MALVADA: A Framework for Generating Datasets of Malware Execution Traces

Razvan Raducu[a], Alain Villagrasa-Labrador[a], Ricardo J. Rodríguez[a,*], Pedro Álvarez[a]

[a]*Engineering Research Institute of Aragon (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain*

Abstract

Malware attacks have been growing steadily in recent years, making more sophisticated detection methods necessary. These approaches typically rely on analyzing the behavior of malicious applications, for example by examining execution traces that capture their runtime behavior. However, many existing execution trace datasets are simplified, often resulting in the omission of relevant contextual information, which is essential to capture the full scope of a malware sample's behavior. This paper introduces MALVADA, a flexible framework designed to generate extensive datasets of execution traces from Windows malware. These traces provide detailed insights into program behaviors and help malware analysts to classify a malware sample. MALVADA facilitates the creation of large datasets with minimal user effort, as demonstrated by the WinMET dataset, which includes execution traces from approximately 10,000 Windows malware samples.

Keywords: Dataset generation, malware behavior, execution traces, malware classification

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v1.1
C2	Permanent link to code/repository used for this code version	`https://github.com/reverseame/MALVADA`
C3	Permanent link to Reproducible Capsule	–
C4	Legal Code License	GPL v3.0
C5	Code versioning system used	Git (GitHub)
C6	Software code languages, tools, and services used	Python
C7	Compilation requirements, operating environments & dependencies	Requirements: Python 3, `sed`, AVClass [1]. Dependencies: matplotlib 3.8.2, pandas 2.2.2, rich 13.7.1, seaborn 0.13.2, ujson 5.9.0
C8	If available Link to developer documentation/manual	`https://github.com/reverseame/MALVADA/blob/main/README.md` `https://github.com/reverseame/MALVADA/tree/main/doc/malvada_workflow`
C9	Support email for questions	reverseame@unizar.es

Table 1: Code metadata (mandatory)

Metadata

1. Motivation and Significance

The rise in cyberattacks involving malware [2] has driven the need for improved detection methods. Several approaches have been proposed [3], in-

*Corresponding author
Email address: `rjrodriguez@unizar.es` (Ricardo J. Rodríguez)

cluding artificial intelligence techniques [4, 5, 6], similarity algorithms [7, 8], and execution signatures [9], among others. These techniques typically rely on malware execution traces that have been previously captured, often in production systems (when a real attack occurs) or in sandbox environments. Malware execution traces are necessary to effectively train and test these methods, as execution traces reveal the actual behavior of the malware at runtime. However, most available execution trace datasets are simplified or optimized to be more efficient for techniques such as artificial intelligence. This simplification, in turn, frequently removes important contextual information, such as API or system call parameters and return values. Producing large datasets of malware execution traces remains a significant challenge due to the requirement for specialized tools, high resource costs, the risk of errors, and the need for user intervention during malware execution.

Many malware detection proposals have focused on Windows systems due to their widespread use [10] and high appeal to attackers [11]. Behavior-based detection for Windows often involves analyzing *execution traces*, which are sequences of system calls or API functions invoked by a program. By examining these sequences, patterns relevant to determining the malware family or type can be identified.

To our knowledge, there are very few publicly accessible datasets on Windows malware execution traces [12, 13, 14, 15]. These datasets typically consist only of sequences of API names or numerical identifiers, which provide a basic representation of execution. They also tend to include a limited number of malware families and types, which are sometimes grouped together and treated as indistinguishable. Furthermore, this simplified representation often omits critical contextual information such as API parameters, results, processes created, synchronization objects, resources accessed, and communications established. This lack of detail hinders a comprehensive understanding of execution behavior, which is particularly important in malware analysis [16].

There are few tools for automatically generating datasets of malware executions. In [17], the authors introduced a tool that gathers information on malicious behavior from various security reports and analysis sources without executing the programs themselves, resulting in datasets based on secondary information. In contrast, the work in [18], more aligned with the approach discussed here, involves executing malware in a controlled environment to generate Windows system call traces. Specifically, the virtualization-based environment integrates tools to gather information about the system calls invoked and the files used during the execution of malware samples. This information is then stored into a relational database so that it can be translated to different output formats. Additionally, each sample is classified using two labels: one indicating the malware category and another one specifying the malware family. Unfortunately, these category labels are too generic and have a limited semantic meaning, such as "Virus", "Trojan", "DangerousObject", or "Packed". This tool was used to generate a public dataset, called `AWSCTD`, which consists only of the anonymized sequence of system calls (the name of system calls have been translated to numerical identifiers and their parameters/results removed). In contrast, MALVADA is based on CAPE's reports, which provide a richer description of the actions involved in the execution of the samples (including information about processes, network communications, synchronization, or the usage of registry, for instance). Besides, modern versions of two labeling algorithms have been used to determine the malware family of each sample, increasing the significance and precision of their classification labels.

In this work, we present MALVADA, a framework designed to generate Windows program execution trace datasets that relies on CAPE Sandbox [19] to execute programs and produce detailed reports. MALVADA filters and processes these reports into traces that include contextual information. Furthermore, these traces are also enriched with metadata, such as the likely malware family the program belongs to, providing a comprehensive dataset. The end result is a collection of traces in JSON, suitable for various malware analysis applications. Our framework allows users to create custom datasets or extend existing ones. In this paper, we also publish the first version of a dataset generated by MALVADA, called *Windows Malware Execution Traces* (WinMET), which comprises approximately 10,000 malware execution traces.

2. Software Description

As shown in Figure 1, we used an enhanced version of CAPE Sandbox to analyze samples and generate execution reports. Modifications to CAPE included increasing the number of API calls intercepted, focusing on those linked to suspicious behaviors such as network communications, file/registry accesses, and memory usage. Additionally, we developed `CAPE Hook Generator`, a tool that facilitates integrating new hooks into CAPE. This tool is publicly available in our GitHub [20].

We use Kernel-based Virtual Machine (KVM) technology to deploy virtual machines (VMs) to run malware samples with CAPE. Each VM generates a report with key events and artifacts from the dynamic analysis, which is then processed by MALVADA. For medium-sized sample collections, we recommend a multi-VM setup. In our setup, we used four VMs on an Ubuntu 22 host, each running Windows 10 x64. Using this setup, we analyzed over

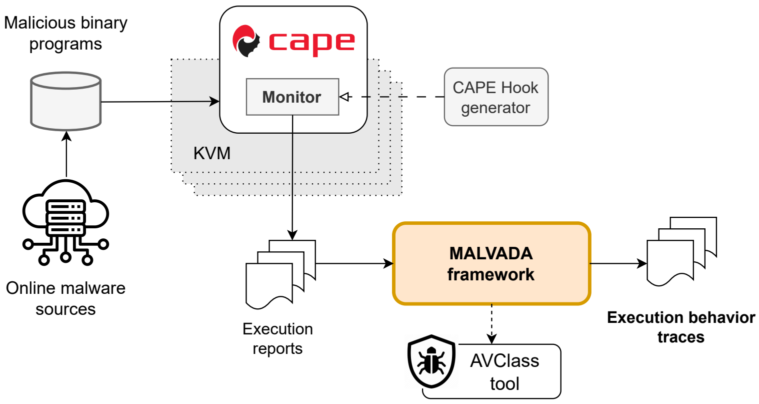

Figure 1: Contextual overview of the MALVADA framework

20,000 samples. We discarded incorrect executions caused by errors, crashes, or connectivity issues with the sandbox environment. The remaining reports were then processed using MALVADA, resulting in the creation of the first version of WinMET.

MALVADA processes CAPE reports to extract key data for understanding malware execution behavior. It generates detailed execution traces for each report, including the process tree, API call sequences, contextual information, accessed operating system resources, and mutex synchronizations, among others. Each report contains VirusTotal labels [21]. These labels are used to assign each trace to a malware family by applying two labeling algorithms: CAPE's algorithm and AVClass [1].

2.1. Software Architecture

Figure 2 shows the architecture of MALVADA, designed as a modular pipeline for processing and generating datasets from input reports. This modular design improves its maintainability, extensibility, and adaptability. Each task in the pipeline can be executed independently, allowing users to customize phases or configure different implementations. The tasks and control algo-

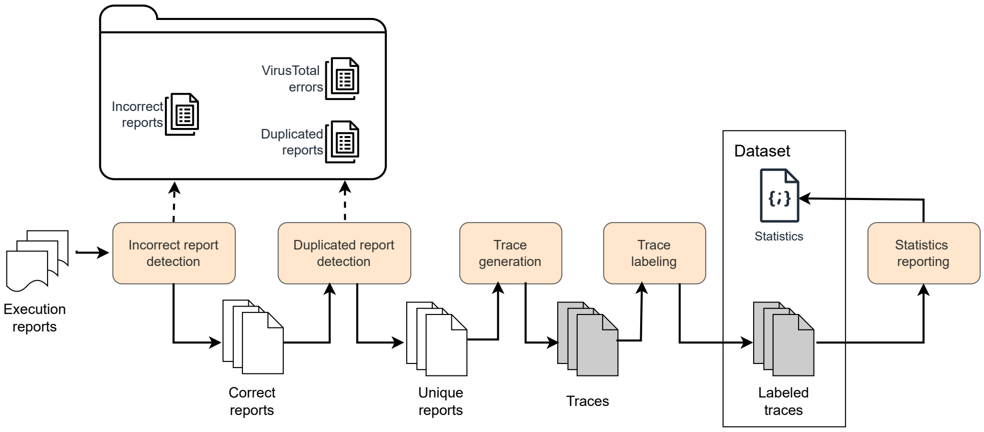

Figure 2: Internal architecture of MALVADA

rithm are implemented in Python, and data input and output are handled in JSON format.

MALVADA automatically processes the CAPE analysis reports given as input in five steps to create a dataset of execution traces. First, it filters out reports of incomplete or failed executions (*incorrect report detection*) and removes duplicates based on sample ID hashes (*duplicate report detection*), retaining only one report per sample. It then transforms the remaining reports into execution traces by extracting and structuring relevant data about the execution behavior and context while anonymizing sensitive information (*trace generation*). The structure of these traces is described in Section 3. Each trace is then tagged with information about the malware family using results from antivirus engines and tools such as AVClass [1] to standardize classification (*trace labeling*). Finally, the system generates statistics about the processing of the reports and the composition of the final dataset (*statistics reporting*).

2.2. Software Functionalities

The framework offers several key functionalities. It includes a configuration functionality to set operational parameters, such as directories for output results, criteria for report duplication, and thresholds for sample classification. The report filtering functionality also records reasons for exclusion of incomplete or erroneous reports and duplicates for the user to review. Trace generation processes CAPE analysis reports into detailed execution traces in JSON format, enriched with malware behavioral characteristics. The framework also incorporates malware family detection using CAPE and AVClass [1] to provide standardized family labels. Traces, labels, and a statistical description of the processing are packaged by MALVADA into a comprehensive dataset. In addition, it provides real-time monitoring of the status and results of each task in the process.

2.3. Software Configuration

The source code for MALVADA is publicly available [22]. This section provides a detailed guide on the necessary steps to execute MALVADA.

MALVADA processes reports generated by `CAPEv2 Sandbox` [19]. Therefore, installing CAPE, the VMs for malware analysis, and other dependencies is the first recommended step [23]. Additionally, to expand the API calls that CAPE hooks and thus improve the contextual information obtained from a program execution, it is necessary to modify the original CAPE monitor (`capemon`). This involves editing and recompiling the `capemon` source code, written in C [24]. Our tool `CAPE Hook Generator` [20] simplifies this by generating hook code skeletons, which can be then integrated into the `capemon` source code. Once these steps are completed, the environment for executing malware samples and generating reports is ready.

Name	Type	Description
`json_dir`	string	Directory containing one or more execution reports.
`-w`	int	Number of workers. Default: `10`.
`-vt`	int	Threshold for VirusTotal positives to consider a sample malicious. Default: `10`.
`-a`	string	Replace the terms in the file provided with "[REDACTED]". Default: `terms_to_anonymize.txt`.
`-s`	bool	Silent mode. Default: `False`.

Table 2: Configuration parameters of MALVADA

MALVADA works with minimal user intervention. To run it, simply execute the main script (`malvada.py`) and specify the directory containing the reports to be analyzed. Users can also customize certain parameters to fine-tune the tool's behavior. Table 2 lists these parameters, along with their type and a brief description. More details are available in our GitHub [22], which contains all the material necessary to execute and test the tool. Table 3 summarizes the structure of the repository.

3. Illustrative Examples

This section aims to explain the results produced by executing MALVADA. We first describe the structure of an execution trace generated by the framework, and then we provide a detailed example of a dataset created by MALVADA.

3.1. Structure of an Execution Trace

The JSON document for a trace comprises several fields that collectively detail the execution behavior of the analyzed sample. Due to space limitations, Listing 1 displays only the most relevant fields.

Folder	Contents
`src`	MALVADA's source code.
`doc`	Documentation for developers rendered in HTML.
`test_reports`	A set of 200 execution reports generated with CAPE. The purpose of these test reports is to check the execution of MALVADA. The folder also contains the expected results after the execution, in order to compare if the tool behaves as expected.
`capemon`	The compiled version of capemon we used.
`cape-hook-generator`	Version 1.0 of CAPE Hook Generator [20].
`WinMET`	Information about WinMET dataset.

Table 3: MALVADA's repository [22] structure.

The key fields include: sample identification using cryptographic and similarity hashes (line 4); details about the binary file type (such as whether it is a Portable Executable [25] with specific attributes such as imports, exports, and sections; lines 15–18); a process tree that records all processes initiated by the sample (line 47); a sequence of API calls made during execution, including their arguments, return values, and categories (lines 33–46); malware classification labels, as determined by CAPE (line 1) and AVClass [1] (line 53) based on the VirusTotal results (line 24); and a summary of the OS resources accessed by the sample, such as files, registry keys, mutexes, and services (line 48).

The API call sequence is the most crucial element in a trace. Each entry in the `"processes"` array (line 34) represents a process started during execution, identified by a `"process_id"` (line 35). The API calls made by each process are stored in the `"calls"` entry (line 37). For each API call, it includes the name of the API (`"api"`; line 39), the category of the call (`"category"`; line 38), the return value (`"return"`; line 30), and the arguments (`"arguments"` array; lines 41–44), among other data.

Listing 1: Main structure of an enhanced report


```
{ "detections": [{...}],
  "target": {
    "file": {
      "md5": "...", "sha256": "...", "ssdeep": "...", ... // Additional hashes
      "imports": {
        "KERNEL32": {
          "dll": "KERNEL32.DLL",
          "imports": [{
            "address": "0x68d13c",
            "name": "LoadLibraryA"
            }, ..., // Additional entries per imported function
          ]
        }, ..., // Additional entries per each imported dll
      },
      "pe":{
        "resources": [{...}],
        ... // Additional fields in the "pe" entry
      },
      "strings": [...],
      "virustotal":{
        "scan_id": "...",
        "positives": 13,
        "total": 73,
        "results":[{
          "vendor": "...",
          "sig": "..."
          }, ..., // Additional entries per vendor
        ], ..., // Additional fields in the "virustotal" entry
      }, ... // Additional fields in the "file" entry
    }, ... // Additional fields in the "target" entry
  },
  "dropped": [{...}],
  "behavior": {
    "processes":[{
      "process_id": 1337,
      "parent_id": 31337,
      "calls":[{
        "category": "filesystem",
        "api": "NtOpenFile",
        "return": "0x00000000",
        "arguments":[{
          "name": "FileHandle",
          "value": "0xDEADBEEF"
        }, ...,] // Additional entries per each argument
      }, ...,] // Additional entries per each API or syscall
    }, ...,], // Additional entries per each process
    "processtree": [ ... ],
    "summary": {
      "files": [ ... ], "read_files": [ ... ], "write_files": [ ... ], "delete_files": [ ... ],  "keys": [ ... ], "read_keys": [ ... ], "write_keys": [ ... ], "delete_keys": [ ... ], "executed_commands": [ ... ], "mutexes": [ ... ],
      ..., // Additional fields in the "summary" entry
    },..., // Additional fields in the "behavior" entry
  },
  "avclass_detection": "...",
  ..., // Additional entries in the report
  }
}
```

3.2. Example of dataset

Using MALVADA, we created the WinMET dataset, which currently contains approximately 10,000 execution traces. The top five malware families represented in the dataset, according to AVClass [1], are `Reline` (22.1%), `Disabler` (7.4%), `Amadey` (5.8%), `Agenttesla` (4.8%), and `Taskun` (3.8%). According to CAPE, the top five families are `Redline` (12.4%), `Agenttesla` (10.2 %), `Crifi` (6.3%), `Amadey` (6.13%), and `Smokeloader` (5.4%). Both labeling approaches are based on labels provided by vendors from VirusTotal. On average, there are 53 labels per report. Additionally, 7% of the samples are labeled as "(n/a)" by AVClass, compared to 26% by the CAPE labeling algorithm. This suggests that AVClass is able to assign a label in most cases. The "(n/a)" label indicates that the respective algorithm could not determine a decision on the malware family.

The WinMET dataset is publicly available at [26], and additional details are provided in our GitHub repository [22].

Creating a dataset that includes all malware families is nearly impossible due to the sheer number of families. However, MALVADA allows for continuous updates and improvements to the dataset by allowing new samples from other families to be analyzed and included, either by the original developers or by third parties. This flexibility ensures that the dataset can be expanded and refined over time.

4. Impact

Cyberattacks are growing exponentially and becoming increasingly sophisticated, posing a significant threat to users and organizations. A major challenge in cybersecurity is developing tools that can efficiently detect and mitigate these attacks to minimize damage. These tools often rely on learning from past data, making the availability of specialized, high-quality datasets essential. The rise of artificial intelligence as a detection method has further amplified the need for diverse, large-scale data sets, particularly since models like deep learning require extensive and varied data to perform accurately and reliably.

MALVADA addresses the scarcity of publicly available malware execution trace datasets. While traditional efforts have focused on collecting malware samples (e.g., VirusShare, VX-underground, Malware Bazaar, and MalShare malware repositories, among others), our framework enables the creation of datasets by processing the reports generated from sandbox environments such as CAPE.

MALVADA offers several advantages for researchers too. Its modular design allows tasks in the process chain to be easily modified and extended, enabling new functionalities and improved reporting processing. The framework is easy to use and requires minimal intervention, and no specialized technical knowledge, making it accessible to users from all backgrounds. Furthermore, datasets can be created incrementally and combined with others, allowing collaborative and progressive development of a complete reference dataset..

In this sense, WinMET [26] represents a significant advancement in malware research [27]. Unlike other public datasets, WinMET provides a wide range of malware behavior characteristics, including detailed process information, API calls, parameters/results, resource access, and synchronization details. This comprehensive dataset enables in-depth analysis of malware operations and interactions, making it a valuable resource for developing effective detection methods. Its size and detailed data provide a robust foundation for building widely accepted reference datasets, and its JSON format simplifies data interpretation and conversion, facilitating its use in current detection technologies such as those based on machine learning.

Both MALVADA and WinMET are open source and publicly available [22, 26], aligning with the principles of open science and aiming to facilitate their widespread use by the research community.

5. Conclusions

In this paper, we introduce MALVADA, a framework for generating malware execution trace datasets from sandbox reports (specifically, CAPE), enhanced with classification tools. Its key advantages are detailed behavioral insights and the ability to incrementally build large datasets. As malware detection increasingly relies on knowledge extraction and AI models, the need for high-quality datasets increases significantly. MALVADA addresses this need and supports the development of improved detection models. We also release the WinMET dataset, aiming to provide a valuable resource for researchers to collaboratively extend and contribute to a comprehensive reference dataset for the malware research community.

Acknowledgements

This research was supported in part by TED2021-131115A-I00 (MIMFA), funded by MCIN/AEI/10.13039/501100011033, by the Recovery, Transformation and Resilience Plan funds, financed by the European Union (Next Generation), by the Spanish National Cybersecurity Institute (INCIBE) under *Proyectos Estratégicos de Ciberseguridad – CIBERSEGURIDAD EINA UNIZAR*, and by the University, Industry and Innovation Department of the Aragonese Government 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, during 2021–2025.

References

[1] S. Sebastián, J. Caballero, AVclass2: Massive Malware Tag Extraction from AV Labels, in: Proceedings of the 36th Annual Computer Security Applications Conference, ACSAC '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 42–53.

[2] K. Zettl-Schabath, J. Bund, M. Müller, C. Borrett, EuRepoC Cyber Conflict Briefing – 2023 Cyber Activity Balance, [Online; `https://eurepoc.eu/publication/eurepoc-cyber-conflict-briefing-2023-cyber-activity-balance/`], accessed on June 17, 2024. (January 2024).

[3] d. A. Aslan, R. Samet, A Comprehensive Review on Malware Detection Approaches, IEEE Access 8 (2020) 6249–6271.

[4] P. K. Mvula, P. Branco, G.-V. Jourdan, H. L. Viktor, Evaluating Word Embedding Feature Extraction Techniques for Host-Based Intrusion Detection Systems, Discover Data 1 (1) (2023) 2.

[5] J. Dai, R. K. Guha, J. Lee, Efficient Virus Detection Using Dynamic Instruction Sequences, J. Comput. 4 (5) (2009) 405–414.

[6] R. Canzanese, S. Mancoridis, M. Kam, System Call-Based Detection of Malicious Processes, in: 2015 IEEE International Conference on Software Quality, Reliability and Security, 2015, pp. 119–124.

[7] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, X. Wang, Effective and Efficient Malware Detection at the End Host, in:

Proceedings of the 18th Conference on USENIX Security Symposium, SSYM'09, USENIX Association, USA, 2009, pp. 351–366.

[8] E. Amer, I. Zelinka, A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence, Computers & Security (2 2020).

[9] Y. Qiao, Y. Yang, L. Ji, J. He, Analyzing Malware by Abstracting the Frequent Itemsets in API Call Sequences, in: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 2013, pp. 265–270.

[10] StatCounter, Desktop Operating System Market Share Worldwide, [Online; https://gs.statcounter.com/os-market-share/desktop/worldwide], accessed on July 22, 2024. (2024).

[11] AV-TEST, Malware Statistics & Trends Report, [Online; https://www.av-test.org/en/statistics/malware/], accessed on Ago 2, 2024. (2024).

[12] Y. Ki, E. Kim, H. K. Kim, A Novel Approach to Detect Malware Based on API Call Sequence Analysis, International Journal of Distributed Sensor Networks 11 (6) (2015).

[13] F. O. Catak, A. F. Yazı, O. Elezaj, J. Ahmed, Deep learning based Sequential model for malware analysis using Windows exe API Calls, PeerJ Computer Science 6 (2020).

[14] C. W. Kim, Ntmaldetect: A machine learning approach to malware detection using native api system calls, ArXiv (2018).

[15] R. J. S. Angelo Schranko De Oliveira, Behavioral Malware Detection using Deep Graph Convolutional Neural Networks, International Journal of Computer Applications 174 (29) (2021) 1–8.

[16] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, M. Ahamad, An Inside Look into the Practice of Malware Analysis, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 3053–3069.

[17] D. Kim, H. K. Kim, Automated Dataset Generation System for Collaborative Research of Cyber Threat Analysis, Security and Communication Networks 2019 (1) (2019) 6268476.

[18] D. Čeponis, N. Goranin, Towards a robust method of dataset generation of malicious activity for anomaly-based HIDS training and presentation of AWSCTD dataset, Baltic Journal of Modern Computing 6 (3) (2018) 217–234.

[19] K. O'Reilly, A. Brukhovetskyy, CAPE: Malware Configuration And Payload Extraction, [Online; `https://github.com/kevoreilly/CAPEv2`], accessed on May 22, 2024. (2019).

[20] R. Raducu, R. J. Rodríguez, P. Álvarez, CAPE Hook Generator, [Online; `https://github.com/reverseame/cape-hook-generator`], accessed on Jul 24, 2024. (Jun. 2024).

[21] VirusTotal, [Online; `https://www.virustotal.com/`], accessed on Jul 22, 2024.

[22] R. Raducu, A. Villagrasa-Labrador, R. J. Rodríguez, P. Álvarez, MALVADA - A Windows Malware Execution Traces Dataset generation

framework, [Online; https://github.com/reverseame/MALVADA], accessed on Jul 24, 2024. (2024).

[23] O'Reilly, Kevin and Brukhovetskyy, Andriy, CAPE Sandbox Book, [Online; https://capev2.readthedocs.io/en/latest/index.html], accessed on Jul 29, 2024. (2024).

[24] O'Reilly, Kevin, capemon: CAPE's monitor, [Online; https://github.com/kevoreilly/capemon], accessed on Jul 29, 2024. (2024).

[25] Microsoft, PE Format, [Online; https://learn.microsoft.com/en-us/windows/win32/debug/pe-format], accessed on Ago 2, 2024. (2024).

[26] R. Raducu, A. Villagrasa-Labrador, R. J. Rodríguez, P. Álvarez, WinMET Dataset, [Online; https://doi.org/10.5281/zenodo.12647555], accessed on Ago 6, 2024. (2024).

[27] M. Botacin, F. Ceschin, R. Sun, D. Oliveira, A. Grégio, Challenges and pitfalls in malware research, Computers & Security 106 (2021) 102287.