Technical Report

Artificial Immune System based General Purpose Intrusion Detection System (AISGPIDS)

"A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables"

January, 2009

FAST National University of Computer & Emerging Sciences, Islamabad, Pakistan

A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables*

M. Zubair Shafiq[1], S. Momina Tabish[1,2], Fauzan Mirza[1,2], Muddassar Farooq[1]

[1]Next Generation Intelligent Networks Research Center (nexGIN RC)
FAST National University of Computer & Emerging Sciences (NUCES)
Islamabad, 44000, Pakistan.
{zubair.shafiq,momina.tabish,muddassar.farooq}@nexginrc.org

[2]Information Security Research Group (ISRG)
School of Electrical Engineering & Computer Sciences (SEECS)
National University of Sciences & Technology (NUST)
Islamabad, 44000, Pakistan.
fauzan@niit.edu.pk

January 2009

Abstract

In this paper, we present an accurate and realtime PE-Miner framework that automatically extracts distinguishing features from portable executables (PE) to detect zero-day malware without any a priori knowledge about them. The distinguishing features are extracted using the structural information standardized by the Microsoft Windows operating system for executables, DLLs and object files. We follow a threefold research methodology: (1) identify a set of structural features for PE files, which is computable in realtime, (2) use an efficient preprocessor for removing redundancy in the features' set, and (3) select an efficient data mining algorithm for final classification. The primary objective of PE-Miner is to distinguish between the benign and malicious executables; while its secondary task is to categorize the malicious executables as a function of their payload. We evaluated PE-Miner on two malware collections, VX Heavens and Malfease datasets that contain 11 and 5 thousand malicious PE files respectively. The results of our experiments show that PE-Miner achieves more than 99% detection rate with less than 0.5% false alarm rate for distinguishing between the benign and malicious executables. Furthermore, it achieves an average detection rate of 90% with an average false alarm rate of less than 5% for categorizing the malicious executables as a function of their payload. It is important to emphasize that PE-Miner has low processing overheads and takes only 0.244 seconds on the average to scan a given PE file. Finally, we evaluate the robustness and reliability of PE-Miner under several regression tests. Our results show that extracted features are "robust" to different packing techniques and PE-Miner is also resilient to majority of "crafty" evasion strategies.

*The research presented in this paper is conducted at nexGIN RC and is funded by National ICT R&D Fund, Ministry of Information Technology, Government of Pakistan under the grant # ICTRDF/AN/2007/37. The information, data, comments, and views detailed herein may not necessarily reflect the endorsements of views of the National ICT R&D Fund.

1 Introduction

A number of non-signature based malware detection techniques have been proposed recently. These techniques mostly use heuristic analysis, or behavior analysis or a combination of both to detect malware. Such techniques are being actively investigated because of their ability to detect zero-day malware without any a priori knowledge about them. Some of them have been integrated into the existing Commercial Off the Shelf Anti Virus (COTS AV) products, but have achieved only limited success [27], [28]. The most important shortcoming of these techniques is that they are not *realtime deployable*[1]. We, therefore, believe that the domain of *realtime deployable* non-signature based malware detection techniques is still open to novel research.

Non-signature based malware detection techniques are primarily criticized because of two inherent problems: (1) high *fp* rate, and (2) large processing overheads. Consequently, COTS AV products mostly utilize signature based detection schemes that provide low *fp* rate and have acceptable processing overheads. But it is a well-known fact that signature based malware detection schemes are unable to detect *zero-day* malware. We cite two reports to highlight the alarming rate at which new malware is proliferating. The first report is by Symantec that shows an increase of 468% in the number of malware from 2006 to 2007 [4]. The second report shows that the number of malware produced in 2007 alone was more than the total number of malware produced in the last 20 years [5]. These surveys suggest that signature based techniques cannot keep abreast with the security challenges of the new millennium because not only the size of the signatures' database will exponentially increase but also the time of matching signatures. These bottlenecks are even more relevant on resource constrained smart phones and mobile devices [32]. We, therefore, envision that in near future signature based malware detection schemes will not be able to meet the criterion of *realtime deployable* as well.

We argue that a malware detection scheme which is *realtime deployable* should use an intelligent but simple static analysis technique. In this paper we propose a framework, called *PE-Miner*, which uses novel *structural features* to efficiently detect malicious PE files. PE is a file format which is standardized by the Microsoft Windows operating systems for executables, dynamically linked libraries (DLL) and object files. We follow a threefold research methodology in our static analysis: (1) identify a set of structural features for PE files which is computable in realtime, (2) use an efficient preprocessor for removing redundancy in the features' set, and (3) select an efficient data mining algorithm for final classification. Consequently, our proposed framework consists of three modules: the feature extraction module, the feature selection/preprocessing module and the detection module.

We have evaluated our proposed detection framework on two independently collected malware datasets with different statistics. The first malware dataset is the VX Heavens Virus collection consisting of more than ten thousand malicious PE files [30]. The second malware dataset is the Malfease dataset, which contains more than five thousand malicious PE files [31]. We also collected more than one thousand benign PE files from our virology lab, which we use in conjunction with both malware datasets in our study. The results of our experiments show that our PE-miner framework achieves more than 99% detection rate with less than 0.5% false alarm rate for distinguishing between the benign and malicious executables. Further, our framework takes on average only 0.244 seconds to scan a given PE file. Therefore, we can conclude that our proposed PE-Miner is *realtime deployable*, and consequently it can be easily integrated into existing COTS AV products.

Remember that PE-Miner framework can also categorize the malicious executables as a function of their payload. This analysis is of great value for system administrators and malware forensic experts. PE-Miner achieves a detection rate of 90% with an average false alarm rate of less than 5% for categorizing

[1]We define a technique as *realtime deployable* if it has three properties: (1) a *tp* rate (or detection rate) of approximately 1, (2) an *fp* rate (or false alarm rate) of approximately 0, and (3) the file scanning time is comparable to existing COTS AV.

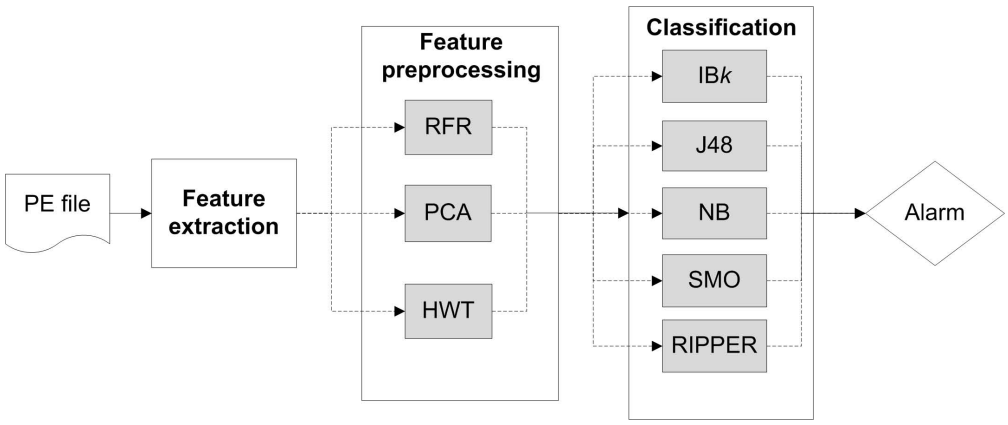


Figure 1: The architecture of our PE-Miner framework

the malicious executables as a function of their payload. We have also compared PE-Miner with other promising malware detection schemes proposed by Perdisci et al [1], Schultz et al [2] and Kolter et al [3]. These techniques use some variation of n-gram analysis for malware detection. PE-Miner provides better detection accuracy[2] with significantly smaller processing overheads compared with these approaches. We believe that the superior performance of PE-Miner is attributable to a rich set of novel PE format specific structural features, which provides relevant information for better detection accuracy [29]. In comparison, n-gram based techniques are more suitable for classification of loosely structured data; therefore, they fail to exploit format specific structural information of a PE file. As a result, they provide lower detection rates and have higher processing overheads as compared to PE-Miner. Our experiments also demonstrate that the detection mechanism of PE-Miner shows no bias towards packed/non-packed PE files. Finally, we investigate the robustness of PE-Miner against "crafty" attacks which are specifically designed to evade detection mechanism of PE-Miner. Our results show that PE-Miner is resilient to majority of such evasive attacks.

The remaining paper is organized as follows. In Section 2, we discuss the architecture of our proposed PE-Miner framework. In Section 3 we present description and statistics of the executable datasets used in our study. We provide a brief overview of related work in Section 4 before discussing the results of our experiments in Section 5. In Section 6, we carry out robustness analysis of PE-Miner. We finally conclude the paper in Section 7. Appendix A contains the listing and functionality of the DLLs used for feature extraction in our study. In Appendix 4, we present a detailed survey of the related work. In Appendix C, we present a review of the data mining algorithms used in our study.

2 PE-Miner Framework

In this section we discuss our proposed PE-Miner framework. We set the following strict requirements on our PE-Miner framework to ensure that our research is enacted with a product development cycle that has a short time-to-market:

- It must be a pure non-signature based framework with an ability to detect zero-day malicious PE files. Moreover, it should have the nice-to-have feature of categorizing malware as a function of their

[2]Throughout this text, the terms *detection accuracy* and *Area Under ROC Curve (AUC)* are used interchangeably. ROC curves are extensively used in machine learning and data mining to depict the tradeoff between the true positive rate and false positive rate of a classifier. The AUC ($0 \leq \text{AUC} \leq 1$) is used as a yardstick to determine the detection accuracy from ROC curve. Higher values of AUC mean high *tp* rate and low *fp* rate [49]. At AUC $= 1$, *tp* rate $= 1$ and *fp* rate $= 0$.

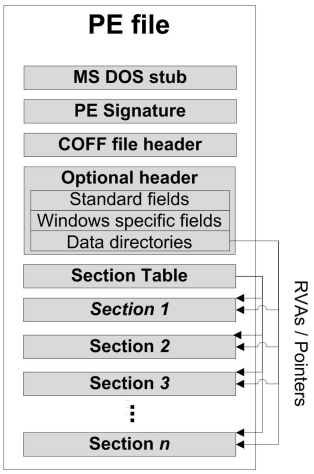


Figure 2: The PE file format

payload.

- It must be *realtime deployable*. To this end, we say that it should have more than 99% *tp* rate and less than 1% *fp* rate. We argue that it is still a challenge for non-signature based techniques to achieve these true and false positive rates. Moreover, its time to scan a PE file must be comparable to those of existing COTS AV products.

- Its design must be modular that allows for the plug-n-play design philosophy. This feature will be useful in customizing the detection framework to specific requirements, such as porting it to the file formats used by other operating systems.

We have evolved the final modular architecture of our PE-Miner framework in a questioned oriented engineering fashion. In our research, we systematically raised following relevant questions, analyzed their potential solutions and finally selected the best one through extensive empirical studies.

1. Which PE format specific features can be statically extracted from PE files to distinguish between benign and malicious files? Moreover, are the format specific features better than the existing n-grams or string-based features in terms of detection accuracy and efficiency?

2. Do we need to deploy preprocessors on the features' set? If yes then which preprocessors are best suited for the raw features' set?

3. Which are the best back-end classification algorithms in terms of detection accuracy and processing overheads.

Our PE-Miner framework consists of three main modules inline with the above-mentioned vision: (1) feature extraction, (2) feature preprocessing, and (3) classification (see Figure 1). We now discuss each module separately.

2.1 Feature Extraction

Let us revisit the PE file format [33] before we start discussing the structural features used in our features' set. A PE file consists of a PE file header, a section table (section headers) followed by the sections' data. The PE file header consists of a MS DOS stub, a PE file signature, a COFF (Common Object File Format)

Table 1: List of the features extracted from PE files

Feature Description	Type	Quantity
DLLs referred	binary	73
COFF file header	integer	7
Optional header – standard fields	integer	9
Optional header – Windows specific fields	integer	22
Optional header – data directories	integer	30
.text section – header fields	integer	9
.data section – header fields	integer	9
.rsrc section – header fields	integer	9
Resource directory table & resources	integer	21
Total		189

Table 2: Mean values of the extracted features. The bold values in every row highlight interesting outliers.

Dataset	VX Heavens									Malfease
Name of Feature	Benign	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	-
WSOCK32.DLL	0.037	**0.503**	0.038	**0.188**	**0.353**	**0.261**	**0.562**	**0.242**	0.053	0.065
WININET.DLL	0.073	**0.132**	0.009	0.013	0.04	**0.141**	0.004	**0.103**	0.019	0.086
Number of Symbols	**430.2**	2.0x10^6	14.7	59.4	25.8	3.5x10^6	38.8	4.1x10^6	1.0x10^6	2.7x10^7
Major Linker Version	**4.7**	14.4	11.2	14.1	12.1	12.3	18.7	12.2	19.3	6.5
Size of Initialized Data (x10^5)	**4.4**	1.1	0.5	0.4	0.8	0.7	0.4	0.4	0.1	0.6
Major Image Version	**163.1**	1.6	6.3	0.4	0.6	11.2	0.3	6.0	53.6	0.2
DLL Characteristics	**5.8x10^3**	0.0	0.0	0.0	0.0	24.9	0.0	3.1	230.8	18.7
Size Export Table (x10^2)	**13.7**	2.4	1.7	**14.1**	5.0	0.3	1.2	2.1	0.9	0.05
Size Import Table (x10^2)	5.8	**19.2**	6.1	7.9	**20.8**	7.1	**23.4**	**10.3**	6.2	4.7
Size Resource Table (x10^4)	**32.6**	5.5	1.5	1.4	6.2	1.0	2.6	2.2	0.5	5.9
Size Exception Table	12.0	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**3.5**
.data Size of Raw Data (x10^3)	**25.2**	8.4	5.6	6.3	6.0	7.9	6.1	5.5	6.7	22.1
Number of Cursors	**14.5**	6.4	6.7	7.4	6.1	5.9	5.8	6.0	3.0	6.8
Number of Bitmaps	**12.6**	1.2	0.0	1.0	0.6	0.7	1.2	1.4	2.4	0.5
Number of Icons	**17.6**	2.5	1.9	2.7	2.0	2.1	1.8	1.9	4.5	2.2
Number of Dialogs	**10.9**	3.2	1.5	3.2	1.5	2.0	1.9	1.7	2.2	2.3
Number of Group Cursors	**11.6**	6.0	6.6	7.2	5.8	5.8	5.4	5.7	2.7	6.7
Number of Group Icons	**4.1**	1.0	0.7	1.0	0.8	0.7	0.5	0.7	1.5	0.6

header and an optional header. It contains important information about a file such as the number of sections, the size of the stack and the heap, etc. The section table contains important information about the sections that follow it, such as their name, offset and size. These sections contain the actual data such as code, initialized data, exports, imports and resources [33], [34].

Figure 2 shows an overview of the PE file format [33], [34]. It is important to note that the section table contains Relative Virtual Addresses (RVAs) and the pointers to the start of every section. On the other hand, the data directories in an optional header contain references to various tables (such as import, export, resource, etc.) present in different sections. These references, if appropriately analyzed, can provide useful information.

We believe that this structural information about a PE file should be leveraged to extract features that have the potential to achieve our primary and secondary objectives. Using this principle, we statically extract a set of large number of features from a given PE file[3]. These features are summarized in Table 1. In the discussion below, we first intuitively argue about the features that have the potential to distinguish between benign and malicious files. We then show interesting observations derived from the executable datasets used in our empirical studies.

2.1.1 DLLs referred

The list of DLLs referred in an executable effectively provides an overview of its functionality. For example, if an executable calls `WINSOCK.DLL` or `WSOCK.DLL` then it is expected to perform network related

[3]A well-known Microsoft Visual C++ utility, called `dumpbin`, dumps the relevant information which is present inside a given PE file [36]. Another freely available utility, called `pedump`, also does the required task [37].

activities. However, there can be exceptions to this assumption as well. In [2], Schultz et al have used the conjunction of DLL names, with a similar functionality, as binary features. The results of their experiments show that this feature helps to attain reasonable detection accuracy. However, our pilot experimental studies have revealed that using them as individual binary features can reveal more information, and hence can be more helpful in detecting malicious PE files. In this study, we have used 73 core functionality DLLs as features. Their list and functionality is detailed in Appendix A. Table 2 shows the mean feature values for the two DLLs[4]. Interestingly, `WSOCK32.DLL` and `WININET.DLL` are used by a majority of backdoors, nukers, flooders, hacktools, worms and trojans to access the resources on the network and the Internet. Therefore, the applications misusing these DLLs might provide a strong indication of a possible covert network activity.

2.1.2 COFF file header

The COFF file header contains important information such as the type of the machine for which the file is intended, the nature of the file (DLL, EXE, or OBJ etc.), the number of sections and the number of symbols. It is interesting to note in Table 2 that a reasonable number of symbols are present in benign executables. The malicious executables, however, either contain too many or too few symbols.

2.1.3 Optional header: standard fields

The interesting information in the standard fields of the optional header include the linker version used to create an executable, the size of the code, the size of the initialized data, the size of the uninitialized data and the address of the entry point. Table 2 shows that the values of major linker version and the size of the initialized data have a significant difference in the benign and malicious executables. The size of the initialized data in benign executables is usually significantly higher compared to those of the malicious executables.

2.1.4 Optional header: Windows specific fields

The Windows specific fields of the optional header include information about the operating system version, the image version, the checksum, the size of the stack and the heap. It can be seen in Table 2 that the values of fields such as the major image version and the DLL characteristics are usually set to zero in the malicious executables. In comparison, their values are significantly higher in the benign executables.

2.1.5 Optional header: data directories

The data directories of the optional header provide pointers to the actual data present in the sections following it. It includes the information about export, import, resource, exception, debug, certificate and base relocation tables. Therefore, it effectively provides a summary of the contents of an executable. Table 2 highlights that the size of the export table is higher for the benign executables and nukers as compared to those of other malicious executables. Another interesting observation in Table 2 is that the backdoors, flooders, worms and trojans mostly have a bigger import table size. It can be intuitively argued that they usually import network functionalities which increases the size of their import table. The size of the resource table, on the other hand, is higher for the benign executables as compared to those of the malicious executables. The exception table is mostly absent in the malicious executables.

[4]The details of the datasets and their categorization are available in Section 3.

2.1.6 Section headers

The section headers provide important characteristics of a section such as its address, size, number of relocations and line numbers. In this study, we have only considered text, data and resource sections because they are commonly present in the executables. Note that the size of the data section (if present) is relatively higher for the benign executables.

2.1.7 Resource directory table & resources

The resource directory table provides an overview of the resources that are present in the resource section of an executable file. We consider the actual count of various types of resources that are present in the resource section of an executable file. The typical examples of resources include cursors, bitmaps, icons, menus, dialogs, fonts, group cursors and user defined resources. Intuitively and as shown in Table 2, the number of these resources is relatively higher for the benign executables.

2.2 Feature Selection/Preprocessing

We have now identified our features' set that consists of a number of statically computable features – 189 to be precise – based on the structural information of the PE files. It is possible that some of the features might not convey useful information in a particular scenario, therefore, it makes sense to remove or combine them with other similar features to reduce the dimensionality of our input feature space. Moreover, this preprocessing on the raw extracted features' set also reduces the processing overheads in training and testing of classifiers, and can possibly also improve the detection accuracy of classifiers. In this study, we have used three well-known features' selection/preprocessing filters. We provide their short descriptions in the following text. More details can be found in [21].

Redundant Feature Removal (RFR). We apply this filter to remove those features that do not vary at all or show significantly large variation i.e. they have approximately uniform-random behavior. Consequently, this filter removes all features that have either constant values or show a variance above a threshold or both.

Principal Component Analysis (PCA). The Principal Component Analysis (PCA) is a well-known filter for dimensionality reduction. It is especially useful when the input data has high dimensionality – sometimes referred to as *curse of dimensionality*. This dimensionality reduction can possibly improve the quality of an analysis on a given data if the dataset consists of highly correlated or redundant features. However, this dimensionality reduction may result in information loss (i.e. reduction in data variance) as well. One has to carefully choose the appropriate balance for this tradeoff. We apply PCA filter to remove/combine correlated features for dimensionality reduction.

Haar Wavelet Transform (HWT). The principle of this technique is that the most relevant information is stored with the highest coefficients at each order of a transform. The lower order coefficients can be ignored to get only the most relevant information. The wavelet transform has also been used for dimensionality reduction. The wavelet transform technique has been extensively used in the image compression but is never evaluated in the malware detection domain. The Haar wavelet is one of the simplest wavelets and is known to provide reasonable accuracy. The application of Haar wavelet transform requires the data to be normalized. Therefore, we have passed the data through *normalize* filter before applying HWT.

2.3 Classification

Once the dimensionality of the input features' set is reduced by applying one of the above-mentioned preprocessing filters, it is given as an input to the well-known data mining algorithms for classification. Most of these algorithms require a training and testing phase. In this study we have used five classifiers: (1) instance based learner (IBk), (2) decision tree (J48), (3) Naïve Bayes, (4) inductive rule learner (RIPPER), and (5)

Table 3: Statistics of the data used in this study.

Dataset	VX Heavens									Malfease
Title of Statistic	Benign	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	-
Quantity	1,447	3,455	367	267	358	243	1,483	3,114	1,052	5,586
Average Size (KB)	1,263	270	234	176	298	156	72	136	50	285
Minimum Size (KB)	4	1	4	3	6	4	2	1	2	1
Maximum Size (KB)	104,588	9,277	5,832	1,301	14,692	1,924	2,733	4,014	1,332	5,746
Packed using UPX	17	786	79	15	32	43	353	622	48	470
Packed using ASPack	2	432	21	16	25	15	66	371	10	187
Packed using Other	372	325	47	31	58	38	471	170	71	1,909
Compiled using Borland C/C++	15	56	8	15	10	6	13	63	18	11
Compiled using Borland Delphi	13	589	13	65	64	8	76	379	71	342
Compiled using Visual Basic	4	719	106	39	126	38	210	674	119	809
Compiled using Visual C++	526	333	19	51	29	59	89	619	96	351
Compiled using Visual C#	56	0	0	0	1	0	5	1	6	1
Compiled using Other	9	49	9	2	3	2	4	15	7	5
Total Non-packed (%)	43.1	50.5	42.2	64.4	65.1	46.5	26.8	56.2	30.1	27.2
Total Packed (%)	27.0	44.7	40.1	23.2	32.1	39.5	60.0	37.4	12.3	46.6
Not Found (%)	29.9	4.8	17.7	12.4	2.8	14.0	13.2	6.4	57.6	26.2

support vector machines using sequential minimal optimization (SMO). An interested reader can find their details in Appendix C.

3 Datasets

In this section, we present an overview of the datasets used in our study. We have collected $1,447$ benign PE files from the local network of our virology lab. The collection contains executables such as Packet CAPture (PCAP) file parsers compiled by MS Visual Studio 6.0, compressed installation executables and MS Windows XP/Vista applications' executables. The diversity of the benign files is also evident from their sizes, which range from a minimum of 4 KB to a maximum of $104,588$ KB (see Table 3).

Moreover, we have used two malware collections in our study. First is the VX Heavens Virus Collection, which is *labeled* and is publicly available for free download [30]. We only consider PE executables to maintain focus. Our filtered dataset contains $10,339$ malicious PE files. The second dataset is the Malfease malware dataset [31], which consists of $5,586$ *unlabeled* malicious PE files.

In order to conduct a comprehensive study, we further categorize the malicious PE files as a function of their payload[5]. The malicious executables are subdivided into eight major categories such as *virus*, *trojan*, *worm*, etc. Moreover, we have combined some categories that have similar functionality. For example, we have combined *constructor* and *virtool* to create a single *constructor + virtool* category. This unification increases the number of malware samples per category. It will be helpful later when we categorize the malicious executables as a function of their payload. We now provide a brief introduction of each malware category, used in our study, to make the paper self contained [47].

Backdoor + Sniffer. A backdoor is a program which allows bypassing of standard authentication methods of an operating system. As a result, remote access to computer systems is possible without explicit consent of the users. Information logging and sniffing activities are possible using the gained remote access.

Constructor + Virtool. This category of malware mostly includes toolkits for automatically creating new malware by varying a given set of input parameters. Virtool and constructor categories are combined because of their similar functionality.

DoS + Nuker. Both DoS and nuker based malware allow an attacker to launch malicious activities at a victim's computer system that can possibly result in a denial of service attack. These activities can result in slow down, restart, crash or shutdown of a computer system.

[5]Since the Malfease malware collection is unlabeled, therefore, it is not possible to divide it into different malware categories.

Email- + IM- + SMS Flooder. The malware in this category initiate unwanted information floods such as email, instant messaging and SMS floods.

Exploit + Hacktool. The malware in this category exploit vulnerabilities in a system's implementation which most commonly results in buffer overflows.

Email- + IM- + IRC- + Net Worm. The malware in this category spreads through instant messaging networks, IRC networks and port scanning.

Trojan. A trojan is a broad term that refers to stand alone programs which appear to perform a legitimate function but covertly do possibly harmful activities such as providing remote access, data destruction and corruption.

Virus. A virus is a program that can replicate and attach itself with other benign programs. It is probably the most well-known type of malware.

Table 3 provides the detailed statistics of the malware used in our study. A careful reader can rightly conclude that the average size of the malicious executables is smaller than that of the benign executables. Further, some executables used in our study are encrypted and/or compressed (packed). The detailed statistics about packing are also tabulated in Table 3. We use PEiD [40] and Protection ID for detecting packed executables [41][6].

Our analysis shows that VX Heavens Virus collection contains 40.1% packed and 47.2% non-packed PE files. However, approximately 12.7% malicious PE files cannot be classified as either packed or non-packed by PEiD and Protection ID. The Malfease collection contains 46.6% packed and 27.2% non-packed malicious PE files. Similarly, 26.2% malicious PE files cannot be classified as packed or non-packed. We can, therefore, say that packed/non-packed malware distribution in the VX Heavens virus collection is relatively more balanced than the Malfease dataset. In our collection of benign files, 43.1% are packed and 27.0% are non-packed PE files respectively. Similarly, 29.9% benign files are not detected by PEiD and Protection ID. An interesting observation is that the benign PE files are mostly packed using nonstandard and custom developed packers. We speculate that a significant portion of the packed executables are not classified as packed because the signatures of their respective packers are not present in the database of PEiD or Protection ID. *Note that we do not manually unpack any PE file prior to the processing of our PE-Miner.*

4 Related Work

We now briefly describe the most relevant non-signature based malware detection techniques. These techniques are proposed by Perdisci et al [1], Schultz et al [2] and Kolter et al [3]. We briefly summarize their working principles in the following paragraphs but an interested reader can find their detailed description in Appendix B.

In [1], the authors proposed McBoost that uses two classifiers, C1 and C2, for classification of non-packed and packed PE files respectively. A custom developed unpacker is used to extract the hidden code from packed PE files and the output of the unpacker is given as an input to the C2 classifier. Unfortunately, we could not obtain its source code or binary due to licensing related problems. Further its implementation is not within the scope of our current work. Consequently, we only evaluate the C1 module of McBoost which works only for non-packed PE files. We, therefore, acknowledge that our McBoost results should be considered only preliminary.

In [2], Schultz et al have proposed three independent techniques for detecting malicious PE files. The first technqiue, uses the information about DLLs, function calls and their invocation counts. However, the authors did not provide enough information about the used DLLs and function names; therefore, it is not

[6]We acknowledge the fact that PEiD and Protection ID are signature based packer detectors and can have significant false negatives.

Table 4: AUCs for detecting the malicious executables. The bold entries in each column represent the best results.

Dataset	VX Heavens									Malfease
Malware	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	Average	-
PE-Miner — RFR										
IBK	0.992	0.996	0.995	0.994	0.998	0.979	0.984	0.994	**0.992**	0.986
J48	0.993	**0.998**	0.987	0.993	0.999	0.979	**0.992**	0.993	**0.992**	0.979
NB	0.971	0.978	0.966	0.973	0.987	0.972	0.974	0.986	0.976	0.976
RIPPER	**0.996**	0.996	0.977	0.981	0.999	**0.988**	0.988	0.996	0.990	0.985
SMO	0.991	0.990	0.991	0.993	0.997	0.975	0.978	0.992	0.988	0.963
PE-Miner — PCA										
IBK	0.989	0.996	0.994	0.995	0.998	0.976	0.984	0.993	0.991	0.984
J48	0.980	0.966	0.929	0.960	0.987	0.936	0.951	0.985	0.962	0.945
NB	0.961	0.990	0.993	0.996	0.996	0.964	0.956	0.990	0.981	0.898
RIPPER	0.982	0.978	**0.996**	0.974	0.977	0.949	0.968	0.987	0.976	0.952
SMO	0.990	0.992	0.989	0.995	0.995	0.958	0.965	0.992	0.985	0.954
PE-Miner — HWT										
IBK	0.991	0.996	**0.996**	**0.998**	**1.000**	0.978	0.985	0.995	**0.992**	0.986
J48	0.995	0.997	0.993	0.988	0.997	0.978	0.991	0.999	**0.992**	0.977
NB	0.989	0.982	0.983	0.987	0.990	0.978	0.972	0.990	0.984	0.960
RIPPER	0.994	0.997	0.982	0.990	0.997	0.983	0.990	**1.000**	**0.992**	**0.987**
SMO	0.990	0.995	0.991	0.996	**1.000**	0.972	0.973	0.994	0.989	0.964
McBoost — C1 only										
IBK	0.941	0.935	0.875	0.960	0.832	0.938	0.930	0.914	0.916	0.949
J48	0.866	0.895	0.809	0.893	0.731	0.906	0.902	0.882	0.860	0.860
NB	0.831	0.924	0.723	0.889	0.795	0.873	0.886	0.844	0.846	0.817
RIPPER	0.833	0.888	0.744	0.918	0.660	0.866	0.838	0.844	0.824	0.860
SMO	0.802	0.887	0.759	0.910	0.678	0.854	0.805	0.827	0.815	0.835
Strings										
IBK	0.949	0.860	0.902	0.980	0.925	0.928	0.863	0.952	0.920	0.944
J48	0.913	0.834	0.862	0.695	0.871	0.908	0.836	0.938	0.857	0.929
NB	0.920	0.830	0.882	0.726	0.886	0.901	0.828	0.905	0.860	0.930
RIPPER	0.843	0.797	0.714	0.578	0.712	0.892	0.743	0.929	0.776	0.927
SMO	0.855	0.817	0.705	0.775	0.583	0.871	0.756	0.883	0.781	0.933
KM										
IBK	0.984	0.934	0.983	0.971	0.983	0.987	0.979	0.986	0.976	0.980
J48	0.953	0.940	0.916	0.907	0.916	0.957	0.951	0.953	0.937	0.952
NB	0.943	0.959	0.961	0.952	0.961	0.968	0.954	0.954	0.957	0.961
RIPPER	0.951	0.944	0.924	0.921	0.924	0.964	0.948	0.948	0.941	0.971
SMO	0.949	0.946	0.952	0.927	0.952	0.961	0.940	0.938	0.946	0.960

possible for us to implement it. But we have implemented the second approach (titled *strings*) which uses strings as binary features i.e. present or absent. The third technique uses two byte words as binary features. This technique is later improved in a seminal work by Kolter et al [3] which uses 4-grams as binary features. Therefore, we include the technique of Kolter et al (titled *KM*) in our comparative evaluation.

5 Experimental Results

Recall that we set two objectives for doing a comparative evaluation of different techniques: (1) the primary objective is to distinguish between benign and malicious PE files, and (2) the secondary objective is to categorize the malicious executables as a function of their payload. We have compared our PE-Miner framework with recently proposed promising techniques by Perdisci et al [1], Schultz et al [2] and Kolter et al [3].

We have used the standard 10 fold cross-validation process in our experiments: the dataset is randomly divided into 10 smaller subsets, where 9 subsets are used for training and 1 subset is used for testing. The process is repeated 10 times for every combination. This methodology helps in evaluating the robustness of a given approach to detect malicious PE files that contain malware without any a priori information. The ROC curves are generated by varying the threshold on output class probability [48], [49]. The AUC is used as a yardstick to determine the detection accuracy of each approach. We have done the experiments on an Intel Pentium Core 2 Duo 2.19 GHz processor with 2 GB RAM. The Microsoft Windows XP SP2 is installed on this machine. We now separately report the detection accuracies of different approaches for our

Table 5: The processing overheads (in seconds/file) of different feature selection, extraction and preprocessing schemes.

	PE-Miner			McBoost	Strings	KM
	(RFR)	(PCA)	(HWT)			
Selection	-	-	-	2.839	5.289	31.499
Extraction	0.228	0.228	0.228	0.198	0.130	0.220
Preprocessing	0.007	0.009	0.012	-	-	-
Total	**0.235**	0.237	0.240	3.037	5.419	31.719

primary and secondary objectives.

5.1 Primary Objective: Malicious PE File Detection

In our first experimental study, we attempt to distinguish between benign and malicious PE files. To get better insights, we have done independent experiments with benign and each of the eight types of the malicious executables. The five data mining algorithms, namely IBk, J48, NB, RIPPER and SMO, are deployed on top of each approach (namely PE-Miner with RFR, PE-Miner with PCA, PE-Miner with HWT, McBoost (C1 only) by Perdisci et al [1], strings approach by Schultz et al [2] and KM by Kolter et al [3]). This results in a total of 270 experimental runs each with 10-fold cross validation. We tabulate our results for this study in Table 4 and now answer different questions that we raised in Section 2 in a chronological fashion.

5.1.1 Which features' set is the best?

Table 4 tabulates the AUCs for PE-Miner using three different preprocessing filters (RFR, PCA and HWT), McBoost, strings and KM [3]. A macro level scan through the table clearly shows the supremacy of PE-Miner based approaches with AUCs more than 0.99 for most of the malware types and even approaching 1.00 for some malware types. For PE-Miner, RFR and HWT preprocessing lead to the best average results with more than 0.99 AUC.

The strings approach gives the worst detection accuracy. The KM approach is better than the strings approach but inferior to our PE-Miner. This is expected because the string features are not stable as compiling a given piece of code by using different compilers leads to different sets of strings. Our analysis shows that KM approach is more resilient to variation in the string sets because it uses a combination of string and non-string features. The results obtained for KM approach are also consistent with the results reported in [3]. Its average AUC is about 0.95. The C1 module of McBoost also provides relatively inferior detection accuracies which are as low as 0.66 for exploit+hacktool category. It is important to note that the C1 module of McBoost is functionally similar to the techniques proposed by Schultz et al and Kolter et al. The only significant difference is that C1 operates only on the code sections of the non-packed PE files whereas the other techniques operate on complete files.

It is important to emphasize that both strings and KM approaches incur large overheads in the feature selection process (see Table 5[7]). Kolter et al have confirmed that their implementation of information gain calculation for feature selection took almost a day for every single run. To make our implementation of n-grams more efficient, we use `hash_map` STL containers in the Visual C++ [35]. Our experiments show that the feature selection process in KM still takes more than 31 seconds per file even with our optimized implementation. The optimized strings approach takes, on the average, more than 5 seconds per file for feature selection. The optimized McBoost (C1 only) approach takes an average of more than 2 seconds per file for feature selection[8]. These approaches have processing overheads because the time to calculate

[7]The results in Table 5 are averaged over 100 runs.

[8]Note that the complete McBoost system also uses unpacker for extraction of hidden code. This process is time consuming according to the authors in [1].

Table 6: The processing overheads (in seconds/file) of different features and classification algorithms.

	IBK	J48	NB	RIPPER	SMO	IBK	J48	NB	RIPPER	SMO
	Training					**Testing**				
PE-Miner (RFR)	-	0.008	0.001	0.269	0.199	0.032	0.001	0.002	0.002	0.002
PE-Miner (PCA)	-	0.007	0.001	0.264	0.179	0.035	0.001	0.001	0.001	0.002
PE-Miner (HWT)	-	0.007	0.001	0.252	0.147	0.032	0.001	0.002	0.001	0.002
McBoost	-	0.021	0.004	1.305	1.122	0.218	0.010	0.007	0.005	0.022
Strings	-	0.009	0.002	0.799	0.838	0.163	0.003	0.003	0.002	0.003
KM	-	0.024	0.004	1.510	1.018	0.254	0.018	0.007	0.005	0.020

information gain increases exponentially with the number of unique n-grams (or strings). On the other hand, PE-Miner does not suffer from such serious bottlenecks. The application of RFR, PCA or HWT filters takes only about a hundredth of a second.

5.1.2 Which classification algorithm is the best?

We can conclude from Table 4 that J48 outperforms the rest of the data mining classifiers in terms of the detection accuracy in most of the cases. Moreover, Table 6 shows that J48 has one of the smallest processing overheads both in training and testing. RIPPER and IBk closely follow the detection accuracy of J48. However, they are infeasible for realtime deployment because of the high processing overheads in the training and the testing phases respectively. The processing overheads of training RIPPER are the highest among all classifiers. In comparison, IBk does not require a training phase but its processing overheads in the testing phase are the highest. Further, Naïve Bayes gives the worst detection accuracy because it assumes independence among input features. Intuitively speaking, this assumption does not hold for all features' sets used in our study. Note that Naïve Bayes has very small learning and testing overheads (see Table 6[9]).

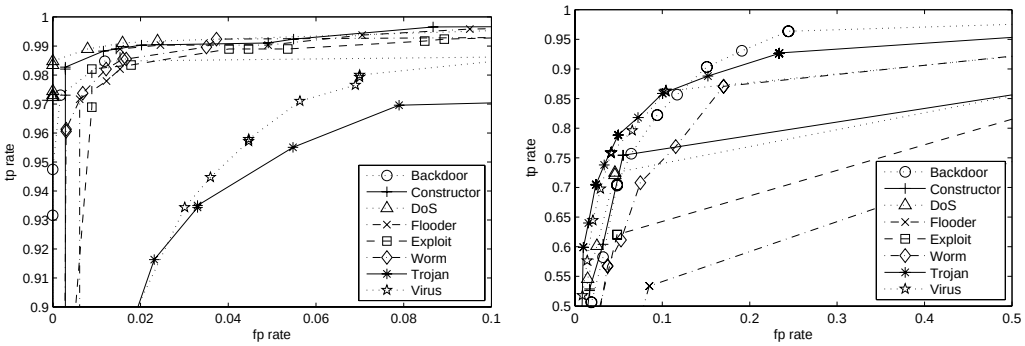
5.1.3 Which malware category is the most challenging to detect?

An overview of Table 4 suggests that the most challenging malware categories are worms and trojans. The average AUC values of the compared techniques for worms and trojans are approximately 0.95. The poor detection accuracy is attributed to the fact that the trojans are inherently designed to appear similar to the benign executables. Therefore, it is a difficult challenge to distinguish between trojans and benign PE files. Our PE-Miner still achieves on the average 0.98 AUC for worms and trojans which is quite reasonable. Figure 3(a) shows that for other malware categories, PE-Miner (with RFR preprocessor) has AUCs more than 0.99.

5.2 Secondary Objective: Malicious Executable Detection as a Function of Payload

In our second comparative study, we attempt to categorize the malicious executables as a function of their payload. This means that we want to know the category of a malware in a given malicious PE file. Recall from Section 3 that we have eight different categories of malware. This is in fact a multi-class classification problem and is therefore significantly more challenging than the primary objective of just categorizing the benign and malicious PE files. The secondary objective is not a necessity from the point-of-view of use in a COTS AV product. However, it is definitely a *nice-to-have* feature for system administrators and malware forensic experts. We follow the same 'one-vs-all' classification approach as is used by the authors in [3]. We have done experiments with the VX Heavens dataset in which each malware category is labeled as a separate class. Table 7 tabulates the results from our second study. We again answer the same questions raised in Section 2 in a chronological fashion for our second study.

[9]The results in Table 6 are averaged over 100 runs.



(a) detecting malicious executables

(b) detecting malicious executables as a function of their payload

Figure 3: The magnified ROC plots, for primary and secondary objectives, using PE-Miner utilizing J48 preprocessed with RFR filter. The results are shown for VX Heavens dataset.

Table 7: AUCs for detecting the malicious executables as a function of their payload (only on VX Heavens dataset). The bold entries in each column represent the best results.

Classifier	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	Average
PE-Miner — RFR									
IBK	0.936	0.853	0.803	0.729	0.849	0.931	0.908	0.890	0.862
J48	**0.972**	**0.946**	0.920	0.840	0.897	**0.969**	**0.957**	0.936	**0.929**
NB	0.747	0.774	0.756	0.653	0.831	0.795	0.837	0.738	0.766
RIPPER	0.937	0.918	**0.926**	0.806	0.884	0.959	0.943	**0.938**	0.914
SMO	0.910	0.928	0.844	0.749	0.936	0.891	0.912	0.892	0.883
PE-Miner — PCA									
IBK	0.933	0.862	0.794	0.735	0.854	0.929	0.910	0.885	0.863
J48	0.862	0.752	0.707	0.665	0.723	0.861	0.833	0.805	0.776
NB	0.816	0.829	0.814	0.784	0.842	0.850	0.802	0.813	0.819
RIPPER	0.847	0.802	0.763	0.716	0.797	0.875	0.824	0.819	0.805
SMO	0.899	0.932	0.822	0.735	0.934	0.873	0.897	0.879	0.871
PE-Miner — HWT									
IBK	0.934	0.856	0.794	**0.886**	0.854	0.932	0.912	0.886	0.882
J48	0.866	0.763	0.703	0.812	0.767	0.865	0.826	0.812	0.802
NB	0.824	0.833	0.816	0.790	0.862	0.858	0.811	0.814	0.826
RIPPER	0.942	0.925	0.908	0.778	0.876	0.958	0.947	**0.938**	0.909
SMO	0.908	0.924	0.849	0.748	**0.949**	0.889	0.913	0.888	0.884
McBoost — C1 only									
IBK	0.795	0.749	0.747	0.650	0.636	0.777	0.696	0.771	0.728
J48	0.739	0.614	0.669	0.597	0.573	0.716	0.644	0.703	0.657
NB	0.714	0.660	0.673	0.609	0.577	0.656	0.622	0.693	0.651
RIPPER	0.644	0.581	0.622	0.579	0.537	0.625	0.600	0.654	0.605
SMO	0.735	0.709	0.726	0.672	0.617	0.734	0.647	0.730	0.696
Strings									
IBK	0.741	0.666	0.780	0.673	0.665	0.690	0.659	0.740	0.702
J48	0.730	0.670	0.718	0.600	0.657	0.674	0.656	0.739	0.681
NB	0.654	0.659	0.770	0.633	0.657	0.592	0.572	0.722	0.657
RIPPER	0.584	0.542	0.621	0.568	0.579	0.594	0.563	0.574	0.578
SMO	0.717	0.654	0.773	0.677	0.692	0.656	0.636	0.721	0.691
KM									
IBK	0.853	0.769	0.800	0.741	0.686	0.851	0.793	0.907	0.800
J48	0.771	0.722	0.727	0.672	0.625	0.731	0.689	0.849	0.723
NB	0.718	0.660	0.644	0.684	0.699	0.715	0.627	0.814	0.695
RIPPER	0.715	0.700	0.705	0.663	0.670	0.678	0.704	0.812	0.706
SMO	-	-	-	-	-	-	-	-	-

5.2.1 Which features' set is the best?

The results of the payload based malicious executable classification follow similar patterns as are observed in the first experimental study. The best detection accuracy is again achieved with the PE format specific structural feature extraction algorithm. PE-Miner with RFR and HWT preprocessing on the average has 0.93 AUCs. In comparison, the detection accuracies of McBoost, strings and KM approaches are significantly

Table 8: Realtime deployable analysis of the best techniques

Technique	Classifier	AUC	Scan Time (sec/file)	Is Realtime Deployable?
PE-Miner (RFR)	J48	0.991	0.244	**"Yes"**
McBoost	IBk	0.926	3.255	No
Strings	IBk	0.927	5.582	No
KM	IBk	0.977	31.973	No
AVG Free 8.0 **[43]**	-	-	0.159	-
Panda 7.01 **[44]**	-	-	0.131	-

deteriorated and the average AUCs are in the range of $0.60 - 0.70$. This further strengthens our thesis that the format specific structural feature extracting scheme can achieve both primary and secondary objectives with the highest detection accuracies and the lowest processing overheads.

5.2.2 Which classification algorithm is the best?

The relative performance of the data mining classification algorithms is similar to the previous study. Here again, PE-Miner using J48 performs significantly better than the other classifiers. We have observed in these these experiments that the processing overheads of all approaches have significantly increased for the payload based malicious executable classification. In fact, the experiments of KM approach with SMO classifier do not finish despite running for several days and hence no values are reported for it in Table 7. Our observation is that the J48 gave the best detection accuracy with relatively smaller processing overheads in the training and testing phases.

5.2.3 Which malware category is the most challenging to categorize?

It is interesting to note in Table 7 that the worm and trojan categories have the best AUCs. This trend is opposite to the pattern observed in the previous study where these categories of malware were the most difficult to detect. Here the flooder and nuker categories have the worst AUCs. Our analysis reveals that several flooder samples are wrongly classified as nuker and vice-versa leading to worst accuracies. This is because both of them have similar functional behavior (refer to Section 3). This trend can also be observed in Figure 3(b).

5.3 Miscellaneous Discussions

We conclude our comparative study with an answer to an important issue: *which of the compared techniques meet the criteria of being realtime deployable?* (see Section 2). We tabulate the AUC and the scan time of the best techniques in Table 8. Moreover, we also show the scan time of two well-known COTS AV products for doing the *realtime deployable* analysis of different non-signature based techniques. It is clear that PE-Miner (RFR) with J48 classifier is the only non-signature based technique that satisfies the criteria of being *realtime deployable*. One might argue that PE-Miner framework provides only a small improvement in detection accuracy over the KM approach. *But then KM has the worst scan time of* 31.97 *seconds per file (see Table 8)*. It is very important to interpret the results in Table 8 from a security expert's perspective. For example, if a malware detector scans ten thousand files with an AUC of 0.97, it will not detect approximately 300 malicious files. In comparison, a detector with an AUC of 0.99 will miss only 100 files, which is a 66.6% improvement in the number of missed files [20]. We therefore argue that from a security expert's perspective, even a small improvement in the detection accuracy is significant in the limiting case when detection accuracy approaches to 1.00.

An additional benefit of PE-Miner is that it provides insights about the learning models of different classifiers that can be of great value to malware forensic experts. We show partial subtrees of J48 for categorizing

Table 9: Portions of the developed decision trees

```
NumMessageTable <= 0
|    SizeLoadConfigTable <= 0
|    |    TimeDateStamp <= 1000000000
|    |     |      NumCursor <= 1
|    |     |       |    NumAccelerators <= 0
|    |     |       |     |      NumBitmap <= 0: malicious
|    |     |       |     |      NumBitmap > 0: benign
|    |     |       |    NumAccelerators > 0:malicious
|    |     |      NumCursor > 1:malicious
```

(a) between benign and backdoor+sniffer

```
NumMessageTable <= 0
|    NumBitmap <= 0
|    |    NumAccelerators <= 1: malicious
|    |    NumAccelerators > 1
|    |     |     NumUserDefined <= 0: malicious
|    |     |     NumUserDefined > 0: benign
|    NumBitmap > 0
|    |    NumUserDefined <= 0: malicious
|    |    NumUserDefined > 0: benign
NumMessageTable > 0: benign
```

(b) between benign and constructor+virtool

```
NumDialog > 8
|    NumIcon <= 2
|    |    NumMenu <= 1: malicious
|    |    NumMenu > 1: benign
|    NumIcon > 2
|    |    WSOCK32.DLL <= 0: benign
|    |    |    NumIcon <= 8: benign
|    |    |    NumIcon > 8: malicious
```

(c) between benign and trojan

benign and malicious PE files in Table 9. The message tables mostly do not exist in the backdoor+sniffer and constructor+virtool categories. The TimeDateStamp is usually obfuscated in the malicious executables. The number of resources are generally smaller in malicious PE files, whereas the benign files tend to have larger number of resources such as menus, icons and user defined resources. Moreover the reference of `WSOCK32.DLL` increases the probability that a given PE file might be doing a trojan-like activity. Similar insights are also provided by the rules developed in the training phase of RIPPER.

We have now established the fact that *PE-Miner is a realtime deployable scheme for zero-day malware detection*. A careful reader might ask whether the statement still holds if the "ground truth" is now changed as: (1) we cannot trust the classification of signature based packer detectors PEiD and Protection ID, and (2) a "crafty" attacker can forge the features of malicious files with those of benign files to evade detection. In the next section, we undertake a study to analyze robustness and reliability of PE-Miner in a number of stress scenarios – including the above-mentioned two scenarios.

6 Robustness and Reliability Analysis of PE-Miner

In this section, we do a stress and regression testing of PE-Miner to analyze robustness of its features and its resilience to potential evasive techniques.

6.1 Robustness Analysis of Extracted Features

It is a well-known fact that signature based packer detector PEiD, which we are using to distinguish between packed and non-packed executables, has approximately 30% false negative rate [8]. In order to convince ourselves that our extracted features are actually "robust", we evaluate PE-Miner in four scenarios: (1) training PE-Miner on 70% non-packed PE files and 30% packed PE files and testing on the remaining 70% packed PE files, (2) training PE-Miner on non-packed PE files only and testing on packed PE files, (3) training PE-Miner on packed PE files only and then testing on non-packed PE files, and (4) testing PE-Miner on a "difficult" dataset that consists of packed benign and non-packed malicious PE files. We assert that the scenarios (2) and (3) – although unrealistic – still provide valuable insight into the extent of bias that PE-Miner might have towards detection of packed/non-packed executables.

We want to emphasize an important point that there is no confusion about "ground truth" for packed executables in above-mentioned four scenarios because a packer only detects a file as "packed" if it has its signature in its database. The confusion about "ground truth", however, stems in the fact that a reasonable proportion of packed PE files could be misclassified as non-packed because of false negative rate of PEiD. Note that the false negatives of PEiD, reported in [8], consist of two types: (1) packed PE files that are wrongly classified as non-packed, and (2) PE files that are unclassified. We have not included unclassified files in our dataset to remove the false negatives of second type.

6.1.1 Scenario 1: Detection of packed benign and malicious PE files

The motivation behind the first scenario is to test if PE-Miner can distinguish between packed benign and packed malware, regardless of the type of packer. In order to ensure that our features are not influenced by the type of packing tool used to encrypt PE files, our "packed-only" dataset contains PE files (both benign and malware) packed using a variety of packers like UPX, ASPack, Armadillo, PECompact, WWPack32, Virogen Crypt 0.75, UPS-Scrambler, PEBundle and PEPack etc. Moreover, the "packed-only" dataset contains on the average 44% and 56% packed malicious and benign PE files respectively. We train PE-Miner on 70% non-packed executables and 30% packed executables and then test it on the remaining 70% packed executables. The results of our experiments for this scenario are tabulated in Table 10. We can easily conclude that PE-Miner has shown good resilience in terms of detecting accuracy once it is tested on packed benign and malicious PE files from both datasets.

6.1.2 Scenarios 2 and 3: Detection of packed/non-packed malicious PE files

In the second experiment, we train PE-Miner on non-packed benign and malicious PE files and test it on packed benign and malicious PE files. Note this scenario is more challenging because the training dataset contains significantly less number of packed files compared with the first scenario. In the third experiment, we train PE-Miner on packed benign and malicious PE files and test on non-packed benign and malicious PE files. The results of these experiments are tabulated in Table 10. It is clear from Table 10 that the detection accuracy of PE-Miner (RFR-J48) drops to 0.96, when it is trained on non-packed executables and tested on the packed executables. Likewise, the average detection accuracy of PE-Miner (RFR-J48) drops to 0.90 for the third scenario. Remember once we train PE-Miner on "packed only" dataset, then it gets 0% exposure to non-packed files and this explains deterioration in the detection accuracy of PE-Miner. We conclude that the detection accuracy of PE-Miner, even in these unrealistic stress testing scenarios, gracefully degrades.

6.1.3 Scenario 4: Detection of packed benign and non-packed malicious PE files

In [1], the authors report an interesting study about the ability of different schemes to detect packed/non-packed executables. They show that the detection accuracy of KM approach degrades on a "difficult" dataset

Table 10: An analysis of robustness of extracted features of PE-Miner (RFR) in different scenarios

Dataset	VX Heavens									Malfease
Malware	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	Average	-
Scenario 1: Detection of packed benign and malicious PE files										
IBK	0.999	1.000	1.000	0.999	0.999	0.998	0.999	0.999	0.999	0.812
J48	0.996	1.000	1.000	0.999	0.999	0.998	0.993	0.999	**0.998**	**0.991**
NB	0.971	0.988	0.963	0.955	0.996	0.980	0.978	0.987	0.977	0.934
RIPPER	0.997	0.996	0.999	0.990	0.993	0.985	0.858	0.998	0.977	0.988
SMO	0.985	0.998	1.000	0.996	0.994	0.994	0.985	0.998	0.994	0.706
Scenario 2: Training using non-packed executables only and testing using packed executables										
IBK	0.986	0.965	0.912	0.963	0.998	0.993	0.850	0.989	0.957	0.917
J48	0.982	0.999	0.998	0.937	0.999	0.963	0.857	0.954	**0.961**	**0.968**
NB	0.927	0.899	0.842	0.809	0.966	0.911	0.857	0.965	0.897	0.780
RIPPER	0.989	0.995	0.998	0.995	0.986	0.962	0.858	0.853	0.954	0.937
SMO	0.983	0.772	0.905	0.691	0.996	0.737	0.651	0.852	0.823	0.859
Scenario 3: Training using packed executables only and testing using non-packed executables										
IBK	0.975	0.965	0.964	0.878	0.793	0.982	0.911	0.904	0.921	0.855
J48	0.951	0.908	0.919	0.940	0.726	0.958	0.903	0.881	**0.898**	**0.903**
NB	0.685	0.965	0.668	0.633	0.689	0.979	0.688	0.688	0.749	0.789
RIPPER	0.979	0.938	0.967	0.972	0.747	0.768	0.840	0.867	0.885	0.904
SMO	0.977	0.941	0.877	0.882	0.536	0.983	0.835	0.904	0.867	0.849
Scenario 4: Detection of packed benign and non-packed malicious PE files ("difficult" dataset)										
IBK	0.999	1.000	1.000	0.999	0.998	0.998	0.998	0.994	0.998	0.992
J48	0.997	0.986	0.999	0.999	0.999	0.999	0.989	0.993	**0.995**	**0.996**
NB	0.954	0.963	0.995	0.988	0.966	0.990	0.975	0.986	0.977	0.948
RIPPER	0.998	0.984	0.998	0.993	0.986	0.999	0.992	0.996	0.993	0.948
SMO	0.989	0.996	1.000	0.997	0.996	0.997	0.984	0.992	0.994	0.945

consisting of packed benign and non-packed malicious PE files. According to the authors in [1], KM shows a bias towards detecting packed PE files as malware and non-packed PE files as benign. We also – in line with this strategy – tested PE-Miner on a "difficult" dataset created from both malware collections used in our study. The results are tabulated in Table 10. It is important to highlight that for these experiments PE-Miner is trained on the original datasets but is tested on the "difficult" versions of both datasets. One can conclude from the results in Table 10 that PE-Miner does not show any bias towards detecting packed executables as malicious and non-packed executables as benign.

Our experiments conclude that *the extracted features are actually "robust", and as a result, PE-Miner doest not show any significant bias towards detection of packed/non-packed executables.*

6.2 Reliability of PE-Miner

Now we test PE-Miner on a "crafty" malware dataset, especially designed to circumvent detection by PE-Miner. We particularly focus our attention on the *false negative rate* (or miss detection rate)[10] of PE-Miner when we replace features in malicious files with those of benign files. Someone can argue that if adversaries exactly know our detection methodology, they might be able to design strategies that evade detection by PE-Miner. The examples of such strategies could be especially crafted packing techniques, insertion of dummy resources, obfuscation of address pointers and other information present in headers etc.

We have conducted an empirical study to analyze the robustness of PE-Miner to such evasive techniques. To this end, we have "crafted" malware files in the datasets to contain benign-like features. Specifically, we have created seven "crafty" datasets in which for every malware file 5, 10, 30, 50, 100, 150 and 189 random features – out of 189 features – are *forged* with the respective features from a randomly chosen benign file. We now analyze the false negative rate of PE-Miner (RFR-J48) across these "crafty" datasets. The results tabulated in Table 11 highlight the robustness of PE-Miner to such crafty attacks. The false negative rate of PE-Miner stays below 1% when fifty features are simultaneously forged. For both datasets, the average false negative rate is approximately 5% even when 100 out of 189 features are forged. This shows that a large set of features, which cover structural information of almost all portions of a PE file, used by PE-Miner make

[10]The false negative rate is defined by the fraction of malicious files wrongly classified as benign.

Table 11: False negative rate for detecting malicious executables with PE-Miner on the "crafty" datasets

Dataset	VX Heavens									Malfease
Malware	Backdoor + Sniffer	Constructor + Virtool	DoS + Nuker	Flooder	Exploit + Hacktool	Worm	Trojan	Virus	Average	-
# Forged Features	False negative rate									
0/189	0.001	0.000	0.000	0.000	0.004	0.000	0.004	0.007	0.002	0.001
5/189	0.002	0.000	0.000	0.000	0.004	0.000	0.004	0.007	0.002	0.001
10/189	0.002	0.000	0.000	0.000	0.004	0.011	0.004	0.014	0.004	0.004
30/189	0.002	0.003	0.000	0.012	0.023	0.011	0.011	0.014	0.009	0.004
50/189	0.002	0.003	0.000	0.012	0.023	0.016	0.011	0.014	0.010	0.004
100/189	0.096	0.003	0.000	0.012	0.023	0.050	0.445	0.176	0.101	0.004
150/189	0.658	0.003	0.000	0.583	0.795	0.611	0.558	0.221	0.429	0.426
189/189	0.996	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.999	0.998

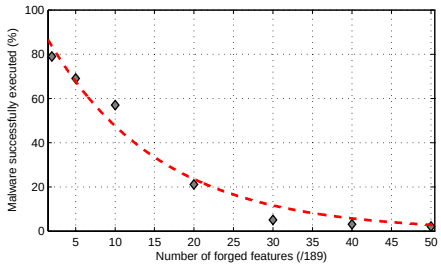


Figure 4: Execution analysis of crafted malware files

it very difficult for an attacker to evade detection – even when it manipulates majority of them at the same time.

Here we should emphasize that simultaneous manipulation of all features of a PE malware file requires significant level of skill, in-depth knowledge about the structure of a PE file and detailed understanding of our detection framework. If an attacker tries to randomly forge, using brute-force, the structural features of a PE malware file with those of a benign PE file then he/she will inevitably end up corrupting the executable image. Consequently, the file will not load successfully into memory. We have manually executed the "crafted" malicious executables. The objective is to understand that how many features a "crafty" attacker can successfully forge without ending up corrupting the executable image. The results of our experiments are shown in Figure 4. This figure proves our hypothesis that the probability of having valid PE files decreases exponentially with an increase in the number of simultaneously forged features. In fact, the successful execution probability approaches to zero as the number of simultaneously forged features approaches to 50. Referring back to Table 11, the average false negative rate of PE-Miner is less than 1% when 50 features are simultaneously forged. We, therefore, argue that it is not a cinch for an attacker to alter malicious PE files to circumvent detection by PE-Miner. However, we accept that an attacker can evade the detection capability of PE-Miner if: (1) he/she knows the exact details of our detection framework – including the detection rules, and (2) also has the "craft" to simultaneously manipulate more than 100 structural features without corrupting the executables.

Last but not least it is also pertinent to mention that the structural features, used by PE-Miner, are not affected by the code obfuscation and restructuring techniques. The majority of new malware, which has recently appeared, consists of different variants of existing malware generated with the help of such techniques [7]. Consequently, we can safely argue that PE-Miner is better equipped to effectively detect zero-day malware.

7 Conclusion

In this paper we present, PE-Miner, a technique for detection of malicious PE files. PE-Miner leverages the structural information of PE files and data mining algorithms to provide high detection accuracy with low processing overheads. Our implementation of PE-Miner completes a single-pass scan of all executables in the dataset (more than 17 thousand) in less than one hour. Therefore it meets all of our requirements mentioned in Section 2.

Our question oriented research methodology helped us in extensively searching almost all dimensions in the design space and the conclusion of the work is that three design options provide the best combination for detecting malicious PE files: (1) a rich PE format specific set of structural features which can be statically extracted from a PE file, (2) the preprocessing filters help in reducing the training and testing time of a classifier but they have minor role in improving the detection accuracy, (3) J48 classifier gives the best detection accuracy with low processing overheads. Therefore, our final PE-Miner framework has RFR preprocessing filter with J48 as the back-end classifier. We have also shown that PE-Miner has no particular bias in detecting packed/non-packed PE files. In future we will further investigate the "ground truth" of packed/non-packed classification of PE files by using dynamic unpackers like Ether. Our reliability analysis has shown that it is not straightforward for an attacker to forge features of a malicious file with those of a benign file without corrupting it.

We believe that our PE-Miner framework can be ported to Unix and other non-Windows operating systems. To this end, we have identified similar structural features for ELF file format in Unix and Unix-like operating systems. Our initial results are promising and show that PE-Miner framework is scalable across different operating systems. This dimension of our work will be the subject of forthcoming publications.

Moreover, PE-Miner framework is also ideally suited for detecting malicious PE files on resource constrained mobile phones (running mobile variants of Windows) because of its small processing overheads. We also plan to evaluate PE-Miner on a much larger executable dataset (with size of 140 GB) that we have just obtained from `offensivecomputing.org`.

Acknowledgments

We acknowledge Marcus A. Maloof and Jeremy Z. Kolter for continuous feedbacks regarding the implementation of byte sequence approach and the experimental setup. We also thank Roberto Perdisci for providing implementation details of McBoost, sharing Malfease malware dataset, and the results of their custom developed unpacker. We also thank VX Heavens moderators for making a huge malware collection publicly available and sharing packing statistics of malware. We thank Danny Quist of `offensivecomputing.org` for providing us their malware collection. We also thank Guofei Gu and S. Ali Khayam for providing useful feedback on an initial draft of this paper.

References

[1] R. Perdisci, A. Lanzi, W. Lee, "McBoost: Boosting Scalability in Malware Collection and Analysis Using Statistical Classification of Executables", Annual Computer Security Applications Conference (ACSAC), pp. 301-310, IEEE Press, USA, 2008.

[2] M.G. Schultz, E. Eskin, E. Zadok, S.J. Stolfo, "Data mining methods for detection of new malicious executables", IEEE Symposium on Security and Privacy (S&P), pp. 38-49, USA, 2001.

[3] J.Z. Kolter, M.A. Maloof, "Learning to detect malicious executables in the wild", ACM International Conference on Knowledge Discovery and Data Mining (KDD), pp. 470-478, USA, 2004.

[4] Symantec Internet Security Threat Reports I-XI (Jan 2002-Jan 2008).

[5] F-Secure Corporation, “F-Secure Reports Amount of Malware Grew by 100% during 2007”, Press release, 2007.

[6] P. Royal, M. Halpin, D. Dagon, R. Edmonds, W. Lee. “Polyunpack: Automating the hidden-code extraction of unpack-executing malware”, Annual Computer Security Applications Conference (ACSAC), pp. 289-300, USA, 2006.

[7] A. Stepan, “Improving Proactive Detection of Packed Malware”, Virus Buletin, March, 2006.

[8] R. Perdisci, A. Lanzi, W. Lee, “Classification of Packed Executables for Accurate Computer Virus Detection”, Elsevier Pattern Recognition Letters, 29(14), pp. 1941-1946, 2008.

[9] J. Kephart, G. Sorkin, W. Arnold, D. Chess, G. Tesauro, S. White. “Biologically inspired defenses against computer viruses”, International Joint Conference on Artificial Intelligence (IJCAI), pp. 985-996, USA, 1995.

[10] R.W. Lo, K.N. Levitt, R.A. Olsson, “MCF: A malicious code filter”, Elsevier Computers & Security, 14(6), pp. 541-566, 1995.

[11] Y. Ye, D. Wang, T. Li, D. Ye, Q. Jiang, “An intelligent PE-malware detection system based on association mining”, Journal in Computer Virology, pp. 323-334, Springer, 2008.

[12] O. Henchiri, N. Japkowicz, “A Feature Selection and Evaluation Scheme for Computer Virus Detection”, IEEE International Conference on Data Mining (ICDM), pp. 891-895, USA, 2006.

[13] P. Kierski, M. Okoniewski, P. Gawrysiak, “Automatic Classification of Executable Code for Computer Virus Detection”, International Conference on Intelligent Information Systems, pp. 277-284, Springer, Poland, 2003.

[14] T. A.-Assaleh, N. Cercone, V. Keselj, R. Sweidan. “Detection of New Malicious Code Using N-grams Signatures”, International Conference on Intelligent Information Systems (IIS), pp. 193-196, Springer, Poland, 2003.

[15] J.H.Wang, P.S. Deng, “Virus Detection using Data Mining Techniques”, IEEE International Carnahan Conference on Security Technology (ICCST), pp. 71-76, China, 2003.

[16] S.J. Stolfo, K. Wang, W.J. Li, “Towards Stealthy Malware Detection, Advances in Information Security”, Vol. 27, pp. 231-249, Springer, USA, 2007.

[17] W.J. Li, S.J. Stolfo, A. Stavrou, E.Androulaki, A.D. Keromytis, “A Study of Malcode-Bearing Documents”, International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment (DIMVA), pp. 231-250, Springer, Switzerland, 2007.

[18] M.Z. Shafiq, S.A. Khayam, M. Farooq, “Embedded Malware Detection using Markov n-Grams”, International Conference on Detection of Intrusions & Malware, and Vulnerability Assessment (DIMVA), pp. 88-107, Springer, France, 2008.

[19] M.Z. Shafiq, S.M. Tabish, F. Mirza, M. Farooq, “A Framework for Efficient Mining of Structural Information to Detect Zero-Day Malicious Portable Executables”, Technical Report, TR-nexGINRC-2009-21, January, 2009, available at `http://www.nexginrc.org/papers/tr21-zubair.pdf`

[20] S. Axelsson, “The base-rate fallacy and its implications for the difficulty of intrusion detection”, ACM Conference on Computer and Communications Security (CCS), pp. 1-7, Singapore, 1999.

[21] I.H. Witten, E. Frank, “Data mining: Practical machine learning tools and techniques”, Morgan Kaufmann, 2nd edition, USA, 2005.

[22] D.W. Aha, D. Kibler, M.K. Albert, “Instance-based learning algorithms”, Journal of Machine Learning, Vol. 6, pp. 37-66, 1991.

[23] J.R. Quinlan, “C4.5: Programs for machine learning”, Morgan Kaufmann, USA, 1993.

[24] M.E. Maron, J.L. Kuhns, “On relevance, probabilistic indexing and information retrieval”, Journal of the Association of Computing Machinery, 7(3), pp.216-244, 1960.

[25] W.W. Cohen, “Fast effective rule induction, “International Conference on Machine Learning”, pp. 115-123, USA, 1995.

[26] J. Platt, “Fast training of support vector machines using sequential minimal optimization”, Advances in Kernel Methods–Support Vector Learning, pp. 185-208, MIT Press, USA, 1998.

[27] Frans Veldman, “Heuristic Anti-Virus Technology”, International Virus Bulletin Conference, pp. 67-76, USA, 1993.

[28] Jay Munro, “Antivirus Research and Detection Techniques”, Antivirus Research and Detection Techniques, ExtremeTech, 2002, available at `http://www.extremetech.com/article2/0,2845,367051,00.asp`.

[29] K. Kendall, C. McMillan, “Practical Malware Analysis”, Black Hat Conference, USA, 2007.

[30] VX Heavens Virus Collection, VX Heavens website, available at `http://hvx.netlux.org`.

[31] Project Malfease, available at `http://malfease.oarci.net/`.

[32] J. Cheng, S.H.Y. Wong, H. Yang, S. Lu, “SmartSiren: virus detection and alert for smartphones”, International Conference on Mobile Systems, Applications and Services (MobiSys), pp. 258-271, USA, 2007.

[33] Microsoft Portable Executable and Common Object File Format Specification, Windows Hardware Developer Central, Updated March 2008, available at `http://www.microsoft.com/whdc/system/platform/firmware/PECOFF.mspx`.

[34] The PE file format, Webster Technical Documentation, available at `http://webster.cs.ucr.edu/Page_TechDocs/pe.txt`.

[35] `hash_map`, Visual C++ Standard Library, available at `http://msdn.microsoft.com/en-us/library/6x7w9f6z.aspx`.

[36] Description of DUMPBIN utility, Article ID 177429, Revision 4.0, Micorsoft Help and Support, 2005.

[37] M. Pietrek, “An In-Depth Look into the Win32 Portable Executable File Format, Part 2”, MSDN Magazine, March, 2002.

[38] F. Bellard, “QEMU, a fast and portable dynamic translator”, USENIX Annual Technical Conference, FREENIX Track, pp. 41-46, 2005.

[39] M.G. Kang, P. Poosankam, H. Yin, “Renovo: A hidden code extractor for packed executables”, ACM Workshop on Recurring Malcode (WORM), pp. 46-53, USA, 2007.

[40] PEiD, available at `http://www.peid.info/`.

[41] Protection ID - the ultimate Protection Scanner, available at `http://pid.gamecopyworld.com/`.

[42] F-PROT antivirus, available at `http://www.f-prot.com/`.

[43] AVG Free Antivirus, available at `http://free.avg.com/`.

[44] Panda Antivirus, available at `http://www.pandasecurity.com/`.

[45] DLL Help Database, Microsoft Help and Support, available at `http://support.microsoft.com/dllhelp/`.

[46] Windows File List, Easy Desk Software, available at http://www.easydesksoftware.com/winlist.htm.

[47] F-Secure Virus Description Database, available at http://www.f-secure.com/v-descs/.

[48] T. Fawcett, "ROC Graphs: Notes and Practical Considerations for Researchers", TR HPL-2003-4, HP Labs, USA, 2004.

[49] S.D Walter, "The partial area under the summary ROC curve", Statistics in Medicine, 24(13), pp. 2025-2040, 2005.

A List of DLLs and their description

Following is the list and a brief description of the DLLs used as features in our PE-Miner framework. The description of DLLs is taken from [45] and [46].

1. ADVAP132.DLL — Advanced Win32 application programming interfaces
2. AWFAXP32.DLL — Mail API fax transport
3. AWFXAB32.DLL — Address book
4. AWPWD32.DLL — Security support
5. AWRESX32.DLL — Resource Executor
6. AWUTIL32.DLL — At Work Security Support
7. BHNETB.DLL — Network monitor SMS client
8. BHSUPP.DLL — Network monitor SMS client
9. CCAPI.DLL — Microsoft Network component
10. CCEI.DLL — Microsoft Network component
11. CCPSH.DLL — Microsoft Network component
12. CCTN20.DLL — Microsoft Network component
13. CMC.DLL — Common messaging calls for Mail API 1.0
14. COMCTL32.DLL — User Experience Controls Library
15. COMDLG32.DLL — Common Dialogue Library
16. CRTDLL.DLL — Microsoft C Runtime Library
17. DCIMAN.DLL — Display Control Interface Manager
18. DCIMAN32.DLL — Display Control Interface Manager
19. DSKMAINT.DLL — Disk Utilities engine
20. GDI32.DLL — GDI Client DLL

21. GROUPPOL.DLL — Group policy support
22. HYPERTERM.DLL — Terminal DLL
23. KERNL32.DLL — Windows NT BASE API Client DLL
24. LZ32.DLL — LZ Expand/Compress API DLL
25. MAPI.DLL — Mail / Exchange component
26. MAPI32.DLL — Extended MAPI 1.0 for Windows NT
27. MFC30.DLL — Shared MFC DLL
28. MPR.DLL — Multiple Provider Router DLL
29. MSPST32.DLL — Microsoft Personal Folder/Address Book Service Provider
30. MSFS32.DLL — MAPI 1.0 Service Providers for Microsoft Mail
31. MSNDUI.DLL — Microsoft Network component
32. MSNET32.DLL — Microsoft 32-bit Network API Library
33. MSSHRUI.DLL — Shell extensions for sharing
34. MSVIEWUT.DLL — Service data-link libraries for display engines
35. NAL.DLL — Network monitor SMS client
36. NDIS30.DLL — Network monitor SMS client
37. NETAPI.DLL — Network API
38. NETAPI32.DLL — Net Win32 API DLL
39. NETBIOS.DLL — NetBIOS API Library
40. NETDI.DLL — Net Device installer
41. NETSETUP.DLL — Network server-based setup
42. NWAB32.DLL — Address book provider
43. NWNET32.DLL — NetWare client
44. NWNP32.DLL — NetWare component
45. OLEDLG.DLL — Microsoft Windows OLE 2.0 User Interface Support
46. POWERCFG.DLL — Advanced Power Management Control Panel
47. RASPI.DLL — Automated Software Profile, Analysis, Removal and Signature Information
48. RASAPI16.DLL — Remote Access Services 16-bit API Library
49. RASAPI32.DLL — Remote Access 16-bit API Library

50. `RPCRT4.DLL` — Remote Procedure Call Runtime
51. `RPCLTC1.DLL` — Remote Procedure Call libraries
52. `RPCTLC3.DLL` — Remote Procedure Call libraries
53. `RPCTLC5.DLL` — Remote Procedure Call libraries
54. `RPCTLC6.DLL` — Remote Procedure Call libraries
55. `RPCTLS3.DLL` — Remote Procedure Call libraries
56. `RPCTLS5.DLL` — Remote Procedure Call libraries
57. `RPCTLS6.DLL` — Remote Procedure Call libraries
58. `RPCNS4.DLL` — Remote Procedure Call Name Service Client
59. `RSRC32.DLL` — Resource Meter
60. `SAPNSP.DLL` — Winsock data-link library
61. `SECUR32.DLL` — Security Support Provider Interface
62. `SHELL32.DLL` — Windows Shell Common DLL
63. `SLENH.DLL` — Advanced Power Management options
64. `SHLWAPI.DLL` — Library for UNC and URL Paths, Registry Entries and Color Settings
65. `UMDM32.DLL` — Universal Modem Driver component
66. `USER32.DLL` — USER API Client DLL
67. `VERSION.DLL` — Version Checking and File Installation Libraries
68. `WININET.DLL` — Internet Extensions for Win32
69. `WINMM.DLL` — MCI API DLL
70. `WINREG.DLL` — Remote Registry support
71. `WINSOCK.DLL` — Socket API for Windows
72. `WS2_32.DLL` — Windows Socket 2.0 32-Bit DLL
73. `WSOCK32.DLL` — Windows Socket 32-Bit DLL

B Detailed Related Work

A significant amount of research has been conducted to analyze the characteristics and effects of malware. To this end, a number of schemes have been proposed to detect malware [2], [3], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]. However, to maintain focus in this paper, we include techniques proposed by Perdisci et al [1], Schultz et al [2] and Kolter et al [3] for detecting `Win32` malicious PE files. These techniques use *static analysis* and utilize n-grams and data mining algorithms to detect malicious PE files.

B.1 Perdisci et al [1] — McBoost

Recently, Perdisci et al have used a set of structural heuristics for detection of *packed executables* [8]. The authors argue to use their technique as a preprocessor before signature-based detectors. In this way, the executables need to be unpacked by a universal unpacker before doing the scanning. They use 9 heuristic features with different pattern recognition techniques for this purpose. The features include the number of standard sections, the number of non-standard sections, the number of executable sections, the number of readable/writable/executable sections, the number of entries in the import address table, the entropy of PE file header, the entropy of the code sections, the entropy of the data sections and the entropy of an entire PE file. A Multi-Layer Perceptron (MLP) classifier is used for eventual classification of the PE files. However, the scope of their work is limited to the detection of packed executables only.

In an extension to their previous work, Perdisci et al proposed *McBoost*, a statistical malware collection tool [1]. McBoost consists of three modules, **A**, **B** and **C**. The module **A** differentiates between packed and non-packed executables. It is further composed of three sub-modules: A1 is a heuristic-based classifier; A2 and A3 are n-gram based classifiers similar to those proposed by Kolter et al [3] (explained later in the section). A2 operates only on the code section of a PE file and A3 operates on an entire PE file. All executables classified as packed by the module **A** are sent to the module **B**. It contains a custom implementation of an unpacker for hidden code extraction. The implementation of module **B** uses QEMU emulator [38] similar to Renovo [39]. The module **C** performs eventual classification of malware. It also consists of two sub-modules: (1) C1 is an n-gram based classifier which operates on the code section of PE files classified as non-packed by the module **A**, and (2) C2 is also an n-gram based classifier which operates on the hidden code (extracted by the module **B**) of PE files classified packed by the module **A**.

The authors performed experiments on a dataset consisting of $5,586$ malicious PE files from Malfease dataset [31] and $2,258$ benign PE files. 37% malicius PE files are packed (detected using PEiD [40] and F-Prot antivirus [42]). Only 3% malicious PE files are non-packed and the rest of the files cannot be classified as packed or non-packed. The authors used Polyunpack [6] and custom developed unpacker for unpacking. On the other hand, only 2% benign executables are packed and rest 98% are non-packed in the original dataset. The skewness in the distribution of packed/non-packed malware and benign files is obvious.

McBoost is primarily a malware collection tool and its utility as an online realtime malware detection tool is limited due to high processing overheads and relatively lower detection rates. In [1], the authors report that McBoost requires approximately 1.06 seconds for detection of non-packed executables, 4.7 and 5.6 minutes for detection of packed PE files and benign PE files respectively. Further, McBoost is unable to analyze $1,030$ (approximately 13%) "heavily packed" executables which effectively require manual analysis for eventual classification.

B.2 Schultz et al [2] — Strings

In [2], Schultz et al use several data mining techniques to distinguish between the benign and malicious executables in Windows or MS-DOS format. They have done experiments on a dataset that consists of $1,001$ benign and $3,265$ malicious executables. These executables have 206 benign and 38 malicious samples in the PE file format. They collected most of the benign executables from Windows 98 systems. They use three different approaches to statically extract features from executables.

The first feature extracts DLL information inside PE executables. Further, the DLL information is extracted using three types of feature vectors: (1) the list of DLLs (30 boolean values), (2) the list of DLL function calls ($2,229$ boolean values), and (3) the number of different function calls within each DLL (30 integer values). RIPPER (an inductive rule-learning algorithm) is used on top of every feature vector for classification. These schemes based on DLL information provides an overall detection accuracy of 83.62%, 88.36% and 89.07% respectively.

The second feature extraction approach extracts the strings from the executables using GNU *strings* program. Naïve Bayes classifier is used on top of extracted strings for detection. This scheme provides an overall detection accuracy of 97.11%.

The third feature extraction approach uses byte sequences (n-grams) using hexdump. The authors do not explicitly specify the value of n used in their study. However, from an example provided in the paper, we deduce it to be 2 (bi-grams). The Multi-Naïve Bayes algorithm is used for classification. This algorithm uses voting by a collection of individual Naïve Bayes instances. This scheme provides an overall detection accuracy of 96.88%.

The authors also compare their proposed schemes with a custom developed signature-based detector that uses traditional byte sequence based signatures. Such schemes are meant for low false positive rates. The signature-based detector provides an overall accuracy of 49.28% only.

The results of their experiments reveal that Naïve Bayes algorithm with strings is the most effective approach for detecting the unseen malicious executables with reasonable processing overheads. The authors acknowledge the fact that the string features are not robust enough and can be easily defeated. Multi-Naïve Bayes with byte sequences also provides a very high detection accuracy, however, it has large processing and memory requirements. These overheads make the approach infeasible for realtime deployment.

B.3 Kolter et al [3] — KM

In [3], Kolter et al use n-gram and data mining approaches to detect malicious executables in the wild. They use n-gram analysis to extract features from $1,971$ benign and $1,651$ malicious PE files. The PE files have been collected from machines running Windows 2000 and XP operating systems. The malicious PE files are taken from an older version of the VX Heavens Virus Collection [30].

The authors evaluate their approach for two classification problems: (1) classification between the benign and malicious executables, and (2) categorization of executables as a function of their payload. The authors have categorized three types of malware, mailer, backdoor and virus due to the limited number of samples.

Top n-grams with the highest information gain as binary features (T if present and F if absent) in every classification problem. The authors have done pilot studies to determine the size of n-grams, the size of words and the number of top n-grams to be selected as features. A smaller dataset consisting of 561 benign and 476 malicious executables is considered in this study. They used 4-grams, one byte word and top 500 n-grams are selected as features.

A number of inductive learning methods, namely instance-based learner, Naïve Bayes, support vector machines, decision trees and boosted versions of instance-based learner, Naïve Bayes, support vector machines and decision trees are used for classification. The same features are provided as an input to all classifiers. They report their results as the area under an ROC curve (AUC) which is a more complete measure compared with the detection accuracy [48], [49]. AUCs show that the boosted decision trees outperform the rest of the classifiers for both classification problems.

C Classification Algorithms

We use the implementations of following classification algorithms which are available in Wakaito Environment for Knowledge Acquisition (WEKA) [21].

C.1 Instance Based Learner

The instance based classifier (IBk) is the simplest of all algorithms used in our comparative study. The classification is done on the basis of a majority vote of k neighboring instances [22]. In our study, we use

default parameters for instance based classifier (IBk) implemented in WEKA. We use the value of k as 5 and the window size is set to be 0 allowing maximum number of instances in the training pool with no replacements. We have not used distance weighting method. No internal cross validation is used in the algorithm to determine the value of k.

C.2 Decision Tree (J48)

Decision trees are usually used to map observations about an item to conclusions about the item's target value using some predictive models [23]. They are very easy to understand and are efficient in terms of time especially on large datasets. They can be applied on both numerical and categorical data, and statistical validation of the results is also possible.

We use C4.5 decision tree (J48) that is implemented in WEKA. We use the default parameters for J48. We do not utilize binary splits on nominal attributes for building trees because all of our selected features are numeric except the class labels. The confidence factor for pruning is set to 0.25, where lower values lead to more pruning. The minimum number of instance per leaf equals 2. The number of folds of training data is set to 3, where one fold is used for pruning and the rest are used for growing the tree.

C.3 Naïve Bayes

Naïve Bayes is a simple probabilistic classifier assuming naïve independence among the features i.e. the presence or absence of a feature does not affect any other feature [24]. The algorithm works effectively and efficiently when trained in a supervised learning environment. Due to its inherent simple structure it often gives very good performance in complex real world scenarios. The maximum likelihood technique is used for parameter estimation of Naïve Bayes models.

We use the default parameters for Naïve Bayes in WEKA. We neither use kernel estimator functions nor numeric attributes for supervised discretization that converts numeric attributes to nominal ones.

C.4 Inductive Rule Learner (RIPPER)

We also use a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), proposed by William W. Cohen as an optimization of IREP [25]. We chose rule based learners due to their inherent simplicity that results in a better understanding of their learner model. RIPPER, performs quite efficiently on large noisy datasets with hundreds of thousands of examples. It caters for missing attributes, numerical variables and multiple classes. The algorithm works by initially making a detection model composed of rules which are improved iteratively using different heuristic techniques. The constructed rule set is used to classify the test cases.

We use default parameters for RIPPER in WEKA. The training set is divided into 3 types, one is used for pruning and the rest are used for growing the rules. The minimum total weight of the instances in the rule is 2. The optimization of the rule set is done twice.

C.5 Support Vector Machines using Sequential Minimal Optimization (SMO)

The concept of Support Vector Machine (SVM), invented by Vladimir Vapnik, in its simplest form aims to develop a hyperplane that separates a set of positive samples from a set of negative samples with a maximum margin [26]. For linear separability of the problem space, SVMs use a kernel function for mapping training data to a higher-dimensional space. It is a quadratic programming (QP) problem, which can be very time consuming on larges dataset like ours. However, we are using SMO (Sequential Minimal Optimization) which is a fast and efficient SVM training algorithm implemented in WEKA [26]. It breaks the QP problem into smaller subsets which are later solved analytically reducing the processing overheads.

We use the default parameters for SMO in WEKA with a linear kernel. The complexity parameter is set at 1. The threshold on round-off error is set to 10^{-12}. We are using training data to generate a logistic model. The tolerance parameter for the experiments is set at 0.0010.