IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 16 January 2025, accepted 3 March 2025, date of publication 12 March 2025, date of current version 5 May 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3550781

SURVEY

From Static to AI-Driven Detection: A Comprehensive Review of Obfuscated Malware Techniques

SARANYA CHANDRAN[1], SREELAKSHMI R. SYAM[1], SRIRAM SANKARAN[1], TULIKA PANDEY[2], AND KRISHNASHREE ACHUTHAN[1], (Senior Member, IEEE)

[1]Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri 690525, India
[2]National E-Governance Division, Ministry of Electronics and Information Technology, Government of India, New Delhi 110003, India

Corresponding author: Saranya Chandran (saranyac@am.amrita.edu)

ABSTRACT The frequency of cyber attacks targeting individuals, businesses, and organizations globally has escalated in recent years. The evolution of obfuscated malware, designed to evade detection, has been unprecedented, employing new and sophisticated mechanisms to breach systems, steal sensitive data, and disrupt operations. This work advances research on obfuscated malware detection by offering a comprehensive review of studies conducted over the past decade on multiple platforms. In addition, the diversity of obfuscation techniques and the effectiveness of detection methods, such as static, dynamic, hybrid, and AI, are presented in a comparative manner. Furthermore, memory forensics, an often underexplored area, is of paramount importance for real-time analysis and the detection of advanced obfuscated malware. Hybrid analysis, which amalgamates the strengths of various approaches, emerges as a robust solution against obfuscated malware detection. The role of AI in detecting advanced ransomware, spyware, and fileless malware by enabling real-time detection and adaptive defenses against these increasingly prevalent threats is presented. In addition, a novel framework is proposed, combining Generative AI and digital twins to simulate and predict malware behavior, offering enhanced detection capabilities. This study synthesizes the findings of 76 approaches for the detection of obfuscated malware, incorporates cutting-edge technologies, and identifies open research challenges, such as ensuring scalability, enhancing generalization across platforms, and reducing resource requirements for constrained environments to guide future advancements in obfuscated malware detection.

INDEX TERMS Android, artificial intelligence, deep learning, digital twins, generative AI, hybrid detection methods, machine learning, memory forensics, obfuscated malware, obfuscation techniques.

I. INTRODUCTION

The growing reliance on the Internet in recent years has attracted hackers, who have changed their tactics from traditional crimes to online threats. As a result, protecting sensitive data and systems from the increasing frequency of cyberattacks has made cybersecurity a major priority for both individuals and organizations [1]. Attackers frequently utilize malware to carry out nefarious tasks on compromised computers [2]. Its goals include launching sophisticated assaults, causing system damage, obtaining financial gains, and stealing sensitive data. Static, dynamic, and hybrid analysis are the main categories into which malware detection techniques fall. Many private and open-source solutions have been created to use these techniques for efficient malware detection. However, these tools have limitations, and as a result, there exists a need for advanced techniques.

Obfuscated malware refers to an advanced category of malware hidden in antivirus software, making it difficult for reverse engineers to understand its true purpose. Obfuscated

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

© 2025 The Authors. This work is licensed under a Creative Commons Attribution 4.0 License.
For more information, see https://creativecommons.org/licenses/by/4.0/

malware emerged in the early 2000s [3]. In 2020, hackers exploited new or underutilized vulnerabilities, which undermined the effectiveness of traditionally popular malware forms. The SolarWinds supply chain attack, uncovered in December 2020, serves as a recent example of obfuscated malware [4]. Malware was inserted into SolarWinds' Orion software updates to launch the attack, which affected numerous clients, including public organizations and major enterprises. The attack utilized malware known as SUNBURST or Solorigate, which was significantly obfuscated to hide detection. In particular, techniques such as code signing and encryption were utilized to make it challenging to reverse engineer.

"The Cybersecurity and Infrastructure Security Agency" (CISA) reported that "Agent Tesla" was one of the top malware threats in 2021 [5]. Agent Tesla incorporated novel techniques, including obfuscation, to escape detection. "The IBM Security X-Force Threat Intelligence Index 2022" uncovered significant measures in malware evasion techniques in one year [6]. In particular, malware authors employed advanced packing and code obfuscation to hide the malware's real purpose and hinder the process of reverse engineering. The techniques involved code rearrangement, embedding hidden data, replacing code segments, renaming identifiers, and utilizing packing. Notably, standard executables employ packing as an obfuscation method to safeguard sensitive data against unauthorized access.

Unlike existing reviews, this work not only provides a comprehensive survey of techniques for the detection of obfuscated malware but also highlights underexplored areas such as memory forensics and real-time analysis. The results are structured to guide researchers and practitioners in selecting appropriate detection methods for specific use cases and identifying potential emerging research topics. This structured way guarantees the inclusion of a diverse range of studies while addressing limitations in prior reviews, contributing to a thorough knowledge of the problems and techniques for obfuscated malware detection.

A. DISTINCT CLASSES OF OBFUSCATED MALWARE

Obfuscated malware is divided into numerous distinct groups, such as packed, oligomorphic, polymorphic, and metamorphic malware [7]. Table 1 shows the different groups of obfuscated malware and their corresponding description.

B. HOW OBFUSCATED MALWARE IMPACTS DIGITAL ECOSYSTEMS

Malware now employs a variety of obfuscation techniques in the most complex and persistent attacks to hide the infection and function in order to evade detection and analysis. Table 2 lists a few effects brought forth by the obfuscated malware. As shown in the table, early versions of obfuscated malware focused on stealing sensitive data, while more recent ones generated notorious variants of malware, thus affecting the entire organization.

TABLE 1. Types of obfuscated malware.

Type	Description
Packed Malware	Malware uses packers to generate multiple versions of the same malicious code. Packers compress the files and sometimes use encryption to make unpacking difficult
Oligomorphic malware	Malware uses a variety of decryption routines, randomly selected to evade detection based on signatures.
Polymorphic malware	This is similar to the Oligomorphic malware, however, it uses mutation engines in addition to a wide range of decryption engines
Metamorphic malware	Instead of changing the appearance, malware changes its code body using a variety of obfuscation techniques

TABLE 2. Impacts of cyberattacks due to obfuscated malware.

Malware	Obfuscation Techniques	Affected Organizations	Impact	Year
Whisper Gate	String reversal, Base64 encoding, Packing	Multiple organizations in Ukraine	System destruction	2022
BatLoader	Obfuscated JavaScript files	Various organizations	Initial access, delivering malware	2022
Hive ransomware	IPfuscation, payload obfuscation	MediaMarkt, U.S. healthcare, others	Disables recovery, stops alerts, clears logs	2021
Maze	Hidden API calls, opaque predicates	LG, Southwire	Data theft, financial losses	2019
NotPetya	Wiper, MBR overwrite	Maersk	Data destruction	2017
TrickBot	Polymorphism, obfuscated scripts	Wells Fargo, PayPal	Credential theft, malware delivery	2016
Zeus botnet	Polymorphic encryption	Financial organizations	Steals banking data	2008

C. EXISTING WORK

Few surveys have been conducted on malware detection [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19], [20]. However, existing works are limited to traditional malware detection [9], [10], [11], [12], [14], [15], [16], [19], machine learning and deep learning based techniques [7], [13], [17] and ransomware detection [18], [20]. Finally, the detection of obfuscated malware was restricted to a specific platform, such as Android [8]. Table 3 shows the comparison of our study on obfuscated malware and its corresponding detection techniques with existing work.

The use of machine learning for malware detection by Daniel Gibert et al. [13], has implications for identifying obfuscated malware as well. Cannarile et al. [17] provided a comparison of deep learning and shallow learning techniques to predict API calls in malware for obfuscated malware detection. Al-Taharwa et al. [21] developed a technique to detect obfuscated JavaScript code which can be applicable to other languages.

TABLE 3. Evolution in survey insights: Contrasting the current findings with past surveys (✓: Topic is covered, X: Topic is not covered).

Survey	Obfuscated malware	Detection of obfuscated malware	Analysis Methods							Platforms		
			Static	Dynamic	Hybrid	Machine Learning	Deep Learning	Forensics	GenAI	Windows	Linux	Android
[8]	✓	✓	✓	✓	✓	X	X	X	X	✓	X	X
[9]	X	X	✓	✓	✓	X	X	X	X	X	X	X
[10]	✓	✓	X	✓	✓	X	X	X	X	X	X	X
[11]	✓	✓	✓	X	✓	✓	X	X	X	✓	✓	X
[12]	X	✓	✓	✓	✓	X	✓	X	X	X	X	X
[13]	X	✓	✓	✓	✓	✓	✓	X	X	X	X	X
[14]	✓	X	✓	✓	✓	X	X	X	X	X	X	X
[15]	X	X	✓	✓	✓	X	X	X	X	✓	X	X
[16]	✓	X	✓	✓	X	✓	✓	X	X	X	X	X
[17]	X	X	✓	✓	X	✓	✓	X	X	✓	X	X
[18]	X	✓	✓	✓	X	✓	✓	X	X	✓	X	✓
[7]	✓	X	✓	✓	✓	✓	✓	X	X	X	X	✓
[22]	X	X	✓	✓	✓	✓	✓	X	X	X	X	✓
[23]	X	X	✓	✓	✓	✓	✓	X	X	X	X	✓
Our Work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

D. CONTRIBUTIONS

The present work provides significant contributions to the literature on obfuscated malware in the following ways. Unlike existing surveys that often focus on specific platforms or a narrow range of techniques, this work provides a broad overview of obfuscated malware detection across multiple platforms, including Windows, Android, IoT, and the Web. Firstly, this platform-specific analysis is crucial, as obfuscation techniques and the effectiveness of detection methods can vary significantly across languages, and this integrated approach provides a broader perspective on the landscape and its associated challenges. Secondly, a detailed examination of various detection techniques, including static, dynamic, hybrid, machine learning, and deep learning approaches not only highlights the strengths and weaknesses of each of the techniques but also showcases the evolution and adaptation of these methods in response to the increasingly sophisticated variants of obfuscation employed by malware authors. The study presents a systematic evaluation of the effectiveness of various detection techniques between different platforms. This comparative analysis is valuable for researchers and practitioners in understanding the effectiveness and limitations of each approach, guiding the selection of appropriate techniques for specific contexts. We highlight the role of memory forensics in detecting obfuscated malware, which is often less emphasized in previous work. This inclusion highlights the significance of real-time analysis and the capability of memory forensics to complement traditional detection methods, offering a new dimension to the identification of obfuscated malware. We also explore the intersection of artificial intelligence, including both machine learning and deep learning, with traditional detection methodologies. Our work also delves into specific obfuscation techniques, including polymorphism, encryption, and packing. By dissecting these techniques, this work highlights the complexity of modern malware and the advanced strategies employed to avoid detection. Finally, by identifying and discussing key research challenges and future directions, this study not only contributes to the existing knowledge but also lays a foundation for further progress in the field. This forward-looking perspective is critical for the continued evolution of obfuscated malware detection techniques.

The sections of this paper are organized as follows:

- Section II discusses the methodology adopted.
- Section III lists the research questions explored in this study.
- Section IV includes the results and discussion as well as addresses the research questions.
- Section V discusses future research directions and open challenges for each of the techniques and targeted platforms.
- Section VI concludes the paper.

II. METHODOLOGY ADOPTED

This systematic review was carried out following the guidelines of the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) statement [22]. The literature search focused on obfuscated malware detection and included studies published between 2014 and 2024. We utilize the Scopus database, which encompasses papers from renowned publishers such as IEEE Xplore, ResearchGate, Elsevier, ScienceDirect, and Springer for this review. Scopus was chosen for its extensive coverage of peer-reviewed literature and its ability to retrieve data from multiple disciplines.

The search was conducted utilizing the following set of keywords in conjunction with Boolean operators: TITLE-ABS-KEY (‘‘obfuscated malware’’ OR ‘‘obfuscated malware detection’’ OR ‘‘machine learning for malware detection’’ OR ‘‘deep learning for malware detection’’ OR ‘‘ransomware detection’’ OR ‘‘obfuscation techniques’’). The inclusion criteria targeted studies written in English, with publicly available abstracts, that focused on obfuscated malware detection methods. Studies primarily discussing general malware detection were included only if they provided insights applicable to obfuscated malware detection. Surveys addressing single

platforms, such as Android, were supplemented by studies covering multi-platform analysis to ensure comprehensive coverage.

Exclusion criteria were applied to eliminate duplicate records, commentary articles, book chapters, and studies not aligned with the scope of this review. Two independent reviewers screened the search results to ensure accuracy and eliminate bias. The search, performed on March 2024 initially retrieved 348 publications. After applying the exclusion criteria, 228 studies were removed, resulting in 120 publications for detailed review. Figure 1 outlines the systematic selection process.

The selected studies were analyzed to identify the range of obfuscation techniques and the corresponding detection methods, including static, dynamic, hybrid, machine learning, and deep learning approaches. A comprehensive comparison was conducted to evaluate the effectiveness of detection across platforms (eg, Windows, Android, IoT, and Web) and techniques (eg, polymorphism, encryption, and packing). Special emphasis was placed on platform-specific analysis and emerging detection methods, such as memory forensics and AI-based techniques, to address advanced obfuscation tactics.

Themes such as ''AI-based malware detection'', ''polymorphic obfuscation'', and ''IoT malware detection'' were highlighted based on co-occurrence and frequency, demonstrating the evolving research focus in the field. The strengths and weaknesses of each detection method were categorized, and their evolution in response to increasingly sophisticated malware tactics was systematically mapped. This bibliometric approach offers important perspectives into the present landscape of obfuscated malware detection and guides future research directions.

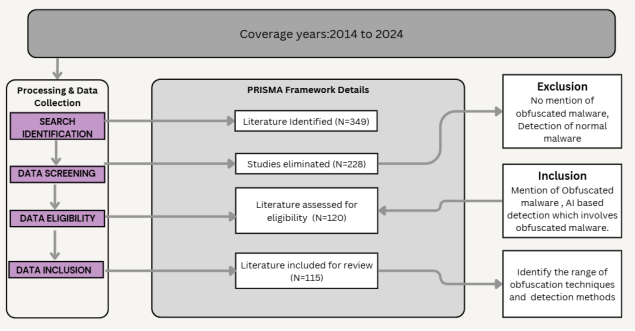

FIGURE 1. PRISMA flowchart.

III. RESEARCH QUESTIONS

In this section, we define the Research Questions for our study on obfuscated malware detection. These questions form a foundation for the selection of articles and constitute a significant portion of current research. Table 4 containing the research questions defines the framework utilized to examine the landscape of obfuscated malware detection.

TABLE 4. Formulated research questions.

No.	Research Question	Purpose
RQ1	How obfuscated malware differs from normal malware?	Identify the characteristics of obfuscated malware
RQ2	Which platforms are frequently targeted by obfuscated malware?	Identify mostly affected platforms
RQ3	How to detect obfuscated malware?	Identify popular detection methods on different platforms
RQ3.1	Which models are used for obfuscated malware detection?	Identify the frequently used models
RQ3.2	Which features are frequently employed across different analysis methods for obfuscated malware detection?	Identify the primary features
RQ3.3	Which datasets are used for obfuscated malware detection?	Identify frequently used malware datasets
RQ3.4	Which performance measures are used for obfuscated malware detection?	Identify the commonly employed performance metrics
RQ4	Which method shows good performance in detecting obfuscated malware?	Identify the appropriate method for the respective platform
RQ5	What methods can be used to accelerate the detection of obfuscated malware?	Identify future research directions

IV. RESULTS AND DISCUSSION

In this work, we analyze and compare the effectiveness of various detection techniques on different platforms, including Android, Windows, IoT, and Web. We also discuss the emerging characteristics of obfuscated malware. This study highlights the strengths and limitations of static, dynamic, and hybrid approaches, with a particular focus on their adaptability to advanced obfuscation techniques. Sections IV-A address RQ1, Section IV-B addresses RQ2, Section IV-C addresses RQ3, RQ3.1, RQ3.2, and RQ3.3. Section IV-D addresses RQ4, Section V addresses RQ5.

A. UNVEILING OBFUSCATED MALWARE

Obfuscated malware distinguishes itself from conventional malware using numerous techniques, including alterations in code sequencing, insertion of superfluous data, substitution of code segments, and renaming of identifiers. Table 5 compares traditional and obfuscated malware. Obfuscated malware obscures the presence or behavior of the malware by concealing code and critical strings. This not only hinders replication and avoids detection but also makes it difficult to comprehend through textual or binary analysis [20]. Figure 2 illustrates various malware families, including Trojans, Adware, Ransomware, Rootkits, and Spyware, which use obfuscation. Table 5 provides the response to RQ1.

B. KEY TECHNIQUES IN MALWARE OBFUSCATION

Obfuscated malware retains its functionality while modifying its signature to hide detection through methods such as code and binary obfuscation, as illustrated in Figures 3. Figure 4 highlights how malware families such as Emotet, TrickBot, and Zeus employ strategies such as encryption

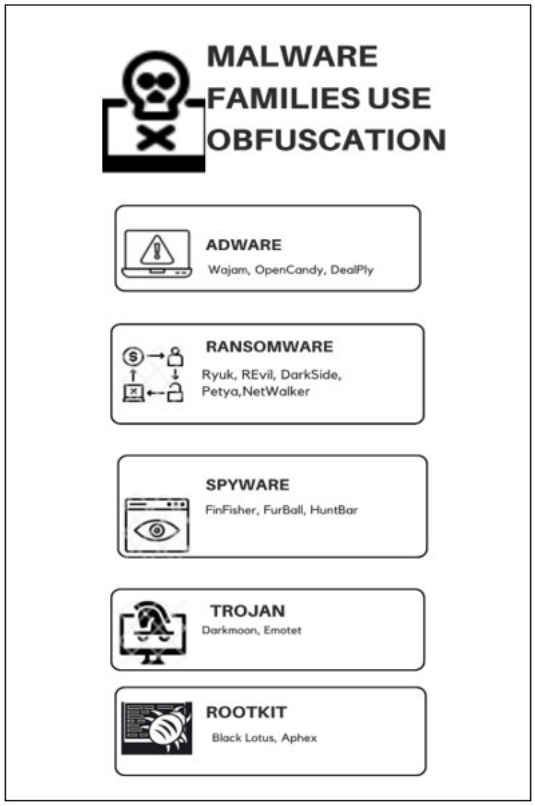

FIGURE 2. Category of obfuscated malware.

TABLE 5. Insightful comparisons in Malware Samples - Normal vs. Obfuscated.

Techniques	Normal Malware	Obfuscated Malware
Change the order of the code	X	✓
Redundant data insertion	X	✓
Code replacement	X	✓
Rename the identifiers	X	✓
Packing the code	✓	✓

and transposition to hinder reverse engineering and bypass security measures, emphasizing the overlap and adaptability of evasion methods. In response to RQ2, it is observed that, similar to conventional malware, Windows platforms are frequently targeted by obfuscated malware. Code obfuscation alters program structures employing methods like dead code insertion, subroutine rearrangement, and instruction replacement, while binary obfuscation targets compiled binaries through packing, encryption, entry point obfuscation, and stealth methods. These sophisticated approaches necessitate transitioning from static and signature-based detection to dynamic, AI-driven frameworks, underscoring the need for robust, multi-pronged strategies and continuous innovation to counter evolving threats effectively.

Obfuscated malware exhibits common behaviors such as large file sizes, compressed or packed files, and encrypted sections. While these characteristics are consistent across platforms, variations arise due to platform-specific techniques. Table 6 summarizes obfuscated malware behaviors across Android, Windows, Web, and IoT platforms. Android uses string encryption, dynamic code loading, and SMS phishing, while Windows employs advanced encryption, packing, and DLL injection. Web-based malware relies on encoded scripts and sandbox evasion, and IoT focuses on lightweight encryption, API obfuscation, and payload modifications. Common techniques include rootkit capabilities, dynamic code execution, and fileless attacks, emphasizing memory-based injection and runtime changes across platforms. These strategies highlight the adaptability of malware and the need for targeted detection methods.

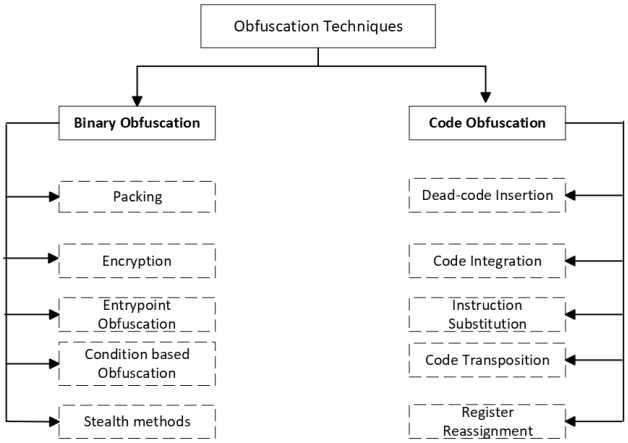

FIGURE 3. Obfuscation techniques.

C. CATEGORIZING APPROACHES TO DETECT OBFUSCATED MALWARE

We divide the detection of obfuscated malware into two categories. As shown in Figure 5, we analyze detection mechanisms based on techniques and platforms and group them accordingly. Addressing RQ3, the detection of obfuscated malware encompasses diverse approaches, including static analysis, dynamic analysis, a combination of both in hybrid analysis, AI-based techniques, and memory forensics.

a: TAXONOMY OF MALWARE DETECTION STRATEGIES

Various detection methods are employed to identify different types of malware. As shown in Figure 6, we observe that significant work has been carried out in Static analysis while little emphasis has been placed on memory forensics for obfuscated malware detection.

1) STATIC ANALYSIS

Static analysis looks at the malware's source code or binary without running it to identify any potentially suspicious activity. Since the obfuscated malware hides its existence through encryption or packing, the executables must be decrypted or unpacked. Features, including API calls, strings, opcodes, program semantics, pictures, permissions, and

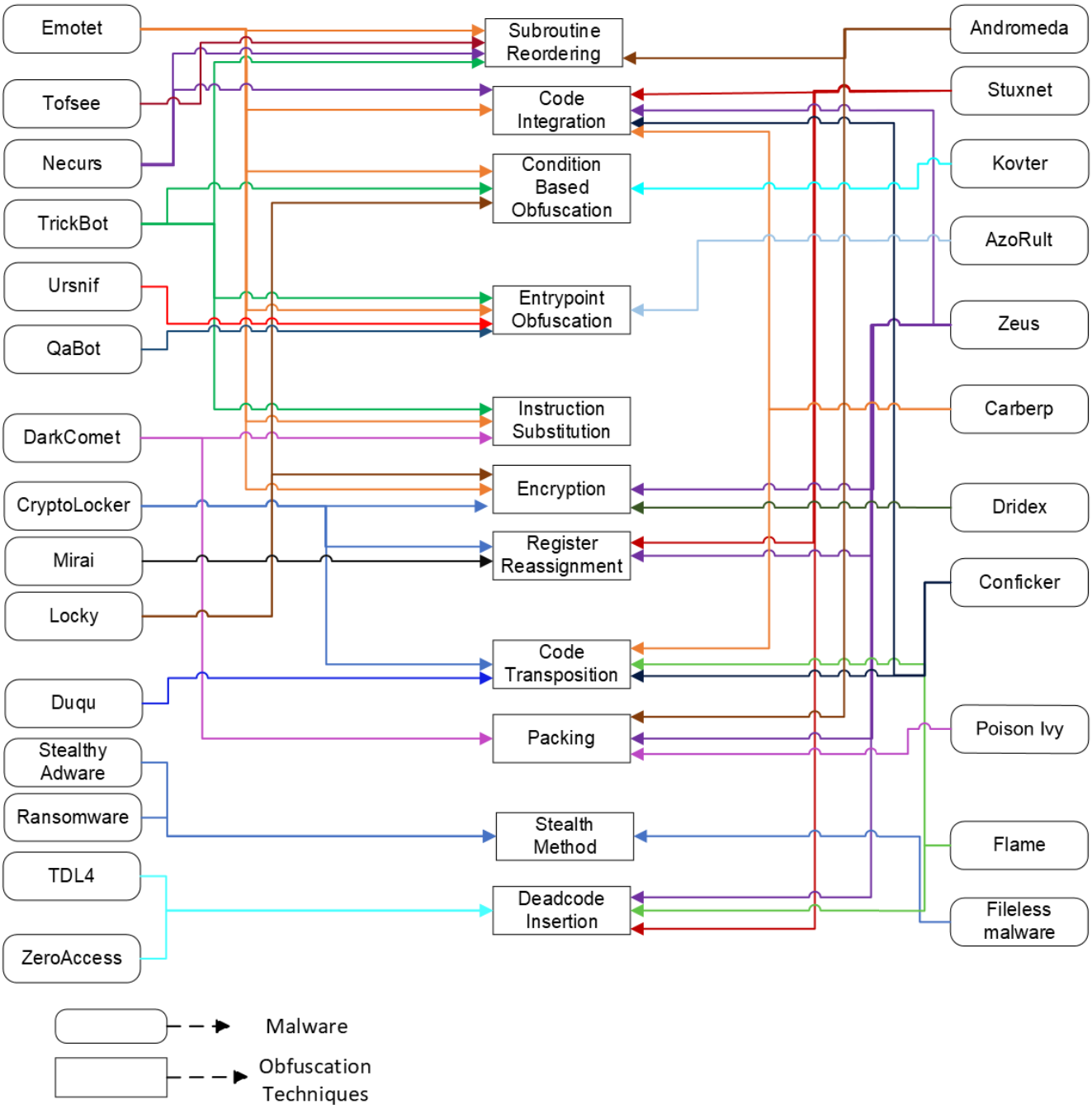

FIGURE 4. Instances of malwares and their employed obfuscation techniques.

more, can be extracted once the executable has been unpacked and decrypted.

- *Program semantics:*
  Attackers use obfuscation techniques, including cryptographic functions, to evade detection by pattern-matching malware scanners. To identify evasions that take advantage of code variation approaches, context-based information is taken out of the code [21]. As suggested in [23], capturing the semantics of potential cryptographic algorithms from the executable aids in their identification and helps security analysts to detect malicious behavior. DeepRefiner [24] detects obfuscated Android malware by analyzing XML data and bytecode semantics. Another approach [25] identifies obfuscated files by detecting irregularities in instruction-based features, such as opaque predicates and control flow graphs, highlighting statistical properties for enhanced detection.

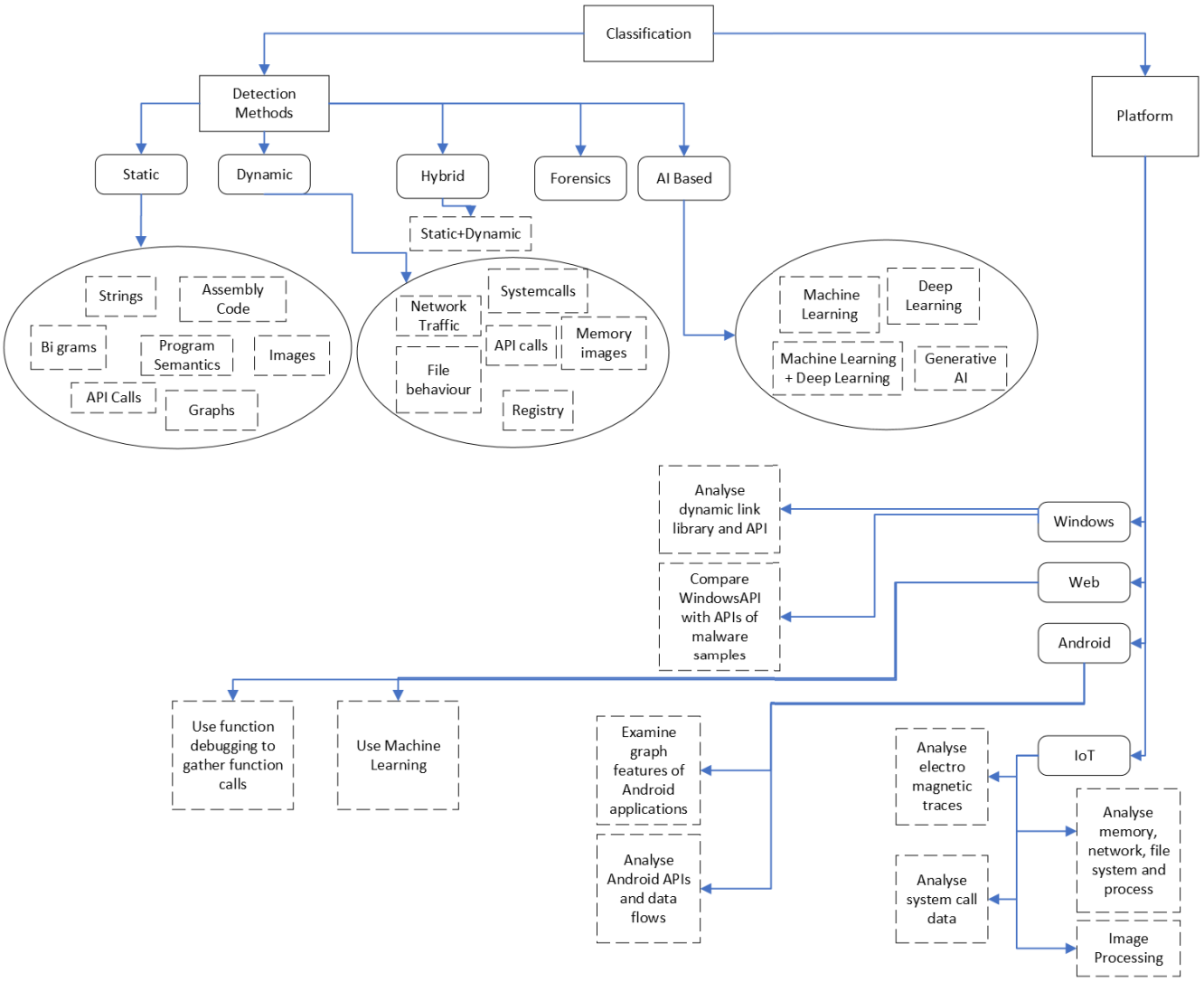

FIGURE 5. Classification of various detection mechanisms of obfuscated malware.

- *Assembly code/bytecode/opcode:* Malware detection involves analyzing native code features extracted from Android applications using techniques like opcode sequences and lexical analysis. Android Runtime converts bytecode into native code for machine learning analysis [26], while [27] uses lexical information such as strings and class names. Convolutional neural networks identify patterns in raw opcode sequences to detect obfuscated malware [28]. Similarly, opcode sequences and classification engines help identify obfuscation techniques like junk code insertion and code reordering [29]. Memory access patterns, including opcodes like MOV and XOR, are also used for detection [30]. MGOPDroid [31] leverages TF-IDF and grayscale image transformation of opcode features to detect obfuscated Android malware.
- *Strings:* Seshagiri et al. [32] analyzed HTML attributes and JavaScript functions to identify potentially malicious activities. Specifically, the detection process focuses solely on standard attributes commonly found on web pages to assess the presence of harmful behavior. The authors in [27] proposed a different method for detecting lexical obfuscation and string obfuscation in Java malware. Their suggested metrics involve examining word count, identifier length, and the frequency of rare bigrams and unreadable characters.
- *Images:* To detect malware, researchers utilize image-based techniques by converting opcode sequences into visual representations. Grayscale malware images map opcode features to pixels, enabling advanced deep-learning models for classification [31], while colored malware images, created by transforming malicious binaries into 2D arrays and applying color maps,

TABLE 6. Incorporation of obfuscated malware characteristics across diverse platforms.

Techniques	Android	Windows	Web	IoT
Code Encryption	Encrypts strings in the code	Obfuscates code using various encryption techniques	Encoded scripts	Employs lightweight encryption algorithms like XOR, AES, or RC4
Anti-Analysis Techniques	Uses obfuscation to evade security detection and permission-based techniques	Packed with UPX or custom tools	Browser fingerprinting, sandbox evasion	Obfuscates readable strings, such as file paths and API calls
Polymorphic/ Metamorphic Behavior	Transforms code structure dynamically	Self-mutates; generates runtime code	Session-specific script generation	uses lightweight encryption or packing methods and it changes for each version
Dynamic Code Loading	Injects code into legitimate apps or processes	Injects malicious code into DLLs or processes	Asynchronous script injection	Remote payload fetching
Rootkit Capabilities	Lacks rootkit capabilities	Manipulates system functions and data structures	Backend script persistence	Intercepts system calls, hides processes/files
Network Communication	Adds SMS phishing, spam emails to its activities	C2 communication, data exfiltration, RCE, exploits vulnerabilities	HTTPS is commonly used to encrypt communication.	Uses SSL/TLS encryption to secure communication with command-and-control servers
Dynamic Behavior	Uses droppers to download additional payloads	Resolves API calls at runtime	Behavior adjusted to browser/ server context	Fetches encrypted modules and decodes them in-memory to execute malicious commands
Fileless Techniques	Typically relies on file-based persistence	Stores code in memory or registry, injects malicious DLLs	Exploits APIs, Document Object Model (DOM), and browser memory	Memory-only execution

improve detection accuracy [33]. These images reveal distinct patterns for malware families. Additionally, Markov-based methods transform binary data into transition probability matrices, visualized as images where pixel values represent scaled probabilities, effectively capturing obfuscation patterns [34]. Combining global and local features in detection remains a promising area for research [35].

- *API calls:* Analyzing API call patterns can reveal malicious behaviors and detect obfuscated malware. Techniques include extracting API sequences as local features for machine learning models [35], generating API family graphs [36], and identifying resilient features like sensitive APIs, data flows, and intent actions [37]. API call statistics are also utilized as features, with automated systems available for extracting API lists from executables [38].
- *Graphs:* Researchers use graph-based models to analyze API call relationships and detect malicious behavior. In [36], API calls are represented as weighted directed graphs, where nodes denote API methods, edges indicate method calls, and weights represent call frequencies. Graph features are used to distinguish malicious apps by analyzing structural similarities. The MCFSM algorithm [39] detects common malicious patterns through frequent subgraph mining, effectively differentiating benign and malicious apps. In [40], graphs of Android APIs and system-level methods, resilient to obfuscation, are utilized to identify sensitive paths that reveal malicious behavior. Additionally, [41] introduces co-opcode graphs to capture engine-specific opcode patterns, generating high-level signatures for each metamorphic malware family.

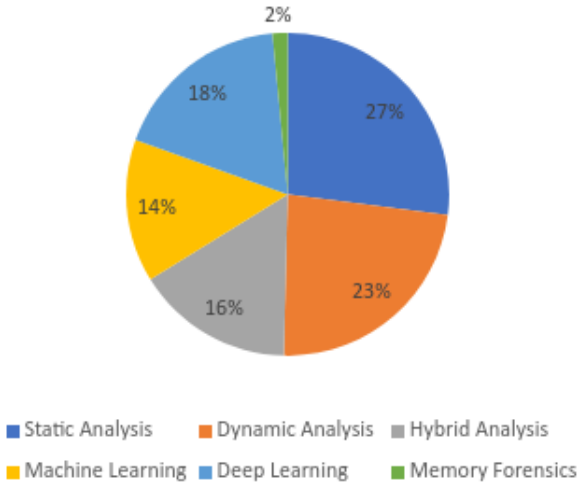

FIGURE 6. Prevalence of research based on detection techniques.

Our analysis reveals that numerous approaches performed static analysis of obfuscated malware on Android platforms. We list the advantages and disadvantages of the most relevant approaches in Table 7. In static analysis, both high-level (API and function calls) and low-level features (opcodes and assembly code) are commonly analyzed for malware indicators. Additional features such as HTML attributes, JavaScript functions, permissions, cryptographic functions, and library calls enhance detection capabilities. Emerging techniques include converting code into grayscale or color images to identify structural similarities through visualization and using graph-based features for malware detection.

Despite these advancements, static analysis is inherently limited in handling dynamically triggered obfuscation techniques such as polymorphism, metamorphism, packing, string encryption, control flow obfuscation, and reflection that conceal their true behavior until runtime. The exponential

increase in the number of malware samples makes it challenging for static analysis to keep up, especially when manual intervention is required to de-obfuscate code. This critical analysis underscores the need for hybrid approaches that combine static and dynamic analysis to provide a more robust defense against obfuscated malware.

TABLE 7. Highlights of techniques used in static analysis.

Title	Adopted Technique	Platform	Advantages	Disadvantages
Unified Detection of Obfuscated and Native Android Malware [26]	Features extracted from native code using ART to translate bytecode into native code	Android	Detects both obfuscated and native malware	Fails to detect dynamically triggered obfuscation
Evaluating Opcodes for Detection of Obfuscated Android Malware [29]	Extracts opcode sequences; trains classification engine	Android	Efficient feature set when using opcode sequences	Efficiency drops with different obfuscation samples
DaDiDroid: Obfuscation Resilient Tool via Call Graph Modelling [36]	Builds weighted directed API call graphs; checks structural similarity	Android	High accuracy (up to 96% on obfuscated apps)	Ignores dynamic features
AMA: Static Analysis of Web Pages for Malicious Script Detection [32]	Deobfuscates strings, applies plaintext attack, checks HTML attributes and JavaScript	Web (JavaScript, HTML)	Simple classification for malicious web pages	Fails for exploits not imported via iframe or JavaScript
Minimal Contrast Frequent Pattern Mining for Malware Detection [39]	Uses minimal contrast frequent subgraph miner for malicious pattern detection	Windows	High detection rates; low false positives	Struggles with dynamic code generation
JSOD: JavaScript Obfuscation Detector [21]	Creates syntax tree to extract context-based information	Web	Detects a wide range of evasive code variations	Focuses on obfuscation detection, not malicious behavior

2) DYNAMIC ANALYSIS

Dynamic analysis, a technique performed in controlled environments, examines malware behavior during execution by monitoring malware's behavior such as system calls, network activity, file system changes, and registry modifications. The authors in [42] proposed a ransomware detection system using dynamic analysis to generate a control flow graph of API calls through behavioral monitoring. Suarez et al. introduced Alterdroid, a dynamic tool that compares the similarities between the original application and various modified versions. Researchers in [25] proposed a method that identifies irregularities in instruction sequences caused by obfuscation techniques such as opaque predicates, junk code insertion, and anti disassembly strategies. The authors in [43] proposed DynODet, a system that focuses on runtime behaviors and leverages dynamic analysis to identify obfuscated malware that adapts its behavior during execution. By analyzing runtime API calls, memory usage patterns, and control flow alterations, the system effectively detects obfuscation tactics such as code unpacking, dynamic function generation, and self-modifying code. The researchers present a method for detecting malware by analyzing the sequence of API calls made by the application [44]. The authors in [45] monitored the behavior of suspicious web pages in a real browser and detected obfuscation by analyzing internal function calls. The authors in [46] monitor the runtime behavior of applications to capture features such as API calls, memory usage, and network activity, which is useful in identifying obfuscated and polymorphic malware. Additionally, researchers in [47] examined the behavioral similarities between the original application and its automatically generated variants.

Dynamic analysis is essential for detecting obfuscated malware by monitoring its real-time behavior, including system calls, network traffic, and interactions with the file system and registry. It helps identify new threats by capturing memory images and analyzing the behavior of executable files, often revealing actions missed by static analysis, such as DLL injection, API call obfuscation, packing, and encryption. However, dynamic analysis has limitations, including its time-consuming nature, which makes it impractical for analyzing a large volume of new malware samples. Additionally, it is challenging to perform dynamic analysis in resource-constrained environments such as IoT, requiring secure, isolated setups that are expensive to maintain. While dynamic analysis can uncover a significant portion of the malware's behavior, it often fails to explore all potential execution paths, reducing its comprehensiveness. To improve effectiveness, researchers are focusing on enhancing dynamic analysis techniques with advanced methods such as Hidden Markov Models and ensemble learning, which increase accuracy and efficiency in detecting obfuscated malware.

3) HYBRID ANALYSIS

Hybrid analysis uses a combination of static and dynamic analysis to aid malware analysts in identifying and analyzing malware behavior. Khanmohammadi et al. [48] proposed a technique that uses both static and dynamic analysis to derive API call traces from the execution of an application's services. Services are important because they possess specific characteristics that make them lucrative to prevent potential misuse. Sun et al. [49] developed a framework to integrate runtime behavior with static structures for malware detection. [50] extracted features such as permissions and intent through static analysis and those such as network traffic using dynamic analysis. Kim et al. [51] computed a similarity measure to classify malware based on attack behavior patterns and tactical features. The authors in [52] introduce an integrative feature extraction algorithm that combines

TABLE 8. Highlights of techniques used in dynamic analysis.

Title	Adopted Technique	Platform	Advantages	Disadvantages
Automatic Ransomware Detection and Analysis Based on Dynamic API Calls Flow Graph [42]	Generates API calls flow graphs (CFG) by monitoring dynamic behaviors	Windows	Effectively detects a variety of ransomware	Limited sample size
Thwarting Obfuscated Malware via Differential Fault Analysis [47]	Analyzes behavioral differences between the original app and automatically generated versions (Differential Fault Analysis)	Android	Detects concealed potentially malicious elements in an application package	Limited to specific Android versions
Detecting Obfuscated JavaScript Malware Using Sequences of Internal Function Calls [45]	Groups similar function call sequences using normalized Levenshtein distance (NLD) metric and generates signatures	Web	Examines suspicious web pages' behavior in an actual web browser	Focuses solely on iFrame injection attacks
Instructions-Based Detection of Sophisticated Obfuscation and Packing [25]	Detects obfuscation based on anomalies in instruction-based characteristics	Windows	Considers structural and instruction features	Struggles with files where unpacked and packed sections are separate
A new malware classification approach based on malware dynamic analysis [48]	Detection based on system calls, memory usage, and file interactions	Windows	Handle large volumes of malware samples	Challenging in resource-constrained environments
A novel approach to detect malware based on API call sequence analysis [44]	Employs DNA sequence alignment algorithms to analyze API call sequences	Windows	Adaptable to various malware categories	Limitations to dynamic analysis

static features, such as API calls, with dynamic features, including file, registry, and network behaviors, using the Simhash algorithm. Hellal et al. [39] combined signature-based techniques with behavior-based techniques to detect polymorphic malware more effectively in a limited time. The researchers in [53] develop a knowledge distillation technique to transfer knowledge learned from aggregated features from a teacher model to a student model trained only on static features. The authors in [54] integrate both static and dynamic analysis, where static analysis is focused on code structure, API calls, and embedded features, while dynamic analysis observes run-time behavior to detect obfuscated malware. Numerous approaches for detecting obfuscated malware using hybrid analysis are summarized in Table 9.

Although hybrid analysis combines the benefits of static and dynamic analysis, it faces limitations due to the single-path execution constraint of dynamic analysis, resulting in incomplete path coverage. Enhancing the accuracy of hybrid models requires focusing on selecting relevant features and exploring multiple execution paths. Finally, cost-effectiveness remains a priority for researchers to make hybrid analysis more efficient.

TABLE 9. Highlights of techniques used in hybrid analysis.

Title	Adopted Technique	Platform	Advantages	Disadvantages
Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android [50]	Combines runtime behavior with static structures	Android	Achieves around 99% accuracy in detection	Limited to keyboard input-based behavior
yDroid: A Hybrid Approach for Generating API Call Traces from Obfuscated Android Applications for Mobile Security [49]	Applies static analysis and generates API call traces from the execution of an app's services	Android	Efficiently retrieves API call traces from application services	Does not handle native code
A simhash-based integrative features extraction algorithm for malware detection [53]	Integrate both static and dynamic features using SimHash algorithm	Windows	Improve detection accuracy	Integration process is complex and affects the scalability
Malware detection by static checking and dynamic analysis of executables [55]	Combining code structures, API calls with dynamic behaviours	Windows	Provide a holistic view of the executable's characteristics	Limitation of dynamic analysis
Minimal Contrast Frequent Pattern Mining for Malware Detection [39]	Detect obfuscation by identifying minimal contrast frequent patterns	Windows	Reduces the complexity of pattern mining	Uncertain about the effectiveness in large scale datasets with numerous malware variants

4) ARTIFICIAL INTELLIGENCE BASED ANALYSIS

Algorithms using artificial intelligence (AI), which includes machine learning and deep learning, are able to examine vast amounts of data and find patterns that are difficult for people to detect.

a: MACHINE LEARNING

Machine learning plays a crucial role in malware detection by uncovering patterns and anomalies within large datasets. Techniques like Support Vector Machines (SVM) excel in identifying obfuscated malware patterns through complex decision boundaries. Algorithms such as C4.5 use information gain to classify obfuscated code, while Random Forest aggregates multiple decision trees for improved accuracy. Logistic regression provides probabilistic insights, and adaptive boosting enhances performance by combining weak learners. Other techniques such as Sequential Minimal Optimization (SMO) refine SVM efficiency, and K-Nearest Neighbors (KNN) classify malware based on proximity to

neighboring samples. Collectively, these approaches form a robust toolkit against obfuscated malware.

We categorize existing literature into the following sections based on the techniques for feature extraction.

- *Static Feature Extraction:* The process of static feature extraction in Machine Learning involves extracting features from the application without execution. Kumar et al. [27] proposed detecting Java malware by analyzing bytecode extracting features such as word count, identifier length, and non-readable character frequency. Li et al. [40] developed a graphical representation to extract features such as path and graph statistics, enabling resilience to code obfuscation.
- *Dynamic Feature Extraction:* Dynamic analysis extracts features during execution in sandbox environments. Galal et al. [55] used API call traces with classifiers like SVM and Random Forest to achieve high accuracy but noted limitations in detecting malware triggered by external events. Han et al. [56] proposed profiling malware through structural and behavioral aspects, transforming profiles into feature spaces for classification.
  Garcia et al. [37] extracted obfuscation-resilient features such as sensitive APIs and data flows but highlighted challenges in comprehensive feature extraction. Hansen et al. [57] used innovative feature representations to reduce feature space, employing Random Forest for classification. Feng et al. [58] developed EnDroid, a framework using behavioral features such as system-level traces, but its focus on IP addresses and ports excluded network-operated malware.
- *Hybrid Feature Extraction:* Wen et al. [59] leveraged static (e.g., permissions, APIs) and dynamic features (e.g., CPU usage, battery life) for obfuscated malware detection. Shaukat et al. ([60] introduced RansomWall, integrating static features (e.g., suspicious strings) with dynamic features (e.g., file operations), achieving near-zero false positives and detecting zero-day samples using Gradient Tree Boosting. Ali et al. [61] created N-grams from execution traces, transformed into binary vectors for machine learning but limited features to API calls. Venkatraman et al. [62] addressed packed and polymorphic malware through statistical n-gram opcode analysis. Xue et al. [63] proposed Malscore, combining static grayscale image analysis with dynamic API call sequences using n-grams, offering probabilistic scoring for classification accuracy.
  Table 10 is a list of some machine learning methods used to identify malware that has been obfuscated. Each technique's benefits and drawbacks are also covered.

TABLE 10. Highlights of techniques used in machine learning approaches.

Title	Adopted Technique	Platform	Advantages	Disadvantages
Obfuscated Malware Detection in IoT Android Applications Using Markov Images and CNN [34]	Markov images trained using CNN	Android + IoT	Sustainable and cost-effective	No information about power consumption
SHIELD: A Multimodal Deep Learning Framework for Android Malware Detection [64]	Markov images of opcodes + dynamic APIs	Android	Automatically discovers relevant features, 99.52% accuracy	Unable to detect zero-day malware, ransomware
Android Malware Obfuscation Variants Detection Based on Multi-Granularity Opcode Features [31]	Image-enhanced deep learning model	Android	Detects variants of obfuscated malware	Ignores packed apps and dynamic code loading
Windows Malware Detection Based on Deep Learning [65]	Malware visualization + CNN	Windows	Combines static and dynamic methods	Performs poorly on older malware and Trojans
Visualization and Deep Learning-Based Malware Variant Detection Using OpCode-Level Features [66]	Semi-supervised approach + image transformation	Windows	Low computational cost	Considers only static features
Dynamic Analysis for IoT Malware Detection with Convolution Neural Network [67]	Behavior analysis in nested cloud environments	IoT	Learns from vast behavior datasets	Fails if malware evades virtualization
DeepRefiner: Multi-Layer Android Malware Detection System [24]	Multi-layer deep neural network	Android	Robust against obfuscated malware	Features are hard to interpret for humans
Deep Learning for Classification of Malware by System Call Sequences [68]	Convolution of n-grams + sequential modeling	Not specified	High precision (85.6%) and recall (89.4%)	Vulnerable to noise in call sequences

b: DEEP LEARNING

Deep learning, a subset of machine learning, processes unstructured data, learns, and extracts features without human input. Deep neural networks, also known as artificial neural networks (ANN), are essential to deep learning applications. Convolutional neural networks (CNNs), a type of ANN, are most popular in malware detection. The early layers of a CNN are responsible for identifying low-level malware features. Subsequent convolution layers combine these features into a more complete representation of the malware.

Tang et al. [31] proposed MGOPDroid, which extracts opcode features at different granularities using the TF-IDF algorithm to compare opcode feature distributions before and after obfuscation. These features are visualized as grayscale images and processed by deep learning models. However, MGOPDroid does not address packed Android apps. Lu et al. [64] developed a hybrid model combining Deep Belief Network (DBN) for static features with a Gated Recurrent Unit (GRU) for dynamic feature sequences. Ngo et al. [65]

introduced a transfer learning model aggregating latent features generated by deep learning, further enhanced by knowledge distillation. Singh et al. [66] proposed SHIELD, which uses Markov images of opcodes and dynamic APIs with a multimodal autoencoder (MAE) to reduce dependency on manual feature engineering.

Xu et al. [24] developed DeepRefiner, a two-layer Android malware detection system analyzing XML files and performing semantic analysis. It eliminates the need for manual feature engineering. Ding et al. [50] employed the Res7LSTM model to classify partially benign and malicious samples detected during static analysis. Darem et al. [67] introduced a semi-supervised technique that combines feature engineering, deep learning, and image transformation to detect obfuscated malware. Binary files were converted to integer vectors and grayscale images for efficient analysis.

Mahdavifar et al. [68] proposed CopperDroid, a virtual machine-based system that reconstructs complex Android object interactions via system call execution analysis. It uses a semi-supervised deep neural network to classify malware categories based on dynamic behavior frequencies. Kolosnjaji et al. [69] developed a combined convolutional and recurrent neural network architecture that integrates sequential modeling with n-gram convolution, outperforming traditional methods for obfuscated malware classification.

Huang et al. [70] combined CNNs and visualization techniques using the VGG16 network. Results from the dynamic analysis were transformed into visualization images via Cuckoo Sandbox, followed by model training on static and dynamic images. However, this model does not work with older malware and QQ password-stealing trojans. Venkatraman et al. [71] proposed a hybrid deep learning and visualization approach for malware detection, leveraging malware behavior pattern similarity measures and cost-sensitive architectures. The scalable model achieves lower computational overhead and supports real-time training for emerging threats.

Deep learning approaches for identifying obfuscated malware are predominantly utilized on Android platforms, as shown in Table 10. These techniques are either used independently or in conjunction with static and dynamic analysis and visualization to enhance the accuracy of the latter techniques if used in a stand-alone manner.

TABLE 11. Techniques and highlights in deep learning-based malware detection.

Title	Adopted Technique	Platform	Advantages	Dis-advantages
Obfuscated Malware Detection in IoT Android Applications [34]	Markov images trained using CNN	Android + IoT	Sustainable and cost-effective	No information on power consumption
SHIELD: Multimodal Framework for Malware Detection [67]	Markov images of opcodes + dynamic APIs	Android	Automatically discovers relevant features, 99.52% accuracy	Unable to detect zero-day malware, ransomware
Malware Detection Using Multi-Granularity Opcode Features [31]	Image-enhanced deep learning model	Android	Detects variants of obfuscated malware	Ignores packed apps and dynamic code loading
Windows Malware Detection via Deep Learning [71]	Malware visualization + CNN	Windows	Combines static and dynamic methods	Performs poorly on older malware and Trojans
Malware Variant Detection Using OpCode Features [68]	Semi-supervised approach + image transformation	Windows	Low computational cost	Considers only static features
Dynamic IoT Malware Detection [73]	Behavior analysis in nested cloud environments	IoT	Learns from vast behavior datasets	Fails if malware evades virtualization
DeepRefiner: Multi-Layer Android Malware Detection [24]	Multi-layer deep neural network	Android	Robust against obfuscated malware	Features are hard to interpret for humans
Malware Classification via System Call Sequences [70]	Convolution of n-grams + sequential modeling	Not specified	High precision (85.6%) and recall (89.4%)	Vulnerable to noise in call sequences

c: INTEGRATION OF MACHINE LEARNING AND DEEP LEARNING

In our analysis, we observed that certain approaches integrate machine learning and deep learning to enhance the effectiveness of obfuscated malware detection. Lee et al. [72] proposed SeqDroid, a classifier that leverages recurrent neural networks (RNNs) and convolutional neural networks (CNNs) to analyze patterns in Android metadata. By stacking RNNs and CNNs, SeqDroid effectively classifies malicious apps based on features such as package names, developer names, and capability information, improving the detection of obfuscated malware.

Asam et al. [73] integrated customized CNNs for feature extraction with SVMs for classification. Similarly, Xue et al. [63] developed Malscore, which combined CNNs with Spatial Pyramid Pooling and similarity algorithms. These algorithms compute the degree of similarity between an unknown binary and predefined rules, classifying binaries as obfuscated malware if they exceed a similarity threshold. It is found that the number of studies that use Similarity Algorithms to detect obfuscated malware is seventeen.

Further, we classify AI-based techniques for obfuscated malware detection into seven categories: Neural Networks (NN), Ensemble Learning (EL), SVM, Decision Trees (DT),

TABLE 12. The distribution of studies related to detection models.

Model	No	Percentage
NN	45	33
EL	27	20
SVM	20	15
Similarity Algorithm	17	13
DT	10	7
Naive Bayes	6	4
Logistic Regression	6	4
Miscellaneous	5	4

Naive Bayes, Logistic Regression (LR), and Miscellaneous. Table 12 displays the count and percentage of the studies with respect to corresponding categories.

Addressing RQ3.1, our analysis reveals that neural networks are the most frequently employed model, comprising the largest portion of 33% of the approaches attributed to CNNs, followed by ensemble learning at 20%. The pie charts in Figure 7 illustrate the most commonly utilized models in their respective categories, along with the number and percentage of studies that incorporated these models.

In machine learning for malware detection, features are extracted through static, dynamic, or hybrid analysis, with API calls commonly serving as key features. The effectiveness of these methods is influenced by the limitations of both static and dynamic analysis. Most existing models are evaluated using accuracy and false positive rate metrics, but critical factors such as sample size, feature selection, dataset, and training set choice significantly influence model accuracy and performance. Deep learning techniques have been widely applied to detect obfuscated malware, with many approaches combining CNNs and visualization-based methods to identify patterns in malware behavior. Models are commonly trained on features like API calls, opcode patterns, XML file attributes, and system calls, leveraging these indicators to recognize obfuscation. Approaches such as semi-supervised learning, Res7LSTM, Gated Recurrent Units (GRUs), and Deep Belief Networks have also been used to improve detection accuracy.

d: ROLE OF AI IN DETECTION OF RANSOMWARE, SPYWARE AND FILELESS MALWARE

Ransomware detection remains challenging due to its evolving nature and the inability of traditional methods to generalize across variants. Abualhaj et al. [74] proposed a fine-tuned decision tree classifier optimized for memory features, focusing on tree depth and splitting criteria. While effective in controlled experiments, scalability and real-time performance in resource-constrained environments remain concerns. Kaur et al. [75] applied transfer learning to system call data for ransomware detection, addressing dataset limitations. However, real-time integration and performance require further study. Zanoramy et al. [76] focused on detecting pre-encryption behaviors; however, ransomware variants that obscure or delay these activities could evade the proposed detection method.

Hassan et al. [77] proposed a hybrid DNN-BiLSTM architecture combining high-level feature extraction with temporal sequence analysis. Though it reduces reliance on manual feature selection, the quality of initial feature extraction remains critical. Integrating this model with anomaly detection or sandboxing could enhance security. Our analysis reveals that advanced deep learning models improve ransomware detection, especially for obfuscated scenarios.

For spyware, Abualhaj et al. [78] developed a framework using optimized decision trees, focusing on behavioral and static features such as API calls and registry modifications. Challenges persist in detecting advanced spyware variants that adapt their behavior to evade detection, but the use of ensemble methods and anomaly detection systems could enhance robustness. Javaheri et al. [79] developed kernel-level monitoring of system calls for malware detection, which is limited in cross-platform compatibility.

Wu et al. [80] developed a framework leveraging behavioral analysis and machine learning to detect malicious PowerShell scripts, commonly used in fileless malware attacks, including obfuscated and polymorphic variants. However, offline environments pose challenges due to the dependence on Antimalware Scan Interface and external services such as VirusTotal. Yang et al. [81] proposed PowerDetector, a multimodal system combining deep learning and semantic fusion to analyze PowerShell scripts. However, scaling the system to large datasets or high-throughput settings remains an issue. Techniques such as memory analysis, PowerShell script detection, and endpoint detection and response (EDR) are being developed to tackle these advanced threats.

Despite these advancements, existing methods often fail to entirely capture the complexity of evolving malware and obfuscation techniques. Few-shot learning, which enables training with limited samples, shows promise in detecting rare or fileless malware variants where large datasets are unavailable. Expanding deep learning models to address diverse obfuscation strategies is of paramount importance for future research.

5) MEMORY FORENSICS

Memory forensics enables real-time analysis of a system's volatile memory, assisting security experts in identifying and responding to active malicious processes and behaviors. The authors in [82] executed malware samples and utilized the Cuckoo Sandbox [83] to analyze file behavior and obtain memory images. Cuckoo generated dynamic analysis reports, while the Volatility tool [83] was utilized to analyze memory images and produce memory analysis reports. In addition, API calls were extracted from both reports, and machine learning algorithms were applied for detection.

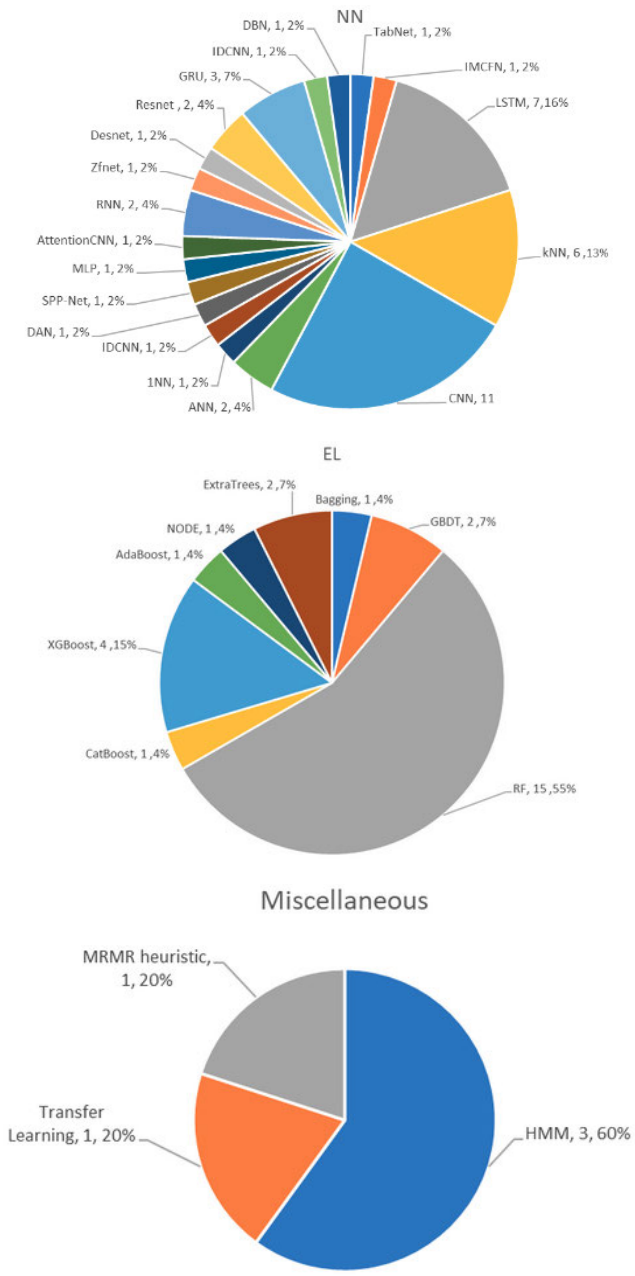

FIGURE 7. The distribution of subcategories with NN, EL and Miscellaneous.

The memory analysis report contained certain APIs not available in the dynamic analysis report, leading the authors to conclude that memory forensic analysis addresses the limitations of single-path execution in dynamic analysis. The researchers in [84] leveraged runtime memory features, such as heap and stack behaviors, memory allocation patterns, and dynamic API call sequences. These features are resilient to static obfuscation methods such as packing or encryption. Similarly, the authors in [85] extracted features from memory, including heap and stack usage, API call sequences, and memory allocation patterns. Their proposed method was tested on a dataset of obfuscated malware samples from various platforms, including Windows, Android, and IoT environments.

Some researchers developed advanced feature engineering techniques to extract relevant information from memory dumps. These features include API call traces, memory allocation patterns, entropy measures, and execution flow structures [86]. Further, researchers in [87] proposed a framework that relies on analyzing run-time memory data and extracting features such as memory usage patterns, entropy, and API call sequences to identify suspicious behaviors and patterns. Advanced plugins such as Rekall, imgmalfind, and other Volatility plugins can be used to detect modified memory pages and pinpoint exact malicious modifications and hidden memory areas. These tools are particularly useful in scenarios where malware employs shared memory to evade detection, thereby improving the detection of obfuscated malware. Collecting multiple memory dumps over time can facilitate comprehensive analysis and improve the accuracy of malicious process detection. However, advanced malware often utilizes techniques such as process injection and system call abuse to evade detection. Therefore, frameworks capable of addressing such sophisticated methods are needed.

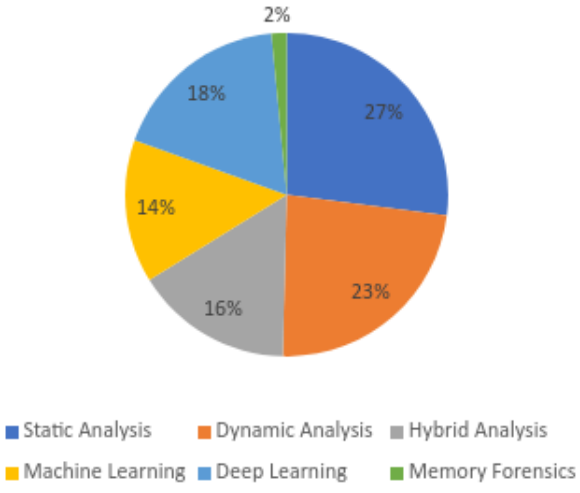

FIGURE 8. Prevalence of research based on detection techniques.

To summarize the classification based on detection techniques, as shown in Figure 8, most detection techniques rely on either static or dynamic analysis. Figure 10 illustrates the distribution of features for obfuscated malware detection, while Figure 11 highlights the commonly used datasets for obfuscated malware across different platforms. The y-axis in Figures 10 and 11 represents the respective counts, while in 10, the inline graph represents the features used in a single publication. Addressing RQ3.2, the primary feature for detecting obfuscated malware across platforms such as Windows, Android, and IoT is API calls. In the context of web platforms, the predominant feature is JavaScript functions.

b) ***Platform-Specific Obfuscated Malware Detection Approaches***

In this subsection, we provide a detailed taxonomy of obfuscated malware detection from the perspective of platforms. Our analysis of existing literature reveals that the detection of obfuscated malware was mainly performed on the Android platform, as shown in Figure 9. In addition, there are only a limited number of works that focus on Web and IoT platforms.

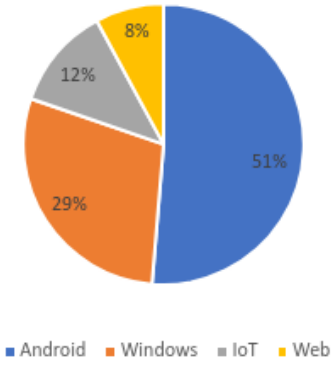

FIGURE 9. Prevalence of obfuscated malware detection in various platforms.

1) ANDROID

The authors in [36] examined the graphical features in Android application code by building weighted directed graphs of API calls. Their findings indicated that malicious apps frequently exhibited structural similarities, distinguishing them from non-malicious apps, even when the majority of applications in the training set are obfuscated. API calls and dependencies are extracted from the Android application package (APK). A novel approach was proposed by [37] to identify features that enhance obfuscation resilience, detection efficiency, and accuracy for the identification of Android malware families. In particular, they extracted APIs, as malware applications are required to call or access Android APIs to perform malicious actions such as stealing sensitive information, sending SMS messages for illegal financial gain, receiving instructions from a remote server, etc. Further, it extracts data flows between Android APIs that indicate possible information leakage. Malware belonging to distinct families is triggered by different actions of intent and messages exchanged between Android components. When data flows and Intent actions are insufficient, Android API usage information is added as a feature to help the classifier differentiate between malware families.

In Android, API calls and opcode sequences are widely used features for detecting obfuscated malware. Some approaches construct graphs based on these features to identify structural similarities with malicious applications. Other key features include permissions, XML file attributes, bytecode, metadata, certificates, cryptographic libraries, DEX-based features, and intents. In addition, many researchers leverage visualization techniques by generating application images and applying machine learning or deep learning models to enhance detection accuracy.

2) WINDOWS

Huda et al. [38] extracted API calls and essential machine-code features from Portable Executable (PE) files and analyzed the malicious behavior to create API call list statistics. Further, Windows APIs from the Microsoft Developer Network (MSDN) were downloaded and matched with those generated in the database for the malware sample collection. The authors in [88] used assembly opcode sequences collected at runtime, employing natural language processing and deep learning techniques to extract behavioral features. Alani et al. [89] proposed a lightweight detector for obfuscated malware that utilizes features extracted from memory dumps. The authors in [90] proposed a lightweight model that uses API call sequences and their arguments to detect obfuscated malware. The researchers in [91] detected obfuscated malware by analyzing behavioral features of Portable Executable (PE) files, such as execution time. The authors in [92] proposed a malware detection scheme that analyses the memory dump of processes. The authors in [93] decompressed the PE code, after which it is provided as input to the PE file parser for parsing, and the detection is based on Windows API calls.

In Windows, the most commonly used feature for detecting obfuscated malware is the Windows API calls extracted from Portable Executable (PE) files. Researchers also utilize assembly code, libraries, system functions, and memory dumps to improve detection capabilities.

3) WEB

The authors in [32] developed a classifier to detect malicious web pages and provide a comprehensive analysis report that includes information on geographical locations, images, and iframes. The researchers in [27] proposed a statistical approach to extract features and developed machine learning techniques to distinguish between malicious and benign JavaScript codes. The approach proposed in [21] constructs a structural representation from JavaScript code to gather contextual details. Features are then derived according to their specific context, and a machine learning model is employed to identify obfuscation. Gorji et al. [45] launched a set of harmful web pages into an actual web browser, capturing a sequence of anticipated function calls for each page through internal function debugging. Sequences are further grouped into clusters using their Normalized Levenshtein Distance (NLD), with a behavioral pattern created for every group. A web page is classified as harmful only when its captured function call sequence matches one of the created behavioral patterns. In summary, Web-based obfuscated malware detection primarily relies on analyzing JavaScript

code. HTML attributes and function calls are also used to identify malicious web pages.

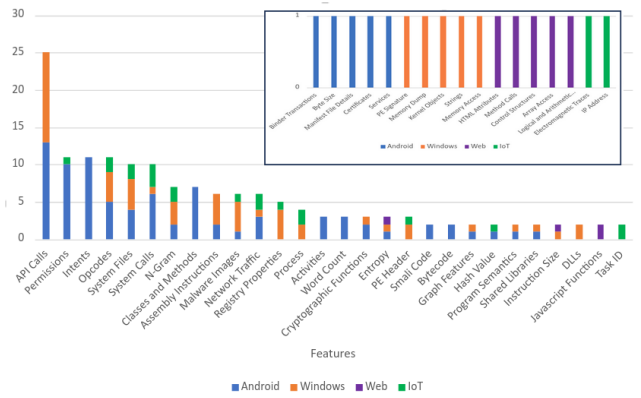

FIGURE 10. Overview of features across platforms.

4) IoT

In [34], the researchers introduced a technique to detect concealed malicious IoT applications by representing program binaries as Markov images. These images are processed through a deep learning model to identify hidden threats and classify the specific obfuscation techniques employed. Image-based malware classification using fine-tuned convolutional neural network architecture (IMCFN) was proposed in [33], combining malware visualization with a refined CNN architecture that utilizes the ImageNet dataset for training. The method was tested on a skewed IoT Android dataset comprising 14,733 harmful samples and 2,486 non-malicious samples. The experimental results showed that the IMCFN algorithm achieved an accuracy of 97.56%. Jeon et al. [94] utilized dynamic analysis to study IoT malware in a nested cloud environment, focusing on behaviors associated with memory, network, virtual file systems, processes, and system calls. The extracted data was transformed into images and analyzed using a CNN for the classification and training of IoT malware images. The authors in [95] created a methodology that utilizes raw electromagnetic traces as input to extract relevant information from the binary. Their analysis shows that this approach can effectively withstand packing, virtualization, and static code modification. The researchers in [96] employed an advanced Markov model to dynamically analyze code behaviors, enabling precise detection of malicious codes, even those obfuscated or concealed using various techniques. Tahir et al. [97] created a dataset by performing dynamic analysis on 524 malicious and legitimate samples across different IoT architectures. The collected system calls were then refined and employed to develop machine learning models. A hybrid malware detection scheme, HyMalD, was proposed in [98], which uses static and dynamic analysis to identify obfuscated malware. The main goal of HyMalD is to safeguard IoT devices and mitigate the detrimental impact of obfuscated malware.

In IoT, obfuscated malware detection often involves analyzing image representations of applications and generating datasets to train detection models. Researchers apply static, dynamic, and hybrid methods to identify malicious IoT applications.

Addressing RQ3.3, in Windows, Web, and IoT, the majority of approaches utilized customized datasets, which were created through the analysis of obfuscated malware. Conversely, in Android, the Android MalwareGenome Project [99] serves as the most widely used dataset for obfuscated malware detection.

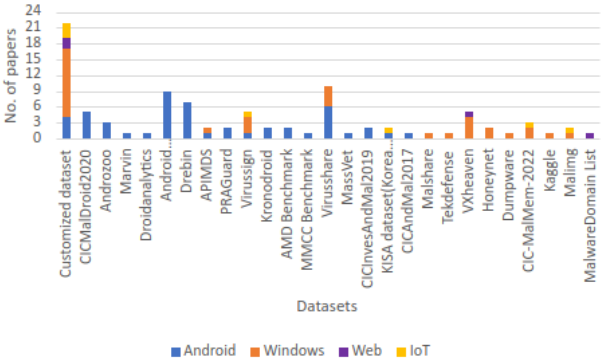

FIGURE 11. Frequency of malware dataset usage across platforms.

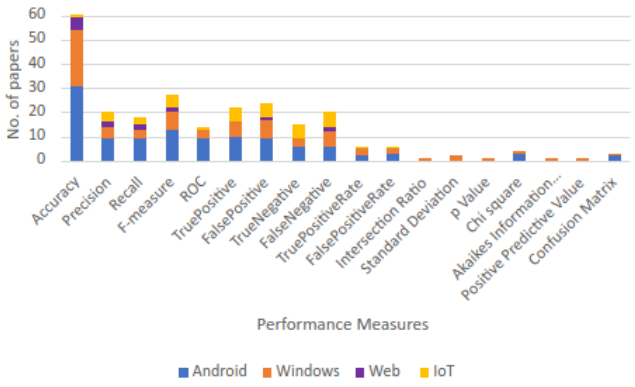

FIGURE 12. Performance metrics analysis across platforms.

D. PERFORMANCE EVALUATION OF OBFUSCATED MALWARE DETECTION MODELS

In the performance comparison, we observed varying detection rates across static, dynamic, hybrid, machine learning (ML), deep learning, and memory forensics-based detection techniques, as summarized in Table 13. Each approach demonstrated a unique trade-off between detection accuracy, false positive (FP) rates, and computational overhead.

Static analysis methods showed detection rates of 89%–99.9% with zero false positives in some cases but with higher

TABLE 13. Performance of malware detection techniques.

Technique	Detection (%)	FP (%)	Overhead
Static [24], [26], [36]	89–99.9	0–4.6	0.9–2.64s
Dynamic [42], [43], [46], [47]	85–99.8	0.1–1.2	0.2s–3min
Hybrid [50], [54], [56], [100]	97–99.7	0.03–0.2	0.2s–0.5s
ML [33], [66], [68], [69]	90–99.8	0.08–0.63	194ms–16s
DL [31], [34], [70], [94]	89.4–99.8	0.08–2.76	194ms–22s
Memory [82], [84], [85]	98.5–99.1	1.7	2–3min

computational overhead (0.9 to 2.64 seconds). Dynamic analysis had detection rates of 85%–99.8%, low false positives (0.1%), but longer detection times (0.2 seconds to 3 minutes), making it less suited for real-time use. Hybrid approaches, combining static and dynamic techniques, achieved detection rates of 97%–99.7%, with minimal false positives (0.03%–0.2%) and lower computational overhead (0.2 seconds to 7% CPU usage). Machine learning methods reached detection rates of 90%–99.8% with false positive rates of 0.08%–0.63%, varying in latency (194 ms to 16 seconds). Deep learning approaches followed similar patterns (89.4%–99.8% detection), but with higher computational cost (194 ms to 22 seconds). Memory forensics methods had strong detection rates (98.5%–99.1%) but higher false positives (1.7%) and longer processing times (2–3 minutes), making them more suitable for post-intrusion analysis. Overall, there is a trade-off between accuracy, speed, and resource efficiency across different methods.

While deep learning and ML models offer high accuracy, they demand greater computational power, whereas hybrid methods strike a balance between accuracy and efficiency. Memory forensics remains invaluable for in-depth forensic investigations, while dynamic and static analysis cater to different security needs depending on real-time constraints. Addressing RQ4, machine learning and deep learning techniques demonstrated competitive accuracy across different platforms, reaffirming their adaptability for modern malware detection challenges.

V. FUTURISTIC APPROACHES TO ACCELERATING SCALABLE OBFUSCATED MALWARE DETECTION

The threat of malware is greater than ever. Traditional security measures are often insufficient against sophisticated, obfuscated malware that hides its malicious intent. Combining the power of machine learning and deep learning with dynamic analysis can significantly enhance malware detection capabilities [100], [101], [102]. There is a critical need to investigate the effects of individual obfuscation methods and their combinations, especially for malware triggered by external events, such as receiving remote commands or specific time-based actions [55]. The use of memory forensics tools, such as Volatility [83], to extract relevant features from memory snapshots, coupled with machine learning algorithms, presents a promising approach to developing robust frameworks for obfuscated malware detection. Another emerging area is in the development of platform-independent memory forensics and deep learning based techniques to enhance detection capabilities [103]. Similarly, platform-independent image-based malware identification holds significant potential. Since malicious software can be embedded within small portions of image representations, identifying these affected regions is critical for accurate classification.

For Android platforms, semantic-based detection techniques require regular updates with newly labeled applications to avoid uncertain predictions [24]. Addressing this challenge involves implementing visualizations, ranking significant features, and developing automated feature selection and reduction algorithms. Additionally, research is needed to detect packed Android applications, as most existing methods do not account for such malware. Current approaches are limited to finite obfuscation techniques, such as API reflection, class name obfuscation, array encryption, and string encryption. Advanced obfuscation methods such as class encryption, junk code insertion, and code reordering require attention.

In the realm of web-based malware, enhancing the detection of obfuscated JavaScript code [32] is crucial, as many current methods rely primarily on common web attributes, which fail to detect malware not imported via iframes or JavaScript. Figure 13 highlights how network behavior analysis and behavioral sandboxing can improve web-based malware detection. Similarly, for Windows-based malware, PE header analysis is a promising area [25], but distinguishing legitimate applications with unusual PE structures from malware remains a challenge. AI-driven sandbox techniques in Figure 13 can improve the detection of malware leveraging anti-forensics and anti-virtualization techniques.

For IoT platforms, detection systems must function on resource-constrained devices. Studies on image-based techniques for detecting obfuscated IoT malware have demonstrated promise, as graphical signatures of malware samples can be used for classification [34], [87]. The lightweight behavioral monitoring approaches shown in Figure 13 provide scalable solutions tailored to IoT constraints.

In critical systems such as power grids, healthcare, and financial institutions, malware attacks can have devastating consequences. With the rise of emerging platforms like the Metaverse, Edge Computing, and Smart IoT ecosystems, new attack vectors demand innovative security solutions. To address RQ5, figure 13 presents a framework integrating digital twins (DTs) and Generative AI, which offers a scalable, AI-enhanced approach to malware detection.

1) LEVERAGING DIGITAL TWINS

Digital twins offer a powerful framework for simulating and safeguarding cyberphysical systems across domains such as safety-critical, mission-critical, business-critical, and noncritical systems as shown in Figure 13. These virtual replicas mirror real-world systems, allowing con-

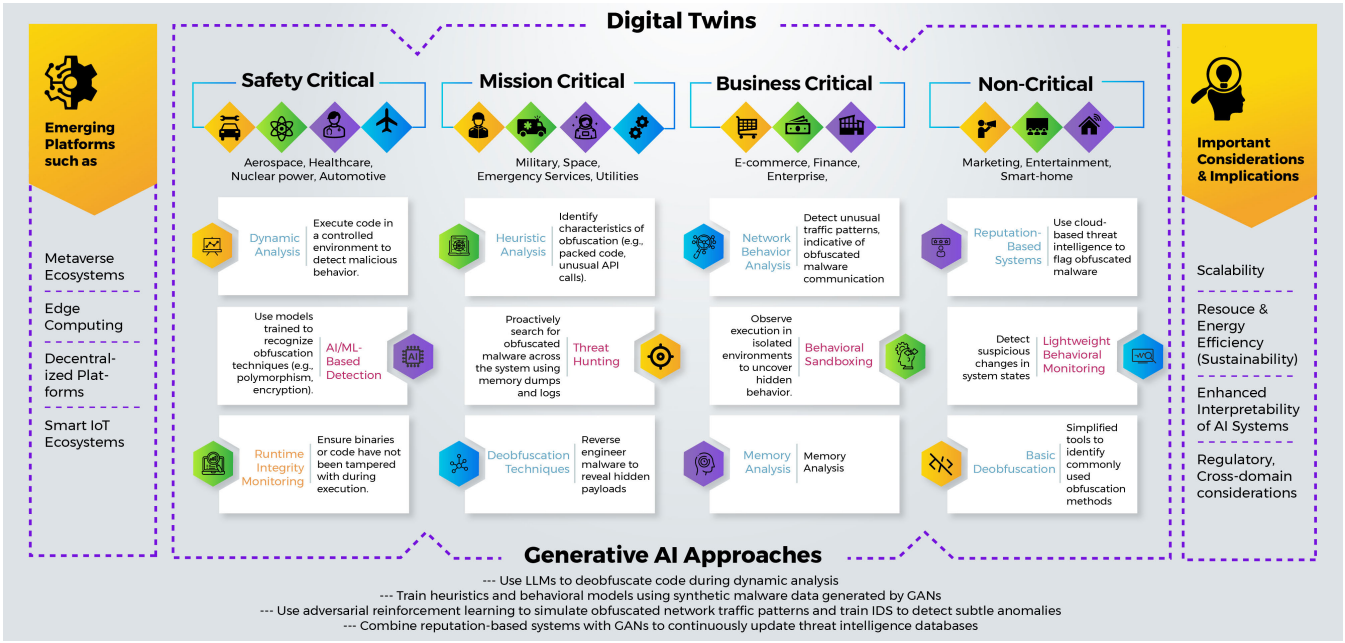

FIGURE 13. Framework to accelerate obfuscated malware detection.

trolled experimentation and analysis to detect and counteract obfuscated malware in an effective manner [104]. Given the lack of standardized datasets and benchmarks for evaluating DT based malware detection, future work should focus on creating a sandboxed DT simulation environment that mimics real-world systems. This involves injecting obfuscated malware samples, collecting system telemetry data (e.g., memory snapshots, API call sequences, network logs), and structuring datasets in reproducible formats. Establishing benchmarking protocols—comparing detection accuracy, latency, and adaptability against traditional AI-based methods—will enable empirical validation. Open-access repositories should store datasets, ensuring broad accessibility and reproducibility.

Future research should focus on constructing a controlled digital twin environment designed to simulate real-world malware behaviors, including polymorphic, metamorphic, and encrypted obfuscation techniques. This simulation can be enhanced by incorporating real-world malware samples, monitoring their interactions within various system architectures, and observing evasion tactics used against AI-based detection systems. Additionally, incorporating feedback loops within the DT framework would enable continuous refinement, allowing for adaptive learning in detecting novel malware strains.

For safety-critical systems such as aerospace, healthcare, nuclear power, and automotive, DTs must emphasize run-time integrity monitoring as well as ensuring that code/binaries remain unaltered during execution. Techniques such as dynamic analysis, where code is executed in a controlled environment, can uncover malicious behaviour. These twins should integrate robust AI/ML-based detection models trained to identify obfuscation techniques such as polymorphism and encryption, minimizing risks in environments where failure is not an option. In mission-critical systems such as military, space, emergency services, and utilities, DTs must support heuristic analysis to identify obfuscation traits such as packed code or unusual API calls. Combined with threat-hunting methods that analyze memory dumps and logs, these twins enable proactive malware detection. Tools for de-obfuscation, including reverse-engineering techniques, can help reveal hidden payloads, ensuring thorough threat analysis. For business-critical systems such as e-commerce, finance, and enterprise networks, DTs must focus on network behaviour analysis to detect anomalies such as unusual traffic patterns. Similarly, features such as behavioural sandboxing simulate isolated environments to observe malware execution and uncover hidden behaviours. Memory analysis within these twins helps detect in-memory indicators of obfuscated malware, enhancing detection accuracy [76]. In non-critical systems such as marketing, entertainment, and smart home setups, lightweight twins prioritize cost-effective and scalable detection methods. Reputation-based systems use cloud-based threat intelligence to flag obfuscated malware, while basic de-obfuscation tools target commonly used obfuscation methods. Lightweight behavioural monitoring detects suspicious changes in system states without extensive computational overhead [105] The proposed workflow for the framework (Figure 13), will include five distinct phases: malware detection initiation, detection and analysis, threat classification and adaptive learning, threat response and mitigation, and continuous update through threat intelligence integration. The process begins with DTs simulating real-world computing

environments, continuously monitoring system behaviors to detect anomalies. Once obfuscated malware is suspected, various detection techniques—including dynamic analysis, heuristic scanning, network traffic inspection, and memory forensics—are applied to analyze the threat. Generative AI plays a crucial role by simulating potential obfuscation techniques, improving detection accuracy. Following detection, the system classifies threats into known or novel malware families. If new obfuscation techniques are discovered, adaptive learning mechanisms refine AI-based detection models, ensuring continuous improvement. The response phase varies depending on the system category: safety-critical systems require immediate automated mitigation, while business-critical systems focus on behavioral sandboxing and anomaly tracking. In non-critical systems, a combination of cloud-based reputation analysis and lightweight behavioral monitoring provides an efficient yet scalable solution.

2) HARNESSING GENERATIVE AI AND ADVANCED MODELS

Generative Adversarial Networks (GANs) have emerged as a powerful tool for addressing obfuscated malware. GANs can generate synthetic malware signatures that mimic real-world obfuscated patterns, significantly enhancing training datasets and the robustness of detection systems. In Figure 13, Generative AI enhances malware detection by leveraging advanced machine learning techniques to address complex obfuscation strategies. One of its primary applications is automated code de-obfuscation, where LLMs analyze and reconstruct obfuscated malware code. By recognizing patterns in obfuscation techniques such as polymorphism, encryption, and junk code insertion, LLMs enable faster and more accurate identification of malicious intent within executable files. This significantly reduces the time and effort required for manual reverse engineering, making threat analysis more efficient. Another critical application is the simulation of obfuscated network traffic using GANs. Cyber attackers often employ network-layer obfuscation techniques, such as packet fragmentation, protocol mimicry, and encrypted tunneling, to evade detection by IDS. GANs can generate realistic but adversarial traffic patterns, training IDS models to distinguish between normal and obfuscated malicious traffic more effectively. By continuously refining detection rules, this approach enhances network security against APTs and stealthy malware campaigns. Furthermore, adversarial reinforcement learning plays a crucial role in generating novel obfuscation patterns for testing detection models. Instead of relying solely on known malware signatures, this method simulates real-world attack evolution by introducing adaptive obfuscation techniques that challenge security systems. As a result, malware detection frameworks become more resilient to zero-day attacks and sophisticated evasion tactics. Additionally, reputation-based systems integrated with GANs provide a continuous update mechanism for threat intelligence databases. Traditional threat intelligence relies on static blacklists and rule-based detection, which can become outdated as new threats emerge. By leveraging GAN-generated malware samples, reputation-based systems can dynamically classify and flag emerging threats, reducing the risk of false negatives and improving response times. This integration ensures that security frameworks remain proactive rather than reactive, continuously adapting to new cyber threats. This integration of Digital Twins with Generative AI allows for continuously improving the detection of complex obfuscated malware over time. For instance, [106] Boundary-Seeking GANs address challenges such as mode collapse, ensuring the generation of high-quality training data. This approach integrates a combination of CNN and LSTM models to capture both short- and long-term dependencies in malware data, enhancing detection accuracy through a multi-layered approach that is resilient to obfuscation techniques. Additionally, hybrid deep-generative models combine global features, such as malware image transformation with local binary code sequences Kim et al. [35]. This approach effectively detects malware variants by leveraging techniques such as class activation mapping (CAM), which enhances interpretability and detection accuracy. Adversarial Deep Reinforcement Learning (DRL) systems, Sewak et al. [107], such as ADVERSARIALuscator, simulate opcode-level obfuscations to create realistic malware variants. These systems model malware generation as a Markov Decision Process (MDP), enabling the creation of diverse malware swarms that train Intrusion Detection Systems (IDS) to counter advanced obfuscation techniques. By preparing IDS to handle metamorphic and polymorphic malware, DRL strengthens the defense mechanisms against sophisticated obfuscation at the opcode level.

LLMs demonstrate potential in de-obfuscating malware by understanding complex structures and reversing obfuscation patterns Patsakis et al. [108]. While not yet fully accurate, LLMs exhibit promising capabilities in supporting AI-driven threat intelligence and malware analysis workflows. Their integration into threat intelligence pipelines can streamline detection processes and enhance the system's ability to adapt to evolving obfuscation techniques. Finally, Arifin et al. [109] conducted a comprehensive survey on the application of GANs in cybersecurity, highlighting their use in Intrusion Detection Systems (IDS), botnet detection, and malware detection. This study emphasizes the potential of GANs to strengthen defenses in digital landscapes despite challenges and limitations that require further investigation. A crucial next step is conducting a comparative analysis between the proposed Generative AI and Digital Twin-based approach (Figure 13) and conventional AI-driven malware detection techniques, such as machine learning-based classifiers, deep learning models, and signature-based detection systems. This benchmarking would focus on key performance indicators such as accuracy, detection latency, false positive rates, and computational efficiency. Evaluating the proposed framework in contrast with well-established detection methodologies would offer quantifiable evidence of its potential benefits and limitations. Additionally, this

comparison would highlight how well the framework generalizes across different obfuscation techniques and computing environments (Windows, Android, IoT, Web).

3) BUILDING SCALABLE AND SUSTAINABLE FRAMEWORKS

Scalability and sustainability are critical considerations in designing effective systems for detecting obfuscated malware. As the volume of data, connected devices, and sophisticated threats grows, detection frameworks must adapt while remaining environmentally responsible. Scalability ensures that detection systems can expand to meet increasing demands. Integrating cloud and edge computing is a key solution; cloud platforms offer centralized scalability for processing large volumes of data, while edge computing enables real-time detection in resource-constrained environments such as IoT devices. Lightweight AI models, including pruned and quantized versions, reduce computational overhead while maintaining accuracy, making them ideal for scalable deployments [110]. In addition, federated learning allows decentralized training of AI models across multiple locations, eliminating the need for centralized data collection and ensuring both scalability and privacy [111]. Modular digital twins further enhance scalability by focusing simulations on high-risk components rather than entire systems, thus optimizing resource allocation [112].

Sustainability focuses on reducing the environmental impact of detection systems [113]. Energy-efficient AI frameworks are of paramount importance to minimize energy consumption during model training and inference. Hosting detection platforms on green cloud services powered by renewable energy further reduces carbon emissions. Resource optimization in digital twins is another key strategy; by prioritizing dynamic simulations over continuous ones, energy use can be significantly reduced. Incorporating lifecycle thinking into system design ensures that hardware and software are modular and durable, reducing e-waste and enabling long-term sustainability [114].

To combine scalability and sustainability, systems must balance performance and efficiency. Hybrid solutions that integrate cloud and edge computing offer both low latency and reduced energy demands. Generative AI can streamline processes, reducing manual intervention and computational resources [115]. In addition, shared digital twin platforms and collaborative research initiatives can prevent duplication of efforts, conserving energy and resources while fostering innovation [116].

VI. CONCLUSION

This paper has presented a comprehensive review of over 75 obfuscated malware detection methodologies across Windows, Android, IoT, and Web platforms over the past decade. In particular, our work has emphasized the importance of obfuscated malware typically used by attackers to evade detection. Our study comprehensively addresses five major categories of detection techniques across four diverse platforms commonly targeted by obfuscated malware. Within these categorizations, we have thoroughly examined sixty-five distinct detection methodologies, as well as eleven specialized methods such as detection based on API calls, opcodes, analysing PE files, extracting contextual information from javascript code, analysing application images etc., resulting in a total of seventy-six methods tailored specifically for the platforms under consideration. In addressing the challenge of detecting obfuscated malware, we meticulously examined distinct obfuscation tactics, including polymorphism, encryption, and packing, to gain a deeper understanding of each method's intricacies. In contrast to existing work that focuses on techniques, this platform-specific analysis is critical as obfuscation techniques and the effectiveness of detection can vary significantly between environments.

The detection techniques examined included static, dynamic, hybrid, machine learning, and deep learning approaches. Static analysis techniques constituted a significant portion, accounting for 27% of the research, while dynamic analysis closely follows at 23%. Importantly, we have highlighted the role of memory forensics in detecting obfuscated malware, an unexplored area in most prior work. Memory forensics complements traditional detection methods, offering a new dimension to the detection and analysis of obfuscated malware. Furthermore, we explored the integration of artificial intelligence, including both machine learning and deep learning, with traditional detection methodologies. Notably, we expanded this work to explore the role of AI in detecting ransomware, spyware, and fileless malware toward enhancing cybersecurity defenses. Notably, the integration of AI-based techniques, such as machine learning and deep learning outperforms static and dynamic analysis, contributing about 32% of existing work. In particular, ensemble learning-based algorithms have emerged as prominent strategies for enhancing detection efficiency.

Within the context of platforms, obfuscated malware detection on Windows contributed towards 51% of the research, followed by Android at 29%, while the Web platform received relatively less emphasis. Across all platforms, accuracy served as the primary performance metric, reflecting its universal importance in assessing detection efficacy. In particular, our analysis reveals that static analysis demonstrated higher accuracy, 96% for obfuscated malware detection in Web-based platforms, while IoT and Windows-based platforms leveraged AI-based techniques for superior performance with 99.8% and 99.82% accuracy respectively. However, both hybrid analysis and AI-based approaches performed with 99% accuracy in Android platforms. Finally, our findings highlight the importance of features such as API calls, system calls for Windows, Android, and IoT environments, while JavaScript functions hold significance in the Web platform in detecting obfuscated malware.

Despite the plethora of detection techniques available, our research identifies several crucial open research challenges in this field along with future directions, which can guide the development of future research depending on the requirements of individual platforms. In conclusion, while various techniques exist for obfuscated malware detection, there remains a need for a platform-independent method capable of detecting various kinds of obfuscation techniques effectively. By addressing platform-specific challenges, incorporating AI's role in detecting advanced threats like ransomware, spyware, and fileless malware, and leveraging emerging technologies like digital twins and Generative AI, obfuscated malware detection can be significantly accelerated.

REFERENCES

[1] K. Achuthan, S. Khobragade, and R. Kowalski, "Public sentiment and engagement on cybersecurity: Insights from Reddit discussions," *Comput. Human Behav. Rep.*, vol. 17, Mar. 2025, Art. no. 100573.

[2] K. Achuthan, S. Khobragade, and R. Kowalski, "Cybercrime through the public lens: A longitudinal analysis," *Humanities Social Sci. Commun.*, vol. 12, no. 1, pp. 1–16, Mar. 2025.

[3] ExtraHop. (2020). *Malware Obfuscation: Techniques, Definition and Detection*. [Online]. Available: https://www.extrahop.com/resources/attacks/

[4] J. Martínez and J. M. Durán, "Software supply chain attacks, a threat to global cybersecurity: SolarWinds' case study," *Int. J. Saf. Secur. Eng.*, vol. 11, no. 5, pp. 537–545, Oct. 2021.

[5] CISA. (2022). *2021 Top Malware Strains: Cybersecurity and Infrastructure Security Agency CISA*. [Online]. Available: https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-216a

[6] IBM Secur. (2022). *X-Force Threat Intelligence Index 2022*. [Online]. Available: https://www.ibm.com/downloads/cas/ADLMYLAZ

[7] L. Caviglione, M. Choras, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska, "Tight arms race: Overview of current malware threats and trends in their detection," *IEEE Access*, vol. 9, pp. 5371–5396, 2021.

[8] W. F. Elsersy, A. Feizollah, and N. B. Anuar, "The rise of obfuscated Android malware and impacts on detection methods," *PeerJ Comput. Sci.*, vol. 8, p. e907, Mar. 2022.

[9] H. M. Deylami, R. C. Muniyandi, I. T. Ardekani, and A. Sarrafzadeh, "Taxonomy of malware detection techniques: A systematic literature review," in *Proc. 14th Annu. Conf. Privacy, Secur. Trust (PST)*, Dec. 2016, pp. 629–636.

[10] A. Abusitta, M. Q. Li, and B. C. M. Fung, "Malware classification and composition analysis: A survey of recent developments," *J. Inf. Secur. Appl.*, vol. 59, Jun. 2021, Art. no. 102828.

[11] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[12] T. Alsmadi and N. Alqudah, "A survey on malware detection techniques," in *Proc. Int. Conf. Inf. Technol. (ICIT)*, Jul. 2021, pp. 371–376.

[13] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *J. Netw. Comput. Appl.*, vol. 153, Mar. 2020, Art. no. 102526.

[14] R. Tahir, "A study on malware and malware detection techniques," *Int. J. Educ. Manage. Eng.*, vol. 8, no. 2, pp. 20–30, Mar. 2018.

[15] R. Sihwail, K. Omar, and K. A. Zainol Ariffin, "A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 8, nos. 2–4, pp. 1662–1671, Sep. 2018.

[16] S. K. Sahay, A. Sharma, and H. Rathore, "Evolution of malware and its detection techniques," in *Proc. Inf. Commun. Technol. Sustain. Develop. (ICT4SD)*. Cham, Switzerland: Springer, Jun. 2019, pp. 139–150.

[17] A. Cannarile, V. Dentamaro, S. Galantucci, A. Iannacone, D. Impedovo, and G. Pirlo, "Comparing deep learning and shallow learning techniques for API calls malware prediction: A study," *Appl. Sci.*, vol. 12, no. 3, p. 1645, Feb. 2022.

[18] U. Urooj, B. A. S. Al-Rimy, A. Zainal, F. A. Ghaleb, and M. A. Rassam, "Ransomware detection using the dynamic analysis and machine learning: A survey and research directions," *Appl. Sci.*, vol. 12, no. 1, p. 172, Dec. 2021.

[19] A. P. Namanya, A. Cullen, I. U. Awan, and J. P. Disso, "The world of malware: An overview," in *Proc. IEEE 6th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2018, pp. 420–427.

[20] S. Sechel, "A comparative assessment of obfuscated ransomware detection methods," *Inf. Economica*, vol. 23, no. 2, pp. 45–62, Jun. 2019.

[21] I. A. Al-Taharwa, H.-M. Lee, A. B. Jeng, K.-P. Wu, C.-S. Ho, and S.-M. Chen, "JSOD: Javascript obfuscation detector," *Secur. Commun. Netw.*, vol. 8, no. 6, pp. 1092–1107, Apr. 2015.

[22] D. Moher, A. Liberati, J. Tetzlaff, and D. G. Altman, "Preferred reporting items for systematic reviews and meta-analyses: The PRISMA statement," *PLoS Med.*, vol. 6, no. 7, Jul. 2009, Art. no. e1000097.

[23] D. Xu, J. Ming, and D. Wu, "Cryptographic function detection in obfuscated binaries via bit-precise symbolic loop mapping," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 921–937.

[24] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: Multi-layer Android malware detection system applying deep neural networks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 473–487.

[25] M. Saleh, E. P. Ratazzi, and S. Xu, "Instructions-based detection of sophisticated obfuscation and packing," in *Proc. IEEE Mil. Commun. Conf.*, Oct. 2014, pp. 1–6.

[26] P. C. Ouk and W. Pak, "Unified detection of obfuscated and native Android malware," *Comput., Mater. Continua*, vol. 70, no. 2, pp. 3099–3116, 2022.

[27] R. Kumar and A. R. E. Vaishakh, "Detection of obfuscation in Java malware," *Proc. Comput. Sci.*, vol. 78, pp. 521–529, Aug. 2016.

[28] S. Millar, N. McLaughlin, J. Martinez del Rincon, P. Miller, and Z. Zhao, "DANdroid: A multi-view discriminative adversarial network for obfuscated Android malware detection," in *Proc. 10th ACM Conf. Data Appl. Secur. Privacy*, Mar. 2020, pp. 353–364.

[29] S. Khalid and F. B. Hussain, "Evaluating opcodes for detection of obfuscated Android malware," in *Proc. Int. Conf. Artif. Intell. Inf. Commun. (ICAIIC)*, Feb. 2022, pp. 44–49.

[30] S. Banin, A. Shalaginov, and K. Franke, "Memory access patterns for malware detection," Tech. Rep., 2016.

[31] J. Tang, R. Li, Y. Jiang, X. Gu, and Y. Li, "Android malware obfuscation variants detection method based on multi-granularity opcode features," *Future Gener. Comput. Syst.*, vol. 129, pp. 141–151, Apr. 2022.

[32] P. Seshagiri, A. Vazhayil, and P. Sriram, "AMA: Static code analysis of Web page for the detection of malicious scripts," *Proc. Comput. Sci.*, vol. 93, pp. 768–773, Jan. 2016.

[33] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Comput. Netw.*, vol. 171, Apr. 2020, Art. no. 107138.

[34] K. A. Dhanya, P. Vinod, S. Y. Yerima, A. Bashar, A. David, T. Abhiram, A. Antony, A. K. Shavanas, and T. G. Kumar, "Obfuscated malware detection in IoT Android applications using Markov images and CNN," *IEEE Syst. J.*, vol. 17, no. 2, pp. 2756–2766, Jun. 2023.

[35] J.-Y. Kim and S.-B. Cho, "Obfuscated malware detection using deep generative model based on global/local features," *Comput. Secur.*, vol. 112, Jan. 2022, Art. no. 102501.

[36] M. Ikram, P. Beaume, and M. Ali Kaafar, "DaDiDroid: An obfuscation resilient tool for detecting Android malware via weighted directed call graph modelling," 2019, *arXiv:1905.09136*.

[37] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S. Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of Android malware," Dept. Comput. Sci., George Mason Univ., Fairfax, VA, USA, Tech. Rep., 2015, vol. 202.

[38] S. Huda, J. Abawajy, M. Alazab, M. Abdollalihian, R. Islam, and J. Yearwood, "Hybrids of support vector machine wrapper and filter based framework for malware detection," *Future Gener. Comput. Syst.*, vol. 55, pp. 376–390, Feb. 2016.

[39] A. Hellal and L. Ben Romdhane, "Minimal contrast frequent pattern mining for malware detection," *Comput. Secur.*, vol. 62, pp. 19–32, Sep. 2016.

[40] Z. Li, J. Sun, Q. Yan, W. Srisa-An, and Y. Tsutano, "Obfusifier: Obfuscation-resistant Android malware detection system," in *Proc. 15th EAI Int. Conf. Secur. Privacy Commun. Netw.*, Orlando, FL, USA. Cham, Switzerland: Springer, Jan. 2019, pp. 214–234.

[41] A. G. Kakisim, M. Nar, and I. Sogukpinar, "Metamorphic malware identification using engine-specific patterns based on co-opcode graphs," *Comput. Standards Interfaces*, vol. 71, Aug. 2020, Art. no. 103443.

[42] Z.-G. Chen, H.-S. Kang, S.-N. Yin, and S.-R. Kim, "Automatic ransomware detection and analysis based on dynamic API calls flow graph," in *Proc. Int. Conf. Res. Adapt. Convergent Syst.*, Sep. 2017, pp. 196–201.

[43] D. Kim, A. Majlesi-Kupaei, J. Roy, K. Anand, K. ElWazeer, D. Buettner, and R. Barua, "DynODet: Detecting dynamic obfuscation in malware," in *Proc. 14th Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Bonn, Germany. Cham, Switzerland: Springer, Jan. 2017, pp. 97–118.

[44] Y. Ki, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *Int. J. Distrib. Sensor Netw.*, vol. 11, no. 6, Jun. 2015, Art. no. 659101.

[45] A. Gorji and M. Abadi, "Detecting obfuscated Javascript malware using sequences of internal function calls," in *Proc. ACM Southeast Regional Conf.*, Mar. 2014, pp. 1–6.

[46] A. Irshad and M. K. Dutta, "Identification of windows-based malware by dynamic analysis using machine learning algorithm," in *Proc. Adv. Comput. Intell. Commun. Technol.* Cham, Switzerland: Springer, 2021, pp. 207–218.

[47] G. Suarez-Tangil, J. E. Tapiador, F. Lombardi, and R. Di Pietro, "Thwarting obfuscated malware via differential fault analysis," *Computer*, vol. 47, no. 6, pp. 24–31, Jun. 2014.

[48] K. Khanmohammadi and A. Hamou-Lhadj, "HyDroid: A hybrid approach for generating API call traces from obfuscated Android applications for mobile security," in *Proc. IEEE Int. Conf. Softw. Quality, Rel. Secur. (QRS)*, Jul. 2017, pp. 168–175.

[49] M. Sun, X. Li, J. C. S. Lui, R. T. B. Ma, and Z. Liang, "Monet: A user-oriented behavior-based malware variants detection system for android," *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 5, pp. 1103–1112, May 2017.

[50] C. Ding, N. Luktarhan, B. Lu, and W. Zhang, "A hybrid analysis-based approach to Android malware family classification," *Entropy*, vol. 23, no. 8, p. 1009, Aug. 2021.

[51] H. M. Kim, H. M. Song, J. W. Seo, and H. K. Kim, "Andro-simnet: Android malware family classification using social network analysis," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–8.

[52] Y. Li, F. Liu, Z. Du, and D. Zhang, "A simhash-based integrative features extraction algorithm for malware detection," *Algorithms*, vol. 11, no. 8, p. 124, Aug. 2018.

[53] M. V. Ngo, T. Truong-Huu, D. Rabadi, J. Y. Loo, and S. G. Teo, "Fast and efficient malware detection with joint static and dynamic features through transfer learning," in *Proc. Int. Conf. Appl. Cryptography Netw. Secur.* Cham, Switzerland: Springer, Jan. 2023, pp. 503–531.

[54] D. Vidyarthi, S. P. Choudhary, S. Rakshit, and C. R. S. Kumar, "Malware detection by static checking and dynamic analysis of executables," *Int. J. Inf. Secur. Privacy*, vol. 11, no. 3, pp. 29–41, Jul. 2017.

[55] H. S. Galal, Y. B. Mahdy, and M. A. Atiea, "Behavior-based features model for malware detection," *J. Comput. Virol. Hacking Techn.*, vol. 12, no. 2, pp. 59–67, May 2016.

[56] W. Han, J. Xue, Y. Wang, Z. Liu, and Z. Kong, "MalInsight: A systematic profiling based malware detection framework," *J. Netw. Comput. Appl.*, vol. 125, pp. 236–250, Jan. 2019.

[57] S. S. Hansen, T. M. T. Larsen, M. Stevanovic, and J. M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2016, pp. 1–5.

[58] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic Android malware detection system with ensemble learning," *IEEE Access*, vol. 6, pp. 30996–31011, 2018.

[59] L. Wen and H. Yu, "An Android malware detection system based on machine learning," in *Proc. AIP Conf.*, vol. 1864, 2017, pp. 1–11.

[60] S. K. Shaukat and V. J. Ribeiro, "RansomWall: A layered defense system against cryptographic ransomware attacks using machine learning," in *Proc. 10th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2018, pp. 356–363.

[61] M. Ali, S. Shiaeles, G. Bendiab, and B. Ghita, "MALGRA: Machine learning and N-Gram malware feature extraction and detection system," *Electronics*, vol. 9, no. 11, p. 1777, Oct. 2020.

[62] S. Venkatraman and M. Alazab, "Use of data visualisation for zero-day malware detection," *Secur. Commun. Netw.*, vol. 2018, pp. 1–13, Dec. 2018.

[63] D. Xue, J. Li, T. Lv, W. Wu, and J. Wang, "Malware classification using probability scoring and machine learning," *IEEE Access*, vol. 7, pp. 91641–91656, 2019.

[64] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android malware detection based on a hybrid deep learning model," *Secur. Commun. Netw.*, vol. 2020, pp. 1–11, Aug. 2020.

[65] M. V. Ngo, T. Truong-Huu, D. Rabadi, J. Yi Loo, and S. G. Teo, "Fast and efficient malware detection with joint static and dynamic features through transfer learning," 2022, *arXiv:2211.13860*.

[66] N. Singh, S. Tripathy, and B. Bezawada, "SHIELD: A multimodal deep learning framework for Android malware detection," in *Proc. 18th Int. Conf. Inf. Syst. Secur. (ICISS)*, Tirupati, India. Cham, Switzerland: Springer, Jan. 2022, pp. 64–83.

[67] A. Darem, J. Abawajy, A. Makkar, A. Alhashmi, and S. Alanazi, "Visualization and deep-learning-based malware variant detection using OpCode-level features," *Future Gener. Comput. Syst.*, vol. 125, pp. 314–323, Dec. 2021.

[68] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic Android malware category classification using semi-supervised deep learning," in *Proc. IEEE Int. Conf. Dependable, Autonomic Secure Comput., Int. Conf. Pervasive Intell. Comput., Int. Conf. Cloud Big Data Comput., Int. Conf. Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2020, pp. 515–522.

[69] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, "Deep learning for classification of malware system call sequences," in *Proc. Aust. Joint Conf. Artif. Intell.*, Hobart, TAS, Australia. Cham, Switzerland: Springer, 2016, pp. 137–149.

[70] X. Huang, L. Ma, W. Yang, and Y. Zhong, "A method for windows malware detection based on deep learning," *J. Signal Process. Syst.*, vol. 93, nos. 2–3, pp. 265–273, Mar. 2021.

[71] S. Venkatraman, M. Alazab, and R. Vinayakumar, "A hybrid deep learning image-based analysis for effective malware detection," *J. Inf. Secur. Appl.*, vol. 47, pp. 377–389, Aug. 2019.

[72] W. Y. Lee, J. Saxe, and R. Harang, "SeqDroid: Obfuscated Android malware detection using stacked convolutional and recurrent neural networks," in *Deep Learning Applications for Cyber Security*, 2019, pp. 197–210.

[73] M. Asam, S. J. Hussain, M. Mohatram, S. H. Khan, T. Jamal, A. Zafar, A. Khan, M. U. Ali, and U. Zahoora, "Detection of exceptional malware variants using deep boosted feature spaces and machine learning," *Appl. Sci.*, vol. 11, no. 21, p. 10464, Nov. 2021.

[74] M. M. Abualhaj, M. Al-Zyoud, M. O. Hiari, Y. Alrabanah, M. Anbar, A. Amer, and A. Al-Allawee, "A fine-tuning of decision tree classifier for ransomware detection based on memory data," *Int. J. Data Netw. Sci.*, vol. 8, no. 2, pp. 733–742, 2024.

[75] H. Kaur, V. Kumar, and A. Daramas, "Detecting ransomware using system calls through transfer learning on a limited feature set," in *Proc. Int. Conf. Web Inf. Syst. Eng.* Cham, Switzerland: Springer, Nov. 2024, pp. 426–440.

[76] W. Zanoramy, M. F. Abdollah, O. Abdollah, and S. M. W. Mohamed, "Ransomware early detection using machine learning approach and pre-encryption boundary identification," *J. Adv. Res. Appl. Sci. Eng. Technol.*, vol. 47, no. 2, pp. 121–137, Jun. 2024.

[77] M. U. R. Shaikh, M. F. Hassan, R. Akbar, K. S. Savita, R. Ullah, and S. Mandala, "Ransomware classification with deep neural network and bi-LSTM," *J. Adv. Res. Appl. Sci. Eng. Technol.*, vol. 47, no. 2, pp. 266–280, Jun. 2024.

[78] M. M. Abualhaj, A. S. Al-Shamayleh, A. Munther, S. N. Alkhatib, M. O. Hiari, and M. Anbar, "Enhancing spyware detection by utilizing decision trees with hyperparameter optimization," *Bull. Electr. Eng. Informat.*, vol. 13, no. 5, pp. 3653–3662, Oct. 2024.

[79] D. Javaheri, M. Hosseinzadeh, and A. M. Rahmani, "Detection and elimination of spyware and ransomware by intercepting kernel-level system routines," *IEEE Access*, vol. 6, pp. 78321–78332, 2018.

[80] M.-H. Wu, F.-H. Hsu, J.-H. Hunag, K. Wang, Y.-Y. Liu, J.-X. Chen, H.-J. Wang, and H.-T. Yang, "MPSD: A robust defense mechanism against malicious PowerShell scripts in windows systems," *Electronics*, vol. 13, no. 18, p. 3717, Sep. 2024.

[81] X. Yang, G. Peng, D. Zhang, Y. Gao, and C. Li, "PowerDetector: Malicious PowerShell script family classification based on multi-modal semantic fusion and deep learning," *China Commun.*, vol. 20, no. 11, pp. 202–224, Nov. 2023.

[82] R. Sihwail, K. Omar, K. Zainol Ariffin, and S. Al Afghani, "Malware detection approach based on artifacts in memory image and dynamic analysis," *Appl. Sci.*, vol. 9, no. 18, p. 3680, Sep. 2019.

[83] (2020). *Volatility*. [Online]. Available: https://github.com/volatility foundation/volatility

[84] K. M. Yogesh, S. Arpitha, T. Stephan, M. Praksha, and V. K. Raghu, "Unravelling obfuscated malware through memory feature engineering and ensemble learning," in *Proc. Int. Conf. Inf. Commun. Technol. Competitive Strategies*. Cham, Switzerland: Springer, Jan. 2024, pp. 323–332.

[85] S. M. Rakib Hasan and A. Dhakal, "Obfuscated malware detection: Investigating real-world scenarios through memory analysis," in *Proc. IEEE Int. Conf. Telecommun. Photon. (ICTP)*, Dec. 2023, pp. 1–5.

[86] T. Carrier, P. Victor, A. Tekeoglu, and A. Lashkari, "Detecting obfuscated malware using memory feature engineering," in *Proc. 8th Int. Conf. Inf. Syst. Secur. Privacy*, 2022, pp. 177–188.

[87] S. S. Shafin, G. Karmakar, and I. Mareels, "Obfuscated memory malware detection in resource-constrained IoT devices for smart city applications," *Sensors*, vol. 23, no. 11, p. 5348, Jun. 2023.

[88] E. S. Parildi, D. Hatzinakos, and Y. Lawryshyn, "Deep learning-aided runtime opcode-based windows malware detection," *Neural Comput. Appl.*, vol. 33, no. 18, pp. 11963–11983, Sep. 2021.

[89] M. M. Alani, A. Mashatan, and A. Miri, "XMal: A lightweight memory-based explainable obfuscated-malware detector," *Comput. Secur.*, vol. 133, Oct. 2023, Art. no. 103409.

[90] D. Rabadi and S. G. Teo, "Advanced windows methods on malware detection and classification," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2020, pp. 54–68.

[91] S. L. S. Darshan and C. D. Jaidhar, "Windows malware detector using convolutional neural network based on visualization images," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 2, pp. 1057–1069, Apr. 2021.

[92] M. R. Naeem, M. Khan, A. M. Abdullah, F. Noor, M. I. Khan, M. A. Khan, I. Ullah, and S. Room, "A malware detection scheme via smart memory forensics for windows devices," *Mobile Inf. Syst.*, vol. 2022, pp. 1–16, Oct. 2022.

[93] M. K. Shankarapani, S. Ramamoorthy, R. S. Movva, and S. Mukkamala, "Malware detection using assembly and API call sequences," *J. Comput. Virology*, vol. 7, no. 2, pp. 107–119, May 2011.

[94] J. Jeon, J. H. Park, and Y.-S. Jeong, "Dynamic analysis for IoT malware detection with convolution neural network model," *IEEE Access*, vol. 8, pp. 96899–96911, 2020.

[95] D.-P. Pham, D. Marion, M. Mastio, and A. Heuser, "Obfuscation revealed: Leveraging electromagnetic signals for obfuscated malware classification," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2021, pp. 706–719.

[96] N. Abanmi, H. Kurdi, and M. Alzamel, "Dynamic IoT malware detection in Android systems using profile hidden Markov models," *Appl. Sci.*, vol. 13, no. 1, p. 557, Dec. 2022.

[97] I. Tahir and S. Qadir, "Machine learning-based detection of IoT malware using system call data," in *Proc. 4th Int. Conf. Digit. Futures Transformative Technol. (ICoDT2)*, Oct. 2024, pp. 1–8.

[98] J. Jeon, B. Jeong, S. Baek, and Y.-S. Jeong, "Hybrid malware detection based on bi-LSTM and SPP-net for smart IoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 7, pp. 4830–4837, Jul. 2022.

[99] (2012). *Android Malware Genome Project*. [Online]. Available: http://www.malgenomeproject.org

[100] S. Aurangzeb and M. Aleem, "Evaluation and classification of obfuscated Android malware through deep learning using ensemble voting mechanism," *Sci. Rep.*, vol. 13, no. 1, p. 3093, Feb. 2023.

[101] A. Almaleh, R. Almushabb, and R. Ogran, "Malware API calls detection using hybrid logistic regression and RNN model," *Appl. Sci.*, vol. 13, no. 9, p. 5439, Apr. 2023.

[102] V. Ravi and M. Alazab, "Attention-based convolutional neural network deep learning approach for robust malware classification," *Comput. Intell.*, vol. 39, no. 1, pp. 145–168, Feb. 2023.

[103] H. Naeem, S. Dong, O. J. Falana, and F. Ullah, "Development of a deep stacked ensemble with process based volatile memory forensics for platform independent malware detection and classification," *Expert Syst. Appl.*, vol. 223, Aug. 2023, Art. no. 119952.

[104] T. Zhao, E. Foo, and H. Tian, "A digital twin framework for cyber security in cyber-physical systems," 2022, *arXiv:2204.13859*.

[105] M. Z. Gunduz and R. Das, "Cyber-security on smart grid: Threats and potential solutions," *Comput. Netw.*, vol. 169, Mar. 2020, Art. no. 107094.

[106] Z. Moti, S. Hashemi, H. Karimipour, A. Dehghantanha, A. N. Jahromi, L. Abdi, and F. Alavi, "Generative adversarial network to detect unseen Internet of Things malware," *Ad Hoc Netw.*, vol. 122, Nov. 2021, Art. no. 102591.

[107] M. Sewak, S. K. Sahay, and H. Rathore, "ADVERSARIALuscator: An adversarial-DRL based obfuscator and metamorphic malware swarm generator," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–9.

[108] C. Patsakis, F. Casino, and N. Lykousas, "Assessing LLMs in malicious code deobfuscation of real-world malware campaigns," 2024, *arXiv:2404.19715*.

[109] M. M. Arifin, M. S. Ahmed, T. K. Ghosh, I. A. Udoy, J. Zhuang, and J. Yeh, "A survey on the application of generative adversarial networks in cybersecurity: Prospective, direction and open research scopes," 2024, *arXiv:2407.08839*.

[110] J. Jithish, S. Sankaran, and K. Achuthan, "A hybrid machine learning approach for intrusion detection in cyber-physical manufacturing systems," in *Intelligent Security Solutions for Cyber-Physical Systems*. Boca Raton, FL, USA: CRC Press, 2024, pp. 156–168.

[111] O. Alshamsi, K. Shaalan, and U. Butt, "Towards securing smart homes: A systematic literature review of malware detection techniques and recommended prevention approach," *Information*, vol. 15, no. 10, p. 631, Oct. 2024.

[112] A. Werner, F. Schuseil, M. Hämmerle, S. Schaper, and K. Hölzle, "Modular digital twin—An approach for generating and exploiting product sustainability information towards service-oriented business models," *Int. J. Prod. Res.*, vol. 2024, pp. 1–19, Jul. 2024.

[113] K. Achuthan, S. Sankaran, S. Roy, and R. Raman, "Integrating sustainability into cybersecurity: Insights from machine learning based topic modeling," *Discover Sustainability*, vol. 6, no. 1, p. 44, Jan. 2025.

[114] V. Bolón-Canedo, L. Morán-Fernández, B. Cancela, and A. Alonso-Betanzos, "A review of green artificial intelligence: Towards a more sustainable future," *Neurocomputing*, vol. 599, Sep. 2024, Art. no. 128096.

[115] M. Regona, T. Yigitcanlar, C. Hon, and M. Teo, "Artificial intelligence and sustainable development goals: Systematic literature review of the construction industry," *Sustain. Cities Soc.*, vol. 108, Aug. 2024, Art. no. 105499.

[116] N. F. Prangon and J. Wu, "AI and computing horizons: Cloud and edge in the modern era," *J. Sensor Actuator Netw.*, vol. 13, no. 4, p. 44, Aug. 2024.

SARANYA CHANDRAN received the bachelor's degree in computer science and engineering and the master's degree in cyber security and networks. She is a Researcher with Amrita Vishwa Vidyapeetham, India. Previously, she was a Senior Information Security Engineer with Infosys, where she gained significant industry experience in cybersecurity. She is dedicated to advancing digital security through innovative research and practical applications. Her areas of interests include malware analysis, reverse engineering, and penetration testing.

SREELAKSHMI R. SYAM received the master's degree in cyber forensics and information security. She is currently a Researcher with Amrita Vishwa Vidyapeetham, India. Previously, she was a Security Operations Center (SOC) Analyst with UST Global, gaining valuable experience in cybersecurity operations. Her areas of interests include cyber forensics, incident response, and penetration testing, with a focus on enhancing cybersecurity resilience and investigative capabilities.

SRIRAM SANKARAN received the B.Tech. degree (Hons.) in computer science and engineering from the Malaviya National Institute of Technology, Jaipur (formerly known as Regional Engineering College), and the M.S. and Ph.D. degrees in computer science and engineering from the University at Buffalo, The State University of New York, USA. He is currently an Associate Professor with the Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, India.

TULIKA PANDEY is currently a Scientist G and the Group Coordinator (Research and Development in Cyber Security and UIDAI), Ministry of Electronics and Information Technology, Government of India. She is an Electronics and Communication(s) Engineer with 33 years of professional experience. She is an Architect of the Internet Governance Initiative in India, with guidelines for the National Internet Exchange of India, Domain Name Registry and Dispute Resolution Policy, and also allocation of Internationalized Domain Names of Country Top level in Indian Scripts (languages). She has research papers in the area of internet technology, including networks (server), communications, cybersecurity, and the IoT. Her current focus is on research and development in cyber security. She is a member of several technical advisory boards and committees and technical associations. She is recognized as one of the top 20 influential Cyber Warriors of the country.

KRISHNASHREE ACHUTHAN (Senior Member, IEEE) is a Professor and the Dean with Amrita Vishwa Vidyapeetham. She is also the CEO of the Amrita Technology Business Incubator. With over 150 peer-reviewed publications and 34 U.S. patents to her credit, her work is characterized by a unique blend of technical expertise along with an understanding of the societal implications of technology. Her efforts have not only advanced academic knowledge but also has a tangible impact on policy-making and the broader discourse on technology and social issues. She has been instrumental in creating joint centers to promote research, development, and outreach with the governments of India, MNCs, and international universities. She led the Technology, Security and Transparency Engagement Group for Civil20 India, in 2023. She has been serving as an International Advisory Board Member for Civil20 Brazil, since 2024. She has continued to champion the cause of women in technology and has reviewed technical publications in several Q1 journals. Her areas of interests include cybersecurity, education, innovation, entrepreneurship, and technologies for societal development.