Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Review:

A survey of malware behavior description and analysis*

Bo YU‡, Ying FANG, Qiang YANG, Yong TANG, Liu LIU

College of Computer, National University of Defense Technology, Changsha 410073, China

E-mail: yubo0615@nudt.edu.cn; fangying15@nudt.edu.cn; q.yang@nudt.edu.cn; ytang@nudt.edu.cn; hotmailliuliu@163.com

Received Nov. 26, 2016; Revision accepted Feb. 21, 2017; Crosschecked May 8, 2018

Abstract: Behavior-based malware analysis is an important technique for automatically analyzing and detecting malware, and it has received considerable attention from both academic and industrial communities. By considering how malware behaves, we can tackle the malware obfuscation problem, which cannot be processed by traditional static analysis approaches, and we can also derive the as-built behavior specifications and cover the entire behavior space of the malware samples. Although there have been several works focusing on malware behavior analysis, such research is far from mature, and no overviews have been put forward to date to investigate current developments and challenges. In this paper, we conduct a survey on malware behavior description and analysis considering three aspects: malware behavior description, behavior analysis methods, and visualization techniques. First, existing behavior data types and emerging techniques for malware behavior description are explored, especially the goals, principles, characteristics, and classifications of behavior analysis techniques proposed in the existing approaches. Second, the inadequacies and challenges in malware behavior analysis are summarized from different perspectives. Finally, several possible directions are discussed for future research.

Key words: Malware behavior; Static analysis; Dynamic Analysis; Behavior data expression; Behavior analysis; Machine learning; Semantics-based analysis; Behavior visualization; Malware evolution

https://doi.org/10.1631/FITEE.1601745 **CLC number:** TP309.5

1 Introduction

The quantity and complexity of malware samples have increased considerably over the past few years. Recent malware samples appear to be highly modular and less functionally typical. This development has been further fueled by the introduction of malware generation tools and the reuse of different malware modules. The situation has become more serious with the expansion of open source technology. In response, there is an urgent need to facilitate new malware analysis techniques to automatically identify and characterize malware variants.

Malware analysis techniques can be generally classified into two categories: static and dynamic. Static approaches focus on binary file information and disassembly codes from a malware sample, but lack a sample execution. Thus, this approach is usually regarded as a lightweight method for malware classification. In contrast, dynamic approaches extract behavioral data by executing the sample in a virtual environment, and then analyze the malware behavior based on logged behavior data. By providing an intuitive understanding of the malware behavior, dynamic approaches help analysts understand the intentions behind the behavior and analyze trends in behavioral evolution.

Over the long term, academia has paid attention to malware analysis techniques for improving detection accuracy and efficiency. Recently much research has been devoted to dynamic analysis, outperforming static analysis methods by neutralizing the effects of obfuscation and morphing techniques (Damodaran et al., 2017). Most dynamic analysis techniques rely on system call traces to analyze the malicious behaviors of malware samples.

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (No. 61472437)

ORCID: Bo YU, http://orcid.org/0000-0001-6576-5555

The understanding of malware behavior analysis in this paper is twofold: behavior analysis using dynamic behavior data and behavior analysis using static behavior data. We have taken this approach because both forms of behavioral data are the basis of behavior analysis, and they can be used to understand malware behaviors and for behavior-based malware detection. The existing research demonstrates that a combination of features from both dynamic and static analysis can yield the best accuracy in behavior-based malware analysis (Anderson et al., 2012; Alazab, 2015).

Malware behavior analysis aims to answer three questions: (1) How can malware behavior data be described in a general way? (2) How can malware samples be detected and classified by leveraging behavior data? (3) What kinds of malicious behavior can malware samples really carry out? Thus, the main contributions of related works typically include behavior data extraction and expression, and behavior-based malware identification and analysis. Additionally, we would argue that behavior visualization is also an important part of behavior analysis because it is useful in assisting the behavior analysis process and understanding the behavior analysis results.

Much research has been carried out within the scope of malware behavior analysis. Several virtual environments have been designed for collecting behavior data, and many approaches have been proposed for in-depth malware behavior analysis. However, a global view of the related research is rare. Egele et al. (2012) explored several dynamic malware analysis techniques and tools. They focused on malware types, propagation modes, and collection methods for behavior data, which are not of primary importance in our work. Bayer et al. (2014) discussed only the classification of system behaviors on the Anubis platform. In contrast, we investigate an even broader scope of behavior data types.

Another similar work (Razak et al., 2016) was aimed to analyze the research trends in malware analysis with a bibliometric method. To uncover the global trends and frontiers in malware publications, research articles are retrieved from Web of Science and analyzed using the following criteria: impact journals, highly cited articles, research areas, productivity, keyword frequency, institutions, and authors. Although Razak et al. (2016) presented an overview of malware research trends, it does not cover topics such as classification of behavior data and behavior analysis methods, which are key topics in our work.

Additionally, studies on behavior analysis of Android malware fall in the scope of our discussions in this survey, because analysis goals, analysis methods, and behavior data types of existing studies on Windows and Android platforms are the same.

2 Malware behavior analysis techniques

As discussed above, behavior data forms the basis of analyzing malware behavior, and a well-designed behavior expression is useful for improving the efficiency and effectiveness of behavior analysis. Thus, in this section, we discuss behavior data types and behavior expression first, and then investigate typical behavior analysis methods in recent works.

2.1 Malware behavior expression

2.1.1 Behavior data types

During the behavior data extraction phase, behavior data is collected in a static approach or dynamic approach and then formalized for storage and subsequent analysis. The behavior data extracted through a static approach includes function call name, file structure information, import tables, strings, control flows, and so on. IDA Pro is a common tool to assist in behavior data extraction from the disassembly code of malware binaries. For example, Shi et al. (2014) extracted dynamic link library information to construct the feature space for malware samples.

However, in a dynamic analysis approach, a virtual environment is needed to execute malware samples and collect program traces. Sandboxes are typical systems used for this purpose, including the Cuckoo sandbox (Cuckoo, 2017), Anubis (Bayer et al., 2006), and Ether (Dinaburg et al., 2008). These monitoring systems have been discussed carefully in the literature (Egele et al., 2012; Kruegel, 2014; Bauman et al., 2015). To design a good hypervisor-based monitoring system, Bauman et al. (2015) conducted a survey that focuses on the practicality, flexibility, coverage, and automation of existing virtual monitors, and discussed different approaches to tackle the semantic problems of observed behavior

data. In contrast, our survey discusses behavior data types and popular techniques and tools for extracting behavior data.

The common types of behavior data generated in a dynamic approach include system call names, together with arguments, return values, and environment variables in context. In particular, data flow, system call graphs, and system states are gathered. dAnubis (Neugschwandtner et al., 2010) is a QEMU-based system designed to monitor malicious behavior in the system device driver. The behavior data gathered by dAnubis includes kernel function calls, system call table hooks, and device communications data.

Kernel data and system states are also considered behavioral data in malware behavior analysis. For example, Bailey et al. (2007) defined the behavioral fingerprint of malware samples in terms of non-transient state changes that the samples impose on an operating system. With the extracted system state data, the malware profiles of individual samples are defined and malware samples are grouped according to the differences between any two profiles. Lanzi et al. (2009) used kernel data to monitor kernel behavior on sensitive data manipulations.

In addition to traditional static analysis and dynamic monitoring systems, program analysis approaches have been used in recent works to assist in malware behavior analysis, with the aim of automatically generating full control flow and data flow information. Typical program analysis techniques include tainted analysis techniques (Moser et al., 2007; Fratantonio et al., 2016), value set analysis techniques (VSA) (Leder et al., 2009), and symbolic execution techniques (Brumley et al., 2008; Yuan et al., 2014).

Symbolic execution techniques are used to capture full function call sequences, while tainted analysis techniques combined with a full system emulation approach obtain the complete data flow. For example, Leder et al. (2009) leveraged VSA techniques to extract sets of characteristic values for detecting and classifying metamorphic malware. In a similar work, Brumley et al. (2008) employed symbolic execution to explore full paths and find trigger-based behavior. The results of the work show that the system can capture the full range of malware behaviors and identify all actions along the different paths. Moser et al. (2007) presented a system based on a tainted analysis technique to explore multiple execution paths in Windows executables, the goal of which was to obtain a more comprehensive overview of the actions that an unknown sample could perform. Additionally, the system automatically provided the information for conditions under which a malicious action would be triggered.

2.1.2 Malware behavior description

Behavior description encompasses behavioral data at different levels and forms the basis for malware behavior analysis. Malware-behavior description methods include XML-based formats, semantic description methods (e.g., ontology-based), several well-defined description languages, and several concepts based on formal description and information-sharing specifications for threat intelligence.

The XML-based format, which is a common type to express structural information, can also be used to express program traces such as function call traces. However, the different levels of XML are inadequate when it comes to expressing behavior data. For example, the semantic behavioral data, such as semantics-related system call paths and semantics-related control flows, cannot be expressed well in XML. As a result, several abstract description languages have been proposed for this purpose.

To bridge the semantic gap in malware behaviors, Huang et al. (2011) leveraged fuzzy ontology (FO) and fuzzy markup language (FML) to assist in a semantic understanding of malware behaviors, and presented a semantic methodology to develop a knowledge model related to malware behaviors and design an intelligent system for behavior identification.

The malware instruction set (MIST) (Trinius et al., 2011) is a new representation of system calls with input and output arguments. This representation is optimized for effective and efficient behavior analysis with machine learning techniques, and can also be used as a meta language to unify behavior reports.

Malware attribute enumeration and characterization (MEAC) (Kirillov et al., 2011) is another behavior representation approach that attempts to develop a legally defensible definition of malware, and it can be used to define the groups of behaviors and attributes that have the potential to be malicious based on executions of the malware.

Abstract malicious behavioral language (AMBL)

(Deschamps, 2008; Jacob et al., 2009) is a specification language that provides a platform- and language-agnostic framework for the detection of malware samples. AMBL is well formed, thus guaranteeing a possible order for semantic attribute valuation. The behavioral signatures declared in AMBL make it easy to build efficient and resilient parsing automata for malware detection.

Malware analysis intermediate language (MAIL), a language presented by Alam et al. (2015), can be used to express an annotated control flow graph of malware samples. With the support of MAIL, malware detection is achieved by subgraph and pattern matching on annotated control flow graphs.

The behavior specification unit (BSU) (Pleszkoch and Linger, 2015) is an abstraction language proposed by the Oak Ridge National Laboratory, which offers a higher level of program behavior so that non-practitioners can easily understand it. BSUs are general and implementation-independent specifications that can be applied to malware analysis.

The behavior specification language (BSL) (Martignoni et al., 2008) is a language that consists of a set of primitives that can be used to define additional behaviors. In creating compositions of some basic primitives, BSLs can specify and then identify novel semantically meaningful behaviors.

The signature specification language (SSL) (Feng et al., 2014) is a language presented to define the semantic properties of complex behavior data such as control flow and data flow. SSL consists of three kinds of predicates to express component type, control-flow relationship, and data-flow relationship separately, and it can provide a new form to construct a high-level representation of malware behavior signatures.

Besides these languages, some new concepts have been proposed to express malware behavior, including the behavior profile (Bayer et al., 2009), behavior patterns (Beaucamps et al., 2010), and common behavior template (Shan and Wang, 2014). The behavior profile concept (Bayer et al., 2009) can accurately describe fine-grained system call events. The operation objects of a system call can vary significantly even when the samples exhibit the same behavior. A behavior profile includes not only a set of actions (such as read, write, and create), but also operation objects, as well as the type of operations and the dependences. To formally describe malware behavior, Beaucamps et al. (2010, 2012) presented a malware analysis approach based on first-order linear temporal logic (FOLTL). The behavior pattern concept is defined as a regular language that describes high-level properties or a relevant behavior sequence. In this approach, the defined term, algebras, consists of Trace, Action, and Data extracted from program traces, and a program behavior is defined by a set of traces that satisfies a closed FOLTL formula. For the common behavior concept, the template (Shan and Wang, 2014) is formed by a set of discrete behaviors that enable the behavior matching process to occur more quickly and also the storage space to be smaller and fixed. This approach is accurate since it identifies a malware based on combined behaviors.

These proposed description languages and concepts provide a rich expression of behavior data, and facilitate the design of behavioral analysis methods. Some information-sharing specifications have also been proposed for cybersecurity situational awareness, real-time network defense, and sophisticated threat analysis. For example, TAXII (Haass et al., 2015), STIX (Barnum, 2012), and CybOX (Kokkonen et al., 2016) are community-driven technical specifications designed to enable automated information sharing and they can help in standardizing threat information.

2.2 Behavior analysis method

According to the emphasis of different analysis processes, we can classify existing behavior analysis methods into two classes, machine-learning-based analysis approaches and semantics-based behavior analysis approaches. The former focuses on feature extraction and automatic learning, and is a frequently used technique to detect or classify malware samples based on malware behavior. In contrast, the latter focuses on identifying malware behavior through a semantic understanding of the behavior data, and provides the ability to determine the capabilities (also called ‘malicious functionality’) of malware samples. Although syntactic-based analysis approaches are common in malware analysis (Feng et al., 2017), they are rarely used for behavior-based malware analysis.

Compared to machine learning based analysis approaches, semantics-based analysis approaches benefit from several advantages, and a prominent one is that the analysis process and result are easy to

interpret. However, semantic rules must be defined manually by a trained security analyst. In the next few paragraphs, the typical process and categories of both approaches are discussed in detail.

2.2.1 Machine learning based behavior analysis

Given the massive quantities of malware samples, approaches based on machine learning play an important role in automatic behavior analysis. By extracting behavior features from the behavior data, machine learning based approaches can learn feature models automatically. Existing research results show that machine learning approaches can perform well in detecting malware accurately and in a lightweight manner (Rieck et al., 2008; Bayer et al., 2009). An outline of machine learning based behavior analysis approaches is discussed in the following basic steps:

1. Feature extraction. Features consist of behavior data with a high-dimensional vector space. Through feature extraction, behavior data is transformed into digital values using techniques such as *N*-grams.

Additionally, feature selection is necessary for acquiring excellent feature sets for model training and reducing the dimension of the original feature set. Some typical statistical analysis methods have been used for feature selection, such as principal component analysis (PCA) (Mithal et al., 2016), information gain (IG) (Moonsamy et al., 2012), and the chi-square test (CHI) (Belaoued and Mazouzi, 2015).

2. Model construction and training. Based on an extracted feature set, analysts construct behavior models with machine learning algorithms and train the model with training data. Support vector machines (SVMs) and decision trees (DTs) have been widely used to set up behavior models based on the behavior features.

3. Evaluation and comparison. Test data is used to examine and analyze the effectiveness of the trained model. Comparing test results, analysts can optimize the model by adjusting related arguments.

In the following part, common types of behavior feature and feature extraction approaches in the literature are discussed in detail.

System call features are commonly used to express the behavior characteristics of malware samples. For example, Kirat and Vigna (2015) presented MalGene, a bioinformatics-inspired system that extracts system call traces and uses inverse document frequency (IDF) to filter out the common execution events. Rieck et al. (2011) proposed a behavior analysis system in which system call sequences are extracted and represented with MIST instructions. Then an *N*-gram method is used to generate a feature set based on the collected system call data. Naval et al. (2015) extracted system call traces by monitoring malware execution and transforming the traces into ordered system-call graphs (OSCGs). In addition, in our previous studies (Wang et al., 2015; Liu et al., 2016), system call frequencies and import functions were used as features to cluster malware samples into groups, and then a shared nearest neighbor (SNN) method was leveraged to compute the distance of two samples.

The second kind of behavior feature is a control flow feature, where a graph containing various execution paths can be taken from the malware samples. Recently, control flow graphs with different representations have been used.

Zhao et al. (2014) extracted the opcode sequences of each block according to the control flow structure of malware samples, and then computed with a hash function to form the feature space. Then, the features were selected with IDF and used to find classification rules between malicious and benign samples.

Ding et al. (2014) translated the control flow graph of malware samples into an execution tree to obtain execution paths. They concatenated all possible paths to form an opcode stream and used an *N*-gram method to extract behavior features. In addition, IG and DF were used to select the behavior features.

The feature space in Cesare et al. (2014) is a decomposition of a set of control flow graphs into either fixed-size *k*-subgraphs or *N*-gram strings. The feature selection approach counts the number of times each feature occurs, and the top 500 are reserved as the selected features.

The function call feature is the third type of behavior feature, extracted using static analysis techniques and often regarded as an effective feature in malware behavior analysis.

Cen et al. (2015) proposed an approach which extracts function call information from the disas-

sembly code of malware samples. Using statistical techniques, the frequencies of API calls are counted, and then truncated into either 0 or 1 with a pre-defined threshold to simplify the feature representation. Finally, to reduce the dimensions of the feature space, IG and CHI techniques are used for feature selection.

Zhang et al. (2014) proposed a malware classification system called DroidSIFT. The system includes graph databases from function call APIs and produces graph-based feature vectors, which bear a non-zero similarity score in one element only if the corresponding graph is the best match with one of the graphs.

Ding et al. (2013) extracted function call APIs from the PE file and selected APIs according to the following criteria: (1) choose the APIs that have high distribution values; (2) choose the APIs that have a strong ability for classification. Finally, features are reduced via an objective-oriented association mining algorithm that can mine the strong discrimination power association rules.

A data flow feature is also used for behavior features in existing approaches. Based on data flow information, Wüchner et al. (2015) proposed a new graph called the 'quantitative data flow graph (QDFG)'. QDFGs are incrementally built on relevant data flow system events. The features are computed by mapping a QDFG node to a real number and calculating the number through statistic methods.

Several works use hybrid features, which contain many different types of feature behavior data, to enhance the expression ability of behavior features.

Yerima et al. (2015) considered two kinds of features, critical API calls and the applied application permissions. Shan and Wang (2014) clustered malware samples using features defined on atomic behaviors and correlated suspicious objects. Watson et al. (2016) proposed an online cloud anomaly detection approach. In this approach, the first feature is the system-level data including memory usage, peak memory usage, and the number of threads and handles. The second is network-level data including packets, bytes, and flows per address pair. For each feature, the authors use mean, variance, and standard deviation approaches to build statistical meta-features. Mohaisen and Alrawi (2015) proposed a behavior-based automatic approach in which the features consist of behaviors in three groups: file system, registry, and network activities.

While the features are extracted and selected, machine learning models and algorithms for behavior analysis need to be built. Typically, there are two categories for the machine learning model, i.e., supervised learning for malware detection and classification with a labeled malware sample set, and unsupervised learning for malware clustering without labels.

Supervised learning is often used for malware classification. This kind of model is built upon features and labels for malware samples. SVM (Watson et al., 2016), naive Bayes (Zhang et al., 2016), and neural networks (Dahl et al., 2013) are general algorithms of supervised learning. In contrast, unsupervised learning models group behavior features based on the similarity of each feature vector. *K*-nearest neighbor (KNN) (Ding et al., 2014) and locality sensitive hashing based clustering (Bayer et al., 2009) are examples.

For the last step in machine learning approaches, some evaluation and comparison metrics are used to validate the effectiveness of behavior features and the machine learning model. The common evaluation indices include true positive (TP), false positive (FP), true negative (TN), and false negative (FN). TP is the number of malware (benign) samples that are classified as malware. TN is the number of benign (malware) samples that are classified as benign. Accuracy and *F*-measure (Wüchner et al., 2015) are two indices used to evaluate the performance of the proposed analysis approaches. ROC curve, a two-dimensional graph, is a comprehensive approach to express the relationships between the TP and FP indices. It depicts relative tradeoffs between TP and FP (Elhadi et al., 2014). The area under the ROC curve can be calculated as a single value between 0 and 1.0. The closer the value to 1.0, the higher the TP rate, and the lower the FP rate.

The classification of machine learning based behavior analysis approaches in existing works will be discussed in Section 3.1.

2.2.2 Semantics-based behavior identification

Semantics-based behavior analysis has recently attracted more attention, and it plays an important role in behavior-based malware detection. The goal of semantics-based behavior analysis is to identify

malicious behavior based on a semantic understanding of the extracted behavior data. To achieve that goal, some works focus on defining semantic rules and behavioral knowledge, which are summarized from the manual analysis process. In addition, there are some works that focus on identifying what kinds of malicious behavior a malware sample has.

According to various behavior data types, several kinds of semantic rules have already been defined, such as system call grouping rules (Das et al., 2016), system call association rules (Naval et al., 2015), and malicious behavior decision rules (Das et al., 2016). Semantically, a decision rule for self-extraction behavior can be explained as actions that read the content of an owned file and write it to another executable file which is regarded as malicious. However, the purposes of these semantic rules are various. The semantic rules on behavior data aim to eliminate the semantic difference in behavior data with the same meaning but different function names, or arguments. Moreover, the semantic rules for complex behavior aim to provide a global view of behavior capabilities for a given malware sample.

To understand malware behavior, Alazab et al. (2010) and Alazab (2015) presented an automatic method to understand the malicious purpose of malware samples. By mapping the API from the MSDN library to meaningful behaviors, such as search files, delete files, and change files, the malicious behavior can be identified based on a statistical analysis of function call sequences.

Comparetti et al. (2010) presented a system to determine the malicious functionality of malware samples. By extracting genotype models from dynamic program traces, new malware samples can be identified with a well-designed genotype-matching algorithm.

Jacob et al. (2009) proposed an attribute-grammar-based behavior analysis model. The work is a part of the worldwide observatory of malicious behaviors and attack threats (WOMBAT) research project, and is supported by the seventh framework programme of the European Community. In that model, a detection layer is designed to define association rules between behavior attributes and works as a behavior automata to infer behavior capabilities over behavior data, and a profiling layer to profile the behavior capabilities of each malware family.

Thomson et al. (2015) leveraged the ACT-R tool with cognitively inspired inference mechanisms to identify high-level behavior capabilities of a malware sample. Behavior attributes for each malware sample are identified and some association rules are defined to infer the high-level behavior capabilities of each malware family. The results showed that in total 30 behavior capabilities were identified. Two kinds of ACT-R cognitive models, instance-based (Lebiere et al., 2015) and rule-based (Nunes et al., 2015), are adopted to generate the probability distribution over a set of malware families, and to infer a set of likely high-level behavior capabilities based upon that distribution.

Huang HD et al. (2011, 2014) also presented an approach to assist semantic understanding of malware behavior based on semantic technologies and computational intelligence methods. In their approach, fuzzy ontology and fuzzy markup language (FML) are integrated to bridge the semantic gap between behavior data and malware behavior.

Many studies have been conducted to identify malware behaviors on Windows and Android platforms. Several kinds of malware behaviors have been studied recently. The typical malicious behaviors include spyware-like behaviors (Kirda et al., 2006), rootkit behaviors (Yin et al., 2008), evasive behaviors (Sun et al., 2011; Kirat et al., 2014; Kirat and Vigna, 2015; Zhang et al., 2015), environment-sensitive behaviors (also called 'trigger-based behaviors') (Martignoni et al., 2009; Lindorfer et al., 2011), and network scan behaviors (Inoue et al., 2009). These malware behaviors have good interpretability when analyzing and detecting malware samples.

To identify spyware-like behaviors, Kirda et al. (2006) proposed a spyware detection system based on COM browser functions and Windows API calls from both dynamic analysis and static analysis. Some suspicious API calls about user privacy and resource consumption were derived and used to characterize spyware-like behaviors.

Naval et al. (2015) adopted the asymptotic equipartition property (AEP) for program semantic analysis to extract semantically relevant paths, providing the ability to semantically understand system call sequences. By constructing an ordered system-call graph (OSCG) and its transition probability matrix (TPM) from a program execution trace,

the work can identify the suspicious behavior of malware binaries. The results showed that the proposed detection model can obtain high detection accuracy and is less vulnerable to call-injection attacks.

UNVEIL (Kharraz et al., 2016) is an automatic system built on top of the Cuckoo sandbox to detect ransomware. A monitoring driver is designed in UNVEIL to obtain file system I/O activities from existing ransomware families, and three main I/O access patterns are identified. With these malicious behavior patterns, suspicious file system activities can be detected. The experimental results show that UNVEIL has performed better than existing AV scanners, and can also be used to detect zero-day ransomware.

Poeplau et al. (2014) proposed a static analysis approach to automatically detect dynamic code loading behavior in Android applications. Five kinds of code loading techniques, including class loader, package context, native code, runtime execution, and APK installation, are carefully analyzed, and the common method invocations in these techniques are identified. Some heuristics are implemented to look for invocations of methods that are associated with the respective techniques. Finally, the suspicious method invocation, together with its parameters, is used to conclude whether an application will load external code.

MALT (Zhang et al., 2015) is a bare-metal debugging system that employs a system management mode to transparently analyze armored malware. As a hardware-assisted debugging system, MALT is immune to hypervisor attacks and can analyze and debug hypervisor-based rootkits and OS kernels. The experimental results demonstrate that MALT remains transparent in the presence of all tested packers, anti-debugging, anti-virtualization, and anti-emulation techniques.

MineSweeper (Brumley et al., 2008) is an automatic analysis system proposed to identify trigger-based behaviors. By leveraging mixed concrete and symbolic execution to automatically and iteratively explore different code paths, MineSweeper can detect the existence of trigger-based behavior and find the conditions that trigger such hidden behaviors.

PolyUnpack (Royal et al., 2006) is another system whose aim is to extract the hidden code of unpack malware. The unpack-executing behaviors are formally defined, and an algorithm is also presented to identify and extract its hidden code by monitoring changes in the malware binary during its execution.

TriggerScope (Fratantonio et al., 2016) is a system based on program analysis techniques to detect malicious application logic executed or triggered under certain circumstances. A trigger analysis technique is proposed to automatically identify the logic bomb triggers in Android applications. First, a symbolic execution technique is used in TriggerScope to recover a CFG annotated with block predicates and abstract program states at all program points. Second, full path predicates are recovered and checked for whether they represent potential triggers for malicious behavior. Finally, suspicious trigger conditions for logic bombs that guard potentially sensitive functionality are identified.

Martignoni et al. (2008) proposed a layered architecture for detecting malicious behaviors. The architecture uses hierarchical behavior graphs to infer high-level behaviors from the composition of low-level system calls. With the embedded data-flow analysis technique, the meaningful malicious behaviors, and especially with behaviors such as evasive malware behavior, the alternative sequences of events that achieve the same goal are also detected. The experimental results show that the architecture can thoroughly identify high-level behaviors. In Martignoni et al. (2009), a cloud-based framework was presented for analyzing trigger-based malicious behavior. By monitoring Windows system calls and their outputs in multiple execution environments, their framework can reveal all the possible trigger conditions of malicious programs.

Targetdroid (Suarez-Tangil et al., 2014) is a new system which can identify targeted malware and trigger malicious behaviors. In the system, a set of program activities is used to characterize malware behavior. In particular, a behavior-triggering stochastic model is developed based on Markov chains to express control flow. The typical triggering conditions, including user presence, location, time, and hardware, are carefully considered in the system.

To reveal malicious behaviors that are typically exhibited during malware execution, Christodorescu and his team presented several methods (Christodorescu et al., 2008; Fredrikson et al., 2010) to automatically mine specifications of malicious

behavior. For example, HOLMES (Fredrikson et al., 2010) is a behavior-based malware detector, which defines a new form of discriminative specification for describing the unique properties of malware samples, and combines graph mining with synthesized discriminative specification. A behavior mining technique is introduced in HOLMES to find a set of behaviors that maximizes the quantity over the given malware samples, by analyzing the dependence graphs from a positive subset and a negative set and identifying the subgraphs that are most useful in uniquely characterizing the programs in the positive subset.

Rootkit behavior is a typical malicious behavior (Wang et al., 2008; Riley et al., 2009). By altering the legitimate kernel behavior of an operating system, a kernel rootkit provides the capability of hiding malicious activities for user-level malware programs. Generally, it is difficult to discover rootkit behaviors with traditional monitoring systems. Several works focus on identifying the hooking behavior of kernel-level rootkits.

K-Tracer (Lanzi et al., 2009) is a kernel rootkit analysis system built on top of the whole-system emulator QEMU. To obtain the behavior data for rootkits, K-Tracer contains a trace extractor engine to trace the execution of event handling code and gather machine-level instructions and memory access in the Windows kernel. Furthermore, an offline slicing engine is designed to analyze data flow, in which a dynamic slicing technique is used to identify malicious data manipulation activities.

SigGENE (Shosha et al., 2012) is another rootkit analysis system which monitors the kernel objects, profiles malicious kernel objects in the malware sample, and determines invariant kernel-object feature values during malware execution. SigGENE gathers behavior data by monitoring kernel function calls and kernel data objects in the virtual machine monitor (VMM). To generate an evasion-resistant malware signature, the system uses kernel object profiles developed during dynamic monitoring and introspection of malware execution and determines invariant values over the kernel object's features.

To reveal malicious network activities, Inoue et al. (2009) proposed a malware analysis system which contains a black hole sandbox using coordinated movements of the dummy DNS and a packet filter to observe the unadulterated scan activity.

Special attention is paid to security-sensitive behaviors in Android applications. Beaucamps et al. (2010, 2012) proposed a malware analysis system, which uses a model checking technique to identify high-level behaviors, such as information leak behaviors. In the system, a program behavior is formally defined as an infinite subset of a sequence of library calls, and a set of behavior patterns with semantic understanding are expressed using first-order linear temporal logic (FOLTL) formulas. AppContext (Yang et al., 2015) is another analysis system that aims to differentiate malicious and benign mobile applications behaviors. AppContext extracts the contexts of security-sensitive behaviors and performs static analysis to locate the security-sensitive behaviors. Apposcopy (Feng et al., 2014) and ASTROID (Feng et al., 2017) are two semantics-based approaches for identifying a prevalent class of Android malware that steals private user information. With a new form of program representation called the 'inter-component call graph (ICCG)' and malware signatures that describe semantic characteristics of malware families, Apposcopy can efficiently detect Android applications that have certain control- and data-flow properties.

2.2.3 Hybrid approach

Specifically, hybrid approaches based on both machine learning and semantics are designed to enhance detection capabilities. By performing a semantic analysis on the behavior data first, a corresponding feature extraction method can be designed with a better understanding of the behavior data.

Cao et al. (2013), together with the derivative work by Miao et al. (2016), presented a behavior abstraction method based on function calls. The function name, string type arguments, and pre-defined system constants in the Windows system were all abstracted with fine-grained abstraction rules. Terms that have the same meaning or functionality were grouped into one abstract description. Based on these rules, 443 minimal security-relevant behaviors were abstracted. Then, the abstracted minimal security-relevant behaviors were used to construct a feature space. Compared with the existing function call API 2-gram approach, the experimental results showed that the proposed method can achieve better classifi-

cation accuracy and AUC in most machine learning algorithms.

CrowdSoucre (Saxe et al., 2014), a system proposed by Invincea Labs, is a statistical natural language processing system for inferring behavior functionality of malware samples. CrowdSource learns mapping rules between low-level APIs and high-level software functionality by leveraging millions of online technical documents. An inference module is proposed in the CrowdSource system to extract function call information and compute the probability of behavior capabilities. The research results showed that at least 14 high-level malware capabilities can be identified in unpacked malware samples. Moreover, CrowdSource includes a Bayesian network model to compute the final probabilities of high-level capabilities for given malware samples.

Lee et al. (2015) presented a malware detection approach based on the *N*-gram features of system calls. The system call names, together with the parameter type, critical value, and parameter code, are all considered behavior features. An algorithm is also proposed for automatically grouping malware samples with similar behavior data. They used the local clustering coefficient to analyze the closeness of a malware sample to a malware family.

GuardOL (Das et al., 2016) is a hardware-enhanced architecture, whose aim is to capture the high-level semantics of malicious behaviors. GuardOL defines four types of semantic rules on system calls and uses a frequency-centric model to construct features using system call patterns of known malware and benign samples. In comparison with other similar work, GuardOL can learn a frequency-centralized model (FCM) on malware semantics behavior, and has a lower false positive rate.

To derive the behavior that a family of malware samples has in common, Park et al. (2013) proposed a graph clustering method to extract a subgraph, which stands for the common behavior of the malware samples. In this method, a graph is developed to represent kernel objects and their attributes for each malware sample.

DREBIN (Arp et al., 2014) is a lightweight method for detection of Android malware that enables identifying malicious applications. By performing a static analysis and gathering the broad features of Android applications, typical behavior patterns can be automatically identified and used to explain the analysis results of DREBIN. The experimental results showed that DREBIN enables efficiently scanning large amounts of applications and that it can be applied directly on mobile phones to protect users from installing applications from untrusted sources.

DroidMat (Wu et al., 2012) is a static feature-based method for detecting Android malware. DroidMat extracts information from the manifest file and API calls, and applies a K-means algorithm that enhances the capability of recognizing different intentions. Finally, DroidMat uses a KNN algorithm to classify the applications as benign or malicious.

DroidMiner (Yang et al., 2014) is another static analysis system, which automatically mines malicious behavior patterns from known Android malware and seeks out the behavior patterns in other unknown Android applications. Based on behavior graph generation and machine learning techniques, DroidMiner can automatically discover and extract malware behavior patterns for malware detection and classification.

Chuang and Wang (2015) proposed a malware detection system based on static analysis and machine learning techniques. In the system, a hybrid-model classifier, combining the normal behavior model and malicious behavior model, is built to improve detection accuracy. The experimental results showed that the proposed hybrid-model classifier can label 79.4% applications with a false positive of zero in the labeling process.

2.3 Visualization techniques of behavior analysis

Visualization techniques have been introduced by several works to support intuitive understanding of type, quantity, and the relationships of malware behavior data. Based on visualization techniques, the structural characteristics of behavior data are easier to discover and investigate. Existing visualization techniques, such as the similarity map, sequence graph, and tree-map, are used to explore behavior relationships between malware samples, aiding malware analysis in a visual way.

Josh et al. from Invincea Labs proposed two visualization methods within an interactive display that present information as a graphical result (Saxe et al., 2012). The first viewing method is a map-like visualization of similarity among malware samples

based on lower dimensional projection of a similarity matrix. The second provides insight into similarities and differences between samples in terms of system call sequences. The proposed visualization approaches allow users to understand the overall structural similarity of a malware family, and inspect how behavioral traits are distributed over the family.

Yavvari et al. (2012) presented a behavioral map represented as a bitmap of projections. These projections can be viewed as shared traits and are represented by rows and arranged according to clusters. Each row shows the shared behavior data of one malware sample with respect to the reference. An additional column on the map also visualizes how much of the behavior data for a malware sample is shared with the reference.

Grégio et al. (2012) presented a visualization framework to aid security analysts in observing malware behavior data. Two interactive visualization tools, behavioral spiral, and malicious timeline, are developed to express malicious chains of behavior events and to spot interesting actions. The goal of the behavioral spiral tool is to represent temporal actions. The spiral representation is useful in illustrating the big picture covering sample behavior and allowing quick visual comparisons between behavior data when there are various malware samples.

Trinius et al. (2009) used a visualization technique to enhance understanding of malware behavior. They used tree maps and thread graphs to display the actions of malware samples and to help analysts identify malicious behavior.

3 Discussion and suggestions for behavior analysis

3.1 Analysis and discussion

In this section, we will discuss the categorization of related works from several perspectives: (1) the goal of the work, (2) what kind of behavior data are obtained, and (3) which analysis techniques are used for a specific behavior-analysis goal.

3.1.1 Classification of analysis goals

The goals of existing behavior analysis approaches fall into three general categories: (1) malware detection, which provides the ability to discriminate malware from benign samples, (2) malware classification, which can determine to which classes the given samples belong, and (3) malware evolution, revealing the ancestor-descendant relationship of malware samples within a family. Table 1 shows the categories of analysis goals within related works.

Current research in the field focuses on both malware detection and classification, and both efforts try to group malware samples according to similar behavior data. Based on the number of different classes in the sample set, detection and classification techniques are chosen according to actual needs. Generally speaking, malware classification is a multi-classification issue, and malware detection can be considered a two-class classification issue. Moreover, analysis methods, behavior data, and evaluation indicators in the two techniques are nearly identical.

Table 1 Comparison of different analysis goals

Analysis goal	Related works
Detection	Kirda et al., 2006; Brumley et al., 2008; Martignoni et al., 2008, 2009; Rieck et al., 2008; Wang et al., 2008; Yin et al., 2008; Inoue et al., 2009; Jacob et al., 2009; Lanzi et al., 2009; Riley et al., 2009; Alazab et al., 2010; Beaucamps et al., 2010; Comparetti et al., 2010; Fredrikson et al., 2010; Huang HD et al., 2011, 2014; Lindorfer et al., 2011; Sun et al., 2011; Babić et al., 2012; Beaucamps et al., 2012; Shosha et al., 2012; Wu et al., 2012; Bos, 2013; Cao et al., 2013; Palahan et al., 2013; Park et al., 2013; Arp et al., 2014; Ding et al., 2014; Feng et al., 2014; Kirat et al., 2014; Poeplau et al., 2014; Saxe et al., 2014; Shan and Wang, 2014; Shi et al., 2014; Sirinda, 2014; Suarez-Tangil et al., 2014; Yang C et al., 2014; Alazab, 2015; Cen et al., 2015; Kirat and Vigna, 2015; Lebiere et al., 2015; Naval et al., 2015; Nunes et al., 2015; Thomson et al., 2015; Wüchner et al., 2015; Yang W et al., 2015; Yerima et al., 2015; Das et al., 2016; Fratantonio et al., 2016; Galal et al., 2016; Kharraz et al., 2016; Miao et al., 2016; Watson et al., 2016; Feng et al., 2017
Classification	Rieck et al., 2008; Bayer et al., 2009; Cao et al., 2013; Dahl et al., 2013; Ding et al., 2013; Cesare et al., 2014; Shi et al., 2014; Yang C et al., 2014; Zhang M et al., 2014; Alazab, 2015; Chuang and Wang, 2015; Mohaisen and Alrawi, 2015; Wang et al., 2015
Evolution	Bailey et al., 2007; Dumitras and Neamtiu, 2011; Jang et al., 2013; Anderson et al., 2014; Lee et al., 2015

However, for malware evolution, there are some differences. In malware evolution research, the relationships considered between different samples include not only similarity, but also the degree of similarity as well as the order of malware samples in a family evolution graph. Thus, behavior data is used to compute the similarity, distance, and likelihood orders of two samples. The malware evolution relationship can provide extremely useful information in many security scenarios. For example, it can help analysts understand trends over time and make informed decisions about which malware samples to analyze first. This is particularly important since the order in which the variants of a malware family are captured does not necessarily mirror the evolution of the malware.

Generally, malware evolution is considered a special case in software evolution. In the literature malware evolution is treated as a composition of the code transformations (Walenstein and Lakhotia, 2012). However, the scope of behavior-based malware evolution analysis is broader than that for software evolution. The behavior characteristics, such as similarity of function call APIs, branches in control flow graphs, and high-level semantic behavior, can be considered the key factors for behavior changes throughout the malware evolution process. Although only a few studies focus on behavior-based evolution analysis, some achievements have been achieved. For example, Lee et al. (2015) leveraged the cosine similarity method to compute the cosine similarities between API sequence subsets, and they used a local clustering coefficient method to calculate the closeness of a malware sample to a family. Thus, the evolution relationships can be reconstructed using these metrics.

Jang et al. (2013) systematically studied software lineage inference based on dynamic analysis. A system called the 'ILINE system' proposed in the work can automatically infer the software lineage of malware samples. The experiment results revealed that partial-order mismatches and graph-arc edit distance often yield the most meaningful comparisons.

Anderson et al. (2014) presented a novel algorithm based on a graphical lasso to analyze malware evolution using both static and dynamic data. With the graphical lasso, the behavior data, which contains the dynamic instruction traces and the dynamic system call traces, is used to create an evolution graph. The evolution graph offers analysts a better understanding of how a malware sample has evolved by clearly illustrating the lineage of its family.

Dumitras and Neamtiu (2011) proposed an evolutionary analysis approach to reconstruct the evolution trees using control-flow graphs, the idea for which was inspired from software evolution theories. The approach can convert control flow graphs into time series, where each data point corresponds to a node in the graph and the time it was observed, and the amplitude for each data point corresponds to the node's topological rank in a control flow graph. The techniques based on time-series similarity can prevent zero-day attacks because if the time series of an unknown sample is similar to the time series of a known malware sample, the sample could possibly be a new, previously unknown strain of an existing malware family.

3.1.2 Classification of behavior data levels

One interesting aspect of malware behavior analysis is that different levels of behavior data are collected or extracted in existing studies. Given the scope of the different kinds of behavior data, we can classify behavior data types into different levels (Table 2).

The first level of behavior data contains the original behavior data obtained from both dynamic and static analyses, such as system calls and function calls. For example, in some studies (Babić et al., 2011, 2012; Palahan et al., 2013; Sirinda, 2014), system call APIs from program execution are collected by the proposed systems to group malware samples. However, in other related works, the contexts of malware execution, such as function argument and environment variables, and even the program execution system states, are collected as factors that are related to malware behavior.

The second level contains control flow and data flow data, which are more complex than the original behavior data. At this level, the relationships of API calls, such as sequence, branch, and loop, are also considered. The behavior model extracted from both control flow and data flow is more useful for computing the similarity of different malware samples.

The third level covers the behavior attributes of security-critical system resources. Behavior attributes

Table 2 Comparison of different levels of behavior data

Behavior data level	Related works
Origin system call data from dynamic analysis (arguments and context information included)	Bailey et al., 2007; Rieck et al., 2008; Bayer et al., 2009; Inoue et al., 2009; Jacob et al., 2009; Lanzi et al., 2009; Martignoni et al., 2009; Riley et al., 2009; Beaucamps et al., 2010; Comparetti et al., 2010; Huang HD et al., 2011, 2014; Lindorfer et al., 2011; Beaucamps et al., 2012; Bos, 2013; Dahl et al., 2013; Kirat et al., 2014; Shan and Wang, 2014; Suarez-Tangil et al., 2014; Kirat and Vigna, 2015; Lee et al., 2015; Yang W et al., 2015; Das et al., 2016; Galal et al., 2016
Origin system call data from dynamic analysis (arguments and context information NOT included)	Martignoni et al., 2008; Babicć et al., 2011, 2012; Dumitras and Neamtiu, 2011; Sun et al., 2011; Palahan et al., 2013; Anderson et al., 2014; Sirinda, 2014; Naval et al., 2015; Wang et al., 2015; Kharraz et al., 2016
Origin function call data and program information from static analysis	Brumley et al., 2008; Fredrikson et al., 2010; Wu et al., 2012; Cao et al., 2013; Ding et al., 2013; Arp et al., 2014; Yang C et al., 2014; Saxe et al., 2014; Feng et al., 2014, 2017; Poeplau et al., 2014; Shi et al., 2014; Cen et al., 2015; Chuang and Wang, 2015; Yerima et al., 2015; Fratantonio et al., 2016; Miao et al., 2016
Origin data from both dynamic and static analysis	Kirda et al., 2006; Alazab et al., 2010; Alazab, 2015
Kernel function call and hardware-level system states from dynamic analysis	Yin et al., 2008; Wang et al., 2008; Lanzi et al., 2009; Neugschwandtner et al., 2010; Shosha et al., 2012; Park et al., 2013; Watson et al., 2016
Data flow	Martignoni et al., 2009; Fredrikson et al., 2010; Feng et al., 2014; Yuan et al., 2014; Wüchner et al., 2015
Control flow	Comparetti et al., 2010; Cesare et al., 2014; Ding et al., 2014; Zhao et al., 2014

performed on system resources, such as File, Registry, Process, and Network, are self-explanatory. For example, a delete action means that an existing file or directory is deleted. In contrast to the original behavior data, behavior attributes leverage the meaning of function APIs on system resources, and thus the basic functionalities of the malware samples can be understood, which is useful in understanding malware behavior.

The last level is a higher level of abstraction behavior. For example, malware abstraction behaviors, such as Keylogger and Exfiltrator, can be used to detect malicious content in the behaviors of malware samples. With a semantic understanding of abstraction behaviors, malware behaviors can be shown clearly and intuitively, and the effectiveness of malware detection is significantly improved with clear boundaries that indicate the malware behavior specifications.

Choosing behavior data depends on the analysis goals, malware confusion techniques, and analysis methods. For example, Anderson et al. (2014) used six different types of behavior data with the aim of covering the most popular data views. The reason is that different malware families leverage different obfuscation techniques, which limits the information on a single data view. Table 2 also demonstrates that the behavior data extracted from both dynamic analysis and static analysis, or behavior data in the different levels, can be used for behavior-based malware analysis.

3.1.3 Classification of analysis techniques

As discussed in Section 2.2, several kinds of machine learning models can be used for behavior analysis. Some basic algorithms, such as naive Bayes, decision tree, and random forest, are usually used for malware classification. The categories of learning algorithms in the existing behavior analysis approaches are shown in Table 3.

Although typical algorithms provide good detection results, some deep learning algorithms, such as multilayer perceptrons and neural networks, are also used to enhance the stability of machine learning based behavior analysis approaches and reduce the workload for feature engineering in traditional machine learning approaches. For example, Dahl et al. (2013) presented a malware classification approach

Table 3 Statistics from machine learning algorithms

Learning algorithm	Related works
DBM-tree	Cesare et al., 2014
Objective-oriented association mining (OOA)	Ding et al., 2013
Hidden Markov model (HMM)	Canfora et al., 2016
Multilayer perceptron (MLP)	Mohaisen and Alrawi, 2015; Das et al., 2016
Naive Bayes (NB)	Cao et al., 2013; Yang C et al., 2014; Zhang M et al., 2014; Cen et al., 2015; Yerima et al., 2015; Zhang H et al., 2016
Decision tree (DT)	Dube et al., 2012; Yang C et al., 2014; Ding et al., 2014; Zhao et al., 2014; Yerima et al., 2015; Galal et al., 2016
Support vector machine (SVM)	Rieck et al., 2008; Cao et al., 2013; O'Kane et al., 2013; Arp et al., 2014; Yang C et al., 2014; Ding et al., 2014; Chuang and Wang, 2015; Mohaisen and Alrawi, 2015; Galal et al., 2016; Zhang H et al., 2016; Miao et al., 2016; Watson et al., 2016
K-nearest neighbor (KNN)	Wu et al., 2012; Ding et al., 2014; Alazab, 2015; Cen et al., 2015; Mohaisen and Alrawi, 2015; Wang et al., 2015
Bayesian network (BN)	Cao et al., 2013; Zhang H et al., 2016
Self-defined cluster algorithm	Bayer et al., 2009; Wu et al., 2012; Shan and Wang, 2014
Random forest (RF)/random tree (RT)	Cao et al., 2013; Yang C et al., 2014; Zhao et al., 2014; Wüchner et al., 2015; Yerima et al., 2015; Galal et al., 2016
Hierarchical clustering	Bailey et al., 2007; Shi et al., 2014; Kirat and Vigna, 2015
Self-defined cluster and classification algorithm	Rieck et al., 2011; Lee et al., 2015
Logistic regression (LR)	Mohaisen and Alrawi, 2015; Yerima et al., 2015
Principal component analysis (PCA)	Cesare et al., 2014
Neural network (NN)	Dahl et al., 2013
J48	Cao et al., 2013

using random projections to reduce the dimension of the original input space. The experimental results showed that an error rate of 0.42% can be achieved.

Besides, the ensemble method is a popular behavior analysis approach. Its idea is to generate multiple predictors used in combination to classify new unseen samples, and its goal is to obtain better predictive performance compared with any of the constituent learning algorithms alone. For example, Yerima et al. (2015) used NB, DT, and RF methods to provide comprehensive results for malware analysis. Galal et al. (2016) proposed a detection framework which employs various classification techniques to evaluate the accuracy.

Table 4 shows a comparison of different analysis techniques in the existing investigations. Both machine learning based and semantics-based approaches have attracted wide attention in academia. As discussed in Section 2.2, semantics-based approaches use semantic rules or behavior knowledge to provide an in-depth understanding of malware behaviors. A prominent advantage of a semantics-based approach is that it has a wide range for adaptation and may be used to analyze malware samples on different platforms, such as a Windows platform with PE format, Linux platform with ELF format, and Android platform with DEX format. The Android platform is an emerging field constantly threatened by malware.

3.2 Challenges and suggestions

Based on the discussion and analysis in Section 3.1, we can summarize several considerable challenges and suggestions. Although there are many works on malware behavior analysis, more efforts are still needed for better effectiveness and accuracy. The challenges under consideration are as follows.

3.2.1 Coverage of behavior data

Although behavior analysis based on program execution traces has been widely studied, a prominent problem of dynamic trace extraction is that the coverage of dynamic traces will be affected by conditions in virtual environments. Various dynamic traces can be obtained in different virtual environments and

Table 4 Comparison of different analysis techniques

Analysis techniques	Related works
Learning-based approaches	Bailey et al., 2007; Rieck et al., 2008; Bayer et al., 2009; Martignoni et al., 2009; Dumitras and Neamtiu, 2011; Sun et al., 2011; Dahl et al., 2013; Ding et al., 2013; Anderson et al., 2014; Arp et al., 2014; Kirat et al., 2014; Saxe et al., 2014; Shan and Wang, 2014; Shi et al., 2014; Zhang M et al., 2014; Alazab, 2015; Cen et al., 2015; Mohaisen and Alrawi, 2015; Naval et al., 2015; Wang et al., 2015; Wüchner et al., 2015; Yerima et al., 2015; Galal et al., 2016; Watson et al., 2016
Semantic-based approaches	Jacob et al., 2009; Alazab et al., 2010; Babić et al., 2011, 2012; Bos, 2013; Palahan et al., 2013; Sirinda, 2014
Semantic-based approaches	Kirda et al., 2006; Brumley et al., 2008; Yin et al., 2008; Wang et al., 2008; Bayer et al., 2009; Inoue et al., 2009; Lanzi et al., 2009; Riley et al., 2009; Beaucamps et al., 2010, 2012; Comparetti et al., 2010; Fredrikson et al., 2010; Neugschwandtner et al., 2010; Dumitras and Neamtiu, 2011; Lindorfer et al., 2011; Shosha et al., 2012; Jang et al., 2013; Huang HD et al., 2014; Shan and Wang, 2014; Poeplau et al., 2014; Lebiere et al., 2015; Naval et al., 2015; Thomson et al., 2015; Yang W et al., 2015; Kharraz et al., 2016; Fratantonio et al., 2016
Hybrid approaches	Lindorfer et al., 2011; Wu et al., 2012; Cao et al., 2013; Park et al., 2013; Arp et al., 2014; Saxe et al., 2014; Yang C et al., 2014; Chuang and Wang, 2015; Kirat and Vigna, 2015; Lee et al., 2015; Das et al., 2016; Miao et al., 2016

different contexts. On the other hand, malware samples may use evasion techniques to avoid being detected. Thus, the behavior traces for different variants of the same malware family may be different, which will affect the effectiveness of the behavior analysis approaches.

A better solution is to combine dynamic and static analyses to obtain behavior data from multiple sources and employ multiple learning algorithms to find a weighted combination of the data sources, which yields the best detection accuracy in behavior analysis.

Another solution is to obtain the full behavioral dependency graph of a malware sample using a tainted analysis technique (Yuan et al., 2014) and symbolic execution technique (Fratantonio et al., 2016). With the powerful computing and storage capabilities of modern computers, these advanced program analysis techniques are becoming increasingly popular and practical. The full behavior information can be obtained using advanced program analysis techniques without regard to the evasions on the part of malware behaviors.

3.2.2 Unknown behavior detection

The detection of unknown samples is a goal of malware behavior analysis. It is known that an inherent problem of signature-based detection techniques is the inability to detect unknown threats. On the contrary, by abstracting semantic representations of malware behaviors, unknown samples can be detected even for a previously unknown family. To achieve this, behavior analysis techniques must define semantic rules and construct semantic inference models. Aiming at unknown malware detection, control flow based and semantics-based behavior analysis approaches are optional solutions to model malware behaviors. With semantic concepts such as behavior profiles and behavior capabilities, a high-level understanding of malware behaviors can be obtained and then detection rules based on the semantic concepts can be well designed.

According to the discussion in Section 2.2.2, there are several semantics-based analysis methods, for example, semantically relevant path analysis (Naval et al., 2015) and high-level semantic analysis of system calls (Das et al., 2016). These techniques can capture the high-level semantics of malicious behaviors; therefore, they are well suited for capturing new and syntactically different but semantically similar unknown malware samples.

3.2.3 Malware adversarial behavior

Adversarial machine learning is an emerging field of study against an adversarial opponent (Huang L et al., 2011). By attacking machine learning algorithms, feature space, training and test data, adversarial machine learning can affect the effectiveness of

machine learning based approaches (Biggio et al., 2014). The test data can be carefully manipulated by a malicious adversary to exploit specific vulnerabilities of learning algorithms. In fact, in the process of machine learning based behavior analysis, behavior data, such as system call traces, system call arguments, and dynamic instruction traces, is easily distorted by modifying the malware sample and inserting forging code segments.

For example, Ming et al. (2015, 2017) presented a new attack method called the 'replacement attack', which operates by concealing similar behaviors among malware samples to poison the behavior data. Two categories of attack strategies are designed to modify system call flows, including inserting redundant dependency and system call dependency graph mutations. The experimental results showed that a replacement attack can not only subvert approaches based on behavior similarity measurement, such as the graph edit distance and the Jaccard index, but also impede behavior-based malware clustering approaches, such as locality-sensitive hashing and single-linkage hierarchical clustering.

A countermeasure is to unify the behavior patterns with semantic equivalent rules before any classification or clustering approach is applied to the behavior data. Another possible solution is to replace traditional machine learning algorithms with deep learning algorithms in the behavior analysis process, because the latter are more resistant to adversarial attacks.

3.2.4 Malware evolution analysis

The role of malware evolution analysis is twofold. First, the evolutionary history of a captured malware sample can speed up the security response to attack events. Second, evolutionary trends enable proactive development of defenses (Gupta et al., 2009). Understanding the evolution direction and degree could yield new techniques for detecting and classifying unknown attacks.

Rigorous experiments and empirical studies have demonstrated the demand for better approaches for malware evolution. For example, Zhou and Jiang (2012) revealed that malware families are evolving rapidly to circumvent detection by existing security solutions. The experimental results showed that the maximum detection rate of malware variants is 79.6% and the minimum is 20.2%. This result clearly illustrates a strong need for a better understanding of the malware evolution history and a better examination of the evolutionary trends in malware families.

Unfortunately, many challenges stand in the way, for example, the lack of sufficient contextual data (such as the contextual information about a malware attack), the lack of metadata about the collection process of existing sample sets, the lack of ground truth, and the difficulty in developing tools and methods for rigorous data analysis. From another point of view, the rise in open source projects has greatly improved the complexity and range of malware evolutions by using or sharing open-source malware modules.

We believe that the changes in malware samples, such as updating patches, adding new functionality modules, and forking and modifying open source modules, will affect the frequency of the behavior types, structural characteristics of control flow graphs, structural characteristics of system call graphs, high-level behavior attributes, etc. The frequency and characteristics are effective proof of malware evolution. Furthermore, these behavior proofs will be more stable and abundant than file structure proof from static analysis. Another advantage of behavior-based evolution research is that the behavior trends that malware variants have in common can be revealed.

Another argument is that cluster-based machine learning can be used to cluster malware samples with similar behavioral characteristics to form malware associations. Thus, cluster-based techniques can assist in the evolution analysis process.

4 Conclusions

Malware behavior analysis is one of the most important measures in the security response to malware threats in cyberspace. Although many studies have been conducted for malware behavior analysis, more efforts are still needed to understand the mechanisms, regularity, and trends in malware behavior.

In this paper, we aim to provide insight into the status of behavior analysis techniques. We have performed a comprehensive review of the latest malware behavior analysis techniques and discussed the existing research classified from five different perspec-

tives, clearly showing the advantages and disadvantages of existing analysis methods. Additionally, we have discussed some inadequacies and challenges that are currently not solved as well as several possible solutions to address the current shortcomings. It is important to understand the characteristics and trends in various malware behaviors to promote the development of efficient and accurate malware behavior analysis techniques.

References

Alam S, Horspool RN, Traore I, et al., 2015. A framework for metamorphic malware analysis and real-time detection. *Comput Secur*, 48:212-233. https://doi.org/10.1016/j.cose.2014.10.011

Alazab M, 2015. Profiling and classifying the behavior of malicious codes. *J Syst Softw*, 100:91-102. https://doi.org/10.1016/j.jss.2014.10.031

Alazab M, Venkataraman S, Watters P, 2010. Towards Understanding malware behaviour by the extraction of API calls. Proc 2nd Cybercrime and Trustworthy Computing Workshop, p.52-59. https://doi.org/10.1109/CTC.2010.8

Anderson B, Storlie C, Lane T, 2012. Improving malware classification: Bridging the static/dynamic gap. Proc 5th ACM Workshop on Security and Artificial Intelligence, p.3-14. https://doi.org/10.1145/2381896.2381900

Anderson B, Lane T, Hash C, 2014. Malware phylogenetics based on the multiview graphical lasso. Proc 13th Int Symposium on Advances in Intelligent Data Analysis XIII, p.1-12. https://doi.org/10.1007/978-3-319-12571-8_1

Arp D, Spreitzenbarth M, Hübner M, et al., 2014. DREBIN: effective and explainable detection of Android malware in your pocket. Proc 17th Network and Distributed System Security Symp, p.1-16. https://doi.org/10.14722/ndss.2014.23247

Babić D, Reynaud D, Song DW, 2011. Malware analysis with tree automata inference. Proc 23rd Int Conf on Computer Aided Verification, p.116-131. https://doi.org/10.1007/978-3-642-22110-1_10

Babić D, Reynaud D, Song DW, 2012. Recognizing malicious software behaviors with tree automata inference. *Form Methods Syst Des*, 41(1):107-128. https://doi.org/10.1007/s10703-012-0149-1

Bailey M, Oberheide J, Andersen J, et al., 2007. Automated classification and analysis of Internet malware. Proc 10th Int Symp on Recent Advances in Intrusion Detection, p.178-197. https://doi.org/10.1007/978-3-540-74320-0_10

Barnum S, 2012. Standardizing cyber threat intelligence information with the structured threat information eXpression (STIX™). https://www.mitre.org/sites/default/files/publications/stix.pdf

Bauman E, Ayoade G, Lin ZQ, 2015. A survey on hypervisor-based monitoring: approaches, applications, and evolutions. *ACM Comput Surv*, 48(1), Article 10. https://doi.org/10.1145/2775111

Bayer U, Kruegel C, Kirda E, 2006. TTAnalyze: a tool for analyzing malware. Proc 15th Annual Conf of the European Institute for Computer Antivirus Research, p.180-192.

Bayer U, Comparetti PM, Hlauscheck C, et al., 2009. Scalable, behavior-based malware clustering. Proc 16th Symp on Network and Distributed System Security, p.1-21.

Bayer U, Habibi I, Balzarotti D, et al., 2014. A view on current malware behaviors. Proc 2nd USENIX Conf on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More, p.8.

Beaucamps P, Gnaedig I, Marion JY, 2010. Behavior abstraction in malware analysis. Proc 1st Int Conf on Runtime Verification, p.168-182. https://doi.org/10.1007/978-3-642-16612-9_14

Beaucamps P, Gnaedig I, Marion JY, 2012. Abstraction-based malware analysis using rewriting and model checking. Proc 17th European Symp on Research in Computer Security, p.806-823. https://doi.org/10.1007/978-3-642-33167-1_46

Belaoued M, Mazouzi S, 2015. A real-time pe-malware detection system based on CHI-square test and pe-file features. Proc 5th IFIP TC 5 Int Conf on Science and Its Applications, p.416-425. https://doi.org/10.1007/978-3-319-19578-0_34

Biggio B, Rieck K, Ariu D, et al., 2014. Poisoning behavioral malware clustering. Proc Workshop on Artificial Intelligent and Security Workshop, p.27-36. https://doi.org/10.1145/2666652.2666666

Bos H, 2013. Analysis report of behavioral features. http://www.wombat-project.eu/2010/07/wombat-deliverable-d16d42-anal.html

Brumley D, Hartwig C, Liang ZK, et al., 2008. Automatically identifying trigger-based behavior in malware. In: Lee W, Wang C, Dagon D (Eds.), Botet Detection. Springer, Boston, MA, p.65-88. https://doi.org/10.1007/978-0-387-68768-1_4

Canfora G, Mercaldo F, Visaggio CA, 2016. An hmm and structural entropy based detector for Android malware: an empirical study. *Comput Secur*, 61:1-18. https://doi.org/10.1016/j.cose.2016.04.009

Cao Y, Miao QG, Liu JC, et al., 2013. Abstracting minimal security-relevant behaviors for malware analysis. *J Comput Virol Hack Tech*, 9(4):193-204. https://doi.org/10.1007/s11416-013-0186-3

Cen L, Gates CS, Si L, et al., 2015. A probabilistic discriminative model for Android malware detection with decompiled source code. *IEEE Trans Depend Sec Comput*, 12(4):400-412. https://doi.org/10.1109/TDSC.2014.2355839

Cesare S, Xiang Y, Zhou WL, 2014. Control flow-based malware variant detection. *IEEE Trans Depend Sec*

Comput, 11(4):307-317. https://doi.org/10.1109/TDSC.2013.40

Chandramohan M, Tan HBK, Shar LK, 2012. Scalable malware clustering through coarse-grained behavior modeling. Proc ACM SIGSOFT 20th Int Symp on the Foundations of Software Engineering, article 27. https://doi.org/10.1145/2393596.2393627

Christodorescu M, Jha S, Kruegel C, 2008. Mining specifications of malicious behavior. Proc 1st India Software Engineering Conf, p.5-14. https://doi.org/10.1145/1342211.1342215

Chuang HY, Wang SD, 2015. Machine learning based hybrid behavior models for Android malware analysis. Proc IEEE Int Conf on Software Quality, Reliability and Security, p.201-206. https://doi.org/10.1109/QRS.2015.37

Comparetti PM, Salvaneschi G, Kirda E, et al., 2010. Identifying dormant functionality in malware programs. Proc IEEE Symp on Security and Privacy, p.61-76. https://doi.org/10.1109/SP.2010.12

Cuckoo, 2017. Cuckoo sandbox. https://cuckoosandbox.org

Dahl GE, Stokes JW, Deng L, et al., 2013. Large-scale malware classification using random projections and neural networks. Proc IEEE Int Conf on Acoustics, Speech and Signal Processing, p.3422-3426. https://doi.org/10.1109/ICASSP.2013.6638293

Damodaran A, di Troia F, Visaggio CA, et al., 2017. A comparison of static, dynamic, and hybrid analysis for malware detection. *J Comput Virol Hack Tech*, 13(1): 1-12. https://doi.org/10.1007/s11416-015-0261-z

Das S, Liu Y, Zhang W, et al., 2016. Semantics-based online malware detection: towards efficient real-time protection against malware. *IEEE Trans Inform Forens Secur*, 11(2): 289-302. https://doi.org/10.1109/TIFS.2015.2491300

Deschamps N, 2008. Specification language for code behavior. http://wombat-project.eu/WP4/FP7-ICT-216026-Wombat_WP4_D08_V01_Specification_language_for_code_behaviour.pdf

Dinaburg A, Royal P, Sharif M, et al., 2008. Ether: malware analysis via hardware virtualization extensions. Proc 15th ACM Conf on Computer and Communications Security, p.51-62. https://doi.org/10.1145/1455770.1455779

Ding YX, Yuan XB, Tang K, et al., 2013. A fast malware detection algorithm based on objective-oriented association mining. *Comput Secur*, 39:315-324. https://doi.org/10.1016/j.cose.2013.08.008

Ding YX, Dai W, Yan SL, et al., 2014. Control flow-based opcode behavior analysis for malware detection. *Comput Secur*, 44:65-74. https://doi.org/10.1016/j.cose.2014.04.003

Dube T, Raines R, Peterson G, et al., 2012. Malware target recognition via static heuristics. *Comput Secur*, 31(1): 137-147. https://doi.org/10.1016/j.cose.2011.09.002

Dumitras T, Neamtiu I, 2011. Experimental challenges in cyber security: a story of provenance and lineage for malware. Proc 4th Conf on Cyber Security Experimentation and Test, p.9.

Egele M, Scholte T, Kirda E, et al., 2012. A survey on automated dynamic malware-analysis techniques and tools. *ACM Comput Surv*, 44(2), Article 6. https://doi.org/10.1145/2089125.2089126

Elhadi AAE, Maarof MA, Barry BIA, et al., 2014. Enhancing the detection of metamorphic malware using call graphs. *Comput Secur*, 46:62-78. https://doi.org/10.1016/j.cose.2014.07.004

Feng Y, Anand S, Dillig I, et al., 2014. Apposcopy: semantics-based detection of Android malware through static analysis. Proc 22nd ACM SIGSOFT Int Symp on Foundations of Software Engineering, p.576-587. https://doi.org/10.1145/2635868.2635869

Feng Y, Bastani O, Martins R, et al., 2017. Automated synthesis of semantic malware signatures using maximum satisfiability. Proc Network and Distributed System Security Symp, p.1-16. https://doi.org/10.14722/ndss.2017.23379

Fratantonio Y, Bianchi A, Robertson W, et al., 2016. Triggerscope: towards detecting logic bombs in Android applications. Proc IEEE Symp on Security and Privacy, p.377-396. https://doi.org/10.1109/SP.2016.30

Fredrikson M, Jha S, Christodorescu M, et al., 2010. Synthesizing near-optimal malware specifications from suspicious behaviors. Proc IEEE Symp on Security and Privacy, p.45-60. https://doi.org/10.1109/SP.2010.11

Galal HS, Mahdy YB, Atiea MA, 2016. Behavior-based features model for malware detection. *J Comput Virol Hack Tech*, 12(2):59-67. https://doi.org/10.1007/s11416-015-0244-0

Grégio ARA, Baruque AOC, Afonso VM, et al., 2012. Interactive, visual-aided tools to analyze malware behavior. Proc 12th Int Conf on Computational Science and Its Applications, p.302-313. https://doi.org/10.1007/978-3-642-31128-4_22

Gupta A, Kuppili P, Akella A, et al., 2009. An empirical study of malware evolution. Proc 1st Int Communication Systems and NETworks and Workshops, p.1-10. https://doi.org/10.1109/COMSNETS.2009.4808876

Haass JC, Ahn GJ, Grimmelmann F, 2015. ACTRA: a case study for threat information sharing. Proc 2nd ACM Workshop on Information Sharing and Collaborative Security, p.23-26. https://doi.org/10.1145/2808128.2808135

Huang HD, Acampora G, Loia V, et al., 2011. Applying FML and fuzzy ontologies to malware behavioural analysis. Proc IEEE Int Conf on Fuzzy Systems, p.2018-2025. https://doi.org/10.1109/FUZZY.2011.6007716

Huang HD, Lee CS, Wang MH, et al., 2014. IT2FS-based ontology with soft-computing mechanism for malware behavior analysis. *Soft Comput*, 18(2):267-284. https://doi.org/10.1007/s00500-013-1056-0

Huang L, Joseph AD, Nelson B, et al., 2011. Adversarial machine learning. Proc 4th ACM Workshop on Security and Artificial Intelligence, p.43-58.

https://doi.org/10.1145/2046684.2046692

Inoue D, Yoshioka K, Eto M, et al., 2009. Automated malware analysis system and its sandbox for revealing malware's internal and external activities. *IEICE Trans Inform Syst*, E92.D(5):945-954. https://doi.org/10.1587/transinf.E92.D.945

Jacob G, Debar H, Filiol E, 2009. Malware behavioral detection by attribute-automata using abstraction from platform and language. Proc 12th Int Symp on Recent Advances in Intrusion Detection, p.81-100. https://doi.org/10.1007/978-3-642-04342-0_5

Jang J, Woo M, Brumley D, 2013. Towards automatic software lineage inference. Proc 22nd USENIX Conf on Security, p.81-96.

Kharraz A, Arshad S, Mulliner C, et al., 2016. UNVEIL: a large-scale, automated approach to detecting ransomware. Proc 25th USENIX Security Symp, p.757-772.

Kirat D, Vigna G, 2015. MalGene: automatic extraction of malware analysis evasion signature. Proc 22nd ACM SIGSAC Conf on Computer and Communications Security, p.769-780. https://doi.org/10.1145/2810103.2813642

Kirat D, Vigna G, Kruegel C, 2014. Barecloud: bare-metal analysis-based evasive malware detection. Proc 23rd USENIX Conf on Security Symp, p.287-301.

Kirda E, Kruegel C, Banks G, et al., 2006. Behavior-based spyware detection. Proc 15th Conf on USENIX Security Symp, Article 19.

Kirillov I, Beck D, Chase P, et al., 2011. Malware attribute enumeration and characterization (MAEC™). http://maec.mitre.org/

Kokkonen T, Hautamaki J, Siltanen J, et al., 2016. Model for sharing the information of cyber security situation awareness between organizations. Proc 23rd Int Conf on Telecommunications, p.1-5. https://doi.org/10.1109/ICT.2016.7500406

Kruegel C, 2014. Full system emulation: achieving successful automated dynamic analysis of evasive malware. Lastline, Inc., Las Vegas, NV, USA.

Lanzi A, Sharif M, Lee W, 2009. K-Tracer: a system for extracting kernel malware behavior. Proc Network and Distributed System Security Symp, p.163-169.

Lebiere C, Bennati S, Thomson R, et al., 2015. Functional cognitive models of malware identification. Proc 13th Annual Conf on Cognitive Modeling, p.90-95.

Leder F, Steinbock B, Martini P, 2009. Classification and detection of metamorphic malware using value set analysis. Proc 4th Int Conf on Malicious and Unwanted Software, p.39-46. https://doi.org/10.1109/MALWARE.2009.5403019

Lee T, Choi B, Shin Y, et al., 2015. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient. *J Supercomput*, p.1-15. https://doi.org/10.1007/s11227-015-1594-6

Lindorfer M, Kolbitsch C, Comparetti PM, 2011. Detecting environment-sensitive malware. Proc 14th Int Symp on Recent Advances in Intrusion Detection, p.338-357. https://doi.org/10.1007/978-3-642-23644-0_18

Liu L, Wang BS, Yu B, et al., 2016. A novel selective ensemble learning based on K-means and negative correlation. Proc 2nd Int Conf on Cloud Computing and Security, p.578-588. https://doi.org/10.1007/978-3-319-48674-1_51

Martignoni L, Stinson E, Fredrikson M, et al., 2008. A layered architecture for detecting malicious behaviors. Proc 11th Int Symp on Recent Advances in Intrusion Detection, p.78-97. https://doi.org/10.1007/978-3-540-87403-4_5

Martignoni L, Paleari R, Bruschi D, 2009. A framework for behavior-based malware analysis in the cloud. Proc 5th Int Conf on Information Systems Security, p.178-192. https://doi.org/10.1007/978-3-642-10772-6_14

Miao QG, Liu JC, Cao Y, et al., 2016. Malware detection using bilayer behavior abstraction and improved one-class support vector machines. *Int J Inform Secur*, 15(4):361-379. https://doi.org/10.1007/s10207-015-0297-6

Ming J, Xin Z, Lan PW, et al., 2015. Replacement attacks: automatically impeding behavior-based malware specifications. Proc 13th Int Conf on Applied Cryptography and Network Security, p.497-517. https://doi.org/10.1007/978-3-319-28166-7_24

Ming J, Xin Z, Lan PW, et al., 2017. Impeding behavior-based malware analysis via replacement attacks to malware specifications. *J Comput Virol Hack Tech*, 13(3):193-207. https://doi.org/10.1007/s11416-016-0281-3

Mithal T, Shah K, Singh DK, 2016. Case studies on intelligent approaches for static malware analysis. In: Shetty NR, Prasad NH, Nalini N (Eds.), Emerging Research in Computing, Information, Communication and Applications. Springer, Singapore, p.555-567. https://doi.org/10.1007/978-981-10-0287-8_52

Mohaisen A, Alrawi O, 2015. AMAL: high-fidelity, behavior-based automated malware analysis and classification. Proc 15th Int Workshop on Information Security Applications, p.107-121. https://doi.org/10.1007/978-3-319-15087-1

Moonsamy V, Tian RH, Batten L, 2012. Feature reduction to speed up malware classification. Proc 16th Nordic Conf on Information Security Technology for Applications, p.176-188. https://doi.org/10.1007/978-3-642-29615-4_13

Moser A, Kruegel C, Kirda E, 2007. Exploring multiple execution paths for malware analysis. Proc IEEE Symp on Security and Privacy, p.231-245. https://doi.org/10.1109/SP.2007.17

Naval S, Laxmi V, Rajarajan M, et al., 2015. Employing program semantics for malware detection. *IEEE Trans Inform Forens Secur*, 10(12):2591-2604. https://doi.org/10.1109/TIFS.2015.2469253

Neugschwandtner M, Platzer C, Comparetti PM, et al., 2010. *d*Anubis—dynamic device driver analysis based on virtual machine introspection. Proc 7th Int Conf on Detection of Intrusions and Malware, and Vulnerability

Assessment, p.41-60.
https://doi.org/10.1007/978-3-642-14215-4_3

Nunes E, Buto C, Shakarian P, et al., 2015. Malware task identification: a data driven approach. Proc IEEE/ACM Int Conf on Advances in Social Networks Analysis and Mining, p.978-985.
https://doi.org/10.1145/2808797.2808894

O'Kane P, Sezer S, McLaughlin K, et al., 2013. SVM training phase reduction using dataset feature filtering for malware detection. *IEEE Trans Inform Forens Secur*, 8(3):500-509.
https://doi.org/10.1109/TIFS.2013.2242890

Palahan S, Babić D, Chaudhuri S, et al., 2013. Extraction of statistically significant malware behaviors. Proc 29th Annual Computer Security Applications Conf, p.69-78.
https://doi.org/10.1145/2523649.2523659

Park Y, Reeves DS, Stamp M, 2013. Deriving common malware behavior through graph clustering. *Comput Secur*, 39:419-430.
https://doi.org/10.1016/j.cose.2013.09.006

Pleszkoch M, Linger R, 2015. Controlling combinatorial complexity in software and malware behavior computation. Proc 10th Annual Cyber and Information Security Research Conf, Article 15.
https://doi.org/10.1145/2746266.2746281

Poeplau S, Fratantonio Y, Bianchi A, et al., 2014. Execute this! Analyzing unsafe and malicious dynamic code loading in Android applications. Proc Network and Distributed System Security Symp, p.23-26.
https://doi.org/10.14722/ndss.2014.23328

Razak MFA, Anuar NB, Salleh R, et al., 2016. The rise of "malware": bibliometric analysis of malware study. *J Netw Comput Appl*, 75:58-76.
https://doi.org/10.1016/j.jnca.2016.08.022

Rieck K, Holz T, Willems C, et al., 2008. Learning and classification of malware behavior. Proc 5th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment, p.108-125.
https://doi.org/10.1007/978-3-540-70542-0_6

Rieck K, Trinius P, Willems C, et al., 2011. Automatic analysis of malware behavior using machine learning. *J Comput Secur*, 19(4):639-668.
https://doi.org/10.3233/JCS-2010-0410

Riley R, Jiang XX, Xu DY, 2009. Multi-aspect profiling of kernel rootkit behavior. Proc 4th ACM European Conf on Computer Systems, p.47-60.
https://doi.org/10.1145/1519065.1519072

Royal P, Halpin M, Dagon D, et al., 2006. PolyUnpack: automating the hidden-code extraction of unpack-executing malware. Proc 22nd Annual Computer Security Applications Conf, p.289-300.
https://doi.org/10.1109/ACSAC.2006.38

Saxe J, Mentis D, Greamo C, 2012. Visualization of shared system call sequence relationships in large malware corpora. Proc 9th Int Symp on Visualization for Cyber Security, p.33-40.
https://doi.org/10.1145/2379690.2379695

Saxe J, Turner R, Blokhin K, 2014. Crowdsource: automated inference of high level malware functionality from low-level symbols using a crowd trained machine learning model. Proc 9th Int Conf on Malicious and Unwanted Software: the Americas, p.68-75.
https://doi.org/10.1109/MALWARE.2014.6999417

Shan ZY, Wang X, 2014. Growing grapes in your computer to defend against malware. *IEEE Trans Inform Forens Secur*, 9(2):196-207.
https://doi.org/10.1109/TIFS.2013.2291066

Shi HB, Hamagami T, Yoshioka K, et al., 2014. Structural classification and similarity measurement of malware. *IEEJ Trans Electr Electron Eng*, 9(6):621-632.
https://doi.org/10.1002/tee.22018

Shosha AF, Liu C, Gladyshev P, et al., 2012. Evasion-resistant malware signature based on profiling kernel data structure objects. Proc 7th Int Conf on Risk and Security of Internet and Systems, p.1-8.
https://doi.org/10.1109/CRISIS.2012.6378949

Sirinda P, 2014. A framework for mining significant subgraphs and its application in malware analysis. PhD Thesis, The Pennsylvania State University, Pennsylvania, USA.

Suarez-Tangil G, Conti M, Tapiador JE, et al., 2014. Detecting targeted smartphone malware with behavior-triggering stochastic models. Proc 19th European Symp on Research in Computer Security, p.183-201.
https://doi.org/10.1007/978-3-319-11203-9_11

Sun MK, Lin MJ, Chang M, et al., 2011. Malware virtualization-resistant behavior detection. Proc 17th Int Conf on Parallel and Distributed Systems, p.912-917.
https://doi.org/10.1109/ICPADS.2011.78

Thomson R, Lebiere C, Bennati S, et al., 2015. Malware identification using cognitively-inspired inference. Proc 24th Annual Behavior Representation in Modeling and Simulation Conf, p.1-8.

Trinius P, Holz T, Göbel J, et al., 2009. Visual analysis of malware behavior using treemaps and thread graphs. Proc 6th Int Workshop on Visualization for Cyber Security, p.33-38. https://doi.org/10.1109/VIZSEC.2009.5375540

Trinius P, Willems C, Holz T, et al., 2011. A malware instruction set for behavior-based analysis.
http://subs.emis.de/LNI/Proceedings/Proceedings170/article5739.html

Walenstein A, Lakhotia A, 2012. A transformation-based model of malware derivation. Proc 7th Int Conf on Malicious and Unwanted Software, p.17-25.
https://doi.org/10.1109/MALWARE.2012.6461003

Wang SW, Wang BS, Yong T, et al., 2015. Malware clustering based on SNN density using system calls. Proc 1st Int Conf on Cloud Computing and Security, p.181-191.
https://doi.org/10.1007/978-3-319-27051-7_16

Wang Z, Jiang XX, Cui WD, et al., 2008. Countering persistent kernel rootkits through systematic hook discovery. Proc 11th Int Symp on Recent Advances in Intrusion Detection, p.21-38. https://doi.org/10.1007/978-3-540-87403-4_2

Watson MR, Shirazi NUH, Marnerides AK, et al., 2016. Malware detection in cloud computing infrastructures. *IEEE Trans Depend Sec Comput*, 13(2):192-205. https://doi.org/10.1109/TDSC.2015.2457918

Wu DJ, Mao CH, Wei TE, et al., 2012. DroidMat: Android malware detection through manifest and API calls tracing. Proc 7th Asia Joint Conf on Information Security, p.62-69. https://doi.org/10.1109/AsiaJCIS.2012.18

Wüchner T, Ochoa M, Pretschner A, 2015. Robust and effective malware detection through quantitative data flow graph metrics. Proc 12th Int Conf on Detection of Intrusions and Malware, and Vulnerability Assessment, p.98-118. https://doi.org/10.1007/978-3-319-20550-2_6

Yang C, Xu ZY, Gu GF, et al., 2014. DroidMiner: automated mining and characterization of fine-grained malicious behaviors in Android applications. Proc 19th European Symp on Research in Computer Security, p.163-182. https://doi.org/10.1007/978-3-319-11203-9_10

Yang W, Xiao XS, Andow B, et al., 2015. AppContext: differentiating malicious and benign mobile app behaviors using context. Proc 37th IEEE Int Conf on Software Engineering, p.303-313. https://doi.org/10.1109/ICSE.2015.50

Yavvari C, Tokhtabayev A, Rangwala H, et al., 2012. Malware characterization using behavioral components. Proc 6th Int Conf on Mathematical Methods, Models, and Architectures for Computer Network Security, p.226-239. https://doi.org/10.1007/978-3-642-33704-8_20

Yerima SY, Sezer S, Muttik I, 2015. High accuracy Android malware detection using ensemble learning. *IET Inform Secur*, 9(6):313-320. https://doi.org/10.1049/iet-ifs.2014.0099

Yin H, Liang ZK, Song D, 2008. HookFinder: identifying and understanding malware hooking behaviors. Proc Network and Distributed System Security Symp, p.1-16.

Yuan JF, Qiang WZ, Jin H, et al., 2014. Cloudtaint: an elastic taint tracking framework for malware detection in the cloud. *J Supercomput*, 70(3):1433-1450. https://doi.org/10.1007/s11227-014-1235-5

Zhang FW, Leach K, Stavrou A, et al., 2015. Using hardware features for increased debugging transparency. Proc IEEE Symp on Security and Privacy, p.55-69. https://doi.org/10.1109/SP.2015.11

Zhang H, Yao DF, Ramakrishnan N, et al., 2016. Causality reasoning about network events for detecting stealthy malware activities. *Comput Secur*, 58:180-198. https://doi.org/10.1016/j.cose.2016.01.002

Zhang M, Duan Y, Yin H, et al., 2014. Semantics-aware Android malware classification using weighted contextual API dependency graphs. Proc ACM SIGSAC Conf on Computer and Communications Security, p.1105-1116. https://doi.org/10.1145/2660267.2660359

Zhao ZQ, Wang JF, Bai JR, 2014. Malware detection method based on the control-flow construct feature of software. *IET Inform Secur*, 8(1):18-24. https://doi.org/10.1049/iet-ifs.2012.0289

Zhou YJ, Jiang XX, 2012. Dissecting Android malware: characterization and evolution. Proc IEEE Symp on Security and Privacy, p.95-109. https://doi.org/10.1109/SP.2012.16