A Survey on Malware Detection Using Data Mining Techniques

YANFANG YE, West Virginia University
TAO LI, Florida International University & Nanjing University of Posts and Telecommunications
DONALD ADJEROH, West Virginia University
S. SITHARAMA IYENGAR, Florida International University

In the Internet age, malware (such as viruses, trojans, ransomware, and bots) has posed serious and evolving security threats to Internet users. To protect legitimate users from these threats, anti-malware software products from different companies, including Comodo, Kaspersky, Kingsoft, and Symantec, provide the major defense against malware. Unfortunately, driven by the economic benefits, the number of new malware samples has explosively increased: anti-malware vendors are now confronted with millions of potential malware samples per year. In order to keep on combating the increase in malware samples, there is an urgent need to develop intelligent methods for effective and efficient malware detection from the real and large daily sample collection. In this article, we first provide a brief overview on malware as well as the anti-malware industry, and present the industrial needs on malware detection. We then survey intelligent malware detection methods. In these methods, the process of detection is usually divided into two stages: *feature extraction* and *classification*/*clustering*. The performance of such intelligent malware detection approaches critically depend on the extracted features and the methods for classification/clustering. We provide a comprehensive investigation on both the feature extraction and the classification/clustering techniques. We also discuss the additional issues and the challenges of malware detection using data mining techniques and finally forecast the trends of malware development.

Categories and Subject Descriptors: D.4.6 [**Security and Protection**]: Invasive Software (e.g., viruses, worms, Trojan horses); I.5.2 [**Pattern Recognition**]: Design Methodology—*Pattern analysis*

General Terms: Security, Design, Algorithms, Experimentation

Additional Key Words and Phrases: Survey, malware detection, data mining

ACM Reference Format:
Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. ACM Comput. Surv. 50, 3, Article 41 (June 2017), 40 pages.
DOI: http://dx.doi.org/10.1145/3073559

The work is partially supported by the U.S. National Science Foundation under grant IIS-1213026, CNS-1461926, and CNS-1618629, Chinese NSF Grant under No. 91646116, and Scientific and Technological Support Project (Society) of Jiangsu Province No. BE2016776.
Authors' addresses: Y. Ye, Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA; email: yanfang.ye@mail.wvu.edu; T. Li, School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA & BDSIP Lab, School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing, 210023, China; email: taoli@cis.fiu.edu; D. Adjeroh, Lane Department of Computer Science and Electrical Engineering, West Virginia University, Morgantown, WV 26506-6109, USA; email: donald.adjeroh@mail.wvu.edu; S. S. Iyengar, School of Computing and Information Sciences, Florida International University, Miami, FL 33199, USA; email: iyengar@cis.fiu.edu.

DOI: http://dx.doi.org/10.1145/3073559

1. INTRODUCTION

As computers and Internet are increasingly ubiquitous, the Internet has been essential in everyday life. It has been reported by the ITU (International Telecommunication Union) that the number of Internet users worldwide, who always use Internet services such as e-banking, e-commerce, instant communication, education, and entertainment, has reached 2.92 billion as of 2014 [ITU 2014]. Just like the physical world, there are people with malicious intentions (i.e., cyber-criminals) on the Internet. They try to take advantage of legitimate users and benefit themselves financially. Malware (short for ***mal***icious soft***ware***), is a generic term widely used to denote all different types of unwanted software programs. These programs include viruses, worms, trojans, spyware, bots, rootkits, ransomware, and so on. Malware has been used by cyber-criminals as weapons in accomplishing their goals. In particular, malware has been used to launch a broad range of security attacks, such as compromising computers, stealing confidential information, sending out spam emails, bringing down servers, penetrating networks, and crippling critical infrastructures. These attacks often lead to severe damage and significant financial loss. To put this into perspective, according to a recent report from Kaspersky Lab, up to $1 billion was stolen in roughly 2 years from financial institutions worldwide due to malware attacks [Kaspersky 2015]. In addition, Kingsoft reported that the average number of infected computers per day was between 2-5 million [Kingsoft 2016].

Numerous malware attacks have posed serious and evolving security threats to Internet users. To protect legitimate users from these threats, anti-malware software products from different companies provide the major defense against malware, such as Comodo, Kaspersky, Kingsoft, and Symantec. Typically, the signature-based method is employed in these widely-used malware detection tools to recognize various threats. A signature is a short sequence of bytes, which is often unique to each known malware, allowing newly encountered files to be correctly identified with a small error rate [Ye et al. 2011]. However, due to the economic benefits, malware authors quickly developed automated malware development toolkits (e.g., Zeus [Song et al. 2008]). These toolkits use techniques, such as instruction virtualization, packing, polymorphism, emulation, and metamorphism to write and change malicious codes that can evade the detection [Beaucamps and ric Filiol 2007; Filiol et al. 2007]. These malware creation toolkits greatly lower the novice attackers' barriers to enter the cyber-crime world (allowing inexperienced attackers to write and customize their own malware samples) and lead to a massive proliferation of new malware samples due to their wide availability. As a result, malware samples have been rapidly gaining prevalence and have spread and infected computers at an unprecedented rate around the world. In 2008, Symantec reported that the release rate of malicious programs and other unwanted codes might exceed that of benign software applications [Symantec 2008]. This suggests that traditional signature-based malware detection solutions may face great challenges since they can be outpaced by the malware writers. For example, according to Symantec's report, about 1.8 million malware signatures were released in 2008, which resulted in 200 million detections per month [Symantec 2008]. In 2013, the suspicious files collected by the anti-malware lab of Kingsoft reached 120 million, 41.26 million (34%) of which were detected as malware [Kingsoft 2014]. While many malware samples have been detected and blocked, a large number of malware samples (e.g., the so-called "zero-day" malware [Wikipedia 2017f]) have been generated or mutated and they tend to evade traditional signature-based anti-virus scanning tools. This has prompted the anti-malware industry to rethink their malware detection methods, as these approaches are mainly based on variants of existing signature-based models.

In order to remain effective, many anti-malware companies started to use cloud (server) based detection [Nachenberg and Seshadri 2010; Ye et al. 2011; Stokes et al.

2012]. Cloud-based malware detection is conducted in a client-server manner with the cloud-based architecture [Ye et al. 2011]: the anti-malware products use a light signature base at the client (user) side to authenticate valid software programs and block invalid software programs, while predicting any unknown files (i.e., the gray list) at the cloud (server) side, and quickly produce the verdict results to the clients. With the development of malware writing and creating techniques, the number of file samples in the gray list is constantly increasing (e.g., more than new 500,000 file samples are collected per day by Kingsoft Cloud Security center). Thus, intelligent methods to automatically detect malware samples from the newly collected files at the cloud side are in urgent need. Consequently, many studies have been reported on using data mining and machine learning techniques to develop intelligent malware detection systems [Schultz et al. 2001; Kolter and Maloof 2004; Karim et al. 2005; Lee and Mody 2006; Ye et al. 2007; Moskovitch et al. 2008a; Kolbitsch et al. 2009; Ye et al. 2010, 2011; Karampatziakis et al. 2013; Tamersoy et al. 2014; Saxe and Berlin 2015; Ni et al. 2016].

In this article, we first provide a brief overview on malware as well as the anti-malware industry and present the industrial needs on malware detection. We then survey intelligent malware detection methods. In these methods, malware detection is a two-step process: *feature extraction* and *classification/clustering*. The performance of such malware detection methods critically depend on the extracted features and the categorization techniques. We provide a comprehensive investigation on both feature extraction and classification/clustering steps. We also discuss the additional issues and challenges of malware detection using data mining techniques and finally forecast the trends of malware development. The rest of this article is organized as follows: Section 2 presents the overview of the malware and anti-malware industry. Section 3 introduces the overall process of malware detection using data mining techniques. Section 4 describes the file representation methods of malware and Section 5 systematically discusses the feature selection methods for malware detection. Section 6 introduces the classification for malware detection, while Section 7 describes clustering for malware detection. Section 8 further discusses the additional issues of malware detection using data mining techniques. Section 9 forecasts the trends of malware development. Finally, Section 10 concludes the article.

2. OVERVIEW OF MALWARE AND ANTI-MALWARE INDUSTRY

Malware is the software program that deliberately meets the harmful intent of malicious attackers [Bayer et al. 2006b]. It has been designed to achieve the goals of attackers. These goals include disturbing system operations, gaining access to computing system and network resources, and gathering personal sensitive information without user's permission. As a result, malware often creates a menace to the integrity of the hosts, availability of the Internet, and the privacy of the users.

Malware can reach the systems in different ways and through multiple channels. These different ways are summarized below: (1) The vulnerable services over the network allow malware to infect accessible systems automatically. (2) The downloading process from the Internet: It has been shown that 70–80% of the malware come from popular websites [Rehmani et al. 2011]. By exploiting the web browser's vulnerability, a drive-by download is capable to fetch malicious codes from the Internet first and then execute the codes on the victims' machines [Egele et al. 2012]. (3) The attackers can also lure the victims into deliberately executing malicious codes on their machines. Typical examples include asking the users to install a provided "codec" to watch the movies which are hosted on the website, or clicking/opening images attached to spam emails [Egele et al. 2012]. In some cases, malware may only affect the system performance and create overload processes. In case of spying, malware hides itself in the system, steals critical information about the computer, and sends information to the attackers.

To protect legitimate users from the malware attacks, the major defense is the software products from anti-malware companies. However, the more successful the anti-malware industry becomes in detecting and preventing the attacks, the more sophisticated malware samples may appear in the wild. As a result, the arms race between malware defenders and malware authors is continuing to escalate. In the following sections, we introduce the taxonomy of malware, elaborate the development of malware industry, and then describe the progress of malware detection.

2.1. Taxonomy of Malware

Based on the different purposes and proliferation ways, malware can be categorized into various types. This section provides a brief overview of most common types of malware, such as viruses, worms, trojans, spyware, ransomware, scareware, bots, and rootkits.

***Viruses*:** A virus is a piece of code that can append itself to other system programs, and when executed, the affected areas are "infected" [Wikipedia 2017b]. Viruses cannot run independently since they need to be activated by their "host" programs [Spafford 1989]. The Creeper virus written by Bob Thoma was an experimental self-replicating program, which was first detected in the early 1970s [Wikipedia 2017b].

***Worms*:** Unlike a virus which requires its "host" program be run to activate it, a worm is a program that is able to run independently. Note that a worm can propagate a fully working copy of itself to other machines [Spafford 1989]. The Morris worm (unleashed in 1988) was the first publicly known program instance that exhibited worm-like behavior [Spafford 1989]. During the Morris appeal process, based on the estimate of the U.S. Court of Appeals, the cost of removing the Morris worms was around $100 million [Wikipedia 2017c]. The infamous worms, such as Love Gate, CodeRed, SQL Slammer, MyDoom, and Storm Worm, have successfully attacked tens of millions of Windows computers and caused great damages. For example, during its first day of release, the Code Red worms (first released in 2001) infected about 359,000 hosts on the Internet [Moore and Shannon 2002], while the worms of MyDoom (sighted in 2004) slowed down global Internet access by 10% and caused the access of certain websites to be reduced by 50% [Bizjournals 2011].

***Trojans*:** Compared with a worm, which is apt to propagate a fully working version of itself to other machines, Trojan is a software program that pretends to be useful but performs malicious actions in the backend [Schultz et al. 2001]. One of the recent notable trojans, Zeus (also called Zbot) is capable of carrying out many malicious and criminal tasks. Zeus has often been used to steal banking-related information by keystroke logging and form grabbing [Wikipedia 2017g]. In June 2009, security company Prevx discovered that over 74,000 FTP accounts had been compromised by Zeus on the websites of many companies (including ABC, Amazon, BusinessWeek, Cisco, NASA, Monster.com, Oracle, Play.com, and the Bank of America) [Wikipedia 2017g].

***Spyware*:** Spyware is a type of malicious program that spies on user activities without the users' knowledge or consent [Borders and Prakash 2004]. The attackers can use spyware to monitor user activities, collect keystrokes, and harvest sensitive data (e.g., user logins, account information).

***Ransomware*:** Ransomware is one of the most popular malware in recent years [Symantec 2016], which installs covertly on a victim's computer and executes a cryptovirology attack that adversely affects it [Wikipedia 2017d]. If the computer is infected by this malware, the victim is demanded to pay a ransom to the attackers to decrypt it.

***Scareware*:** Scareware is a recent type of malicious file that is designed to trick a user into buying and downloading unnecessary and potentially dangerous software, such as fake antivirus protection [Wikipedia 2016], which has posed severe financial and privacy-related threats to the victims.

***Bots*:** A bot is a malicious application that allows the bot master to remotely control the infected system [Stinson and Mitchell 2007]. Typical spread methods of bots are

exploiting software vulnerabilities and employing social engineering techniques. Once a system has been infected, the bot master can install worms, spyware, and trojans, and transform the individual victimized systems into a botnet. Botnets are widely used in launching Distributed Denial of Service (DDoS) attacks [Kanich et al. 2008], sending spam emails, and hosting phishing fraud. Agobot and Sdbot are two of the most notorious bots.

***Rootkits*:** A rootkit, a stealthy type of software, is designed to hide certain processes or programs and enable continued privileged access to computers [Wikipedia 2017e]. Rootkit techniques can be used at different system levels: they can instrument Application Programming Interface (API) calls in user-mode or tamper with operating system structures as a device driver or a kernel module.

***Hybrid Malware*:** Hybrid malware combines two or more other forms of malicious codes into a new type to achieve more powerful attack functionalities.

Some other categories of commonly encountered Internet pests can also be a nuisance to computer users, such as "Spamware," "Adware," and the like. Actually, these typical types of malware are not mutually exclusive. In other words, a particular malware sample may belong to multiple malware types simultaneously.

2.2. Development of Malware Industry

Viruses were the first instances of malware. The initial motivation for malware writers was often to highlight/identify the vulnerabilities of security or simply to demonstrate their technical capability. In addition, in order to evade the detection of anti-malware scanners, malware authors and attackers developed and applied various concealment techniques: (1) *Encryption*: Encrypted malware consists of an encryption algorithm, encryption keys, encrypted malicious code, and a decryption algorithm [Sung et al. 2004]. The key and decryption algorithm are used to decrypt the malicious component in the malware. Attackers use new generated key and encryption algorithms and produce new versions of the malware to evade the detection. (2) *Packing*: Packing is a technique that is used to encrypt or compress the executable file [Kendall and McMillan 2007]. Usually, a phase of unpacking is necessary to reveal the overall semantics of the packed malicious program. (3) *Obfuscation*: Obfuscation [Sung et al. 2004] aims to hide the program's underlying logic and prevent others from having any related knowledge of the code. Typical obfuscation techniques include adding garbage commands, unnecessary jumps, and the like. By applying obfuscation, the malicious codes and all their harmful functionality remain incomprehensible until they are activated. (4) *Polymorphism* [Bazrafshan et al. 2013]: A polymorphic malware is programmed to look different each time it is replicated, while keeping the original code intact. Different from simple encryption, a polymorphic malware can use an unlimited number of encryption algorithms, and in each execution, a part of the decryption code will change. Depending on the malware types, different malicious actions performed by the malware can be placed under the encryption operations. Usually, a transformation engine is embedded in the encrypted malware. Note that at any change the engine generates a random encryption algorithm. Then, the engine and malware are encrypted using the produced algorithm and a new decryption key is connected to them. (5) *Metamorphism* [Bazrafshan et al. 2013; Crandall et al. 2005]: Metamorphic malware is the most complex type of malware. In metamorphic malware, the malicious codes change themselves so that a new instance has no resemblance to the original one. The malware does not have any coding engine. In each transmission, changes occur automatically in the malware source code.

The motivations of malware writers changed as time passed by. With the pervasiveness of computers and the high development of the Internet, e-commerce has gained its popularity especially in banking and financial industries. eMarketer's latest forecasts

reported that worldwide business-to-consumer (B2C) e-commerce sales would reach $1.5 trillion [EMarketer 2014]. As a result, there is a flourishing underground malware economy in today's e-commerce [Zhuge et al. 2008]. By spreading a destructive payload, malware can first infect and gain control of vulnerable computer systems and use them to obtain illegal money [Zhuge et al. 2008]. The money prospect, instead of the fun factor, becomes the driving force of the malware development. Driven by considerable economic benefits, both the diversity and sophistication of malware samples have significantly increased over the last few years [Hu 2011]. For example, Ramnit, originally a generic worm detected in 2010, was altered by malware writers to steal 45,000 Facebook accounts. With some codes taken from the Zeus trojan to capture data from web sessions, the current version of Ramnit is a hybrid version of the original worm which allows hackers to commit financial fraud. Trojans, designed to provide unauthorized and remote access to computers, allowing hackers to steal sensitive information (e.g., Facebook accounts or e-bank accounts), have increased in popularity and currently account for the majority (73%) of malware collection [Ye 2010].

Nowadays, there has been a mature industrial chain of Trojans (as shown in Figure 1) that is profitable in terms of economic benefits: from Trojan creation to evading detection, propagation, bot controlling, account stealing, and sale. The division of labors is quite clear and the whole process and organization form a sound and complete assembly-line. In China, the growth of such a black chain is very fast and its output value can reach over $1 billion per year [Ye 2010].

However, despite the fact that malware is more sophisticated, the required knowledge to write and customize malware samples has actually decreased greatly [Hu 2011]. This is mainly because of the availability of many automatic malware creation toolkits, which are also easy-to-use, such as Zeus [TrendMicro 2010] and SpyEye [Coogan 2010]. These toolkits allow inexperienced attackers to customize and create malware programs to commit cyber crime, which leads to a massive proliferation of malware variants. Furthermore, many malware samples are constantly mutated to bypass anti-malware vendors detection. Instead of writing a new malware from scratch, the malware authors always adopt a more cost-effective strategy to refine existing malware samples (either source codes or binaries) by slightly modifying them to bypass anti-malware scanners' detection, using a diverse set of tools and technologies which typically include instruction re-ordering, garbage insertion, equivalent code substitution, and runtime packing. As a result, such variants of malware samples have evolved into a stream-lined process [Ollmann 2010]. Since malware variants can be generated automatically and rapidly, malware writers can replace the malware that are outdated and grant their variants with a favorable attack window before new detection signatures are developed and updated [Hu 2011]. The ease of this malware mutation process has resulted in an exponential growth of new malware file samples [Ye et al. 2009c]. Based on the data provided by Kingsoft Cloud Security Center [Kingsoft 2014, 2015, 2016], Figure 2 demonstrates the increasing trend of malware samples in China from 2003 to 2015. The figure shows that the number of malware samples has increased sharply since 2008 and the total number of new malware samples collected in 2015 reached over 40.66 million [Kingsoft 2016]. Unfortunately, this trend is likely to continue, and malware will remain as one of the greatest security threats faced by Internet users.

2.3. Progress in Malware Detection

2.3.1. Signature-based Malware Detection. To protect legitimate users from malware threats, software products from anti-malware companies (e.g., products from Comodo, Kaspersky, Kingsoft, MacAfee, and Symamtec) provide the major defense. Typically, they use the *signature-based method* to identify the known threats. A signature is a

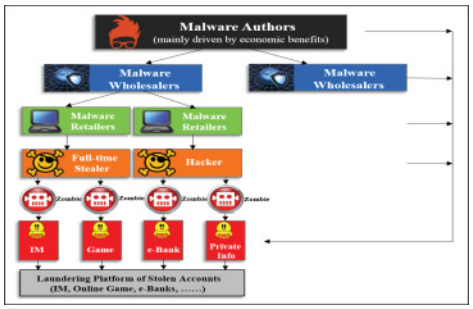

Fig. 1. Industrial chain of Trojans.

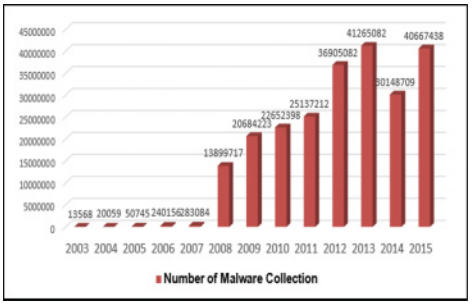

Fig. 2. The increment of malware samples.

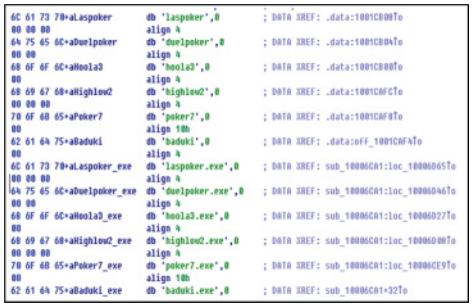

Fig. 3. An example of an online game Trojan's signature.

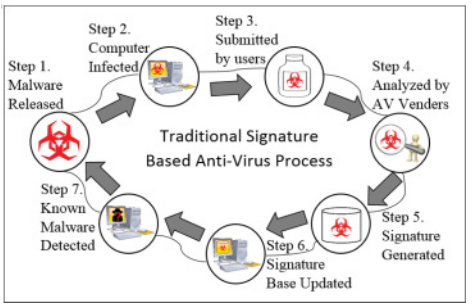

Fig. 4. Process of traditional signature-based malware detection.

short sequence of bytes unique to each known malware, which allows newly encountered files to be correctly identified with a small error rate [Ye et al. 2011]. Figure 3 shows an example signature used for the detection of an online game Trojan.

Signature-based method identifies unique strings from the binary code [Moskovitch et al. 2009]. The process of this traditional detection method can be illustrated in Figure 4. Whenever a new type of malware is unleashed, anti-malware vendors need to obtain an instance of the new malware, analyze the instance, create new signatures, and deploy in their clients [Micropoint 2008; Moskovitch et al. 2009]. Traditionally, the signature bases are often manually generated, updated, and disseminated by domain experts. This process is often known as time and labor consuming. This kind of detection method makes the anti-malware software tools less responsive to new threats. It may even allow some malware samples to bypass the detection and stay undetected for a long time. For example, a typical time window between a malware's release and its detection by anti-malware software tools is about 54 days [Hu 2011]. In the worse case, after 180 days, 15% of samples are still undetected [Damballa 2008].

2.3.2. Heuristic-based Malware Detection. By applying the counter-measures described above (e.g., encryption, packing, obfuscation, polymorphism, and metamorphism), malware writers can easily bypass the signature-based detection. From the late 1990s until 2008, the *heuristic-based method* was the most important way for malware detection. Heuristic-based detection is based on rules/patterns determined by experts to discriminate malware samples and benign files. These rules/patterns should be generic enough to be consistent with variants of the same malware threat, but not falsely matched on benign files [Egele et al. 2012]. However, the analysis of malware samples and the construction of rules/patterns by domain experts is often error-prone and

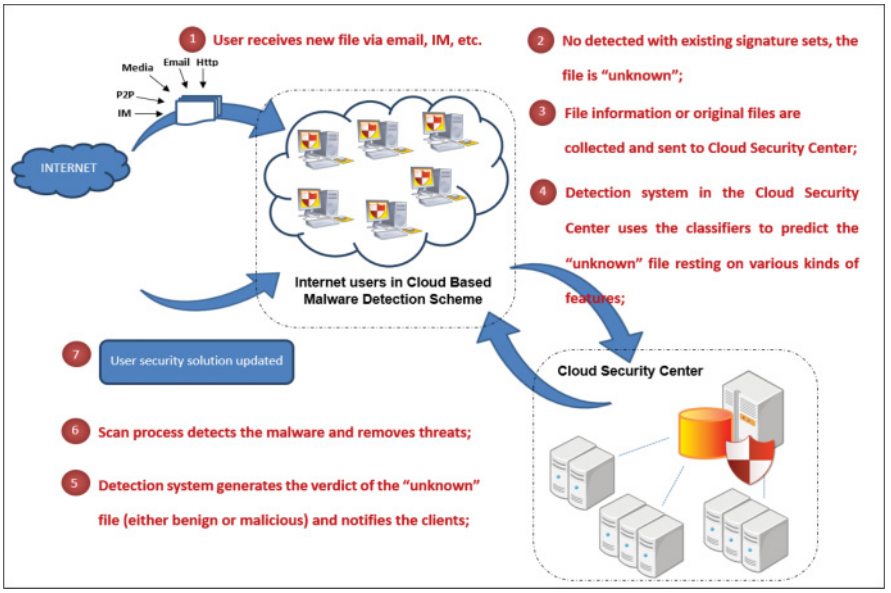

Fig. 5. The workflow of cloud-based malware detection scheme.

time-consuming. More importantly, as introduced above, driven by economic benefits, the malware industry has invented automated malware development toolkits (such as Zeus [TrendMicro 2010]) to create and mutate thousands of malicious codes per day which can slip through such traditional signature-based or heuristic-based detection [Symantec 2008]. As a result, manual analysis has become a major bottleneck in the workflow of malware analysis, demanding for intelligent techniques to analyze incoming samples automatically. Such intelligent techniques can allow anti-malware vendors to keep pace with the rapid malware generation and deployment and also reduce their response time to new malware threats. Besides, the speed of malware creation (e.g., over 10,000 new malware/day) is faster than signature generation and the increase of the signatures makes the clients becoming more and more "heavy."

2.3.3. Cloud-based Malware Detection. To overcome the above challenges and in order to remain effective, many anti-malware vendors have used cloud (server) based detection. The cloud-based detection workflow is presented in Figure 5. In particular, the scheme can be described below [Ye et al. 2011]:

(1) Users receive new files from the Internet via different channels on the client side.
(2) The signature sets on the clients are first used by anti-malware products for scanning the new files. The files will be marked as "unknown" if they can not be detected by existing signatures.
(3) Information of the unknown files (e.g., file reputations, file features, and/or even the files) will be collected and sent to the cloud sever.
(4) On the cloud sever, the classifier(s) will classify the unknown file samples and generate the verdicts (either benign or malicious).
(5) The verdict results will be sent to the clients immediately.
(6) Based on the results from the cloud server, the scanning process on the client side then performs the detection.
(7) With the quick response and feedback from the cloud server, client users will have the up-to-date security solutions.

To summarize, malware detection is now conducted in a client-server manner with the cloud-based architecture [Ye et al. 2011]: blocking invalid software programs from a blacklist and authenticating valid software programs from a white list at the client(user) side, and predicting any unknown files (i.e., the gray list) at the cloud (server) side and quickly producing the verdict results to the clients. The gray list contains unknown software files and these files could be either benign or malicious. Traditionally, the gray list was rejected or authenticated manually by malware analysts. With the development of the malware writing and creating techniques, the number of file samples in the gray list is constantly increasing. For example, the gray list collected by either Kingsoft or Comodo Cloud Security center usually contains more than 500,000 file samples per day [Ye 2010]. There is thus an urgent need for the development of intelligent techniques for supporting efficient and effective malware detection on the cloud (server) side.

In recent years, commercial products including Kingsoft's security products [Ye et al. 2009c, 2010], Comodo's AntiVirus (AV) products [Ye et al. 2011; Chen et al. 2015; Hardy et al. 2016], Symantec's AntiMalware (AM) products [Chau et al. 2011], and Microsoft's Internet Explorer [Stokes et al. 2012] have started using data mining techniques to perform malware detection.

3. OVERALL PROCESS OF MALWARE DETECTION BY APPLYING DATA MINING TECHNIQUES

In the past years, many research efforts have been reported on malware detection based on data mining techinques. These techniques are capable of classifying previously unseen malware samples, identifying the malware families of malicious samples, and/or inferring signatures. In these systems, the detection is generally a two-step process: *feature extraction* and *classification/clustering*. Figure 6 shows the overall process of malware detection using data mining techniques. In the first step, various features (see Section 4) such as API calls, binary strings, and program behaviors are extracted statically and/or dynamically to capture the characteristics of the file samples. In the second step, intelligent techniques such as classification or clustering are used to automatically categorize the file samples into different classes/groups based on the analysis of feature representations. Note that these data-mining-based malware detectors mainly differ on the feature representation and the employed data mining techniques.

Classification: To classify any unknown file, which could be either benign or malicious, the classification process can be divided into two consecutive steps: model construction and model usage. In the first step, training samples including malware and benign files are provided to the system. Then, each sample is parsed to extract the features representing its underlying characteristics. The extracted features are then converted to vectors in the training set. Both the feature vectors and the class label of each sample (i.e., malicious or benign) are used as inputs for a classification algorithm (e.g., Artificial Neural Network (ANN), Decision Tree (DT), and Support Vector Machines (SVM)). By analyzing the training set, the classification algorithm builds a classification model (or a classifier). Then, during the model usage phase, a new collection of unknown file samples, which could be either benign or malicious, are presented to the classifier generated from the training set. Note that the representative vectors of the new file samples are first extracted (using the same feature extraction techniques as in the training phase). The classifier will then classify the new file samples based on the extracted feature vectors.

Clustering: In many cases, very few labeled training samples exist for malware detection. Hence, researchers have proposed the use of clustering to automatically group malware samples that exhibit similar behaviors into different groups. Clustering is the task of grouping a set of objects such that objects in the same group (called a

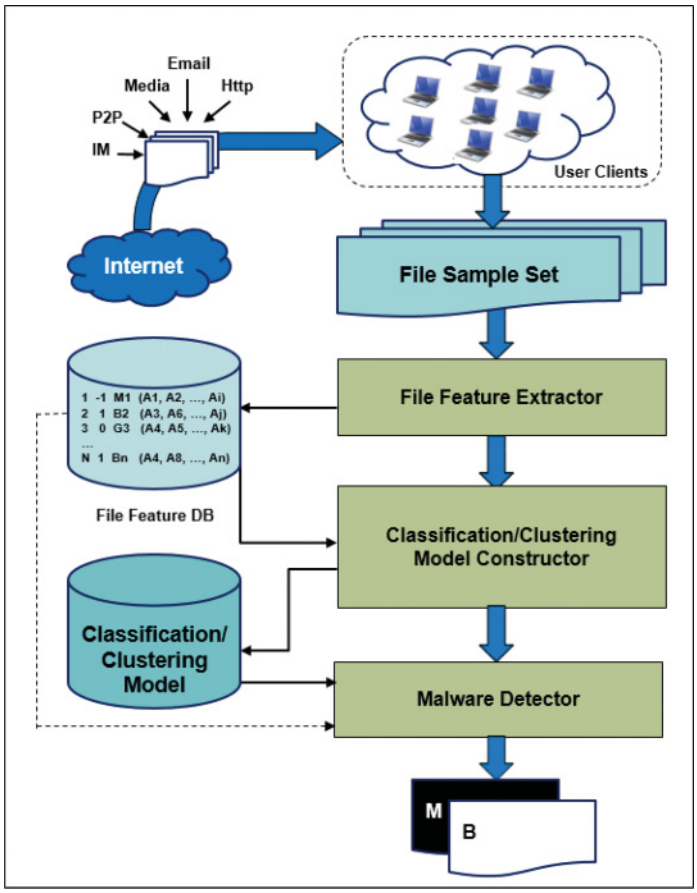

Fig. 6. The overall process of malware detection using data mining techniques.

Table I. Measures of Classification-Based Malware Detection Performance

Measures	Specification
True Positive (*TP*)	Number of file samples correctly classified as malicious
True Negative (*TN*)	Number of file samples correctly classified as benign
False Positive (*FP*)	Number of file samples wrongly classified as malicious
False Negative (*FN*)	Number of file samples wrongly classified as benign
TP Rate(*TPR*)	$TP/(TP+FN)$
FP Rate(*FPR*)	$FP/(FP+TN)$
Accuracy (*ACY*)	$(TP+TN)/(TP+TN+FP+FN)$

cluster) are more similar (e.g., using certain distance or similarity measures) to each other than to those in other groups (clusters). Clustering allows automatic malware categorization and also enables the signature generation for detection.

For evaluation purposes, the following classical measures shown in Table I are usually employed to evaluate the performance of *classification*-based malware detection. Note that in malware detection, malware samples are often used as positive instances. The True Positive Rate (TPR) measure is the rate of malware samples (i.e., positive instances) correctly classified by the classification model, while the False Positive Rate (FPR) is the rate of benign files (i.e., negative instances) wrongly classified

(i.e., misclassified as malware samples). The Accuracy (ACY) measures the rate of the correctly classified file instances, including both positive and negative instances. The evaluation approach using the traditional measures is often called as the *cumulative* approach [Nachenberg and Seshadri 2010]. The cumulative approach measures the performance of malware detection methods/systems at the instance level (i.e., per file level). There is another type of measurement approach, named the *interactive-based* measurement approach [Ramzan et al. 2013], which attempts to measure the performance of malware detection methods/systems at the transaction level (i.e., considering different user bases). The interactive-based measurement approach measures the true protection and false positive impact of malware detection methods/systems based on the actual user base. For example, imagine that at time T, there are two testing malware files, A (which has 1 million users) and B (which has 10 users), and the malware detection method/system wrongly classified the first file A (e.g., a false negative, classifying the malware as a benign file), but classified the second file B correctly as a malware sample. Based on just these two files, the cumulative measurement would give the ACY of 50%, since it classified one file correctly and one file incorrectly; but in the interactive-based measurement approach, it would assign different weights to the false negatives and true positives based on the users of file A and B. For the *clustering*-based methods, the performance of different algorithms are usually evaluated by using Macro-F1 and Micro-F1 measures, which emphasize the performance of the system on rare and common categories, respectively [Sebastiani 2002].

4. FEATURE EXTRACTION

Feature extraction method extracts the patterns used for representing the file samples. In this article, we mainly discuss the detection on Windows Portable Executable (PE) files. Note that PE is a common file format for Windows operating systems and PE malware is the majority of malware samples. Note that CIH, CodeBlue, CodeRed, Killonce, LoveGate, Nimda, Sircam, and Sobig all aim at PE files [Ye et al. 2007]. There are mainly two different types of feature extraction in malware detection: static analysis and dynamic analysis.

4.1. Static Analysis

Static analysis analyzes the PE files without executing them. The target of static analysis can be binary or source codes [Christodorescu and Jha 2003]. A PE file needs to be decompressed/unpacked first if it is compressed by a third-party binary compression tool (e.g., UPX and ASPack Shell) or embedded within a homemade packer [Ye et al. 2007]. To decompile windows executables, the disassembler and memory dumper tools can be used. Disassemble tools (e.g., IDA Pro [IDAPro 2016]) display malware codes as Intel $\times 86$ assembly instructions. Memory dumper tools (e.g., OllyDump [2006] and LordPE [2013]) are used to obtain protected codes located in the main memory and dump them to a file [Gandotra et al. 2014]. Memory dump is quite useful for analyzing packed executables that are difficult to disassemble. After the executable being unpacked and decrypted, the detection patterns used in static analysis can be extracted, such as Windows API calls, byte n-grams, strings, opcodes (operational codes), and control flow graphs.

—**Windows API Calls:** Windows API calls are used by almost all programs to send the requests to the operating system [Orenstein 2000]. Thus, Windows API calls can reflect the behavior of program code pieces. For example, the Windows API of *"GetVersionExA"* in *"KERNEL32.DLL"* can be used by malware to check the version of the current OS, which is achieved by invoking a system call. Therefore, the associations and relationships among the Windows APIs may capture the underlying semantics

of the malware behaviors and can be used as essential features for malware detection, such as, the Windows API calls in the *"KERNEL32.DLL"* of *"OpenProcess," "CopyFileA," "CloseHandle," "GetVersionEx A," "GetModuleFileNameA," and "WriteFile"* always co-occur in malware sample collection while rarely show together in a benign file set [Ye et al. 2007].

—**N-grams:** N-grams are all substrings in the program code with a length of *N* [Henchiri and Japkowicz 2006a]. For example, the *"82EDD875"* sequence is segmented (represented) into 5-gram as *"82EDD," "2EDD8," "EDD87,"* and *"DD875."* Over the past decade, many researches have conducted unknown malware detection based on the binary code content; see the studies in Kolter and Maloof [2004], Abou-Assaleh et al. [2004], Karim et al. [2005], Elovici et al. [2007], Masud et al. [2007], and Anderson et al. [2012].

—**Strings:** The interpretable strings are the high-level specifications of malicious behaviors. These strings can reflect the attacker's intent and goal since they often contain the important semantic information [Ye et al. 2009]. For example, "*<html><scriptlanguage = 'javascript'>window.open('readme.eml')*" always exists in the worms of "Nimda," implicating that the worms try to infect the scripts. Another example is the string "*&gameid = %s&pass = %s; myparentthreadid = %d; myguid = %s*," which indicating the attackers' intention of stealing the password of the online game and sending it back to the server. Besides, the strings are robust features and it is not easy for the malware authors to evade the string-based detection. This is because even if the malware variants can be generated by re-compiling or adopting obfuscation techniques, modifying all the interpretable strings is not practical in most programs [Ye et al. 2009].

—**Opcodes:** An OpCode (i.e., Operational Code) is the subdivision of a machine language instruction that identifies the operation to be executed [Wikipedia 2017a]. More specifically, a program is defined as a series of ordered assembly instructions. An instruction is a pair composed of an operational code and an operand or a list of operands (e.g., *"mov ebx ebx," "add eax 1," "xor eax eax," and "call sub_401BCD"*). Instruction segments can often reflect the program functionality. Studies have shown that, in practice, malware samples derived from the same source code or belong to the same family often share a large number of instruction blocks/segments [Ye et al. 2010].

—**Control Flow Graphs (CFGs):** A CFG is a graph that represents the control flow of a program. CFGs are widely used in software analysis and have also been widely studied [Anderson et al. 2012].

There are also many other static features to represent the file samples, such as file property, file resource information, and export table. Static analysis is able to explore/investigate all possible execution paths in malware samples. Hence, it has the advantage of being exhaustive in detecting malicious logic. In other words, static analysis does not have the coverage problem that dynamic analysis suffers from. Another advantage of static analysis is that the analyzer's machines cannot be attacked by the malware under study. One disadvantage of static analysis is, when dealing with certain situations, its inability due to undecidability (e.g., indirect control transfer through function pointers). So tradeoffs between precision and efficiency have to be made whenever points-to analysis is involved. Other drawbacks of static analysis include a lack of support for runtime packing code and the limitation related to complex obfuscation. Moser et al. [2007] discussed the drawbacks of static analysis. They suggested that dynamic analysis can be a necessary and useful complement to static analysis.

4.2. Dynamic Analysis

Dynamic analysis techniques (e.g., debugging and profiling) observe the execution (on a real or virtual processor) of the PE files to derive features [Egele et al. 2012]. Various techniques, such as autostart extensibility points, function parameter analysis, function call monitoring, information flow tracking, and instruction traces can be applied to perform dynamic analysis [Egele et al. 2012; Gandotra et al. 2014]. Typical dynamic analysis tools include Valgrind [Nethercote and Seward 2007], QEMU [Qemu 2016], and strace. There has been a substantial amount of studies on dynamic analysis of malware, varying in the execution environment for the malware and analysis granularity. Below we categorize the dynamic analysis techniques according to the execution environment:

—**Debugger:** A debugger such as GDB [Loukides and Oram 1996], Windbg [Robbins 1999], or Softice [of Compuware 1999] can be used for fine-grained analysis of binary code, including malware, at the instruction level. However, most if not all malware have become smart enough to detect the existence of a debugger by monitoring changes to their code that are necessary to support breakpoints, a chief vehicle for debugging. To counter such anti-debug malware, new ways of debugging have been investigated. VAMPiRE [Vasudevan and Yerraballi 2005] supports stealth breakpoints to assist in debugging of self-modifying and/or self-checking malware by leveraging virtual memory and hardware single-stepping mechanisms. Built on top of VAMPiRE, Cobra [Vasudevan and Yerraballi 2006] is a fine-grained malware analysis framework that can be selectively deployed only on malware-specific code streams to improve analysis efficiency. Finally, Ether [Dinaburg et al. 2008] is a debugging tool leveraging hardware virtualization extensions such as Intel VT [Intel 2013] to remain invisible to the malware.
—**Simulator:** This group of tools runs the malware in a controlled environment and monitors its actions. For example, Detours [Hunt and Brubacher 1998] is a dynamic instrumentation tool that can detect Windows APIs invoked by a malware sample, and CWSandbox [Willems et al. 2007] monitors Windows API calls by the malware by performing API hooking and DLL injection.
—**Emulator:** TTAnalyze [Bayer et al. 2006a] is a QEMU-based dynamic malware analysis framework that monitors Windows API calls and Windows native system calls, as well as function call parameters. TEMU in the BitBlaze [Song et al. 2008] binary analysis platform is an emulator that supports dynamic instrumentation at the instruction level and whole-system taint tracking. A number of specialized analysis tools [Caballero et al. 2007; Kang et al. 2007; Yin et al. 2007] have been built on top of TEMU. K-Tracer [Lanzi et al. 2009] dynamically traces the execution of Windows kernel-level rootkits to find their system data manipulation behaviors.
—**Virtual Machine:** Strider HoneyMonkeys [Wang et al. 2006a] determines malicious web sites by visiting them in a Microsoft Virtual PC- and Virtual Server-based environment and identifying persistent system state changes caused by such visits. vGrounds [Jiang et al. 2005] studies worm behavior in a User Mode Linux (UML) based virtual environment.

In dynamic feature extraction, the configuration- or environment-dependent information (such as variable value, system configuration, and program input) has been resolved during the extraction. This is one big advantage of dynamic analysis. Those environment- or configuration-dependent pose difficulty for static analysis because they are beyond the code itself. Dynamic analysis is especially useful in analyzing packed malware because in most cases, when the malware runs, at some point the malware has to unpack itself, and its original code will be in the main memory. Renovo [Kang et al.

2007] is such a tool that extracts hidden code by tracking the newly-written memory areas of the malware program, even if it is hidden through multiple layers of compression and encryption. The limited coverage is a disadvantage of dynamic analysis. The reason is that one execution session can only explore one particular program path in the malware. But some malware behaviors may depend on some special condition (e.g., on a particular time, when a particular file is received, or a certain command/operation is executed).

Besides, since it takes time for a malware sample to expose all its behaviors while running, dynamic analysis is often more time consuming and requires more resources than static analysis. As a result, these properties limit the adoption of dynamic analysis in commercial analysis systems.

4.3. Hybrid Analysis

Both static and dynamic feature extraction approaches have their own advantages and limitations. Compared with dynamic feature representation, the static approach is cheaper and can cover all code paths (including the program pieces that are not always executed), therefore providing a more accurate and complete characterization of the program functionalities [Hu 2011]. However, it has high performance overhead due to low-level mutation techniques (such as obfuscation and packing). On the contrary, dynamic analysis is resilient to low-level obfuscation and is suitable for detecting malware variants and new families, but often performs poorly on trigger-based malware samples. Besides, dynamic analysis is cost expensive and not scalable due to its limited coverage. Based on the statistics from Comodo Cloud Security Center, about 80% of the file samples can be well-represented using static features, while just around 40% of the file samples can successfully run dynamically [Ye et al. 2011].

Due to their respective pros and cons, neither static nor dynamic-based feature extraction approaches can provide a perfect solution to the feature extraction in malware analysis [Hu 2011]. Therefore, a comprehensive approach that integrates both static and dynamic analysis and gains the benefits of both would be desirable. *Hybrid analysis* is such an approach that combines the respective advantages of both static and dynamic analysis. For example, the packed malware can first go through a dynamic analyzer such as PolyUnpack [Royal et al. 2006], where the hidden-code bodies of a packed malware instance are extracted by comparing the runtime execution of the malware instance with its static code model. Once the hidden-code bodies are uncovered, a static analyzer can continue the analysis of the malware.

4.4. Other Novel Features

There is also some research using the semantics of file content to represent the file samples. For example, a semantics-aware malware detection method is proposed in Christodorescu et al. [2005]. In the proposed framework, malicious behavior was described using templates, which were instruction sequences where variables and symbolic constants were used. Based on the extracted features, a semantics-aware matching algorithm was proposed for malware detection.

Besides static and dynamic features extracted from file content, are there any other features that can represent malware or benign files? As the moral says, *"man is known by the company he keeps."* Actually, the relationships among different file samples may imply their interdependence and can provide critical information about their properties and characteristics [Ye et al. 2011; Tamersoy et al. 2014]. More precisely, a file's legitimacy can be inferred by analyzing its relations (co-occurrences) with other labeled (either benign or malicious) peers. For example, *if an unknown file always co-occurs with many trojans, then there is a high possibility that the file is a malicious trojan-downloader* [Ye et al. 2011] that can download and install multiple unwanted

applications (e.g., trojan, adware) from remote servers. Ye et al. combined both file relations and file content for malware detection. In particular, their proposed methods have been successfully incorporated into Comodo's anti-malware software products [Ye et al. 2011]. Recently, more commercial anti-malware products including Symantec's AM products [Nachenberg and Seshadri 2010] and Microsoft's Internet Explorer [Stokes et al. 2012] have started using the features beyond file contents for malware detection: (1) Symantec's AM products inferred the file reputations by analyzing file-to-machine relations [Chau et al. 2011] and also used file-relation graphs for malware detection [Tamersoy et al. 2014]; (2) Microsoft's Internet Explorer used file placements for malware detection [Venzhega et al. 2013].

Table II summarizes some representative studies of different feature extraction methods in malware detection.

5. FEATURE SELECTION

The previous section introduced methods for feature extraction for malware analysis and detection. Before describing popular classification methods used for malware detection in the next section, here we first discuss key methods used for feature selection and how they have been used in malware detection. Feature selection is an important step in many classification or prediction problems [Langley 1994; Jain et al. 2000; Kwak and Choi 2002; Murphy 2012]. The need for feature selection is motivated by the fact that, in certain classification and prediction problems, the number of features could be quite large, at times running into the millions, and thus could significantly exceed the capacity of the underlying machine to process within a reasonable time. Further, improvement in classification performance and minimization of classification errors could significantly depend on the ability of the system to quickly identify, select, and use only the most representative features. For the case of malware detection in particular, it might be impractical to construct the required model using each and every extracted feature. For instance, the number of features generated using n-grams grows exponentially with n, and using all the features could be quite computationally intensive, in terms of both memory usage and CPU time. The large number of features could also introduce unnecessary noise and large amounts of redundant and irrelevant features, further confounding the classifier. To reduce these problems, malware classification methods often adopt a feature selection for dimensionality reduction and to improve the compactness of the feature representation.

Feature Selection is essentially the process of selecting a subset of relevant and informative features from a larger collection of features for use in model construction [Guyon and Elisseeff 2003]. The central assumption is that the data contains many redundant or irrelevant features, which can be eliminated without a significant negative impact on later classification or prediction performance. Redundant features are those that provide no additional information beyond what is already provided by the currently selected features. Irrelevant features are those that do not provide any useful information in the given context. Thus, what is irrelevant could be dependent on the specific application being considered. Three important considerations in feature selection are determining the starting point for the selection process, how the selection proceeds given this starting point, and the stopping criteria for the selection procedure. For instance, in Forward Feature Selection (FFS), the process starts with an empty feature set and with new features added successively to the set. An alternative is Backward Feature Selection (BFS), whereby the process starts by using the set of all available features, and then successively removes features from the set. Removal of features (in BFS) or addition of features (as in FFS) are usually performed based on certain selection criteria, for instance, by ranking the features based on their estimated discrimination ability. Thus, independent of the direction of the search for features to

Table II. Summary of Some Typical Feature Extraction Methods in Malware Detection

Survey	Static Analysis	Dynamic Analysis	Other Analysis
Schultz et al. [2001]	DLL call information, strings, and byte sequences	✗	✗
Kolter and Maloof [2004]	Binary *n*-grams	✗	✗
Abou-Assaleh et al. [2004]	Binary *n*-grams of various lengths	✗	✗
Henchiri and Japkowicz [2006b]	16-byte sequences	✗	✗
Wang et al. [2006b]	DLLs and APIs	Modifications upon system files, registries, and network activities	✗
Ye et al. [2007, 2009a, 2009b]	Windows API calls	✗	✗
Elovici et al. [2007]	5-grams byte sequences and PE header	✗	✗
Masud et al. [2007, 2008]	Binary *n*-grams, assembly instruction sequences, and DLL call information	✗	✗
Moskovitch et al. [2008a, 2008c]	n-gram OpCode sequences, byte n-grams	✗	✗
Siddiqui et al. [2009]	Variable length instruction sequence	✗	✗
Ye et al. [2009]	Interpretable strings	✗	✗
Ye et al. [2009c]	Windows API calls and strings	✗	✗
Tian et al. [2010]	✗	API call sequences running in a virtual environment	✗
Firdausi et al. [2010]	✗	Behaviors extracted in sandbox environment	✗
Anderson et al. [2011]	✗	Graphs constructed from dynamically collected instruction traces	✗
Santos et al. [2011b, 2011a]	*n*-gram distributions	✗	✗
Chau et al. [2011]	✗	✗	File-to-machine relation graphs
Ye et al. [2011]	✗	✗	File content combining file relations
Anderson et al. [2012]	2-gram byte sequences, disassembled OpCodes, control flow graph, and miscellaneous file information	Instruction traces, system call traces	✗
Santos et al. [2013]	Sequence of operational codes	System calls, operations, and raised exceptions	✗
Islam et al. [2013]	Function length frequency and printable sting information	API function names and API parameters	✗
Karampatziakis et al. [2013]	✗	✗	Graphs induced by file relationships
Tamersoy et al. [2014]	✗	✗	File-to-file relation graphs
Saxe and Berlin [2015]	Byte entropy histogram, PE import information, and numerical PE fields	✗	✗
Das et al. [2016]	✗	Resource-critical system call patterns	✗

select, another key consideration is how the feature subsets are evaluated for inclusion in the selected subset.

There are three major approaches used for this evaluation: filter, wrapper, and embedded methods. (1) *Filter*: Filters determine the best feature subsets by using statistical approaches. Each feature is scored individually on certain specified criteria and the features are then ranked based on the scores. Feature selection then simply becomes a manner of selecting the features based on this ranking. Alternatively, subsets of features could be selected by computing some quantitative measures on the discriminative capability of the feature subsets, based on which the subsets can be ranked [Guyon and Elisseeff 2003]. Filter-based strategies consider the features as being independent of the later classification stage, or of the specific prediction problem being addressed. Thus, they exploit the general characteristics of the training datasets to select certain features while eliminating others. One key advantage of filter methods is that they are generally fast but still often capture the essence of the feature sets. Another is the independence from the specific method used for later classification, implying that the filters could provide a powerful mechanism for pre-processing before actual prediction or classification is performed. (2) *Wrapper*: With wrapper methods, the actual classification or prediction algorithm is used to evaluate the performance of different feature subsets. Here, a candidate feature subset is used to perform the required analysis (whether classification or prediction), and the performance in the analysis task using some training data is then used to evaluate the feature subset [Langley 1994]. Clearly, training a new model for each given feature subset is a very computationally intensive process. How to choose the subsets efficiently, and yet maintain a good performance in terms of robustness and accuracy still remains a challenge. However, for a given classification algorithm, the wrapper approach often leads to the best performing feature subsets. Another key advantage is the general applicability of the approach, independent of any specific classification algorithm. (3) *Embedded*: With embedded methods, feature selection is performed as part of the model construction process. An embedded method is usually specific to a given classification algorithm [Guyon and Elisseeff 2003]. Some view embedded approaches as a type of wrapper method [Wald et al. 2013], while others view them as lying between filters and wrappers in terms of computational complexity. For instance, some embedded methods perform more efficiently than wrapper methods by directly optimizing an objective function, often defined by two or more parameters – one to encourage the goodness-of-fit and the other to penalize for a large number of variables [Rakotomamonjy 2003; Perkins et al. 2003]. But these are still generally slower than filter methods. The close interaction with the specific classification scheme, and the added ability to factor in potential interactions between the features, provide some important advantages in terms of robustness and classification performance.

From the foregoing, we can see that each category of techniques has its own benefits and drawbacks [Guyon and Elisseeff 2003; Wald et al. 2013]. Filter techniques are much faster than the other methods (since they evaluate each feature once, rather than evaluating a large number of feature subsets), but they might require larger feature subset sizes before finding the best sets of features. Wrapper and embedded techniques are both significantly slower, but they often select better performing feature subsets for the specific classification algorithm used. Thus, each class of techniques still has its place in the general problem of feature selection. Given that filter techniques are highly scalable (important and critical for high-dimensional datasets), relatively simple and efficient, and independent of the underlying classification algorithms [Saeys et al. 2007], they represent the most popular feature selection approach in malware detection.

In a filter approach, each feature is ranked based on some measure of significance and the highest ranked features are selected. Most filter techniques and significance

measures used in malware detection are univariate. Typical examples include inverse document frequency, information gain, gain ratio, Fisher Score (FS), maximum-relevance algorithm, hierarchical approach, and max-relevance algorithm. We describe these briefly below.

—**Document Frequency (DF):** This follows the earlier study by Salton et al. [1975], which a vector space model is proposed to represent a textual document as basically a bag of words. The Term Frequency Inverse Document Frequency (TF-IDF), sometimes denoted simply as Document Frequency (DF), measures to what extent the occurrence of a given word (term) tends to be predominantly on relatively few documents. The TF-IDF is easy to compute.

Let f_{ij} be the frequency of term i in document j. Let N be the number of documents in the collection. Let n_i be the documents (out of the N documents), where term i appeared. The term frequency TF_{ij} is given by:

$$TF_{ij} = \frac{f_{ij}}{max_k f_{kj}}. \tag{1}$$

Essentially, TF_{ij} is simply f_{ij} normalized using the maximum number of occurrences of any term in the given document. The inverse document frequency (IDF) for the i-th term is given as:

$$IDF_i = \log_2 (N/(1 + n_i)). \tag{2}$$

The $TF - IDF$ is the computed as follows:

$$TFIDF = TF_{ij} \times IDF_i. \tag{3}$$

Often times, the terms with the highest TF-IDF scores correspond to those that best capture the topic of the document. With respect to malware detection, file samples can be viewed as analogous to documents, while file features such as byte n-grams and OpCode n-grams can be seen as terms (words). Thus, the TF-IDF has been used as the basis for selecting top features for malware detection for given training sets [Moskovitch et al. 2008a].

—**Information Gain (IG):** The notion of IG was originally introduced in the context of the construction of DTs [Quinlan 1986]. Suppose we divide the samples in the training set based on their values for a given feature, say A. The information gain captures the expected reduction of entropy that will result from the partitioning. Thus, the information gain attempts to quantify the effectiveness of the given attribute (A in this case) in classifying the samples. More formally, let S be a set of items that can be divided into C non-overlapping subsets or subclasses. Let S_c denote the c-th subset. The entropy of S is given by:

$$H(s) = -\sum_{c \in C} \frac{|S_c|}{|S|} \times \log_2 \frac{|S_c|}{|S|}. \tag{4}$$

Let V be the set of possible values for the feature A, and let S_v be the subset of samples with feature value v for the feature A. The IG for attribute A is then given by:

$$IG(S, A) = H(S) - \sum_{v \in V(A)} \frac{|S_v|}{|S|} \times H(S_v). \tag{5}$$

We can observe that for categorical variables, the information gain $IG(S, A)$ defaults to $I(S, A)$, the mutual information between S and A [Murphy 2012]. Based on the byte n-grams, Kolter and Maloof [2006], Zhang et al. [2007], and Masud et al. [2011]

performed malware detection by using the IG as a measure to select the best features from the training set.

—**Information Gain Ratio (IGR):** Given that entropy essentially measures the uncertainty in the input data, the IG tends to favor features or attributes whose values show more diversity (more entropy). The IGR attempts to reduce this bias. The idea is to consider how the given feature partitions the input samples (Equations (6) and (7)). Let $SI(S, A)$ be the entropy of S after dividing the samples according to the given attribute A. Then we have,

$$SI(S, A) = -\sum_{v=1}^{|V|} \frac{|S_v|}{|S|} \times \log_2 \frac{|S_v|}{|S|}. \tag{6}$$

Using SI and IG, we then compute the IGR as follows:

$$IGR(S, A) = \frac{IG(S, A)}{SI(S, A)}. \tag{7}$$

—**FS:** The FS provides a measure of the average inter-class distance when compared with the average intra-class distance, using a given feature. Let $\{x_1, x_2, \ldots, x_n\}$ be the set of input samples with corresponding labels $\{y_1, y_2, \ldots y_n\}$. Suppose we have C available labels (classes), i.e., $y_i \in \{1, 2, \ldots |C|\}$. Let n_i be the number of samples in class i. Further, let μ_i and σ_i be the mean and standard deviation for class i, corresponding to the k-th feature. Denote μ and σ as the mean and standard deviation for the entire dataset. The FS is then given as :

$$F_k = \frac{\sum_{i=1}^{|} C|n_i(\mu_i - \mu)^2}{\sum_{i=1}^{|} C|n_i\sigma_i}. \tag{8}$$

From the above, the FS (F_k) is essentially the ratio of the average inter-class distance to the average intra-class distance. Thus, higher values of F_k imply that members belonging to different classes are further separated using the k-th feature, while members in the same class are closer together. The discrimination ability for the k-th feature increases with increasing values of F_k [Golub et al. 1999]. For the malware detection, the issue is often a two-class problem—positive (malicious) class or negative (benign) class. The FS then reduces to a simple form.

A comparative analysis of the various feature ranking methods (DF, Gain Ratio (GR), and FS) in malware detection was performed by Moskovitch et al. [2008a, 2008c]. Each method was used to select the top-ranked features for later classification and detection of malware. More specifically, in Moskovitch et al. [2008c], they used byte n-gram features on sample sets with 7,688 malware and 22,735 benign files. From the dataset, they extracted 16,777,216; 1,084,793,035; 1,575,804,954; and 1,936,342,220 features, for the 3-gram, 4-gram, 5-gram, and 6-gram model, respectively. Using each of the described feature selection techniques, they ranked the features, and selected the top 50, 100, 200, and 300 features, respectively, for malware classification. Their evaluation results showed that the FS was very accurate using just the top 50 features. This high-level of performance was observed to be stable with an increasing number of features. On the other hand, when the number of features is larger than 200, the accuracy of the DF decreased, while that of GR increased significantly. In Moskovitch et al. [2008a], a similar comparative analysis was reported using OpCode n-gram features rather than byte n-gram features. Their results showed that the FS performed very well (especially when the number of features is small, e.g., top 50 or top 100 features) and the DF was also very accurate for most of the top features.

—**Hierarchical Feature Selection (HFS):** Rather than an approach that performed explicit ranking of the features, in Henchiri and Japkowicz [2006a], a hierarchical feature selection was presented, which performed selection by jointly considering the frequency and coverage of the features. Under this approach, feature selection was performed by considering the *n*-gram features that appeared frequently at rates more than certain thresholds in a specific malware family (i.e., frequency), and appeared, as well, in above a minimal number of malware classes or families (i.e., coverage).

—**Max-Relevance Algorithm (MRA):** To minimize the classification error, feature selection often requires that the target class c has maximal statistical dependency on the selected features. One approach to realize maximal dependency (Max-Dependency) is maximal relevance (Max-Relevance) feature selection [Peng et al. 2005]: selecting the features that have the highest relevance with the target class c. Relevance is usually measured in terms of correlation or mutual information, of which the latter is a popular approach to define dependency between variables. In Ye et al. [2007, 2008], the authors used the Max-Relevance algorithm for feature selection in malware detection. The extracted Windows API calls were ranked using Max-Relevance and then a subset of API calls with the highest relevance to the target class was selected for later classification given a_i, which represents the API with ID i, and the class of file sample c ("0" represents benign file and "1" is for malware). Their mutual information is defined using their appearance frequencies $p(a_i)$, $p(c)$, and $p(a_i, c)$ as

$$I(a_i, c) = \int \int p(a_i, c) log \frac{p(a_i, c)}{p(a_i)p(c)} d(a_i)d(c). \tag{9}$$

Using Equation (9), the top m APIs were ranked in the descending order of $I(a_i, c)$, and the best m individual features correlated to the classes of the file samples were then selected.

6. CLASSIFICATION FOR MALWARE DETECTION

After feature extraction (and feature selection), each file sample is now represented in a feature space using the extracted feature vectors. The feature space is the basis for subsequent analysis stages, such as classification or clustering. Classification is performed by using the feature space generated from a sample training set as the input to a learning algorithm. By analyzing the input vectors, the learning algorithm determines parameters and factors that will be applied on the feature vectors in order to classify them into some pre-defined classes. At the time of testing, a testing set containing a separate collection of malware and benign file samples is used. Similar to the training samples, representative features are extracted from each test sample. Each sample in the testing is then classified as either benign or malicious, using the same set of parameters and factors determined from the training samples. Popular classification algorithms include ANN, DT, Naive Bayes (NB), SVM, and recently Deep Learning (DL). Below, we first briefly describe these popular classification methods, and then we survey how these schemes have been used for the specific task of malware detection.

6.1. Typical Classification Methods and Their Applications in Malware Detection

6.1.1. Typical Classification Methods.

DT: In *DT learning* [Quinlan 1993], the classifier is represented as a tree, where the internal nodes represent the input variables. Each internal node has an outgoing edge corresponding to each possible value of the variable represented at the node. A path from the root to a leaf node captures the sequence of feature values for each given variable on the path. Thus, a given leaf node represents the class label (final

classification decision), given the feature values of the nodes encountered on the path from the root node to the leaf node. In practice, a greedy heuristic scheme is often used to generate the tree using the input data, for instance, by splitting the features or variables using the expected information gain. The problem of over-fitting is handled by pruning the tree. With the principled splitting of the nodes based on the feature values, the DT can be easily represented or implemented in the form of simple rules. The studies, such as Kolter and Maloof [2004], Henchiri and Japkowicz [2006a], Elovici et al. [2007], Moskovitch et al. [2008a], Siddiqui et al. [2009], Tian et al. [2010], and Santos et al. [2013], have applied DTe for malware detection.

An improved technique based on DTs is *Random Forests* [Breiman 2001]. This has become very popular in recent years given its performance on different classification problems. The random forest algorithm grows a collection of DTs, called a forest, and uses these for classifying a data point into one of the classes. Two types of randomness, bootstrap sampling and random selection of input variables, are used in the algorithm to ensure that the classification trees grown in the forest are dissimilar and uncorrelated from each other. Growing a forest of trees, and using randomness in building each classification tree in the forest, leads to better predictions as compared to a single classification tree, and helps to make the algorithm robust to noise in the data set. Compared to a single DT, the random forest approach decreases the variance in the prediction results and has been used for malware detection [Siddiqui et al. 2009; Tian et al. 2010; Islam et al. 2013].

Naive Bayes Classifier (NBC): *NBC* [John and Langley 1995] is based on the Bayes rule assuming that the attributes are conditionally independent. With the Bayes rule, given an observation, NBC computes the class conditional probabilities using the joint probabilities of sample observations and classes. It assigns an instance x with attribute values $(A_1 = v_1, A_2 = v_2, \ldots, A_m = v_m)$ to class C_i with the maximum posterior probability $Prob(C_i|(v_1, v_2, \ldots, v_m))$ for all i.

$$Prob(C_i|(v_1, v_2, \ldots, v_m)) = \frac{P((v_1, v_2, \ldots, v_m)|C_i) \times P(C_i)}{P(v_1, v_2, \ldots, v_m)}. \tag{10}$$

$$\begin{aligned} Prob((v_1, v_2, \ldots, v_m)|C_i) &= P(A_1 = v_1|C_i) \times P(A_2 = v_2|C_i) \times \cdots \times P(A_m = v_m|C_i) \\ &= \prod_{h=1}^{m} P(A_h = v_h|C_i). \end{aligned} \tag{11}$$

NBC is simple but has been empirically shown to perform well in a variety of application domains [Domingos and Pazzani 1997]. In malware detection, there have been several researches using NBC for model construction [Schultz et al. 2001; Kolter and Maloof 2004; Henchiri and Japkowicz 2006a; Moskovitch et al. 2008c; Firdausi et al. 2010].

Bayesian network (BN), also called belief network, is a graphical model representing a set of variables and their causal influences [Pearl 1987]. It is a graphical structure which enables the explicit representation of dependencies among variables. Different from NB, the variables in BN are not assumed to be conditionally independent. A standard Bayesian network consists of two components: (1) a directed acyclic graph where random variables are represented as nodes, and the edges represent probabilistic dependence between corresponding variables, and (2) Conditional Probability Tables (CPT) for the variables. BN has also been used in malware detection [Elovici et al. 2007].

K-Nearest Neighbor (kNN): The kNN algorithm [Fix and Jr. 1951; Cover and Hart 1967; Aha et al. 1991] is among the oldest and simplest classification schemes. kNN

uses instance-based reasoning. The basic philosophy is that two objects in the same class have some kind of similarity that can be measured using some distance metric on the variables. An instance, with an unknown class, is thus classified into the same category as its first nearest neighbor, or the winning class by the votes of its k nearest neighbors, where k is usually a small odd number, e.g., 1, 3, 5, or 7. The number of neighbors, k is often chosen by empirical cross-validation. Nearest neighbors are found by a distance or dissimilarity measure. This is usually computed by considering the objects as points or position vectors in a multidimensional feature space. The Euclidean distance and its generalized form, the L_q norm or Minkowski distance, are popular distance measures often used with kNN. More complicated distance measures, such as the Mahalanobis distance, and locally adaptive distance measures, have also been used. The same approach can be applied for regression or prediction. Here, given an object or sample, to predict the value for a given property or attribute of the object, we simply use the average of the attribute value of its k nearest neighbors. In some cases, the contribution of the neighbors are weighted, for instance by their distance, to ensure that the nearer neighbors provide more influence on the predicted value when compared with the more remote neighbors. kNN, as a nonparametric model, does not assume any parametric form for the underlying distributions of the random variables. With this flexibility, it is usually a good classifier for many situations where the joint distribution is hard to model or unknown. This is often the case for the high dimensional datasets in malware classification [Firdausi et al. 2010; Santos et al. 2013]. Recently, the Random kNN (RkNN), a variation of the kNN method, specially designed for classification of high dimensional datasets was introduced in Li et al. [2011, 2014].

Artificial Neural Network (ANN): ANN [Bishop 1995] is a computational model inspired by biological nervous systems (e.g., human brain), which is presented as a system of interconnected "neurons" that can compute values from inputs. Roughly speaking, an ANN is a set of connected input/output units in which each connection has an associated weight. During the training phase, the network adjusts the weights to correctly predict the class label of the input tuples. Back-propagation [Werbos 1974] is the most popular neural network learning algorithm. Equation (12) illustrates the output computation of a two-layered ANN. Note f is the activation function, x is the input vector, w_{ij} is the weight of a hidden neuron, θ_i is a weight in the output neuron, and b_i and b_0 are biases.

$$O(x) = f\left[\sum_i \theta_i f\left(\sum_j w_{ij}x_j + b_i + b_0\right)\right]. \tag{12}$$

In Elovici et al. [2007] and Moskovitch et al. [2008a, 2008c], the authors have applied ANN for malware detection.

SVM: SVM [Joachims 1998] is a binary classifier which searches for a hyperplane that separates the two classes with the largest margin. Note that the margin is defined as the distance between the hyperplane and its closest point within each class. Later, extensions for handling multi-class classification were developed. SVM has the advantage of handling high-dimensional data sets (e.g., with large feature representation) without over-fitting and has achieved state-of-the-art results in binary classification problems. A key technique in SVM is the kernel function which transforms the original data into a high dimensional feature space for better separation. Note that a linear boundary in the feature space corresponds to a nonlinear boundary in the original space. The output of a linear SVM is $u = w \times x - b$, where x is the input vector, w is the normal weight vector to the hyperplane, and b is the bias. Maximizing the margin can

be formulated as the optimization problem below:

$$\text{minimize } \frac{1}{2}\|w\|^2, \text{ subject to } y_i(w \cdot x_i + b) \geq 1, \forall i. \tag{13}$$

Note that for the i-th training example, x_i, y_i is the correct output. Intuitively, classifiers with a large margin will have low expected risks and, hence, good generalization. In malware detection, SVMs also show good performance in discriminating malware from benign files [Abou-Assaleh et al. 2004; Wang et al. 2006b; Ye et al. 2009, 2009c; Firdausi et al. 2010; Anderson et al. 2012].

Associative Classifier (AC): *Associative Classifier* (AC) utilizes the association mining techniques to construct classifiers [Thabtah 2007]. It can be considered as a special case of association rule discovery where only the class attribute can appear on the right-hand side (i.e., the consequent) of a rule. For example, in an association rule $X \rightarrow Y$, Y must be a class attribute. One of the main advantages of associative classifier is that it makes the model interpretable because the output of an AC algorithm can be easily represented as simple *if-then* rules. Furthermore, AC classifiers can be incrementally updated or tuned [Thabtah 2007]. AC has also been applied in malware detection [Ye et al. 2007, 2009a, 2009b].

DL: DL, a new frontier in data mining and machine learning, is starting to be leveraged in industrial and academic research for different applications (e.g., Computer Vision). A multilayer DL architecture is of superior ability in feature learning. Furthermore, DL architectures overcome the learning difficulty through layerwise pre-training, i.e. pre-training multiple layers of feature detectors from the lowest level to the highest level to construct the final classification model. Typical DL models [Bengio et al. 2007; Bengio 2009] include Stacked AutoEncoders (SAEs), Deep Belief Networks with Restricted Boltzmann Machine, Convolutional Neural Networks, and the like.

In a recent comprehensive study of the performance of various classification algorithms, Fernández-Delgado et al. [2014] analyzed 179 classifiers from 17 families, including the popular ones listed above, using 121 datasets, and concluded that the random forest family of classification schemes produced the overall best performance, followed closely by SVMs, using the Gaussian kernel. Their results point to the overall performance of the algorithms on general classification problems. Below, we consider how these classification schemes have performed on the specific problem of malware detection.

6.1.2. Malware Detection by Applying Typical Classification Methods. Schultz et al. first introduced the idea of applying data mining methods for different types of malware detection based on three static features: Dynamic Link Library (DLL) call information, strings, and byte sequences [Schultz et al. 2001]. A rule induction algorithm Ripper [Cohen 1995] was applied to discover patterns based on the DLL call's data set, and NB was used to build the classifiers resting on the strings and 2-byte sequences. Based on their data collection consisting of 4,266 files (3,265 malicious and 1,001 benign), the NB algorithm achieved the highest classification performance (i.e., with 97.11% accuracy). The authors concluded that the malware detection rate using the data mining method was twice of the signature-based method on their data collection.

In Henchiri and Japkowicz [2006a], the authors extracted 16-byte sequences from their data collection consisting of 3,000 file samples, 1,512 of which were malware and 1,488 of which were benign files. They applied different kinds of classifiers on the extracted feature set, including ID3 and J48 DTs, NB, and SVM, and obtained better results than those with traditional feature selection methods as shown in Schultz et al. [2001].

Note that associative classifiers can discover interesting relationships among file features and malware/benign file classes. Ye et al. proposed Intelligent Malware Detection System (IMDS) for malware detection using Object Oriented Association (OOA) mining based on the extracted Windows API calls [Ye et al. 2007]. In order to further improve the efficiency and scalability for malware detection, the authors further proposed a post-processing method, named CIDCFP [Ye et al. 2009b], for classification model construction. To address the imbalance class distribution problem, they also further developed the Hierarchical Associative Classifier (HAC) [Ye et al. 2009a] for malware detection.

Moskovitch et al. [2008c] compared different classifiers based on the byte n-grams extracted from 30,000 sample collection: ANN, DT, Boosted Decision Trees (BDT), NB, Boosted Naive Bayes (BNB), and three types of SVMs. Their experiments were conducted on various parameter settings including the feature selection method (e.g., Document Frequency, FS, or Gain Ratio), the size of n-grams (e.g., 3, 4, 5, or 6), the number of top selected features (e.g., 50, 100, 200, or 300), and the term representation method (e.g., TF or TF-IDF). The best representation setting identified in the experiments was the top 300 5-gram terms (represented using TF) based on the FS. The classifiers were then compared based on the best representation setting. The classification results showed that the ANN, DT, and BDT outperformed the BDT, BNB, NB, and SVM classifiers in malware detection based on their collected sample set and experimental settings. Similar results were also given in Moskovitch et al. [2008a] based on the n-gram OpCode sequence features.

Many researchers used *dynamic analysis techniques* for feature extraction to improve the effectiveness and efficiency of malware detection. By running in a virtual environment, Tian et al. [2010] extracted API call sequences from executables. Several classifiers available in WEKA [Hall et al. 2009] were used for malware detection, including SVM, Random Forest, DT, and Instance-based Classifier. A data set consisting of 1,368 malware and 456 benign files is used in the experiments. Their work has over 97% accuracy in the experiments.

In Firdausi et al. [2010], the authors extracted the behaviors of malware samples in a sandbox environment using Anubis [2010]. The extracted features were pre-processed into sparse vectors for five different classifiers: NB, J48 DT, kNN, Multilayer Perceptron Neural Network (MLP), and SVM. A small data collection of 220 malware and 250 benign samples was used in the experiments. Their experimental results indicated that the overall best performance was achieved by J48 DT, with a precision of 97.3%, a recall of 95.9%, an accuracy of 96.8%, and a false positive rate of 2.4%.

A detection method based on instruction traces was proposed by Anderson et al. [2011]. To dynamically collect the traces, the modified Ether malware analysis framework was used [Firdausi et al. 2010]. The similarity matrix was constructed using graph kernels between instances and the transition probabilities in the Markov chain was estimated using 2-grams. Two distinct similarity measures, a Gaussian kernel, and a spectral kernel were used to construct the kernel matrix. Note that the first kernel measures the local similarity between graph edges, and the second kernel measures the global similarity between the graphs. With the kernel functions, SVM was then trained to classify the testing sample set. Based on the data collection with 1,615 malware and 615 benign files, the performance of their proposed multiple kernel learning method was excellent. However, the computation complexity of their proposed work is quite high, thus limiting its usage in real applications.

Note that both static analysis and dynamic analysis have limitations. Neither of them is sufficient for classifying malicious file samples efficiently and accurately. Therefore, researchers have used hybrid techniques which incorporate *both dynamic and static features* to perform malware detection.

Wang et al. [2006b] developed a surveillance spyware detection system that used both static and dynamic extracted features. Static features were the DLLs and APIs extracted from the file sample set, while dynamic features were the modifications upon system files, registries, and network activities. Based on their data collection consisting of 407 spyware and 740 benign programs, using the information gain method, they selected 81 features from the ranked 790 features, 67 of which were static and 14 of which were dynamic. They then built an SVM classifier for detection. The overall accuracy of their developed system reached 96.43% by using 10-fold cross validation. They also claimed that the system's performance was better than some well-known anti-virus applications in spyware detection.

Anderson et al. [2012] conducted their researches for malware detection based on multiple feature extractions: 2-gram byte sequences, control flow graph, disassembled OpCodes, dynamic instruction traces, miscellaneous file information, and system call traces. Kernels based on the Markov chain graphs were used for the 2-gram byte sequences, disassembled OpCodes, dynamic instruction traces, and system call traces.

A graphlet kernel and a standard Gaussian kernel were used for the control flow graph and the file information feature vector, respectively. A weighted combination of the data sources was learned using multiple kernel learning. Finally, SVM was trained as the classifier. Their proposed method achieved an accuracy of 98.07% based the sample set of 780 malware and 776 benign files.

In Santos et al. [2013], the authors developed a hybrid malware detector by utilizing both static and dynamic features. Static analysis extracts the features by modeling executables as sequences of operational codes. Dynamic analysis extracts features by monitoring operations, system calls, and raised exceptions. The experimental results on the sample collection of 1,000 malware and 1,000 benign files demonstrated that this hybrid approach enhanced the classification performance.

Islam et al. [2013] also classified the executables into malware and benign files by using both static (e.g., function length frequency and printable sting) and dynamic features (e.g., API function names along with parameters). Based on the sample collection of 2,939 executable files (2,398 malware and 541 benign files), the experiments were conducted on classifiers including SVM, Random Forest, DT, and Instance-based Classifier. Their experimental results showed that integrating features improved the classification performance and meta-Random Forest performed best for all the cases.

In Saxe and Berlin [2015], based on the features of byte entropy histogram, PE import information, and numerical PE fields, the authors applied deep neural network and Bayesian calibration model for malware detection. Their developed system achieved a 95% detection rate at a 0.1% false positive rate, based on more than 400,000 software binaries sourced directly from their customers and internal malware databases. Hardy et al. also applied DL architecture, which used the SAEs model for malware detection based on Windows API calls [Hardy et al. 2016].

6.2. Ensembles of Classifiers and Their Applications in Malware Detection

Ensemble methods construct a set of base (individual) classifiers and then classify new samples by taking a (weighted) vote of the predictions of base classifiers [Dietterich 2000]. Ensemble methods are often used to overcome instability and improve prediction performance. A typical ensemble framework contains: (1) individual base classifiers whose error rates should be below 0.5 and are at least somewhat uncorrelated [Dietterich 2000], and (2) conclusion combiner, which is responsible for combining the classification results of individual base classifiers. Many ensemble methods have been proposed in machine learning literature [Dietterich 1997]: *Bagging* [Breiman 1996] and *Boosting* [Freund and Schapire 1997] are two popular ensemble learning algorithms [Oza and Russell 2001]. Bagging generates K bootstrap samples from the training data and builds base classifiers for each bootstrap sample [Breiman 1996]. A

bootstrap sample is obtained by sampling (with replacement) the original training set uniformly. Adaboost [Freund and Schapire 1997] dynamically changed the weights of each instance being selected based on the classification results in the previous round. For conclusion combination of the base classifiers, there are two main methods [Rokach 2010]: *weighting* methods (e.g., majority voting, Dempster-Shafer theory of evidence) and *meta-learning* methods (e.g., stacking). Weighting methods are useful if the base classifiers have a similar performance, while meta-learning methods are often useful when based classifiers have a disparate performance (e.g., different classifiers may consistently misclassify different instances) [Rokach 2010]. In malware detection, ensembles of classifiers are also employed to further improve the prediction accuracy (e.g., the predication of different malware families).

In Kolter and Maloof [2004], based on the extracted binary *n*-gram feature set, the authors applied different classification methods for malware detection, including DT and their boosted versions. Their experimental results on a collected data set with 1,971 benign and 1,651 malicious executables demonstrated that boosted DTs outperformed other classifiers.

Abou-Assaleh et al. [2004] used the Dempster-Shafer theory of evidence to combine DT, SVM, and kNN classifiers. In their approach, class profiles of different lengths are defined by the number of the most frequent *n*-grams, using various *n*-gram sizes.

Elovici et al. [2007] extracted two types of static features: 5-grams byte sequences and PE header from 7,694 malware and 22,736 benign files for malware detection. For the 5-grams byte sequences data set, they first applied TF-IDF to choose the top 5,500 features and then used FS to select the top 300 5-grams out of the 5,500 5-grams. ANNs, BNs, and DTs were used to induce the individual base classifiers trained on the selected 5-grams byte sequences and PE head features. The ensemble of the base classifiers using a simple voting algorithm is shown to be superior as it achieved the best true positive rate along with a minimum false positive rate.

In Masud et al. [2007], based on the data collection of 3,887 executables, 1,967 of which are benign files and 1,920 are malware, the authors created a hybrid feature set using assembly instruction sequences (obtained from the disassembled executables), binary n-grams from the executable, and DLL call information (extracted from the program headers). Resting on the extracted feature set, they first used Information Gain for feature selection and then applied SVM and boosted DTs for the classification. In Masud et al. [2008], using the same feature extraction methods for malware detection, the authors further extended their work by using different classifiers (e.g., DTs, NB, SVM), boosted DTs, and boosted NB for comparisons.

Siddiqui et al. [2009] used a variable length instruction sequence for detecting worms in the wild. Before disassembling the files, they detected compilers and packers. Rare items (e.g., sequences whose frequency of occurrence is less than 10%) were removed. DT, random forest, and bagging were used for classification. Their experimental results showed that random forest performed best based on their data collection of 2,774 samples including 1,444 worms and 1,330 benign files.

In Ye et al. [2009], based on the "interpretable" strings, the authors used SVM ensemble with bagging to predict the malware types. In Ye et al. [2009c], they further developed an Intelligent File Scoring System (IFSS) using an ensemble of heterogeneous base classifiers (e.g., SVM and associative classifier) with different feature representations (e.g., Windows API calls and strings) on dynamic training sets.

6.3. Other Methods for Malware Detection

In addition to the typical classification methods introduced above, some other existing researches also adopted different data mining approaches for malware detection based on various kinds of feature representations.

Frequent pattern mining has been used in Christodorescu et al. [2007] to identify the differences between malware samples and benign executables. Note that the differences may be used as specifications for characterizing malicious behaviors (malspecs). Temporal event pattern mining can also be applied in malware detection to detect sequential/temporal relationships among different features [Li 2015; Zeng et al. 2017]. Kolbitsch et al. [2009] also applied supervised learning for malware detection. But they extracted behavior graphs out of known malware samples from a given family and used a specialized subgraph matching algorithm to detect similar malware, without resorting to much data mining or machine learning techniques. HOLMES [Fredrikson et al. 2010] detected malware families using discriminative specifications by combining the concept analysis and frequent subgraph mining. However, it had a limited coverage of possible program paths during dynamic analysis. Combining HOLMES with other classification methods based on different feature representations (i.e., ensemble classification) would increase the malware detection rate. It has been pointed out that, for malware detection, a large amount of labeled executables (both malicious and benign) is often required for supervised learning (classification) [Santos et al. 2011b]. Santos et al. proposed a semi-supervised learning approach for malware detection, which was designed to build a classifier using both labeled and unlabeled file samples. Based on the extracted *n*-gram distributions, a semi-supervised algorithm, Learning with Local and Global Consistency (LLGC), was applied for model construction. The semi-supervised algorithm was able to learn a good solution by exploiting the intrinsic structure displayed by both labeled and unlabeled file samples. The main contribution of their research was to reduce required labeled file samples without sacrificing the detection precision or accuracy much. However, it was shown that previous supervised learning approaches presented in Kolter and Maloof [2004] and Moskovitch et al. [2008c] obtained better results (above 90% of accuracy) than the proposed semi-supervised learning approach. Further, in Santos et al. [2011a], a collective learning approach was proposed to malware detection by utilizing partially-labeled data.

Besides file contents, file-to-machine relation graphs [Chau et al. 2011] and file-to-file relation graphs [Ye et al. 2011; Tamersoy et al. 2014] were also used as the features for malware detection. In Chau et al. [2011] and Tamersoy et al. [2014], based on the file-to-machine and and file-to-file relation graphs, the authors used Belief Propagation (BP) algorithm (a promising method for solving inference problems over graphs) for malware detection. Note that BP has been successfully used in many domains (e.g., computer vision, coding theory) [Yedidia et al. 2001]. In Ye et al. [2011], the authors studied how file relations could help improve malware detection. They developed "Valkyrie," a file verdict system based on a semi-parametric classification model by integrating the file content and file relations for malware detection. Their experiments were conducted based on a large dataset from Comodo company. The dataset included 30,950 malware samples (i.e., 225,830 benign files and 434,870 unknown files). Karampatziakis et al. [2013] built regression classifiers based on graphs induced by file relationships for malware detection. They showed that the system's detection accuracy could be significantly improved using the proposed method.

Table III summarizes some typical classification methods used for malware detection.

7. CLUSTERING FOR MALWARE DETECTION

The classification methods typically require a large number of labeled samples. In recent years, there have been research initiatives in automatic malware categorization using unsupervised techniques (i.e., clustering techniques). *Clustering*, a form of unsupervised learning, is the process of partitioning a given data set into groups (i.e., clusters) based on the pre-defined distance measures, such that the data points in a cluster should be close to each other and data points in different clusters are far away

Table III. Summary of Typical Classification Methods Used in Malware Detection

Survey	Classification Methods	Description
Schultz et al. [2001]	Ripper, NB	Based on the string feature set, the NB algorithm performed best with the accuracy of 97.11%.
Kolter and Maloof [2004]	SVM, NB, DT, and their boosted versions	Using binary *n*-grams, based on their collected data set with 1,971 benign and 1,651 malicious executables, their experimental results indicated that boosted DTs performed best in malware detection.
Abou-Assaleh et al. [2004]	SVM, DT, kNN classifiers, Dempster-Shafer theory of evidence	Based on a small sample collection of 65 malware and benign files, using binary *n*-grams of various lengths, the proposed method achieved 98% detection accuracy.
Henchiri and Japkowicz [2006b]	NB, DTs, SVM, and Sequential Minimal Optimization (SMO)	Based on the 16-byte sequences, the obtained results were better than those obtained through traditional feature selection.
Wang et al. [2006b]	SVM	Using both static and dynamic extracted features, based on their data collection consisting of 407 spyware and 740 benign programs, when 10-fold cross validation was considered, its overall accuracy reached 96.43%.
Ye et al. [2007, 2009a, 2009b]	NB, DTs, SVM, Associative Classification	Based on the Windows API call features, the associative classifier outperformed other classifiers.
Elovici et al. [2007]	ANN, DTs, BNs, Ensemble of Classifiers	Based on the 5-grams byte sequences and PE header, the ensemble of the base classifiers using simple voting algorithm outperformed each individual classifier.
Masud et al. [2007, 2008]	SVM, DT, NB, BDT, and BNB	Based on the hybrid feature sets, the results from Boosted J48 were almost the same as SVM.
Moskovitch et al. [2008a, 2008c]	ANN, DTs, BDT, NB, BNB, and SVMs	The results indicated that the BDT, DT, and ANN outperformed the NB, BDT, BNB, and SVM classifiers, based on the *n*-gram OpCode sequences and byte *n*-grams.
Siddiqui et al. [2009]	DT, Random Forest, and Bagging	Their experimental results showed that Random Forest performed best based on the variable length instruction sequences extracted from 2,774 samples including 1,444 worms and 1,330 benign files.
Ye et al. [2009]	DT, NB, SVM, Bagging	The experimental results showed that the ensemble of SVMs with bagging performed best based on the extracted interpretable strings.
Ye et al. [2009c]	Associative Classifier, SVM, Ensemble of heterogeneous base-level classifiers	The ensemble of heterogeneous base-level classifiers using different learning methods (associative classifier and SVM), with different feature representations (Windows API calls and strings) on dynamic training sets performed best in real application.
Tian et al. [2010]	SVM, DT, Random Forest, Instance-based Classifier	They used the API call sequences dynamically extracted from 1,368 malware and 456 benign files to demonstrate their work and achieved an accuracy of over 97%.
Firdausi et al. [2010]	kNN, NB, DT, SVM, Multilayer Perceptron Neural Network (MLP)	The obtained results based on the dynamically extracted behaviors depicted that overall best performance was achieved by J48 DT.

(Continued)

Table III. Continued

Survey	Classification Methods	Description
Anderson et al. [2011]	Markov Chain	The performance of multiple kernel learning method was demonstrated and shown to be excellent based on the graphs constructed from dynamically collected instruction traces.
Santos et al. [2011a, 2011b]	Semi-supervised algorithm LLGC , collective learning approach	The main contribution of the research was to reduce the number of required labeled file samples while maintaining high precision.
Chau et al. [2011]	BP	The developed system was evaluated on billion-node file-to-machine graphs and attained a high true positive rate of 87% in detecting malware.
Ye et al. [2011]	Semi-parametric classifier model	By combining file content and file relations, the experimental results demonstrated that the accuracy and efficiency of their developed system outperformed popular anti-malware software tools and alternate data-mining-based detection systems.
Anderson et al. [2012]	SVM	By combining both static and dynamic analysis, it was tested on a dataset of 780 malware and 776 benign instances giving an accuracy of 98.07%.
Santos et al. [2013]	DT, kNN, BN, and SVM	It has been found that the hybrid approach enhanced the performance of both approaches when run separately, based on the static and dynamic analysis.
Islam et al. [2013]	DT, Random Forest, SVM, and Instance-based Classifier	By combining both static and dynamic analysis, the obtained results showed that meta-Random Forest performed best.
Karampatziakis et al. [2013]	Regression Classifier	Based on the graphs induced by file relationships, the system's detection accuracy could be significantly improved using the proposed method, particularly with low false positive rates.
Tamersoy et al. [2014]	BP	Based on the file-to-file relation graphs, the developed system attained early labeling of 99% of benign files and 79% of malicious files.
Saxe and Berlin [2015]	Deep Neural Network and Bayesian Calibration Model	Using the statically extracted features, their system achieves a 95% detection rate at 0.1% false positive rate, based on more than 400,000 software binaries.
Hardy et al. [2016]	DL Architecture using SAEs	Based on the extracted Windows API calls, the developed deep learning framework outperformed ANN, SVM, NB, and DT in malware detection.

from each other [Xu and Wunsch 2005]. In malware detection, a cluster is a group of file samples sharing some common traits while being "dissimilar" to the malware samples from different clusters. *Partitioning* and *Hierarchical* clustering are two common types of clustering methods with different characteristics [Xu and Wunsch 2005]. Hierarchical clustering methods can handle data sets with irregular cluster structures, while partitioning clustering (e.g., *K-means*) is generally effective when the clusters have a globular shape. The choice of clustering algorithms in malware detection is largely based on the extracted features and their underlying feature distributions.

In Karim et al. [2005], the authors created a malware phylogeny using permutation of codes. In particular, *n*-grams were mutated to generate *n*-perms and the similarity scores were computed by combining TF-IDF weighting and the cosine similarity. The phylogeny model was obtained by clustering with these similarity scores.

Lee and Mody [2006] applied *K*-medoids clustering algorithm on the collected sequences of system call events to group malware families. All the samples were executed in virtual environments. The system calls and their associated arguments were monitored. Then, a behavioral profile was created for each file sample based on its interaction with the system resources (e.g., files, network activities, and registry keys). The similarities between samples were measured by their profiles and then *K*-medoids clustering was used to group different samples into different clusters. Note for prediction, new and unknown samples were assigned to the clusters with the closest "medoids."

Bailey et al. [2007] extracted system call traces as features and employed Normalized Compression Distance (NCD) to characterize malware similarities. They pointed out that the characterizations of the malware were not concise in their semantics and were also not consistent across different AV products. They first defined malware behaviors as non-transient state changes on the system, and then applied hierarchical clustering algorithms for grouping malware samples. The activities (such as files written, processes created, and network connections) were used to create the behavioral fingerprint of malware.

Bayer et al. [2009] applied Locality Sensitive Hashing (LSH) [Indyk and Motwani 1998] on the behavior profiles (more concise than system call traces) to achieve efficient and scalable malware clustering. Their proposed technique makes use of Anubis [2010] to generate execution traces. The clustering algorithm based on LSH provided efficient support for finding approximate nearest neighbors.

In Ye et al. [2010], using the function-based instruction sequences and instruction frequency, the authors developed an Automatic Malware Categorization System (AMCS) based on cluster ensemble to automatically group malware samples into families. To construct the cluster ensemble, the base clustering algorithms included a hybrid hierarchical clustering algorithm (an integrated version of the hierarchical clustering and *K*-medoids clustering algorithms) and a weighted subspace *K*-medoids algorithm. They also proposed a principled cluster ensemble framework for clustering combination, where the instance-level constraints can be easily incorporated.

8. ADDITIONAL ISSUES AND CHALLENGES

Previous studies proved that data mining techniques have been successfully used in malware detection and in the anti-malware industry [Ye et al. 2009c, 2010, 2011; Chau et al. 2011; Stokes et al. 2012; Tamersoy et al. 2014]. However, there are still many additional issues that need further investigation.

—**Evaluation and use of the detection results:** Though applying data mining techniques, like classification/clustering methods, can detect malware from the unknown file sample collection, the verification of the potentially malicious files is one of the challenging issues in real application. This is because the inspection of these potentially malicious files always requires the knowledge of domain experts and the manual inspection is always time-consuming.

—**Incremental learning:** Training a classifier using an historical file sample collection (containing both benign and malicious samples) is able to detect newly released malware. However, new malware samples are constantly produced on a daily basis and malware techniques are continuously evolving. To account for the temporal trends of malware writing, data-mining-based malware detection systems need to take the most recent file sample collection into consideration. In other words, to make the classifier(s) remain effective, the training sets should dynamically change to include new samples while retaining the main properties of historical data collection. Therefore, incremental learning is one of the issues of data-mining-based malware detection systems.

—**Active learning:** To further improve the detection accuracy, selecting representative sample(s) from large unknown file collections for labeling is also very important. For example, before being detected, the newly released Trojan-Downloader and its related trojans are collected from the clients and marked as unknown. If we can recognize the Trojan-Downloader and have it labeled, then, based on the extracted features and using classification/clustering methods, its related trojans can be correctly detected. Active learning, as an effective paradigm to address the data scarcity problem, optimize the learning benefit from domain experts' feedback, and reduce the cost of acquiring labeled examples for supervised learning, has been intensively studied in recent years [Lewis and Gale 1994; Muslea et al. 2006; Seung et al. 1992; Nguyen and Smeulders 2004]. In malware detection, there is limited research using active learning to select a representative sample(s) from the large file sample collection. For instance, to further improve the performance of a classifier, Moskovitch et al. [2008b] applied active learning for the most important file selection in a stream data.
—**Prediction of malware prevalence:** Except for detecting malware from the unknown file collection, predicting the trend of malware prevalence is also very important. However, there is little research on the prediction of malware prevalence.
—**Adversarial Learning:** In malware detection, using the data mining approaches opens the possibility for the malware attachers to come up with ways to "mistrain" the classifiers (e.g., by changing the data distribution or feature importance) [Dalvi et al. 2004; Venkataraman et al. 2008]. Thus, questions arise as to how to develop techniques that are robust and secure in adversarial scenarios.

9. THE TREND OF MALWARE DEVELOPMENT

2013 was called the the year of the mega breach [Symantec 2014a], and severe vulnerabilities such as Shellshock and Heartbleed [Kingsoft 2015] in 2014 also showed that the Internet security still faces serious threats. The arms race between malware attackers and protectors is likely to intensify: (1) The volume of malware threats in PC platform is still very large; (2) With an exponential growth in the number of mobile devices, more and more attacks are focusing on the smart devices explicitly; (3) With the development of the Internet of Things (IoT), people will have increased connectivity through their gadgets, devices, and machines. However, with this connectivity comes the potential for a whole new range of security risks. Will the IoT usher in a new wave of security attacks? (4) As cloud service becomes more and more popular, security of the cloud will also be a major issue; (5) With the development of malware, what role will data mining/machine learning play in malware detection? In this section, we forecast the trend of malware development.

—**The volume of malware threats in PC platform is still very large:** According to the report from Qihoo, which is the biggest Internet security company in China, there were 324 million malware samples detected in 2014 [Qihoo 2015]. Compared with 2013, though the number of malware collection in PC platform decreased in 2014, the online banking malware volume steadily rose throughout the first half of 2014 [Kingsoft 2015], which means there is still a flourishing underground economy based on malware. Driven by considerable economic benefits, the black chain of trojans won't disappear in coming years and the growing threats posed by the trojans will continue.
—**Mobile devices turn to be even more attractive targets for malware attackers:** Mobile platforms are more and more popular. In recent years, there has been an exponential growth in the number of mobile device users around the world: It is estimated that 77.7% of all devices connected to the Internet will be smart phones

in 2019, leaving PCs falling behind at 4.8% [Hou et al. 2016]. Mobile devices become even more prevalent and valuable as many retail stores and mobile carriers transition to mobile payments [Symantec 2014b]. The increasing use of Google Wallet and other similar payment modes also acts as a catalyst for mobile payment to become mainstream. Cyber criminals are looking to monetize their exploits as simply and efficiently as possible [Garnaeva et al. 2014]. Therefore, more and more crime packs and tools are focusing on the smart devices explicitly. At the moment, the majority of malware for non-Windows platforms is targeted at Android, with the vast majority of it posing as legitimate applications and tricking the user into installing their nasty code [Lyne 2014]. The number of Andriod malware has exponentially increased in recent years [Garnaeva et al. 2014]. In 2014, there were 4,643,582 malicious installation packets detected by Kaspersky Lab, 53% of which were trojans designed to steal the users' money [Garnaeva et al. 2014]. In 2015, there were 280 million smart phones infected by Andriod malware [Kingsoft 2015]. This trend will undoubtedly continue [Tam et al. 2017].

—**Attacks on the IoT move from proof-of-concept to mainstream risks:** The IoT means that consumer products (e.g., from TVs to refrigerators) are digitally connected [WEBSENSE 2014]. We are bound to see greater adoption of smart devices like smart cameras and TVs in the next few years. Though the popularity of the embedded and small devices continues, many of these devices are deployed without considering the Internet security. As factors like market pressure push device manufacturers to launch more and more smart devices sans security in mind to meet the rising demand, so will attackers increasingly find vulnerabilities to exploit for their own gain [TrendLabs 2014]. When attackers begin to better understand the IoT ecosystem, they will employ scarier tactics, such as ransomware and scareware, to extort money from or blackmail device users [TrendLabs 2014]. For instance, they can hold smart car drivers hostage until they pay up.

—**Cloud security will also be a major issue:** In the coming years, the cloud will host more and more data. As the cloud gains more data, organizations facilitate data access with various types of devices (e.g., desktop, tablet, or mobile) [WEBSENSE 2014]. For consumers, an infinite amount of personal information will also be hosted remotely in the cloud. As a result, cyber-criminals will attack the mobile devices to gain data access in the cloud [WEBSENSE 2014]. Therefore, the research issue about the access right and control and the protection of the private data in the cloud will be a hot topic in the years to come.

—**In the fight against cyber crime, data mining/machine learning will be a game changer:** In malware detection, staying "proactive" against threats is a critical need. Data mining/machine learning methods are able to help security vendors stay one step ahead of cyber criminals instead of just reacting to them. The ability of predicting cyber attacks will improve the detection rates and may be the key for reversing the trend on cyber crime.

10. CONCLUSION

In recent years, a few research efforts have been conducted on surveys of data-mining-based malware detection methods [Idika and Mathur 2007; Siddiqui et al. 2008; Shabtai et al. 2009; Egele et al. 2012; Mathur and Hiranwal 2013; Bazrafshan et al. 2013; Damshenas et al. 2013; Gandotra et al. 2014]. In Idika and Mathur [2007] and Damshenas et al. [2013], the authors reviewed the malware propagation, analysis and detection; while in Siddiqui et al. [2008], Mathur and Hiranwal [2013], Bazrafshan et al. [2013], and Gandotra et al. [2014], the researchers surveyed the feature representation and classification methods for malware detection. Shabtai et al. conducted a comprehensive study on static features using machine learning classifiers for malware

detection [Shabtai et al. 2009], while Egele et al. [2012] surveyed automated dynamic malware analysis techniques and tools. In this article, we not only overview the development of malware and anti-malware industry and present the industrial needs on malware detection, but also provide a comprehensive study on data-mining-based methods for malware detection based on both static and dynamic representations as well as other novel features. Furthermore, we also discuss the additional issues and challenges of malware detection using data mining techniques and finally forecast the trends of malware development.

Due to the exponential growth of malware samples, intelligent methods for efficient and effective malware detection at the cloud (server) side are urgently needed. As a result, much research has been conducted on developing intelligent malware detection systems using data mining and machine learning techniques. In these methods, the process of malware detection is generally divided into two steps: *feature extraction* and *classification / clustering*. We provide a comprehensive investigation on both the feature extraction and the classification/clustering steps. We conclude that data-mining-based malware detection framework can be designed to achieve good detection performance with high accuracy while maintaining low false positives. Many developed systems have been successfully integrated into commercial anti-malware products.

The following are some practical insights from our view: (1) Based on different data sets with different feature representation methods, there is no single classifier/clustering algorithm always performing best. In other words, the performance of such malware detection methods critically depend on the extracted features, data distributions, and the categorization methods. (2) Generally, compared with individual classifiers, an ensemble of classifiers can always help improve the detection accuracy. In real applications, a successful malware detection framework should utilize multiple diverse classifiers on various types of feature representations. (3) For feature extraction, both static and dynamic analysis approaches have their own advantages and limitations. In real applications, we suggest using static analysis at first, since over 80% of the file sample collection can be well-represented by static features. If the file cannot be well-represented by static extraction, then we can try dynamic analysis. Novel features, such as file-to-file relation graphs, can also provide invaluable information about the properties of file samples. (4) In order to achieve the best detection performance in real applications, it is often better to have enough training samples with balanced distributions for both classes (malware and benign files).

REFERENCES

Tony Abou-As saleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. 2004. N-gram-based detection of new malicious code. In *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC)*.

David W. Aha, Dennis Kibler, and Marc K. Albert. 1991. Instance-based learning algorithms. *Machine Learning* 6, 1 (1991), 37–66.

Blake Anderson, Daniel Quist, Joshua Neil, Curtis Storlie, and Terran Lane. 2011. Graph based malware detection using dynamic analysis. *Journal in Computer Virology* 4 (2011), 247–258.

Blake Anderson, Curtis Storlie, and Terran Lane. 2012. Improving malware classification: Bridging the static/dynamic gap. In *Proceedings of 5th ACM Workshop on Security and Artificial Intelligence (AISec)*.

Anubis. 2010. Anubis: Analyzing Unknown Binaries. Retrieved from http://anubis.iseclab.org/.

Michael Bailey, Jon Oberheide, Jon Andersen, Z. Morley Mao, Farnam Jahanian, and Jose Nazario. 2007. Automated classification and analysis of internet malware. In *Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection*.

Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. 2009. Scalable, behavior-based malware clustering. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium*.

Ulrich Bayer, Christopher Kruegel, and Engin Kirda. 2006a. TTAnalyze: A tool for analyzing malware. In *EICAR*.

Ulrich Bayer, Andreas Moser, Christopher Kruegel, and Engin Kirda. 2006b. Dynamic analysis of malicious code. *Journal in Computer Virology* 2(1) (2006), 67–77.

Zahra Bazrafshan, Hashem Hashemi, Seyed Mehdi Hazrati Fard, and Ali Hamzeh. 2013. A survey on heuristic malware detection techniques. In *Proceedings of the 5th Conference on Information and Knowledge Technology (IKT)*.

Philippe Beaucamps and ric Filiol. 2007. On the possibility of practically obfuscating programs towards a unified perspective of code protection. *Journal in Computer Virology* 3, 1 (2007), 3–21.

Yoshua Bengio. 2009. Learning deep architectures for AI. *Foundations and Trends in Machine Learning* 2, 1 (2009), 1–127.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. 2007. Greedy layer-wise training of deep networks. *Advances in Neural Information Processing Systems* 19 (2007).

Christopher M. Bishop. 1995. Neural networks for pattern recognition. *Oxford, Clarendon Press*.

Bizjournals. 2011. McAfee: Trends in a decade of cybercrime. Retrieved from http://www.bizjournals.com/sanjose/news/2011/01/25/mcafee-trends-in-a-decade-of-cybercrime.html?page=all.

Kevin Borders and Atul Prakash. 2004. Web tap: Detecting covert web traffic. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*.

Leo Breiman. 1996. Bagging predicators. *Machine Learning* 24, 2 (1996), 123–140.

Leo Breiman. 2001. Random forests. *Machine Learning* 45, 1 (2001), 5–32.

Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. 2007. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*.

Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-scale graph mining for malware detection. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*.

Lingwei Chen, William Hardy, Yanfang Ye, and Tao Li. 2015. Analyzing file-to-file relation network in malware detection. In *Proceedings of the International Conference on Web Information Systems Engineering (WISE)*.

Mihai Christodorescu and Somesh Jha. 2003. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th Conference on USENIX Security Symposium*.

Mihai Christodorescu, Somesh Jha, and Christopher Kruegel. 2007. Mining specifications of malicious behavior. In *Proceedings of ESEC/FSE*.

Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. 2005. Semantics-aware malware detection. In *Proceedings of IEEE Symposium on Security and Privacy*.

William W. Cohen. 1995. Fast effective rule induction. In *Proceedings of 12th International Conference on Machine Learning*.

Peter Coogan. 2010. SpyEye Bot Versus Zeus Bot. Retrieved from http://www.symantec.com/connect/blogs/spyeye-bot-versus-zeus-bot.

Thomas Cover and Peter Hart. 1967. Nearest nieghbor pattern classification. *IEEE Transaction on Information Theory* IT-13, 1 (1967), 21–27.

Jedidiah R. Crandall, Zhendong Su, S. Felix Wu, and Frederic T. Chong. 2005. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*.

Nilesh Dalvi, Pedro Domingos, Mausam, Sumit Sanghai, and Deepak Verma. 2004. Adversarial classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 99–108.

Damballa. 2008. 3% to 5% of Enterprise Assets Are Compromised by Bot-Driven Targeted Attack Malware. Retrieved from http://www.prnewswire.com/news-releases/3-to-5-of-enterprise-assets-are-compromised-by-bot-driven-targeted-attack-malware-61634867.html.

Mohsen Damshenas, Ali Dehghantanha, and Ramlan Mahmoud. 2013. A survey on malware propagation, analysis, and detection. *International Journal of Cyber-Security and Digital Forensics (IJCSDF)* 2, 4 (2013), 10–29.

Sanjeev Das, Yang Liu, Wei Zhang, and Mahintham Chandramohan. 2016. Semantics-based online malware detection: Towards efficient real-time protection against malware. *IEEE Transactions on Information Forensics and Security* 11, 2 (2016), 289–302.

Thomas Dietterich. 1997. Machine learning research: Four current directions. *Artificial Intelligence Magzine* 18, 4 (1997), 97–36.

Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *Proceedings of the 1st International Workshop on Multiple Classifier Systems*.

Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. 2008. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*.

Pedro Domingos and Michael Pazzani. 1997. On the optimality of simple Bayesian classifier under zero-one loss. *Machine Learning* 29, 2–3 (1997), 103–130.

Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Computing Surveys (CSUR)* 44, 2 (2012), 6.

Yuval Elovici, Asaf Shabtai, Robert Moskovitch, Gil Tahan, and Chanan Glezer. 2007. Applying machine learning techniques for detection of malicious code in network traffic. *KI: Advances in Artificial Intelligence* (2007).

EMarketer. 2014. Global B2C Ecommerce Sales to Hit $1.5 Trillion This Year Driven by Growth in Emerging Markets. Retrieved from http://www.emarketer.com/Article/Global-B2C-Ecommerce-Sales-Hit-15-Trillion-This-Year-Driven-by-Growth-Emerging-Markets/1010575.

Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Gomes Amorim. 2014. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research* 15, 1 (2014), 3133–3181.

Eric Filiol, Gregoire Jacob, and Mickael Le Liard. 2007. Evaluation methodology and theoretical model for antiviral behavioural detection strategies. *Journal in Computer Virology* 3, 1 (2007), 23–37.

Ivan Firdausi, Alva Erwin, and Anto Satriyo Nugroho. 2010. Analysis of machine learning techniques used in behavior based malware detection. In *Proceedings of 2nd International Conference on Advances in Computing, Control and Telecommunication Technologies (ACT)*.

Evelyn Fix and Joseph L. Hodges Jr. 1951. Discriminatory analysis-nonparametric discrimination: Consistency properties. *US Air Force, School of Avaiation Medicine, Tech. Rep* 4 (1951), 5–32.

Matt Fredrikson, Somesh Jha, Mihai Christodorescu, Reiner Sailer, and Xifeng Yan. 2010. Synthesizing near-optimal malware specifications from suspicious behaviors. In *Proceedings of IEEE Symposium on Security and Privacy*.

Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comp. Syst. Sci.* 55, 1 (1997), 119–39.

Ekta Gandotra, Divya Bansal, and Sanjeev Sofat. 2014. Malware analysis and classification: A survey. *Journal of Information Security* 5, 2 (2014), 56–64.

Maria Garnaeva, Victor Chebyshev, Denis Makrushin, Roman Unuchek, and Anton Ivanov. 2014. Kaspersky Security Bulletin 2014. Retrieved from http://securelist.com/analysis/kaspersky-security-bulletin/68010/kaspersky-security-bulletin-2014-overall-statistics-for-2014/.

Todd R. Golub, Donna K. Slonim, Pablo Tamayo, Christine Huard, Michelle Gaasenbeek, Jill P. Mesirov, and Hilary Coller. 1999. Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring. *Science* 286, 5439 (1999), 531–537.

Isabelle Guyon and Andr Elisseeff. 2003. An introduction to variable and feature selection. *Jouranl of Machine Learning Research* 3 (March 2003), 1157–1182.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *ACM SIGKDD Explorations Newsletter* (2009).

William Hardy, Lingwei Chen, Shifu Hou, Yanfang Ye, and Xin Li. 2016. DL4MD: A deep learning framework for intelligent malware detection. In *Proceedings of the International Conference on Data Mining (DMIN)*.

Olivier Henchiri and Nathalie Japkowicz. 2006a. A feature selection and evaluation scheme for computer virus detection. In *Proceedings of the 6th International Conference on Data Mining*.

Olivier Henchiri and Nathalie Japkowicz. 2006b. A feature selection and evaluation scheme for computer virus detection. In *Proceedings of ICDM*.

Shif Hou, Aaron Saas, Yanfang Ye, and Lifei Chen. 2016. DroidDelver: An android malware detection system using deep belief network based on API call blocks. In *Proceedings of the International Conference on Web-Age Information Management*. 54–66.

Xin Hu. 2011. *Large-scale malware analysis, detection, and signature generation.* Ph.D. Dissertation, Department of Computer Science and Engineering, University of Michigan.

Galen Hunt and Doug Brubacher. 1998. Detours: Binary interception of win32 functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*.

IDAPro. 2016. The Interactive Disassembler. Retrieved from https://www.hex-rays.com/products/ida/support/download_freeware.shtml.

Nwokedi Idika and Aditya P. Mathur. 2007. A survey of malware detection techniques. *Research Report in Purdue University* (2007).

Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbor: Towards removing the curse of dimensionality. In *Proceedings of 30th Annual ACM Symposium on Theory of Computing*.

Virtualization Technology Intel. 2013. Retrieved from http://www.intel.com/technology/virtualization.

Rafiqul Islam, Ronghua Tian, Lynn M. Batten, and Steve Versteeg. 2013. Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Application* 36, 2 (2013), 646–656.

ITU. 2014. ITU releases 2014 ICT figures. Retrieved from https://www.itu.int/net/pressoffice/press_releases/2014/23.aspx.

Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. 2000. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 1 (2000), 4–37.

Xuxian Jiang, Dongyan Xu, Helen Wang, and Eugene Spafford. 2005. Virtual playgrounds for worm behavior investigation. In *Proceedings of the 8th International Symposium on Recent Advances in Intrusion Detection*.

Thorsten Joachims. 1998. Making large-scale support vector machine learning practical. *Advances in Kernel Methods: Support Vector Machines* (1998).

George H. John and Pat Langley. 1995. Estimating continuous distributions in Bayesian classifiers. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Min Gyung Kang, Pongsin Poosankam, and Heng Yin. 2007. Renovo: A hidden code extractor for packed executables. In *Proceedings of the 5th ACM Workshop on Recurring Malcode (WORM)*.

Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Vern Paxson, Geoffrey M. Voelker, and Stefan Savage. 2008. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS)*.

Nikos Karampatziakis, Jack W. Stokes, Anil Thomas, and Mady Marinescu. 2013. Using file relationships in malware classification. In *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment*.

Md Enamul Karim, Andrew Walenstein, Arun Lakhotia, and Laxmi Parida. 2005. Malware phylogeny generation using permutations of code. *Journal in Computer Virology* 1, 1–2 (2005), 13–23.

Kaspersky. 2015. The Great Bank Robbery. Retrieved from http://www.kaspersky.com/about/news/virus/2015/Carbanak-cybergang-steals-1-bn-USDfrom-100-financial-institutions-worldwide.

Kris Kendall and Chad McMillan. 2007. Practical Malware Analysis. Retrieved from https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Presentation/bh-dc-07-Kendall_McMillan.pdf.

Kingsoft. 2014. 2013-2014 Internet Security Report in China. Retrieved from http://www.ijinshan.com/news/2014011401.shtml.

Kingsoft. 2015. 2014-2015 Internet Security Research Report in China. Retrieved from http://www.cssn.cn/xwcbx/xwcbx_gcsy/201501/P020150122566733317860.pdf.

Kingsoft. 2016. 2015-2016 Internet Security Research Report in China. Retrieved from http://cn.cmcm.com/news/media/2016-01-14/60.html.

Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFengWang. 2009. Effective and efficient malware detection at the end host. In *Proceedings of the 18th Conference on USENIX Security Symposium*.

Jeremy Z. Kolter and Marcus A. Maloof. 2004. Learning to detect malicious executables in the wild. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

J. Zico Kolter and Marcus A. Maloof. 2006. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research* 7 (Dec. 2006), 2721–2744.

Nojun Kwak and Chong-Ho Choi. 2002. Input feature selection by mutual information based on parzen window. *IEEE Trans. Pattern Anal. Mach. Intell.* 24, 12 (2002), 1667–1671.

Pat Langley. 1994. Selection of relevant features in machine learning. In *Proceedings of AAAI Fall Symposium*.

Andrea Lanzi, Monirul Sharif, and Wenke Lee. 2009. K-Tracer: A system for extracting kernel malware behavior. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS)*.

Tony Lee and Jigar J. Mody. 2006. Behavioral classification. In *Proceedings of the European Institute for Computer Antivirus Research Conference (EICAR)*.

David D. Lewis and William A. Gale. 1994. A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Springer-Verlag, New York, Inc., 3–12.

Shengqiao Li, E. James Harner, and Donald A. Adjeroh. 2011. Random KNN feature selection - A fast and stable alternative to random forests. *BMC Bioinformatics* 12, 1 (2011), 450.

Shengqiao Li, E. James Harner, and Donald A. Adjeroh. 2014. Random KNN. In *Proceedings of the 2014 IEEE International Conference on Data Mining Workshops*. 629–636.

Tao Li (Ed.). 2015. *Event Mining: Algorithms and Applications*. CRC Press.

LordPE. 2013. PE Tools - LordPE. Retrieved from http://www.malware-analyzer.com/pe-tools.

Mike Loukides and Andy Oram. 1996. Getting to know gdb. *Linux Journal* (1996).

James Lyne. 2014. Security threat trends 2015. Retrieved from https://www.sophos.com/threat-center/media library/PDFs/other/sophos-trends-and-predictions-2015.pdf.

Mohammad M. Masud, Tahseen Al-Khateeb, Kevin W. Hamlen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani M. Thuraisingham. 2011. Cloud-based malware detection for evolving data streams. *ACM Trans. Management Inf. Syst.* 2, 3 (2011), 16.

Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2008. Mining concept-drifting data stream to detect peer to peer botnet traffic. *Tech. rep. UTDCS-05-08, The University of Texas at Dallas, Richardson* (2008).

Mohammad M. Masud, Latifur Khan, and Bhavani Thuraisingham. 2007. A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers* 10, 1 (2007), 33–45.

Kirti Mathur and Saroj Hiranwal. 2013. A survey on techniques in detection and analyzing malware executables. *International Journal of Advanced Research in Computer Science and Software Engineering* 3, 4 (2013), 422–428.

Micropoint. 2008. Micropoint Antivirus. Retrieved from http://www.micropoint.com.cn/Channel//200806261 14608.html.

David Moore and Colleen Shannon. 2002. Code-red: A case study on the spread and victims of an internet worm. In *Proceedings of the Internet Measurement Workshop*.

Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*.

Robert Moskovitch, Clint Feher, and Yuval Elovici. 2009. A chronological evaluation of unknown malcode detection. *LNCS: Intelligence and Security Informatics* 5477 (2009), 112–117.

Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. 2008a. Unknown malcode detection using OPCODE representation. In *Proceedings of the European Conference on Intelligence and Security Informatics (EuroISI)*.

Robert Moskovitch, Nir Nissim, and Yuval Elovici. 2008b. Acquisition of malicious code using active learning. In *PinKDD*.

Robert Moskovitch, Dima Stopel, Clint Feher, Nir Nissim, and Yuval Elovici. 2008c. Unknown malcode detection via text categorization and the imbalance problem. In *IEEE Intelligence and Security Informatics*.

Kevin P. Murphy. 2012. Machine learning: A probabilistic perspective. In *The MIT Press, Cambridge, Massachuseтts*.

Ion Muslea, Steven Minton, and Craig A. Knoblock. 2006. Active learning with multiple views. *Journal of Artificial Intelligence Research* 27 (2006), 203–233.

Carey Nachenberg and Vijay Seshadri. 2010. An analysis of real-world effectiveness of reputation-based security. In *Proceedings of the Virus Bulletin Conference (VB)*.

Nicholas Nethercote and Julian Seward. 2007. Valgrind: A framework for heavyweight dynamic binary instrumentation. In *Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation*.

Hieu T. Nguyen and Arnold Smeulders. 2004. Active learning using pre-clustering. In *Proceedings of the 21st International Conference on Machine Learning*. ACM, 79.

Ming Ni, Tao Li, Qianmu Li, Hong Zhang, and Yanfang Ye. 2016. FindMal: A file-to-file social network based malware detection framework. *Knowledge-Based Systems* 112 (2016), 142–151.

Corporation of Compuware. 1999. Debugging blue screens. *Technical Paper* (September 1999).

Gunter Ollmann. 2010. Serial variant evasion tactics techniques used to automatically bypass antivirus technologies. Retrieved from http://www.damballa.com/downloads/rpubs/WPSerialVariantEvasionTactics. pdf.

OllyDump. 2006. PE Tools - OllyDump. Retrieved from http://www.openrce.org/downloads/details/108/ OllyDump.

David Orenstein. 2000. Application programming interface (API). In *Quick Study: Application Programming Interface (API)*.

Nikunj C. Oza and Stuart Russell. 2001. Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of SIGKDD*.

Judea Pearl. 1987. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence* 32, 2 (1987), 245–258.

Hanchuan Peng, Fuhui Long, and Chris Ding. 2005. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 8 (2005), 1226–1238.

Simon Perkins, Kevin Lacker, and James Theiler. 2003. Grafting: Fast incremental feature selection by gradient descent in function space. *JMLR* 3 (March 2003), 1333–1356.

Qemu. 2016. (2016). http://www.qemu-project.org/index.html.

Internet Security Center Qihoo. 2015. 2014 Internet Security Research Report in China. Retrieved from http://zt.360.cn/report/.

J. Ross Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (1986), 81–106.

J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* San Francisco, CA: Morgan Kaufmann Publishers, Inc. (1993).

Alain Rakotomamonjy. 2003. Variable selection using SVM-based criteria. *JMLR* 3 (March 2003), 1357–1370.

Zulfikar Ramzan, Vijay Seshadri, and Carey Nachenberg. 2013. Reputation-based security: An analysis of real world effectiveness. In *Symantec Security Response*.

Rizwan Rehmani, G. C. Hazarika, and Gunadeep Chetia. 2011. Malware threats and mitigation strategies: A Survey. *Journal of Theoretical and Applied Information Technology* 29, 2 (2011), 69–73.

John Robbins. 1999. Debugging windows based applications using windbg. *Microsoft Systems Journal* (1999).

Lior Rokach. 2010. Ensemble-based classifiers. *Artif Intell Rev* 33, 1 (2010), 1–39.

Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. 2006. PolyUnpack: Automating the hidden-code extraction of unpack-executing malware. In *Proceedings of the 22nd Annual Computer Security Applications Conference*.

Yvan Saeys, Inaki Inza, and Pedro Larranaga. 2007. A review of feature selection techniques in bioinformatics. *Bioinformatics* 23, 19 (2007), 2507–2517.

Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.

Igor Santos, Jaime Devesa, Felix Brezo, Javier Nieves, and Pablo Garcia Bringas. 2013. OPEM: A static-dynamic approach for machine learning based malware detection. In *Proceedings of International Conference CISIS-ICEUTE, Special Sessions Advances in Intelligent Systems and Computing*.

Igor Santos, Carlos Laorden, and Pablo G. Bringas. 2011a. Collective classification for unknown malware detection. In *Proceedings of the International Conference on Security and Cryptography*.

Igor Santos, Javier Nieves, and Pablo G. Bringas. 2011b. Semi-supervised learning for unknown malware detection. In *International Symposium on Distributed Computing and Artificial Intelligence Advances in Intelligent and Soft Computing*.

Joshua Saxe and Konstantin Berlin. 2015. Deep neural network based malware detection using two dimensional binary program features. In *Proceedings of the 10th International Conference on Malicious and Unwanted Software (MALWARE)*.

Matthew G. Schultz, Eleazar Eskin, F. Zadok, and Salvatore J. Stolfo. 2001. Data mining methods for detection of new malicious executables. In *Proc. of the IEEE Symposium on Security and Privacy*.

Fabrizio Sebastiani. 2002. Text categorization. *Comput. Surveys* 34, 1 (2002), 1–47.

H. Sebastian Seung, Manfred Opper, and Haim Sompolinsky. 1992. Query by committee. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*. ACM, 287–294.

Asaf Shabtai, Robert Moskovitch, Yuval Elovici, and Chanan Glezer. 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. *Information Security Technical Report* 14, 1 (2009), 16–29.

Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. 2008. A survey of data mining techniques for malware detection using file features. In *Proceedings of ACM-SE*.

Muazzam Siddiqui, Morgan C. Wang, and Joohan Lee. 2009. Detecting internet worms using data mining techniques. *Journal of Systemics, Cybernetics and Informatics* 6, 6 (2009), 48–53.

Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. 2008. BitBlaze: A new approach to computer

security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security*.

Eugene H. Spafford. 1989. The internet worm incident. In *Proceedings of the 2nd European Software Engineering Conference*.

Elizabeth Stinson and John C. Mitchell. 2007. Characterizing bots' remote control behavior. *LNCS: Detection of Intrusions and Malware, and Vulnerability Assessment* 4579 (2007), 89–108.

Jack W. Stokes, John C. Platt, Helen J. Wang, Joe Faulhaber, Jonathan Keller, Mady Marinescu, Anil Thomas, and Marius Gheorghescu. 2012. Scalable telemetry classification for automated malware detection. *Computer Security - ESORICS* (2012).

Andrew H. Sung, Jianyun Xu, Patrick Chavez, and Srinivas Mukkamala. 2004. Static analyzer of vicious executables (SAVE). In *Proceedings of the 20th Annual Computer Security Applications Conference*.

Symantec. 2008. Symantec global internet security threat report. Retrieved from http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiii_04-2008.en-us.pdf.

Symantec. 2014a. Internet Security Threat Report 2014. Retrieved from http://www.symantec.com/security_response/publications/threatreport.jsp.

Symantec. 2014b. The Threat Landscape in 2014 and Beyond: Symantec and Norton Predictions for 2015, Asia Pacific and Japan. Retrieved from http://www.symantec.com/connect/blogs/threat-landscape-2014-and-beyond-symantec-and-norton-predictions-2015-asia-pacific-japan.

Symantec. 2016. Internet Security Threat Report. Retrieved from https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.

Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 76.

Acar Tamersoy, Kevin Roundy, and Duen Horng Chau. 2014. Guilt by association: Large scale malware detection by mining file-relation graphs. In *Proccedings of ACM International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD)*.

Fadi Abdeljaber Thabtah. 2007. A review of associative classification mining. *Knowledge Engineering Review* 22, 1 (2007), 37–65.

Ronghua Tian, Rafiqul Islam, Lynn Batten, and Steve Versteeg. 2010. Differentiating malware from cleanwares using behavioral analysis. In *Proceedings of 5th International Conference on Malicious and Unwanted Software (Malware)*.

TrendLabs. 2014. The invisible becomes visible: Trend micro security predictions for 2015 and beyond. (2014). http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-the-invisible-becomes-visible.pdf.

Trend Threat Research Team TrendMicro. 2010. Zeus: A Persistent Criminal Enterprise. Retrieved from http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_zeuspersistent-criminal-enterprise.pdf.

Amit Vasudevan and Ramesh Yerraballi. 2005. Stealth breakpoints. In *Proceedings of the 21st Annual Computer Security Applications Conference*.

Amit Vasudevan and Ramesh Yerraballi. 2006. Cobra: Fine-grained malware analysis using stealth localized-executions. In *Proceedings of 2006 IEEE Symposium on Security and Privacy*.

Shobha Venkataraman, Avrim Blum, and Dawn Song. 2008. Limits of learning-based signature generation with adversaries. In *NDSS*.

Andrei Venzhega, Polina Zhinalieva, and Nikolay Suboch. 2013. Graph-based malware distributors detection. In *Proceedings of the 22nd International Conference on World Wide Web Companion (WWW)*.

Randall Wald, Taghi M. Khoshgoftaar, and Amri Napolitano. 2013. Comparison of stability for different families of filter-based and wrapper-based feature selection. In *ICMLA*.

Tzu-Yen Wang, Shi-Jinn Horng, Ming-Yang Su, Chin-Hsiung Wu, Peng-Chu Wang, and Wei-Zen Su. 2006b. A surveillance spyware detection system based on data mining methods. *Evolutionary Computation* (2006), 3236–3241.

Yi-Min Wang, Doug Beck, Xuxian Jiang, and Roussi Roussev. 2006a. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *NDSS*.

SECURITY LABS WEBSENSE. 2014. 2015 Security Predictions. Retrieved from http://www.websense.com/assets/reports/report-2015-security-predictions-en.pdf.

Paul Werbos. 1974. Beyond regression: New tools for prediction and analysis in the behavioral science. Ph.D. Dissertation, Harvard University.

Wikipedia. 2016. Scareware. Retrieved from https://en.wikipedia.org/wiki/Scareware.

Wikipedia. 2017a. Assembly Language. Retrieved from http://en.wikipedia.org/wiki/Assembly_language.

Wikipedia. 2017b. Computer Virus. Retrieved from http://en.wikipedia.org/wiki/Computer_virus.

Wikipedia. 2017c. Morris Worm. Retrieved from http://en.wikipedia.org/wiki/Morris_worm.

Wikipedia. 2017d. Ransomware. Retrieved from https://en.wikipedia.org/wiki/Ransomware.

Wikipedia. 2017e. Rootkit. Retrieved from http://en.wikipedia.org/wiki/Rootkit.

Wikipedia. 2017f. Zero-day (computing). Retrieved from https://en.wikipedia.org/wiki/Zero-day_(computing).

Wikipedia. 2017g. Zeus (malware). Retrieved from http://en.wikipedia.org/wiki/Zeus_(malware).

Carsten Willems, Thorsten Holz, and Felix Freiling. 2007. Toward automated dynamic malware analysis using cwsandbox. In *IEEE Security and Privacy*.

Rui Xu and Donald Wunsch. 2005. Survey of clustering algorithms. In *IEEE Transactions on Neural Networks* 16, 3 (2005), 645–678.

Yanfang Ye. 2010. Research on intelligent malware detection methods and their applications. *Ph.D. Dissertation, Department of Computer Science, Xiamen University* (2010).

Yanfang Ye, Lifei Chen, Dingding Wang, Tao Li, Qingshan Jiang, and Min Zhao. 2009. SBMDS: An interpretable string based malware detection system using SVM ensemble with bagging. *Journal in Computer Virology* 5, 4 (2009), 283–293.

Yanfang Ye, Tao Li, Yong Chen, and Qingshan Jiang. 2010. Automatic malware categorization using cluster ensemble. In *Proccedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Yanfang Ye, Tao Li, Kai Huang, Qingshan Jiang, and Yong Chen. 2009a. Hierarchical associative classifier (HAC) for malware detection from the large and imbalanced gray list. *Journal of Intelligent Information Systems* 35, 1 (2009), 1–20.

Yanfang Ye, Tao Li, Qingshan Jiang, Zhixue Han, and Li Wan. 2009c. Intelligent file scoring system for malware detection from the gray list. In *Proccedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Yanfang Ye, Tao Li, Qingshan Jiang, and Youyu Wang. 2009b. CIMDS: Adapting post-processing techniques of associative classification for malware detection system. *IEEE Transactions on Systems, Man, and Cybernetics* 40, 3 (2009), 298–307.

Yanfang Ye, Tao Li, Shenghuo Zhu, Weiwei Zhuang, Egemen Tas, Umesh Gupta, and Melih Abdulhayoglu. 2011. Combining file content and file relations for cloud based malware detection. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Yanfang Ye, Dingding Wang, Tao Li, and Dongyi Ye. 2007. IMDS: Intelligent malware detection system. In *Proccedings of ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD)*.

Yanfang Ye, Dingding Wang, Tao Li, Dongyi Ye, and Qingshan Jiang. 2008. An intelligent PE-malware detection system based on association mining. *Journal in Computer Virology* 4, 4 (2008), 323–334.

Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. 2001. Understanding belief propagation and its generalizations. In *Mitsubishi Electric Research Laboratories*.

Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. 2007. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS)*.

Chunqiu Zeng, Liang Tang, Wubai Zhou, Tao Li, Larisa Shwartz, and Genady Ya.Grabarnik. 2017. An integrated framework for mining temporal logs from fluctuating events. *IEEE Transactions on Services Computing (TSC)* (2017). In Press.

Boyun Zhang, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. 2007. Malicious codes detection based on ensemble learning. *Autonomic and Trusted Computing* (2007).

Jianwei Zhuge, Thorsten Holz, Chengyu Song, Jinpeng Guo, Xinhui Han, and Wei Zou. 2008. Studying malicious websites and the underground economy on the Chinese web. In *Proceedings of the 7th Workshop on Economics of Information Security*.

Received August 2015; revised November 2016; accepted March 2017