Computers and Electrical Engineering 92 (2021) 107143

Contents lists available at ScienceDirect

Computers and Electrical Engineering

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

A comprehensive survey of tools and techniques mitigating computer and mobile malware attacks

S. Abijah Roseline, S. Geetha *

School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Vandalur - Kelambakkam Road, Chennai, TamilNadu-600127, India

ARTICLE INFO

Keywords:
Computer malware
Dynamic analysis
Machine learning
Malware datasets
Malware detection
Static analysis
Survey

ABSTRACT

In this era of modernization, digital technology plays a major role in all facets of life. We are accustomed to using computers and smartphones to access information, create, express, communicate, and collaborate, contributing to our personal, social, and professional survival at a comfortable level. As the legitimate use of digital space has grown, so have the prospects for malice via swindlers, blackmailers, vandals, and other criminals, to benefit by creating and propagating malware. While the main motive of malware designers is to make surplus profit illegally, other trivial motives range from activism and pranks, espionage, cyber theft, as well as serious crimes like privacy breaches. Policing the expanding attack surface needs the development of competent anti-malware solutions with more generalization to detect even zero-day malware and resolve their incidences with minimal human intervention. Anti-malware solutions pick up effective strategies from static, dynamic, reverse engineering, and advanced machine learning techniques. An arms-race situation always exists between the malware authors and the anti-malware community. In this paper, we present a comprehensive investigation of computer-based and mobile-based malware, their countermeasures, and various detection methods. We also discuss the additional issues and the challenges of malware detection and finally highlight a few open research issues, directing the trends of malware/anti-malware development.

1. Introduction

The software is defined as a collection of programs, which constitute a set of instructions to perform a specific task. Some examples of software programs are operating systems (OS), utility programs, device drivers, word processors, spreadsheets, networking, database management systems, etc. Akin to useful category programs, there exists another category of programs created with malicious intent to target computing systems. These programs which aim to violate a computer system's security policy in terms of confidentiality, integrity, and availability of data are called malware or bad programs. Malware is a serious security threat to critical applications like education, communication, hospitals, banking, etc. There has been a proliferation in the variety and exoticism of malware behavior in recent years. Further, the associated intensity of damage caused on users' data, time, and productivity has increased significantly. It has culminated to a level where a malware attack can completely paralyze an entire system and destroy the

This paper is for regular issues of CAEE. Reviews processed and recommended for publication by Area Editor Dr. G. Martinez Perez.
* Corresponding author.
E-mail addresses: abijahroseline.s2017@vitstudent.ac.in (S. Abijah Roseline), geetha.s@vit.ac.in (S. Geetha).

https://doi.org/10.1016/j.compeleceng.2021.107143
Received 4 October 2020; Received in revised form 30 March 2021; Accepted 1 April 2021
Available online 18 April 2021

data and the hardware infrastructure. Fig. 1 shows the malware threat levels on various Operating Systems (OS) in the year 2020. Threat level indicates the percentage of malware infections on different operating systems. As observed from Fig. 1, Windows OS demonstrates the highest malware infections in comparison to other operating systems.

Attackers continually change their malware to enhance complexity and prevent detection as much as possible. They write new malware, which is termed advanced malware, owing to their capability to change their forms and disguise themselves to obfuscate or fool malware analysts. This mutant malware is coined polymorphic malware. Metamorphic viruses are insidious, and they do not require a decryption routine. They vary the body of the code directly by carrying a morpher (code for morphing transformations) within itself to generate malware variants. Common morphing techniques used by malware authors include dead code insertion, variable renaming, statement reordering, etc. Most of the metamorphic or polymorphic malware exhibit complex and mutating characteristics that make it harder to detect.

The new malware can detect and evade malware detection systems and mask malicious features while operating in a sandbox. They tend to use pre-defined malware for identifying and preventing virtual machines and code analyzers from gathering details about the user's systems. Evasion techniques [2] fall into three broad categories, namely, (i) anti-security techniques; (ii) anti sandbox techniques; and (iii) anti-analyst techniques. Anti-security techniques are applied to escape detection from antivirus software, firewalls, etc. Anti-sandbox techniques by-pass inspection of monitoring tools that report the behavior of malware. Malware authors learn the design flaws of artifacts such as registry keys, specific files, processes, etc. of virtual environments. They write intelligent code in such a way that it disrupts the actual execution flow.

Anti-analyst techniques are employed to protect malware from being analyzed. Malware analysis diagnoses whether malware runs in a virtualized or sandbox environment, is monitored by any tool such as process monitor, Wireshark, etc., or whether it is being debugged. The usage level of counter malware detection techniques by malware attackers in 2020 is shown in Fig. 2. Anti-sandbox techniques are used more frequently compared to other anti-detection techniques by malware attackers. By inserting these techniques into the malicious code, they remain hidden and persistent on the target. Counter malware detection techniques have established an endless arms race between malware authors and malware detection techniques.

The contribution of this paper is as follows.

- This paper provides an exhaustive and comprehensive survey of malware attacks and techniques to combat them. In the field of Malware Detection Systems, there have been numerous surveys related to machine learning, cloud, IoT, and a few surveys on Man in the Middle, Stealth malware, and rootkits. However, none of the existing work addresses the complete bird's eye view of malware and mitigation techniques.
- This paper presents an overview of the malware mitigation techniques. Further, it differentiates Static Analysis (SA) and Dynamic Analysis (DA) from machine learning-based detection approaches. A detailed taxonomy of the malware mitigation techniques is presented and supported with suitable diagrammatic illustrations.
- This paper includes the performance evaluation of different malware detection approaches to provide a better understanding for malware researchers to focus on developing smart and efficient malware detection systems.
- Finally, it is realized that the observations tendered in this paper will (a) help researchers become more familiar with the malware types, attack model, attack techniques, propagation mechanism, persistent methods, and intelligence (evasion methods) incorporated in combating malware, (b) give a clear insight about the possible research directions under real-life conditions, and finally, (c) provide directed design guidelines in machine learning techniques specifically for malware detection.

The terms specific to malware, which are used in this paper, and their explanations are given in Table 1.

The paper is organized as follows: Section 2 presents an overview of various types of malware. Section 3 discusses the prevalence of malware in Windows OS, the general structure of the detection model, and tools employed in malware detection. Section 4 summarizes the taxonomy of malware detection and a very detailed discussion of each method. Section 5 presents the possible research directions

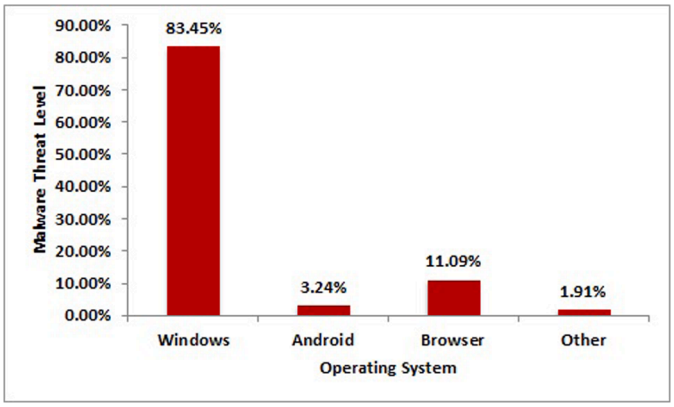

Fig. 1. Malware threat level on various OS in the year 2020 [1].

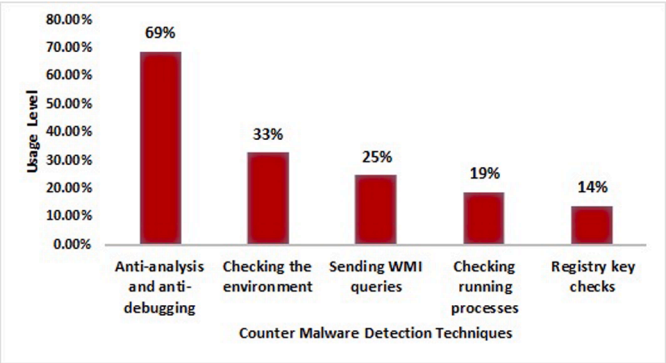

Fig. 2. Counter malware detection techniques by attackers in 2020 [2].

Table 1
Terms used in this paper.

S. No	Terms	Explanations
1	Malware	A program that is harmful to a computer or mobile user and performs malicious activities such as stealing, encrypting or removing sensitive data, monitoring users' activity without their permission, etc.
2	Stealthy Malware	Secret or hidden malware
3	Metamorphic Malware	Malware that is capable of changing its code and signature patterns with each iteration
4	Polymorphic Malware	Malware that constantly changes its identifiable features in order to evade detection
5	Detection Methods	Malware identification techniques
6	Code Obfuscation	A method of performing mutations such as variable renaming, code reordering, etc. to create malware variants, evading detection by malware detectors
7	Evasion	The act of avoiding detection by detectors
8	Packers	Software that unpacks itself in memory when the packed file is executed
9	Reverse Engineering	A technique which analyzes the malware code to understand the aim and functionality of malware
10	Sandbox	A secure and virtual environment in which malware code is executed and tested
11	System Calls	A program that provides an interface between a process and operating system, requesting services from the kernel of the operating system
12	Semantics	Meaning of various elements of a program code
13	Machine Learning	The ability of systems to automatically learn
14	Malware Authors	Malware writers
15	Static Analysis	Analysis of a binary without executing it
16	Dynamic Analysis	Observing the behavior of the malware during its execution
17	False Positives	Labeling clean samples as malicious

in this domain. Lastly, Section 6 concludes this study.

2. Overview of malware

Malware is a general term that describes several types of malicious programs such as viruses, worms, rootkits, Trojan horses, backdoors, botnets, spyware, adware, ransomware, etc. Variants of malware are classified based on certain unique characteristics like propagation methods, infection types, etc. A brief outline of the definition, impact, spreading strategies, and examples of different types of malware are presented in Table 2.

Cybercriminals constantly develop new distribution techniques to penetrate a victim's system using malware. Malware spreads across host systems through direct or indirect user actions. Attackers trick innocent users by allowing them to download and run malware on their systems, without their knowledge. Malware damages a computer's boot sector, files, installed software, and BIOS, thereby tainting users' data and files. Some of the commonly used distribution modes of malware are discussed below.

2.1. E-mail

Cybercriminals tend to propagate malware by incorporating malicious attachments and links in emails. Phishing e-mails appear to be legitimate, since the source is disguised as friends, esteemed organizations, or other reliable sources that users trust. When users download the attachments or click on the links, the malware is infested on their computers, causing serious security risks. Some malware on e-mails infects a user's system when a user merely attempts to preview the e-mail, without downloading or clicking links.

Table 2
Types of malware.

S. No	Malware Type	Description	Propagation Methods	Impact	Examples
1	Virus	• an unwelcome computer program that can infect legitimate host programs • attaches to a useful application or making copies of itself • damages the host and the integrity of information • require user action to activate propagation	• Removable media • Internet downloads • E-mail attachments	• Periodical illegitimate message alerts • Slow startup • Degrades computer performance, formats disks • Damages files or even crashes systems	Elk Cloner, Brain boot sector virus, etc.
2	Worm	• self-replicating program • spreads from one device to another using different vectors such as Universal Serial Bus (USB) devices or E-mail [68] • spreads on their own over a computer network • Does not need any user action to activate them	• Removable media like USB • E-mail attachments	• Exploits vulnerabilities in operating systems or installed programs • Causes network performance issues • Utilizes large amount of systems' memory resources	Creeper, Morris worm, Michelangelo, SQL Slammer worm, Duqu worm, Internet worm, Xerox worm, Stuxnet
3	Rootkit	• Takes privileges (root-level access) of a system administrator • Characterize undetectability by being persistent and stealthy • The attacker installs a rootkit and conceals its activities from the detection systems	• Malicious file that appears benign • Downloadable plug-in • E-mail attachment • Infected mobile apps	Gains control over the device's functions and other software, including security software	Knark, Rkit Cloaker, VGA rootkit, SubVert, Blue Pill, Rovnix, Stoned Bootkit, Vanquish, Aphex, and Hacker Defender
4	Trojan Horse	• Acts as a legitimate program and performs malicious actions in the background disguise of a useful program • Downloaded automatically • Spreads to other devices that are connected to the same network		• Acquires confidential information • Modifies or deletes files • Monitors user activity • Disrupts normal functionality • Drops payloads such as file, registry, network, etc. • Scans networks for vulnerabilities	Trojan-Banker, Trojan-Downloader, Trojan-DDoS, Trojan-Dropper, etc.
5	Backdoor	• Performs unauthorized remote access to a computer system • Logs user activity and tracks web browsing behaviors	• E-mail attachments • File-sharing programs or infection of remote systems	• Spies and gains access control over user activities and their system • Steals credentials of victim	Netcat, Finspy, KeyBoy
6	Spyware	• Remote monitoring software that monitors user's activities • Keeps track of personal and confidential details of the user without their knowledge • The gathered sensitive information is sent back to the attacker or others to perform malicious activities	Through free software downloads	Changing browser settings	Caveat, CoolWebSearch, HuntBar, Cydoor, 180SearchAssistant, etc.
7	Botnet	• Conquers and infects a computer and becomes a node on the bot network • Later, it spreads and infects thousands of computers remotely across the network		Spreads viruses and worms sends spam e-mails, allows Denial of Service attacks on websites, drive-by downloads, etc	Agobot, Mirai, Conficker, Zeus, Waledac, Mariposa, Kelihos, Kaiten, Hybrid_V.1.0, etc
8	Adware	Automatically displays or downloads unwanted advertisements when the user executes another program	Free games, peer-to-peer clients, etc	• Keeps track of confidential information of users • Takes control of browser activities when it is installed into the user's system	Plankton, Fireball, Appearch, DollarRevenue, Gator, DeskAd, etc
9	Ransomware	• Originally known as cryptoviral extortion • Insidious malware that works with three tasks, such as reading the original data, and then encryption is done, and original data is wiped off		Encrypts an individual's data or organizational data and threatens to deny access to demanding a ransom	WannaCry, Cryptolocker, Locker, Bad Rabbit, Goldeneye, Zcryptor, Petya, GandCrab, SamSam, etc
10	FakeAV				

(*continued on next page*)

Table 2 (*continued*)

S. No	Malware Type	Description	Propagation Methods	Impact	Examples
		Imitates security product like an antivirus, and display a GUI window which scans the victims' system for malware		Reports several fake infections and attempts to scare victims into buying the full version of the fake software to clean the machine	

2.2. Drive-by download

Malware on web pages could infect users and their computers when they browse the Internet. A potential victim is one who is connected to the Internet and performs browsing activities. Users download malware infected-pirated software disguised as a legitimate one.

2.3. Local area networks (LANs)

LANs are a network of locally-connected computers, where one machine that gets infected with malware propagates it across other machines in the network.

2.4. Social networks

Malware authors use many popular social networks as attack vectors. A user's account is infected with malicious programs, which, when visited by other third-party profiles, get infected and propagates onto other users' systems.

2.5. Storage/ removable media

Storage media like USBs and CDs are used as attack vectors that can be inserted into a victim's computer system. It is good practice to scan any external drive before copying files to or from them. However, some malware stays hidden and is undetectable by anti-malware scanners. The Autorun component of Windows is the main infection vector for this removable drive infection worm.

Once malware is loaded into the system, its main aim would be to evade detection, secretly launch, be persistent (i.e., continue to run regardless of system restart), and carry out its core functionality like spreading itself to other machines. The malware works by dropping payloads, which can be based on file or registry, or some other payload to perform these activities.

The behavior of malware is studied by first identifying its payloads. Any system resource that malware uses in an unusual way is called a payload, and this payload causes overhead on the system. Malware payloads include file payload, registry payload, network payload, directory payload, and other miscellaneous payloads. *File payload* refers to the act of writing a file on the hard disk (e.g.) a copy of the malware in a different location. *Registry payload* involves creating a registry entry (e.g.) an autostart entry.

Network payload refers to accessing a website (e.g.) to download more malware.

Directory payload involves creating folders (e.g.) repositories for configuration data. Other *miscellaneous payloads* create, modify, corrupt, or access other crucial information.

3. Computer-based malware detection

3.1. General setup for computer-based detection

A malware analysis lab is essential to understand and gain a deeper insights into any malware, and to develop methods to reduce and control the menace of its infection in the future. The prerequisite for setting up a lab is to set up a network to control and analyze what enters the system and what leaves. Furthermore, the lab systems are isolated from other connected systems in a network. Virtualization is an important step as it allows an analyst to host everything on a single machine. The execution of any application in the VM cannot affect the host OS. A virtual machine (VM) is a safe environment where malware is allowed to run without infecting the host. The system state prior and successive to the execution of the malware is traced. Snapshots are taken to revert to the safe uninfected state once the VM is infected with malware.

Malware analysis steps include, (i) install and configure malware analysis platforms to carry out static or dynamic malware analysis (ii) install analysis tools (iii) set up the network configuration (iv) save a snapshot in a clean state. Before performing analysis, it is necessary to test the virtual environment for any connection of the analysis machine with outside networks.

3.1.1. Monitoring tools

Malware analysis tools generate analysis reports, which provide detailed insights for a security analyst to understand the nuances of runtime behaviors and actions of samples. A security analyst devises detection techniques based on the knowledge gained from dynamic malware analysis, and the exhaustive process is explained later. The tools for performing dynamic analysis are used to monitor

file changes on the system, as well as registry changes, process activities, web access history, etc.

The following are the malware analysis tools that are used to analyze or monitor malware samples.

Activity Loggers: Regshot and InCtrl are some of the activity-logger tools that log the changes in the system between two timelines (e.g.) before and after malware has been executed. The major limitation of this method is that if a particular change is done and undone between the snapshots, it would not be logged.

SysInternals Tools: RegMon, FileMon, ProcMon, and NetMon were previously distributed as separate tools to monitor malware. Later, these tools were all integrated into a single tool as ProcMon. ProcMon continually monitors the changes in the registry, file system, network, and process activities, and logs them. The advantage of ProcMon-like tools is that every activity (done and undone) is logged as separate events and recorded for future reference.

3.1.2. Windows portable executable (PE) file format

Windows executable files conform to the PE file format and are represented by EXEs, DLLs, SYS, etc. The Windows Loader parses the executable file's header structure and metadata fields to load them into Windows memory at runtime. Since most computer-based malware samples are in the form of Windows executables, malware analysts are required to understand the PE file format for performing static analysis. Fig. 3 shows the general layout of the PE File Header. The PE Header is the beginning structure of a PE file containing metadata, which gives specific information about the file at runtime. The metadata information may include the location and size of various data contained in the file, and what to be loaded in memory, with what attributes, etc.

3.2. Generic model of computer-based malware detection

The Windows PE file is given to the malware detection system to check whether it is malware or a benign file. The disassembler or debugger analyzes the file statically or dynamically and captures its behavioral characteristics. The malware, if obfuscated or packed, is subjected to deobfuscation/unpacking, to get the raw malware sample. Then, various features such as binary strings, API calls, etc. are extracted. Machine learning techniques are employed to classify the samples as malware or benign, based on the feature representations. The malware database is regularly updated with the new and modified inspected malware signatures. The generic procedure for the computer-based malware classification process is shown in Fig. 4.

3.3. Taxonomy of computer-based malware detection methods

The rise of new and zero-day malware has motivated researchers to focus on measures to identify and mitigate the same. Malware analysis is a crucial step in developing effective malware detectors. Malware analysts exert a broad set of malware analysis and defense techniques to discover a clear picture of malware. The computer-based malware analysis and detection approaches are broadly classified as static analysis, dynamic analysis, and hybrid analysis. The taxonomy of methods used for computer-based malware analysis is shown in Fig. 5.

3.3.1. Static analysis

Static analysis allows an analyst to examine static properties such as strings, hashes, signatures, metadata, of a suspicious file. Most antivirus software employs a signature-based approach to discriminate malicious and benign files. These methods operate with a list of

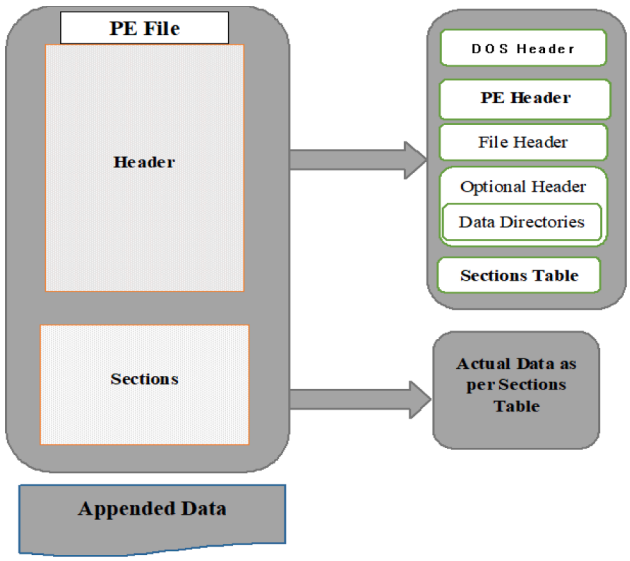

Fig. 3. General layout of PE file header.

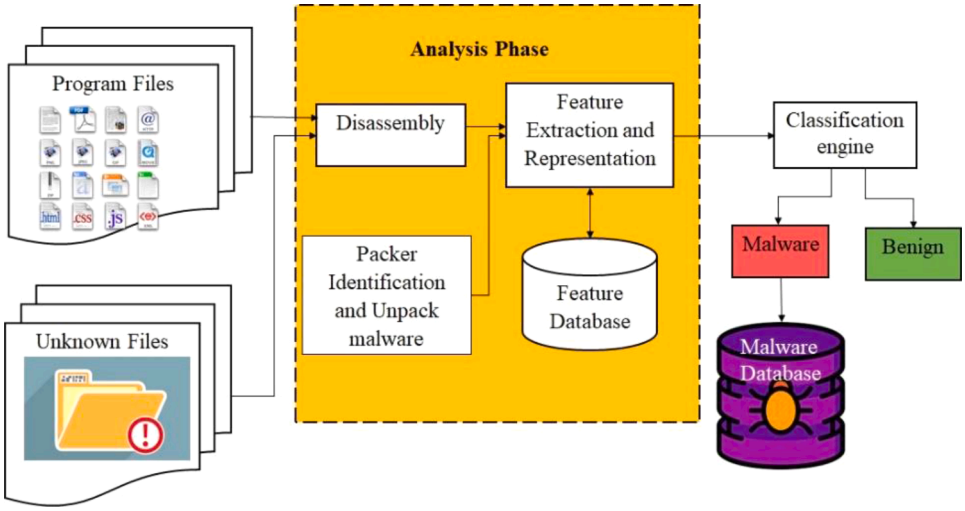

Fig. 4. Generic model of computer-based malware detection.

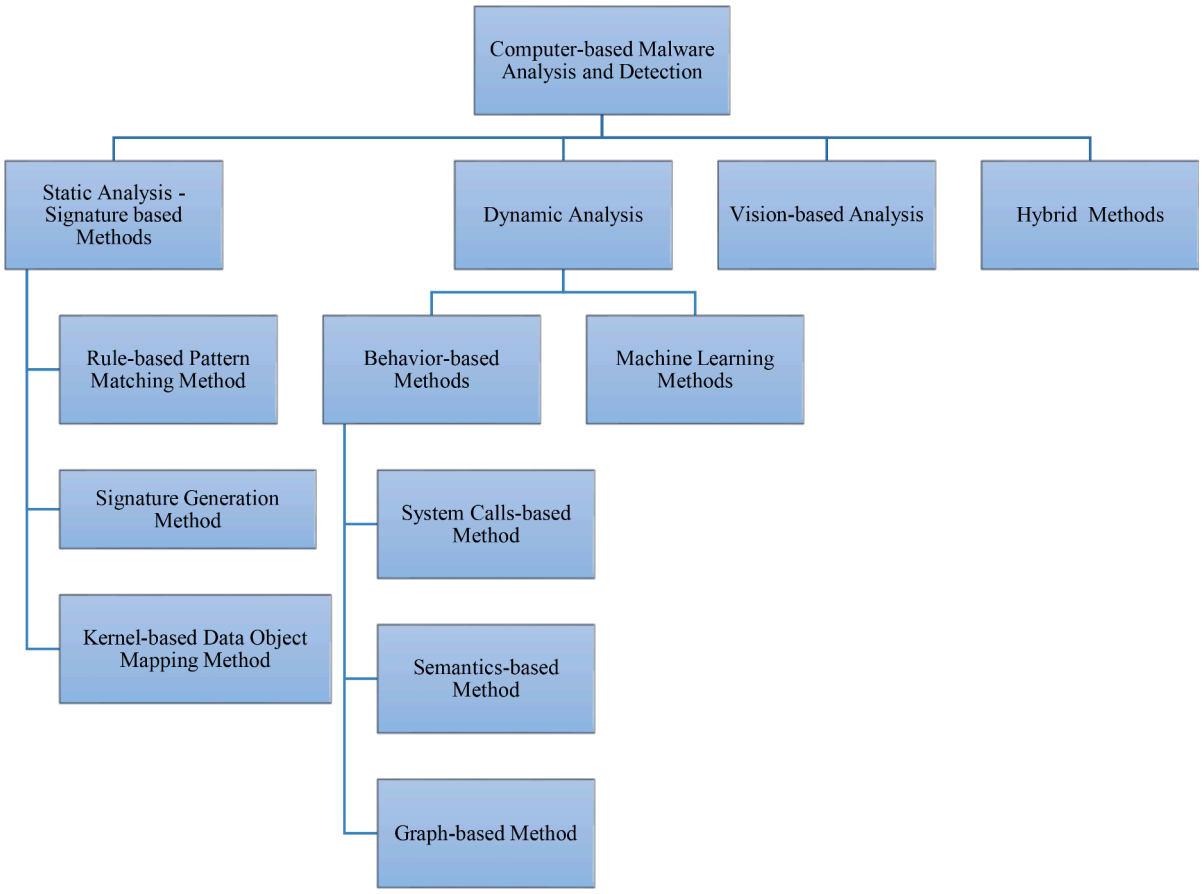

Fig. 5. Taxonomy of computer-based malware detection methods.

pre-defined virus signatures, commonly called a signature database. A signature is a series of bytes, which uniquely identifies a particular malware. If the input program file matches a signature in the database, the program is marked as malicious. Signature detection methods are widely preferred in the computer domain since they are simple, easy-to-use, low computational complexity, and low false-positive rate. However, they suffer from the drawback that they are only capable of detecting known malware attacks.

Further, they lack robustness against evasion techniques. They involve a time-consuming, manual string signature creation process due to which they are less preferred. However, current researches aim to develop automatic signature detection systems, replacing the traditional manual signature generation. Signature-based detection methods are classified into rule-based pattern matching methods, automatic signature generation method and kernel-based data object mapping method.

a) *Rule-based Pattern Matching Method:* Pattern matching involves detecting security breaches by monitoring unusual patterns of system usage from the system's audit records. Denning [3] presented an Intrusion Detection Expert System (IDES) model that works on the principles of Rule-based Pattern Matching. The model consists of profiles for representing the legitimate behaviors and rules for understanding the anomalous behavior from audit records. IDES detected a wide range of intrusions such as viruses, trojan horses, masquerading, penetrations, covert channels, etc. However, the model failed to observe the defects in the target system and the action that exploited the respective defect. The actual penetration detection mechanism requires monitoring all supervisor calls, which is impractical. They inferred that some key features, such as soundness, completeness, timeliness, metrics, statistical models, and profiles, are needed to make the system more robust.
b) *Signature Generation Methods:* Signatures have to be adequately long and unique. They should not allow the misclassification of benign files as malware. Signatures are used to detect malware in real-time network security systems. The significant challenge in high-speed networking is that a byte-string signature may split into two or more packets. Intelligent signatures that comply with the drawbacks of high-speed networking detection systems are used to achieve high accuracy systems.

Function-based Method: Functions are good signature candidates, which are used to mark the start and endpoints. The generation of simple and byte-string signatures could be used to further detect and remove malware at the network level. The function-based signature is compared against the Common Function Library (CFL). The signature extraction process first identifies and discards the Common Function Code (CFC) by analyzing the malware based on the CFL. The false-positive errors are minimized by removing functions from the signature list in the CFL. Shabtai et al. [4] proposed an automatic and function-based signature generation technique called F-Sign. The intelligent candidate selection strategy generated signatures based on the entropy score. F-Sign tackles allergy attacks which are attacks against Automatic Signature Generation systems. State Machine (SM) and Interactive Dis-Assembler (IDA-Pro) methods were used for function extraction and their false-positive rates were compared for different CFL sizes. When a large CFL is used, the false-positive rate is low.

a) *Kernel-based Data Object Mapping Method:* Signatures can be generated based on malware's unique data access behavior and common attack patterns on the kernel. Rhee et al. [5] proposed a malware characterization system called DataGene, which exploits a data-based OS kernel. The malware was detected and characterized using the properties of data objects of the attacks. Their system is comprised of two system components. The first component comprised of the runtime kernel object mapping system that enabled an external monitor to identify the access behavior of data objects. The second component described the malware characterization approach that generated a malware signature using unique data object access schemes.

3.3.2. Dynamic analysis

Dynamic analysis techniques involve the execution of malware samples and observing their registry, file system, process, and network activities in safe mode. The analyst typically interacts with the malicious program in order to effectively analyze its behavior. This analysis is carried out in an isolated virtual machine to avoid risk or damage to the host operating system. It records the typical features such as instruction sequences, API, and system calls for each executable. This analysis requires a considerable amount of computational power and memory. This method was originally developed for computer-based malware where the constraints on memory and computational resources are not as high as they are on mobile devices. Dynamic analysis includes behavior-based detection methods and machine learning-based detection methods.

a) Behavior-based Detection Methods: Malware authors employ various detection avoidance techniques like code obfuscation and packing to evade static analysis techniques. Therefore, in order to thoroughly understand the nature of the malware, we cannot rely on static analysis techniques alone. Behavior-based detection methods are a good solution to overcome such sophisticated, hard-to-detect malware. Every component of the binary is analyzed at run time irrespective of how obfuscated the executable is and the packer with which the code is packed. Many types of behavior-based malware detection methods were used in the literature.

System calls based method

Most malware detection systems are based on the dynamic behaviors of system calls. The anti-detection feature of modern malware like system-call injection attacks should be taken into consideration. Naval et al. [6] proposed a new detection approach that characterized the program behavior based on semantically relevant paths using the Asymptotic Equipartition Property of information theory. They analyzed the execution flow of malware code based on system calls and represented it as an Ordered System Call Graph (OSCG). They built a new feature set by quantifying the semantically-relevant paths using the Average Logarithmic Branching Factor (ALBF) metric. Though their method was robust against attacks based on system calls, it was susceptible to evasion. The performance was not affected despite the introduction of thousands of system calls into malicious programs. Their solution was effective and efficient in identifying real malware instances, but came with high computational overhead.

Graph-based method

Li et al. [7] presented a solution for detecting packed malware using a graph-based mining technique. Their method was based on consistent execution and obtaining CEG from disassembling code and semantic extraction. A similarity measure was constructed from the graph using the Weisfeiler-Lehman shortest-path kernel. The performance evaluations were done on large-scale datasets, which

Table 3
Experimental results of various methods for computer malware detection.

S. No	Authors	Methods	Performance Results				Datasets					
			Accuracy (%)	Precision	Recall	F1-score	Malware No. of Sample/ Dataset Size	No. of Malware Classes	Collection Period	Collection Source	Dataset URL	Benign No. of Samples/ Dataset Size
1	Rieck et al. [13]	Dynamic Analysis+ Machine Learning	96.18	0.9600	0.9598	0.9599	3133	24	3 years	CWSandbox website	http://www.mwanalysis.org Malheur dataset: http://pi1.informatik.unimannheim.de/malheur	
2	Mohaisen et al. [24]	Dynamic Analysis+ Machine Learning	96.22	0.9592	0.9642	0.9617	115,157	11	13 months (July 15, 2011)	Customer submissions, Internal submissions, and AV vendor samples		
3	Naval et al. [6]	Dynamic Analysis	94.62	0.9414	0.9460	0.9437	2435		2011–2014		https://virusshare.com/	1316
4	Kolter et al. [10]	Dynamic Analysis+ Machine Learning	97.56	0.9718	0.9750	0.9734	1651		2003	VxHeavens website	http: //vx.netlux.org	1971
5	Anderson et al. [27]	Static Analysis+ Machine Learning	91.34	0.9086	0.9128	0.9107	300 K (training)/ 200 K (test)	3	2017–2018	VirusTotal	https://pubdata.endgame.com/ember/ember_dataset_2017_2.tar.bz2 https://pubdata.endgame.com/ember/ember_dataset_2018_2.tar.bz2	300K
6	Nataraj et al. [11]	Vision-based Analysis+ Machine Learning	97.18	0.9657	0.9685	0.9671	9339	25	2011	Malimg dataset	https://www.dropbox.com/s/ep8qjakfwh1rzk4/malimg_dataset.zip?dl=0	–
7	Roseline et al. [28]	Vision-based Analysis+ Machine Learning	98.65 97.2 97.43	0.9886 0.9761 0.9753	0.9863 0.9679 0.9732	0.9874 0.9720 0.9742	9339 10,868/0.5TB 8750	25 9 25	2011 2015 2019	Malimg dataset Kaggle Microsoft BIG2015 dataset MaleVis dataset	https://web.cs.hacettepe.edu.tr/~selman/malevis/	350

included manually packed benign apps, wild packed, and unpacked malware.

Semantics-based method

The semantics-based approach incorporated the semantics of the program trace to characterize the behavior of malware and the suspicious program. The abstract interpretation eliminates irrelevant behavior features. Preda et al. [8] presented a formal definition and a semantics-based malware detection framework. Their system experimented with obfuscation methods such as code reordering, semantic-nop insertion, the substitution of equivalent commands, variable renaming, etc. Their system exhibited robustness and completeness in detecting malware with obfuscations.

b) Machine learning-based detection methods: due to the challenges posed by the rise of stealthy and obfuscated malware, traditional signature-based detection is relatively less preferred. To tackle this, solutions based on flexible detection algorithms picked from machine learning have been meticulously designed and developed through recent research and study. Santos et al. [9] studied that operational codes better discriminate malware from benign samples and unique opcodes prove to be better predictors than typical opcodes. Their method determined the relevance of individual opcodes and computed the weight for each opcode. The weighting represented the frequency of occurrences of malware or benign samples. The similarity among two input files was computed to recognize malware variants and machine learning classifiers were trained to identify unknown malware. The processing overhead increases with the length of the opcode sequences. Their method is static, and hence cannot handle packed malware.

Each trace can be encoded as a training sample based on n-gram features, and the most relevant n-grams could be selected for prediction. J. Kolter et al. [10] proposed a Malicious Executable Classification System (MECS), based on machine learning models for the classification of unknown malware in the wild. The evaluation was based on inductive methods such as Naive Bayes, decision trees, support vector machines, and boosting. The performance of the methods was assessed using the payload's function such as opening a backdoor and mass mailing. The boosted J48 decision trees performed well and are scalable to a larger collection of executables. They suggested that performance could be improved by removing obfuscation and with additional training examples.

3.3.3. Vision-based analysis

Nataraj et al. [11] proposed a novel approach of analyzing malware executables using vision-based analysis [28,29] and considering global features. Their method converted malware into grayscale images, which helped to visualize the similarity among the malware classes in terms of layout and texture. The vision-based analysis does not require disassembly or running the samples in safe environments. Their method was resilient to obfuscation techniques like section encryption. The limitations of their approach include ineffectiveness to adversarial attacks like section relocations and unnecessary redundant data additions. Cui et al. [12] converted malware samples into grayscale images and used a CNN to classify them. The data imbalance problem was addressed using a bat algorithm-based data equilibrium method. Their approach was limited to fixed-size input images.

3.3.4. Hybrid methods

This type of approach includes any combination of static, dynamic, or vision-based methods. A combination of vision-based with dynamic (machine learning and deep learning) approaches was presented for developing intelligent malware detectors. A substantial set of typical features is created by combining signature-based and behavior-based techniques.

The clustering of behavior is the grouping of new malware classes with similar behavior, and the unknown malware is grouped into these clustered classes, which are known as the classification of behavior. Rieck et al. [13] proposed a new framework for analyzing malware behavior based on scalable clustering and classification. They used prototypes for clustering and classification of new malware variants, allowing for run-time improvements with a minimal approximation error. They introduced a combined incremental approach for behavior-based analysis using clustering and classification. Their approach showed reduced runtime and significantly lower memory overhead compared to other analysis methods. An efficient automatic malware behavior analysis is achieved by combining their learning-based model with other detection techniques.

3.4. Computer malware datasets

Machine learning techniques can be used to build classification tools to detect unknown malware by depending on datasets containing various distinct features of malware and clean samples. Labeled samples are known samples, whereas unlabeled samples are unknown samples. Datasets of labeled samples belonging to the same class are known as partially labeled datasets. Multiple sources of malware samples were collected from the CWSandbox website, Sunbelt Software, customer submissions, internal submissions, AV vendor samples, VxHeavens website, internet (browser, downloader, etc.), accessibility (onscreen keyboard), drive-by-download servers, VirusShare website, Malware blacklist, etc. The benign samples were collected from personal computers, VirusTotal web portal, system programs, shared libraries, McAfee Labs, manual collection from various versions of Windows, Microsoft. Some datasets were manually inspected and labeled. The samples were randomly selected from a larger population of malware samples using any strategy such as a priority queuing strategy.

The experimental results of various methods on several datasets for computer-based malware detection are given in Table 3. The accuracy, precision, recall, and f1-score are used as performance metrics to analyze the efficiency of some malware detection methods used in the literature. The details of various datasets used including the number of classes and samples used for evaluation are also provided.

4. Mobile-based malware detection

Mobile devices are a vital part of people's lives owing to the widespread convenience of on-demand Internet connecting anywhere and anytime around the globe. The features and benefits of personal computers have become handy through smartphones. The operating systems on mobile devices run applications and provide advanced functions. They have become more user-friendly by providing third-party applications like games, fitness, monitoring, healthcare, etc. Wireless providers have started providing more advanced mobile phones with data plans. Android OS for mobile phones is popular worldwide and dominates other mobile operating systems due to its remarkable features including but not limited to its open-source nature, cost-effectiveness, and support to third-party markets and applications. The diverse usage of smartphones has led to the origin of malicious applications which have penetrated open marketplaces. These applications are contaminated by malicious Android applications, which disguise themselves as benign applications or bundled with benign applications. Malware applications stealthily perform operations to acquire a user's privilege or misuse system resources with the aim of monetary benefits. Some of the capabilities of malware include controlling or disabling the device remotely or locally, tracking user activities, stealing users' sensitive data stored on the device. Trojans, Ransomware, Adware, Rootkits are the common types of mobile malware threatening mobile users.

Malware propagates on mobile devices through clicking a malicious link in an e-mail or Short Message Service (SMS), downloading infected apps from Google or third-party PlayStore, or setting up internet connections with malicious WiFi networks. At present, the most severe vulnerability addressed is a patch for the Android runtime that could let an attacker gain access to certain OS features without user interaction.

4.1. Android security model

The important security features integral to Android security are (i) Permissions (ii) Application Isolation (iii) Dalvik/Android Run Time (ART) (iv) Signing (v) Verify Apps.

4.1.1. Permissions

This feature is also known as capabilities or access permissions. As the name indicates, permissions are the access levels, which an Android Application Package (APK) would need to be granted to perform its operations. Permissions are requested from the user during the install time of an application. Permissions requested from the user are defined in a file called AndroidManifest.xml file inside the application package.

4.1.2. Application isolation

Android isolates applications installed on the device from each other at the system level by assigning each application, and its dedicated process, a unique user identity that is visible only to the system. Whenever an installed application or one of its components is called to perform an action, the Linux kernel identifies the target application by its assigned user ID and starts the corresponding process in its separate sandbox. No two processes can generally run with the same user ID, (i.e.) within the same process space. This protects an application from intrusion by another application at the same privilege level.

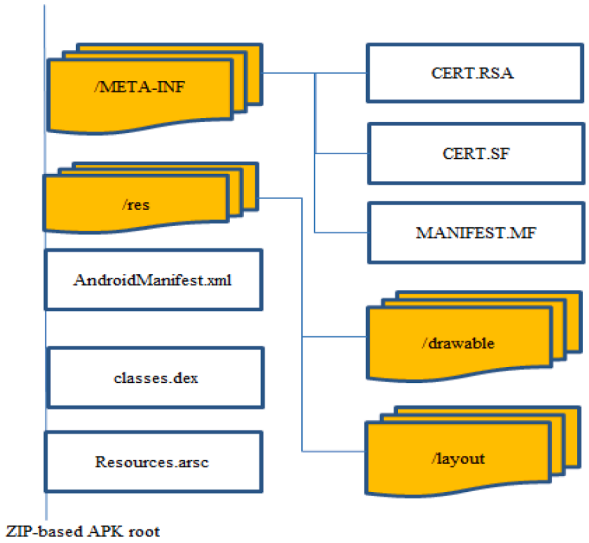

Fig. 6. APK file structure.

4.1.3. Dalvik VM/ART

In earlier versions of the Android OS, each isolated application ran inside a virtual environment (VM), called Dalvik VM, which was later enhanced to a VM technology called ART. The host OS is protected from applications, and applications are protected from each other.

4.1.4. Signing

Each APK is required to be signed either by a certifying authority or by the author. Package signing confirms that the source of an application has not been modified since it was signed. Certificates uniquely identify the application's author. The policy of allowing self-signed APKs to load opens the door and creates vulnerabilities for attacks to exploit since malware authors are not vetted and a certificate is generated and used for free. Hence, Android malware is self-signed.

4.1.5. Verify apps

This feature monitors applications during installation for any malicious activity. It warns the user if it is a suspicious application. The user is still allowed to install the application at his own risk. This feature also enables Google to detect malware proactively.

4.2. APK file structure

APK is the container file format for executable applications on the Android mobile operating system. An APK is a Zip file with specific contents, each meant for different purposes. The significant components of an APK file include classes.dex, AndroidManifest.xml, Resources, and META-INF. Fig. 6 shows the APK file structure.

4.2.1. Classes.dex

Classes.dex is the executable code component of an APK. The d*ex* is derived from "Dalvik executable". Like PE files for Windows, dex files also have defined headers, metadata, and executable code. Classes.dex is a single binary composed of a group of Java class files defined for the APK. The class files are compiled to a proprietary bytecode, interpreted, and executed within a Virtual Machine. Android VMs are different from the classical Java Virtual Machine (JVM). While JVM uses pure Java class files transformed to JVM bytecode, the bytecode running in Android VMs (Dalvik/ART) is transformed from traditional JVM bytecode to the dex format by translating and collating Java class files with the conversion tool *dx*.

4.2.2. AndroidManifest.xml

APKParser.jar is a Java tool that scans through an APK and parses AndroidManifest.xml in that APK for understanding the contents of AndroidManifest.xml. APKParser.jar can be exected as follows,

```
C:\Program Files\Java\jre7\bin>Java –jar APKParser.jar
<APK_to_be_parsed>
```

Each Android application contains a manifest file that carries the following information such as package name and class names used in the application (indicate potential app functionality at a glance).

- API level and libraries required for the application to run (the API level defines the minimum compatibility level of the Android OS version required by the application).
- Application Component information
  - *Activities* define the activities for each of the classes defined.
  - *Services* declare the processes related to the application.
  - *Intents Filters* specify the type of intents that the component would like to receive.
- Permissions required by the application and granted by the user at install time, for the app's proper execution.

4.2.3. META-INF

META-INF is the security meta-data directory for an Android application. This folder includes the following files:

a) *MANIFEST.MF*

This file contains the list of filenames and their hashes. It validates the integrity of the file content of the APK.

a) *CERT.SF*

This contains the hash of MANIFEST.MF and list of filenames and hashes of their text entries in MANIFEST.MF. It validates the integrity of MANIFEST.MF.

a) *CERT.RSA*

This contains the x.509 certificate containing the public key and signed blob of CERT.SF file. It effectively validates the integrity of overall file content inside APK. It contains the signer name and other signer-related metadata.

The SHA1-Digest (hash) information for various files in an APK will be verified by the operating system at the application installation time. APK integrity is validated only at the package content level. Android does not currently provide a mechanism to validate the top-level APK as a whole, (i.e.) inclusive of the Zip structure and metadata. This leaves the APK's top-level Zip structure vulnerable to manipulations, which can break the Android Security Model. The generic procedure for the mobile-based malware classification process is shown in Fig. 7.

4.3. Taxonomy of mobile-based malware detection methods

The taxonomy of mobile-based malware detection methods is depicted in Fig. 8. Mobile-based (Android) malware analysis and detection are broadly classified as static, dynamic, vision-based, and hybrid analysis.

4.3.1. Static analysis

The signature-based method is well suited for mobile malware detection due to its high scanning speed and low memory usage.

a) *Hash-based Signature Matching Generation Method:* Venugopal et al. [14] proposed a hash-based signature matching method by viewing it as a multi-pattern matching problem. Memory consumption was found to be very high as the number and length of the signatures grow. To reduce memory consumption, the hash value of the signature is stored instead of the signature itself. A hash table is used in this algorithm to store the hash values of signatures. The hash table also increases the scanning speed in detecting mobile malware. The computation of the index is done efficiently by computing a filter-hash which is used to acquire the index in the hash table. Large hash table and memory access impacted the computational time. Various experiments were conducted for different hash table sizes for determining the size with minimal computational time. New mobile malware were not detected due to memory and power considerations.

b) Semantic-based Signature Method: Alam et al. [15] proposed DroidNative for detecting both bytecode and native code Android malware. The system used specific control flow patterns to overcome obfuscations and provided automation. The cross-platform semantic-based signatures were built at the native code level for Android. Techniques such as packing and applications that download malware cannot be properly analyzed by the system. Their system could not detect unknown, zero-day malware. The system detected known malware variants and identified zero-day malware if its control structure matched an existing malware trace in the training database. It did not detect malware that changed the control flow of code based on a threshold value.

c) Permission-based Method: The frequently used permissions by malicious applications and the significant permission features are analyzed through this method. Jiang et al. [16] proposed a static Fine-grained Dangerous Permission (FDP) approach by collecting features that best discriminate benign and malicious applications for Android malware detection. Their work efficiently detected malware families and required less time for analyzing applications. Their work does not extract more information if the application is obfuscated using techniques such as dynamic loading, reflective mechanism, and encryption. Their approach is ineffective, as many applications do not allow decompilation.

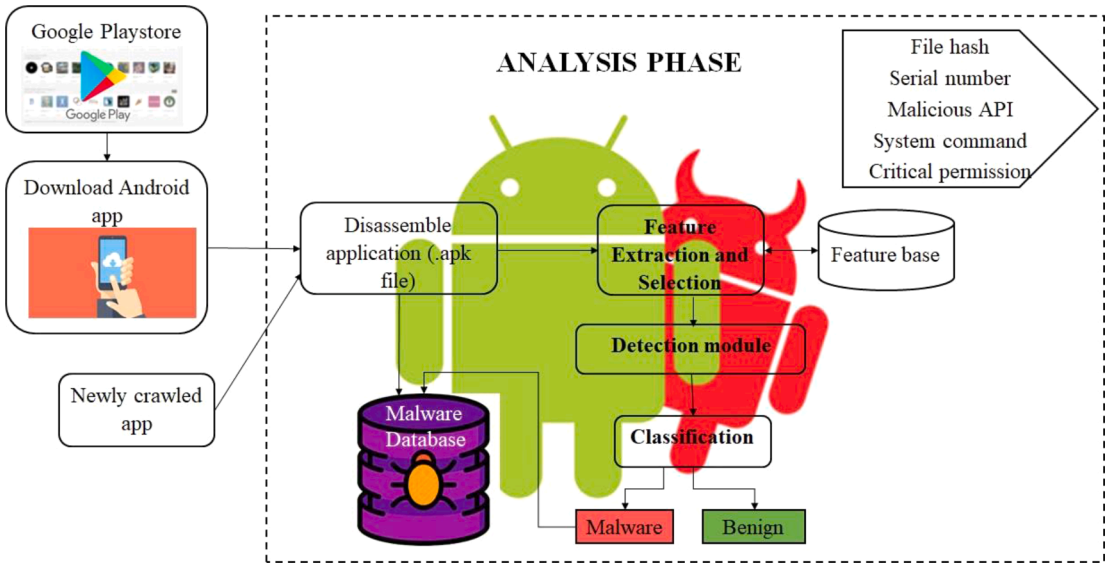

Fig. 7. Generic model of mobile-based malware detection.

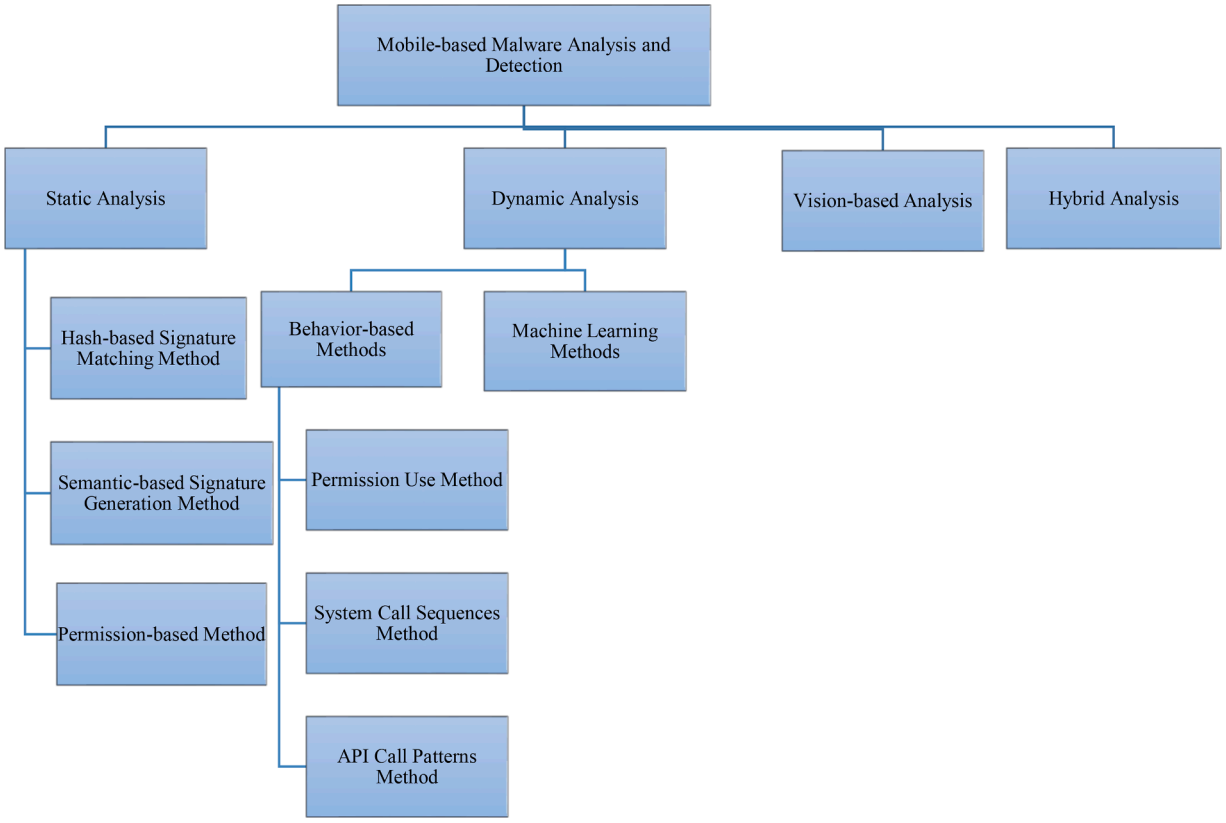

Fig. 8. Taxonomy of mobile-based malware detection methods.

4.3.2. Dynamic analysis

Traditional signature-based methods lacked robustness in the detection of unknown malware, and hence efficient methods are required for zero-day and stealthy/inconspicuous Android malware detection. Significant dynamic features such as permissions and API call features exhibit the behavior patterns of Android applications.

a) Behavior-based methods

Permission-based Behavior Analysis Method

The permission use behaviors include the usage of permissions to applications for accessing sensitive system resources and further usage of obtained permission-sensitive resources. Zhang et al. [17] presented VetDroid, which performed permission use analysis for examining sensitive behaviors in Android apps. VetDroid remodeled very precise malicious behaviors with real-world Android malware. VetDroid was able to find more information leaks than TaintDroid. Their system identified fine-grained vulnerabilities in some applications where it was difficult to detect. The sensitive behaviors of applications were represented as permission use graphs, reducing the irrelevant actions. VetDroid was an effective approach for examining and vetting Android apps, and useful in various fields like malware analysis and detection, vulnerability analysis, etc. Their approach did not track implicit flows and native code. It was relatively slower in comparison to static analysis solutions.

System call sequences method

Lin et al. [18] proposed a new method called System Call Sequence Droid, which is based on system call sequences. Malicious Repackaged Applications (MRAs) repackaged malware into various benign apps. The system call sequences in a particular layer were compared using the Layering MultiThread Comparison (LMTC) mechanism effectively. The common subsequences that are malicious were extracted from the system call sequences of repackaged apps belonging to the same family using the Longest Common Substring (LCS) algorithm. The common subsequences used Bayes Theorem for identifying any application without demanding an actual benign application. Detection of some MRAs like DroidDream and BaseBrige was very difficult as their behavior was disguised to appear like benign applications. The longest typical subsequences extracted clean subsequences compared to fixed-length subsequences for accurately detecting the MRAs.

API call patterns method

Fan et al. [19] detected Android piggybacked applications using Sensitive SubGraph (SSG) analysis. The proposed system worked with two assumptions to analyze the various invocation schemes of sensitive APIs of a malicious app and in its host app. Then, a subgraph was constructed to record the most skeptical behavior of an app. The invocation patterns were represented with five features from SSG which were used by machine learning algorithms for malware classification. Extensive empirical evaluation was done on a

Table 4
Experimental results of various methods for mobile malware detection.

S. No	Author	Methods	Performance Metrics				Datasets					
			Accuracy (%)	Precision	Recall	F1-score	Malware No. of Samples	No.of Classes	Collection Period	Collection Source	Dataset URL	Benign No. of Samples
1	Arp et al. (DREBIN) [25]	Static Analysis+ Machine Learning	92.58	0.9186	0.9254	0.9220	5560	179	August 2010-October 2012	Google Play, Chinese Markets, Russian Markets, Android websites, Malware forums and Security blogs	https://www.sec.cs.tu-bs.de/~danarp/drebin/	123,453
2	Narayanan et al. [20]	Static Analysis										
I.	Benchmark dataset (DREBIN)		96.22	0.9614	0.9626	0.9620	5560	179	January 2014 to August 2014	Google Play, Anzhi, AppChina, SlideMe, HiApk, FDroid, Angeeks	https://sites.google.com/view/casandrantu/dataset	5000
II.	In-the-wild (ITW) dataset		85.26	0.8474	0.8530	0.8502	42,910					44,347
3	Alam et al. [15]	Static Analysis	92.62	0.9241	0.9252	0.9246	1758	20		Google Play and various sources		3732
4	Gheorghe et al. [26]	Dynamic Analysis+ Machine Learning	94.29	0.9412	0.9425	0.9418	400			VirusShare, AVCaesar, Contagio Mobile, and Android Malware Genome Project	1. VirusShare. http://virusshare.com/ 2. AVCaesar. https://avcaesar.malware.lu/ 3. Contagio Mobile. http://contagiominidump.blogspot.ro/ 4. Android Malware Genome Project. http://www.malgenomeproject.org/	400
5	Garcia et al. [22]	Static Analysis+ Machine Learning	94.27	0.9388	0.9421	0.9404	30,203			Malware Genome Project, Drebin, VirusShare, VirusTotal	https://seal.ics.uci.edu/projects/revealdroid/	24,679

huge real dataset containing malware and legitimate apps. High detection accuracy was observed. Their method could also be used with permission-based and API-based methods by employing five features in a combined manner.

b) Machine learning-based method

Narayanan et al. [20] proposed an online learning detector called Context-aware, Adaptive and Scalable ANDRoid mAlware (CASANDRA) for the detection of malware. The security-sensitive behaviors of apps and their information were identified from dependence graphs using graph kernel, allowing an accurate detection. Their model was scalable and adaptive to the population growth and rise in new malware features. The significant features were explained to discriminate malicious apps from benign. Their system was tested on real-world datasets and benchmark datasets.

4.3.3. Vision-based analysis

Fang et al. [21] presented an Android malware family classification approach using data section features of the Dalvik Executable (DEX) file. Each DEX file is visualized as an RGB image and plaintext. The color and texture of the image, as well as the text, were extracted as features. The classification of benign and malapps was performed using a multiple kernel learning-based feature fusion algorithm. Their results showed better performance in classifying Android malware families. Their approach is time-efficient compared to the frequent subsequence approach.

4.3.4. Hybrid analysis

Hybrid analysis is performed on Android applications by combining static, dynamic, and vision-based features for effective malware classification. Garcia et al. [22] proposed an Android malware detection and classification system called RevealDroid that does not require complex program analysis or more features. Their system depends on features including API calls, indicative calls characterized by the extent to which the invoked methods can be accomplished, and native binary invocations to internal and external functions. The efficiency of their model was evaluated using a larger dataset with 54,000 malware and benign applications. They also assessed the obfuscation resiliency of RevealDroid by applying various transformations on malware applications. Their approach proved 13 times faster than other methods that use information-flow-based feature extraction. They have used a balanced dataset for evaluating their model. The risk of external validity problems may occur due to the fewer number of applications in the dataset. This leads to the issue of generalizability in the results. Internal validity issues may also occur due to mislabeling of samples. Construct validity is another issue due to the dataset transformations and non-realistic obfuscations.

Ma et al. [23] proposed an integrated method for detecting Android malware using machine learning. They extracted API information by constructing control flow graphs for Android applications. They developed three datasets, namely- boolean, frequency, and time-series datasets using the API information in the form of API calls, API frequency, and API sequence. Their method reduced the redundant APIs that resulted in better performance. They assessed three machine learning-based detection techniques based on the three datasets. They observed that their datasets distinguished malware and benign samples accurately.

4.4. Mobile malware datasets

Mobile malware datasets are crucial for building efficient malware detection and classification systems. There are various Android malware datasets publicly available for the evaluation of malware detectors. The dataset samples are collected from different app markets. The most widely used datasets available are Drebin and Malgenome. AndroZoo dataset includes many applications from various sources like GooglePlay, AppChina, etc. VirusShare, Koodous, Contagio mobile, VirusTotal, and SlideME Market are other repositories containing a large number of Android applications. The apps are labeled as malware or benign by checking using the VirusTotal service.

Table 4 provides the experimental results of various methods on several datasets for Android malware detection. The performance measures such as Accuracy, precision, recall, and f1-score are calculated to assess the performance of the detection methods. Information such as the number of classes and samples used and collection period and sources are also included.

5. Issues and challenges in malware detection

Automated detection systems are developed and successfully used in Windows/Android malware detection. Still, many severe issues are identified and need to be addressed to achieve efficient anti-malware solutions [30]. The malware detection system should focus to accomplish a good balance between timely identification and detection performance. The detection model should be real-time, automated, and powerful despite evasion techniques, targeted and unknown stealthy malware. Effective malware detection methods should be designed to perform better in terms of correctness estimation, less computational time, the low false-positive rate with the exponential growth of malware. The major issues and challenges of malware detection solutions are analyzed through perception of machine learning and the way in which they can be considered and employed by the anti-malware industry. They key concerns are presented below:

5.1. Machine learning

The state-of-the-art malware detection systems use machine learning algorithms. The performance of the classifiers was observed to be better in terms of low false positives and computational complexity. The detection of new malware remains critical. The learning aspects to address the challenges that affect the performance of the detection system must be considered.

5.1.1. Improved learning

Improved learning is the active learning of unknown samples to improve the performance of the classifier. Unknown file samples are detected based on the identification of known malware samples. The labeling of typical malware samples from the unknown sample database is essential. Before being detected, the new malware sample is labeled as unknown and the related variants of the sample are classified correctly.

5.1.2. Adversarial learning

Malware attackers design adversarial techniques to cause the classifier model to mistrain (e.g.) giving false data as input. The system should be able to learn adversarial techniques leading to a more robust and secure detection system.

5.1.3. Dynamic learning

Handling a larger training set of known and advanced malware such as behavior-based, dynamic, and unknown malware remains a challenge. The detection system is typically trained on the dataset containing previously written malware samples. The most recent malware samples should be considered for training the detection models. The classifier should learn the dynamic database with the arrival of newly evolving malware. The extraction of significant features leads to a more comprehensive detection system.

5.2. Automated malware analysis and detection for real-time use

The automated analysis and detection of malicious files in real-time is a challenge, requiring the knowledge of domain experts. The dynamic and behavioral features, as well as static features of malicious executables, should be extracted to achieve a more generalized detection system. The system should consider all of the interesting details about the malware. The system should characterize soundness and completeness to all conservative obfuscations. The system should be adaptive to the new population and the evolution of malware, and should effectively address more scalability issues. Automated real-time detection is need-of-the-hour with the arrival of zero-day malware attacks.

5.3. Second opinion to manual expert malware analysis

The classification results provided by these intelligent engines, however high the level of accuracy being reported is, are not accepted as is, by the commercial anti-malware products. Actions performed to either stop the malware or to permit a genuine application, based on these automated malware analysis reports with the assistance of machine learning techniques, have not been implemented in real-time practice yet. Those results just serve as a second opinion, augmenting the expert manual classification process. The time when the automated analysis results are used as the single source of action marks the success of these intelligent anti-malware engines.

6. Conclusion

The exponential growth of advanced and new malware has led to the need for intelligent detection systems. In this paper, we presented a survey on traditional and contemporary techniques for alleviating computer and mobile malware attacks. This survey paper provides an in-depth and lucid understanding of malware analysis, existing detection methods, and their taxonomy for computer and mobile platforms are presented. With the existence of evasion methods adopted by malware writers, the detection of malware is becoming a difficult task. The static, dynamic, machine learning and hybrid detection methods are used in the detection and classification of malware and are not able to manage the flow of dynamic and unknown malware samples. The detection systems based on machine learning techniques are an efficacious solution for intelligent malware detection. A hybrid combination of methods could be used to achieve better results in a real-world scenario. Open research problems are presented as pointers for security researchers and analysts to develop robust and real-time techniques to cope with the cyber world. It is consequent that the research on unknown malware and its prevention will take a big leap in the future.

Declaration of Competing Interest

None

Acknowledgement

The authors are grateful to VIT management for providing the Research Seed Grant (AY 2019-20) to execute this work.

Our sincere thanks to Mr. J. Kesavardhanan, Founder and Chairman, K7 Computing Pvt., Ltd., and CEO of The Association of Anti-Virus Asia Researchers (AVAR) for providing malware samples from their laboratory for analysis, and for the discussions, helping us out with malware research directions.

References

[1] The Statista Portal. https://www.statista.com/statistics/, 2021 (accessed 18 February 2021).

[2] The Positive Technologies Portal. https://www.ptsecurity.com/ww-en/analytics/antisandbox-techniques/, 2020 (accessed 18 February 2021).
[3] Denning DE. An intrusion-detection model. IEEE Trans Softw Eng 1987;SE-13:222–32. https://doi.org/10.1109/TSE.1987.232894.
[4] Shabtai A, Menahem E, Elovici Y. F-sign: automatic, function-based signature generation for malware. IEEE Trans Syst Man Cybern Part C Appl Rev 2011. https://doi.org/10.1109/TSMCC.2010.2068544.
[5] Rhee J, Riley R, Lin Z, Jiang X, Xu D. Data-centric OS kernel malware characterization. IEEE Trans Inf Forensics Secur 2014;9:72–87. https://doi.org/10.1109/TIFS.2013.2291964.
[6] Naval S, Laxmi V, Rajarajan M, Gaur MS, Conti M. Employing program semantics for malware detection. IEEE Trans Inf Forensics Secur 2015;10:2591–604. https://doi.org/10.1109/TIFS.2015.2469253.
[7] Li X, Wang X, Chang W. Cipher X ray: exposing cryptographic operations and transient secrets from monitored binary execution. IEEE Trans Dependable Secur Comput 2014;11:101–14. https://doi.org/10.1109/TDSC.2012.83.
[8] Preda MD, Christodorescu M, Jha S, Debray S. A semantics-based approach to malware detection. ACM Trans Program Lang Syst 2008. https://doi.org/10.1145/1387673.1387674.
[9] Santos I, Brezo F, Ugarte-Pedrero X, Bringas PG. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Inf Sci (Ny) 2013. https://doi.org/10.1016/j.ins.2011.08.020.
[10] Zico Kolter J, Maloof MA. Learning to detect and classify malicious executables in the wild. J Mach Learn Res 2006;7:2721–44.
[11] Nataraj L, Karthikeyan S, Jacob G, Manjunath BS. Malware images: visualization and automatic classification. In: ACM Int. Conf. Proceeding Ser; 2011. https://doi.org/10.1145/2016904.2016908.
[12] Cui Z, Xue F, Cai X, Cao Y, Wang GG, Chen J. Detection of malicious code variants based on deep learning. IEEE Trans Ind Informatics 2018. https://doi.org/10.1109/TII.2018.2822680.
[13] Rieck K, Trinius P, Willems C, Holz T. Automatic analysis of malware behavior using machine learning. J Comput Secur 2011. https://doi.org/10.3233/JCS-2010-0410.
[14] Venugopal D, Hu G. Efficient signature based malware detection on mobile devices. Mob Inf Syst 2008. https://doi.org/10.1155/2008/712353.
[15] Alam S, Qu Z, Riley R, Chen Y, Rastogi V. DroidNative: automating and optimizing detection of android native code malware variants. Comput Secur 2017. https://doi.org/10.1016/j.cose.2016.11.011.
[16] Jiang X, Mao B, Guan J, Huang X. Android malware detection using fine-grained features. Sci Program 2020. https://doi.org/10.1155/2020/5190138.
[17] Zhang Y, Yang M, Yang Z, Gu G, Ning P, Zang B. Permission use analysis for vetting undesirable behaviors in android apps. In: IEEE Trans. Inf. Forensics Secur.; 2014. https://doi.org/10.1109/TIFS.2014.2347206.
[18] Lin YD, Lai YC, Chen CH, Tsai HC. Identifying android malicious repackaged applications by thread-grained system call sequences. Comput Secur 2013. https://doi.org/10.1016/j.cose.2013.08.010.
[19] Fan M, Liu J, Wang W, Li H, Tian Z, Liu T. DAPASA: detecting android piggybacked apps through sensitive subgraph analysis. In: IEEE Trans. Inf. Forensics Secur.; 2017. https://doi.org/10.1109/TIFS.2017.2687880.
[20] Narayanan A, Chandramohan M, Chen L, Liu Y. Context-aware, adaptive, and scalable android malware detection through online learning. In: IEEE Trans. Emerg. Top. Comput. Intell.; 2017. https://doi.org/10.1109/TETCI.2017.2699220.
[21] Fang Y, Gao Y, Jing F, Zhang L. Android malware familial classification based on dex file section features. IEEE Access; 2020. https://doi.org/10.1109/ACCESS.2020.2965646.
[22] Garcia J, Hammad M, Malek S. Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Trans Softw Eng Methodol 2018. https://doi.org/10.1145/3162625.
[23] Ma Z, Ge H, Liu Y, Zhao M, Ma J. A combination method for android malware detection based on control flow graphs and machine learning algorithms. IEEE Access; 2019. https://doi.org/10.1109/ACCESS.2019.2896003.
[24] Mohaisen A, Alrawi O, Mohaisen M. AMAL: high-fidelity, behavior-based automated malware analysis and classification. Comput Secur 2015. https://doi.org/10.1016/j.cose.2015.04.001.
[25] Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K. Drebin: effective and explainable detection of android malware in your pocket. 2014. https://doi.org/10.14722/ndss.2014.23247.
[26] Gheorghe L, Marin B, Gibson G, Mogosanu L, Deaconescu R, Voiculescu VG, Carabas M. Smart malware detection on Android. Secur Commun Networks 2015. https://doi.org/10.1002/sec.1340.
[27] Anderson HS, Roth P. Ember: an open dataset for training static pe malware machine learning models. 2018. arXiv preprint arXiv:1804.04637.
[28] Roseline S, Geetha S, Kadry S, Nam Y. Intelligent vision-based malware detection and classification using deep random forest paradigm. IEEE Access; 2020. https://doi.org/10.1109/access.2020.3036491.
[29] Hemalatha J, Roseline SA, Geetha S, Kadry S, Damaševičius R. An Efficient DenseNet-Based Deep Learning Model for Malware Detection. Entropy 2021. https://doi.org/10.3390/e23030344.
[30] Kayes ASM, Rahayu W, Watters P, Alazab M, Dillon T, Chang E. Achieving security scalability and flexibility using fog-based context-aware access control. Futur Gener Comput Syst 2020. https://doi.org/10.1016/j.future.2020.02.001.

S. Abijah Roseline received the B.E. degree (2008) and the M.E. degree (2011) in Computer Science and Engineering in Anna University affiliated colleges, India. She is currently pursuing Ph.D. degree in the School of Computer Science and Engineering at VIT University, Chennai, India. Her research interests include cybersecurity, malware detection, computer vision, and machine learning.

S. Geetha received her B.E., (2002) from Madurai Kamaraj University and M.E. (2004) and Ph.D (2014) in Computer Science and Engineering from Anna University. She joined VIT University, Chennai Campus in 2014, where she serves as a Professor and Associate Dean of School of Computer Science and Engineering. Her current field of interests are cyber security and computer vision.