem better, and focuses the user's attention on important variables. They write, "The best way to select attributes is manually, based on a deep understanding of the learning problem and what the attributes actually mean"[16]. We used our experience in malware analysis to select a set of 100 features from the initial 645 features. We included almost all the features of metadata in the PE header, data about each of the 10 sections, and all imports and exports-related features. We created a dataset of 5193 dirty files and 3722 clean files to evaluate these 100 features. We ran the Random Forest algorithm 100 times using each of the 100 features at a time[17]. A Random Forest classifier that used the DebugSize feature alone had 92.34% accuracy. The 13 features that had the highest individual accuracy are listed in Table I in decreasing order of accuracy.

We recognized that some features are correlated highly to each other and that these would be in the same parts of the PE file. In order to separate highly correlated features, we sorted the features in decreasing order of individual accuracy in classification and then categorized them into seven buckets according to where in the PE file the features originated. We would define a new bucket whenever we encountered a feature going down this list from a different part of the PE file. The seven resulting buckets can be labeled DataDirectory, OptionalHeader, Imports, Exports, Resources, Sections, and FileHeader.

TABLE I
Feature Evaluation

Feature Name	Accuracy
DebugSize	0.9234
DebugRVA	0.9224
ImageVersion	0.8898
OperatingSystemVersion	0.8850
SizeOfStackReserve	0.8837
LinkerVersion	0.8599
DllCharacteristics	0.8273
IatRVA	0.8249
ExportSize	0.8146
ExportRVA	0.8122
ExportNameLen	0.8084
ResourceSize	0.8025
ExportFunctionsCount	0.8001

A. Feature Selection Algorithm

To find a minimum feature set, we used the intuition that the most important, less-correlated features would be the features with the highest individual accuracy from each bucket. In this respect, the first feature in each of the seven buckets was DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, VirtualSize2, and NumberOfSections respectively. We ran thirteen experiments with six different machine-learning classifiers to find a minimum feature using the following algorithm:

Let n be the number of features during an iteration. Start with n=1, with a feature set F compromised of the feature with the highest accuracy (DebugSize) from the bucket with the highest accuracy (DataDirectory). An iteration is defined as:

1) Run IBk, J48, J48 Graft, PART, Random Forest, and Ridor classifiers using F[18][19][20][21][17][22]. Note accuracy of model
2) Go to the next bucket in this order: OptionalHeader, Imports, Exports, Resources, Sections, FileHeader. After each bucket has been visited, wrap around to the DataDirectory bucket and continue in this order
3) Pick the feature with the highest individual accuracy from the current bucket that has not already been picked. Add this feature to F. F now contains n+1 features
4) Go to step 1.

We terminated after 13 iterations.

V. Results

The results of the above experiments are graphed in Figures 1 through 6.

Fig. 1. Classification using IBk

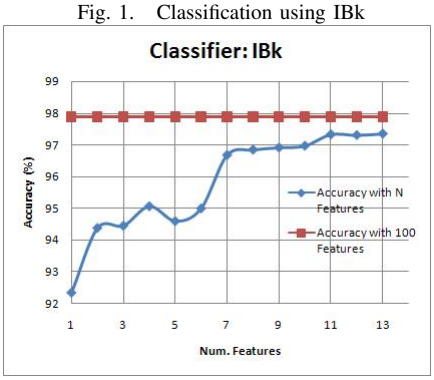


Fig. 2. Classification using J48

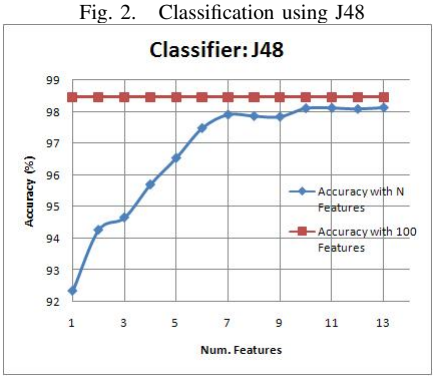


Fig. 3. Classification using J48 Graft

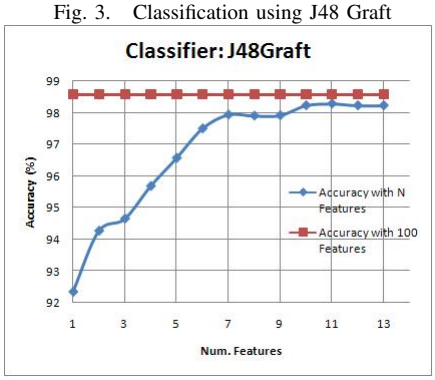
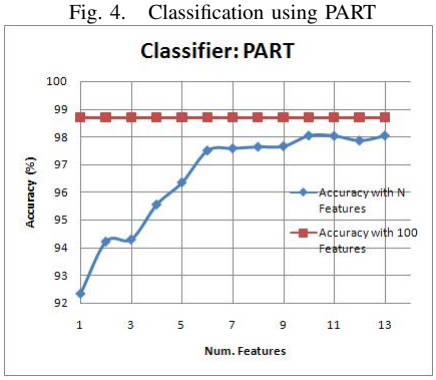


Fig. 4. Classification using PART



In the graphs we see a steep gain in accuracy with each iteration until the eighth one, where the incremental gain in accuracy starts to become small. The features used in the seventh experiment are DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, VirtualSize2, and NumberOfSections. These are the features with the highest individual accuracy taken from each of the seven buckets described earlier. We attempt to explain why these seven features lend themselves to good classification from the definitions of these fields and inspecting our data post-hoc:

1) *DebugSize*. Denotes the size of the debug-directory table. Usually, Microsoft-related executable files have a debug directory. Hence many clean programs may have a non-zero value for DebugSize
2) *ImageVersion*. Denotes the version of the file. It is user-definable and not related to the function of the program. Many clean programs have more versions and a larger image-version set. Most malware have an ImageVersion value of 0
3) *IatRVA*. Denotes the relative-virtual address of the import-address table. The value of this feature is 4096 for most clean files and 0 or a very large value for virus files. Many malware may not use import functions or might obfuscate their import tables[23]
4) *ExportSize*. Denotes the size of the export table. Usually, only DLLs, not executable programs, have export tables. Hence the value of this feature may be non-zero for clean files, which contain many DLLs, and 0 for virus files
5) *ResourceSize*. Denotes the size of the resource section. Some virus files may have no resources. Clean files may have larger resources
6) *VirtualSize2*. Denotes the size of the second section. Many viruses have only one section and the value of this field is 0 for them
7) *NumberOfSections*. Denotes the number of sections. The value of this feature varies in both virus and clean files and it is not clear from inspection how this feature helps separate malware and clean files.

A. Classification with Fewer Features

We divided our dataset of 100,000 malware and 16,000 clean files into five parts, and used the seven features we selected in our experiment as input to IBk, J48, J48 Graft, PART, Random Forest, and Ridor. The averaged evaluations of classifiers, run with ten-fold cross-validation, from the five sets of experiments are presented in Table II. The metrics used are:

- True Positives (TP): Number of dirty files classified as dirty
- True Negatives (TN): Number of clean files classified as clean
- False Positives (FP): Number of clean files classified as dirty
- False Negatives (FN): Number of dirty files classified as clean

Fig. 5. Classification using Random Forest

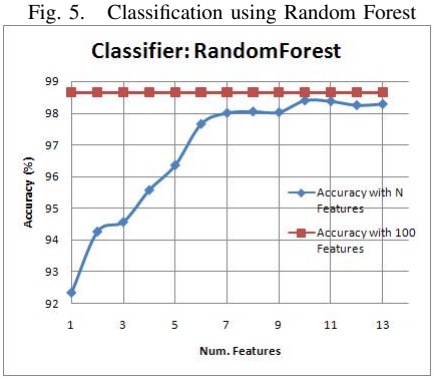
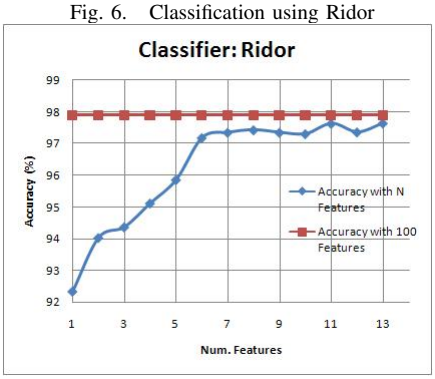


Fig. 6. Classification using Ridor



- True positive rate =

$$TP/(TP+FN) \tag{1}$$

- False positive rate =

$$FP/(TN+FP) \tag{2}$$

- Accuracy =

$$TP+TN/(TP+TN+FP+FN) \tag{3}$$

TABLE II
EVALUATING CLASSIFIERS FED SEVEN FEATURES

Classifier	TP Rate	FP Rate	Accuracy
IBk	0.9730	0.0936	0.9730
Random Forest	0.9822	0.0670	0.9821
J48	0.9856	0.0568	0.9854
J48 Graft	0.9856	0.0592	0.9855
Ridor	0.9792	0.0738	0.9791
PART	0.9822	0.0670	0.9821

B. Discussion

J48 was the best classification algorithm data for these data out of the six classifiers tried. The features DebugSize, ImageVersion, IatRVA, ExportSize, ResourceSize, VirtualSize2, and NumberOfSections each capture key data about different parts of a PE file. When used as input to machine-learning classifiers, they result in models with evaluation results comparable to existing research projects.

We have implemented the classification rules resultant from the models for J48, J48 Graft, PART, and Ridor as a Python script for others to study and extend. This script can be considered a prototypical malware classifier.

Here is a comparison of the performances of our best classifier to results in related research where the number of features used for classification was called out:

TABLE III
COMPARISON TO RELATED RESEARCH

Paper	No. Features Used	TP Rate	FP Rate
Shafiq et al.[11]	189	99%	0.5%
Siddiqui et al.[9]	84	92.4%	9.2%
Khan et al.[14]	42	78.1%	6.7%
This paper	**7**	**98.56%**	**5.68%**

VI. CONCLUSION

In this paper, we presented a set of seven key features that can be used to distinguish between malware and clean programs. To identify these features, we had used the intuition that features from different parts of a PE file would be correlated less with each other and more with the class of the file, dirty or clean. These features can be used as input to machine-learning algorithms to classify malware. The results of this classification can be used by antivirus programs to improve their detection rates.

ACKNOWLEDGMENT

We wish to acknowledge Profs. Athina Markopoulos, Michael Franz, and Donald J. Patterson, and Dr. Christophe Magnan of UC Irvine for their guidance.

REFERENCES

[1] F. Cohen, "Computer viruses: theory and experiments," *Comput. Secur.*, vol. 6, pp. 22–35, February 1987.

[2] D. M. Chess and S. R. White, "An undetectable computer virus," in *In Proceedings of Virus Bulletin Conference*, 2000.

[3] P. Szor, *The Art of Computer Virus Research and Defense*. Upper Saddle River, NJ: Addison Wesley, 2005.

[4] R. Guess and E. Salveggio, *Malicious Code*, 5th ed. John Wiley & Sons, 2009, ch. 16.

[5] A. Walenstein, M. Venable, M. Hayes, C. Thompson, and Lakhotia, "A.: Exploiting similarity between variants to defeat malware: vilo method for comparing and searching binary programs," in *In: Proceedings of BlackHat DC 2007. (2007) https: //blackhat.com/presentations/bh-dc-07/Walenstein/Paper/bh-dc-07-walenstein-WP.pdf*.

[6] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymorphic worm detection using structural information of executables," in *Proc. Eighth Symp. Recent Advances in Intrusion Detection (RAID)*, 2005.

[7] M. Pietrek, "Peering inside the pe: A tour of the win32 portable executable file format," *MSDN Magazine*, March 1994.

[8] ——, "An in-depth look into the win32 portable executable file format," *MSDN Magazine*, February 2002.

[9] M. Siddiqui, M. C. Wang, and J. Lee, "Detecting trojans using data mining techniques." in *IMTIC*, ser. Communications in Computer and Information Science, D. M. A. Hussain, A. Q. K. Rajput, B. S. Chowdhry, and Q. Gee, Eds., vol. 20. Springer, 2008, pp. 400–411.

[10] M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo, "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 38–. [Online]. Available: http://portal.acm.org/citation.cfm?id=882495.884439

[11] M. Z. Shafiq, S. M. Tabish, F. Mirza, and M. Farooq, "Pe-miner: Mining structural information to detect malicious executables in realtime," in *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 121–141.

[12] Y. Ye, D. Wang, T. Li, and Ye, "Imds: Intelligent malware detection system," in *Proccedings of ACM International Conference on Knowlege Discovery and Data Mining (SIGKDD 2007)*, 2007.

[13] Y. Ye, L. Chen, D. Wang, T. Li, Q. Jiang, and M. Zhao, "Sbmds: an interpretable string based malware detection system using svm ensemble with bagging," *Journal in Computer Virology*, vol. 5, no. 4, pp. 283–293, 2009.

[14] H. Khan, F. Mirza, and S. Khayam, "Determining malicious executable distinguishing attributes and low-complexity detection," *Journal in Computer Virology*, pp. 1–11, 2010, 10.1007/s11416-010-0140-6. [Online]. Available: http://dx.doi.org/10.1007/s11416-010-0140-6

[15] G. Holmes, A. Donkin, and I. Witten, "Weka: a machine learning workbench," in *Intelligent Information Systems,1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*, November 1994, pp. 357 –361.

[16] I. Witten, "Weka: Practical machine learning tools and techniques with java implementations," 1999.

[17] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[18] D. Aha and D. Kibler, "Instance-based learning algorithms," *Machine Learning*, vol. 6, pp. 37–66, 1991.

[19] R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers, 1993.

[20] G. Webb, "Decision tree grafting from the all-tests-but-one partition." San Francisco, CA: Morgan Kaufmann, 1999.

[21] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization." in *Proc 15th International Conference on Machine Learning*, ser. Madison, Wisconsin. Morgan Kaufmann, 1998, pp. 144–151.

[22] B. R. Gaines and P. Compton, "Induction of ripple-down rules applied to modeling large databases," *J. Intell. Inf. Syst.*, vol. 5, no. 3, pp. 211–228, 1995.

[23] E. U. Kumar, A. Kapoor, and A. Lakhotia, "Virus bulletin technical feature: Doc - answering the hidden 'call' of a virus," 2005.