Check for updates

ACM DIGITAL LIBRARY

Association for Computing Machinery

acm open

Latest updates: https://dl.acm.org/doi/10.1145/3638552

PDF Download
3638552.pdf
21 December 2025
Total Citations: 66
Total Downloads: 29949

Published: 22 January 2024
Online AM: 28 December 2023
Accepted: 21 December 2023
Revised: 24 September 2023
Received: 16 December 2022

Citation in BibTeX format

SURVEY

Malware Detection with Artificial Intelligence: A Systematic Literature Review

MATTHEW G GABER, Edith Cowan University, Perth, WA, Australia

MOHIUDDIN AHMED, Edith Cowan University, Perth, WA, Australia

HELGE JANICKE, Edith Cowan University, Perth, WA, Australia

Open Access Support provided by:

Edith Cowan University

ACM Computing Surveys, Volume 56, Issue 6 (June 2024)
https://doi.org/10.1145/3638552
EISSN: 1557-7341

Malware Detection with Artificial Intelligence: A Systematic Literature Review

MATTHEW G. GABER, MOHIUDDIN AHMED, and HELGE JANICKE, Edith Cowan University School of Science, Australia

In this survey, we review the key developments in the field of malware detection using AI and analyze core challenges. We systematically survey state-of-the-art methods across five critical aspects of building an accurate and robust AI-powered malware-detection model: malware sophistication, analysis techniques, malware repositories, feature selection, and machine learning vs. deep learning. The effectiveness of an AI model is dependent on the quality of the features it is trained with. In turn, the quality and authenticity of these features is dependent on the quality of the dataset and the suitability of the analysis tool. Static analysis is fast but is limited by the widespread use of obfuscation. Dynamic analysis is not impacted by obfuscation but is defeated by ubiquitous anti-analysis techniques and requires more computational power. Sophisticated and evasive malware is challenging to extract authentic discriminatory features from and, combined with poor quality datasets, this can lead to a situation where a model achieves high accuracy with only one specific dataset.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation** • **Computing methodologies** → *Artificial intelligence*; *Machine learning approaches* • **General and reference** → *Surveys and overviews*;

Additional Key Words and Phrases: Malware, artificial intelligence, machine learning, deep learning, computer security, malware repository, malware analysis techniques, feature selection, evasive malware, sophisticated malware

ACM Reference format:
Matthew G. Gaber, Mohiuddin Ahmed, and Helge Janicke. 2024. Malware Detection with Artificial Intelligence: A Systematic Literature Review. *ACM Comput. Surv.* 56, 6, Article 148 (January 2024), 33 pages.
https://doi.org/10.1145/3638552

1 INTRODUCTION

In recent years, there has been increasing interest in malware detection using **Artificial Intelligence (AI)**, which is an umbrella term for a number of **Machine Learning (ML)** and **Deep Learning (DL)** models. AI in the cybersecurity domain can be divided into three main areas: malware detection, network intrusion detection, and phishing detection [6, 7, 93]. Network intrusion detection attempts to discover and prevent malicious network traffic across a wide range of security threats, including **Denial of Service (DoS)** and **Distributed Denial of Service (DDoS)** attacks, botnet activities, malware communication with **Command and Control (C2)** servers, crypto mining attacks, and exfiltration of data [6, 7, 20, 25, 75]. Phishing detection attempts to

Authors' address: M. G. Gaber, M. Ahmed, and H. Janicke, Edith Cowan University School of Science, 270 Joondalup Dr, Joondalup, Western Australia, Australia, 6027; e-mails: mgaber@our.ecu.edu.au, {mohiuddin.ahmed, h.janicke}@ecu.edu.au.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
0360-0300/2024/01-ART148
https://doi.org/10.1145/3638552

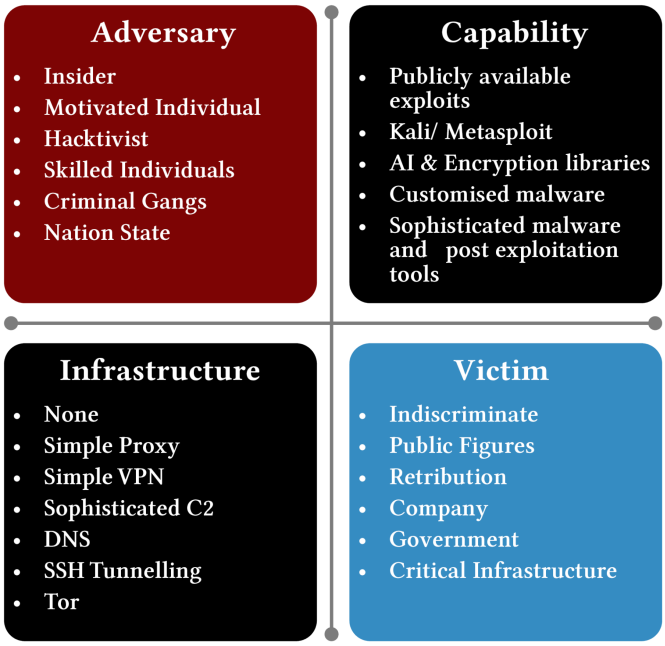

Fig. 1. Adversarial threat model.

identify and prevent a large set of techniques that attempt to deceive individuals into disclosing sensitive information, opening malicious files, and following links to malicious websites [6, 7, 47]. This article focuses on malware detection that includes trojans, ransomware, crypto miners, worms, botnets, and others, each with distinct functionality and purpose [72, 99].

The cyber arms race between malicious adversaries and security researchers is cyclical, where once a vulnerability is discovered and an exploit developed, then mitigations are implemented [24, 69]. Accordingly, malware authors continuously develop new anti-analysis techniques to evade analysis tools, where, if the malware detects it is under analysis, then it hides its malicious intent or ceases execution [29, 30, 74, 77]. Evasive malware is widespread and employs varied anti-analysis techniques to evade widely used static and dynamic analysis tools, such as disassemblers, decompilers, debuggers, sandboxes, and **Dynamic Binary Instrumentation (DBI)** [22, 30, 69, 74]. Anti-analysis directly impacts AI models, as the detection accuracy and resilience of a model is dependent on the quality and authenticity of the features it is trained with [25, 45, 90]. Additionally, cyber-attacks continue to evolve where malware authors have easy access to feature rich programming languages and powerful open-source encryption, as well as AI libraries, that enable the development of new sophisticated malware [14, 48, 81, 84, 104].

During 2022, IBM Security studied 550 organizations across 17 countries impacted by data breaches and found the average total cost was USD 4.35 million per incident [41]. In addition, the AV Test Institute registers 450,000 new malware and potentially unwanted applications every day. Further, the total number of malware targeting Windows, Android, Mac OS, and Linux has more than doubled from 450 million in 2018 to 970 million in 2022 [11]. Dark web marketplaces offer numerous and diverse hacking products and services. Marin et al. [60] have collected data from 20 marketplaces, where they found 51,902 products offered by 7,055 vendors. The threat landscape is diverse and can be summarized with an adversarial threat model that illustrates how an adversary can use a capability and infrastructure against a victim (Figure 1). Ransomware, which

Table 1. Summary and Comparison of Recent Literature Reviews with Our Article

Paper	Malware Sophistication	Static & Dynamic Analysis	Malware Datasets	Feature Selection	ML & DL	Results	Challenges
[33]	≈	✓	x	✓	✓	x	✓
[85]	x	x	≈	x	✓	✓	✓
[71]	✓	✓	x	≈	x	x	x
[8]	✓	≈	✓	✓	≈	≈	≈
[20]	✓	≈	x	x	≈	≈	≈
[90]	✓	✓	✓	✓	≈	≈	✓
This article	✓	✓	✓	✓	✓	✓	✓

Covered ✓; Partially Covered ≈; Not Covered x.

includes crypto that encrypts specific folders, and a locker, which encrypts the entire hard drive, is one of the most widespread types of malware, where the decryption key is only provided if a ransom is paid [48]. This is of particular concern and a major risk for both individuals and organizations [48, 84]. Ransomware toolkits, **Ransomware as a Service (RaaS)**, and the large infection vector have created significant growth in ransomware attacks [84].

For the reasons outlined above, AI is becoming essential in detecting malware. However, optimizing an AI model that is dependent on high-quality features to detect novel, evasive, and sophisticated malware is a significant task [10, 12, 90]. The effectiveness of an AI model is dependent on the quality and quantity of the features it is trained with [3, 45, 57, 90, 103]. In turn, the quality and authenticity of these features is dependent on the quality of the dataset and the suitability of the analysis tool. Where the objective is to deploy an AI model in a production environment, it must be accurate and generalizable so it can detect malware it has never seen before, but also be practical and lightweight. This is challenging for a number of reasons. There are numerous types of malware where each category behaves differently and may not have any commonalities [72, 99]. Further, sophisticated and evasive malware that use anti-analysis techniques impacts analysis tools, extracted features, and the accuracy of the AI model. This article surveys state-of-the-art methods across five critical aspects of building an accurate and robust AI malware-detection model: malware sophistication, analysis techniques, malware repositories, feature selection, and ML vs. DL.

The remainder of this article is structured as follows: Section 2 presents a summary of previous literature reviews and their focus and findings. Section 3 defines the review methodology of this article including the research questions, search strategy, and exclusion and inclusion criteria. Section 4 presents the current issues and challenges, and Section 5 discusses these trends and the core obstacles. Section 6 presents our conclusions.

2 RELATED WORK

Research studies and literature surveys in this field have tended to focus on specific aspects of malware detection such as: malware sophistication and evasiveness, or static and dynamic analysis techniques, or malware repositories, or feature selection and AI models. This splintered approach has led to a situation where many of the research papers claim to outperform others; however, the results are contradictory. Some studies claim that DL is more accurate and efficient than ML and vice versa. This section provides a summary of recent literature reviews and their focus, where Table 1 summarizes the contributions.

Gibert et al. [33] have presented an in-depth systematic review that focuses on categorizing AI techniques based on the malware analysis method, being either static or dynamic. A taxonomy of the features that can be extracted with static and dynamic analysis is presented along with

a detailed description of each. The types of AI models used with various features are detailed, and Gibert et al. [33] have considered how adversarial malware can defeat AI. The limitations of sophisticated malware and how it can evade analysis tools along with the challenges faced by researchers, including unbalanced datasets and concept drift, are also discussed [33].

Shaukat et al. [85] have identified and provided detailed information on the varied ML and DL techniques widely used in cybersecurity. They provide an extensive systematic review on AI used for a number of cybersecurity fields, including spam detection, intrusion detection, and malware detection. The dataset, analysis technique, AI model, and results are identified for the surveyed research papers. Further, Shaukat et al. [85] identify the requirement for current, large, and diverse benchmark datasets. Or-Meir et al. [71] have identified dynamic analysis as being more robust than static analysis, presenting a systematic review of dynamic malware analysis. Their study provides a comprehensive overview of malware and classifications based on type, behavior, and privileges and also comprehensively covers anti-analysis techniques used by evasive malware. The focus of their survey is dynamic analysis, and the tools and techniques that are widely used are discussed in detail. Aslan and Samet [8] have provided a review on malware classification approaches. In their study, the challenges with sophisticated and evasive malware are covered, and a number of features and malware repositories are identified. However, the main focus of their review is the varied malware detection approaches including signature-based, behavior-based, deep learning, and cloud-based, among others. The results and the features used across the various detection approaches are covered in detail.

Caviglione et al. [20] have provided a systematic review on the evolution of malware, information hiding, evolution of malware detection, and the application of AI. Their review surveys recent developments in malware types and techniques as well as the evolution of obfuscation and evasion. Where early malware used encryption and code obfuscation to escape static signature detection, more recent malware employs ever-increasingly sophisticated techniques including polymorphism and anti-analysis techniques. Various steganographic techniques are covered that include hiding malicious content in benign files and in covert network channels using various legitimate TCP/IP protocols. Caviglione et al. [20] review the evolution of malware detection from signature-based to behavior and heuristic methods to AI models. Widely used ML and DL models are considered, and emerging techniques such as blockchain-based malware detection and transfer learning AI models are presented. The main focus of their review is to provide a detailed perspective across many domains, including the evolution of malware and evolving detection techniques used by security researchers.

Ucci et al. [90] have provided the most complete systematic literature review, considering malware sophistication, static and dynamic analysis techniques, malware repositories, feature selection, and the challenges and limitations associated with each. However, information on the various AI models nor the results for the papers surveyed are provided.

Given the limitations of recent works, this study presents a systematic review on AI techniques for malware detection, examining state-of-the-art methods across the five critical aspects of building accurate and robust AI malware detection models.

3 REVIEW METHODOLOGY

This study systematically identifies and critically analyzes state-of-the-art methods and challenges for malware detection using AI. The methodology is based on an analysis of relevant literature and a synthesis of research findings in a systematic, transparent, and reproducible way. The main focus of this article is Windows malware, however, a limited number of papers that studied Android, Linux, and IoT malware are also evaluated due to the novel approaches presented.

3.1 Research Questions

RQ1. What are the major challenges for malware detection using AI?
RQ2. What are the emergent technologies used by malware authors?
RQ3. How does sophisticated malware impact static and dynamic analysis?
RQ4. What are the limitations of existing malware repositories?
RQ5. What features are optimal for training an AI model?
RQ6. Which AI models are most successful for malware detection and what are their advantages and limitations?

3.2 Search Strategy

An extensive literature search was conducted in the ACM Digital Library,[1] IEEE Xplore,[2] and the Scopus database.[3] ACM Digital Library was chosen because it provides access to **Association for Computing Machinery (ACM)** journals, proceedings, and conferences. Similarly, IEEEXplore was chosen because it provides access to **Institute of Electrical and Electronics Engineers (IEEE)** conference papers and journals. Both ACM and IEEE are predominant in cybersecurity. Scopus was chosen because it is the most comprehensive database for the relevant subject area. The results were limited to articles published no earlier than 2018 to focus on the latest research in a rapidly evolving field. Google Scholar was also used, as it allows a broad search across many sources and to specifically target the latest research that examines the use of **Large Language Models (LLM)** for cybersecurity.

3.2.1 ACM Digital Library. Search string: [[Title: "artificial intelligence"] OR [Title: "machine learning"] OR [Title: "deep learning"]] AND [[Title: malware] OR [Title: ransomware]] AND NOT [[Title: survey] OR [Title: review]] AND [E-Publication Date: (01/01/2018 TO 12/31/2022)] This search returned 57 results (6 Journals and 49 other) on November 10, 2022.

3.2.2 IEEEXplore. Search string: ("Document Title": "artificial intelligence" OR "Document Title": "machine learning" OR "Document Title": "deep learning") AND ("Document Title": malware OR "Document Title": ransomware) NOT ("Document Title": survey OR "Document Title": review) Limit: 2018–2022 This search returned 305 results (Journals 30 and 262 Conferences) on November 4, 2022.

3.2.3 Scopus. Three search strings were used for Scopus where the search was in the title, abstract, and keywords, limited to Computer Science, Engineering, Mathematics, and Decision Science, and sorted by relevance.

(1) (artificial AND intelligence) OR (machine AND learning) OR (deep AND learning) AND (malware) OR (ransomware). Date range 2019–2023. This search returned 4,576 results on November 14, 2022, where sorted by relevance the first 300 results were analyzed.
(2) (malware OR ransomware) AND (sampl*) AND (database OR dataset OR repository) AND (taxonom* OR famil* OR categor*) AND NOT (android OR linux). Date range 2018–2022. This search returned 143 results on November 14, 2022.
(3) (malware OR ransomware) AND (dynamic OR sandbox OR analysis) AND (evad* OR evas* OR anti) AND NOT (android). Date range 2018–2022. This search returned 1,413 results on November 14, 2022, where sorted by relevance the first 300 results were analyzed.

[1] https://dl.acm.org
[2] https://ieeexplore.ieee.org/Xplore/home.jsp
[3] https://www.scopus.com/home.uri

3.2.4 Google Scholar. Search string: "artificial intelligence" OR "machine learning" OR "deep learning" AND malware OR ransomware -android -adversarial -survey -review -classification. Limit: 2018 onwards. This search returned 2,630 results on November 10, 2022. The first 10 pages of results were analyzed, which returned 4 relevant papers.

3.2.5 Google Scholar. Search string: malware AND "generative AI" OR "ChatGPT" OR "LLM" OR "Large Language Models" OR "BARD" Limit: 2022 onwards. This search returned 1,170 results on September 13, 2023. The first 10 pages of results were analyzed, which returned 4 relevant papers.

3.3 Inclusion and Exclusion Criteria

The objective was to select scientific papers on recent advances in malware detection using AI.
Inclusion

- Quantitative research design that uses an experimental methodology to evaluate AI techniques.
- Papers that focus on malware detection.
- Papers published from 2018 onwards.
- A minimum 80% of the selected papers to be peer-reviewed from ACM Digital Library, Elsevier's Scopus, or IEEE Xplore Digital Library.
- A maximum 20% of the selected papers may be grey literature.

Exclusion

- Studies that focus on malware family classification.
- Studies that do not explicitly define the AI techniques used.
- Studies that do not explicitly define the malware and benign software datasets.
- Studies that do not explicitly define the feature extraction method used for training and testing the AI model.
- Studies that do not explicitly define the results.
- Technical documents, reports, thesis, and books.
- Studies that are not published in the English language.

4 RESULTS

After applying the inclusion and exclusion criteria, we selected 77 papers for the final analysis, including 10 grey papers. The following sections discuss state-of-the-art methods for the five key aspects of building an accurate and robust AI malware-detection model.

4.1 Malware Sophistication

Anti-virus and malware scanners typically use signatures to detect known and previously classified threats and heuristic-based detection [10, 40, 43, 48, 72]. A malware signature is a sequence of bytes that represent a pattern of behavior, code, or strings found in a malware file [17, 48]. Heuristic-based malware detection uses a set of rules defined by malware analysts to identify suspicious behavior but can be prone to error [43, 102]. As cyber-attacks continue to become more advanced, the sophistication of modern malware means that signature-based and heuristic detection can be easily defeated [22, 48, 51, 58, 95]. Further, recycled malware that has been slightly changed or obfuscated bypasses signature detection engines, where defining a signature for new malware or its variants is time-consuming, both in terms of the manual analysis required and also the client update, which may not be performed for months [34, 48, 72, 102]. Evasive malware that seeks to hide its malicious intent, malware that uses AI to conceal payloads with target and

trigger conditions, and novel malware that does not have a signature further complicate modern malware detection [17, 22, 73, 84, 95]. This is not limited to Windows but also impacts Android, **Internet of Things (IoT)**, and Linux VM cloud servers, where similar sophisticated techniques can be implemented [25, 70, 72].

4.1.1 Evasive Malware. Malware authors continuously develop new anti-analysis techniques to evade analysis tools, either by obfuscating a file or using anti-analysis techniques where if the malware detects it is under analysis during execution, then it hides its malicious intent [29, 30, 48, 72, 74, 77]. The longer malware can evade analysis, the more opportunity it has to inflict damage or extort ransom payments [30]. Evasive malware is ubiquitous and resists analysis by widely used tools, such as decompilers, debuggers, sandboxes, and DBI [12, 22, 30, 69, 73, 74].

Malware authors can use numerous techniques to resist analysis. Obfuscation is often used to make the malware files more difficult to detect and analyze and primarily affects static analysis. Code Packing is a type of obfuscation that compresses parts of a malicious file that are only decompressed when the program is executed [48]. Metamorphic obfuscation can be used to transform the Header and Sections in previously detected malicious **Portable Executable (PE)** files. Metamorphic malware may use dead code insertion, **Central Processing Unit (CPU)** register reassignment, and code reordering [34]. These changes to a malicious file do not alter its functionality or behavior but only its composition, which directly impacts features extracted using static analysis [34, 99]. Polymorphic obfuscation is more challenging to deal with, as it encrypts and mutates parts of the internal code to change its shape and signature, whereby the decryption method may only decrypt parts of the file as required by execution [48]. Legitimate software can use commercial packers, which also implement evasive behaviors and obfuscation techniques, to prevent analysis and reverse engineering of proprietary products [74].

In a dynamic analysis environment, evasive malware can use varied anti-analysis techniques to detect the environment in which it is running, and then to behave differently and hide its malicious intent if it detects it is under analysis [12, 31, 40, 69]. Malware can probe its environment by performing a series of API calls to fingerprint the runtime environment and avoid malicious execution in dynamic analyzers such as debuggers, sandboxes, and DBI [30, 40, 66].

Galloro et al. [30] have surveyed existing literature to identify evasive techniques used by Windows malware to defeat analysis, identifying 92 evasive techniques that are classified into 16 categories according to the type of operation they perform. Further, Galloro et al. [30] have experimentally quantified the prevalence of the evasive techniques in modern malware using the Intel Pin DBI tool.

The scope of research conducted by Galloro et al. [30] is comprehensive, considering a large number of evasive techniques, live malware samples, and temporal and taxonomic analysis. The study's analysis of evasive techniques used 45,375 malware samples and 516 legitimate PE files [30]. The Intel Pin tool, with a framework based on Arancino [77], was used to automatically analyze the evasive techniques. The framework was then extended and validated by implementing 92 techniques in test programs, after which experiments were performed to confirm the framework and whether or not DBI could detect and circumvent all 92 evasive techniques. The framework operates by circumventing four types of operations detailed in Table 2 and implementing a blacklist of terms used to fingerprint execution environments.

The framework implemented by Galloro et al. [30] inserts hooks before and after any function or operation that is associated with the 92 evasive techniques. A predefined bypass mechanism and logging routine was then executed to circumvent the evasive operation. If the malware used multiple evasive techniques, then this method of bypassing the probes continued. Additionally, when the malware accessed file information by executing syscalls, the framework examined the file path; and if it contained any of the terms in the blacklist, for example "debugger," then a NOT

Table 2. Operations Used in Evasive Behavior

Operation [30]	Details
Instruction Pointer (IP)	The IP is monitored for every instruction that is processed by the Central Processing Unit (CPU).
Application Programming Interface (API)	Allows programs to communicate and malware can interact with the OS and use its libraries.
System Calls (syscalls)	Allows programs to request a service from the kernel, malware can request a service and, based on the return, determine if it is running on bare metal.
Memory Access	Malware can attempt to read certain parts of the memory to detect if it is running on bare metal or a virtualized or instrumented environment.

FOUND was returned to the calling function. A similar approach was generated whenever the Windows registry was enumerated. Further to this, DBI introduced significant overhead, so when the evasive behavior involves time measurements, the returned values were reduced accordingly. It is worth noting that 15 of the 92 techniques were not found in any of the samples and that approximately 80% of the malware samples were evasive [30].

Park et al. [74] have surveyed existing literature to identify evasive techniques used by Windows malware to defeat analysis and classified 29 techniques into 3 groups as based on **Operating System (OS)** libraries and instructions. Park et al. [74] implemented their 29 evasive techniques and conducted experiments to determine if the standard installation of the Intel Pin tool could escape detection of the probes performed by the evasive techniques. Eight evasive techniques were subsequently identified to detect the Intel Pin runtime environment. A plugin for Intel Pin was then created, with three algorithms to automatically intercept the eight evasive techniques and to deploy countermeasures so all 29 evasive techniques could be automatically circumvented [74]. The techniques, Prefix Handling, Memory Breakpoint, and Self-Modification, that were identified by Park et al. [74] detected the Intel Pin tool, but were not considered by Galloro et al. [30]. The techniques that were considered by both Park et al. [74] and Galloro et al. [30] are presented in Table 3.

Kim et al. [51] and Polino et al. [77] have reported anti-instrumentation, that is, anti-DBI techniques to be found in 16.21% and 15.6% of analyzed malware samples, respectively. Contrastingly, Galloro et al. [30] and Sharma et al. [83] have reported anti-analysis techniques to be evident in 80% and 99.36% of the malware samples they analyzed, respectively, which indicates the evasive techniques used by malware authors are not focused on DBI but rather debuggers and sandboxes. New evasive techniques are constantly being discovered, where the methods for circumventing the previously identified anti-analysis techniques will not work with new anti-analysis techniques [30]. Therefore, the results presented by Galloro et al. [30] may underestimate the prevalence of evasiveness, while Park et al. [74] may have overestimated the transparency of the Intel Pin tool to evasive malware. Further to this, Maffia et al. [59] have shown that approximately 68% of the malware samples they analyzed used some form of obfuscation, which severely limits static analysis. Because evasive behavior is widespread, Chen et al. [21] have proposed the use of artefacts to imitate VMs and debuggers, which could be used to protect real machines from malware.

4.1.2 Novel Malware. In this article, the term novel malware is used to describe any malware that has not been seen before and would not be detected by a malware detection engine that uses signature matching. This includes new malware that does not fit existing families, malware that employs new anti-analysis or obfuscation techniques, and zero-day malware that exploits zero-day vulnerabilities.

Advanced Persistent Threat (APT) malware is characterized by: its sophistication; advanced **Tactics, Techniques, and Procedures (TTPs)**; and its lengthy campaigns, which can last for

Table 3. Evasive Techniques, whether They Can Detect the Intel Pin Tool and Their Prevalence

EvasiveTechnique [74]	Pin Detected [74]	Malware (%) [30]	Benign (%) [30]
IsDebuggerPresent	x	1,516 (3.34)	17 (1.79)
CheckRemoteDebuggerPresent	x	432 (0.95)	1 (0.11)
OutputDebugString	x	794 (1.75)	26 (2.74)
FindWindow	x	2,245 (4.95)	51 (5.37)
QueryInformationProcess	x	1,028 (2.27)	4 (0.42)
SetInformationThread	x	350 (0.77)	1 (0.11)
OutputDebugString()FormatString	x	N.A.	N.A.
SeDebugPrivilege OpenProcess[a]	x	N.A.	N.A.
QIP(ProcessDebugFlags)	✓	1,028 (2.27)	4 (0.42)
QIP(DebugHandleObject)[b]	x	10,651 (23.47)	157 (16.53)
QueryPerformanceCounter	✓	6,038 (13.31)	140 (14.74)
GetTickCount	x	11,029 (24.32)	198 (20.84)
timeGetTime	x	805 (1.77)	32 (3.37)
CloseHandle	x	3,104 (6.84)	19 (2.0)
Hardware Breakpoints	x	46 (0.1)	0 (0)
Control-C Vectored Exception	x	0 (0)	0 (0)
RDTSC	✓	9,518 (20.98)	168 (17.7)
INT 3 Exception (0XCC)	x	747 (1.65)	0 (0)
INT 2D (Kernel Debugger Interrupt)	✓	39 (0.09)	0 (0)
ICE Breakpoint	x	61 (0.13)	0 (0)
Single Step Detection	✓	141 (0.31)	0 (0)
Unhandled Exception Filter	x	15,651 (34.49)	475 (50)

[a]Some techniques that use the Windows API have been deprecated or changed. SeDebugPrivilege OpenProcess is only available for Administrator and SYSTEM accounts and is not enabled by default in Windows 10 [64].
[b]QueryInformationProcess (QIP) DebugHandleObject has changed to NtQuerySystemInformation (SYSTEM PROCESS INFORMATION) in the Windows API Win32 [63].

years with the aim of sabotaging digital infrastructure or espionage [61, 83]. APTs use obfuscation, anti-analysis techniques, and AI to bypass security solutions but also have the resources to develop zero-day exploits and payloads [83]. There are numerous examples of APT attacks against nations and companies, with a couple of prominent examples detailed below.

The SolarWinds attack is an example of an APT that involved compromising a supply chain to deliver malware through the Orion Network Management System [83]. In this context, the HAFNIUM APT group used four zero-day exploits against vulnerabilities in the Microsoft Exchange Server [39, 83]. The attack affected approximately 400,000 Exchange Servers and allowed the attackers to download emails from affected companies and install backdoors to maintain access after the vulnerabilities were patched [76]. Stuxnet is probably the most well-known APT, where it targeted the SCADA control system of the centrifuges at the Natanz nuclear enrichment plant [62]. In this attack, the centrifuge speeds were modulated in a way to induce vibration and ultimately their destruction. Stuxnet used custom exploits for four zero-day vulnerabilities and used stolen digital certificates to create genuine software signatures [62].

Further to these examples, even less sophisticated adversaries can use increasingly complex TTPs, because technically complex components such as powerful AI and encryption libraries are open source and malicious code is readily available [54].

4.1.3 AI-powered Malware. AI technologies are pervasive and for the most part beneficial, but their underlying technologies can be used maliciously [46]. Powerful AI frameworks, such as

TensorFlow, are open source and publicly available and can be used to drive powerful attacks. Next-generation malware that uses AI to enhance its offensive capabilities are emergent threats and typically use **Neural Networks (NN)** to power evasiveness and targeting [46]. NNs are not easily interpretable and lack transparency, where, because of this, they can be used to hide malicious payloads and conceal target and trigger conditions [14]. AI can also be used in conjunction with existing malware approaches to define the target of an attack, concealing the malicious payload and automating the exploitation [46, 88]. For example, DeepLocker was presented by IBM at Blackhat 2018, where it used a known malicious ransomware payload and a **Convolutional Neural Network (CNN)** to target an individual [54]. The ransomware payload was encrypted and was only decrypted when the CNN identified the target it had been trained on, via the webcam. In this way, DeepLocker was presented as a novel malware not detected by antivirus software [54].

Further, AI can be used by malware to target the **Graphical User Interface (GUI)**. Yu et al. [104] have demonstrated an attack that used the TensorFlow Object Detection CNN model to detect browser shortcut icons, emulating a click on the icon and stealthily logging in to the user's blackboard account. Although the GUI attack is relatively simple, its approach can be extended in sophistication, where the malware could detect if the user was on a banking website and then launch an attack to transfer money to another account [104].

AI-powered malware may be immune to analysis where the malicious payload is encrypted and only decrypted when the target is detected. The target class and target instance and encryption key may be embedded in the trained NN, which is essentially a black box [46, 54].

4.2 Analysis Techniques

Diverse features, such as **Application Programming Interface (API)** calls, **operational code (opcode)** sequence, flow control, and Windows registry interactions, can be extracted from malware using static and dynamic analysis. Static analysis does not execute the malware, but rather is analyzed statically and can use disassembly tools to generate assembly language from the file. However, malware programs that use obfuscation are resistant to static analysis, whereas dynamic analysis that includes DBI and sandboxes are not affected [29, 69, 95]. This is because they run the malicious file in a controlled environment, where its behavior can be observed [61]. Nonetheless, evasive malware can implement multiple types of anti-sandbox and anti-instrumentation techniques to defeat dynamic analysis [69, 72].

There are a number of widely used static and dynamic analysis tools, such as decompilers, Python libraries, debuggers, sandboxes, and DBI [22, 30, 69, 74]. Typically, when analyzing malware, these tools are run in a VM as a security measure, where they efficiently revert to a known good state. Debuggers will not be covered due to the time-consuming manual nature of this type of analysis. VMs will not be covered because they are not an analysis tool as such, but rather an environment in which most analysis tools are run.

4.2.1 Static Analysis. Low-level features, such as byte and opcode sequences, as well as high-level features, including function calls and API calls, can be extracted using static analysis [22, 61, 102]. Malware that targets Windows is normally created using the PE file specification [34]. As shown in Figure 2, a PE file incorporates a Header and Sections, which includes the information necessary for Windows to run the file. In this way, a PE file specifies imports and how it is mapped to memory as well as the code that is executed.

There are various static analysis tools that can be applied, including the disassembler IDA Pro and the Python library Pefile, summarized in Table 4. IDA Pro can be used to disassemble binary files to **Assembly Language (ASM)** from which opcode sequence and frequency, byte sequence, API and system calls, and control flow graphs can be extracted [12, 22, 90]. Pefile can be used to extract information from the PE Header and the section's details and data [12, 18].

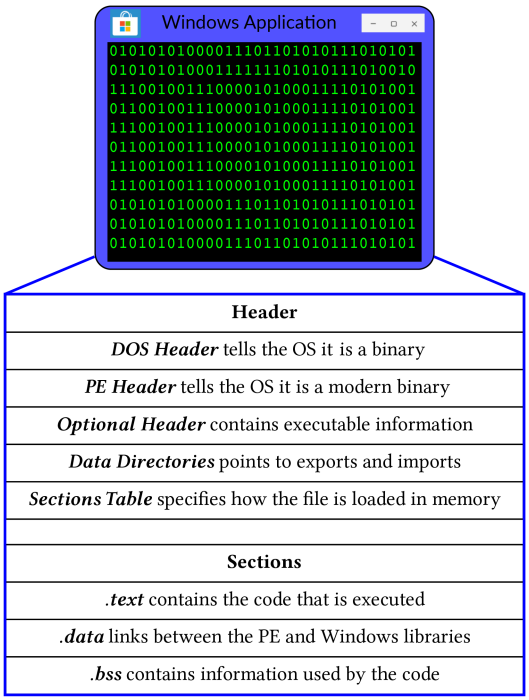

Fig. 2. Main components of the Windows Portable Executable (PE) file format.

Table 4. Static Analysis Features and Tools

Features	Tools
ASM Opcode, header, functions, strings	IDA Pro [22]
PE file header & data, strings, imports	Pefile (Python) [12, 18]
API calls and accessed DLL's	Peframe (Python) [103]
Byte sequences & n grams	Binary file bytes and N grams [90]
File, header and text section sizes	Pefile [12, 103]
Label benign or malicious	VirusTotal [12]

Malware authors can use numerous techniques to resist static analysis. Obfuscation is commonly used to make the malware files more difficult to analyze, which directly impacts static analysis [3, 84, 103]. Obfuscation techniques include code packing, metamorphism, and polymorphism, as detailed in Section 4.1.1, Evasive Malware.

Maffia et al. [59] have suggested that obfuscation is so widespread that static analysis is not a viable option for malware analysis, where approximately 60% of the samples they analyzed used some form of obfuscation. However, static analysis can explore all possible execution paths, where dynamic analysis may be limited [102]. The ability to cover all code paths can help provide a complete characterization of functionality of PE files, where this has led to novel applications of static analysis such as file-to-image conversions [52].

PE files can be easily converted to RGB and greyscale images, where each byte in the file and each pixel in an image has a value between 0–255 [70, 94]. One simple approach uses a vector of bytes extracted using static analysis, after which a color is assigned to an RGB pixel based on the corresponding byte value. The generated image can then be used to train varied AI models [32, 52].

Table 5. Dynamic Analysis Features and Tools

Features	Tools
API calls	Cuckoo Sandbox [66], DBI Intel Pin [30, 59]
Registry Access	Cuckoo Sandbox [66]
System Calls	Cuckoo Sandbox [22], DBI Intel Pin [30, 59]
Hardware Performance Counters	perf [10]
I/O access	Bitvisor [40]
File operations, memory dumps	Cuckoo Sandbox [40]
Network	Cuckoo Sandbox [66], Wireshark [98]
CPU Registers	DBI Intel Pin [30, 59], Debugger [103]

While converting PE files to images that are then used to train AI models is a novel approach, Verma et al. [94] outline that the technique has limitations, it is not effective for detecting novel malware, and can be easily defeated by inserting malicious code into benign carrier applications. Further, detection accuracy significantly declines when obfuscation is used by malware samples [70].

In research conducted by Branco et al. [15], more than 4 million malware files were analyzed to detect various evasive behaviors. However, only static analysis was used, where packed or otherwise obfuscated malware were shown to be resistant to static analysis [30]. The results indicate that obfuscation was used by 68.95% of samples, with Anti-disassembly used by 12.13% of the samples, Anti-debugging by 43.21%, and Anti-VM by 81.40%. Moreover, 88.96% of the samples used at least one anti-analysis technique, with 6.42% using at least one technique from each of these four categories [15].

For an AI malware detection system to be deployed in a production environment, it would have to be practical and lightweight. The deployed AI model would be required to extract features efficiently, where static analysis uses less computational resources and is faster than sandbox and DBI analysis [3, 61].

4.2.2 Sandbox Analysis. Sandboxes are dynamic analysis tools that can resolve obfuscation because the PE file is executed in a controlled environment; however, they can be limited by anti-sandbox techniques, trigger mechanisms, and function call branching [48, 52, 69, 102, 103]. Cuckoo is an automated sandbox that can analyze PE files and provides a **JavaScript Object Notation (JSON)** report. The JSON report includes API calls, loaded **Dynamic Link Libraries (DLLs)**, file operations, changes to Windows registry files, and network activity logs, summarized in Table 5 [44, 65, 69, 86].

API calls and interactions with the Windows registry are particularly discriminative for malware detection [3, 86, 90]. In this way, API calls reveal what system services have been used, such as encryption and networking libraries. Windows registry interactions are also very revealing, as malware can gain persistence on a target machine by adding an entry to the Windows SYSTEM registry [66, 90]. Further, certain paths in the registry can be used to start a process in a privileged environment, whereby particular keys in the registry can be manipulated to attach and run arbitrary executables as debuggers to legitimate processes.

Cuckoo [35] is a mature, resilient tool that is under active development, where these features are important attributes for ensuring accurate and reliable dynamic analysis [65]. Given the maturity of Cuckoo, several plugins have been developed to assist the tool in malware analysis. Malware can probe aspects of the network it is run in to determine if it is under analysis and to communicate with its **Command and Control (C2)** server. Accordingly, the network simulator INetSim can spoof DNS, HTTP, and SMTP internet services. INetSim has been shown to successfully trick

malware requiring internet access but not be of assistance where the malware required external resources from C2 servers, because there is no actual internet connection [65]. While Cuckoo is a widely used malware analysis tool, its popularity with security researchers has also captured the attention of malware authors [69].

Cuckoo can be defeated by numerous anti-analysis techniques [69]. Galloro et al. [30] found approximately 80% of malware samples to be evasive with anti-analysis techniques that target dynamic analysis tools, including sandboxes, DBI, and the underlying VM. Further to this, Sharma et al. [83] have created a framework to analyze the prevalence of anti-analysis techniques in APT malware. The framework was based on Cuckoo and designed to detect 26 anti-analysis techniques [83]. Their study showed that 99.36% of the 4,403 APT malware samples analyzed implemented at least one anti-analysis technique [83].

Assen et al. [9] have developed a tool called SecBox, which is based on Linux containers. SecBox is a lightweight sandbox that utilizes the OS-level isolation and the ability to run several instances on the same host provided by Linux containers. Key metrics including CPU, memory, system calls, and network usage are displayed visually during live analysis, and the reports can be exported after the analysis has concluded [9]. Data is collected directly from the shell of each sandbox using Python, where the information is piped to the monitors and written to logs [9]. SecBox is open source and modular and could potentially be extended to automate the process of malware sample collection, labelled analysis logs, and a more comprehensive set of features.

4.2.3 Dynamic Binary Instrumentation. DBI tools such as Intel Pin can be used to execute and analyze evasive malware with the aim of exposing its true behavior [30, 74]. DBI provides mechanisms for deep and precise control of instrumented PE files [23, 77]. DBI tools instrument PE files by inserting additional code that executes seamlessly as part of the PE [29, 30, 74]. Injecting code and additional functionality into an instrumented PE provides insights into low-level operations, such as CPU register instructions, as well as high-level operations that include API calls and Windows registry interactions [74, 77]. This is facilitated by a **Just In Time (JIT)** compiler, whereby the injected instrumentation code executes transparently as part of the normal instruction stream.

DBI is immune to most anti-analysis but can be defeated by anti-instrumentation techniques; consequently, several methods have been developed to defeat anti-instrumentation [23, 77]. These methods are all variations on intercepting the anti-instrumentation probes and deploying countermeasures against them [74]. Accordingly, function calls can be intercepted and the arguments, return values, and flow control can be manipulated to counter anti-instrumentation techniques [23, 30, 74].

There is an existing body of research that comprehensively details anti-instrumentation techniques and is summarized in Table 6 [23, 29, 30, 74, 77]. Polino et al. [77] created the ARANCINO framework for the Intel Pin DBI tool, showing that 15.6% of the 7,006 samples analyzed used at least one of the anti-instrumentation techniques listed in Table 6. Kim et al. [51] have similarly used an experimental approach to determine the prevalence of anti-instrumentation techniques in 763,985 malware samples, showing that 16.21% of their malware samples used at least one anti-instrumentation technique. Not all of the techniques analyzed in the studies by Kim et al. [51] and Polino et al. [77] are explicitly anti-instrumentation, where some are generic anti-analysis techniques. Further, some anti-instrumentation techniques, including runtime overhead, common API calls, and memory page permissions, are not reliable and can provide false positives. While these techniques are found in the literature, their lack of reliability may deter malware authors from using them [29]. In addition to anti-instrumentation, D'Elia et al. [23] have suggested that DBI tools may be vulnerable to escape, where malicious code may escape the DBI and execute directly on the CPU. However, specific escape attempts were mitigated by concealing instruction pointers and enforcing **no execute (NX)** privileges on certain memory regions [23].

Table 6. Summary of Anti-instrumentation Techniques across Five Categories

Category	Details
Indirect Evasion Techniques	DBI tools replicate the behavior of the underlying **Operating System (OS)** and the **Central Processing Unit (CPU)** architecture. The DBI tool may not implement every possible behavior, and this provides a mechanism for two anti-instrumentation techniques, Unsupported Assembly Instructions, and Unsupported Behaviors [29]. If the DBI tool does not support every available CPU assembly instruction and behavior, then this can be exploited to crash the DBI if the malware implements an unsupported instruction or behavior [29]. Further, analysis environments are typically allocated limited resources, and anti-instrumentation techniques can exploit this by attempting to exhaust available resources [29, 74].
Code Cache Artifacts	An instrumented binary file is written to memory differently than a binary file run on bare metal. This changes the **Instruction Pointer (IP)** that normally holds the memory address of the next assembly instruction to be executed, because it is executed in the code cache memory [77]. The changes to the IP can be detected by probes that make comparisons of addresses that are normally expected [23, 30].
Environment Artifacts	Probes that seek to fingerprint aspects of memory layout can detect DBI tools inside the process memory of the instrumented binary, whereby the memory layout for an instrumented binary is different and the DBI tool is normally the parent process of the instrumented binary [23, 30, 74, 77].
JIT Compiler Detection	JIT compilers attempt to conceal the presence of DBI tools by hiding their API and system calls. These hidden calls can be detected by comparing memory addresses and offsets [77].
Overhead Detection	DBI tools add considerable overhead that impacts execution time, which can be detected by probing the timing of various instructions and resource usage [23, 29, 74, 77].

An Intel Pin DBI tool labelled the Pin-based Evasive Program Profiler was developed by Maffia et al. [59], where it was used to dynamically analyze 183,340 malware PE files. The tool was capable of detecting and circumventing 53 anti-analysis techniques [59]. Benign software was also analyzed, where 4 techniques that had been previously used to classify malware as evasive were found to be widely used by legitimate software [59]. Accordingly, GetTickCount, CPU is Hypervisor, GetCursorPos, and NumberOfProcessors are widely used for legitimate purposes [59]. Excluding the techniques used by legitimate software, approximately 40% of the samples analyzed utilized at least 1 evasive technique [59].

DBI introduces overhead and is slower and more computationally expensive than sandboxes and static analysis; however, it appears to be the best approach for extracting authentic features from evasive malware [30, 77]. Consequently, the computational effort required to extract features must be balanced with the features that are most discriminative between malware and legitimate software [3, 34].

4.3 Malware Repositories

Creating a public open dataset that includes legitimate and malicious files is challenging [4, 100]. There are legal and copyright restrictions associated with the dissemination of proprietary Windows software. Further, there are potential security liabilities where live malicious files are released to a public that may not take the correct precautions when handling them [4, 100]. For these reasons, it is easier to use small, private datasets in research. However, this makes it difficult to measure progress and results, where studies may not be repeatable, as researchers work on different datasets [13, 37, 45, 95]. Thus, it is advantageous to the research community if different

Table 7. Summary of Malware Datasets

Dataset	EMBER [5]	BODMAS [101]	SOREL-20M [38]	Virus Share [96]	Malware Bazaar [1]
Total Samples	1.1M	134,435	20M	55M	700k
Dates	2018	2019–2020	2017–2019	N.A.	Yes
Timestamp	Partial	Yes	Yes	Yes	Yes
Taxonomy	Partial	Complete	No	No	Yes
Malware Samples	400,000	57,293	10M	48M	700k
Benign Samples	400,000	77,142	10M	N.A.	N.A.
Malware Binaries	No	Yes	Disarmed	Yes	Yes
Benign Binaries	No	No	No	No	No
Feature Vectors	Yes	Yes	Yes	No	No
Features	Static	Static	Static	Binaries	Binaries
Feature Extractor	Yes	Yes	No	No	No

approaches can be applied to a common dataset [37, 90]. In this context, five publicly available datasets were analyzed, with their findings presented in Table 7.

4.3.1 EMBER. Anderson and Roth [4] have released EMBER, a dataset of features extracted from 1.1 million PE files. The code, which could be used to extract the same features from other PE files, was also released. Because the dataset consists of extracted features and not PE files, copyright is not an issue for the legitimate software. However, there are several limitations with this dataset. Because the intact malicious and benign files are not included, different features cannot be extracted. Further, the features were extracted using static analysis, which can be limited with obfuscated malware [4, 5]. There may also be issues with the size and partition of the dataset, where because there are only 200,000 test samples, it may be difficult to measure false positive rates lower than 1 in 1,000 [37]. Moreover, Harang and Rudd [37] suggest that even baseline classifiers are capable of achieving greater than 0.999 **Area Under Curve (AUC)** with the EMBER dataset, which could indicate it is unsuitable for further research.

4.3.2 BODMAS. Yang et al. [100] have released a dataset of recent, timestamped, and categorized malware called BODMAS that was collected between August 2019 and September 2020. BODMAS contains features extracted from 77,142 benign samples and both the extracted features and intact PE files for 57,293 malware samples. Yang et al. [100] have highlighted the need for public datasets and benchmarks so researchers can easily compare models. As these samples are timestamped with the first seen time from VirusTotal reports, temporal analysis around taxonomy can be performed [97, 100]. The taxonomy information covers 14 categories, such as Trojans and Ransomware, as well as definitions for 581 families such as wacatac and upatre [100, 101].

4.3.3 SOREL-20M. Harang and Rudd [37] have released SOREL-20M, which contains 20 million samples. Their dataset contains 10 million extracted feature vectors and the intact but disarmed malware files, where the header field flags for headers.subsystem and file header.machine are set to 0. Additionally, the dataset contains metadata and extracted feature vectors from 10 million benign samples. The complete dataset consists of three databases and a folder of the disarmed malware. [37, 38].

4.3.4 VirusShare. This malware repository provides security researchers access to live malware [96]. It contains more than 55 million live malware files with timestamps, hash, detection report,

VirusShare.com - Because Sharing is Caring

Home • Hashes • Research • About

Please login to search and download.

System currently contains 55,030,761 malware samples.

Report for a sample recently added to the system:
401cfc7c59ca397f0b15d5c3a5fa903b6e882504c23a2418f0753b75154a841f

VirusShare info last updated 2022-12-16 00:00:00 UTC

MD5	5dab03a9ebc279457a513e20a2e6f578
SHA1	e46f8473b592e7b0102a92ce9c47df39a2658d4f
SHA256	401cfc7c59ca397f0b15d5c3a5fa903b6e882504c23a2418f0753b75154a841f
SSDeep	3072:vQZosimPTDyXsLCaZg8eMxWBZBs5ZrTiLRAFIY7LIvuiTT:4RTCjBZc3Y7LIvug
Authentihash	9f0afdb2ea9e2738ab7ace5d122d96e5961b1ce7a7e55794d85929297a408d17
Size	187,904 bytes
File Type	PE32 executable, for MS Windows
Mime Type	application/x-dosexec
Extension	exe
TrID	OS/2 Executable (generic) (33.6%) Generic Win/DOS Executable (33.1%) DOS Executable Generic (33.1%)
Detections (42/68)	

APEX	Malicious
Acronis	suspicious
Ad-Aware	Trojan.Obfus.3.Gen
AhnLab-V3	Trojan/Win32.Nabucur.C622804
Antiy-AVL	Virus/Win32.PolyRansom.a
Arcabit	Trojan.Obfus.3.Gen
BitDefender	Trojan.Obfus.3.Gen
BitDefenderTheta	AI:FileInfector.1F8DFD280F
ClamAV	BC.Win.Virus.Ransom-9157.A
Comodo	Packed.Win32.Graybird.B@5hgpd5
Cybereason	malicious.9ebc27
Cynet	Malicious (score: 100)
Cyren	W32/S-accd10d9!Eldorado
DrWeb	Win32.VirLock.1
Elastic	malicious (high confidence)
Emsisoft	Trojan.Obfus.3.Gen (B)
FireEye	Trojan.Obfus.3.Gen
Fortinet	W32/VirRansom.D9F1!tr
GData	Trojan.Obfus.3.Gen
Google	Detected

Fig. 3. Information provided by VirusShare via VirusShare [96].

and other file information, as illustrated in Figure 3. The repository has been widely used in research because it allows for static and dynamic analysis to be performed on any number of intact malware files [10, 56, 73].

4.3.5 Malware Bazaar. This malware repository also provides security researchers access to live malware [1]. It contains more than 700,000 live malware files with timestamps, hash, and, importantly, both category and family information [1]. The repository has been used in recent research because live samples are available that are searchable by category and include family and variant information [79, 91]. However, there is a file download limit of 2,000 samples per day [1].

4.4 Feature Selection

Static and dynamic analysis tools can be used to extract features from malware and benign software, summarized in Tables 4 and 5. The features can then be processed and used to train and test AI malware-detection models. However, feature extraction, engineering, processing, and selection can be problematic, as sophisticated malware use anti-analysis techniques to evade analysis, which makes it very difficult to standardize and optimize feature extraction. Further, the datasets used in research may be unbalanced, whereby the large number of malware families may not be represented equally [99]. This can lead to fragile AI models, where a model may achieve high accuracy with one specific dataset but is not generalizable and achieves poor results with others [3, 26, 45, 90, 99, 103].

The process of optimizing feature extraction for use by AI to detect novel, evasive, and sophisticated malware is a considerable task [10, 12, 45, 90]. Further, static analysis is efficient but is impacted by obfuscation, which is ubiquitous. While sandboxes and DBI can provide more comprehensive and authentic features, they are impacted by anti-sandbox and anti-instrumentation and are slower than static analysis.

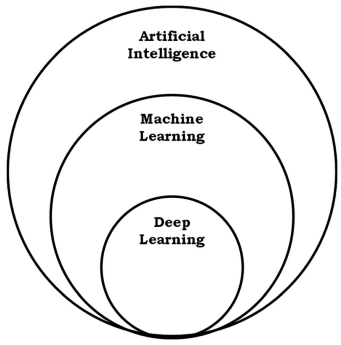

Fig. 4. Artificial intelligence includes both ML and DL via deep learning: Hope or Hype [2].

Several features that can be captured using static and dynamic analysis were provided in Section 4.2, Analysis Techniques. The Windows registry is particularly important, as records can be created and modified so malware can gain persistence on the machine [66]. The CPU uses registers to store values and operations to process the instructions of PE files. That information, and the way a PE file uses the registers, reveals its behavior at the byte level [90]. Further, the files a PE interacts with can be detected, where whatever is created, deleted, or modified can be tracked [90]. API allows programs to communicate, whereby a PE or **Dynamic Link Libraries (DLLs)** file can interact with the **Operating System (OS)** and use its libraries and modules. System calls allow these files to request services from the kernel. The API and system calls and the libraries used can reveal characteristics of the PE or DLL, for example, where it may use OS encryption libraries [69].

A PE file can be disassembled to ASM, which uses opcodes and operands. Accordingly, the opcodes and their operands are the operations and data that in combination are the instructions executed by the CPU. Opcode frequency and sequence can reveal information about PE and DLL characteristics [90]. Converting malware files and network traffic to images is another area of static analysis that is being researched [32, 92]. Network traffic details including IP addresses, ports, encryption, and protocols can also be used to help characterize malicious files [90].

Understanding the underlying phenomena of what syntactic and or semantic features are present in modern malware but not benign software is pivotal. That involves determining what features are optimal to train an AI model that generalizes beyond specific characteristics and seeks to classify never-before-seen malware. Further, determining what AI model is the most accurate and robust when trained with those features is not easily solved. In the following section, a number of research studies are analyzed to highlight the diverse AI models, datasets, and features that have been applied to malware detection.

4.5 Machine Learning vs. Deep Learning

AI is an umbrella term for a number of ML and DL techniques, as illustrated in Figure 4. AI is successfully used across many fields, including language processing, speech recognition, computer vision, and generative models, as well as cybersecurity fields including spam detection and intrusion detection [3, 7]. AI for malware detection is a very active area of research, and yet the deployment of AI models in a production environment is progressing slowly [7].

ML algorithms are trained on structured and labelled data to recognize patterns and make classifications on new data. ML algorithms that are widely used in research include **Logistic Regression (LR), K-nearest neighbor (KNN), Naïve Bayes (NB), Random Forest (RF)**, and **Support Vector Machine (SVM)**.

DL is based on **Artificial Neural Networks (ANN)** and is typically more complex than ML. The basic building block is the artificial neuron, and when connected to other neurons the resultant networks are able to perform complex classification and predictive tasks, in part due to their parallel architecture [106]. A basic ANN contains an input layer, one or more hidden layers, and an output layer [87]. DL can use several hidden layers and normally requires more training data and computational power than ML [7]. Supervised DL requires labelled data, for example, malware or benign, whereas unsupervised DL does not require labelled data. There are several types of DL architectures, including **Deep Neural Networks (DNN), Convolutional Neural Networks (CNNs), Deep Belief Networks (DBN)**, and **Recurrent Neural Networks (RNNs)** [70, 95]. Generative AI is a rapidly developing field where the **Large Language Models (LLM)** are a type of DNN with numerous layers that are crucial for capturing high-level abstractions and handling complex tasks, such as natural language understanding and generation [16, 36].

When the objective is to deploy an AI model in a production environment it must be accurate, detect malware it has never seen before, and be practical and lightweight. This is challenging for a number of reasons. Malware includes trojans, ransomware, crypto miners, botnets, worms, and more, where each category behaves differently and they may not have any commonalities [72]. Further, anti-analysis techniques, covered in detail above, directly impact AI models, as the detection accuracy of a model is dependent on the quality of the features it is trained with [3, 45, 90, 103]. This is further complicated when the actual evasive behavior is used to train AI models [30, 69]. If evasive malware runs on a normal user machine, not in an analysis environment, then it behaves differently and reveals its malicious intent. Consequently, it would defeat an AI model that was trained on the behaviors observed during analysis [30, 69].

The application of AI for malware detection is heterogeneous, where many different AI models, approaches, parameters, analysis techniques, features, and datasets can be used. A number of research studies have been analyzed below to highlight this diversity, and a summary of datasets, features, models, and results are presented in Table 8.

Generative AI that includes LLMs such as ChatGPT and Google Bard are used both defensively and offensively in cybersecurity. Offensively, the LLMs have been used to write malicious code, create polymorphic malware, craft phishing attacks, generate malicious payloads, and more [36, 67]. Defensively they have been used for malware detection, threat intelligence, incidence response plans, and phishing detection. Koide et al. [55] developed a novel method to detect phishing websites using ChatGPT, where a crawler collected information from varied websites and then generated prompts that were presented to ChatGPT to determine if the website was malicious or not [55]. LLMs, specifically **Autoregressive Language Models (ALM)**, that use Transformer architecture are exceptional at assessing probability distributions of text sequences [16, 28, 36]. How the power of LLMs can be leveraged defensively and specifically for malware detection is an ongoing area of research; for example, could ChatGPT be used to generate functional descriptions of opcode sequences to capture the semantics that persist across syntactically distinct malware variants that may use polymorphic or and metamorphic techniques [28, 36]?

Ferrag et al. [28] developed Security LLM, which uses the **Bidirectional Encoder Representations from the Transformers (BERT)** model [89] for threat detection and FalconLLM [42] for incidence response. The EdgeIIoT dataset, which contains numerous attacks related to the IoT, was used to evaluate the efficiency of the model [28]. The performance of several ML and DL models was compared to Security LLM, where RF was the highest-performing ML model at 81%, CNN was the highest-performing DL model at 95%, and Security LLM achieved 98% [28].

Amer and Zelinka [3] have proposed a method to detect Windows malware based on a **Natural Language Processing (NLP)**-inspired contextual understanding and correlation of malicious API call sequences. Word embedding and a clustering similarity matrix were used to generate a vector

Table 8. Summary of Datasets, Features, Models, and Results

Paper	Dataset	Features	Model	Acc. %
[28]	Edge-IIoTset [27]	DDoS-UDP, DDoS-ICMP, SQL-injection, Password, Vulnerability-scanner, DDoS-TCP, DDoS-HTTP, Uploading, Backdoor, Port-Scanning, XSS, Ransomware, MITM, Fingerprinting	SecurityLLM	98
			CNN	95
			Transformer Model	95
			RNN	94
			DNN	93
			RF	81
			KNN79	93
			SVM	78
			DT	67
[3]	[49]: 23,080 malware and 21,116 benign samples	Clusters of dynamically extracted API call sequences	API sequence cluster transition matrices	99.9
	[50]: 151 malware & 69 benign samples			99
	CSDMC2010: 320 malware & 68 benign samples			98.5
	[19]: 7,107 malware & 169 benign samples			98.7
[84]	Benign applications from Software-informer Originally 1,232 ransomware samples across 14 families from VirusShare & VirusTotal Functional samples: 483 ransomware & 754 benign	50 FastICA features from initial 15,972 Cuckoo features	TensorFlow CNN with 512 nodes in Layer 1 & Layer 2 500 epochs	94.84
		Original 15,972 features from Cuckoo JSON report	TensorFlow CNN with 1,024 nodes in Layer 1 & Layer 2 500 epochs	95.96
			RF	90.95
			SVM	89.96
			MC	88.12
[48]	150 ransomware & 150 benign	DNA sequence for 26 most significant using MOGWO & BCS	Proposed LR & AL	87.91
			AB	83.22
			NB	78.52
			DS	75.83
[53]	13,075 malware & 19,747 benign samples	5 static feature vectors, including strings, opcode, API, library, permission, component & environmental features	Multi Modal Keras DNN, 5 initial networks with 2 hidden layers & final network with 2 hidden layers	98.0
[95]	70,140 benign & 69,860 malware samples	EMBER static features	DNN	98.9
			RF	97.0
			DT	96.9
			SVM	96.1
			KNN	95.1
			AB	83.0
			LR	54.0
			NB	53.8

(Continued)

Table 8. Continued

Paper	Dataset	Features	Model	Acc. %
	121,701 benign & 118,717 malware samples	Cuckoo JSON report	SVM	96.1
			CNN	93.6
			DNN	91.0
			RF	89.5
			DT	86.0
			KNN	81.5
			AB	73.3
			LR	67.4
			NB	54.6
	52,245 benign & 50,792 malware samples	Cuckoo JSON report & Python Psutil	CNN	96.6
			DNN	90.4
			RF	89.9
			SVM	89.0
			KNN	85.9
			DT	82.5
			AB	82.1
			LR	57.34
			NB	50.5
[25]	Android 4,354 malware & 5,065 benign samples, only 429 malware & 1,700 benign had network traffic	Static permission, intent & component from manifest.xml Network traffic 18.5 G benign & 19.0 G malicious, converted to images	First NN for static feature vector, CACNN for network traffic images if classified as benign by first NN	99.19
[26]	Android 18,000 malware & 18,000 benign	2,290 API calls & 625 manifest properties TensorFlow models	RNN Bidirection GRU	96.78
			RNN GRU	96.75
			RNN Stacked GRU	96.67
			RNN Stacked LSTM	96.64
			RNN Bidirection LSTM	96.61
			RNN LSTM	96.56
			CNN	95.11
	18,000 malware & 18,000 benign samples Android	2,290 API and 625 manifest properties TensorFlow Lite	RNN GRU	96.75
	70,130 malware 21 families	2,290 API & 625 manifest properties	RNN GRU	94.45
[75]	Real crypto mining traffic with web surfing, video and audio streaming, file transfer, email & others	51 network traffic features from Tstat tool & 8 features from NetFlow metrics	NN	100
			RF	100
			DT	100
			LR	100
			CART	99.99

(Continued)

Table 8. Continued

Paper	Dataset	Features	Model	Acc. %
[17]	43,530 malware samples from VirusShare & 3,591 benign samples from Windows 7 OS	Feature vector from debugger & count of the 610 opcodes in the Intelx86/x64 architecture	RC	99.06
			RSS	99.05
			RF	99.05
			AB	99.02
			Bagging	98.96
			PART	98.51
			IBk	98.34
			LWL	98.33
			J48	98.12
			KStar	98.09
			J RIP	97.83
			REPTree	97.63
			RT	97.06
			DT	94.70
			HT	92.11
			OneR	90.42
			DS	81.91
			ZeroR	49.95
[68]	VirusShare: 31,609, VXHeaven: 20,713 & MALICIA: 11,368 malware & 13,752 benign samples	Images generated from dynamic CFG *Recall as accuracy not presented	YOLO-based CNN	90.26*
			AIS	85.88*
			Simple-CNN	84.85*
			SVM	74.36*
[72]	56 benign and 50 malware samples for 21,800 volatile memory dumps from Ubuntu VMs: DNS server & HTTP server	171 from the memory dumps. All samples used for training, but trained with different dumps & behaviors for same sample	DNS RF	98.7
			DNS ANN	98.2
			DNS DNN	97.9
			DNS SVM	97.8
			DNS KNN	97.7
			DNS LR	95.6
			DNS NB	77.6
			HTTP ANN	99.9
			HTTP KNN	99.9
			HTTP RF	99.8
			HTTP SVM	99.8
			HTTP DNN	99.5
			HTTP LR	99.5
			HTTP NB	94.0
		171 extracted from memory dumps, 8 benign and 8 malware samples used for testing	DNS DNN	95.9
			DNS RF	93.8
			DNS LR	93.5
			DNS ANN	87.9
			DNS SVM	84.5
			DNS KNN	80.4

(Continued)

Table 8. Continued

Paper	Dataset	Features	Model	Acc. %
			DNS NB	67.3
			HTTP KNN	98.9
			HTTP RF	98.5
			HTTP DNN	97.3
			HTTP SVM	96.7
			HTTP NB	96.0
			HTTP ANN	95.0
			HTTP LR	95.0
[103]	VirusChaser: 139,384 malware & 10,475 benign samples, labelled with VirusTotal	79 Pefile utility static features, 513 Cuckoo dynamic	AI-Hydra: RF and MLP	85.1
			Sophos AV	74.9
			Clam AV	74.5
			Bitdefender AV	52
[105]	582 ransomware in 11 families & 942 benign samples	Dynamic, binary strings Windows API, Windows registry, file, system file & directory operations, CAE features reduced from 16,382 to 500 & 100 features	CSPE-R ensemble: CFH, SVM, RF, LR, DNN	93
			CFH-RF100	92
			CFH-SVM500	92
			CFH-SVM100	90
			CFH-LR100	90
			LR	90
			CFH-RF500	89
			CFH-LR500	89
			SVM	88
			RF	80
[43]	BODMAS: 400,000 training, 200,000 testing & 19,000 GAN samples	Static BODMAS features	NLP with BERT & TensorFlow DNN	85.82
[78]	Malware DB: 3,653 malware samples & 554 benign samples, 5 categories: backdoor, password stealer, rogue, trojan & worm	Intel Pin DBI: opcode frequency, memory addresses, memory reads, memory writes, and unaligned memory access	Specialized DNN Average over the five specialized detectors	93.0
			Specialized LR Average over the five specialized detectors	91.0
			General DNN Average over the five types of malware	89.0
			General LR Average over the five types of malware.	87.0

model and malware and benign application clusters from their API call sequence [3]. A transition model was generated from the sequences and clusters to capture the relationship between API calls. **Maximum Likelihood Estimation (MLE)** was used to determine the transition probability between the states of the transition model and to create the malware and benign application cluster transition matrices [3]. For each test sample, the sequence of API calls was extracted and clustered, after which the transition probability for the cluster sequence was calculated and compared to the malware and benign application cluster transition matrices using MLE to make a classification. Four datasets were used to validate the approach, where the average detection accuracy was 99% using the model outlined above and only 52.3% based on non-contextual clustering [3].

Sharmeen et al. [84] used the TensorFlow CNN and ML classifiers—SVM, RF, and **Multi Class Classifier (MCC)**—to determine the most accurate approach for detecting Windows ransomware. The initial dataset contained 1,232 ransomware samples and 1,308 benign samples. However, after deduplication and removing samples that did not run, only 483 ransomware samples and 754 benign samples remained. The significant drop in the number of ransomware samples could be due to the C2 servers were down, or the samples may have been evasive. The Cuckoo sandbox JSON report contained 15,972 features for each of the samples that ran. Sharmeen et al. [84] used **Fast Independent Component Analysis (FastICA)** to reduce the features to sets to 40, 50, 80, and 100. Three different CNN architectures were used, each with a different number of nodes and epochs for each of the FastICA features sets, as well as the complete set of features from Cuckoo. The highest accuracy with the CNN and the FastICA features was obtained with the dataset containing 50 features, where the accuracy was 94.83%. The accuracy obtained with the CNN and the complete set of features was 95.96%, however, the authors noted that it took considerably longer to train the model with this large number of features. The SVM, RF, and MCC achieved accuracies of 89.98%, 90.95%, and 88.12%, respectively. The dataset was small, which could lead to overfitting and models not performing well with other ransomware families and different types of malware.

Khan et al. [48] have used **Multi-Objective Grey Wolf Optimization (MOGWO)** and **Binary Cuckoo Search (BCS)** algorithms to select 26 key features from 16,383 pre-processed features, generating digital DNA sequence and a k-mer frequency vector. Each two-bit pair has four possible values—00, 01, 10, and 11—which were mapped to the four biological DNA bases of adenine **(A), thymine (T), guanine (G)**, and **cytosine (C)**. Further to this, constraints were applied on the composition and k-mer frequency; however, the exact method for this was not specified. The initial dataset included 582 Windows ransomware samples and 942 benign applications; however, this was reduced to 150 ransomware samples and 150 benign samples, where no explicit reason for this reduction was given. Khan et al. [48] reported that: NB achieved a 78.5% detection accuracy; **Decision Stump (DS)** 75.8%; AdaBoost 83.2%; and their proposed active learning algorithm, which is a variation on LR with a feedback loop, achieved an accuracy of 87.9%. Once again, the dataset was small and the models may not perform well with other types of malware.

Vinayakumar et al. [95] have used static and dynamic feature extraction with ML and DL classifiers for Windows malware detection. Different datasets and different techniques were used for static analysis and dynamic analysis. The EMBER dataset and privately collected malware and benign samples were used for static analysis, where the features were extracted using the source code provided with the EMBER dataset. The dataset from research conducted by Rhode et al. [80] and benign applications from Softonic and Source forge were used with Cuckoo dynamic feature extraction. **Windows Static Brain Droid (WSBD)**, which is composed of both ML and DL models, was used in the various experiments. Vinayakumar et al. [95] reported that DL outperformed ML with all datasets and with both static and dynamic feature extraction. With regard to static feature extraction, the DNN was most accurate at 98.9%. With dynamic feature extraction, the CNN achieved the highest accuracy at 96.6%.

Carlin et al. [17] have applied dynamic opcode analysis, which examines the assembly language instructions passed to a CPU at runtime, under escalating data conditions. The debugger OllyDbg V2 was used to run the PE files and extract the opcode traces. The anti anti-debug plugin Strong OD was used to hide the debugger from the malware, but no other evasive techniques were countered. Carlin et al. [17] performed a count of each of the 610 Intel x86/x64 architecture opcodes to create the feature vector. Carlin et al. [17] further identified a number of gaps between academia and production deployment. Given the large datasets available, underfitting of AI models should be a concern, where the limitations of memory and time may constrain the models. That is, underfitting

should be a problem for AI models trained on large datasets; however, this does not appear to be the case, which indicates the datasets underpinning current research are inadequate [17]. Further, little attention has been paid to the application of AI to escalating data, where, instead, studies have focused on single static datasets. Carlin et al. suggest that most research studies highlight the exponential growth of malware but use fixed size datasets. Carlin [17] created a dataset of 43,530 malware samples from VirusShare, with 3,591 benign applications from the Windows 7 OS. Initially a broad search across all available WEKA 3-9 ML classifiers was performed to determine which performed the best with the feature vectors. Two ML classifiers were dropped due to their very long training times, and 23 ML classifiers were used in the next stage. Several phases of hyper parameter tuning and feature selection optimization were then performed. Of the 23 ML classifiers, 15 achieved a greater than 98% detection accuracy, where Carlin et al. [17] stated that RF was the optimal classifier for several reasons; it was fast to train, it was fast to classify, it was able to handle data explosion, and was robust and highly accurate.

Nguyen et al. [68] have used a CNN and a **You Only Look Once (YOLO)**-based CNN. The dataset contained 13,752 benign Windows applications and 50,000 malware samples from several repositories. Nguyen et al. [68] used the **Binary Emulation for Pushdown Model (BE-PUM)** analysis tool, which dynamically runs the PE file and generates a **Control Flow Graph (CFG)**. A CFG is a directed graph of the PE file under analysis, where each vertex corresponds to an instruction in the file and the transitions between vertices are the execution flow of the program [68]. BE-PUM supports lazy binding, where if an instruction at a location changes, then the CFG is updated to reflect the operation of the program [68]. Further, each vertex is considered a state and has three segments: the value of the register, the flag, and the memory. An innovative approach was used in this scenario to reduce the size of the memory component, where its contents were too large to include every vertex. That is, an MD5 hash of the memory was generated for each point [68]. The CFG vertices were then converted to an adjacency matrix, which transformed the values and was used to generate the pixel values of an RGB image. Nguyen et al. [68] note there are limitations with this approach, particularly the time it takes to generate the features and train the models. Accuracy was not reported in this context, and the other metrics were at the low end of comparable studies, but the approach to feature generation is novel.

Yoo et al. [103] have created AI-Hydra, which uses a hybrid approach for malware detection that implements **Multi-Layer Perceptron (MLP)**, RF, and a rule-based voting method to enhance detection performance. The Python library Pefile was used to extract 79 static features, and the Cuckoo sandbox was used to extract 513 dynamic features. To evaluate appropriate classifiers, 10 popular ML and DL models were tested using Weka and a dataset that contained 149,859 malware samples from Virus Chaser and 10,475 benign Windows applications that were labelled with VirusTotal. From the results, RF and MLP were chosen for implementation in AI-Hydra [103]. A part of the **Korean Internet and Security Agency (KISA)** 2017 dataset was used to evaluate AI-Hydra, with 6,395 benign and malicious samples. KISA is considered a challenging dataset, as it contains benign files that exhibit malicious behavior [103]. Including the time for feature extraction, AI-Hydra took an average of 60.9 seconds for a detection accuracy of 85.7% with a **False Positive (FP)** rate of 16.1%. The AV applications BitDefender, ClamAV, and Sophos were also tested, and while the detection accuracy of the AV products was lower than AI-Hydra, they were much faster, and their FP was very low to zero [103].

Zahoora et al. [105] have created a novel multi-phase framework, **Cost Sensitive Pareto Ensemble (CSPE-R)**, to detect novel Windows ransomware. The dataset contained 582 malware samples across 11 families from Sgandurra et al. [82] and 942 benign samples. The dataset was split into seen and unseen samples, as the goal was to detect zero-day ransomware. In the feature extraction phase, **Core Feature Hunting (CFH)** was implemented using an unsupervised

Contractive Autoencoder (CAE) to extract core ransomware features. In the second phase, the vectors were extracted from the core features identified in the first phase. In the third phase, SVM, RF, and LR classifiers were trained on the feature vectors and a cost matrix was generated. In the last phase, a Pareto optimality strategy and cost matrix were used to select and aggregate the classifiers. The proposed CSPE-R ensemble classifier achieved an accuracy of 93% for detecting never-before-seen ransomware [105].

Ismail et al. [43] have used a natural language processing model, **Bidirectional Encoder Representations from Transformers (BERT)**, to detect Windows malware. The dataset consisted of 600,000 samples from EMBER and a **Generative Adversarial Network (GAN)** was used to generate 19,000 new malware examples using the EMBER metadata file. The samples were tokenized, and a Self-Supervised TensorFlow model was implemented. The model was pre-trained using unlabeled data and was fine-tuned with a much smaller labelled dataset. The model achieved a detection accuracy of 85.52% [43].

Ponomarev et al. [78] have implemented an ensemble detector with feature extraction and LR and NN models on a **Field Programmable Gate Array (FPGA)**. The **Hardware Malware Detectors (HMD)** were specialized for a particular family of malware. The dataset consisted of 3,653 malware samples across five categories: Backdoor, Password Stealer, Rogue, Trojan, and Worm from MalwareDB, and 554 benign samples. The specialized detectors were trained on only 1 of the 5 different malware categories, and Intel Pin was used to collect dynamic traces after 150 system calls for a duration of 1,500 system calls or 15 million committed instructions, whichever came first. The detectors did not generalize well: When new types of malware were introduced, the performance declined significantly [78]. The specialized models achieved accuracies ranging from 87% to 93% when classifiying the type of malware it had been trained with [78].

The study by Pastor et al. [75] supports the assertion that AI models are dependent on the quality of the features they are trained on. It is challenging to profit from cryptocurrency mining, and this has led to the creation of legitimate mining pools but also illegal botnets. In both these cases, the mining traffic is encrypted and can use proxy servers. Pastor et al. [75] have captured 32 million packets of network traffic that included real, encrypted mining pool traffic amongst web surfing, video and audio streaming, file transfer, email, and others. Pastor et al. [75] used a DNN and ML classifiers RF and LR, all of which achieved 100% detection accuracy of the mining traffic. Importantly, the captured network traffic was 100% genuine, which provides support for the argument that if genuine features can be extracted from sophisticated and evasive malware, then the resultant AI model should be very accurate.

The deployment of AI malware detection on mobile devices is problematic because of performance constraints. Consequently, most Android malware detection is done on the server side, that is, within the markets [25]. Feng et al. [25] have proposed a new approach for mobile device deployment using TensorFlow light. Feng et al. [25] used static feature extraction, network traffic data, a DNN, and a **Cascading CNN (CACNN)** with a TensorFlow backend for the proposed Android malware detection system. Static analysis was used to generate feature vectors from the manifest.xml properties: permission, intent, and component information. Furthermore, the traffic data was converted to greyscale images. The static features were used by the first DNN, where only those applications that were classified as benign had their network traffic data, if it was available, input into the CACNN to confirm the benign classification. The model proposed by Feng et al. [25] achieved a high detection accuracy of 99.19%.

Feng et al. [26] created MobiTive that uses a TensorFlow Lite **Gated Recurrent Unit (GRU), Recurrent Neural Network (RNN)**, static analysis, and a performance-based feature selection algorithm to detect Android malware on the device. Feng et al. [26] indicate that most Android malware detection is done on the server side and not on the device and suggest that, with recent

hardware advances, lightweight AI models could provide a foundation for universal deployment. During experimentation, Feng et al. [26] used varied RNN architectures, features, and dataset sizes. The features were extracted directly from the binary files, where they were not decompiled but rather extracted by analyzing the structure of the .dex file and manifest.xml, which can be obtained from unzipping an APK. A feature dictionary of 613 manifest properties and 1,509 API calls was initially created to which an additional 781 new API calls and 12 new manifest properties were added from Symantec Threats. A dataset of 18,000 malware samples and 18,000 benign samples was assembled from numerous sources where VirusTotal was used to validate the sample labels. Feng et al. [26] note that DL models are often customized, where the size, efficiency, and memory consumption are altered before deploying to constrained platforms, such as Android, and that this can affect the accuracy of the trained models. However, both the trained TensorFlow RNN GRU and the migrated TensorFlow lite RNN GRU that was deployed to a mobile device in their experiments achieved a detection accuracy of 96.75%. The TensorFlow lite RNN GRU was deployed on six different Android phones where the total time, including unzipping, feature extraction, and prediction, ranged from 0.46 to 3.96 seconds. Given the model is run on a device, if a file is detected as malicious, then its name and checksum could be saved in a local database, and this could also be pushed to the cloud where other devices could synchronize with it [26]. A number of limitations were highlighted in their study, including that while static analysis is faster and dynamic analysis does require more resources, it may provide higher-quality features that are more discriminatory between malware and benign applications. Further, MobiTive may not detect new malware, because just the manifest properties and API calls do not provide enough information for the model to generalize to new types of malware. Feng et al. [26] also highlight that an adversarial attack would not break the entire functionality of MobiTive and suggest that more focus should be on the evasive techniques used by malware.

Kim et al. [53] have used a multimodal deep learning approach for Android malware detection. Their initial dataset contained 41,260 malware and benign samples. Static analysis and disassembly tools were used to extract features; however, only 78% of the dex files, 79% of the manifest files, and 69% of the .so files could be analyzed. No explicit reasons were given for this, but it is possible that the samples that could not be analyzed used obfuscation techniques. The resulting dataset contained 13,075 malware samples and 19,747 benign samples. Kim et al. [53] used seven features: string feature, method opcode feature, method API feature, shared library function opcode feature, permission feature, component feature, and environmental feature. However, permission, component, and environment features were merged into one vector, resulting in 5 vectors for each sample. Five parallel DNNs were used, one for each of the feature vectors. The last layers of the initial five are the input for the final DNN that produces the classification result. The Keras library was used, and this multimodal DL architecture achieved 98% accuracy [53].

While malware has typically targeted Windows and Android, there are increasing reports of sophisticated ransomware and crypto miners that target Linux virtual servers that are widely used by cloud service providers [72]. Panker and Nissim [72] have generated features from volatile memory snapshots of virtual Linux servers and tested their features with numerous ML and DL models to detect known and unknown malware. The experimental setup included 56 benign applications and 50 malware samples from nine categories run on Ubuntu DNS and HTTP VMs. As the behavior of the applications changed over time, multiple memory snapshots were captured for each benign and malware sample. The feature set encompassed a broad range of process interactions, information gathering, and generic behaviors represented by 171 individual features [72]. DL and ML models were used, and the KNN and RF classifiers achieved the highest accuracies with never-before-seen samples at 98.9% and 98.5% respectively.

5 DISCUSSION

It is challenging to create a highly accurate and robust AI model to detect malware for numerous reasons. Sophisticated malware that may be evasive, novel, or use AI are challenging to analyze and extract genuine features. Evasive malware can use obfuscation to limit static analysis and anti-analysis techniques to change behavior and hide its malicious intent [3, 29, 30, 48, 72, 74, 77, 84, 103]. Novel and AI-powered malware present even greater challenges [17, 22, 73, 84, 95]. Novel malware that has not been seen before and may exploit zero-day vulnerabilities, and AI-powered malware that embed the target class and instance in an NN, may be immune to analysis unless a triggered sample is captured [46, 54].

There are numerous malware analysis tools used by security researchers, where each have their limitations. Maffia et al. [59] have shown that approximately 60% of the malware samples they analyzed used some form of obfuscation, which severely limits static analysis. Dynamic analysis tools, including debuggers, sandboxes, and DBI, are not affected by obfuscation, as they run the malware in a controlled environment. However, evasive malware can implement multiple types of anti-debug, anti-sandbox, anti-instrumentation, and anti-VM techniques to defeat dynamic analysis [69, 72]. Kim et al. [51] and Polino et al. [77] have reported anti-instrumentation techniques, which target DBI, in 16.21% and 15.6% of the analyzed malware samples. Contrastingly, Galloro et al. [30] and Sharma et al. [83] have reported the more general anti-analysis techniques in 80% and 99.36% of the malware samples analyzed. This indicates the evasive techniques used by malware authors are not focused on DBI.

DBI is immune to anti-analysis techniques that target VMs, debuggers, and sandboxes but can be defeated by anti-instrumentation techniques. Consequently, several methods have been developed to defeat anti-instrumentation used by malware, and the countermeasures are comprehensively detailed in the existing body of research [23, 29, 30, 74, 77]. Further, DBI allows deep and precise control of the file under analysis, which in turn allows genuine, high-quality, and comprehensive feature extraction.

The accuracy and effectiveness of an AI model is dependent on the quality and authenticity of the features it is trained with [3, 45, 90, 103]. Further, the quality and authenticity of the features is dependent on the quality of the analysis tool and the repository. Many of the papers analyzed in this survey highlight the explosive growth of malware, yet many studies use small, private, curated, and unbalanced datasets, which makes it difficult to measure progress, compare results, and can lead to overfitting of AI models. Creating an open repository of malware and benign applications is challenging, as there are copyright restrictions on legitimate software. However, generative AI has the potential to produce synthetic instances of both benign and malicious samples. It would be advantageous to the research community if different approaches could be applied to a versioned and regularly updated repository, ideally with a linked DBI framework to extract authentic features.

Determining what syntactic and or semantic features are most discriminatory between malware and benign applications and what feature representation is optimal for varying AI models is not easily resolved. The selection of quality features is pivotal to creating an AI model that generalizes beyond specific samples to detect variants and never-before-seen malware. Each study examined employed a distinct dataset and a specific set of features. Despite the difficulties in comparing these studies, it is noteworthy that models achieving an accuracy exceeding 99% used dynamically extracted features: API calls, network traffic, opcode sequence, and memory snapshots.

Diverse types of AI are used for malware detection, where many different models, architectures, approaches, and parameters are used. Comparing the performance of the different models is complicated by the use of different datasets, analysis tools, and features. In the papers that compared

ML and DL using the same dataset and features, there was no clear standout performer, where both ML and DL achieved the highest accuracies [72, 75, 78, 84, 95]. The overarching issue here, however, is that small, unbalanced, or otherwise poor-quality datasets are being used with analysis tools that are easily defeated by evasive malware.

6 CONCLUSION

In this article, we systematically surveyed the state-of-the-art methods across five critical aspects of building an accurate and robust AI malware-detection model: malware sophistication, analysis techniques, malware repositories, feature selection, and ML vs. DL and generative AI.

We showed there are numerous challenges for malware detection using AI. The sophisticated and evasive techniques used by malware authors impact the various analysis tools differently, where there is no general solution. Static analysis is fast and efficient but is limited by the widespread use of obfuscation. Dynamic analysis is not impacted by obfuscation but is defeated by ubiquitous anti-analysis techniques and requires more computational power. The dynamic analysis tool DBI is least impacted by anti-analysis and allows deep and precise control of the file under analysis, which in turn allows genuine, high-quality, and comprehensive feature extraction.

Creating a public open repository that includes benign and malicious files is challenging. Consequently, researchers tend to use small, private, and unbalanced datasets, which makes it difficult to measure progress and compare results and can lead to overfitting of AI models. The importance of a large open repository that implements version control, is regularly updated, with a linked DBI framework to extract authentic features is clear, where we plan to investigate this form of implementation in future research.

It is evident that the accuracy and efficiency of an AI model is dependent on the quality and authenticity of the features it is trained on. Poor-quality features from small datasets can lead to a situation where a model may achieve high accuracy with one specific dataset but is fragile, not generalizable, and does not perform well with new variants or novel data. Finding the most effective AI model and set of high-quality features that can reliably detect sophisticated and novel malware is not easily resolved. Consequently, future research should focus on the extraction of genuine features from a high-quality repository as the foundation for building an AI model that could be deployed in a production environment.

ACKNOWLEDGMENTS

We would like to thank Michael Stein from Edith Cowan University for his editorial assistance.

REFERENCES

[1] Abuse.ch. 2023. Malware Bazaar. Retrieved from https://bazaar.abuse.ch

[2] Mohiuddin Ahmed and A. K. M. Najmul Islam. 2020. Deep learning: Hope or hype. *Ann. Data Sci.* 7, 3 (2020), 427–432. DOI: https://doi.org/10.1007/s40745-019-00237-0

[3] Eslam Amer and Ivan Zelinka. 2020. A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence. *Comput. Secur.* 92 (2020). DOI: https://doi.org/10.1016/j.cose.2020.101760

[4] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An open dataset for training static PE malware machine learning models. *arXiv preprint arXiv:1804.04637* (2018).

[5] Hyrum S. Anderson and Phil Roth. 2018. EMBER Elastic Malware Benchmark for Empowering Researchers. (2018). Retrieved from https://github.com/elastic/ember

[6] Giovanni Apruzzese, Michele Colajanni, Luca Ferretti, Alessandro Guido, and Mirco Marchetti. 2018. On the effectiveness of machine and deep learning for cyber security. In *10th International Conference on Cyber Conflict (CyCon'18)*. 371–390. DOI: https://doi.org/10.23919/CYCON.2018.8405026

[7] Giovanni Apruzzese, Pavel Laskov, Edgardo Montes de Oca, Wissam Mallouli, Luis Búrdalo Rapa, Athanasios Vasileios Grammatopoulos, and Fabio Di Franco. 2022. The role of machine learning in cybersecurity. *Digital Threats: Research and Practice* 4, 1 (2022), 1–38.

[8] Ömer Aslan Aslan and Refik Samet. 2020. A comprehensive review on malware detection approaches. *IEEE Access* 8 (2020), 6249–6271. DOI: https://doi.org/10.1109/ACCESS.2019.2963724

[9] Jan von der Assen, Alberto Huertas Celdrán, Adrian Zermin, Raffael Mogicato, Gérôme Bovet, and Burkhard Stiller. 2023. SecBox: A lightweight container-based sandbox for dynamic malware analysis. In *IEEE/IFIP Network Operations and Management Symposium*. 1–3. DOI: https://doi.org/10.1109/NOMS56928.2023.10154293

[10] Sana Aurangzeb, Rao Naveed Bin Rais, Muhammad Aleem, Muhammad Arshad Islam, and Muhammad Azhar Iqbal. 2021. On the classification of Microsoft-Windows ransomware using hardware profile. *PeerJ. Comput. Sci.* 7 (2021), e361. DOI: https://doi.org/10.7717/peerj-cs.361

[11] AV-Test. 2022. The Independant IT Security Institute. Retrieved from https://www.av-test.org/en/statistics/malware/

[12] Elshan Baghirov. 2021. Techniques of malware detection: Research review. In *IEEE 15th International Conference on Application of Information and Communication Technologies (AICT'21)*. 1–6. DOI: https://doi.org/10.1109/AICT52784.2021.9620415

[13] Eduardo Berrueta, Daniel Morato, Eduardo Magana, and Mikel Izal. 2020. Open repository for the evaluation of ransomware detection tools. *IEEE Access* 8 (2020), 65658–65669. DOI: https://doi.org/10.1109/ACCESS.2020.2984187

[14] Tais Fernanda Blauth, Oskar Josef Gstrein, and Andrej Zwitter. 2022. Artificial intelligence crime: An overview of malicious use and abuse of AI. *IEEE Access* 10 (2022). DOI: https://doi.org/10.1109/ACCESS.2022.3191790

[15] Rodrigo Rubira Branco, Gabriel Negeira Barbosa, and Pedro Drimel Neto. 2012. Scientific but Not Academical Overview of Malware Anti-Debugging, Anti-Disassembly and AntiVM Technologies. Retrieved from https://kernelhacking.com/rodrigo/docs/blackhat2012-paper.pdf

[16] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *34th International Conference on Neural Information Processing Systems (NIPS'20)*. Curran Associates Inc., Red Hook, NY, Article 159, 25 pages.

[17] Domhnall Carlin, Philip O'Kane, and Sakir Sezer. 2019. A cost analysis of machine learning using dynamic runtime opcodes for malware detection. *Comput. Secur.* 85 (2019), 138–155. DOI: https://doi.org/10.1016/j.cose.2019.04.018

[18] Ero Carrera. 2022. Pefile. Retrieved from https://github.com/erocarrera/pefile

[19] Ferhat Ozgur Catak and Ahmet Faruk Yazı. 2021. A Benchmark API Call Dataset for Windows PE Malware Classification. (2021). arxiv:cs.CR/1905.01999

[20] L. Caviglione, M. Choraś, I. Corona, A. Janicki, W. Mazurczyk, M. Pawlicki, and K. Wasielewska. 2021. Tight arms race: Overview of current malware threats and trends in their detection. *IEEE Access* 9 (2021), 5371–5396. DOI: https://doi.org/10.1109/ACCESS.2020.3048319

[21] X. Chen, J. Andersen, Z. Morley Mao, M. Bailey, and J. Nazario. 2008. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. *International Conference on Dependable Systems and Networks*. 177–186. DOI: https://doi.org/10.1109/DSN.2008.4630086

[22] Hamid Darabian, Sajad Homayounoot, Ali Dehghantanha, Sattar Hashemi, Hadis Karimipour, Reza M. Parizi, and Kim-Kwang Raymond Choo. 2020. Detecting cryptomining malware: A deep learning approach for static and dynamic analysis. *J. Grid Comput.: Grids Cloud Federat.* 18, 2 (2020), 293–303. DOI: https://doi.org/10.1007/s10723-020-09510-6

[23] Daniele Cono D'Elia, Emilio Coppa, Simone Nicchi, Federico Palmaro, and Lorenzo Cavallaro. 2019. SoK using dynamic binary instrumentation for security (and how you may get caught red handed). In *ACM Asia Conference on Computer and Communications Security*. 15–27. DOI: https://doi.org/10.1145/3321705.3329819

[24] Thomas W. Edgar and David O. Manz. 2017. *Research Methods for Cyber Security*. Syngress. Retrieved from https://www.sciencedirect.com/science/book/9780128053492

[25] Jiayin Feng, Limin Shen, Zhen Chen, Yuying Wang, and Hui Li. 2020. A two-layer deep learning method for Android malware detection using network traffic. *IEEE Access* 8 (2020), 125786–125796. DOI: https://doi.org/10.1109/ACCESS.2020.3008081

[26] Ruitao Feng, Sen Chen, Xiaofei Xie, Guozhu Meng, Shang-Wei Lin, and Yang Liu. 2021. A performance-sensitive malware detection system using deep learning on mobile devices. *IEEE Trans. Inf. Forens. Secur.* 16 (2021), 1563–1578. DOI: https://doi.org/10.1109/TIFS.2020.3025436

[27] Mohamed Amine Ferrag, Othmane Friha, Djallel Hamouda, Leandros Maglaras, and Helge Janicke. 2022. Edge-IIoTset: A new comprehensive realistic cyber security dataset of IoT and IIoT applications for centralized and federated learning. *IEEE Access* 10 (2022), 40281–40306. DOI: https://doi.org/10.1109/ACCESS.2022.3165809

[28] Mohamed Amine Ferrag, Mthandazo Ndhlovu, Norbert Tihanyi, Lucas C. Cordeiro, Merouane Debbah, and Thierry Lestable. 2023. Revolutionizing Cyber Threat Detection with Large Language Models. (2023). arxiv:cs.CR/2306.14263

[29] Ailton Santos Filho, Ricardo J. Rodríguez, and Eduardo L. Feitosa. 2022. Evasion and countermeasures techniques to detect dynamic binary instrumentation frameworks. *Digit Threats: Res. Pract.* 3, 2 (2022), 1–28. DOI: https://doi.org/10.1145/3480463

[30] Nicola Galloro, Mario Polino, Michele Carminati, Andrea Continella, and Stefano Zanero. 2022. A systematical and longitudinal study of evasive behaviors in windows malware. *Comput. Secur.* 113 (2022).

[31] P. García-Teodoro, J. A. Gómez-Hernández, and A. Abellán-Galera. 2022. Multi-labeling of complex, multi-behavioral malware samples. *Comput. Secur.* 121 (2022). DOI: https://doi.org/10.1016/j.cose.2022.102845

[32] Sahil Garg, Kuljeet Kaur, Neeraj Kumar, Georges Kaddoum, Albert Y. Zomaya, and Rajiv Ranjan. 2019. A hybrid deep learning-based model for anomaly detection in cloud datacenter networks. *IEEE Trans. Netw. Serv. Manag.* 16, 3 (2019), 924–935.

[33] Daniel Gibert, Carles Mateu, and Jordi Planes. 2020. The rise of machine learning for detection and classification of malware: Research developments, trends and challenges. *J. Netw. Comput. Applic.* 153 (2020), 102526. DOI: https://doi.org/10.1016/j.jnca.2019.102526

[34] Daniel Gibert, Carles Mateu, Jordi Planes, and Joao Marques-Silva. 2021. Auditing static machine learning anti-malware tools against metamorphic attacks. *Comput. Secur.* 102 (2021). DOI: https://doi.org/10.1016/j.cose.2020.102159

[35] Claudio Guarnieri, Alessandro Tanasi, Jurriaan Bremer, Mark Schloesser, Koen Houtman, Ricardo van Zutphen, and Ben de Graaff. 2023. Cuckoo Automated Malware Analysis. Retrieved from https://cuckoosandbox.org/

[36] Maanak Gupta, Charankumar Akiri, Kshitiz Aryal, Eli Parker, and Lopamudra Praharaj. 2023. From ChatGPT to ThreatGPT: Impact of generative AI in cybersecurity and privacy. *IEEE Access* 11 (2023), 80218–80245. DOI: https://doi.org/10.1109/ACCESS.2023.3300381

[37] Richard Harang and Ethan M. Rudd. 2020. SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection. (2020). arxiv:cs.CR/2012.07634

[38] Richard E. Harang and Ethan M. Rudd. 2022. SoReL-20M. Retrieved from https://github.com/sophos/SOREL-20M

[39] Hinne Hettema. 2021. Rationality constraints in cyber defense: Incident handling, attribution and cyber threat intelligence. *Comput. Secur.* 109 (2021). DOI: https://doi.org/10.1016/j.cose.2021.102396

[40] Manabu Hirano, Ryo Hodota, and Ryotaro Kobayashi. 2022. RanSAP: An open dataset of ransomware storage access patterns for training machine learning models. *Forens. Sci. Internat.: Digit. Investig.* 40 (2022). DOI: https://doi.org/10.1016/j.fsidi.2021.301314

[41] IBMSecurity. 2022. Cost of a Data Breach Report 2022. Retrieved from https://www.ibm.com/downloads/cas/3R8N1DZJ

[42] Technology Innovation Institute. 2023. Falcon LLM. Retrieved from https://falconllm.tii.ae/index.html

[43] Setia Juli Irzal Ismail, Hafiz Pradana Gemilang, Budi Rahardjo and Hendrawan. 2022. Self-supervised learning implementation for malware detection. *2022 8th International Conference on Wireless and Telematics (ICWT)*, Yogyakarta, Indonesia, 2022, 1–6. DOI: 10.1109/ICWT55831.2022.9935463

[44] Chani Jindal, Christopher Salls, Hojjat Aghakhani, Keith Long, Christopher Kruegel, and Giovanni Vigna. 2019. Neurlux: Dynamic malware analysis without feature engineering. In *35th Annual Computer Security Applications Conference.* 444–455.

[45] Yuki Kajiwara, Junjun Zheng, and Koichi Mouri. 2021. Performance comparison of training datasets for system call-based malware detection with thread information. *IEICE Trans. Inf. Syst.* E104D, 12 (2021), 2173–2183.

[46] Nektaria Kaloudi and Jingyue Li. 2020. The AI-based cyber threat landscape a survey. *ACM Comput. Surv.* 53, 1 (2020), 1–34. DOI: https://doi.org/10.1145/3372823

[47] Houssain Kettani and Polly Wainwright. 2019. On the top threats to cyber systems. In *IEEE 2nd International Conference on Information and Computer Technologies (ICICT'19).* 175–179. DOI: https://doi.org/10.1109/INFOCT.2019.8711324

[48] Firoz Khan, Cornelius Ncube, Lakshmana Kumar Ramsay, Seifedine Kadry, and Yunyoung Nam. 2020. A digital DNA sequencing engine for ransomware detection using machine learning. *IEEE Access* 8 (2020), 119710–119719. DOI: https://doi.org/10.1109/ACCESS.2020.3003785

[49] Youngjoon Ki, Eunjin Kim, and Huy Kang Kim. 2015. A novel approach to detect malware based on API call sequence analysis. *Int. J. Distrib. Sensor Netw.* 11, 6 (2015). DOI: https://doi.org/10.1155/2015/659101

[50] Chan Woo Kim. 2018. NtMalDetect: A Machine Learning Approach to Malware Detection Using Native API System Calls. (2018). arxiv:cs.CR/1802.05412

[51] Minho Kim, Haehyun Cho, and Jeong Hyun Yi. 2022. Large-scale analysis on anti-analysis techniques in real-world malware. *IEEE Access* 10 (2022), 75802–75815.

[52] Sungjoong Kim, Seongkyu Yeom, Haengrok Oh, Dongil Shin, and Dongkyoo Shin. 2020. Automatic malicious code classification system through static analysis using machine learning. *Symmetry* 13, 1 (2020), 35. DOI: https://doi.org/10.3390/sym13010035

[53] TaeGuen Kim, BooJoong Kang, Mina Rho, Sakir Sezer, and Eul Gyu Im. 2019. A multimodal deep learning method for Android malware detection using various features. *IEEE Trans. Inf. Forens. Secur.* 14, 3 (2019), 773–788. DOI: https://doi.org/10.1109/TIFS.2018.2866319

[54] Dhilung Kirat, Jiyong Jang, and Marc Stoecklin. 2018. DeepLocker Concealing Targeted Attacks with AI Locksmithing. Retrieved from https://i.blackhat.com/us-18/Thu-August-9/us-18-Kirat-DeepLocker-Concealing-Targeted-Attacks-with-AI-Locksmithing.pdf

[55] Takashi Koide, Naoki Fukushi, Hiroki Nakano, and Daiki Chiba. 2023. Detecting Phishing Sites Using ChatGPT. (2023). arxiv:cs.CR/2306.05816

[56] Ke Kong, Zhichao Zhang, Zi-Yuan Yang, and Zhaoxin Zhang. 2022. FCSCNN: Feature centralized Siamese CNN-based Android malware identification. *Comput. Secur.* 112 (2022). DOI: https://doi.org/10.1016/j.cose.2021.102514

[57] Nir Kshetri. 2021. Economics of artificial intelligence in cybersecurity. *IT Professional* 23, 5 (2021), 73–77. DOI: https://doi.org/10.1109/MITP.2021.3100177

[58] Sanjeev Kumar, B. Janet, and Subramanian Neelakantan. 2022. Identification of malware families using stacking of textural features and machine learning. *Expert Syst. Applic.* 208 (2022), 118073. DOI: https://doi.org/10.1016/j.eswa.2022.118073

[59] Lorenzo Maffia, Dario Nisi, Platon Kotzias, Giovanni Lagorio, Simone Aonzo, and Davide Balzarotti. 2021. Longitudinal Study of the Prevalence of Malware Evasive Techniques. Retrieved from https://arxiv.org/abs/2112.11289

[60] Ericsson Marin, Mohammed Almukaynizi, Eric Nunes, and Paulo Shakarian. 2018. Community finding of malware and exploit vendors on darkweb marketplaces. In *1st International Conference on Data Intelligence and Security (ICDIS'18)*. 81–84. DOI: https://doi.org/10.1109/ICDIS.2018.00019

[61] Luis Francisco Martín Liras, Adolfo Rodríguez de Soto, and Miguel A. Prada. 2021. Feature analysis for data-driven APT-related malware discrimination. *Comput. Secur.* 104 (2021). DOI: https://doi.org/10.1016/j.cose.2021.102202

[62] Zaheer Masood, Raza Samar, and Muhammad Asif Zahoor Raja. 2019. Design of a mathematical model for the Stuxnet virus in a network of critical control infrastructure. *Comput. Secur.* 87 (2019). DOI: https://doi.org/10.1016/j.cose.2019.07.002

[63] Microsoft. 2021. NtQuerySystemInformation Function (winternl.h). Retrieved from https://docs.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation

[64] Microsoft. 2022. OpenProcess Function (processthreadsapi.h). Retrieved from https://docs.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-openprocess

[65] Cody Miller, Dae Glendowne, Henry Cook, DeMarcus Thomas, Chris Lanclos, and Patrick Pape. 2017. Insights gained from constructing a large scale dynamic analysis platform. *Digit. Investig.: Supplem.* 22, Supplement (2017), S48–S56. DOI: https://doi.org/10.1016/j.diin.2017.06.007

[66] Ricardo Misael Ayala Molina, Sadegh Torabi, Khaled Sarieddine, Elias Bou-Harb, Nizar Bouguila, and Chadi Assi. 2022. On ransomware family attribution using pre-attack paranoia activities. *IEEE Trans. Netw. Serv. Manag.* 19, 1 (2022), 19–36.

[67] Aaron Mulgrew. 2023. I Built a Zero Day Virus with Undetectable Exfiltration using Only ChatGPT Prompts. Retrieved from https://www.forcepoint.com/blog/x-labs/zero-day-exfiltration-using-chatgpt-prompts

[68] Minh Hai Nguyen, Dung Le Nguyen, Xuan Mao Nguyen, and Tho Thanh Quan. 2018. Auto-detection of sophisticated malware using lazy-binding control flow graph and deep learning. *Comput. Secur.* 76 (2018), 128–155. DOI: https://doi.org/10.1016/j.cose.2018.02.006

[69] Matthew Nunes, Pete Burnap, Philipp Reinecke, and Kaelon Lloyd. 2022. Bane or Boon: Measuring the effect of evasive malware on system call classifiers. *J. Inf. Secur. Applic.* 67 (2022). DOI: https://doi.org/10.1016/j.jisa.2022.103202

[70] Aslan Ömer and Yilmaz Abdullah Asim. 2021. A new malware classification framework based on deep learning algorithms. *IEEE Access* 9 (2021), 87936–87951. DOI: https://doi.org/10.1109/ACCESS.2021.3089586

[71] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. 2019. Dynamic malware analysis in the modern era—A state of the art survey. *ACM Comput. Surv.* 52, 5 (2019), 1–48. DOI: https://doi.org/10.1145/3329786

[72] Tomer Panker and Nir Nissim. 2021. Leveraging malicious behavior traces from volatile memory using machine learning methods for trusted unknown malware detection in Linux cloud environments. *Knowl.-based Syst.* 226 (2021). DOI: https://doi.org/10.1016/j.knosys.2021.107095

[73] Anil Singh Parihar, Shashank Kumar, and Savya Khosla. 2022. S-DCNN: Stacked deep convolutional neural networks for malware classification. *Multim. Tools Applic.* 81, 21 (2022), 30997–31015. DOI: https://doi.org/10.1007/s11042-022-12615-7

[74] Juhyun Park, Yun-Hwan Jang, Soohwa Hong, and Yongsu Park. 2019. Automatic detection and bypassing of anti-debugging techniques for Microsoft Windows environments. *Adv. Electric. Comput. Eng.* 19, 2 (2019), 23–28. DOI: https://doi.org/10.4316/AECE.2019.02003

[75] A. Pastor, A. Mozo, S. Vakaruk, D. Canavese, D. R. López, L. Regano, S. Gómez-Canaval, and A. Lioy. 2020. Detection of encrypted cryptomining malware connections with machine and deep learning. *IEEE Access* 8 (2020), 158036–158055. DOI: https://doi.org/10.1109/ACCESS.2020.3019658

[76] Alexis M. Pitney, Spencer Penrod, Molly Foraker, and Suman Bhunia. 2022. A systematic review of 2021 microsoft exchange data breach exploiting multiple vulnerabilities. *2022 7th International Conference on Smart and Sustainable Technologies (SpliTech)*, Croatia, 2022, 1–6. DOI: 10.23919/SpliTech55088.2022.9854268

[77] Mario Polino, Andrea Continella, Sebastiano Mariani, Stefano D'Alessio, Lorenzo Fontana, Fabio Gritti, and Stefano Zanero. 2017. Measuring and defeating anti-instrumentation-equipped malware. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10327 LNCS (2017), 73–96. DOI: https://doi.org/10.1007/978-3-319-60876-1_4

[78] Dmitry Ponomarev, Nael Abu-Ghazaleh, Caleb Donovick, Meltem Ozsoy, and Khaled N. Khasawneh. 2020. EnsembleHMD: Accurate hardware malware detectors with specialized ensemble classifiers. *IEEE Trans. Depend. Sec. Comput.* 17, 3 (2020), 620–633. DOI: https://doi.org/10.1109/TDSC.2018.2801858

[79] Mohan Anand Putrevu, Venkata Sai Charan Putrevu, and Sandeep Kumar Shukla. 2023. Early detection of ransomware activity based on hardware performance counters. In *Australasian Computer Science Week (ACSW'23)*. Association for Computing Machinery, New York, NY, 10–17. DOI: https://doi.org/10.1145/3579375.3579377

[80] Matilda Rhode, Pete Burnap, and Kevin Jones. 2018. Early-stage malware prediction using recurrent neural networks. *Comput. Secur.* 77 (2018), 578.

[81] Sherif Saad, William Briguglio, and Haytham Elmiligi. 2019. The curious case of machine learning in malware detection. (2019). arxiv:cs.CR/1905.07573

[82] Daniele Sgandurra, Luis Muñoz-González, Rabih Mohsen, and Emil C. Lupu. 2016. Automated Dynamic Analysis of Ransomware: Benefits, Limitations and Use for Detection. (2016). arxiv:cs.CR/1609.03020

[83] Amit Sharma, Brij B. Gupta, Awadhesh Kumar Singh, and V. K. Saraswat. 2022. Orchestration of APT malware evasive manoeuvers employed for eluding anti-virus and sandbox defense. *Comput. Secur.* 115 (2022). DOI: https://doi.org/10.1016/j.cose.2022.102627

[84] Shaila Sharmeen, Yahke Abukar Ahmed, Shamsul Huda, Bari Koçer, and Mohammad Mehedi Hassan. 2020. Avoiding future digital extortion through robust protection against ransomware threats using deep learning based adaptive approaches. *IEEE Access* 8 (2020), 24522–24534. DOI: https://doi.org/10.1109/ACCESS.2020.2970466

[85] K. Shaukat, S. Luo, V. Varadharajan, I. A. Hameed, and M. Xu. 2020. A survey on machine learning techniques for cyber security in the last decade. *IEEE Access* 8 (2020), 222310–222354. DOI: https://doi.org/10.1109/ACCESS.2020.3041951

[86] Jagsir Singh and Jaswinder Singh. 2020. Detection of malicious software by analyzing the behavioral artifacts using machine learning algorithms. *Inf. Softw. Technol.* 121 (2020). DOI: https://doi.org/10.1016/j.infsof.2020.106273

[87] Mark Stamp, Mamoun Alazab, and Andrii Shalaginov. 2021. *Malware Analysis Using Artificial Intelligence and Deep Learning*. Springer. DOI: https://doi.org/10.1007/978-3-030-62582-5

[88] Cong Truong Thanh and Ivan Zelinka. 2019. A survey on artificial intelligence in malware as next-generation threats. *Mendel* 25, 2 (2019). DOI: https://doi.org/10.13164/mendel.2019.2.027

[89] Chandra Thapa, Seung Ick Jang, Muhammad Ejaz Ahmed, Seyit Camtepe, Josef Pieprzyk, and Surya Nepal. 2022. Transformer-based language models for software vulnerability detection. In *38th Annual Computer Security Applications Conference (ACSAC'22)*. Association for Computing Machinery, New York, NY, 481–496. DOI: https://doi.org/10.1145/3564625.3567985

[90] Daniele Ucci, Leonardo Aniello, and Roberto Baldoni. 2019. Survey of machine learning techniques for malware analysis. *Comput. Secur.* 81 (2019), 123. DOI: https://doi.org/10.1016/j.cose.2018.11.001

[91] Anandhi V., Vinod P., Varun G. Menon, Abhijith Krishna E. R., Akshay Shilesh, Akshay Viswam, and Amin Shafiq. 2023. Malware detection using dynamic analysis. In *International Conference on Advances in Intelligent Computing and Applications (AICAPS'23)*. 1–6. DOI: https://doi.org/10.1109/AICAPS57044.2023.10074588

[92] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. 2020. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Comput. Netw.* 171 (2020). DOI: https://doi.org/10.1016/j.comnet.2020.107138

[93] Rakesh M. Verma, Victor Zeng, and Houtan Faridi. 2019. Data quality for security challenges: Case studies of phishing, malware and intrusion detection datasets. In *ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*. Association for Computing Machinery, New York, NY, 2605–2607. DOI: https://doi.org/10.1145/3319535.3363267

[94] Vinita Verma, Sunil K. Muttoo, and V. B. Singh. 2020. Multiclass malware classification via first- and second-order texture statistics. *Comput. Secur.* 97 (2020), 101895. DOI: https://doi.org/10.1016/j.cose.2020.101895

[95] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman. 2019. Robust intelligent malware detection using deep learning. *IEEE Access* 7 (2019), 46717–46738. DOI: https://doi.org/10.1109/ACCESS.2019.2906934

[96] VirusShare. 2023. Because Sharing Is Caring. Retrieved from https://virusshare.com/
[97] VIRUSTOTAL. 2023. Analyse Suspicious Files, Domains, IPs and URLs to Detect Malware and Other Breaches, Automatically Share Them with the Security Community. Retrieved from https://www.virustotal.com/gui/home/upload
[98] Wireshark. 2023. Wireshark. Retrieved from https://www.wireshark.org/
[99] Guoqing Xiao, Jingning Li, Yuedan Chen, and Kenli Li. 2020. MalFCS: An effective malware classification framework with automated feature extraction based on deep convolutional neural networks. *J. Parallel Distrib. Comput.* 141 (2020), 49–58. DOI: https://doi.org/10.1016/j.jpdc.2020.03.012
[100] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2021. BODMAS: An open dataset for learning based temporal analysis of PE malware. In *IEEE Security and Privacy Workshops (SPW'21)*. 78–84. DOI: https://doi.org/10.1109/SPW53761.2021.00020
[101] Limin Yang, Arridhana Ciptadi, Ihar Laziuk, Ali Ahmadzadeh, and Gang Wang. 2022. BODMAS Malware Dataset. Retrieved from https://whyisyoung.github.io/BODMAS/
[102] Yanfang Ye, Tao Li, Donald Adjeroh, and S. Sitharama Iyengar. 2017. A survey on malware detection using data mining techniques. *ACM Comput. Surv.* 50, 3 (2017), 1–40. DOI: https://doi.org/10.1145/3073559
[103] Suyeon Yoo, Sungjin Kim, Seungjae Kim, and Brent Byunghoon Kang. 2021. AI-HydRa: Advanced hybrid approach using random forest and deep learning for malware classification. *Inf. Sci.* 546 (2021), 420–435. DOI: https://doi.org/10.1016/j.ins.2020.08.082
[104] Ning Yu, Zachary Tuttle, Carl Jake Thurnau, and Emmanuel Mireku. 2020. AI-powered GUI attack and its defensive methods. In *ACM Southeast Conference*. 79–86. DOI: https://doi.org/10.1145/3374135.3385270
[105] Umme Zahoora, Asifullah Khan, Muttukrishnan Rajarajan, Saddam Hussain Khan, Muhammad Asam, and Tauseef Jamal. 2022. Ransomware detection using deep learning based unsupervised feature extraction and a cost sensitive Pareto Ensemble classifier. *Scient. Rep.* 12, 1 (2022), 15647. DOI: https://doi.org/10.1038/s41598-022-19443-7
[106] J. Zupan. 1994. Introduction to artificial neural network (ANN) methods: What they are and how to use them. *Acta Chim. Sloven.* 41, 3 (1994), 327.

Received 16 December 2022; revised 24 September 2023; accepted 21 December 2023