Contents lists available at ScienceDirect

SoftwareX

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

Original software publication

MalGraphIQ: A tool for generating behavior representations of malware execution traces

Razvan Raducu, Ricardo J. Rodríguez [*], Pedro Álvarez

Engineering Research Institute of Aragón (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain

ARTICLE INFO

Keywords:
Malware analysis
Behavioral patterns
Visual analytics
Execution traces
Comparative malware analysis

ABSTRACT

Understanding and interpreting malware behavior remains an open challenge in the field of cybersecurity. The dynamic analysis of malware execution traces has emerged as a promising approach for discovering behavioral insights that allow the visual explanation of malware activity. MalGraphIQ is an open-source tool for the analysis and visualization of malware behavior. It is based on a structured and hierarchical taxonomy of API-based behavior patterns, which facilitates the interpretation of malware objectives, strategies, and low-level interactions with the attacked system. These interpretations support the comparative analysis of collections of suspicious programs, particularly across malware families and types, enhancing security research, malware triage, and the development of behavior-aware detection systems.

Code metadata

Current code version	v1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-25-00491
Permanent link to Reproducible Capsule	
Legal Code License	GNU General Public License (GPL) version 3.0
Code versioning system used	Git (GitHub)
Software code languages, tools, and services used	Python
Compilation requirements, operating environments & dependencies	Requirements: Python 3. Dependencies: graphviz 0.20.1, matplotlib 3.8.2, networkx 2.6.3, numpy 1.24.2, pandas 2.2.3, Requests 2.32.3
If available Link to developer documentation/manual	https://github.com/reverseame/MalGraphIQ/blob/main/README.md https://github.com/reverseame/MalGraphIQ/tree/main/doc/malgraphiq
Support email for questions	reverseame@unizar.es

1. Motivation and significance

Security analysts need to understand the behavior of malicious programs for improving the detection of attacks, developing stronger defense mechanisms, and enhancing the incident response management. The observation of this behavior involves identifying certain relevant insights during the execution of programs, for example: which critical system functions are invoked, what files, registry and OS-related configurations are modified, what communication flows are created, or what persistence mechanisms are employed. Malicious programs that exhibit similar execution behavior are typically grouped into malware families and/or types. These classifications play an important role in the programming of antiviruses and in the development of malware recognition systems [1].

Static and dynamic analysis are two different approaches for discovering relevant behavioral insights of malicious programs [2]. Static analysis techniques examine code without executing it, whereas dynamic analysis involves running the code, typically in a controlled environment, and observing its execution behavior. The main advantages of dynamic approaches are their ability to detect the actual runtime behavior of a program that static methods might miss (such as resource usage, system performance, or execution errors), offering a more comprehensive assessment of program execution and its interaction with the system. These advantages make dynamic analysis techniques particularly useful for analyzing modern malware, which is characterized by changing its behavior during execution to evade

* Corresponding author.
E-mail address: rjrodriguez@unizar.es (R.J. Rodríguez).

https://doi.org/10.1016/j.softx.2025.102407
Received 11 July 2025; Received in revised form 8 September 2025; Accepted 9 October 2025
Available online 21 October 2025

detection systems and hamper forensic analysis. This evasive ability complicates the identification of suspicious behaviors and the defense of target systems, keeping the continued need for tools able to analyze executions and to provide interpretable representations of such behaviors. Therefore, the discovery, interpretation, and description of these dynamic behaviors is an ongoing open challenge in the field of system security, particularly in *Microsoft Windows* OS systems, which have become the primary target of malware over the last decade [3].

Most of the dynamic analysis approaches are based on the identification of behavioral patterns within execution traces [4]. These patterns are typically sequences of system calls that involve access to sensitive resources. The knowledge extracted from this pattern-based analysis is usually interpreted to be translated into visual representations that highlight the critical actions involved in the malicious behavior, helping security analysts quickly contextualize what happened during the execution of the program [5]. Alternatively, the growing interest in the application of artificial intelligence within the cybersecurity field has led to the proliferation of visual representations aimed at optimizing malware classification and recognition methods [6,7]. However, the representations aimed toward artificial intelligence and hardly interpretable by a human analyst are beyond the scope of this work.

In the research literature, different tools have been proposed to visually represent malware behavior in *Windows* programs. Most of the examined tools rely on behavioral analysis of execution traces. These traces typically include detailed records of system activity during program execution, such as API calls, file operations, registry access, network connections, and thread or process manipulation. The main differences among these tools lie in the methods applied to extract knowledge from traces and in how this knowledge is interpreted and transformed into visual representations, as discussed in the following paragraphs. Unfortunately, the reviewed tools are not publicly available, which would otherwise facilitate a comparative evaluation based on performance, result interpretability or accuracy.

Most of the tools process execution traces to identify subsequences of API calls that could be associated with malicious behaviors. These subsequences are usually expressed as behavioral patterns represented through different formalisms that are subsequently applied to analyze traces using pattern-matching techniques. Tools such as *KAMAS* [8], *RanViz* [9] or *Eventpad* [10] use rules, n-grams, or regular expressions, respectively, to represent and identify behaviors of interest through pattern matching. Other tools, such as *MalView* [11] and the one introduced in [12], do not explicitly define patterns but behave in a similar way by directly processing traces to search for specific API invocations or sequences of invocations. Regarding the interpretation of results and corresponding visual representations, all these tools usually display the number of occurrences and frequency of individual API calls and structured API patterns, and the specific moments at which those patterns appear in the traces. Some proposals abstract these visualizations by grouping the results according to API categories rather than individual API calls [11,12].

Execution traces contain diverse information about the events and actions that occurred during program execution. In general, the mentioned tools analyze this information as a whole. However, some solutions focus on specific aspects of executions. For example, *MalView* exclusively analyzes the use of system resources, while *Eventpad* focuses on network traffic and communications. *MalwareVis* [13] is another tool specialized in analyzing network communications of suspicious programs. It is not based on pattern-matching techniques, and the visual results mainly display the number of connections, IP addresses and domains involved, and traffic statistics such as the number and size of packets. Finally, *SEEM* [14] restricts its analysis to detect similarities across executions by checking the use of system resources, DLL file imports, or accesses to external IP addresses, among others.

All the above tools provide a graphical interface to visualize the analysis results, with the exception of [12], which directly generates static images to be downloaded. Tools such as *KAMAS*, *Eventpad*, *MalView*, and *SEEM* provide interactive interfaces, allowing analysts to filter the results displayed and interact with the graphical representations.

In this work, we present MalGraphIQ, a tool for generating visual behavioral representations from the results of the analysis of *Windows* malware execution traces. This analysis is based on graph pattern-matching techniques. The tool uses a structured and hierarchical taxonomy of patterns based on sequences of API calls, called *Windows Behavior Catalog* (WBC) [15], to identify suspicious behaviors. This taxonomy enables the description of behaviors at different levels of abstraction, offering the possibility to interpret them in terms of high-level malware objectives or low-level system-call invocations. These interpretations are then represented using a variety of formats, including different types of graphs, radar chart, and bar plots. MalGraphIQ applies this taxonomy-driven analysis approach to collections of traces, revealing behavioral similarities and differences among them. It is particularly useful for the comparative analysis of malware families and types, as will be described.

MalGraphIQ provides several interesting contributions compared to prior tools. First, it organizes its behavioral patterns into a hierarchical taxonomy inspired by MITRE's Malware Behavior Catalog, a widely recognized knowledge base in the field of cybersecurity. Thanks to this taxonomy, the tool interprets and visually represents malware behavior at different abstraction levels, beyond API invocations or API sequence levels as in most existing tools. This second contribution allows security analysts to better understand the objectives of suspicious behaviors and the specific actions executed to achieve them. Third, the tool combines a variety of visual representations to enhance the behavioral knowledge extracted from the execution traces and provides access to all the intermediate data computed during the analysis process. Finally, MalGraphIQ is fully open source and publicly available [16], which facilitates reproducibility and community-driven extensions.

2. Software description

MalGraphIQ assists security analysts in understanding how malware operates with the final goal of protecting vulnerable systems against future similar attacks. This section describes the architectural design and the functionalities of the tool, as well as the configuration options for its execution.

2.1. Software architecture

Fig. 1 shows the architecture of MalGraphIQ, designed to generate visual representations of malware behavior from execution traces. The input traces contain detailed information about the execution behavior of malicious samples, for example, invoked system calls, accessed files, established network connections, registry changes, etc. These samples were previously executed in a controlled environment, such as in the MALVADA [17] framework. Once the input traces are processed, the output is a set of visual representations that illustrate the relative influence of certain behavior patterns in the executed samples. These patterns help security analysts understand the nature of threats and the actions involved in their operations.

MalGraphIQ is internally programmed as a pipeline to improve its maintainability and extensibility. It consists of three independent phases. The first, called *Behavioral Data Processing*, processes the input traces and transforms each one into a collection of directed graphs. Specifically, it generates a *behavior graph*, which represents the sequence of API or syscall invocations of the corresponding sample, and different *category* graphs, each representing the sequence of functions related to a particular Windows category, as defined in our *Windows API and Syscall Categories* [18]. This information is essential to build the

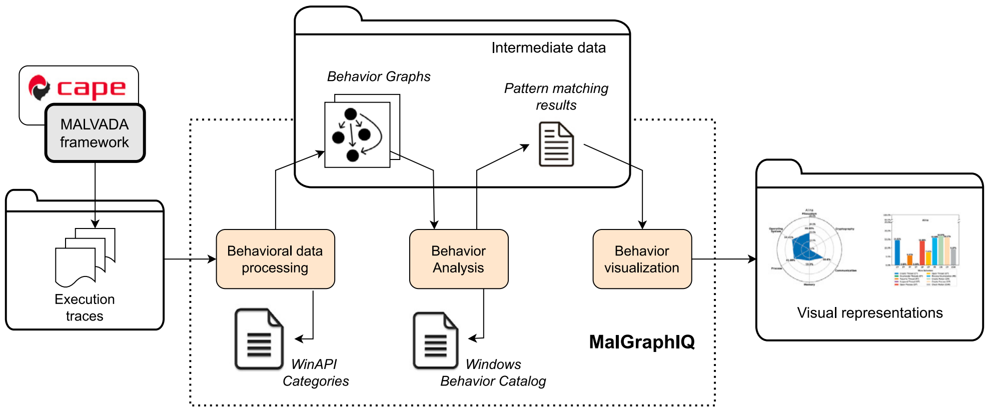

Fig. 1. Architecture of MalGraphIQ.

category graphs. For instance, `NtCreateFile` and `NtWriteFile` belong to “Files and I/O (Local file system)”, whereas `RegCreateKey` and `NtQueryKey` are examples of the “Registry” category. Our classification contains a total of 157 categories that comprise the vast majority of Windows API and system calls.

The second task, *Behavior Analysis*, aims at identifying behaviors in the generated graphs, counting the number of occurrences of each pattern from the *Windows Behavior Catalog (WBC)*, which is based on *MITRE's Malware Behavior Catalog (MBC) v3.2* [19]. The catalog is a taxonomy that comprises six malware objectives: File System, Communication, Memory, Process, Operating System, and Cryptography. This catalog is the central component of the analysis approach. It is structured in three hierarchical levels: malware objectives, behavior patterns, and methods, inspired by the *MBC*. Each pattern consists of a sequence of *Windows* API and syscalls and was implemented using at least one associated method. For instance, the sequence `NtCreateFile` → `NtReadFile` → `NtQueryAttributesFile` → `NtOpenFile` is an example of a behavior pattern related to the objective of querying the attributes of a given file. The task applies a loop-free Depth-First Search [20] backtracking algorithm to compare each of these patterns against the category graphs. The *NetworkX* library [21] is also used to make this matching more flexible, allowing intermediate nodes in the graphs that are not part of the behavior pattern. The result of counting occurrences of the patterns is stored in an intermediate JSON file.

The accuracy of this analysis task in detecting behaviors is inherently linked to the coverage of the WBC and the effectiveness of the underlying pattern-matching process. The tool will reliably detect those behaviors that are explicitly described in the WBC. Unfortunately, malware may exhibit behaviors that are not yet included in the taxonomy and, therefore, these will be wrongly ignored or regarded as absent. Regarding the application of malware evasion techniques, it is important to note that the tool relies on dynamic execution traces. Typical obfuscation techniques designed to hinder static reverse engineering (e.g., packing, code obfuscation, or encryption of binaries) do not prevent detection, since the malware must eventually invoke system calls to interact with the operating system. However, the analysis of polymorphic or metamorphic malware is limited to the behavior exhibited in each particular execution, which may result in the recognition of different patterns across multiple executions (although the same malicious objectives will be interpreted from them).

Finally, the task *Behavior Visualization* filters and processes the results of pattern matching before plotting them to illustrate the relative influence of each malware objective and the contribution of each behavior pattern to these objectives. Radar charts and bar plots are respectively used to represent these behavioral interpretations. Figs. 2 and 3 are examples of data visualizations produced in this task.

2.2. Software functionalities

MalGraphIQ offers functionality to configure its operational parameters and to process execution traces in order to generate different intermediate results and final visualizations. The configuration includes the option of executing the entire pipeline of tasks or a part of it, of defining the directories in which input data and results are stored, and setting different variables and thresholds involved in the pattern matching and data plotting processes. The processing functionality computes intermediate data that are probably to be of interest to analysts for performing various types of behavioral analysis: visual representations of behavior graphs based on system functions and their categories, the transition matrix of previous graphs, and statistical data about malware objectives and patterns from the WBC, which are specially interesting for alternative analysis approaches based on artificial intelligence techniques. In addition to these intermediate results, MalGraphIQ generates a set of visualizations representing the insights obtained from the pattern matching processing, as explained in Section 3. Finally, the tool also provides real-time monitoring of the status and results of each task in the process.

2.3. Software configuration

MalGraphIQ is open source and publicly available at [16]. This section details the steps necessary for a standard execution.

The main input of MalGraphIQ consists of analysis reports generated with `CAPEv2 Sandbox` [22] or execution traces created with MALVADA [17]. From now on, we will refer to either of them as *execution traces*. By default, MalGraphIQ operates in the `all` execution mode, in which all phases are executed sequentially as a pipeline, with the output of each phase serving as the input for the next one. In this mode, MalGraphIQ parses a set of execution traces and produces the final visual representations. Alternatively, users may execute each phase independently by specifying one of the following execution modes: `graphs`, `occurrences`, or `plots`. This approach is useful when there is a need to regenerate the output of a specific phase using custom parameters. For instance, refining the visual representations requires running only the `plots` phase with user-defined values for its associated parameters.

While MalGraphIQ offers a high degree of customization, it is designed to work with minimal user input, as most parameters are assigned default values. To execute the tool, run the main script (`malgraphiq.py`) specifying the execution mode, the directory containing the execution traces to be parsed, and the path to the WBC file, which is provided with the tool. Table 1 summarizes the most relevant parameters to the `all` execution mode. Additional documentation, along with all necessary files to run MalGraphIQ, is available in our GitHub [16]. Table 2 presents the structure of the repository.

Table 1
Main configuration parameters of MalGraphIQ.

Name	Type	Description
`json_dir`	string	Directory containing one or more execution traces.
`-w`	string	Path to `winapi_categories.json` file. If the file does not exist, the program will attempt to download it unless `-nd` is specified. Default: `Current working directory`.
`-c`	string	Path to the Windows Behavior Catalog (WBC) in JSON format.
`--plots_dir`	string	Output directory for the final visual representations. Created if it does not exist. Default: `./PLOTS/`.
`-q`	bool	Quiet mode, prints only error and critical messages. Default: `False`.
`-s`	bool	Silent mode, nothing is printed. Default: `False`.

Table 2
MalGraphIQ's repository [16] structure.

Folder	Contents
`src`	MalGraphIQ's source code.
`doc`	Documentation for developers rendered in HTML.
`test_reports`	Set of execution traces of 12 different malware families (100 e. t. per family), including `GCleaner` and `Remcos`, to test MalGraphIQ and check its execution. The folder also contains the produced visual representations.
`wbc`	WBC in JSON format.
`winapi_categories`	Our Windows API and Syscalls categorization in JSON format.

3. Illustrative examples

Malware is classified into *types* and *families*. A type is a set of programs that exhibit a common behavior, use similar infection mechanisms, and produce the same destructive effects. Examples of well-known malware types are viruses, worms, ransomware, trojans or loader. Each type is structured in a collection of families, which share certain characteristics and behaviors, but have particularities that differentiate them from each other [1,23]. For example, *Wannacry* or *Petya* are two different `ransomware` families. Unfortunately, there is no agreement on the definition and classification of existing types and families [24]. Nevertheless, the execution behavior of these programs is considered the key element that distinguishes them. These behaviors are typically described using textual representations, which are often ambiguous and insufficient to capture some of their more relevant details.

As shown in this example, we have selected execution traces of `GCleaner` and `Remcos` families. These families belong to two different types of malware: `loader` and `trojan`, respectively. 100 traces of each family were obtained from the WinMET dataset [25] and processed and interpreted using MalGraphIQ. The tool was executed in `all` mode with default parameters on a system equipped with an Intel i7-10700 @ 2.90 GHz CPU with 32 GB of DDR4 3200 MHz RAM. Processing `GCleaner` execution traces took approximately 103 min, while processing `Remcos` traces took around 35 min. Due to space constraints, this section presents results only for the `GCleaner` and `Remcos` families. However, MalGraphIQ's repository [16] also includes reports and visualizations for 10 additional malware families from the WinMET dataset. Specifically, all the generated visual representations are publicly accessible in the `test_reports` folder.

Both `GCleaner` and `Remcos` usually exhibit a certain common behavior. It includes typical malware actions like checking file names, creating files in order to replicate and disguise themselves, querying and modifying the registry with persistence purposes, and opening or enumerating processes looking for specific targets [26,27]. Nevertheless, in this example, the focus is on the behavior that differentiates them from each other, which is the reason they belong to different malware families and types. `GCleaner` is characterized by establishing Internet communications in an attempt to download other malware [26,28,29], whereas `Remcos` uses specific cryptographic routines that enable the malware to hide from antivirus [27] and to encrypt malicious traffic [30]. Fig. 2 shows the radar charts generated by MalGraphIQ that represent the behavior objectives of both malware families. The Communication and Cryptography objectives of each family are clearly different and they correspond with the expected behavior in each particular case. For the remaining objectives, both families show activity with higher or lower relative influence. Additionally, if a particular objective has no matches, this does not necessarily mean that the samples do not exhibit that behavior. Instead, it may indicate that the current version of the WBC does not yet consider such behavior patterns.

As was previously described, MalGraphIQ also helps understanding the relative influence of each behavior pattern in the corresponding objective. For the sake of space, we discuss here only the behavior of the two malware families with respect to the Process objective. Both families exhibited process actions as part of their common behavior, as shown in the radar charts of Fig. 2. Fig. 3 shows that behaviors like "Create Mutex" and "Check Mutex" are common to both families due to their mutant-related activity [31,32]. Both families match "Create Process" and "Process Enumeration" since they spawn different processes and enumerate existing ones in an attempt to profile the environment and discover possible targets [33,34]. Similarly, "Create Thread" and "Open Thread" are shared behaviors between both families as these are activities usually employed for evasive purposes like unpacking within a new thread or creating encryption-specific threads, just to mention some. Opening existing threads is a typical behavior malware uses to hijack or inspect threads attempting to inject and execute malicious code or evade debugging. The main difference between `GCleaner` and `Remcos` reflected in the figure is that the latter matches the "Resume Thread" and "Open Process" behaviors, which corresponds to its process injection capabilities [35]. `GCleaner`, on the other hand, focuses on downloading and directly executing different payloads rather than masquerading or injecting them. Finally, there are no matches with the WBC's behavior patterns corresponding to "Enumerate Threads" and "Suspend Thread" objectives.

While MalGraphIQ is compatible with execution traces generated with MALVADA and CAPEv2, in this work we limited the evaluation to traces from the WinMET dataset. MALVADA itself is designed to operate with CAPEv2 as its underlying sandbox, and, given its open source nature, it could be extended to support additional environments. However, such extensions are not yet implemented, and therefore a systematic comparison of runtime performance, resource usage, or detection coverage across different sandboxes is beyond the scope of this paper.

Finally, in terms of efficiency, the processing time of the trace set ranges from several minutes to several hours, depending on the complexity of the execution traces. Complexity is defined by the number of processes within a trace and the amount of activity performed by each process. Longer sequences of API and system calls result in higher complexity. The current pipeline-based design enables flexible execution deployments to meet specific user requirements, allowing the analysis of more complex datasets. Furthermore, the open-source nature of the tool allows customization of its internal mechanisms to support concurrency, parallelization, cloud deployment, or other computational paradigms as needed.

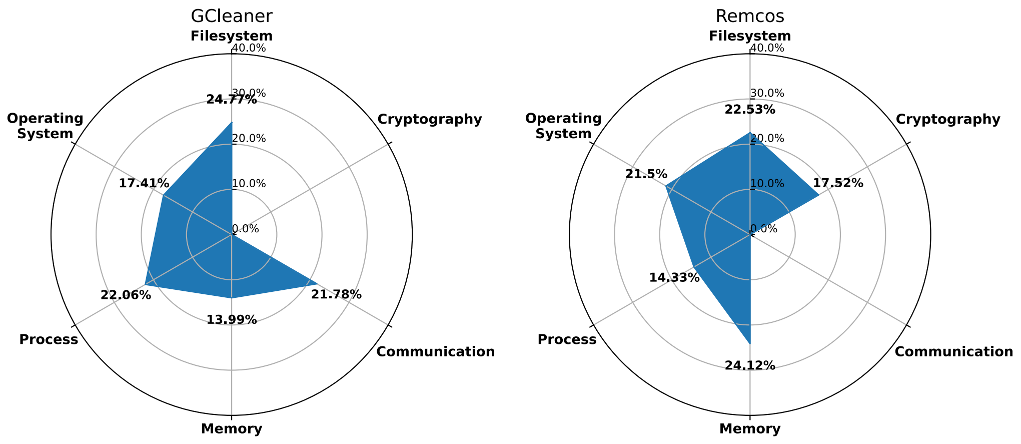

Fig. 2. Malware objectives visual representations.

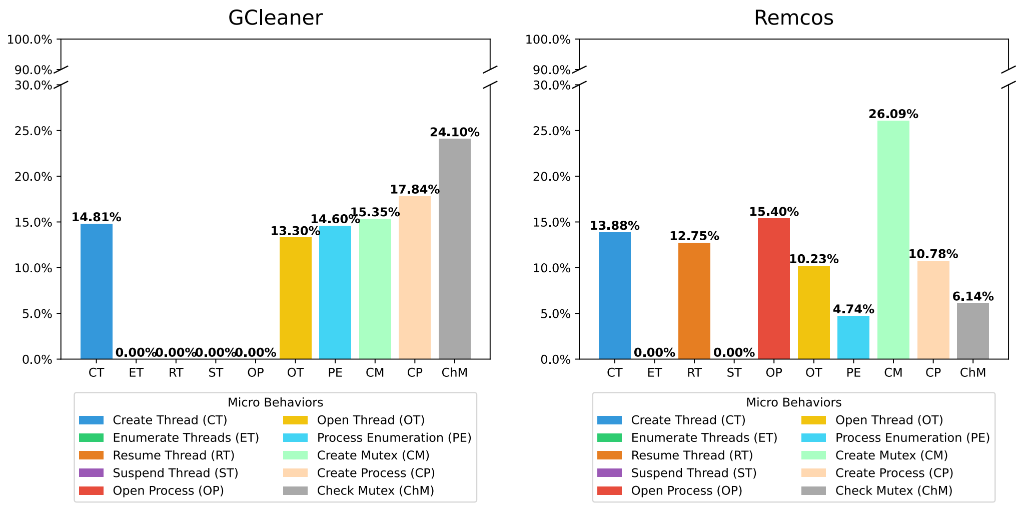

Fig. 3. Process behaviors visual representations.

4. Impact

The number of cyberattacks rises each year despite ongoing improvements in organizations' security prevention, detection and incident response systems. These attacks are becoming increasingly sophisticated, including advanced techniques able to evade defense mechanisms, to remain undetected during long-term periods, or to adapt their actions in real-time to the conditions of systems to be exploited. Security analysts need tools that help them understand the innovative nature and behavior of these attacks, relate the most recent attacks with other well-known, or identify zero-day exploits [36].

That understanding must be accompanied by novel mechanisms able to precisely describe how malware behaves. The conceptualization of all behavioral aspects involved in malware execution remains an open research challenge. Such conceptualizations are essential for characterizing both similarities and particularities among existing malware families and types. Advances in these behavioral definitions will be relevant to improve current malware classification and recognition systems, many of them based on artificial intelligence techniques, and therefore to enhance the security of systems and applications [37].

MalGraphIQ contributes to these research challenges by providing advances in the discovery of behavioral insights related to the execution of malicious programs, the description of those insights at different abstraction-levels, and the graphical conceptualization of behaviors that are shared by, or differing across, collections of malware samples. The MITRE-based taxonomy of patterns plays a central role in achieving these advances and serves as a foundational reference for discovering and defining malware behaviors. In contrast to other malware visualization tools, the taxonomy's hierarchical structure allows security analysts to explore relationships among suspicious behaviors and interpret them in terms of malware objectives, attack strategies, and malware family/type characteristics. This visual interpretability is particularly valuable for studying malware variants, designing behavior-aware defense mechanisms, and enhancing behavioral threat intelligence models.

While the tool currently emphasizes high-level summaries through radar and bar charts, its generation of intermediate data (e.g., graphs, transition matrices, and pattern occurrences) provides a foundation for more granular and interactive analyses in future extensions, and for alternative analysis techniques, including artificial intelligence-based models. Moreover, the taxonomy not only establishes a baseline of known patterns but it can be easily extended as new attack behaviors are reported.

From a technical point of view, MalGraphIQ offers a reusable and extensible analysis pipeline that can be adapted and integrated in different scenarios and use cases. It enables reproducible experiments, consistent evaluations across datasets, and parameter tuning for visualization and matching tasks. Besides, MalGraphIQ is integrated with *MALVADA* [17], a framework for generating large-scale datasets of malware execution traces. The integration of both applications results in a value chain that provides a comprehensible solution and facilitate the analysis of any set of executable programs, in contrast to other approaches found in the literature.

Finally, MalGraphIQ is open source and publicly available [16], aligning with the principles of open science and aiming to facilitate their widespread use by the research community.

5. Limitations

Despite its contributions and impact, MalGraphIQ has also some limitations, which are briefly discussed below.

MalGraphIQ depends on the Windows Behavior Catalog (WBC) to perform behavior matching, so its detection capability is restricted to the coverage of this taxonomy. Behaviors not yet represented in the WBC may remain undetected. As malware is continuously evolving, the WBC should be regularly updated to include new malicious behaviors. This requirement is inherent for any system designed for malware identification, recognition, or protection. Aligning these updates with the evolving MITRE's Malware Behavior Catalog and incorporating recent malware techniques would further improve MalGraphIQ's applicability. Given that both WBC and MalGraphIQ are open source, this updating process can be community-driven, ensuring that the tool adapts to the changing malware landscape and maintains long-term applicability.

As was described, the tool is currently integrated with MALVADA, which relies on CAPEv2 as its sandbox environment for malware execution. The integration of alternative sandboxes constitutes a promising future improvement, with the objective of enhancing the functionality of MalGraphIQ. This enhancement is facilitated by its modular design and open-source nature, and would primarily involve extending the initial task of input trace interpretation to support the processing of alternative trace formats. Additionally, it would compare different sandboxes in terms of computational performance and resource usage associated with execution trace generation, and select the best option for executing other available malware datasets.

The integration of MalGraphIQ into real-world analysis and prevention systems, including Security Operations Centers (SOCs), is an open challenge, as it implies transferring the research results into an adaptable final product. This process will involve making usability evaluations with analysts and case studies of interest and adapting or adding new visual representations that synthesize the knowledge required in each particular context. Nevertheless, currently, MalGraphIQ represents a commitment to open science, providing the community a flexible and open solution that can be easily adapted it according to their needs.

Another limitation we identified, not only in MalGraphIQ but also in all tools from the related work, is the absence of a comparative evaluation of result interpretability and reliability in the context of real malware analysis, as well as their practical usefulness for analysts. Ideally, such an evaluation would rely on a generic interpretability benchmark and the participation of analysts with varying levels of expertise, who could provide feedback on the usefulness of the results and visualizations. Although this is an interesting direction for future research, this type of comparison is unfortunately not feasible since no such benchmark exists and we do not have access to the tools from the related work. Given this difficulty, as future work, we will consider the design of an individual evaluation of MalGraphIQ, analyzing the analysts' usage context, conceptualizing their requirements, defining an evaluation methodology, and completing a significant number of evaluation tests.

6. Conclusions

In this paper, we presented MalGraphIQ, an open-source tool for behavioral analysis and visualization of malware execution traces. The tool parses execution traces, applies graph-based pattern-matching using the Windows Behavior Catalog (WBC) and generates different visual representations, enabling analysts to identify and interpret the behavioral strategies employed by malware samples in a structured and reproducible way.

Unlike existing tools, MalGraphIQ emphasizes semantic clarity and human interpretability. It produces high-level visual summaries that represent the presence and relative influence of behaviors across six major malware objectives and their underlying patterns. These visual representations support malware triage, comparison across families, and the generation of ground truth for future AI-based detection systems.

The tool is fully open source and designed to be extensible. It has been tested on real-world malware execution traces from the WinMET dataset, demonstrating its practical utility for research and analysis.

CRediT authorship contribution statement

Razvan Raducu: Writing – review & editing, Writing – original draft, Validation, Software, Resources, Methodology, Investigation, Data curation, Conceptualization. **Ricardo J. Rodríguez:** Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization. **Pedro Álvarez:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Data curation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported in part by TED2021-131115A-I00 (MIMFA), funded by MCIN/AEI, Spain/10.13039/501100011033, by the Recovery, Transformation and Resilience Plan funds, Spain, financed by the European Union (Next Generation), by the Spanish National Cybersecurity Institute (INCIBE) under *Proyecto Estratégico de Ciberseguridad – CIBERSEGURIDAD EINA UNIZAR* and under *Cátedra Internacional de Ciberseguridad UNIZAR*, and by the University, Industry and Innovation Department of the Aragonese Government, Spain under "Programa de Proyectos Estratégicos de Grupos de Investigación" (DisCo research group, ref. T21-23R). The work of Razvan Raducu was supported by the Government of Aragon through the Diputación General de Aragón (DGA) Predoctoral Grant, Spain, during 2021–2025.

References

[1] Sikorski M, Honig A. Practical malware analysis: the hands-on guide to dissecting malicious software. No Starch Press; 2012.

[2] Yong Wong M, Landen M, Antonakakis M, Blough DM, Redmiles EM, Ahamad M. An inside look into the practice of malware analysis. In: Proceedings of the 2021 ACM SIGSAC conference on computer and communications security. CCS '21, New York, NY, USA: Association for Computing Machinery; 2021, p. 3053–69.

[3] AV-TEST. AV-ATLAS - Malware & PUA. 2025, Online; https://portal.av-atlas.org/malware. [Accessed on 11 July 2025].

[4] Or-Meir O, Nissim N, Elovici Y, Rokach L. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. ACM Comput Surv 2019;52(5). http://dx.doi.org/10.1145/3329786.

[5] Wagner M, Fischer F, Luh R, Haberson A, Rind A, Keim DA, Aigner W. A Survey of Visualization Systems for Malware Analysis. In: Borgo R, Ganovelli F, Viola I, editors. Eurographics conference on visualization (euroVis) - sTARs. The Eurographics Association; 2015, http://dx.doi.org/10.2312/eurovisstar.20151114.

[6] Machado GR, Silva E, Goldschmidt RR. Adversarial machine learning in image classification: A survey toward the defender's perspective. ACM Comput Surv 2021;55(1). http://dx.doi.org/10.1145/3485133.

[7] Bensaoud A, Kalita J, Bensaoud M. A survey of malware detection using deep learning. Mach Learn Appl 2024;16:100546. http://dx.doi.org/10.1016/j.mlwa.2024.100546.

[8] Wagner M, Rind A, Thür N, Aigner W. A knowledge-assisted visual malware analysis system: Design, validation, and reflection of KAMAS. Comput Secur 2017;67:1–15. http://dx.doi.org/10.1016/j.cose.2017.02.003.

[9] Mokoma V, Singh A. RanViz: Ransomware visualization and classification based on time-series categorical representation of API calls. IEEE Access 2025;13:56237–54. http://dx.doi.org/10.1109/ACCESS.2025.3555163.

[10] Cappers BC, Meessen PN, Etalle S, van Wijk JJ. Eventpad: Rapid malware analysis and reverse engineering using visual analytics. In: 2018 IEEE symposium on visualization for cyber security (vizSec). 2018, p. 1–8. http://dx.doi.org/10.1109/VIZSEC.2018.8709230.

[11] Nguyen HN, Abri F, Pham V, Chatterjee M, Namin AS, Dang T. MalView: Interactive visual analytics for comprehending malware behavior. IEEE Access 2022;10:99909–30. http://dx.doi.org/10.1109/ACCESS.2022.3207782.

[12] Trinius P, Holz T, Göbel J, Freiling FC. Visual analysis of malware behavior using treemaps and thread graphs. In: 2009 6th international workshop on visualization for cyber security. 2009, p. 33–8. http://dx.doi.org/10.1109/VIZSEC.2009.5375540.

[13] Zhuo W, Nadjin Y. MalwareVis: entity-based visualization of malware network traces. In: Proceedings of the ninth international symposium on visualization for cyber security. VizSec '12, New York, NY, USA: Association for Computing Machinery; 2012, p. 41–7. http://dx.doi.org/10.1145/2379690.2379696.

[14] Gove R, Saxe J, Gold S, Long A, Bergamo G. SEEM: a scalable visualization for comparing multiple large sets of attributes for malware analysis. In: Proceedings of the eleventh workshop on visualization for cyber security. VizSec '14, New York, NY, USA: Association for Computing Machinery; 2014, p. 72–9. http://dx.doi.org/10.1145/2671491.2671496.

[15] Raducu R. behavior analysis for vulnerability and malware detection [Ph.D. thesis], Zaragoza, Spain: Universidad de Zaragoza; 2025.

[16] Raducu R, Rodríguez RJ, Álvarez P. MalGraphIQ. 2025, Online; https://github.com/reverseame/MalGraphIQ. [Accessed 14 May 2025].

[17] Raducu R, Villagrasa-Labrador A, Rodríguez RJ, Álvarez P. MALVADA: A framework for generating datasets of malware execution traces. SoftwareX 2025;30:102082. http://dx.doi.org/10.1016/j.softx.2025.102082.

[18] Raducu R, Rodríguez RJ, Álvarez P. Windows API and Syscalls categories. 2024, Online; https://github.com/reverseame/winapi-categories. [accessed 19 May 2025].

[19] MITRE. Malware Behavior Catalog. 2024, Online; https://github.com/MBCProject/mbc-markdown, [Accessed 19 May 2025].

[20] Kozen DC. The Design and Analysis of Algorithms . In: The design and analysis of algorithms. New York, NY: Springer New York; 1992, p. 19–24.

[21] Hagberg AA, Schult DA, Swart PJ. Exploring Network Structure, Dynamics, and Function using NetworkX. In: Varoquaux G, Vaught T, Millman J, editors. Proceedings of the 7th python in science conference. Pasadena, CA USA; 2008, p. 11–5.

[22] O'Reilly K, Brukhovetskyy A. CAPE: Malware Configuration And Payload Extraction. 2019, [Online;https://github.com/kevoreilly/CAPEv2. [Accessed on 15 May 2025].

[23] Cucci K. Evasive Malware: A Field Guide to Detecting, Analyzing, and Defeating Advanced Threats. No Starch Press; 2024.

[24] Hahn K. Malware family naming hell is our own fault. 2021, [Online; https://www.gdatasoftware.com/blog/malware-family-naming-hell. [Accessed 22 May 2025].

[25] Raducu R, Villagrasa-Labrador A, Rodríguez RJ, Álvarez P. WinMET Dataset. 2024, http://dx.doi.org/10.5281/zenodo.12647555, [Online; Accessed 22 May 2025].

[26] ANYRUN. GCleaner Stealer Malware Analysis, Overview by ANY.RUN. 2025, Online; https://any.run/malware-trends/gcleaner/. [Accessed 22 May 2025].

[27] ANYRUN. Remcos Malware Analysis, Overview by ANY.RUN. 2025, [Online; https://any.run/malware-trends/remcos/. [Accessed 22 May 2025].

[28] Elshinbary A. Deep Analysis of GCleaner. 2023, Online; https://n1ght-w0lf.github.io/malware%20analysis/gcleaner-loader. [Accessed 22 May 2025].

[29] Zigel H, Kupreev O, Ushkov A. NullMixer: oodles of Trojans in a single dropper. 2022, Online; https://securelist.com/nullmixer-oodles-of-trojans-in-a-single-dropper/107498/. [Accessed 22 May 2025].

[30] Check Point Software. Remcos Malware. 2025, Online; https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-malware/remcos-malware. [Accessed 23 May 2025].

[31] Sandbox J. Automated Malware Analysis Report. 2025, [Online; https://www.joesandbox.com/analysis/1604030/0/html. [Accessed 23 May 2025].

[32] Zahravi A. Analysis: New Remcos RAT Arrives Via Phishing Email. 2019, Online; https://www.trendmicro.com/en_us/research/19/h/analysis-new-remcos-rat-arrives-via-phishing-email.html. [Accessed 23 May 2025].

[33] SonicWall Capture Labs Threat Research Team. GCleaner is Packed and Ready to Go. 2025, Online; https://www.sonicwall.com/blog/gcleaner-is-packed-and-ready-to-go. [Accessed 23 May 2025].

[34] Hackers Arise. Malware Analysis: Process Injection in the REMCOS RAT. 2025, Online; https://hackers-arise.com/malware-analysis-process-injection-in-the-remcos-rat/. [Accessed 23 May 2025].

[35] François C, Bousseaden S. Dissecting REMCOS RAT: An in- depth analysis of a widespread 2024 malware, Part One. 2024, Online; https://www.elastic.co/security-labs/dissecting-remcos-rat-part-one. [Accessed 23 May 2025].

[36] Gandotra E, Bansal D, Sofat S. Zero-day malware detection. In: 2016 sixth international symposium on embedded computing and system design. ISED, 2016, p. 171–5. http://dx.doi.org/10.1109/ISED.2016.7977076.

[37] Botacin M, Ceschin F, Sun R, Oliveira D, Grégio A. Challenges and pitfalls in malware research. Comput Secur 2021;106:102287.