Journal of Information Security and Applications 59 (2021) 102828

Contents lists available at ScienceDirect

Journal of Information Security and Applications

journal homepage: www.elsevier.com/locate/jisa

Malware classification and composition analysis: A survey of recent developments

Adel Abusitta [*], Miles Q. Li, Benjamin C.M. Fung

McGill University, Montréal, Canada

ARTICLE INFO

Keywords:
Malware analysis
Malware classification
Security
Anti-analysis techniques
Composition analysis

ABSTRACT

Malware detection and classification are becoming more and more challenging, given the complexity of malware design and the recent advancement of communication and computing infrastructure. The existing malware classification approaches enable reverse engineers to better understand their patterns and categorizations, and to cope with their evolution. Moreover, new compositions analysis methods have been proposed to analyze malware samples with the goal of gaining deeper insight on their functionalities and behaviors. This, in turn, helps reverse engineers discern the intent of a malware sample and understand the attackers' objectives. This survey classifies and compares the main findings in malware classification and composition analyses. We also discuss malware evasion techniques and feature extraction methods. Besides, we characterize each reviewed paper on the basis of both algorithms and features used, and highlight its strengths and limitations. We furthermore present issues, challenges, and future research directions related to malware analysis.

1. Introduction

In the recent years, many cyber-security mechanisms have been designed and developed to defend against evolving security threats. Nevertheless, recent statistics [1] indicate that malware are still evolving and becoming more sophisticated than ever. As a result, they become harder to detect and understand their innerworkings. This mainly stems from two essential reasons. The first is that attackers have now become more proficient in launching attacks and hiding their malicious behavior using anti-analysis techniques such as obfuscation and packing. The second reason is that the current communication and computing infrastructure is becoming more and more dynamic and heterogeneous, which enables a single malware to take various forms that are semantically but not structurally similar. This, in turn, makes malware analysis even more challenging.

Malware (or Malicious software) is a software that is designed to harm users, organizations, and telecommunication and computer system. More specifically, malware can block internet connection, corrupt an operating system, steal a user's password and other private information, and/or encrypt important documents on a computer and demand ransom. For the latest years, malware has been a growing threat to computer users and in 2017 the number of new malware increased by 22,9% over 2016 to reach 8,400,058 [2–5]. Moreover, malware has become the primary medium to launch large-scale attacks, such as compromising computers, bringing down hosts and servers, sending out spam emails, crippling critical infrastructures and penetrating data centers [6–8]. These attacks lead to severe damage and significant financial loss [9–11].

Most antivirus engines detect and classify malware by continuously scanning files and comparing their signatures with known malware signatures. The malware signatures are typically created by human antivirus experts (known as malware defenders) who examine the collected malware samples. These malware signatures can be filename, text strings, or regular expressions of byte code [12,13]. Obviously, signature-based methods can only detect traditional malware that do not change significantly. However, malware can hide its malicious behavior using anti-analysis techniques such as obfuscation, packing, polymorphism and metamorphism, in such a way that the code would look quite different from its original version. Thus, the primary shortcoming of the signature-based method is that they entail high precision but low recall. Also, the process of creating malware signatures is labor-intensive. Considering that there is a large number of new malware that appear every day, there is a pressing need to develop new intelligent malware analysis methods to tackle the challenges.

To alleviate the burden of manual signature crafting, researchers propose automatic signature generation methods [14,15]. The content of the signatures can be Windows system call combinations [16], control flow graph [15], and functions [14].

Researchers also propose to use machine learning models to detect and classify malware [12,17–27]. Different from other machine

[*] Corresponding author.
E-mail addresses: adel.abusitta@mcgill.ca (A. Abusitta), miles.qi.li@mail.mcgill.ca (M.Q. Li), ben.fung@mcgill.ca (B.C.M. Fung).

https://doi.org/10.1016/j.jisa.2021.102828

Available online 26 April 2021

learning-driven classification tasks, such as image classification, there is a competition between malware creators and defenders. When malware defenders propose a new malware analysis system using some features and machine learning models, malware creators often update their malware design to avoid being detected. Then malware defenders would propose new systems to detect and analyze the new generation of malware and so forth. The race between malware defenders and attackers may never come to an end.

Recently, many researchers have started to use deep learning models to enhance the detection and classification accuracy of malware classification [24–27]. Although promising results have been achieved through the ability to extract robust and useful features using the state-of-the-art deep learning architectures, the proposed models were shown to be highly vulnerable to adversarial examples, which can be easily designed (simply by perpetuating parts of the inputs) by attackers to fool Artificial Intelligence (AI)-driven malware analysis systems and make them generate erroneous decisions [24–29]. As a result, several methods have been proposed to defend against adversarial examples [28,29].

In addition to malware classification, researchers in malware analysis have improved new techniques and methods to analyze the composition of malware samples by matching their functionalities and behaviors to multiple known malware families. This, in turn, helps reverse engineers discern the intent of a malware sample and the attacker. Moreover, these composition methods enable the reverse engineers and organizations to effectively triage their resources.

1.1. The scope

This literature review classifies and compares the recent and main findings in malware classification. Unlike other similar works which only focus either on AI-driven malware classification [30–32] or on non-AI-driven malware classification [33,34], this paper includes both AI-driven and non-AI-driven recent works. We are also surveying methods and approaches that recently have been proposed to analyze the composition of malware samples, in order to understand their functionalities and behaviors. To the best of our knowledge, this is the first work that survey the existing composition analysis techniques. This survey also aims at identifying the main issues and challenges related to recent malware classification and composition analysis techniques. In particular, our analysis leads to recognize three major problems to address. The first is the need to overcome modern evading techniques (or anti-analysis techniques) such as metamorphism. The second relates to the efficiency and scalability of malware search engines as the number of functions in the repository might need to scale up to millions. The third concerns the vulnerability of malware classification system to evolving adversarial examples. We also uncover possible topics that need further study and investigation, such as sustainable malware analysis system. In this regard, we propose a few guidelines to prepare efficient and trustworthy malware detection and analysis system.

1.2. Contribution

The main contributions of this survey are:

- Proposing a new taxonomy for describing and comparing the recent and main findings in malware classification and composition analysis.
- Designing a new framework for analyzing the existing malware classification and composition analysis techniques.
- Identifying and presenting open issues and challenges related to malware analysis.
- Identifying a number of trends on the topic, with guidelines on how to improve existing solutions to address new and continuing challenges.

1.3. Organization

The rest of this paper is organized as follows. In Section 2, we discuss the related survey papers. In Section 3 and Section 4, we present the proposed taxonomy for organizing reviewed malware classification and composition analysis approaches, respectively. Section 5 characterizes reviewed papers according to the proposed taxonomy. The challenges and current issues are pointed out in Section 6. Section 7 suggests possible research topics in malware analysis. Finally, Section 8 concludes the paper.

2. Related surveys

Other works have already surveyed contributions in malware classification. For example, Bazrafshan et al. [33] classify malware detection and classify methods into three types: signature-based, behavior-based and heuristic-based methods. Also, they recognize five classes of features based on the proposed heuristic-based method: opcodes, API calls, control flow graphs, n-grams, and hybrid features. Another work presented by Shabtai et al. [34], which studies how to detect malware using static features. In this paper, we study more features (static and dynamic features) used for malware classification.

Ucci et al. [30] survey the literature on machine learning approaches for malware detection and analysis. They classify the surveyed articles into three categories: objectives (expected output), features, and algorithm used. They also highlight a set of problems and challenges and identify the new research directions. Similarly, the survey presented by [31] presents a comparative analysis on intelligence-based malware classification. In particular, they report cons, pros and problems associated with each machine learning-based malware classification technique. Souri and Hosseini [32] also provide a taxonomy of AI-driven malware detection techniques. Our paper looks at a larger range of articles by including many works on malware classification and composition analysis. We also include other works related to non-AI-driven classification techniques. Furthermore, We also include new challenges related to AI-driven malware classification techniques.

Also, Basu et al. [35] study different works relying on AI-powered malware classification techniques. In particular, they coin five types of features: a PI call graph, byte sequence, PE header and sections, assembly code frequency and system calls. Also, Ye et al. [36] study many different aspects of malware classification processes. More specifically, they spot the light on a number of issues such as incremental learning, and adversarial learning. Recently, Ori et al. [37] survey the literature on techniques used for dynamic malware analysis, which includes a description of each technique. In particular, they present an overview of machine-learning methods used to improve the capability of dynamic malware analysis. Compared to the above-motioned works, this paper determines the main issues and challenges on malware classification and composition analysis. Also, we identify a number of trends on the topic, with guidelines on how to improve solutions to address new and continuing challenges.

In addition, Barriga and Yoo [38] survey the literature on malware evasion techniques and their impact on malware analysis techniques. This paper extends beyond that and includes recent AI-driven works used to overcome malware evasion techniques.

3. Taxonomy of malware classification

We present in this section the taxonomy of malware classification. We define two categories (or dimensions) to organize the existing works. The first category presents the features that our work is based on. In particular, we discuss the different methodologies used for extracting features, e.g., dynamic and/or static techniques, and what types of features are used, e.g., assembly code. The second is concerned with the type of algorithm that is adopted for the detection and analysis, e.g., artificial inelegance-driven algorithm.

Fig. 1 shows the proposed taxonomy. The rest of this section is organized as follows (according to the proposed taxonomy). Section 3.1 describes malware analysis features, while Section 3.2 discusses existing algorithms.

3.1. Malware analysis features

This subsection presents the features of samples that are used for the analysis. In Section 3.1.1, we show how features are extracted, while in Section 3.1.2, we show type of features that are taken into account.

3.1.1. Feature extraction methods

In this section, we review the following three feature extraction methods: static, dynamic and hybrid methods.

Static method. Static feature extraction is a method to extract features from the content of the executables without running them [39]. The static features can be extracted using the file format, e.g., Portable Executable (PE) and Common Object File Format (COFF) [12,18,22,25]. The static features can also be extracted without any knowledge of the format. Features extracted this way can be byte sequences, file size, byte entropy, etc. [12,17,20,25]. The advantage of the static feature extraction method is that it covers the complete binary content. But the problem is that static features are prone to packing and polymorphism since most of the features that are statically extracted come from encrypted contents rather than the original program body [40].

Dynamic method. Dynamic feature extraction consists of running the executable usually in an insulated environment which can be a virtual machine (VM) or an emulator and then extract features from the memory image of the executable or from its behaviors [39]. Since malware equipped with packing and polymorphism has to exhibit the real malicious code to achieve their goals, dynamic feature extraction is more resistant to those malware techniques compared with static feature extraction method [40].

Anderson et al. [21,41] use Xen[1] and Royal et al. [42], Dai et al. [19], and Islam et al. [22] use VMWare[2] to create their VMs and perform dynamic analysis. Kolosnjaji et al. [27] use Cuckoo sandbox[3] which is an open source automated malware analysis system to extract API calls. Other researchers who work for an anti-virus engine use the VMs as parts of their anti-virus engines to dynamically extract features [24,26].

In fact, there are two categories of an emulator: a full-system emulator and application level emulator. A full-system emulator is a computer program that emulates every component of a computer, including its memory, processor, graphics card, hard disk, etc., with the purpose of running an unmodified operating system. Qemu[4] is a full-system emulator used by several systems [23,40,43]. Considering the time-consuming of full-system emulator, Cesare and Xiang [15] propose to use application level emulation to unpack malware more efficiently so that only the parts which are necessary to execute the file including instruction set, API, virtual memory, thread and process management, and OS specific structures are implemented.

One problem of dynamic feature extraction methods is that it does not reveal all the possible execution paths [40]. Malware may have detection routines to check whether it is executed in a virtual machine or emulator. When malware finds itself executing in such an environment, it will halt its execution so dynamic models will fail to recognize it as malware. The methods to detect whether an executable is executed inside a VM can be found from several papers [44,45]. Another problem of dynamic methods lies in its execution time which takes much more than static feature extraction [40].

Hybrid method. This method is used to achieve higher detection rate by merging some of the static feature extraction characteristics with some of the dynamic feature extraction characteristics [39].

Our survey has revealed that most of the surveyed papers were based on the dynamic feature extraction approach [21,24,46–63]. while the others adopt, in equal proportions, either the static approach alone [64–83] or a hybrid approach [22,23,41,47,84–86].

3.1.2. Type of features

In this section, we classify the features that are used by malware analysts and explain how each type is practically extracted and represented.

Printable strings. A printable string is a sequence of ASCII characters terminated with a null character. Schultz et al. [12] find that malware have some similar strings that distinguish it from and that Goodware also has some common strings that distinguish them from malware. Printable strings are represented as binary features, where “1” represents a string that is present in an executable and “0” represents that it is absent from all systems [12,22,24,26].

Schultz et al. [12] extract printable strings from the headers of PE files. The extraction is straight-forward since the header is in plain text format.

Dahl et al. [24] and Huang and Stokes [26] extract null-terminated objects dumped from images of a file in memory [24,26] as printable strings. The coverage of their methods is better than just extract printable strings from header [12] but their could be some false positive results.

Islam et al. [22] use the strings utility in IDA Pro[5] to extract printable strings from the whole file.

Different from other works, Saxe and Berlin [25] do not take printable strings as binary features but use their hash values and the logarithm of the string lengths to create a histogram and use the counts of printable strings in each bin of the histogram as features. They take all the byte sequences of length six or more that are in the ASCII code range as printable strings which is also slightly different from other works.

Essentially, the functionality of most malware does not rely on printable strings. Thus, when malware creators find that some strings accidentally are used by malware detectors, they can eliminate them or even if the printable strings are necessary, they can break them into characters that are distributed in different positions. Therefore, printable strings are not reliable features.

Byte sequences (byte code). Executable files consist of byte sequences (also known as byte code). A byte sequence may belong to the metadata, code, or data of an executable file. As has been stated, byte sequences are important signatures of malware since malware may share some common sequences that are exactly the same or follow the same regular expression. Thus, byte sequences are also appropriate to be features for malware analysis systems [12,17,25,41].

Schultz et al. [12] use bigram byte sequences in the form of binary features and they claim byte sequence feature is the most informative feature because it represents the machine code in an executable. In fact, this is not entirely true since some byte sequences come from metadata or data section. Even if a byte sequence is from code section, since instructions have variable length in some architectures, byte sequences may not match machine code. And their byte sequence feature has the problem of dimension explosion since there are too many different bigram byte sequences and it is too large to fit into memory so they could only split the byte sequence set into several sets and feed them to multiple native bayes models.

[1] https://www.xenproject.org/.

[2] https://www.vmware.com/.

[3] https://cuckoosandbox.org/.

[4] https://www.qemu.org/.

[5] https://www.hex-rays.com/products/ida/.

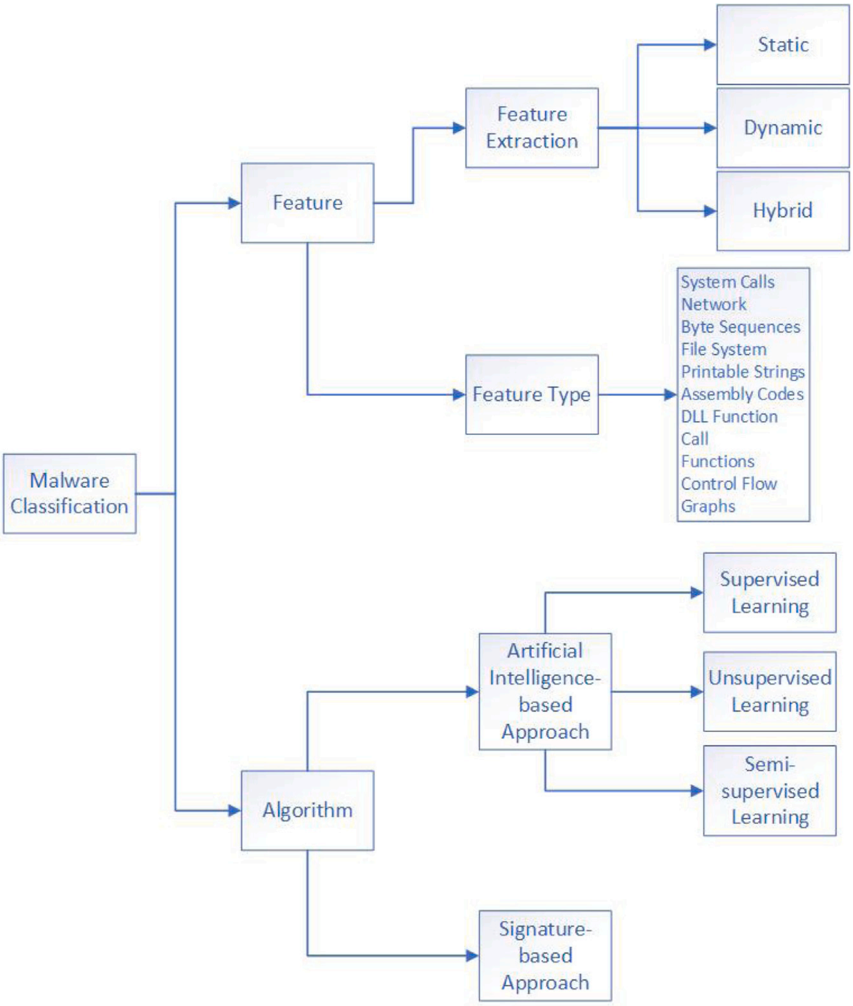


Fig. 1. The proposed taxonomy.

To solve the dimension explosion problem, Kolter and Maloof [17] use information gain to select the top 500 informative 4-gram byte sequences as binary features from 255 million distinct 4-grams.

Different from the above two works, Anderson et al. [41] do not use byte sequences per se as features but fit byte sequences into a Markov Model so essentially the feature they use is transition probability from one byte to another.

Chen et al. [25] use the byte entropy of each 1024 byte window and the occurrence of each byte to form a histogram and evenly separate each axis into 16 bins to form a 256 length feature vector.

Nataraj et al. [20] convert the whole byte sequence of a file into a picture in which each byte represents the gray scale of a pixel. They find that the malware that belongs to the same family appear very similar in layout and image. The width of the image that is used to transform the 1D byte sequence into a 2D matrix is determined by the size of the file. The image feature of the malware image is computed using the algorithm proposed by Oliva and Torralbat [87]. The main advantage of image-based techniques is that they are robust against many types of obfuscations [88].

Byte sequences are not reliable in most cases. This is due to the fact that obfuscation techniques such as instruction substitution and register reassignment can change the opcodes and oprands respectively, which means that the machine code is changed. In all these works, the byte code is statically extracted but the main program body encrypted with different algorithms or keys through Packing and Polymorphism will change the byte sequences.

Assembly code. Machine code and assembly code can be translated to one another through assembly and disassembly. Assembly code has some advantages over machine code as a feature for malware analysis. First, assembly code can be understood by a programmer and therefore as a kind of feature, assembly code is more convenient to be preprocessed (e.g., grouped into categories according to the function, filtered, truncated etc.) to appear as a more informative feature. In addition, malicious code is often encrypted by packing or polymorphism so it is impossible to get it from the original byte sequence, however, dynamically extracted assembly code has been decrypted so it includes the malicious code.

Moskovitch et al. [18] propose that assembly code can be more robust than machine code for the analysis of malware since the same malicious engine may locate in different locations of a file, and thus may be linked to different addresses in RAM or even perturbed slightly so by dropping the oprands and just using opcode the robustness is improved. They extract assembly code by dissembling the executables with IDA Pro. They try both term frequency (TF) and term frequency–inverse document frequency (TF-IDF) of each opcode n-gram (n=1,2, …,6) as features and use document frequency (DF), information gain ratio, or Fisher score to select features. Their best result is achieved using TF values of opcode bigram as features filtered by Fisher score. One disadvantage of their method is that it is still prone to dead code insertion, operation transpositions, packing, and polymorphism. Another one is dropping operands causes loss of information which may subsequently lead to loss of precision.

To counter packing and polymorphism, Dai et al. [19] run malware in a VM and record the sequence of the running byte code which will be disassembled to assembly code. They use three kinds of two-opcode combinations: unordered opcodes in a block, ordered but not necessarily consecutive opcodes in a block, consecutive opcodes in a block. This way their features is more resistant to dead code insertion and reorder of operations. They use the association between the frequency of a feature in training dataset and a class as criterion and apply a variant of Apriori [89] to select top L features. Although unordered opcodes and ordered (but not necessarily consecutive opcodes) in a block improve the resistance to dead code insertion and reorder of operations, those features are too flexible so they also bring more false positive situations.

Royal et al. [42] is another work aiming to detect code that is hidden and can only be seen dynamically. The way they do it is to store the static code of an executable and check whether each operation executed is within the stored static code area. If it is not, it is a part of hidden-code. They claim that the main malware engine should be in the hidden-code if both of them exist and experiment results also illustrate the hidden-code enhances the accuracy of ClamAV[6] and McAfee Antivirus.[7]

Anderson et al. [21,41] use the transition probability from one opcode to another as features, which is similar to how they use byte sequence feature. In their paper [21], they just extract assembly code by recording the execution of an executable in a VM which is similar to the way Royal et al. [42] use. In their second paper [21], they also use IDA Pro to disassemble the executable, and the assembly code from the two sources are used as two independent feature sets. In addition, they also group instructions into categories in several granularities according to the functions of the instructions to reduce the impact of instruction substitution in their second paper [21]. In their preliminary experiment, they also find if they use instructions with oprands, the performance will be worse [21].

Santos et al. [23] disassemble executables to acquire their assemble code and then use weighted opcode n-gram frequencies as one of their features. The weight is the product of the information gain of all opcodes in the n-gram times the normalized TF of the n-gram.

API/DLL system call. DLL files and functions of DLL files used by an executable expose the system services they use. Native system calls and Windows API calls an executable invokes are shown by the functions of DLL files it depends on. Therefore, what behaviors it may intend to do or what it would be able to do can be inferred.

Schultz et al. [12] extract the DLL files by an executable used, the functions in DLL files, and the number of function of each DLL as features from metadata in order to understand how resources affected an executable's behavior and how heavily each DLL is used. The first two are used as binary features and the third is a real-valued feature.

Bayer et al. [40] and Santos et al. [23] extract calls to Windows API functions dynamically using an emulator. Then, they use those API functions to acquire actions of an executable during execution including I/O activity, registry modification activity, process creation/termination activity, network connection activity of an executable, self-protection behavior, system information stealing, errors caused by the execution, and interactions with Windows Service Manager.

Fredrikson et al. [43] also use an emulator to monitor system calls. Then, they use the relations between system calls and their parameters to form a dependency graph in which nodes are system calls and edges connect system calls sharing some parameter. They define a behavior to be a subgraph of it and behaviors that can be adopted to distinguish malware from Goodware will be mined and used to detect malware.

Anderson et al. [41] and Huang and Stokes [26] group the system calls into high-level categories where each category represents functionally similar groups of system calls, such as painting to the screen or writing to files. Anderson et al. [41] then feed the trace of groups of system calls to a Markov chain so that they use transition probability of system calls to be the feature. Huang and Stokes [26] use those high-level API call events as binary features.

Islam et al. [22] and Dahl et al. [24] extract Windows API function calls and their parameters by running an executable in a VM. Islam et al. [22] treat Windows API functions and parameters as separate entities and use the occurrence frequency of each entity as their feature. Dahl et al. [24] use combination of a single system API call, one input parameter, and API tri-grams which consist of three consecutive API function calls, as binary features which are subsequently selected using mutual information.

Kolosnjaji et al. [27] use the dynamic malware analysis system Cuckoo sandbox to extract the sequence of the Windows system calls invoked by an executable. They use one-hot representation of them and feed the full sequence of system calls with the order to a sequential deep learning model.

Similar to assembly code, Windows API call sequences can also be obfuscated. For instance, malware authors can make an executable invoke some irrelevant API calls and submerge the API calls they use to fulfill their purpose in them. Thus, this feature is not reliable in most cases.

Control flow graphs. A control flow graph is a directed graph that represents the flow of the program, where nodes are the instructions while the edge between two nodes represents the order of sequence of execution of the two instructions. A vertex in the graph is a basic block in the middle of which there is no jump or branch instructions. A directed edge represents jumps in the control flow. Control flow graphs are used as features or signatures to detect malware in several papers [15,41].

Cesare and Xiang [15] state that similar malware usually have similar high-level structured control flows. They find that compressed and encrypted data have relatively high entropy so they first use entropy of byte sequence to detect whether an executable is packed or not. If so, they use an application level emulator to extract hidden code. They still use entropy of byte sequence to detect completion of hidden code extraction. Then the memory image of the binary is disassembled using speculative disassembly [90]. Finally, they use the process of structuring to recover high-level structured control flows from control flow graphs of procedures and represent them using strings of character tokens. The strings representing control flow graphs are all saved as signatures. An example of the relation between a control flow graph and the signature string is shown in Fig. 2.

Anderson et al. [41] also find that it is largely not easy for a polymorphic virus to build a semantically similar version of itself while changing its control flow graph enough to avoid detection. Therefore, they use control flow graphs as features. More specifically, they use the occurrence frequency of each k-graphlet (a subgraph of k nodes) in the control flow graph to represent control flow graph.

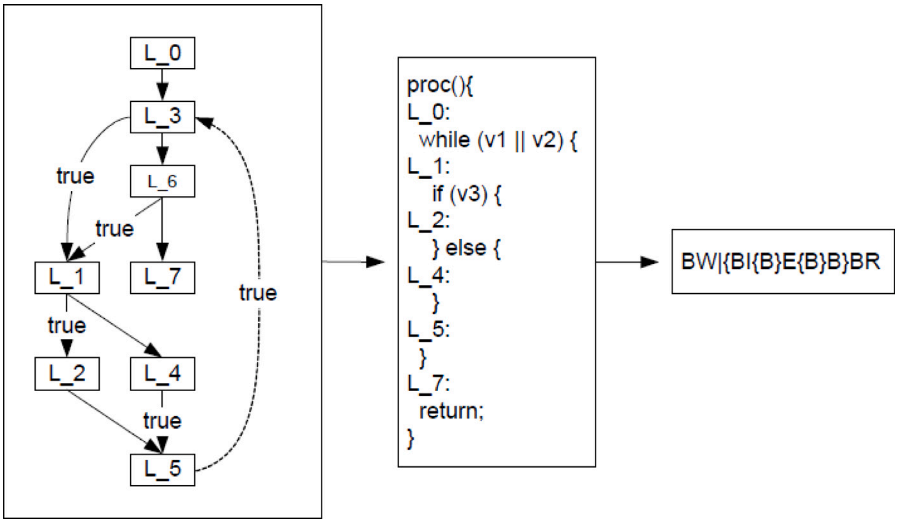
To counter the detection using control flow graphs, malware authors can use control flow flattening and bogus control flow obfuscation techniques to change the control flow without affecting the functionality so that the effectiveness of control flow graph feature will be harmed [91,92].

Function. Some papers (e.g., Islam et al. [22] and Chen et al. [14]) use function level features for malware classification.

In particular, Islam et al. [22] find function length that consists of statistically useful information in distinguishing between families of malware. After obtaining the assembly code of each executable, they calculate the length of them by measuring the number of bytes of code and use the occurrence frequency of each function lengths as a feature. However, obviously, function length is the least robust feature against

[6] http://www.clamav.net/.

[7] https://www.mcafee.com/en-us/index.html.



```
proc(){
L_0:
 while (v1 || v2) {
L_1:
  if (v3) {
L_2:
  } else {
L_4:
  }
L_5:
 }
L_7:
 return;
}
```

Fig. 2. The relationship between a control flow graph, a high level structured graph, and a signature.

obfuscation. Function length can be arbitrarily increased by inserting dead code or decreased by splitting them into multiple functions.

One should note that two functions which are semantically similar to each other are considered to be clones of each other. To this end, Chen et al. [14] assume that some files that belong to the same malware family share some functions which are connected using clone relation. So they cluster functions to groups in which any two functions can be connected directly or indirectly using clone relation and pick one function from each group as an exemplar to be a signature. They use NiCad [93] to detect whether two functions are clone to each other. However, to use one function to represent a group of functions is problematic. Since the same function evolves over generations, the newest version may look quite different from the original one. If the older version is picked as the exemplar, the clone detector may fail to identify some unknown new generation of it. Although their system works on Android APK files, the methodology can be directly applied to classifying executable malware.

Miscellaneous file information. Some miscellaneous file properties can help engineers distinguish malware from Goodware since the average or majority values of them are significantly different between the two groups. So that those properties are also used as features. They are file size [40,41], exit code [40], time consumption [40], entropy [41,94], packed or not [41], number of static/dynamic instructions [41], and number of vertices/edges in control flow graph [41]. These features may be helpful but obviously not informative enough.

Conclusive remarks. The effectiveness of using all the aforementioned features can be somehow diminished or they are not informative enough. So many papers use multiple features [12,22–26,41]. The intuition is that any single feature source can be obfuscated to evade the detection but it is extremely difficult to obfuscate all features simultaneously without hindering the functionality [22,41].

3.2. Malware classification algorithms

The extracted features introduced in the previous section are fed into malware detection/classification systems. They can be categorized as signature-based approaches and artificial intelligence-based approaches.

3.2.1. Signature-based approaches

Signature-based detection is the most papular approach used in most antivirus engines. Those signatures are created by human malware defenders through examining the collected malware samples [12,13]. More specifically, the antivirus engines detect or classify malware by checking whether the files to be analyzed contain malware signatures. The signatures of malware can take many formate including filename, text strings, or regular expressions of byte code [12,13]. Signatures are usually also hashing of the entire file. One should note that signature-based techniques can only detect malware originates from known malware which does not change significantly. As a result, attackers can exploit these techniques by hiding the malicious behavior of malware using anti-analysis techniques such as packing, obfuscation, polymorphism, and metamorphism (Section 6 provides more details about these techniques). Therefore, the code looks quite different from its original version. The main shortcoming of signature-based method is it has high precision but low recall and the other one is labor-intensive.

Some works [14–16] address the problem of manual signature crafting by proposing automatic signature generation techniques. The content of the signatures can be windows system call combinations, control flow graph, and functions.

3.2.2. Artificial intelligence-based approaches

The section discusses artificial intelligence-based malware classification approaches. These approaches can be categorized as traditional machine learning models, deep learning models, association mining, graph mining and concept analysis, and signature creation and search methods. The existing artificial intelligence-based approaches also can be classified according to the learning method used as follows: supervised, unsupervised or semi-supervised.

In a supervised malware classification model [21–25,46,50,54,55,57–65,67,69,71,72,74,76,80–82,85,95–99], the classification algorithm learns on a labeled dataset, which enable the algorithm to evaluate its accuracy on training data. In contrast, an unsupervised malware classification model [47,49,53,62,69,75,83,84,100–102], provides unlabeled data that the algorithm tries to make sense of by extracting patterns without guidance. Semi-supervised malware classification models [68,75,78,103] combine both labeled and unlabeled data.

Traditional machine learning models. The most popular traditional machine learning models used by surveyed papers are Naive Bayes classifier (NBC) [50,58,60,63–65,81], rule-based classifier [46,59,64,81,95,96], decision tree (DT) [22,23,50,55,58,60,62,65,72,74,80,82,96], K-nearest neighbors (K-NN) [22,50,60,62,71,72,96,97], Bayesian Network [23,72,85], Neural Network (NN) [24,25], Random Forest (RF) [22,54,58,60,63,67,76,80,98,99], Hidden Markov Models (HMM) [9,104–106] and Support Vector Machine (SVM) [21–23,50,54,57,58,60–63,65,69,71,72,76,81,96]. Those papers which use traditional machine learning models normally try multiple machine learning models [12,17–19,22,23].

Below, we briefly introduce the above mentioned machine learning models.

Naive Bayes Classifier (NBC) An NBC [107] uses Bayes' theorem to determine the conditional probability of a sample belonging to a class given the input features which can be formally described in the following equation:

$$P(C_i|x) = \frac{P(x|C_i)}{P(x)} P(C_i) \tag{1}$$

where x is a sample and C_i is the probability the sample belongs to class i. It is based on the Naive Bayes conditional independence assumption that all the features are independent to each other given the class it belongs to:

$$P((x_1, x_2, \dots, x_n)|C_j) = P(x_1|C_j)P(x_2|C_j)...P(x_n|C_j) \tag{2}$$

where x_j is a feature of x. Although the assumption do not hold, the prediction results are good in many occasions and the result is explainable which means how much each feature contributes is visible.

Decision Tree (DT) A DT classifier [108] uses a tree structure to represent the classification process. Internal nodes of a DT are tested on the values of features and edges correspond to a choice on values of a variable. Leaf nodes represent the final class of samples fall into it. The tree structure is constructed based on the informativeness of each feature conditioned on the current choices such as information gain ratio and Gini index. A DT is also an interpretable classifier and a DT can be translated sets of if-else-then rules.

K-Nearest Neighbor (KNN) A KNN [109] is an instance-based classifier. The model finds the K nearest neighbors of a given sample with some distance metrics (e.g., Euclidian, cosine), and predict it to be the (weighted) majority vote of the classes of the k nearest neighbors.

Support Vector Machine (SVM) An SVM [110] is a binary classifier which calculates a hyperplane that separates samples from two classes with the largest margin. An important characteristic of an SVM is it can utilize kernel trick to map samples from the original feature space to a high-dimensional (even infinite) feature space to perform non-linear classification.

Bayesian Network (BN) A BN [111] is a probabilistic graphical model which represents variables as vertices and the dependencies as directed edges. The graph is used for the inference of probability of any variable.

Rule-based classifier A rule-based classification [112] refers to any classification method that allows us to use of IF-THEN rules for prediction. An example of a rule-based classification is RIPPER [113], which is used to build a set of rules to classify samples while minimizing the error of the number of misclassified training samples.

Neural Network (NN) An NN [114] is a biologically-inspired programming paradigm that allows a computer to learn from observational data. It consists of a network of functions (i.e., parameters) which enables the computer to learn, and to fine tune itself, through analyzing new data.

Random Forest (RF) An RF classifier [115] constructs a set of DTs from the subset of training set (selected randomly). The votes are then aggregated from trees in order to decide the final class of the test sample.

Deep learning models. Deep learning models allow us to automatically abstract and extract robust and useful features for efficient and reliable malware classification. This can be done using multiple layers of abstraction to learn the "good" representation of the data [116]. An example of deep learning models are autoencoder [117], stacked denosing autoencoder [116], restricted Boltzmann Machine (RBM) [118].

Dahl et al. [24] applies their 179,000 binary features to a deep learning model. The first layer is a random projection layer which maps the input features to a much lower dimensional space (4000 dimension). The difference between the random projection layer and a normal fully connected layer is the weight of the projection matrix is not updated. The entries of it are sampled following an independent and identically distribution over -1,0,1. On top of that, they apply 1 to 3 fully connected layers with sigmoid activation functions and a 136-way softmax layer as output. They also try using a Gaussian–Bernoulli restricted Boltzmann machine (RBM) to pre-train the hidden layers. The best result is achieved by the model with 1-hidden layer without pre-training which is 9.53% test error rate. They also find the random projection performs better than Principal Component Analysis (PCA).

Saxe and Berlin [25] propose a deep feed-forward neural network consisting of four fully connected layers, where the dimensions of the first three layers are 1024 followed by a dense layer to get the output. They apply dropout to the first three layers. The activation functions of the first two layers are parametric rectified linear units (PReLU) to yield improved convergence rate without loss of performance and the activation function of the third layer is sigmoid. They also use Bayesian Calibration to calculate the unbiased probability that an executable is malware. They achieve a detection rate of 95% and a false positive rate of 0.1% on a dataset of 431,926 samples.

Huang and Stokes [26] propose a neural network for multi-task training. One task is a malware detection to predict whether an unknown software is malicious or benign and the other is to predict if it belongs to one of 98 important malware families. Huang and Stokes [26] also use a random projection layer to reduce the dimension to 4,000 from 50,000 and then they normalize each of the 4,000 dimension to be zero mean and unit variance. Then they use 4 hidden layers with dropout and RELU activation. On top of it is two single layers for each of the two classification task. The final loss function is a weighted sum of each of the individual loss functions. Experiment results show that multi-task learning only improve the performance of malware detection and harm the performance of malware classification in most experiment settings. Specifically, the best result for malware detection is 0.3577% test error which uses two hidden layers and multi-task learning and the best result for malware classification is 2.935% test error which uses one hidden layer and either single task or multi-task learning.

Kolosnjaji et al. [27] propose a combination of convolutional neural network (CNN) and Long Short-Term Memory (LSTM) networks to predict the family of an executable using the dynamically extracted system call sequence. They first use two convolution layers to capture the correlation between consecutive API calls and then apply max-pooling to reduce the dimensionality. The output sequence is fed to a LSTM layer to model the sequential dependencies of API calls. Then a mean-pooling layer is used to extract important features from the LSTM output. They also use Dropout to prevent overfitting and a softmax layer to output the probability of each class. Their proposed deep learning model significantly outperforms feed-forward neural networks, CNN, SVM, and Hidden Markov Model and achieves 85.6% on precision and 89.4% on recall. The advantage of their model is it can fully utilize the order of system calls which may also be a drawback if the system call sequence is obfuscated. One problem of their model is they use mean-pooling rather than max-pooling to extract features of highest importance produced by LSTM is not quite reasonable.

Associative classifier. An associative classifier relies on association rules that can be used to distinguish samples between two classes to perform classification. It is a special case of association rule mining where only the class of a sample can be the consequent (a.k.a. right-hand-side) of a rule. Ye et al. [16] proposes to use hierarchical associative classifiers (HAC) to classify executables based on API calls. There are three techniques regarding the creation of an associative classifier: (1) adopt FP-Growth algorithm to find candidate association rules (i.e., combination of API calls) (2) prune the candidate rules based on χ^2, data coverage, pessimistic error estimation, significance w.r.t to its ancestors (3) reorder rules: first rank the rules whose confidences are 100 by confidence support size of antecedent (CSA) and then reorder the remaining rules by χ^2 measure. Using those three techniques, they create a 2-level associative classifier to detect malware from a gray list labeled by a signature-based anti-virus engine. The first-level associative classifier is aimed for higher recall of malware. It only keeps the rules of Goodware with 100% confidence and the rules of malware with confidence greater than a pre-defined threshold; then it uses the rule pruning technique to decrease the generated rules and create the classifier; finally uses “Best First Rule” technique to find samples from the gray list. The samples labeled to be malware by the first associative classifier are fed to the second level associative classifier which is aimed at optimizing the precision. It works with the following steps: select those samples whose prediction rules of malware have 100% confidences, marking them as “confident” malware; ranking the remaining minority class files in an descending order based on their prediction rules’ χ^2 values; select the first k files from the remaining ranking list and marking them as “candidate” malware; mark the remaining files as “deep gray” files. Experiment results show the proposed HAC is effective. In addition, HAC is also an interpretable classifier which can be easily represented as simple if-then rules.

Graph mining and concept analysis. Fredrikson et al. [43] extract behaviors (dependency graphs of system calls and their parameters) that can distinguish malware from Goodware using structural leap mining [119]. Then they use the behaviors to form discriminative specifications. A specification is a set of behaviors and a characteristic function that describes one or more subsets of the set. A software matches a specification if it matches all of the behaviors in at least one characteristic subset. A specification is entirely discriminative if it matches malicious software but does not match benign software. They use formal concept analysis [120] and Simulated Annealing algorithm [121] to find an approximate optimal specification which has true positive larger than a threshold and lowest false positive among all specification larger than that true positive rate. During test, if a program matches a specification, it will be classified to be malware. The created specification can be used in the detection of unseen malware with a 86% true positive rate and 0 false positives on a dataset of 961 samples.

Signature search methods. Cesare and Xiang [15] first convert the control flow graphs of each procedure in an unknown executable to character strings in the same way they create signatures. Each procedure is assigned a weight using the length of its string:

$$weight_x = \frac{\text{len}(s_x)}{\sum_i \text{len}(s_i)} \tag{3}$$

Then they use BK Trees to retrieve the strings in the signature database which have less Levenshtein distance with strings representing procedures of the target file than a threshold. For a particular malware, once a matching graph is found, this graph is ignored for subsequent searches of the remaining graphs in the input binary. If a graph has multiple matches in a particular malware and it is uncertain which procedure should be selected as a match, the greedy solution is taken. The graph that is weighted the most is selected. For each malware that has matching signatures, the similarity ratios of those signatures:

$$w_{ed} = 1 - \frac{ed(x,y)}{\max(\text{len}(x),\text{len}(y))} \tag{4}$$

are accumulated proportional to the weights of the procedure. The final similarity between the unknown executable and a malware in the database is the product of two asymmetric similarities: a similarity that identifies how much of the input binary is approximately found in the database malware, and a similarity to show how much of the database malware is approximately found in the input binary. If the program similarity of the examined program to any malware in the database equals or exceeds a threshold of 0.6, then it is deemed to be a variant. Experiment results show that their method achieves 86% detection rate with 0 false positives which is better than 55 for commercial signature-based antivirus (AV) and 62–64 for behavior-based AV. Since they use a symmetric similarity calculated as the product of two asymmetric similarities, it cannot handle asymmetric situations. For instance, if a very large unknown executable contains the whole program of a malware sample in the database but that malicious program only take up 1% of its whole content, the similarity would still be small and it cannot be predicted to be malware.

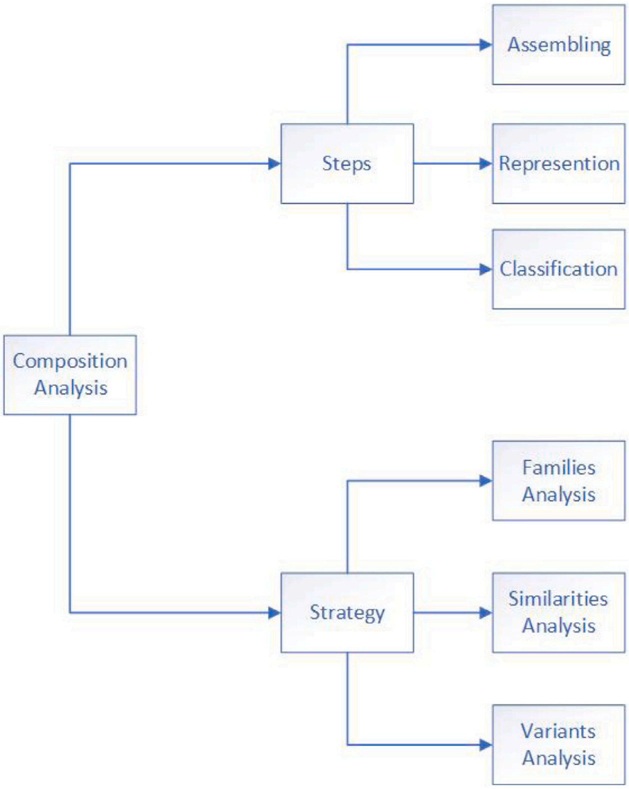


Fig. 3. The proposed taxonomy.

Chen et al. [14] uses NiCad [93] to detect whether an APK file contains any function that is clone of an exemplar function which represents a signature of a malware family. If a match is found, the file is predicted to be an instance of that malware family. They achieve 96.88% accuracy on a dataset of 1170 APK files from 19 malware families.

4. Taxonomy of composition analysis techniques

This section introduces the taxonomy of malware composition analysis techniques. We identify two major dimensions along which surveyed papers can be conveniently organized. The first one shows the steps used for composition analysis. The second dimension identifies the objective (i.e., strategy) of the analysis. Fig. 3 shows a graphical representation of the proposed taxonomy.

4.1. Steps

Composition analysis allows reverse engineers to analyze the composition of malware samples in order to understand their functionalities and behaviors. This, in turn, allows engineers to discern the intent of malware samples and the attackers. Moreover, it allows reverse engineers to rank the malware by severity and allows them to effectively triage their resources.

Basically, there are three main steps used for composition analysis: disassembling, representation, and classification.

4.1.1. Disassembling

Most software programs are delivered to users with compiled executables, rather than source code. Disassemblers make it feasible for reverse engineers to analyze software programs without source code. Technically speaking, a disassembler is a process of converting or translating machine language into assembly language. The inverse operation of "disassembler" is an "assembler". There are many tools used for this purpose (e.g., IDA Pr[8]).

Disassembly methods can be categorized into the following two classes: static techniques and dynamic techniques. Methods that belong to the first class analyze the binary components statistically, parsing the opcodes in the binary file. Methods belong to the second class monitor the execution traces of a program in order to identify the instructions and recover disassembled version of the binary.

Both dynamic and static methods have pros and cons. Static analysis takes into consideration the whole program, while dynamic analysis can only focus on the executed instructions. As a result, it is not easy to ensure that the entire executable was visited when adapting dynamic analysis. However, dynamic analysis guarantees that the output (i.e., disassembly output) only contains actual instructions.

Generally speaking, there are two approaches for static analysis techniques. The first approach is called linear sweep [122]. This approach begins at the first byte of the binary and starts decoding one instruction after another. The main shortcoming of using linear sweep disassemblers is the high probability of errors which result from data embedded in the program. The second approach is called recursive traversal [123], which allows engineers to fix the problem of "embedded data" by following the Control Flow (CF) of the program [15,41]. However, the problem with this approach is that it could fail to successfully analyze parts (i.e., functions) of the code. This is due to the fact that a control transfer instruction (e.g., jump) cannot be determined statically. This problem can be addresses by using a linear sweep algorithm to analyze unreachable regions in the code [124].

4.1.2. Representation learning

The success of any malware classification and composition analysis technique generally depends on data representation. Although specific domain knowledge may help engineers design representations and a feature vector for an executable, a manual feature engineering process fail to consider the relationships between features and define those unique patterns that can distinguish executables.

Indeed, representation learning is a set of methods and/or techniques that enables a system to automatically extract the representation needed for malware classification from raw data (i.e., assembly code). This process replaces manual feature engineering and enables a malware classification system to learn the useful features and integrates them to perform a classification.

The motivation behind using feature learning is the fact that composition analysis methods often need inputs that are robust against anti-analysis techniques such as obfuscation and packing.

Deep learning approaches (e.g., stacked autoencoders [125], stacked Denoising autoencoders [116], Deep belief networks [126], …) are known and considered as the (best) approaches for extracting robust features, which are used for building robust malware and similarity analysis tools for large-scale heterogeneous environment.

4.1.3. Classification

After disassembling executable samples, the assembly code functions are used to feed a representation learning module in order to obtain robust features and "good" representation of data. The function representation are then fed into any classification algorithms such as Naive Bayes classifier (NBC) [64], rule-based classifier [64], decision tree (DT) [65], K-nearest neighbors (K-NN) [71], Bayesian Network [85], Neural Network (NN) [24], Random Forest (RF) [67], Hidden Markov models (HMM) [127], and Support Vector Machine (SVM) [65]. The classification method enables us to identify the relationships between functions taking into account the following three analysis strategies: variants analysis, similarities analysis, and families analysis.

Variants Analysis (VA). VA [46,47,59,79,80,83] enables engineers to realize that a malware sample is actually a variant of a known malware in the repository. This strategy allows us to understand to which extent malware have been evolved over time.

Similarity Analysis (SA). SA [48,49,53,56,128] allows engineers to recognize what parts (i.e., functions) of a malware sample are similar to known functions in the repository. This strategy allows us to focus only on new parts and prevent unnecessary investigation.

Families Analysis (FA). FA [22,24,51,55,60–62,70,71,76,97,101,102]. enables engineers to associate undefined malware to defined families. This strategy works under the assumption that malware from the same family are similar to each other in terms of functionality. The difficulty to recognize them comes from the fact that some malware authors use anti-analysis techniques (e.g., obfuscation, packing, polymorphism, and metamorphism) to conceal that similarity.

5. Characterization of surveyed papers

In this section, we characterize each reviewed paper. Table 1 provides information about both algorithms and features used for each paper and highlights the main limitations. The table also shows the scalability of each work in terms of its ability to work in the presence of incremental update of the repository. The last column shows whether the proposed classification techniques are robust against anti-analysis techniques or not. As can be seen in Table 1, most of the works use more than one classification algorithm for detecting and classifying malware in order to guarantee more accurate results. In Table 2, different approaches are compared w.r.t the of the main objective: malware detection and similarity analysis, families analysis and variants analysis.

6. Challenges and issues

Based on the characterization explained in Section 5, we discuss here the challenges and/or issues of the surveyed articles.

6.1. Malware evading techniques

In this section, we introduce the common techniques that are used by malware authors to evade detection.

6.1.1. Obfuscation

The term of obfuscation mainly refers to the techniques that are used to create a variant of the original code without affecting its functionality. The purpose of obfuscation is usually to hide the real logic of the original code or to evade signature-based detector or function clone detector. A few commonly used obfuscation techniques are as follows:

1. Dead-Code Insertion [13]: insert useless instructions (e.g., nop) or insert some instructions that only affect unused variables.
2. Code Transposition [13]: change the order of the independent instructions.
3. Register Reassignment [13]: exchange the usage of registers for the storage of data/address in a specific live range.
4. Instruction Substitution [13]: replace an instruction with equivalent instructions.

[8] https://www.hex-rays.com/products/ida/.

Table 1
Summary of extraction methods, classification methods, and limitation in malware classification.

Work	Classification method	Features	Limitations	Scalability (Yes/No)	Robust against noisy inputs (Yes/No)
[129]	k-NN and SVM	Byte Code	Not robust against unseen inputs	Yes	No
[130]	NN	Byte Code	Vulnerable to adversarial attacks	Yes	Yes
[131]	k-NN and NN	Byte Code	Vulnerable to adversarial attacks	Yes	Yes
[65]	DT, Naïve Bayes, and SVM	Byte Code	Not robust against noisy inputs	Yes	No
[132]	k-NN, NN, and SVM	Byte Code	Vulnerable to adversarial attacks	Yes	Yes
[73]	RF	Miscellaneous File Information	Needs a large number of labeled examples (malicious and benign)	Yes	Yes
[74]	DT, RF	Miscellaneous File Information	Works only under the assumption that the new samples are not packed	Yes	No
[57]	SVM	Internet Traffic	Not scalable (tested using vary small datasets)	No	Yes
[75]	Cluster Analysis	Miscellaneous File Information	Unable to classify new examples/samples	Yes	No
[64]	NBC	Printable Strings and Byte Code	Not robust against noisy inputs	Yes	No
[96]	DT, NBC, SVM	API	Not scalable (tested using very small datasets)	No	Yes
[103]	BN	Miscellaneous File Information	Not efficient giving new samples	Yes	No
[50]	DT, NBC, SVM, k-NN, NN and SVM	API and Miscellaneous File Information	Not scalable (tested using small datasets)	No	Yes
[21]	SVM	Byte Code and API	Not scalable (tested using very small datasets)	No	Yes
[41]	SVM	Byte Code, Assembly Codes and API	not scalable (tested using very small datasets)	No	Yes
[85]	BN	API	Not robust against noisy inputs	Yes	No
[23]	BN, DT, k-NN classification, SVM	Assembly Codes and API	Not robust against noisy inputs	Yes	No
[58]	DT, RF, Naïve Bayes, SVM	Byte Code and API	Not scalable (tested using very small datasets)	No	Yes
[78]	BN	Miscellaneous File Information	Not robust against unseen inputs	Yes	No
[59]	Rule-based classifier	API	Not scalable (tested using very small datasets)	No	Yes
[98]	RF	Internet Traffic	Not robust against unseen inputs	Yes	No
[99]	RF	API and Miscellaneous File Information	Not robust against noisy inputs	Yes	No
[25]	NN	Printable Strings and Miscellaneous File Information	Not robust against noisy inputs and not scalable (tested using very small datasets)	No	yes
[46]	Rule based classification	API and Miscellaneous File Information	not scalable (tested using very small datasets)	No	Yes
[47]	Cluster analysis	API and Miscellaneous File Information	Requiring user interactions	Yes	No
[101]	Cluster analysis	Byte Code	Not scalable (tested using small datasets)	No	Yes
[51]	Matching (graph theory)	API	Not robust against noisy inputs	Yes	No
[102]	Cluster analysis	Assembly Codes	Not robust against noisy inputs	Yes	No

(*continued on next page*)

Table 1 (*continued*).

Work	Classification method	Features	Limitations	Scalability (Yes/No)	Robust against noisy inputs (Yes/No)
[24]	NN	Byte Code and API	High error rate	Yes	No
[70]	Clustering	Assembly Codes	Not robust against noisy inputs	Yes	No
[22]	DT, k-NN classification, RF, SVM	Byte Code and API	Not robust against unseen inputs	Yes	No
[71]	k-NN classification and SVM	Assembly Codes and Miscellaneous File Information	Not robust against unseen inputs	Yes	No
[55]	DT	Internet Traffic	Not scalable (tested using very small datasets)	No	Yes
[76]	SVM, RF and DT	Internet Traffic and Byte Code, Assembly Codes and API	Not robust against noisy inputs	Yes	No
[61]	SVM, RF and DT	Internet Traffic and Byte Code and API	Not scalable (tested using very small datasets)	No	Yes
[60]	DT, RF, k-NN classification and NBC	API	Not robust against unseen inputs	Yes	No
[62]	DT, k-NN classification and SVM	Miscellaneous File Information and network	Not robust against noisy inputs	Yes	No
[133]	k-Means	Assembly Codes	Not robust against noisy inputs	Yes	No
[48]	Hierarchical Clustering	API, Miscellaneous File Information, and Internet Traffic	Not scalable (tested using very small datasets). Not robust against noisy inputs	Yes	No
[49]	Cluster analysis	API	Not robust against noisy inputs	Yes	No
[53]	Cluster analysis	Byte Code and API	Not robust against noisy inputs	Yes	No
[56]	NN	API	Not robust against noisy inputs and not scalable (tested using small datasets)	No	Yes
[72]	DT, k-NN classification, BN and RF	Assembly codes	not scalable (tested using very small datasets)	No	Yes
[63]	NBC, RF, and SVM	Byte Code, API and file system	Not robust against noisy inputs	Yes	No
[97]	k-NN classification	Byte Code	Not robust against noisy inputs	Yes	No
[104]	HMM	opcode sequences	Not robust against severe obfuscations techniques	Yes	Yes
[105]	HMM	mnemonic opcode sequences	Not robust against severe obfuscations techniques	Yes	Yes
[106]	HMM	opcode sequences	Not robust against severe obfuscations techniques	Yes	Yes
[9]	HMM	opcode sequences	Not robust against severe obfuscation techniques	Yes	Yes

5. Control Flow Flattening [134]: (1) break up the body of the function to basic blocks (2) put all basic blocks which were originally at different nesting levels next to each other (3) encapsulate the basic blocks in a selective structure (a switch statement in the C++) (4) encapsulate the selection in a loop.
6. Bogus Control Flow [135]: for a basic block, add a new basic block which contains an opaque predicate and then make a conditional jump to the original basic block.

6.1.2. *Packing*

Packing is a technique to compress/encrypt an executable, where those packed files will be uncompressed/decrypted during runtime. It means that a static analyzer cannot see the real code since it does not run the executable. Packing is used not only for malware but also for the protection of Goodware schemes [15,41]. According to the statistics conducted by Anderson et al. [41], 47.56% of the malware are packed and 19.59% of the Goodware are packed in their dataset.

6.1.3. *Polymorphism*

Polymorphism is also a technique that is based on encryption and decryption. A polymorphic malware contains two parts: the polymorphism engine and the real program which performs the malicious functions. The former mutates the encryption algorithms and keys when it replicates and the code of the latter per se is fixed but it is encrypted by the former in different ways during runtime. This way, the whole polymorphic malware program would look different at each generation [136].

6.1.4. *Metamorphism*

A metamorphic malware re-programs itself when it replicates. Consequently, in each generation, the whole program body is modified using code obfuscation techniques while the functionality is kept unchanged [136]. Metamorphic malware is considered to be more difficult to write than polymorphic malware.

Table 2
Comparison summary (SA: Similarity Analyzes; FA: Families Analysis; VA: Variants Analysis.

Paper	Detection	SA	FA	VA
Schultz et al [64]	✓			
Kolter and Maloof [65]	✓			
Ahmed et al. [96]	✓			
Chau et al. [103]	✓			
Firdausi et al. [50]	✓			
Anderson et al. [21]	✓			
Anderson et al. [41]	✓			
Eskandari et al. [85]	✓			
Santos et al. [23]	✓			
Vadrevu et al. [73]	✓			
Bai et al. [74]	✓			
Kruczkowski and Szynkiewicz [57]	✓			
Tamersoy et al. [75]	✓			
Uppal et al. [58]	✓			
Chen et al. [78]	✓			
Ghiasi et al. [59]	✓			✓
Kwon et al. [98]	✓			
Mao et al. [99]	✓			
Saxe and Berlin [25]	✓			
Wuchner et al. [63]	✓			
Raff and Nicholas [97]	✓		✓	
Gharacheh et al. [79]				✓
Khodamoradi et al. [80]				✓
Upchurch et al. [83]				✓
Liang et al. [46]				✓
Vadrevu and Perdisci [47]				✓
Huang et al. [101]			✓	
Park et al. [51]			✓	
Ye et al. [102]			✓	
Dahl et al. [24]			✓	
Hu et al. [70]			✓	
Islam et al. [22]			✓	
Kong and Yan [71]			✓	
Nari and Ghorbani [55]			✓	
Ahmadi et al. [76]			✓	
Lin et al. [61]			✓	
Kawaguchi and Omote [60]			✓	
Mohaisen et al. [62]			✓	
Pai et al. [133]		✓		
Bailey et al. [48]		✓		
Bayer et al. [49]		✓		
Chen et al. [14]			✓	
Cesare and Xiang [15]			✓	
Anderson et al. [41]			✓	
Cordy et al. [93]			✓	
Fredrikson et al. [43]			✓	
Rieck et al. [53]		✓		
Palahan et al. [56]		✓		
Santos et al. [72]		✓		
Egele et al. [128]		✓		
Kolter and Maloof [17]	✓			
Moskovitch et al. [18]	✓			

6.2. Adversarial attack and defense

Since the direction of the recent research is to automate the process of malware analysis using machine learning techniques, the proposed solutions should be robust against adversarial examples, which are inputs designed by an attacker to fool the machine learning models and make it generate erroneous decisions (e.g., making the malware analysis tools unable to detect malicious code). It has been recently shown that machine learning models, including deep neural networks, are quite vulnerable to adversarial examples. It is easy for an attacker to create “adversarial examples” [137] to fool a machine learning model through simply perpetuating parts of the inputs.

6.2.1. Adversarial attack

Adversarial samples are crafted from normal samples with minimum perturbations on input variables to confuse a classifier without breaking the functionality of the original samples. It is natural that the perturbations should be based on the derivative of the loss function with respect to the classifier’s input variables since derivatives show the directions of changes on the input that is the most effective for changing the output. So a differentiable classifier is required to create adversarial samples and deep learning models are just differentiable and effective classifiers. Studies show that adversarial samples generated to fool one model can fool a totally different model [138,139]. Therefore, as deep learning models are proposed for the malware detection field, malware authors have better opportunities to craft adversarial examples to evade the detection of any machine learning models.

A formal description of the problem to craft an adversarial x^* to be misclassified by a classifier f is

$$min \ \|\delta_x\| \tag{5}$$

$$s.t. \quad x^* = x + \delta_x, f(x^*) \neq f(x) \tag{6}$$

where $\|\cdot\|$ can be any norm and x is the sample to be perturbed.

Goodfellow et al. [140] present a fast gradient sign method in which the adversarial perturbation is determined by multiplying the gradients’ sign of the sample S with some coefficient to control the scale of perturbation. Papernot et al. [141] propose a forward derivative method which evaluates the sensitivity of the output to each input component using its Jacobian matrix and then constructs adversarial saliency maps based on the Jacobian matrix, indicating which input features to be included in the perturbation.

Compared with perturbing an adversarial image sample, there are some constraints on perturbing a malware sample since most of the features of malware are discrete rather than real-valued and the functionality should be intact. Thus, previous methods for perturbation of real-valued features need to be adapted and some binary features cannot be changed from “1” to “0” since “1” means that the feature exists and that the change in this direction may break the functionality.

Grosse et al. [28] propose a technique to craft adversarial Android malware. Inspired by Papernot et al. [28,141] use the Jacobian matrix to examine which features have the greatest potential to lead to the prediction of a malicious program as being Goodware. They only allow distortions to no more than 20 features. All the features are binary features. To maintain the functionality of the adversarial example, they add two constraints: (1) only adjust manifest features that relate to the AndroidManifest.xml file. This file is available in any Android application; (2) it should be done by adding a single line of code to it. Using their method, a state-of-the-art feed-forward neural network which achieves 98% of accuracy on the original dataset is misled by 63% of the adversarial malware samples.

6.2.2. Adversarial defense

Grosse et al. [28] try two methods to defend against adversarial attack. The first is to apply distillation [141,142] to counter adversarial samples, which successfully reduces misclassification rate by 38.5% in some case. The second is adversarial training [140] which consists of training the model on the original dataset and then training the model again only on the adversarial samples for a few epochs. The misclassification rate is reduced to 67% from 73% through adversarial training.

Wang et al. [29] defend against adversarial attacks by randomly nullifying input features. Their nullification is similar to dropout since in both mechanisms some input features are randomly set to 0. The main difference with dropout is that the model do not drop any input feature during the test but in nullification some features are still dropped randomly during the test. Specifically, for each sample in any dataset, a nullification rate is sampled under a Gaussian distribution and the dimensions (features) to drop are sampled uniformly. The intuition is that nullification makes their architecture non-deterministic so that the attackers cannot examine the importance of features and so it is hard for them to detect and exploit the “blind spots” of classifiers.

In their experiments, the features are the invoked windows system DLL files and they use Jacobian-based saliency map to pick up to 10 features for each sample to perturb. Experimental results show that their method can improve the resistance to adversarial samples and that the best resistance is 64.86% and is achieved with a nullification rate of 10%. However, a theoretic problem of their approach is when adversarial samples are cross-model [138,139]. Thus, even though nullification can harm the ability of an adversary to use this model to craft adversarial samples, the adversary can use other models (i.e., the same neural network without nullification) to craft adversarial samples which can also evade the one equipped with nullification. Therefore, there is no theoretic proof or evidence to show whether nullification can improve the resistance against adversarial samples crafted from other deep learning models.

6.3. Efficiency and scalability

A practical malware search engine can help security engineers obtain malware search results on-the-fly when they are making analysis. Instant feedback provides the engineer the structure of a given malware that is under investigation [92]. One should note that scalability is an important factor as the number of malware in the database needs to scale up to millions. It is also a critical issue for producing a reliable malware search engine. For practical applications, a malware search engine' efficiency and scalability should be evaluated using a large repository in order to measure both its accuracy and latency.

7. Research direction

The above contributions are effective in addressing some interesting research gaps in the literature. However, some points still need further study and investigation. The following research avenues could be further explored based on our literature review:

7.1. Robust solutions

Although the discussed solutions in the literature review have paved the road for a reliable Malware Detection System (MDS) through extracting robust and useful features, the solution still needs to reduce human interaction. Thus, an automated system is required to take the data and automatically abstract and extract robust features from them. For this purpose, deep learning techniques could be the best candidate to replace the existing feature extraction approaches. The solution can be designed and implemented using different Deep Learning architectures (e.g., Generative Adversarial Networks, Stacked Denoising Autoencoder, Restricted Boltzmann Machine, and Variational Autoencoder) for auto-abstraction and extraction of robust features to significantly enhance the detection under heterogeneous, changing and noisy environments.

Recently, Ding et al. [143] propose a robust and accurate assembly clone search platform named Asm2Vec. The proposed platform enables engineers to automatically learns a vector representation of any assembly function by discriminating it from others functions. Also, the platform allows engineers to jointly learn the semantic relationships of assembly functions based on assembly code [143]. This, in turn enables us to construct useful and robust features to make efficient and reliable assembly clone search. The proposed learning representation is inspired by the Distributed Memory Model of Paragraph Vectors (PV-DM) model, which is used to learn a vectorized representation of a text paragraph [144]. The PV-DM model is fundamentally based on Word2Vec [145], which is used to learn vector representation of words. This is done by enabling words with similar meaning to be mapped to a similar position in the vector space. For example, "good" and "great" are close to each other, whereas "great" and "Japan" are more distant. Learning the vector representation of words becomes possible thanks to the concept of Distributed Vector Representation (DVR) of words, a well known method used for learning the word vectors. In particular, DVS exploits the power of machine learning models (usually Neural Networks) by training machine learning models to predict a word (i.e., target word) given the other words in a context. In the process of predicting the target word, we learn the vector representation of the target word.

The PV-DM model is inspired by Word2Vec by using the idea for learning the word vectors. In the PV-DM model, both word vectors and paragraph vectors are asked to contribute to the prediction of the target word given many contexts sampled from the paragraph [144]. This process (i.e., predicting the target word) allows us to learn the vector representation of the paragraph. Ding et al. [143] exploit the power of the PV-DM model to learn the vector representation of assembly functions based on assembly code. This is done by mapping assembly function (i.e., repository function) and the function's input tokens (i.e., instructions) to a unique vector. The machine learning model is then trained to predict a target token given the function and its tokens in a context. This process enables us to learn the vector representation of the function.

In fact, the solution should be able not only to accommodate unknown variants of known malware but also to accommodate unknown variants of unknown malware. These solutions should also be robust against adversarial attacks. Although some works have already addressed this problem, these solutions are mostly based on adversarial training [146] and are not mature enough to combine the extraction of robust and useful features to protect the system against adversarial examples. Thus, the solution should not only be robust against complex and noisy data but also against adversarial examples.

7.2. Collaborative solutions

Computer and communication systems are becoming more and more complex and vulnerable to intrusions. Cyber attacks are also becoming more complex and harder to analyze and recognize. In fact, it became increasingly difficult for a single MDS to recognize all intrusions, because of limited knowledge about the evolution of malware. The recent works in intrusion detection and malware analysis [147–149] have shown experimentally that the detection accuracy can be significantly improved, compared to the traditional single MDS, when MDSs cooperate with each other. In collaborative environment, each MDS can consult other MDSs about suspicious malware to increase the decision accuracy. Fig. 4 shows an example of cooperative MDS.

Recently, Man and Huh [147] and Singh et al. [148] design a collaborative MDS, which enables malware-detection-alerts to be exchanged from different distributed detectors. Moreover, knowledge are enabled to be exchanged between nodes. In addition, Dermott et al. [150] propose a collaborative MDS in a cloud-computing environment. The proposed framework use the Dempster-Shafer theory of evidence [151] in order to combine the decisions form different malware detectors. The received decisions are aggregated to take the final decision regarding a suspicious malware. This technique has a shortcoming: its centralized-based architecture, whereby a reliable third-party is used for combining feedback and coordinating MDS.

In fact, the design of a cooperative MDS should take into consideration the following three properties (challenges): trustworthiness, fairness and sustainability. By trustworthiness, we mean that the MDS should be able to ensure that it will consult, cooperate and share knowledge with trusted parties (i.e., MDSs). By fairness, we mean that the MDS should be able to guarantee that mutual benefits will be achieved through minimizing the chance of cooperating with selfish MDSs. This is useful to give MDSs the motivation to participate in the community. Finally, by sustainability, we mean enabling an MDS to proactively take decisions about suspicious attacks, regardless if the complete feedback have been received from consulted MDSs or not. Thus, the proposed solution will be applicable in real-time environments, where MDSs should take decisions about suspicious malware quickly.

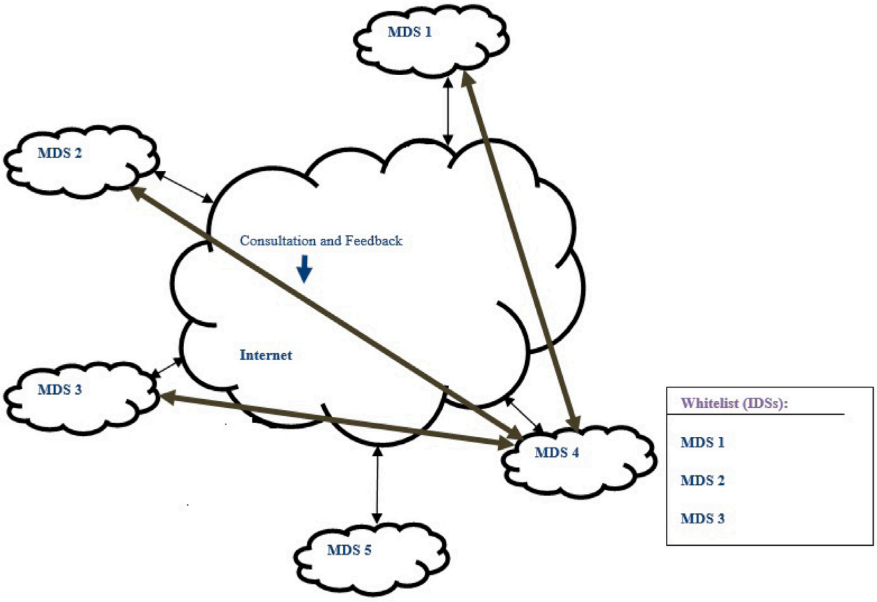


Fig. 4. The proposed taxonomy.

7.3. Sustainable solutions

The power of most malware analysis tools is largely based on the amount of knowledge that they have about Malware and dangerous attacks. In fact, supervised machine learning algorithms such as SVM, used by MDS, are heavily dependent on labeled data to learn how to effectively classify malicious and normal behaviors [152]. However, obtaining data on malicious behaviors is challenging and dangerous, especially if we are required to launch real attacks on production systems and put users, applications and systems at risk. To address this problem, we may need to have an efficient approach to synthesize new malware and augment our training data, in order to improve machine learning-based MDSs.

Generative models such as Generative adversarial Networks (GANs) [153] can be used to generate synthetic malware and enhance the detection accuracy of machine learning-based MDS, by augmenting Malware training sets. We encourage researchers to investigate the use of GANs, which have shown unprecedented ability in generating high quality new synthetic data, to generate malware variants. In particular, they need to design new algorithms to effectively and efficiently train GANs on the existing malware that are available in the repository in order to learn how to generate variants of them. To this end, researchers are required to collect a large volume of malware samples that consists of different attributes (vulnerabilities, targeted users, targeted hosts, etc.) from the public domain. Since GANs are only defined for real-valued, continued data and the design of malware is based on sequences of discrete tokens (bytes), special extensions should be applied on the original GANs theory. For example, we may need to integrate GANs with recurrent neural networks (RNNs) to tackle the problem of sequenced data [154]. Moreover, to address the problem of discrete data, we may need to place in parallel a dense layer per categorical variable, followed by Gumbel-Softmax activation and a concatenation to get the final output [155].

8. Conclusion

In this paper, we provide a comprehensive survey on publications that contributed to malware classification and composition analysis. There are four main contributions in our work. First, we proposed an organization of reviewed paper according to three dimensions: the purpose of the analysis (malware classification or composition analysis), the type of features obtained from samples, and the algorithms used to manipulate these features. Second, we provided a comparative analysis of the existing malware classification and composition analysis techniques, while structuring them according to the proposed taxonomy. Third, We determined the main issues and challenges associated with malware classification and composition analysis. Finally, we identified a number of emergent topics in the discussed field, such as collaborative malware analysis system, with guidelines on how to improve solutions to address the new challenges.

The above contributions are effective in addressing some interesting research gaps in the literature. However, some points still need further study and investigation. The following research avenues could be further explored in order to achieve better accuracy and efficient solutions compared to the state-of-the-art. The first avenue is the design of cooperative MDS to address the problem of limited and incomplete knowledge about malware. Through collaboration, an MDS can consult other MDSs about suspicious malware and increase the decision accuracy. To this end, we identify three challenges that should be addressed in cooperative MDS: trustworthiness, fairness and sustainability. Second, the design of robust MDS by enabling the automatic extraction of robust features from samples. The solution should be able not only to accommodate unknown variants of known malware but also to accommodate unknown variants of unknown malware. Moreover, the solution should be robust against adversarial attacks. Finally, the design of sustainable MDS by enabling an MDS to synthetically generate new malicious and benign code in order to enhance the accuracy of machine learning-based malware classification methods.

CRediT authorship contribution statement

Adel Abusitta: Conceptualization, Methodology, Data curation, Writing - original draft, Validation, Writing - reviewing and editing, Supervision, Visualization, Investigation. **Miles Q. Li:** Conceptualization, Methodology, Data curation, Writing - original draft, Validation, Writing - reviewing and editing, Visualization, Investigation. **Benjamin C.M. Fung:** Conceptualization, Methodology, Supervision, Funding acquisition, Writing - original draft, Writing - reviewing and editing, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported in part by the DND Innovation for Defence Excellence and Security, Canada (W7714-207117/001/SV), NSERC, Canada Discovery Grants (RGPIN-2018-03872), and Canada Research Chairs Program (950-230623). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Malware statistics and facts for 2020. 2020. https://www.comparitech.com/antivirus/malware-statistics-facts/. [Accessed 17 March 2020].
[2] Malware Numbers 2017. 2019. https://www.gdatasoftware.com/blog/2018/03/30610-malware-number-2017. [Accessed 17 August 2019].
[3] Suarez-Tangil G, Tapiador JE, Peris-Lopez P, Ribagorda A. Evolution, detection and analysis of malware for smart devices. IEEE Commun Surv Tutor 2013;16(2):961–87.
[4] Tailor JP, Patel AD. A comprehensive survey: ransomware attacks prevention, monitoring and damage control. Int J Res Sci Innov 2017;4(15):116–21.
[5] Vignau B, Khoury R, Hallé S. 10 years of IoT malware: A feature-based taxonomy. In: 2019 IEEE 19th international conference on software quality, reliability and security companion. IEEE; 2019, p. 458–65.
[6] Xu Z, Wang H, Xu Z, Wang X. Power attack: An increasing threat to data centers. In: NDSS. 2014.
[7] Kimani K, Oduol V, Langat K. Cyber security challenges for IoT-based smart grid networks. Int J Crit Infrastruct Prot 2019;25:36–49.
[8] Jakobsson M, Ramzan Z. Crimeware: understanding new attacks and defenses. Addison-Wesley Professional; 2008.
[9] Wong W, Stamp M. Hunting for metamorphic engines. J Comput Virol 2006;2(3):211–29.
[10] Tariq N. Impact of cyberattacks on financial institutions. J Internet Bank Commer 2018;23(2):1–11.
[11] Chen L, Ye Y, Bourlai T. Adversarial machine learning in malware detection: Arms race between evasion attack and defense. In: 2017 European intelligence and security informatics conference. IEEE; 2017, p. 99–106.
[12] Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Security and privacy, 2001. S&P 2001. Proceedings. 2001 IEEE symposium on. IEEE; 2001, p. 38–49.
[13] Christodorescu M, Jha S. Static analysis of executables to detect malicious patterns. Technical report, Wisconsin Univ-Madison Dept of Computer Sciences; 2006.
[14] Chen J, Alalfi MH, Dean TR, Zou Y. Detecting android malware using clone detection. J Comput Sci Tech 2015;30(5):942–56.
[15] Cesare S, Xiang Y. Classification of malware using structured control flow. In: Proceedings of the eighth Australasian symposium on parallel and distributed computing-volume 107. Australian Computer Society, Inc.; 2010, p. 61–70.
[16] Ye Y, Li T, Huang K, Jiang Q, Chen Y. Hierarchical associative classifier (HAC) for malware detection from the large and imbalanced gray list. J Intell Inf Syst 2010;35(1):1–20.
[17] Kolter JZ, Maloof MA. Learning to detect malicious executables in the wild. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM; 2004, p. 470–8.
[18] Moskovitch R, Feher C, Tzachar N, Berger E, Gitelman M, Dolev S, Elovici Y. Unknown malcode detection using opcode representation. In: Intelligence and security informatics. Springer; 2008, p. 204–15.
[19] Dai J, Guha RK, Lee J. Efficient virus detection using dynamic instruction sequences. J Comput Phys 2009;4(5):405–14.
[20] Nataraj L, Karthikeyan S, Jacob G, Manjunath B. Malware images: visualization and automatic classification. In: Proceedings of the 8th international symposium on visualization for cyber security. ACM; 2011, p. 4.
[21] Anderson B, Quist D, Neil J, Storlie C, Lane T. Graph-based malware detection using dynamic analysis. J Comput Virol 2011;7(4):247–58.
[22] Islam R, Tian R, Batten LM, Versteeg S. Classification of malware based on integrated static and dynamic features. J Netw Comput Appl 2013;36(2):646–56.
[23] Santos I, Devesa J, Brezo F, Nieves J, Bringas PG. Opem: A static-dynamic approach for machine-learning-based malware detection. In: International joint conference CISIS'12-ICEUTE 12-SOCO 12 special sessions. Springer; 2013, p. 271–80.
[24] Dahl GE, Stokes JW, Deng L, Yu D. Large-scale malware classification using random projections and neural networks. In: Acoustics, speech and signal processing, 2013 IEEE international conference on. IEEE; 2013, p. 3422–6.
[25] Saxe J, Berlin K. Deep neural network based malware detection using two dimensional binary program features. In: Malicious and unwanted software, 2015 10th international conference on. IEEE; 2015, p. 11–20.
[26] Huang W, Stokes JW. MtNet: a multi-task neural network for dynamic malware classification. In: International conference on detection of intrusions and malware, and vulnerability assessment. Springer; 2016, p. 399–418.
[27] Kolosnjaji B, Zarras A, Webster G, Eckert C. Deep learning for classification of malware system call sequences. In: Australasian joint conference on artificial intelligence. Springer; 2016, p. 137–49.
[28] Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P. Adversarial examples for malware detection. In: European symposium on research in computer security. Springer; 2017, p. 62–79.
[29] Wang Q, Guo W, Zhang K, Ororbia II AG, Xing X, Liu X, Giles CL. Adversary resistant deep neural networks with an application to malware detection. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2017, p. 1145–53.
[30] Ucci D, Aniello L, Baldoni R. Survey of machine learning techniques for malware analysis. Comput Secur 2018.
[31] Sahu MK, Ahirwar M, Hemlata A. A review of malware detection based on pattern matching technique. Int J Comput Sci Inf Technol 2014;5(1):944–7.
[32] Souri A, Hosseini R. A state-of-the-art survey of malware detection approaches using data mining techniques. Human-centric Comput Inf Sci 2018;8(1):3.
[33] Bazrafshan Z, Hashemi H, Fard SMH, Hamzeh A. A survey on heuristic malware detection techniques. In: The 5th conference on information and knowledge technology. IEEE; 2013, p. 113–20.
[34] Shabtai A, Moskovitch R, Elovici Y, Glezer C. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. Inf Secur Tech Rep 2009;14(1):16–29.
[35] Basu I, Sinha N, Bhagat D, Goswami S. Malware detection based on source data using data mining: A survey. Am J Adv Comput 2016;3(1):18–37.
[36] Ye Y, Li T, Adjeroh D, Iyengar SS. A survey on malware detection using data mining techniques. ACM Comput Surv 2017;50(3):41.
[37] Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic malware analysis in the modern era—A state of the art survey. ACM Comput Surv 2019;52(5):88.
[38] Barriga J, Yoo S. Malware detection and evasion with machine learning techniques: A survey. Int J Appl Eng Res 2017;12(318).
[39] Damodaran A, Di Troia F, Visaggio CA, Austin TH, Stamp M. A comparison of static, dynamic, and hybrid analysis for malware detection. J Comput Virol Hacking Tech 2017;13(1):1–12.
[40] Bayer U, Moser A, Kruegel C, Kirda E. Dynamic analysis of malicious code. J Comput Virol 2006;2(1):67–77.
[41] Anderson B, Storlie C, Lane T. Improving malware classification: bridging the static/dynamic gap. In: Proceedings of the 5th ACM workshop on security and artificial intelligence. ACM; 2012, p. 3–14.
[42] Royal P, Halpin M, Dagon D, Edmonds R, Lee W. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Computer security applications conference, 2006. ACSAC'06. 22nd annual. IEEE; 2006, p. 289–300.
[43] Fredrikson M, Jha S, Christodorescu M, Sailer R, Yan X. Synthesizing near-optimal malware specifications from suspicious behaviors. In: Security and privacy, 2010 IEEE symposium on. IEEE; 2010, p. 45–60.
[44] Force UA. Analysis of the Intel Pentium's ability to support a secure virtual machine monitor. In: Proceedings of the 9th USENIX security symposium. 2000. p. 129.
[45] Rutkowska J. Redpill: Detect VMM using (almost) one CPU instruction. 2004, http://invisiblethings.org/papers/redpill.html.
[46] Liang G, Pang J, Dai C. A behavior-based malware variant classification technique. Int J Inf Educ Technol 2016;6(4):291.
[47] Vadrevu P, Perdisci R. Maxs: Scaling malware execution with sequential multi-hypothesis testing. In: Proceedings of the 11th ACM on Asia conference on computer and communications security. ACM; 2016, p. 771–82.
[48] Bailey M, Oberheide J, Andersen J, Mao ZM, Jahanian F, Nazario J. Automated classification and analysis of internet malware. In: International workshop on recent advances in intrusion detection. Springer; 2007, p. 178–97.
[49] Bayer U, Comparetti PM, Hlauschek C, Kruegel C, Kirda E. Scalable, behavior-based malware clustering. In: NDSS, vol. 9. Citeseer; 2009, p. 8–11.
[50] Firdausi I, Erwin A, Nugroho AS, et al. Analysis of machine learning techniques used in behavior-based malware detection. In: 2010 second international conference on advances in computing, control, and telecommunication technologies. IEEE; 2010, p. 201–3.
[51] Park Y, Reeves D, Mulukutla V, Sundaravel B. Fast malware classification by automated behavioral graph matching. In: Proceedings of the sixth annual workshop on cyber security and information intelligence research. ACM; 2010, p. 45.
[52] Lindorfer M, Kolbitsch C, Comparetti PM. Detecting environment-sensitive malware. In: International workshop on recent advances in intrusion detection. Springer; 2011, p. 338–57.
[53] Rieck K, Trinius P, Willems C, Holz T. Automatic analysis of malware behavior using machine learning. J Comput Secur 2011;19(4):639–68.

[54] Comar PM, Liu L, Saha S, Tan P-N, Nucci A. Combining supervised and unsupervised learning for zero-day malware detection. In: 2013 Proceedings IEEE INFOCOM. IEEE; 2013, p. 2022–30.
[55] Nari S, Ghorbani AA. Automated malware classification based on network behavior. In: 2013 international conference on computing, networking and communications. IEEE; 2013, p. 642–7.
[56] Palahan S, Babić D, Chaudhuri S, Kifer D. Extraction of statistically significant malware behaviors. In: Proceedings of the 29th annual computer security applications conference. ACM; 2013, p. 69–78.
[57] Kruczkowski M, Szynkiewicz EN. Support vector machine for malware analysis and classification. In: Proceedings of the 2014 IEEE/WIC/ACM international joint conferences on web intelligence (WI) and intelligent agent technologies (IAT)-Volume 02. IEEE Computer Society; 2014, p. 415–20.
[58] Uppal D, Sinha R, Mehra V, Jain V. Malware detection and classification based on extraction of api sequences. In: 2014 international conference on advances in computing, communications and informatics. IEEE; 2014, p. 2337–42.
[59] Ghiasi M, Sami A, Salehi Z. Dynamic VSA: a framework for malware detection based on register contents. Eng Appl Artif Intell 2015;44:111–22.
[60] Kawaguchi N, Omote K. Malware function classification using APIs in initial behavior. In: 2015 10th Asia joint conference on information security. IEEE; 2015, p. 138–44.
[61] Lin C-T, Wang N-J, Xiao H, Eckert C. Feature selection and extraction for malware classification. J Inf Sci Eng 2015;31(3):965–92.
[62] Mohaisen A, Alrawi O, Mohaisen M. Amal: High-fidelity, behavior-based automated malware analysis and classification. Comput Secur 2015;52:251–66.
[63] Wüchner T, Ochoa M, Pretschner A. Robust and effective malware detection through quantitative data flow graph metrics. In: International conference on detection of intrusions and malware, and vulnerability assessment. Springer; 2015, p. 98–118.
[64] Schultz MG, Eskin E, Zadok F, Stolfo SJ. Data mining methods for detection of new malicious executables. In: Proceedings 2001 IEEE symposium on security and privacy. IEEE; 2000, p. 38–49.
[65] Kolter JZ, Maloof MA. Learning to detect and classify malicious executables in the wild. J Mach Learn Res 2006;7(Dec):2721–44.
[66] Attaluri S, McGhee S, Stamp M. Profile hidden Markov models and metamorphic virus detection. J Comput Virol 2009;5(2):151–69.
[67] Siddiqui M, Wang MC, Lee J. Detecting internet worms using data mining techniques. J Syst Cybern Inform 2009;6(6):48–53.
[68] Santos I, Nieves J, Bringas PG. Semi-supervised learning for unknown malware detection. In: International symposium on distributed computing and artificial intelligence. Springer; 2011, p. 415–22.
[69] Chen Z, Roussopoulos M, Liang Z, Zhang Y, Chen Z, Delis A. Malware characteristics and threats on the internet ecosystem. J Syst Softw 2012;85(7):1650–72.
[70] Hu X, Shin KG, Bhatkar S, Griffin K. Mutantx-s: Scalable malware clustering based on static features. In: Proceedings of the USENIX annual technical conference. 2013. p. 187–98.
[71] Kong D, Yan G. Discriminant malware distance learning on structural information for automated malware classification. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2013, p. 1357–65.
[72] Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inform Sci 2013;231:64–82.
[73] Vadrevu P, Rahbarinia B, Perdisci R, Li K, Antonakakis M. Measuring and detecting malware downloads in live network traffic. In: European symposium on research in computer security. Springer; 2013, p. 556–73.
[74] Bai J, Wang J, Zou G. A malware detection scheme based on mining format information. Sci World J 2014;2014.
[75] Tamersoy A, Roundy K, Chau DH. Guilt by association: large scale malware detection by mining file-relation graphs. In: Proceedings of the 20th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2014, p. 1524–33.
[76] Ahmadi M, Ulyanov D, Semenov S, Trofimov M, Giacinto G. Novel feature extraction, selection and fusion for effective malware family classification. In: Proceedings of the sixth ACM conference on data and application security and privacy. ACM; 2016, p. 183–94.
[77] Caliskan-Islam A, Harang R, Liu A, Narayanan A, Voss C, Yamaguchi F, Greenstadt R. De-anonymizing programmers via code stylometry. In: Proceedings of the 24th USENIX security symposium. 2015, p. 255–70.
[78] Chen L, Li T, Abdulhayoglu M, Ye Y. Intelligent malware detection based on file relation graphs. In: Proceedings of the 2015 IEEE 9th international conference on semantic computing. IEEE; 2015, p. 85–92.
[79] Gharacheh M, Derhami V, Hashemi S, Fard SMH. Proposing an HMM-based approach to detect metamorphic malware. In: 2015 4th Iranian joint congress on fuzzy and intelligent systems. IEEE; 2015, p. 1–5.
[80] Khodamoradi P, Fazlali M, Mardukhi F, Nosrati M. Heuristic metamorphic malware detection based on statistics of assembly instructions using classification algorithms. In: 2015 18th CSI international symposium on computer architecture and digital systems. IEEE; 2015, p. 1–6.
[81] Sexton J, Storlie C, Anderson B. Subroutine based detection of APT malware. J Comput Virol Hacking Tech 2016;12(4):225–33.
[82] Piyanuntcharatsr SSW, Adulkasem S, Chantrapornchai C. On the comparison of malware detection methods using data mining with two feature sets. Int J Secur Appl 2015;9(3):293–318.
[83] Upchurch J, Zhou X. Variant: a malware similarity testing framework. In: 2015 10th international conference on malicious and unwanted software. IEEE; 2015, p. 31–9.
[84] Jang J, Brumley D, Venkataraman S. Bitshred: feature hashing malware for scalable triage and semantic analysis. In: Proceedings of the 18th ACM conference on computer and communications security. ACM; 2011, p. 309–20.
[85] Eskandari M, Khorshidpour Z, Hashemi S. HDM-analyser: a hybrid analysis approach based on data mining techniques for malware detection. J Comput Virol Hacking Tech 2013;9(2):77–93.
[86] Graziano M, Canali D, Bilge L, Lanzi A, Shi E, Balzarotti D, van Dijk M, Bailey M, Devadas S, Liu M, et al. Needles in a haystack: Mining information from public dynamic analysis sandboxes for malware intelligence. Proceedings of the 24th USENIX security symposium. 2015. p. 1057–72.
[87] Oliva A, Torralba A. Modeling the shape of the scene: A holistic representation of the spatial envelope. Int J Comput Vis 2001;42(3):145–75.
[88] Bhodia N, Prajapati P, Di Troia F, Stamp M. Transfer learning for image-based malware classification. 2019, arXiv preprint arXiv:1903.11551.
[89] Agrawal R, Srikant R, et al. Fast algorithms for mining association rules. In: Proc. 20th Int. Conf. Very Large Data Bases, VLDB, vol. 1215. 1994. p. 487–99.
[90] Kruegel C, Kirda E, Mutz D, Robertson W, Vigna G. Polymorphic worm detection using structural information of executables. In: International workshop on recent advances in intrusion detection. Springer; 2005, p. 207–26.
[91] Ding SHH, Fung BCM, Charland P. Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: 2019 IEEE symposium on security and privacy. IEEE; 2019, p. 472–89.
[92] Ding SHH, Fung BCM, Charland P. Kam1n0: Mapreduce-based assembly clone search for reverse engineering. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2016, p. 461–70.
[93] Cordy JR, Roy CK. The NiCad clone detector. In: Program comprehension (ICPC), 2011 IEEE 19th international conference on. IEEE; 2011, p. 219–20.
[94] Baysa D, Low RM, Stamp M. Structural entropy and metamorphic malware. J Comput Virol Hacking Tech 2013;9(4):179–92.
[95] Tian R, Batten LM, Versteeg S. Function length as a tool for malware classification. In: 2008 3rd international conference on malicious and unwanted software. IEEE; 2008, p. 69–76.
[96] Ahmed F, Hameed H, Shafiq MZ, Farooq M. Using spatio-temporal information in API calls with machine learning algorithms for malware detection. In: Proceedings of the 2nd ACM workshop on security and artificial intelligence. ACM; 2009, p. 55–62.
[97] Raff E, Nicholas C. An alternative to ncd for large sequences, lempel-ziv jaccard distance. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2017, p. 1007–15.
[98] Kwon BJ, Mondal J, Jang J, Bilge L, Dumitraş T. The dropper effect: Insights into malware distribution with downloader graph analytics. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. ACM; 2015, p. 1118–29.
[99] Mao W, Cai Z, Towsley D, Guan X. Probabilistic inference on integrity for access behavior based malware detection. In: International symposium on recent advances in intrusion detection. Springer; 2015, p. 155–76.
[100] Polino M, Scorti A, Maggi F, Zanero S. Jackdaw: Towards automatic reverse engineering of large datasets of binaries. In: International conference on detection of intrusions and malware, and vulnerability assessment. Springer; 2015, p. 121–43.
[101] Huang K, Ye Y, Jiang Q. Ismcs: an intelligent instruction sequence based malware categorization system. In: 2009 3rd international conference on anti-counterfeiting, security, and identification in communication. IEEE; 2009, p. 509–12.
[102] Ye Y, Li T, Chen Y, Jiang Q. Automatic malware categorization using cluster ensemble. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. ACM; 2010, p. 95–104.
[103] Nachenberg C, Wilhelm J, Wright A, Faloutsos C. Polonium: Tera-scale graph mining for malware detection. 2010.
[104] Kalbhor A, Austin TH, Filiol E, Josse S, Stamp M. Dueling hidden Markov models for virus analysis. J Comput Virol Hacking Tech 2015;11(2):103–18.
[105] Raghavan A, Di Troia F, Stamp M. Hidden Markov models with random restarts versus boosting for malware detection. J Comput Virol Hacking Tech 2019;15(2):97–107.
[106] Annachhatre C, Austin TH, Stamp M. Hidden Markov models for malware classification. J Comput Virol Hacking Tech 2015;11(2):59–73.
[107] Russell SJ, Norvig P. Artificial intelligence: a modern approach. Malaysia: Pearson Education Limited; 2016.
[108] Quinlan JR. Induction of decision trees. Mach Learn 1986;1(1):81–106.
[109] Altman NS. An introduction to kernel and nearest-neighbor nonparametric regression. Amer Statist 1992;46(3):175–85.

[110] Boser BE, Guyon IM, Vapnik VN. A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on computational learning theory. ACM; 1992, p. 144–52.

[111] Jensen FV. An introduction to Bayesian networks, vol. 210. UCL Press London; 1996.

[112] Liu B, Ma Y, Wong CK. Improving an association rule based classifier. In: European conference on principles of data mining and knowledge discovery. Springer; 2000, p. 504–9.

[113] Cohen WW. Learning trees and rules with set-valued features. In: AAAI/IAAI, Vol. 1. 1996. p. 709–16.

[114] Hansen LK, Salamon P. Neural network ensembles. IEEE Trans Pattern Anal Mach Intell 1990;(10):993–1001.

[115] Pal M. Random forest classifier for remote sensing classification. Int J Remote Sens 2005;26(1):217–22.

[116] Vincent P, Larochelle H, Lajoie I, Bengio Y, Manzagol P-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. J Mach Learn Res 2010;11(Dec):3371–408.

[117] Ng A, et al. Sparse autoencoder. CS294A Lecture notes 2011;72(2011):1–19.

[118] Fink O, Zio E, Weidmann U. Fuzzy classification with restricted boltzman machines and echo-state networks for predicting potential railway door system failures. IEEE Trans Reliab 2015;64(3):861–8.

[119] Yan X, Cheng H, Han J, Yu PS. Mining significant graph patterns by leap search. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data. ACM; 2008, p. 433–44.

[120] Wille R. Restructuring lattice theory: an approach based on hierarchies of concepts. In: Ordered sets. Springer; 1982, p. 445–70.

[121] Brémaud P. Markov chains: Gibbs fields, Monte Carlo simulation, and queues, vol. 31. Springer Science & Business Media; 2013.

[122] Kruegel C, Robertson W, Valeur F, Vigna G. Static disassembly of obfuscated binaries. In: USENIX security symposium, vol. 13. 2004. p. 18.

[123] Cifuentes C, Gough KJ. Decompilation of binary programs. Softw - Pract Exp 1995;25(7):811–29.

[124] Cifuentes C, Van Emmerik M. UQBT: Adaptable binary translation at low cost. Computer 2000;33(3):60–6.

[125] Shin H-C, Orton MR, Collins DJ, Doran SJ, Leach MO. Stacked autoencoders for unsupervised feature learning and multiple organ detection in a pilot study using 4D patient data. IEEE Trans Pattern Anal Mach Intell 2012;35(8):1930–43.

[126] Boureau Y-l, Cun YL, et al. Sparse feature learning for deep belief networks. In: Advances in neural information processing systems. 2008, p. 1185–92.

[127] Eddy SR. Hidden Markov models. Curr Opin Struct Biol 1996;6(3):361–5.

[128] Egele M, Woo M, Chapman P, Brumley D. Blanket execution: Dynamic similarity testing for program binaries and components. In: Proceedings of the 23rd USENIX security symposium. 2014. p. 303–17.

[129] Narayanan BN, Djaneye-Boundjou O, Kebede TM. Performance analysis of machine learning and pattern recognition algorithms for malware classification. In: 2016 IEEE national aerospace and electronics conference (NAECON) and Ohio innovation summit (OIS). IEEE; 2016, p. 338–42.

[130] Kebede TM, Djaneye-Boundjou O, Narayanan BN, Ralescu A, Kapp D. Classification of malware programs using autoencoders based deep learning architecture and its application to the microsoft malware classification challenge (big 2015) dataset. In: 2017 IEEE national aerospace and electronics conference. IEEE; 2017, p. 70–5.

[131] Messay-Kebede T, Narayanan BN, Djaneye-Boundjou O. Combination of traditional and deep learning based architectures to overcome class imbalance and its application to malware classification. In: NAECON 2018-IEEE national aerospace and electronics conference. IEEE; 2018, p. 73–7.

[132] Davuluru VSP, Narayanan BN, Balster EJ. Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs. In: 2019 IEEE national aerospace and electronics conference. IEEE; 2019, p. 273–8.

[133] Pai S, Di Troia F, Visaggio CA, Austin TH, Stamp M. Clustering for malware classification. J Comput Virol Hacking Tech 2017;13(2):95–107.

[134] László T, Kiss Á. Obfuscating C++ programs via control flow flattening. Ann Univ Sci Budapest Rolando Eötvös Nominatae Sect Comput 2009;30:3–19.

[135] Bogus Control Flow. 2020. https://github.com/obfuscator-llvm/obfuscator/wiki/Bogus-Control-Flow. [Accessed 10 March 2020].

[136] Li X, Loh PK, Tan F. Mechanisms of polymorphic and metamorphic viruses. In: Intelligence and security informatics conference, 2011 European. IEEE; 2011, p. 149–54.

[137] Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. 2016, arXiv preprint arXiv:1607.02533.

[138] Bruna J, Szegedy C, Sutskever I, Goodfellow I, Zaremba W, Fergus R, Erhan D. Intriguing properties of neural networks. 2013.

[139] Papernot N, McDaniel P, Goodfellow I. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. 2016, arXiv preprint arXiv:1605.07277.

[140] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. 2014, CoRR abs/1412.6572.

[141] Papernot N, McDaniel P, Jha S, Fredrikson M, Celik ZB, Swami A. The limitations of deep learning in adversarial settings. In: Security and privacy (EuroS&P), 2016 IEEE European symposium on. IEEE; 2016, p. 372–87.

[142] Hinton G, Vinyals O, Dean J. Distilling the knowledge in a neural network. 2015, arXiv preprint arXiv:1503.02531.

[143] Ding SH, Fung BCM, Charland P. Asm2Vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization. In: Proc. of the 40th international symposium on security and privacy. San Francisco, CA: IEEE Computer Society; 2019, p. 38–55.

[144] Le Q, Mikolov T. Distributed representations of sentences and documents. In: International conference on machine learning. 2014. p. 1188–96.

[145] Mikolov T, Chen K, Corrado G, Dean J. Efficient estimation of word representations in vector space. 2013, arXiv preprint arXiv:1301.3781.

[146] Carlini N, Wagner D. Audio adversarial examples: Targeted attacks on speech-to-text. In: 2018 IEEE security and privacy workshops. IEEE; 2018, p. 1–7.

[147] Man ND, Huh E-N. A collaborative intrusion detection system framework for cloud computing. In: Proceedings of the international conference on IT convergence and security 2011. Springer; 2012, p. 91–109.

[148] Singh D, Patel D, Borisaniya B, Modi C. Collaborative ids framework for cloud. Int J Netw Secur 2016;18(4):699–709.

[149] Fung CJ, Zhu Q. FACID: A trust-based collaborative decision framework for intrusion detection networks. Ad Hoc Netw 2016;53:17–31.

[150] Mac Dermott A, Shi Q, Kifayat K. Collaborative intrusion detection in federated cloud environments. J Comput Sci Appl 2015;3(3A):10–20.

[151] Shafer G. Dempster-Shafer theory. Encycl Artif Intell 1992;1:330–1.

[152] Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In: Proceedings of the 28th USENIX security symposium). 2019. p. 729–46.

[153] Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y. Generative adversarial nets. In: Advances in neural information processing systems. 2014, p. 2672–80.

[154] Im DJ, Kim CD, Jiang H, Memisevic R. Generating images with recurrent adversarial networks. 2016, arXiv preprint arXiv:1602.05110.

[155] Jang E, Gu S, Poole B. Categorical reparameterization with gumbel-softmax. 2016, arXiv preprint arXiv:1611.01144.