12-3-2012

Human-Readable Real-Time Classifications of Malicious Executables

Anselm Teh
Defence Science and Technology Organisation

Arran Stewart
Defence Science and Technology Organisation

DOI: 10.4225/75/57b55339cd8d3
10th Australian Information Security Management Conference, Novotel Langley Hotel, Perth, Western Australia, 3rd-5th December, 2012

HUMAN-READABLE REAL-TIME CLASSIFICATIONS OF MALICIOUS EXECUTABLES

Anselm Teh[1] and Arran Stewart[2]
[1, 2]Defence Science and Technology Organisation
[1]anselm.teh@dsto.defence.gov.au, [2]arran.stewart@dsto.defence.gov.au

Abstract

Shafiq et al. (2009a) propose a non–signature-based technique for detecting malware which applies data mining techniques to features extracted from executable files. Their technique has a high level of accuracy, a low false positive rate, and a speed on par with commercial anti-virus products. One portion of their technique uses a multi-layer perceptron as a classifier, which provides little insight into the reasons for classification. Our experience is that network security analysts prefer tools which provide human-comprehensible reasons for a classification, rather than operating as "black boxes". We therefore build on the results of Shafiq et al. by demonstrating a technique which uses decision trees to distinguish packed from non-packed files, producing a classification diagram which can be understood by analysts. We show that the resulting detector still provides high accuracy and classifies files rapidly.

Keywords
Computer security, malicious executable detection, malware detection, data mining, decision tree.

INTRODUCTION

Commercial anti-virus and anti-malware software typically relies on a set of signatures – strings of bytes – extracted from previously encountered malware, and thus cannot usually detect malware which has not been encountered before. Furthermore, malware authors frequently encrypt or *pack* (compress) their malicious code, transforming the original byte sequence into random-looking data (Lyda and Hamrock, 2007).

Shafiq et al. (2009a) propose a malware detection technique, "PE-Probe", which, rather than using signature-based detection, applies data mining to features extracted from Portable Executable (PE) files – the executable format used on the Microsoft Windows operating system (Microsoft, 2010). Their technique boasts a high level of accuracy, with a true positive rate of approximately 99.5% and false positive rate of 0.6%. Furthermore, the time needed to analyse files was comparable with commercial anti-virus products, making their technique amenable to use in real-time analysis. However, one portion of their design relies on use of a multi-layer perceptron to classify executables as either packed or non-packed. Our experience is that network security analysts have a preference for tools which can provide human-comprehensible reasons for a classification, rather than operating as "black boxes", and multi-layer perceptrons provide analysts with little insight into the reasons for classification.

In this paper, therefore, we build upon the results of Shafiq et al. (2009b) by demonstrating a technique which distinguishes packed from non-packed files using decision trees, which produce far more understandable output. Raftopoulos and Dimitropoulos (2012) highlight this feature of decision trees, noting that:

> "It is important that a security analyst can understand which feature contributed in every step of the process of a decision, without requiring expert statistical knowledge …"

We show that using decision trees to detect packing results in a classifier which provides comprehensible reasons for its classification, suitable for use by a human analyst in guiding their investigation of an executable file, while still providing reasonably high accuracy and classifying files rapidly. Our intent is that this classifier could be used as a quick, "first-pass" filter applied to incoming executable files, and its results could be used by analysts to help prioritise their work and guide their investigation. Files flagged by our classifier as likely to be malicious could be subjected to more intensive analysis.

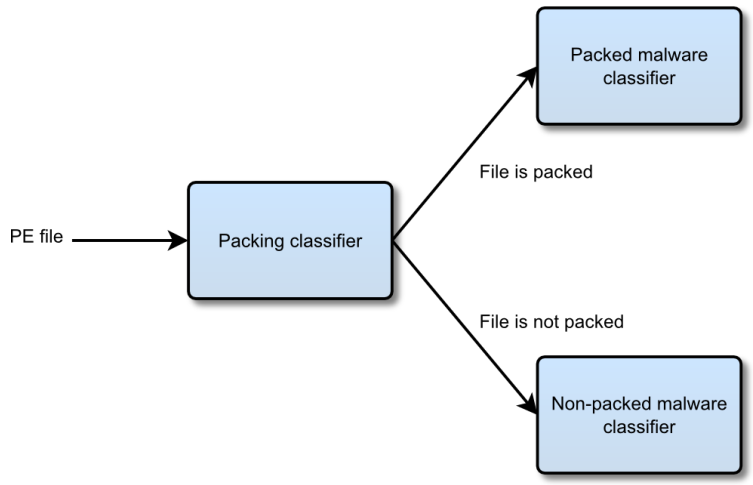

Figure 1 – The PE-Probe classification process. Adapted from Shafiq et al. (2009a)

BACKGROUND

Shafiq et al. (2009a,b,c) proposed two related techniques for detecting malicious PE files.

The first, "PE-Miner" (Shafiq et al. 2009b,c), worked by extracting distinguishing features from the PE files, applying several feature reduction and preprocessing techniques (for example, principal component analysis, or PCA) to the features, and using the resulting features as input to multiple classification algorithms implemented in Weka, an open source data mining toolkit developed at the University of Waikato (Hall et al., 2009). Shafiq et al. aimed to produce a non–signature-based technique with a high detection rate and low false positive rate, which could scan executable files with a speed comparable to commercial off-the-shelf anti-virus products (and thus was a target for deployment in a real-time environment).

The authors found, however, that PE-Miner's performance dropped significantly when trained on non-packed executables, but then tested on packed executables. They therefore proposed a refinement of the technique, "PE-Probe" (Shafiq et al., 2009a). This first classified executables as either packed or non-packed, and then, depending on the result, processed them with one of two possible further classifiers: one specialised to work with packed files, and the other with non-packed files. An overview of this process is shown in Figure 1.

Their packing classifier used a technique proposed by Perdisci et al. (2008a), in which a number of features are extracted from the PE file header, the Shannon entropy of various parts of the file is calculated, and the resulting features are fed into classification algorithms implemented in Weka. Although Perdisci et al. experimented with multiple classification algorithms, Shafiq et al. used a multi-layer perceptron as a classifier, as Perdisci et al. reported that this gave the best results when tested on files that could not be detected as packed by a leading signature-based packing detector.

METHOD

Our aim was to determine whether decision trees could be substituted for multi-layer perceptrons in the technique of Shafiq et al. (2009a), thus providing a human-readable explanation of the output classifications, while still maintaining a high classification speed and level of accuracy.

We obtained benign files from the same sources as Shafiq et al. – a fresh installation of the MS Windows operating system, the Sourceforge web site (Geeknet, Inc, 2012), and the `download.com` site operated by CNET (CBS Interactive, 2012) – although as Shafiq et al. did not provide a full list of files used, we cannot be sure how similar our dataset is to theirs.

These files were scanned with two commercial anti-virus programs – Kaspersky PURE and Sophos Anti-Virus – to ensure they were indeed benign.

To create a dataset of files known to be benign and non-packed, we excluded from the benign dataset any files reported as being packed by the Protection ID tool (CDKiLLER and TippeX, 2010), and any files from which

PE features could not be extracted using the Python `pefile` library (Carrera, 2012). The resulting dataset comprised 679 benign, non-packed files.

To ensure we had a sufficiently large dataset of files known to be benign and packed, we randomly selected non-packed files from the benign dataset, and manually packed them using freely available packers downloaded from the Internet. Each file was packed using the default settings for the packer. If the packer failed with a runtime error, a different member of the benign dataset was randomly selected. After packing, we again excluded files which could not be processed using `pefile`, giving a dataset of 330 packed, benign files.

Malicious files were obtained from two sources – the VX Heavens virus collection (Baranovich, 2012) and the Offensive Computing malware collection (Quist, 2012). Both of these collections labelled some of their files as being "packed". Some files labelled as "packed" were not detected as such by Protection ID, but inspection revealed that they had a Shannon entropy of close to 8 bits per symbol – the maximum for a binary file (Cover and Thomas, 2006, Lyda and Hamrock, 2007) – suggesting that the contents were indeed compressed, encrypted, or both. We therefore included these in our dataset. As with the benign datasets, we excluded any files unable to be processed by `pefile`.

For a dataset of files known to be malicious and packed, we used the subset of the VX Heavens and Offensive Computing files which were labelled as being "packed", and added to these any files which were not expressly labelled as packed, but which were detected as such by Protection ID. The result was a total of 500 packed malicious files.

For a dataset of malicious, non-packed files, we used the remainder of the VX Heavens and Offensive Computing files – that is, files which were not labelled as packed, and for which Protection ID detected no packing. The result was a total of 925 non-packed malicious files.

A breakdown of the datasets is shown in Table 1.

Packing	Benign	Malware	Total
Non-packed	679	925	1604
Packed	330	500	830
Total	1009	1425	2434

Table 1 – Data set breakdown

Feature type	Explanation
Number of standard sections	Number of sections in the following list of standard section names: .text, .data, .rdata, .idata, .edata, .rsrc, .bss, .crt, and .tls.
Number of non-standard sections	Number of sections not in the previous list.
Number of executable sections	Number of sections with the "executable" flag set.
Number of readable/writable/ executable sections	Number of sections with their "read", "write" and "execute" flags all set.
Number of entries in the Import Address Table (IAT)	The IAT contains the addresses of library functions called by the executable – the more entries in the table, the more external functions are called.
Shannon entropy	Shannon entropy was used to calculate four features: entropy of the PE file header, entropy of the "code" sections of the file, entropy of the "data" sections of the file, and entropy of the entire file.

Table 2 – Packer detection features

Feature type	Explanation
DLLs referred to	Which of a list of 73 core DLLs are referred to by the PE file – giving 73 Boolean features.
Image file header fields	Referred to by Shafiq et al. (2009a,b,c) as the "COFF file header". The values of 7 fields from this header, which indicate things such as the target processor type, the number of sections and the number of symbols.
Optional header fields	The content of 9 standard fields, 22 Windows-specific fields, and 30 fields relating to data directories.
Section header fields	The content of the 9 header fields from the .text, .data and .rsrc sections of the file, for a total of 27 fields.
Resource directory table & resources	Counts of various types of resources (such as icons and dialog boxes) used by the program, giving a total of 21 integer attributes.

Table 3 – Malware detection features

Feature extraction

We wrote two programs in the Python programming language to extract distinguishing features from the PE files in our datasets. In this section, we describe these feature extraction programs and give a brief overview of the structure of PE files.

The first program extracted features to be used for packing detection. It consisted of code developed by Perdisci et al. (2008a), modified to output its results in the format used by Weka (Hall et al., 2009). The features used for packing detection are listed in Table 2, and explained in detail in Perdisci et al. (2008a), but we briefly discuss some of them here.

Non-packed files normally draw their section names from a standard list, but packers create sections with non-standard names such as `UPX1`, `.petite`, and `a4z.pq07`. In non-packed files, a section is not normally flagged as both writable and executable, whereas in packed files this is necessary for the unpacking to work. Finally, in non-packed files, the Shannon entropy of the various portions of the file is usually low, indicating redundancy and repetitiveness in the contents, but in packed files is much higher.

Our second program used `pefile` to extract features used by Shafiq et al. (2009c) (listed in Table 3) to distinguish benign from malicious files – these are explained in detail in Shafiq et al. (2009c) and in the PE format specification, but we discuss some of them briefly here.

Benign files typically make use of a wide range of Dynamic-Link Libraries (DLLs), but malicious files use far fewer. Shafiq et al. also note that in their collection of PE files, the malicious files either had far fewer or far more symbols than benign files.

The "Optional Header" of the PE file is optional in that some files – namely, object files – do not require it, although executable files do. It consists of three parts: the "standard fields" (which are defined even for some UNIX variants of this file format) the "Windows-specific fields" (which support Windows-only features) and "data directory" fields – lists of the addresses and sizes of tables contained in the image file, such as the Import Address Table or IAT (Microsoft, 2010).

Malicious files often have atypical values for some of these header fields. For instance, the major version number is often set to zero, and the number of resources such as icons and dialogs is typically lower than for benign files.

Each section header consists of one text field (the name) and nine numeric fields describing the attributes of the section (for instance, the total size of the section when loaded into memory).

Classifier construction

Running our feature extraction programs on our datasets produced, for each file, a list of feature values in the format used by Weka.

Raw lists of features are typically preprocessed to improve accuracy – for instance, Shafiq et al. (2009b) used PCA to preprocess features. However, we avoided using preprocessors which would make the output less comprehensible to analysts. Using PCA, for instance, would result in decision trees that did not contain references to the original PE file attributes, but to synthetic attributes composed of some linear combination of the originals.

To reduce the dimensionality of the malware detection feature set, information gain was used as a metric to evaluate features, and the top 50% of features were retained. For the packing detection feature set, the number of features was already low (nine in total) so no feature selection was performed.

To construct the three decision tree classifiers used in our technique, we applied Weka's implementation of the C4.5 decision tree creation algorithm (Quinlan, 1993) and used 10-fold stratified cross-validation (Han and Kamber, 2006) to test the classifiers. For small datasets such as ours, estimating accuracy using 10-fold stratified cross-validation is generally recommended over alternatives such as the "holdout" method (Hastie et al., 2001, Witten and Frank, 2005).

Feature extraction, model building and testing were done on a Windows XP operating system running on an Intel Pentium Core 2 Duo 3.16 GHz processor with 8 GB of RAM. The version of Weka used was 3.6.4.

	Total instances	**Cross-validation accuracy (%)**	**FPR (%)**	**ROC area**
Result	2434	93.3	8.6	0.925
Result of Perdisci et al.	4493	99.6	—*	0.996

* Not reported

Table 4 – Packing classification results – comparison with Perdisci et al. (2008a)

		Total instances	**Accuracy (%)**	**TPR (%)**	**ROC area**	**FPR (%)**
Non-packed	Result	1604	96.2	96.4	0.962	3.9
	Result of Shafiq et al.	—*	—*	99.4	—*	0.8
Packed	Result	830	92.7	92.7	0.924	7.4
	Result of Shafiq et al.	—*	—*	99.6	—*	0.3

* Not reported

Table 5 – Malware classification results – comparison with Shafiq et al. (2009a)

RESULTS AND ANALYSIS

For each of the classifiers in our technique, we give the results of cross-validation and compare them with previous results.

Table 4 shows results for our packing classifier when compared with the results of Perdisci et al. (2008a) for decision tree classification. (Perdisci et al. did experiment with other classifiers besides basic decision trees. Their highest accuracy was produced using an ensemble of decision trees in a method called "Bagged J48".) The cross-validation accuracy is the proportion of correctly identified samples (Bouckaert et al., 2012). The False Positive Rate (FPR) is

$$\frac{false\ positives}{total\ benign\ files}$$

or in other words, the proportion of incorrectly identified benign files.

A Receiver Operating Characteristic (ROC) graph (Fawcett, 2004) shows the trade-off between true positive and false positive rates for a given model – accuracy can be increased (that is, more true positives identified), at the cost of introducing more false positives. A model with 100% accuracy and no false positives would have an area under the ROC curve of 1; a model with 0% accuracy and all false positives would have an area of 0.

Overall, our model gave similar results to those of Perdisci et al. (2008a), despite the fact that Perdisci et al. used a larger dataset for training. (As previously noted, Perdisci et al. did experiment with other classifiers, but we only compare our results with their decision tree classifier.)

Table 5 compares the results of our malware classifiers with those of Shafiq et al. (2009a). Shafiq et al. do not report the accuracy or ROC area for their detector, but instead report the "True Positive Rate" (TPR) –

$$\frac{\textit{detected malicious files}}{\textit{total malicious files}}$$

and the FPR.

Our non-packed detector is of comparable accuracy to that of Shafiq et al., but our packed detector is somewhat less accurate.

There are several possible reasons for this.

Firstly, the dataset we used for construction of the classifiers is much smaller than that used by Shafiq et al. Shafiq et al. used "about half a million malicious executables" and "several thousand benign executables".

Secondly, Shafiq et al. applied several preprocessing techniques to their data, which we avoided so as not to impair the comprehensibility of the decision trees produced.

Another possible explanation is that Shafiq et al. used a small number of packers to manually pack their benign files (they mention only four packers) while their malicious files were packed using a wide range of packers. These differences might have resulted in an apparent improvement in the accuracy of their packed malware classifier.

The time taken to construct our models was under one minute each, as was the time taken to test them. Shafiq et al. (2009a) do not report the time taken for model construction or testing, and model construction and testing time does not affect suitability for real-time deployment, so we have not included detailed analysis of these times here.

Task	Average time per file (s)
Extract packing detection features	0.546
Extract malware detection features	0.062
Total extraction time	0.608

Table 6 – Feature extraction time

Classification speed

Shafiq et al. (2009b) measured the average classification time per file of PE-Miner and of two commercial anti-virus programs. With a classification time of approximately 0.25 s per file, they concluded their technique was "real-time deployable".

Our classifier is likewise intended for real-time deployment. Therefore, the time taken to extract the required features from a file and classify it should be low – ideally less than a second.
The average time per file for feature extraction is shown in Table 6. (The time needed to run the classifier on the extracted features is extremely short – in the order of milliseconds per file – and does not add significantly to the processing time.)

Decision trees

Although the full decision trees produced by our classifier are too large to reproduce here in full, Figures 2-4 show the top levels of the decision trees produced, and we briefly discuss some of the features used.

Understanding the trees does require familiarity with the PE format, but it can be expected that security analysts will possess this familiarity.

In the packing classification tree (Figure 2) we can see that entropy of various sections is a major distinguishing feature between packed and non-packed files – the decision tree uses the entropy of the "code" and "data" sections of the file. The tree classifies a file as packed if its entropy is high (greater than 7.18 bits per symbol) and it has more than one section with read, write and execute flags set. As noted in the Method section, having all these flags set for a section is unusual for non-packed files.

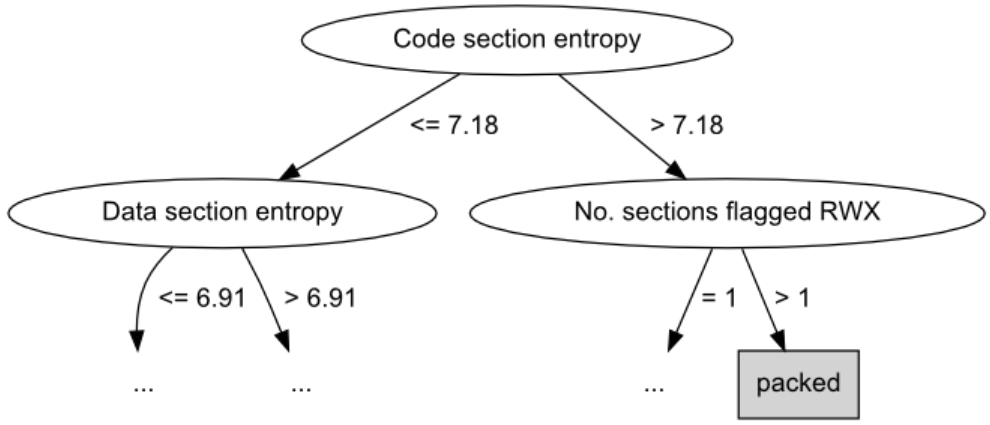

Figure 2 – Top levels of decision tree levels for packing classification

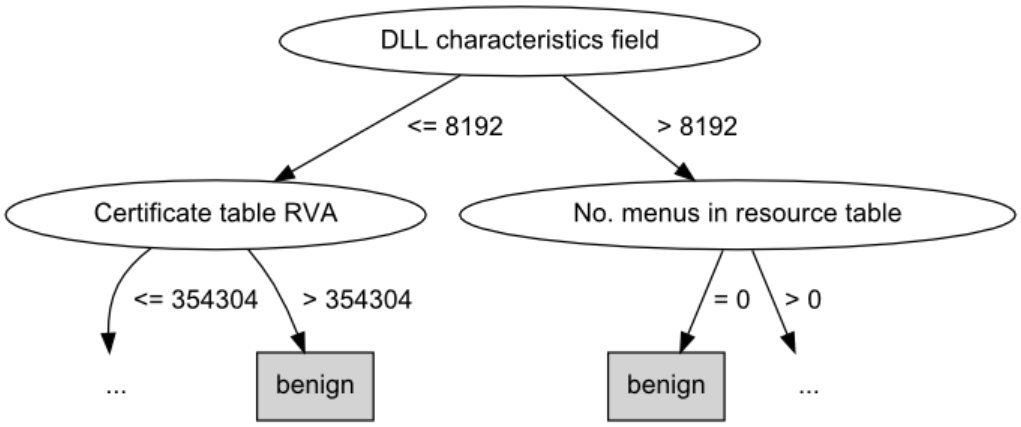

Figure 3 – Top levels of decision tree levels for non-packed malware classification

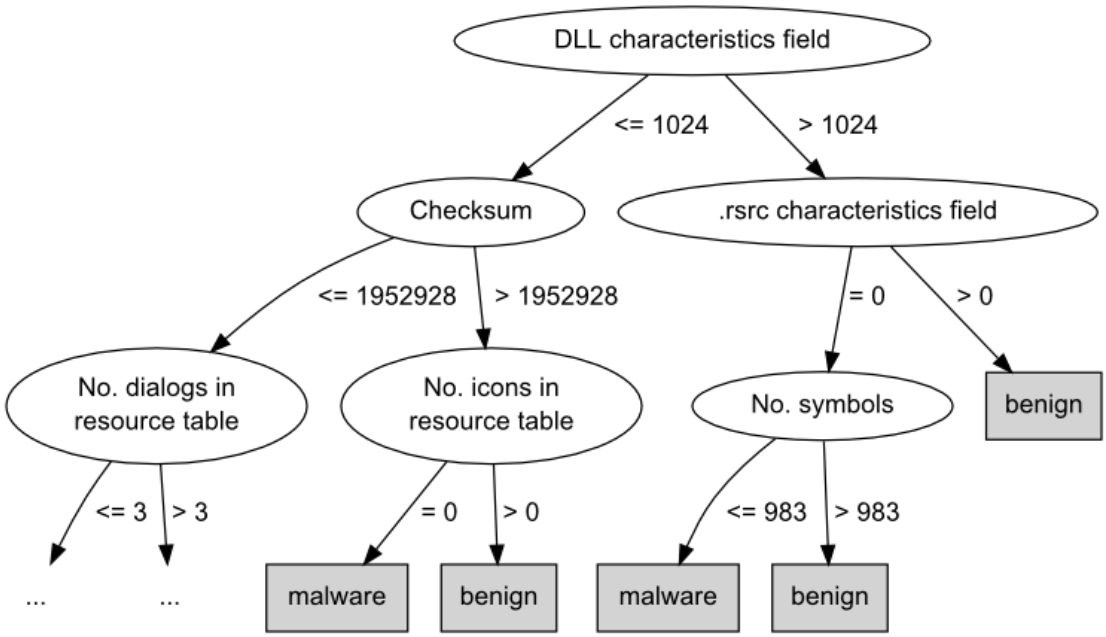

Figure 4 – Top levels of decision tree levels for packed malware classification

In the classification trees for distinguishing malicious from benign executables (Figures 3-4), the "DLL characteristics field" is a major distinguishing feature. This field consists of a set of binary flags providing information about DLL loading and compatibility. Flags in this field refer to properties such as whether code integrity checks are forced or whether a DLL can be relocated at load time. If the value is larger than 8192, this normally indicates that the file is a device driver. Typical users of a corporate computer network are unlikely to download device drivers, so this flag is unlikely to be set. Authors of malicious executables, however, might well use this portion of the file header for other purposes.

WEAKNESSES

The dataset we used for building and testing our classifiers is different to, and much smaller than, the one used by Shafiq et al. (2009a), so we cannot be sure our comparison is purely a comparison of technique. Collecting more PE files would at least rule out collection size as a confounding factor.

Furthermore, we excluded from our analysis files that could not be analysed with the Python `pefile` module, whereas Shafiq et al. mention no such exclusion. Experimentation with other PE analysis tools could identify a more robust tool that can process more of our dataset.

RELATED WORK

Non–signature-based malware detection techniques such as that of Shafiq et al. (2009a) fall into the general category of static heuristic analysis. Heuristic techniques identify features which may indicate malware, without definitively doing so.

Two approaches to heuristic analysis include the statistical analysis of *n*-grams (that is, all sub-sequences of *n* bytes), and data mining of function call information. Statistical *n*-gram analysis has been applied to malware detection by Perdisci et al. (2008b), Schultz et al. (2001) and Kolter and Maloof (2004). Data mining of function information was used in a technique proposed by Sami et al. (2010).

An alternative approach to heuristic techniques is *dynamic analysis*, where executables are actually run in a virtualized or sandboxed environment and scrutinized for signs of malicious behaviour. A recent survey of dynamic analysis techniques is provided by Egele et al. (2012).

CONCLUSION

Shafiq et al. (2009a) proposed a non–signature-based technique for detecting malicious PE files which had a high detection rate, low false positive rate and low classification time.

We have demonstrated that a modification of this technique can be used to produce classification decisions which are easily interpretable by network security analysts.

Our accuracy in classifying malicious, packed executables was lower than that reported by Shafiq et al. (2009a), suggesting further research is warranted. Increasing the size of the dataset used, and experimenting with the preprocessing techniques which Shafiq et al. applied to their extracted features, should help identify the exact reasons for the difference.

REFERENCES

Baranovich, A. (2012). VX Heavens. Retrieved from http://vx.netlux.org.

Bouckaert, R. R., Frank, E., Hall, M., Kirkby, R., Reutemann, P., Seewald, A., and Scuse, D. (2012). WEKA manual for version 3-6-7. Retrieved from http://ftp.jaist.ac.jp/pub/sourceforge/w/project/we/weka/documentation/3.6.x/WekaManual-3-6-7.pdf.

Carrera, E. (2012). *pefile*. Retrieved from http://code.google.com/p/pefile/.

CBS Interactive (2012). CNET Download.com. Retrieved from http://download.cnet.com/.

CDKiLLER and TippeX (2010). Protection ID. Retrieved from http://pid.gamecopyworld.com/.

Cover, T. M. and Thomas, J. A. (2006). *Elements of Information Theory*. Wiley Interscience, 2nd edition.

Egele, M., Scholte, T., Kirda, E., and Kruegel, C. (2012). A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)*, 44(2), 6.

Fawcett, T. (2004). ROC graphs: Notes and practical considerations for researchers. *Machine Learning*, 31(HPL-2003-4):1–38.

Geeknet, Inc (2012). SourceForge. Retrieved from http://sourceforge.net/.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18.

Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2nd edition.

Hastie, T., Tibshirani, R., and Friedman, J. H. (2001). *The elements of statistical learning: data mining, inference, and prediction*. Springer, New York.

Kolter, J. Z., and Maloof, M. A. (2004). Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on knowledge discovery and data mining (KDD '04)*, pages 470–478, Seattle, WA, USA. ACM Press.

Lyda, R. and Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware. *IEEE Security & Privacy*, 5(2):40–45.

Microsoft (2010). *Microsoft PE and COFF Specification*. Microsoft Corporation, Redmond, WA, revision 8.2. Retrieved from http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx.

Perdisci, R., Lanzi, A., and Lee, W. (2008a). Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters*, 29(14):1941–1946.

Perdisci, R., Lanzi, A., and Lee, W. (2008b). McBoost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC)*, pages 301–310, Anaheim, CA, USA. ACM Press.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Quist, D. (2012). Offensive computing. Retrieved from http://www.offensivecomputing.net/.

Raftopoulos, E. and Dimitropoulos, X. (2012). Shedding light on log correlation in network forensics analysis. In *Proceedings of the 9th international conference on detection of intrusions and malware, and vulnerability assessment (DIMVA)*, Lecture Notes in Computer Science, Heraklion, Crete, Greece. Springer-Verlag.

Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., and Hamze, A. (2010). Malware detection based on mining API calls. *ACM Symposium on Applied Computing (SAC '10)*, pages 1020–1025, Sierre, Switzerland. ACM Press.

Schultz, M. G., Eskin, E., Zadok, F., and Stolfo, S. J. (2001). Data mining methods for detection of new malicious executables. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 38–49, Los Alamitos, CA, USA. IEEE Press.

Shafiq, M., Tabish, S., and Farooq, M. (2009a). PE-Probe: Leveraging packer detection and structural information to detect malicious portable executables. In *Proceedings of Virus Bulletin Conference 2009 (VB2009)*.

Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009b). A framework for efficient mining of structural information to detect zero-day malicious portable executables. Technical Report TR-nexGINRC-2009-21, Next Generation Intelligent Networks Research Center, Islamabad, Pakistan. Retrieved from http://nexginrc.org/nexginrcAdmin/PublicationsFiles/tr21-zubair.pdf.

Shafiq, M. Z., Tabish, S. M., Mirza, F., and Farooq, M. (2009c). PE-Miner: Mining structural information to detect malicious executables in realtime. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID '09)*, pages 121–141, Berlin, Heidelberg. Springer-Verlag.

Witten, I. H. and Frank, E. (2005). *Data mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2nd edition.