Characterizing and Codifying Malware Sophistication

Angelo Porcella
Montana State University
Bozeman, USA
angeloporcella@montana.edu

Zachary Wadhams
Montana State University
Bozeman, USA
zachary.wadhams@montana.edu

Clemente Izurieta
Montana State University
Bozeman, USA
clemente.izurieta@montana.edu

Jonathan Crussell
Sandia National Laboratories
Livermore, CA
jcrusse@sandia.gov

Ann Marie Reinhold
Montana State University
Bozeman, USA
reinhold@montana.edu

***Abstract*—"Sophisticated" is widely used to describe malware, yet it lacks a consistent definition within academic literature. While existing software quality and complexity metrics offer some insight into malware structure, they do not capture the broader adversarial and operational traits that contribute to real-world threat potential. This paper presents a systematization of existing approaches for assessing malware quality using static binary analysis. We define malware sophistication through a quality-focused lens by reinterpreting select characteristics from the ISO/IEC 25010 software quality standard, including reliability, maintainability, flexibility, and security, and evaluating their applicability to malware binaries. We identify which characteristics are both relevant to malware and measurable through static analysis, forming the basis for future frameworks that consistently assess malware sophistication when source code is unavailable or dynamic execution is infeasible**

***Index Terms*—malware, quality, sophistication, software, threat analysis, static analysis**

I. Introduction

The number of unique malware samples each year has been increasing at an extraordinary rate. In 2024, AV-Test.org reported an average of more than 450,000 new malicious programs registered daily [1]. As malware becomes easier to develop and distribute, it is increasingly important to establish a systematic method for assessing the sophistication of a sample using only its binary. Such a method would help analysts better understand the capabilities and intent of adversarial groups, enabling more effective threat prioritization and resource allocation. However, this is a formidable task. Sophistication reflects a combination of characteristics, such as code complexity, obfuscation, novelty, and functionality, many of which are difficult to measure without access to source code, particularly because malware is deliberately engineered to resist analysis.

This systematization of knowledge (SoK) investigates how existing software quality metrics and other statically measurable characteristics can be applied to assess malware sophistication. Despite the growing importance of this task, no standardized framework exists for evaluating sophistication based solely on binary analysis. Most current approaches to malware analysis focus on classification or behavior-based labeling rather than quantification [48], [49]. Notably, in 2001, the Malware Rating System proposed assessing threat levels based on behavioral criteria such as payload potential, proliferation potential, and hostility level [5]. However, more than two decades later, this model has seen little adoption. In response, we propose a broader set of criteria that are measurable from malware binaries, supported by examples of tools that assist in quantifying sophistication and, by extension, threat level.

II. Background

Efforts to evaluate malware sophistication remain fragmented, in part because no standardized model currently defines how to quantify the complexity, quality, or functionality of malicious software [50]. A consistent framework for evaluating sophistication would enable analysts to better interpret threat-actor intent and assess the potential impact of newly discovered samples. While traditional software quality metrics offer mature methodologies for assessing engineering quality, they do not translate directly to malware, particularly when analysis is limited to binaries alone. This section describes what constitutes malware sophistication, reviews how malware analysis is currently performed, examines how those methods inform notions of sophistication, and explains how software quality is measured using established standards.

A. Malware Sophistication

"Sophistication" is frequently used in threat reports and academic papers to describe advanced malware, yet the term lacks a consistent definition. Some authors use it to refer to technical complexity, while others emphasize evasion capabilities, operational stealth, or novelty. This ambiguity complicates comparative analysis and hinders the development of standardized assessment methods.

We propose that sophistication reflects the amount of specialized knowledge and effort expended in the development of malware. This definition distinguishes sophisticated samples from commodity malware, which may be functionally effective but requires little expertise to produce

TABLE I
DEFINITIONS RELATED TO SOFTWARE AND MALWARE SOPHISTICATION

Term	Definition
Malware	Software or firmware intended to perform an unauthorized process with an adverse impact on the confidentiality, integrity, or availability of a system.t [2].
Sophistication	The amount of specialized knowledge and effort expended in the development of a piece of software to satisfy the goals and objectives of the development team.
Quality	How well a software product conforms to its requirements and meets the needs of its users. From a software product perspective, quality refers to characteristics such as reliability, usability, performance, and security [29].
Novelty	Degree in which a malware sample is unique, purpose built, or unrecognizable from pattern matching antivirus software.
Obfuscation	The act of making software harder to discover or understand, either through manual or automated analysis, without altering the required functionality of the software [27].
Complexity	The amount and predictability of emergent behaviors arising from interactions both internally (i.e., components) and externally (i.e., the larger system). Complex software has components whose interactions evolve, with potentially unbounded states. [28].

or deploy. Under this framing, sophistication is not simply a function of what malware does, but how well it is engineered to achieve its objectives while resisting detection and analysis.

Three major barriers complicate the measurement of malware sophistication. First, source code is rarely available. In nearly all real-world cases, analysts must examine compiled binaries without access to original development artifacts [33]. This limitation restricts applicability of many established software quality metrics, such as maintainability indices or Halstead complexity, which rely on source-level constructs [30]. Compounding this, malware authors actively deploy obfuscation techniques ranging from simple packers and LLVM obfuscation passes to polymorphic engines that can prevent meaningful disassembly altogether [18]. These practices introduce the risk of a *deceptive simplicity paradox*: a technically advanced sample may appear unsophisticated to current analysis methods due to deliberate evasion of widely-used tools such as Ghidra, CAPA, or YARA.

Second, malware spans a wide spectrum of goals, capabilities, and operational environments. A trojan [46] may depend heavily on evasion while ransomware [47] emphasizes capability and impact over stealth. Metrics meaningful for one class may not apply others, requiring sophistication to be evaluated in context. Third, no unified framework exists for evaluating malware quality. While prior work has explored quality subcharacteristics using software engineering metrics [3], there is no established standard tailored to malicious software.

B. Current Methods of Malware Analysis

Researchers have explored various methods for assessing malware quality and complexity, often adapting traditional software engineering metrics to the domain of malicious code. These studies assume access to malware source code or high-fidelity decompilation and apply metrics such as LOC, cyclomatic complexity, and object-oriented design indicators to evaluate maintainability, modularity, or development effort [3], [8]. While these approaches can reveal trends in malware evolution, their reliance on source-level artifacts limits their applicability in operational contexts.

The MalSource study by Calleja et al. analyzed 456 malware samples with available source code spanning four decades, using metrics such as source lines of code, function points, and maintainability indices [3]. Their findings show a progression from small, individually developed malware toward larger, modular systems. Similarly, Mercaldo et al. applied software metrics to decompiled Android APKs to infer increasing sophistication over time [8]. Both studies demonstrate that traditional metrics can capture meaningful properties of malware development, but only when high-level representations are available. In practice, analysts almost always operate on compiled binaries, often in the presence of obfuscation or packing [33].

In the absence of source code, analysts rely primarily on static analysis—inspecting binaries without execution using techniques such as disassembly, control-flow graph extraction, and string inspection. While dynamic analysis in sandboxed environments can reveal runtime behaviors, it is resource-intensive, susceptible to evasion, and beyond the scope of this work. While some tools exist for binary-level analysis, there are far fewer in number and less mature than those designed for source code [44], [45].

C. Sophistication Proxies

In traditional software engineering, effort is often estimated using metrics like lines of code, function points, or development time. However, malware authors may intentionally manipulate these surface-level metrics. Dead code insertion adds non-functional instructions solely to bloat control-flow graphs, while binary padding appends junk data to exceed scanning limits or evade hash-based defenses. Both strategies are recognized in MITRE ATT&CK® as forms of defense evasion [38]. These challenges suggest that metrics of effort derived from binaries must be interpreted cautiously—apparent complexity may reflect defensive measures rather than underlying design.

Malware threat analysis aims to assess the risk, intent, and technical characteristics of malicious software. Although its primary goal is not to measure sophistication, many of the same characteristics, such as capability, platform compatibility, and structural complexity, are relevant to understanding

how advanced a sample may be. Existing threat assessment approaches provide useful insight into how observable features from binary analysis can inform evaluations of malware.

Maasberg, Ko, and Beebe propose a structured approach to malware threat assessment that emphasizes comparability across samples [7]. Their model draws on MITRE's Malware Attribute Enumeration and Characterization (MAEC) framework, which organizes threat information into standardized attributes. These include the objectives of the malware author, the capabilities of the malware, and its ability to operate on multiple operating systems. By framing these elements as components of threat level, the authors indirectly capture several traits that also signal sophistication, such as technical breadth and operational versatility. While their work is not focused on quality measurement directly, it demonstrates how structured attribute models can measure meaningful differences between malware samples using observable static features.

Dai et al. present a multi-stage model for assessing malware threats based on the fusion of static and dynamic analysis data [6]. Their static analysis component focuses on extracting structural features from the binary, including disassembled code, control flow graphs, and file metadata. These features are then combined with behavioral observations to produce a composite threat score. While the primary aim of this system is to classify malware based on risk, the inclusion of control-flow complexity and structural analysis highlights how low-level features can serve as indirect indicators of sophistication. The model underscores the potential of binary-derived metrics to inform broader evaluations, even in the absence of source code or execution traces.

Although existing threat assessment models focus primarily on risk, they demonstrate that static features such as control flow, compatibility, and functional scope can infer meaningful differences between samples, providing a foundation for more targeted models of malware quality.

D. Software Quality Measurement

The ISO/IEC 25010 standard defines a set of characteristics used to evaluate both software product quality and quality in use [4]. It is widely adopted in software engineering and forms the basis for several quality assessment models, including Hierarchical Software Quality Assurance (HSQA), implemented in tools like PIQUE [12] and incorporated into frameworks such as the Consortium for IT Software Quality (CISQ) reliability measures [11]. Static analysis plays a central role in these models by providing the inputs to HSQA models, such as PIQUE. By examining source code without executing it, static analysis tools can extract a wide range of software metrics, such as cyclomatic complexity, code duplication, coupling, and rule violations. Popular tools include commercial products like SonarQube, Coverity, and Fortify, as well as open-source tools such as PMD, Cppcheck and Bandit.

However, most static analysis tools rely on access to source code. In addition, many ISO/IEC 25010 characteristics and related metrics depend on constructs like class hierarchies, function definitions, or comments, which are unavailable in compiled binaries. Consequently, traditional quality models cannot be applied directly to malware in real-world scenarios, where source code is rarely accessible. This limitation highlights the need to reinterpret quality measurement for malicious software, focusing instead on features that can be extracted from binaries alone. Notably, ISO/IEC 25010 has also been applied in combination with complementary standards, demonstrating its flexibility as part of broader evaluation frameworks [24].

III. Definitions of Malware Sophistication

Before presenting our quality assessment framework, it is important to establish definitions for terms such as "sophistication", "quality", "complexity", and related concepts. These terms are often used inconsistently in the literature and are rarely defined in ways that facilitate their measurement. For this work, we adopt and adapt definitions that allow these concepts to be applied to malicious software, particularly when only binary artifacts are available. Table I summarizes the core terminology used throughout the paper. The definition of sophistication is central to this work: it emphasizes specialized knowledge and effort, distinguishing sophisticated malware from commodity variants.

We do not yet know how to directly measure the specialized knowledge and effort expended in malware development. However, we hypothesize that malware quality serves as an incomplete but partial measure of sophistication. The following section outlines how we propose to measure malware quality using inspiration from the ISO/IEC 25010 software quality standard.

IV. Application of ISO/IEC 25010 to Malware Quality Assessment

Currently, no unified set of malware quality metrics exist that are equivalent to those used in traditional software engineering. To support consistent assessment of malware sophistication, it is first beneficial to establish a standardized framework of software quality applicable to malicious programs. To explore this goal, we examine the ISO/IEC 25010 standard, which defines a set of quality characteristics typically used to assess conventional software systems [4]. Although it was originally designed for legitimate applications, these characteristics can provide a structured lens through which to evaluate malware, provided they are reinterpreted carefully. Table II summarizes each characteristic, with applicable characteristics listed first and excluded characteristics marked at the bottom. In the sections that follow, we first analyze the applicable characteristics in detail, then identify characteristics that are inapplicable or unmeasurable for malware.

A. Reliability

In malware, reliability translates to the ability of the sample to operate without faults, crashes, or unintended behavior during execution. Characteristics such as faultlessness and fault tolerance are particularly relevant when malware is designed to run persistently over extended periods.

From a binary analysis perspective, reliability may be inferred from structural indicators such as error-handling routines, redundant functionality, or fallback mechanisms. For

TABLE II
ISO/IEC 25010 QUALITY CHARACTERISTICS

Characteristic	Definition
Applicable to Malware	
Reliability	Maintenance of performance under specified conditions
Security	Protection of information and systems from unauthorized access
Maintainability	Ease of modification for defects, improvements, or adaptation
Flexibility	Adaptability to different requirements or environments
Excluded	
Functional Suitability	Degree to which functions meet specified requirements
Performance Efficiency	Resource utilization relative to intended function
Compatibility	Ability to operate within shared environments
Interaction Capability	Effectiveness of user interaction with software
Safety	Mitigation of risk to people, property, or environment

example, multiple startup persistence mechanisms (e.g., registry keys, scheduled tasks, service installation) suggest design that tolerates removal attempts. The Nexe backdoor creates persistence via a scheduled task and checks mutex values to prevent redundant instances, illustrating fault tolerance [53].

Static analysis methods may increasingly support assessments of reliability. The Consortium for Information & Software Quality (CISQ) Reliability Measures define 74 Common Weakness Enumerations (CWEs) intended to guide construction of reliable software in alignment with ISO/IEC 25010 [23]. Although this framework primarily targets source code, tools like CWE-Checker can detect a subset of reliability-related CWEs directly from compiled binaries.

B. Security

When applied to malware, security subcharacteristics are interpreted in reverse: sophisticated malware seeks to protect itself and its operators from discovery, analysis, and attribution. Confidentiality and integrity may be implemented through encryption, packing, or obfuscation to prevent reverse engineering [13]. Non-repudiation and accountability are typically subverted through removal of logging or anonymized communication. Resistance, the most relevant subcharacteristic, refers to a program's ability to withstand external attacks, including antivirus evasion, anti-debugging, and sandbox detection.

Concrete examples illustrate these security traits. Nexe disables Windows security instrumentation by patching AMSI and ETW, encrypts network traffic, and obfuscates its payload [53]. WikiLoader incorporates custom code and sandbox evasion to resist analysis [55]. Sophisticated malware often incorporates C2 protocols that resist disruption and complicate attribution [51], using encryption or domain generation algorithms to evade detection.

C. Maintainability

Code reuse is a particularly telling signal. Shared codebases or modular architectures suggest efforts to develop and maintain malware at scale. Techniques like fuzzy hashing and function similarity analysis can identify commonalities across samples, revealing modular design or iterative versioning. Lehman's law observes that software tends to become more complex over time [37], and in malware, rising complexity may indicate ongoing development [35], [36].

Although maintainability is traditionally measured using source code metrics, several indicators can be inferred from binaries. Cyclomatic complexity can be approximated from disassembled code [25]. However, some indicators may be distorted by obfuscation; file size may be artificially inflated through dead code insertion or padding [38]. Additionally, maintainability features can conflict with an attacker's goal of avoiding attribution, as code reuse creates linkages that aid threat actor profiling [52].

Several campaigns reinforce these observations. The KANDYKORN malware exhibits a five-stage infection chain comprising multiple distinct components [54], while WikiLoader exhibits modularity and is believed to be sold to multiple actors [55].

D. Flexibility

In malware, flexibility can be understood as the ability to operate across varied system configurations, respond to changes in the target environment, or maintain functionality across a wide range of infected hosts.

Adaptability may be reflected in support for multiple operating systems or processor architectures, or in the ability to receive dynamic configuration updates from C2 servers. From a binary analysis perspective, flexibility indicators include conditional logic tied to OS versioning, abstraction layers for file paths or system APIs, and multiple payload stubs within a single executable. Malware may also alter behavior based on execution environment; for example, checks for sandboxing or virtualization may trigger modified execution paths [26].

A Lazarus-linked cross-platform infostealer illustrates this trait well. It deploys payloads compatible with Windows, Linux, and macOS, and includes functionality for file theft, keystroke logging, and crypto mining [56]. Microsoft has similarly observed phishing campaigns that adapt delivery vectors and remote-access tools dynamically for each target [57].

E. Excluded Characteristics

Five ISO/IEC 25010 characteristics are either inapplicable to malware or not reliably measurable through static binary analysis.

Functional suitability requires determining whether software achieves its intended functionality correctly and completely. Without access to source code or specifications, this cannot be assessed for malware. Functional suitability also falls under "Quality in Use," which requires user interaction [20] that does not reflect real-world malware operation.

Performance efficiency can be partially inferred from static properties such as optimization level or avoidance of costly operations, but exact resource utilization requires

dynamic analysis. While efficient resource usage may indicate mature development, this characteristic is most directly assessed at runtime.

Compatibility involves operating within shared environments without interference. However, malware strategy varies significantly: stealthy malware prioritizes interoperability [21], while destructive malware like HermeticWiper [22] intentionally causes system crashes. This strategic variation complicates inclusion in a unified quality model.

Interaction capability is perspective-dependent, encompassing both attacker-side configuration and victim-side engagement. While some indicators (argument parsing, GUI components, localized strings) can be extracted from binaries, many malware samples lack interactive components entirely, making this characteristic difficult to generalize.

Safety attributes are largely inapplicable, as malware is inherently designed to operate without regard for user or system safety. Fail-safe behaviors in malware (such as terminating when a debugger is detected) serve stealth rather than safety and are more appropriately associated with reliability or security.

V. Conclusion and Future Work

This paper proposed a definition of sophistication as it pertains to malware. To address the lack of a formal malware quality framework, we adapted the ISO/IEC 25010 software quality standard by identifying characteristics applicable to malicious software. We focus on operationalizing Quality through reinterpretation of the ISO/IEC 25010 standard for static binary analysis. To advance malware quality studies, further work is needed to explore how existing software quality frameworks can be adapted. The long-term goal is supporting implementation and validation of a system that quantifies malware sophistication. A practical challenge lies in aggregating heterogeneous tool outputs under a unified scoring hierarchy. It may be possible to assign learned weights to sub-characteristics using empirical data, but this requires a labeled corpus reflecting malware sophistication degrees. No such public dataset currently exists; future efforts must address this gap through expert annotation, consensus labeling, or semi-supervised learning. A near-term priority is applying this framework to real malware samples to validate the proposed measurement process and assess whether the identified characteristics yield meaningful, consistent signals in practice. Once an operational model is established, statistical testing could confirm or challenge this assumption and further refine our understanding of adversarial software engineering.

Acknowledgements

This work was supported by DHS S&T at Sandia National Laboratories, a multi-mission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC (NTESS), a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration (DOE/NNSA) under contract DE-NA0003525. This written work is authored by an employee of NTESS. The employee, not NTESS, owns the right, title and interest in and to the written work and is responsible for its contents. Any subjective views or opinions that might be expressed in the written work do not necessarily represent the views of the U.S. Government. The publisher acknowledges that the U.S. Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this written work or allow others to do so, for U.S. Government purposes. The DOE will provide public access to results of federally sponsored research in accordance with the DOE Public Access Plan. The ChatGPT Large Language Model was used in this paper for spell-checking and minor grammatical enhancements.

References

[1] AV-TEST, "Malware Statistics & Trends Report — AV-TEST," Av-test.org, Apr. 25, 2025. https://www.av-test.org/en/statistics/malware

[2] "malware - Glossary — CSRC," csrc.nist.gov. https://csrc.nist.gov/glossary/term/malware

[3] A. Calleja, J. Tapiador, and J. Caballero, "The MalSource Dataset: Quantifying Complexity and Code Reuse in Malware Development," IEEE Transactions on Information Forensics and Security, vol. 14, no. 12, pp. 3175–3190, Dec. 2019, doi: https://doi.org/10.1109/TIFS.2018.2885512.

[4] ISO25000, "ISO/IEC 25010," iso25000.com, 2022. https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[5] R. Bagnall and G. French, "The Malware Rating System (MRS)TM RATING SYSTEM FOR SELF-REPLICATING MALICIOUS CODE -2.". http://dodccrp.org/events/6th_ICCRTS/Tracks/Papers/Track7/105_tr7.pdf

[6] C. Dai, J. Pang, X. Zhang, G. Liang, and H. Bai, "A Novel Information Fusion Model for Assessment of Malware Threat," International Journal of Security and Its Applications, vol. 10, no. 5, pp. 1–16, May 2016, doi: https://doi.org/10.14257/ijsia.2016.10.5.01.

[7] M. Maasberg, M. Ko, and N. Beebe, "Exploring a Systematic Approach to Malware Threat Assessment," Jan. 2016, doi: https://doi.org/10.1109/hicss.2016.682.

[8] F. Mercaldo, A. Di Sorbo, C. A. Visaggio, A. Cimitile, and F. Martinelli, "An exploratory study on the evolution of Android malware quality," Journal of Software: Evolution and Process, vol. 30, no. 11, p. e1978, Aug. 2018, doi: https://doi.org/10.1002/smr.1978.

[9] A. Lockett, "Assessing the Effectiveness of YARA Rules for Signature-Based Malware Detection and Classification," arXiv:2111.13910 [cs], Nov. 2021, Available: https://arxiv.org/abs/2111.13910

[10] J. Kornblum, "Identifying almost identical files using context triggered piecewise hashing," Digital Investigation, vol. 3, pp. 91–97, Sep. 2006, doi: https://doi.org/10.1016/j.diin.2006.06.015.

[11] "CWE - CWE-1305: CISQ Quality Measures (2020) (4.17)," Mitre.org, 2020. https://cwe.mitre.org/data/definitions/1305.html.

[12] C. Izurieta et al., "A Generalized approach to the operationalization of Software Quality Models," PeerJ Computer Science, vol. 10, p. e2357, Oct. 2024, doi: https://doi.org/10.7717/peerj-cs.2357.

[13] S. Kumar, J. Cox, B. Reaves, and L. Williams, "A Comparative Study of Software Secrets Reporting by Secret Detection Tools," Jul. 2023, Available: https://arxiv.org/pdf/2307.00714

[14] S. L. Garfinkel, "Digital media triage with bulk data analysis and bulk_extractor," Computers & Security, vol. 32, pp. 56–72, Feb. 2013, doi: https://doi.org/10.1016/j.cose.2012.09.011.

[15] C. Cohen, "The Pharos Static Analysis Framework for Software Assurance," Carnegie Mellon University Software Engineering Institute, 2018. Available: https://apps.dtic.mil/sti/tr/pdf/AD1084679.pdf

[16] F. Barr-Smith, X. Ugarte-Pedrero, M. Graziano, R. Spolaor, and I. Martinovic, "Survivalism: Systematic Analysis of Windows Malware Living-Off-The-Land," 2021 IEEE Symposium on Security and Privacy (SP), May 2021, doi: https://doi.org/10.1109/sp40001.2021.00047.

[17] D. Li, Q. Li, Y. (Fanny) Ye, and S. Xu, "Arms Race in Adversarial Malware Detection: A Survey," ACM Computing Surveys, vol. 55, no. 1, pp. 1–35, Jan. 2023, doi: https://doi.org/10.1145/3484491.

[18] K. Brezinski and K. Ferens, "Metamorphic Malware and Obfuscation: A Survey of Techniques, Variants, and Generation Kits," Security and Communication Networks, vol. 2023, p. e8227751, Sep. 2023, doi: https://doi.org/10.1155/2023/8227751.

[19] R. Murali and C Shunmuga Velayutham, "Adapting novelty towards generating antigens for antivirus systems," Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1254–1262, Jul. 2022, doi: https://doi.org/10.1145/3512290.3528693.

[20] P. Haindl, "Assessing and evaluating functional suitability of software," Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, Sep. 2018, doi: https://doi.org/10.1145/3238147.3241531.

[21] E. M. Rudd, A. Rozsa, M. Günther, and T. E. Boult, "A Survey of Stealth Malware: Attacks, Mitigation Measures, and Steps Toward Autonomous Open World Solutions," arXiv.org, 2016. https://arxiv.org/abs/1603.06028

[22] C. D. Fierro and J. Dwyer, "New destructive malware cyber attacks Ukraine," Ibm.com, Feb. 24, 2022.

[23] "CODE QUALITY RULES - CISQ," CISQ, Oct. 04, 2022. https://www.it-cisq.org/coding-rules/.

[24] E. Sheppard, Z. Wadhams, D. Arford, C. Izurieta, and A. Reinhold, "Wicked Problem, Parsimonious Solution: Securing Electric Vehicle Charging Station Software."

[25] S. K. S. Kumar, S. P. Kulyadi, P. Mohandas, M. J. S. Raman, and V. S. Vasan, "Computation of Cyclomatic Complexity and Detection of Malware Executable Files," 2021 13th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), pp. 1–5, Jul. 2021, doi: https://doi.org/10.1109/ecai52376.2021.9515044.

[26] S. Naval, Vijay Laxmi, M. S. Gaur, S. Raja, Muttukrishnan Rajarajan, and M. Conti, "Environment–Reactive Malware Behavior: Detection and Categorization," Lecture notes in computer science, pp. 167–182, Jan. 2015, doi: https://doi.org/10.1007/978-3-319-17016-9_11.

[27] "Malware Obfuscation: Techniques, Definition & Detection," Extrahop.com, 2025. https://www.extrahop.com/resources/attacks/malware-obfuscation

[28] "Complex Software — NIST," NIST, Jun. 12, 2023. https://www.nist.gov/glossary-term/20781.

[29] IEEE Computer Society, "What is Software Quality?," IEEE Computer Society. https://www.computer.org/resources/what-is-software-quality

[30] Yayi Zou, Yixiang Zhang, Guanghao Zhao, Yueming Wu, Shuhao Shen, Cai Fu. BinCoFer: Three-stage purification for effective C/C++ binary third-party library detection, Journal of Systems and Software, Volume 229, 2025, 112480, https://doi.org/10.1016/j.jss.2025.112480.

[31] "Awesome Executable Packing," GitHub, Dec. 05, 2022. https://github.com/packing-box/awesome-executable-packing

[32] Sudhakar and S. Kumar, "An emerging threat Fileless malware: a survey and research challenges," Cybersecurity, vol. 3, no. 1, Jan. 2020, doi: https://doi.org/10.1186/s42400-019-0043-x.

[33] J. Gennari, "Path Finding in Malicious Binaries: First in a Series," Carnegie Mellon University, Software Engineering Institute's Insights (blog). Carnegie Mellon's Software Engineering Institute, 10-Dec-2018 https://insights.sei.cmu.edu/blog/path-finding-in-malicious-binaries-first-in-a-series/.

[34] R. A. DeMillo, R. J. Lipton, and F. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," Computer, vol. 11, no. 4, pp. 34–41, Apr. 1978, doi: https://doi.org/10.1109/c-m.1978.218136.

[35] E. E. Ogheneovo, "On the Relationship between Software Complexity and Maintenance Costs," Journal of Computer and Communications, vol. 02, no. 14, pp. 1–16, 2014, doi: https://doi.org/10.4236/jcc.2014.214001.

[36] R. D. Banker, S. M. Datar, and D. Zweig, "Software complexity and maintainability," Proceedings of the tenth international conference on Information Systems - ICIS '89, 1989, doi: https://doi.org/10.1145/75034.75056.

[37] M. M. Lehman, J. F. Ramil, P. Wernick, D. E. Perry, and W. M. Turski, "Metrics and laws of software evolution-the nineties view," IEEE International Software Metrics Symposium, Nov. 1997, doi: https://doi.org/10.1109/metric.1997.637156.

[38] MITRE Corporation, "Junk Code, Binary Padding" MITRE ATT&CK Framework, https://attack.mitre.org/techniques/T1027/001/.

[39] A. Johnson, "The Analysis of Binary File Security Using a Hierarchical Quality Model," Thesis, Montana State University, 2021.

[40] P. Harrison, "Analyzing the security of C# source code using a hierarchical quality model," Thesis, Montana State University, 2022.

[41] B. Boles, E. O'Donoghue, A. Redempta Manzi Muneza, G. Perkins, C. Izurieta, and A. Marie Reinhold, "Deciphering Discrepancies: A Comparative Analysis of Docker Image Security," 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM), pp. 254–259, Oct. 2024, doi: https://doi.org/10.1109/scam63643.2024.00034.

[42] E. O'Donoghue, A. M. Reinhold, and C. Izurieta, "Assessing Security Risks of Software Supply Chains Using Software Bill of Materials," pp. 134–140, Mar. 2024, doi: https://doi.org/10.1109/saner-c62648.2024.00023.

[43] D. Rice, "AN EXTENSIBLE, HIERARCHICAL ARCHITECTURE FOR ANALYSIS OF SOFTWARE QUALITY ASSURANCE," Thesis, Montana State University, 2020.

[44] R. Ann Marie et al., "Surmounting Challenges in Aggregating Results from Static Analysis Tools," vol. 7, no. 1, p. 6, 2024, https://s3.wp.wsu.edu/uploads/sites/3267/2024/07/Challenges-in-Aggregating-Static-Analysis-Tool-Results.pdf

[45] A. M. Reinhold, T. Weber, C. Lemak, D. Reimanis, and C. Izurieta, "New Version, New Answer: Investigating Cybersecurity Static-Analysis Tool Findings," IEEE Xplore, Jul. 01, 2023. https://ieeexplore.ieee.org/abstract/document/10224930

[46] C. C. Editor, "trojan horse - Glossary — CSRC," csrc.nist.gov. https://csrc.nist.gov/glossary/term/trojan_horse

[47] NIST, "Ransomware," NIST, Sep. 27, 2021. https://www.nist.gov/itl/smallbusinesscyber/guidance-topic/ransomware

[48] P. Maniriho, A. N. Mahmood, and M. J. M. Chowdhury, "A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges," Future Generation Computer Systems, vol. 130, pp. 1–18, May 2022, doi: https://doi.org/10.1016/j.future.2021.11.030

[49] V. Vasani, A. K. Bairwa, S. Joshi, A. Pljonkin, M. Kaur, and M. Amoon, "Comprehensive Analysis of Advanced Techniques and Vital Tools for Detecting Malware Intrusion," Electronics, vol. 12, no. 20, p. 4299, Jan. 2023, doi: https://doi.org/10.3390/electronics12204299.

[50] B. De Sutter, S. Schrittwieser, B. Coppens, and P. Kochberger, "Evaluation Methodologies in Software Protection Research," ACM Computing Surveys, vol. 57, no. 4, pp. 1–41, Dec. 2024, doi: https://doi.org/10.1145/3702314.

[51] C. Rossow and C. Dietrich, "ProVeX: Detecting Botnets with Encrypted Command and Control Channels," in Proc. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA), LNCS vol.7967, Cham, Switzerland: Springer, 2013, pp.21–40. DOI:10.1007/978-3-642-39235-1_2.

[52] O. Mirzaei, R. Vasilenko, E. Kirda, L. Lu, and A. Kharraz, "SCRUTINIZER: Detecting Code Reuse in Malware via Decompilation and Machine Learning," in Proc. Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2021), LNCS 12756, Cham, Switzerland: Springer, 2021, pp.130–150.

[53] Cyble Research and Intelligence Labs, "Nexe Backdoor Unleashed: Patchwork APT Group's Sophisticated Evasion of Defenses," Sept. 26, 2024. https://cyble.com/blog/nexe-backdoor-unleashed-patchwork-apt-group-sophisticated-evasion-of-defenses

[54] C. Wilhoit, R. Ungureanu, S. Goodwin, and A. Pease, "Elastic catches DPRK passing out KANDYKORN in multi-stage macOS intrusion," Elastic Security Labs, Oct. 31, 2023. https://www.elastic.co/security-labs/elastic-catches-dprk-passing-out-kandykorn

[55] Proofpoint Threat Research, "Out of the Sandbox: WikiLoader Digs Sophisticated Evasion," Jul. 31, 2023. https://www.proofpoint.com/us/blog/threat-insight/out-of-the-sandbox-wikiloader-digs-sophisticated-evasion

[56] I. A. Baltariu, A. Anton-Aanei, and A. Bîzgă, "Lazarus Group Targets Organizations with Sophisticated LinkedIn Recruiting Scam," Bitdefender Labs, Feb. 5, 2025. https://www.bitdefender.com/en-us/blog/labs/lazarus-group-targets-organizations-with-sophisticated-linkedin-recruiting-scam

[57] Zophar, G. Khandelwal, A. Chaudhuri, M. Mesa, S. Patil, U. Oren, and K. Ramakrishnan, "Phish, Click, Breach: Hunting for a Sophisticated Cyber Attack," Microsoft Security Experts Blog, Oct. 15, 2024. https://techcommunity.microsoft.com/blog/microsoftsecurityexperts/phish-click-breach-hunting-for-a-sophisticated-cyber-attack

[58] S. H. H. Ding, B. C. M. Fung and P. Charland, "Asm2Vec: Boosting Static Representation Robustness for Binary Clone Search against Code Obfuscation and Compiler Optimization," 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 472-489, doi: 10.1109/SP.2019.00003.

[59] D. Kim, E. Kim, S. K. Cha, S. Son and Y. Kim, "Revisiting Binary Code Similarity Analysis Using Interpretable Feature Engineering and Lessons Learned," in IEEE Transactions on Software Engineering, vol. 49, no. 4, pp. 1661-1682, 1 April 2023, doi: 10.1109/TSE.2022.3187689.

[60] Plohmann, D. ., Blatt, M. ., & Enders, D. (2023). MCRIT: The MinHash-based Code Relationship & Investigation Toolkit. The Journal on Cybercrime and Digital Investigations, 8(1), 7-18. https://doi.org/10.18464/cybin.v8i1.45