

Article

# Going beyond API Calls in Dynamic Malware Analysis: A Novel Dataset

Slaviša Ilić <sup>1,2,\*</sup> , Milan Gnjatović <sup>2</sup> , Ivan Tot <sup>1</sup>, Boriša Jovanović <sup>1</sup> , Nemanja Maček <sup>3</sup>   
and Marijana Gavrilović Božović <sup>4</sup> 

<sup>1</sup> Department of Military Electronic Engineering, University of Defence, Veljka Lukića Kurjaka 1, 11000 Belgrade, Serbia; ivan.tot@va.mod.gov.rs (I.T.); borisa.jovanovic@vs.rs (B.J.)

<sup>2</sup> Department of Information Technology, University of Criminal Investigation and Police Studies, Cara Dušana 196, 11080 Beograd, Serbia; milan.gnjatovic@kpu.edu.rs

<sup>3</sup> School of Electrical and Computer Engineering, Academy of Technical and Art Applied Studies, Vojvode Stepe 283, 11000 Beograd, Serbia

<sup>4</sup> Faculty of Engineering, University of Kragujevac, Sestre Janjić 6, 34000 Kragujevac, Serbia; marijana.gavrilovic@kg.ac.rs

\* Correspondence: ilic.slavisa@gmail.com

**Abstract:** Automated sandbox-based analysis systems are dominantly focused on sequences of API calls, which are widely acknowledged as discriminative and easily extracted features. In this paper, we argue that an extension of the feature set beyond API calls may improve the malware detection performance. For this purpose, we apply the Cuckoo open-source sandbox system, carefully configured for the production of a novel dataset for dynamic malware analysis containing 22,200 annotated samples (11,735 benign and 10,465 malware). Each sample represents a full-featured report generated by the Cuckoo sandbox when a corresponding binary file is submitted for analysis. To support our position that the discriminative power of the full-featured sandbox reports is greater than the discriminative power of just API call sequences, we consider samples obtained from binary files whose execution induced API calls. In addition, we derive an additional dataset from samples in the full-featured dataset, whose samples contain only information on API calls. In a three-way factorial design experiment (considering the feature set, the feature representation technique, and the random forest model hyperparameter settings), we trained and tested a set of random forest models in a two-class classification task. The obtained results demonstrate that resorting to full-featured sandbox reports improves malware detection performance. The accuracy of 95.56 percent obtained for API call sequences was increased to 99.74 percent when full-featured sandbox reports were considered.

**Keywords:** dynamic malware analysis; classification; cuckoo; sandbox; API; random forest; dataset



**Citation:** Ilić, S.; Gnjatović, M.; Tot, I.; Jovanović, B.; Maček, N.; Gavrilović Božović, M. Going beyond API Calls in Dynamic Malware Analysis: A Novel Dataset. *Electronics* **2024**, *13*, 3553. <https://doi.org/10.3390/electronics13173553>

Academic Editors: Lixin Wang, Qiang Ye and Jianhua Yang

Received: 6 August 2024

Revised: 4 September 2024

Accepted: 5 September 2024

Published: 6 September 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Due to the rise in the number and complexity of cyber threats, dynamic malware analysis has become an important part of cyber defense. This type of malware analysis is often performed in a sandbox environment, i.e., a virtual machine (VM) environment in which suspected binaries are seeded and their dynamic behavior is monitored. The monitoring result is represented by a report containing evidence of actions performed by a given seeded binary.

However, the interpretation of those reports may be a challenging task, e.g., a benign installation package that generates a significant number of files (which are written to the disk), registry changes, and other modifications to the target system (starting multiple processes, establishing network communication, etc.), might not be easily distinguished from a malware which applies evasion techniques (i.e., a malware that changes its behavior when it detects that it is being run in a sandbox environment) [1]. Thus, the interpretation of sandbox reports often requires a human expert, which makes it time consuming.

To address this problem, relevant machine learning approaches to dynamic malware analysis are dominantly focused on application programming interface (API) calls extracted by a sandbox system. However, in our study, we show that a significant part of both malware and benign software samples do not generate API calls when run in a sandbox environment (cf. Section 3). Thus, we argue that the extension of a feature set beyond API calls may improve malware detection performance. In line with this, the paper makes the following contributions:

(i) We produce a novel dataset for dynamic malware analysis, which contains 22,200 labeled samples (benign and malware). Each sample represents a report generated by the open-source sandbox system Cuckoo when a corresponding binary file is submitted for analysis. These reports contain information on API calls but also on other dynamic behavior features related to file and registry modification, network traffic, etc. It is important to note that there are no other publicly available datasets containing both malware and benign samples represented by a more comprehensive set of features extracted by a sandbox system (cf. Section 2). In addition, we describe the configuration of the sandbox system used in the production of the dataset, aimed at addressing the problem of evasion techniques applied by some malware samples.

(ii) We practically demonstrate that the extension of the feature set beyond API calls improves the malware detection performance. We report on two groups of random forest (RF) models (cf. Ho [2]). The models from the first group were trained and tested on the reported dataset containing full-featured Cuckoo sandbox reports (including, but not limited to, API call sequences). The models from the second group were trained and tested on a dataset derived from the reported dataset, in which each sample is represented by the extracted sequence of API calls. The obtained results show that the models from the first group display better performance in terms of accuracy and macro average F1 measure.

The paper is structured as follows. Section 2 covers related and background work. Section 3 describes the production of the dataset and methods used. Section 4 reports on the experimental settings, results, and optimization. In Section 5, we discuss the results. Finally, Section 6 concludes the paper.

## 2. Background and Related Work

With respect to the feature extraction task, the field of machine learning for dynamic malware analysis tasks includes two main research directions. The first direction is dominant and supported by datasets containing only API call sequences (cf. Section 2.1). The other direction is significantly less practiced but promising and is based on the assumption that there are additional significant features beyond the API call sequence (cf. Section 2.2).

### 2.1. API Call Feature Sets

In the given context, API refers to a part of an operational system which, inter alia, enables programs to access resources such as system files, processes, services, etc. (Mira et al. [3]). In the cyber-security community, API call sequences have been acknowledged as indicative of malware detection (Deore et al. [4]). The main idea underlying these approaches is that API call sequences reflect the malware behavior and that malware variants display similar API call-based behavior. For an overview of malware detection using API call sequences, the reader may consult [3].

Düzgün et al. [5] and Alshmarni et al. [6] report on datasets produced by the Cuckoo sandbox. The dataset presented by Düzgün et al. [5] contains a significant number (i.e., 24,411) of categorized malware samples but does not include benign samples. The dataset presented by Alshmarni et al. [6] contains both benign and malware samples, but their number is somewhat reduced (i.e., 1080 benign and 2576 malware). In addition, both of these datasets contain a restricted feature set since each sample is represented as only a sequence of API calls.

Syeda and Asghar [7] report on a dataset containing samples from Windows Portable Executable files. This dataset is relatively small (582 malware and 438 benign samples) and contains API calls only. Six machine learning models were evaluated on this dataset, including inter alia

ensemble methods RF and XGBoost. The RF model displayed the best classification accuracy. Zhang et al. [8] propose an approach to malware detection based on recurrent neural networks (including TextRNN and bidirectional long short-term memory network) and self-attention mechanisms applied to API call sequences. Huang et al. [9] apply a clustering algorithm to API call sequences to capture a common behavior of malware families and then train a neural network on the recorded sequences. Other papers also deal with different approaches to API calls: Huang et al. [10] propose a model based on a sequence-to-sequence neural network and an attention mechanism and evaluate it on sequences of API calls. Chen et al. [11] propose an image-based interpretation of API existence, API sequence, and API frequency features and apply deep transfer learning to generated images.

To further improve malware detection performance, a significant number of researchers have considered additional API call-based features. Alhashmi et al. [12] apply the TF-IDF transformation to extract features from API calls retrieved by means of the static analysis of PE files and dynamic analysis of API hooks. Lee et al. [13] report on a sandbox-like implementation called ARBDroid, designed to analyze obfuscated malware on the Android platform and extract instructions, string values, and API calls. Chen et al. [14] extend the set of API calls with indicators of compromise. Yau et al. [15] consider additional API-based statistical features such as return values and the number of times the API is called. Xu and Chen [16] model a malware instance as a behavioral tree generated from an API call sequence, which describes the control structure between the API calls. Li et al. [17] propose a hybrid feature encoder aimed at extracting semantic features from API names and arguments and generating an API call graph that captures the relations between the API calls. Li et al. [18] apply dynamic analysis and embedding techniques to encode an API sequence, introducing a soft threshold mechanism in a residual network to achieve active filtering of noise components in API sequences. They resort to a BiLSTM model to enhance the temporal correlation among API sequence calls and improve the classification performance. Nunes et al. [19] emphasize the difference between capturing API calls at the user and kernel levels with respect to the classifier's ability to detect malware. Li et al. [20] introduced the PrefixSpan algorithm designed to mine frequent API call sequences from various threads within an execution trace and capture malware behavior sequences.

## 2.2. More Comprehensive Feature Sets

In contrast to these API call-focused approaches to dynamic malware analysis, some recent efforts are based on more comprehensive feature sets. Jindal et al. [21] produced four datasets containing reports generated by the Cuckoo sandbox and another commercial sandbox. Two sources have served as input to the sandboxes: a vendor dataset (13,760 benign and 13,760 malware) and a dataset derived by random sample selection from the publicly available EMBER dataset for static file analysis (cf. Anderson and Rothl [22]). However, none of these four datasets are publicly available. In the data preprocessing phase, Jindal et al. [21] tokenize the sandbox reports and keep the top 10,000 most common words. In addition, they introduce a classification model that integrates a convolutional neural network, a bidirectional long short-term memory network, and an attention network.

Another commendable approach is introduced by Bosansky et al. [23]. They report on a malware dataset with a significant number of samples (i.e., 48,976 samples) generated by the CAPE sandbox. Another advantage of this dataset is that each sample is labeled with respect to the malware family it belongs to. On the other hand, this dataset does not contain benign samples and considers only six malware families. For example, ransomware samples are not included, which explains the fact that the maximum report size of the considered samples does not exceed 1 GB (the size of the entire unpacked dataset is 563 GB). Bosansky et al. [23] apply the hierarchical multi-instance learning (HMIL) paradigm to process the collected sandbox reports.

Herrera-Silva and Hernández-Álvarez [24], Irshad and Dutta [25], and Sraw and Kumar [26] extend the API call feature set with limited sets of hand-selected features derived from the Cuckoo sandbox reports. These sets contain 50, 5, and 8 additional features, respectively,

relating to, e.g., registry entries, network traffic, etc. They demonstrate that additional features improve malware detection performance. Finally, Sethi et al. [27] extend the API call feature set with a set of features automatically extracted from the Cuckoo sandbox reports. The additional features were extracted by means of the chi-square test and RF algorithms. However, none of these approaches to feature engineering utilize complete sandbox reports.

### 2.3. Position of Our Approach

Our approach is in line with the second research direction. In line with Refs. [21,23], we produce a dataset containing reports generated by the Cuckoo sandbox. However, in contrast to Ref. [21], we do not consider only a subset of the most common words in the generated reports. Starting from the expectation that less frequent words may have significant information gain, we consider entire and unaltered reports generated by the sandbox system. This means that we do not apply any preprocessing or cleaning technique prior to the feature extraction stage.

In contrast to Ref. [23], we include benign samples and additional malware families, which increases the size of the generated reports, e.g., over 1 GB for 228 malware samples. In addition, it should be noted that there are no other publicly available datasets containing full-featured sandbox reports with such a large number of samples.

Another distinction of our approach is related to the methodology. Recent methodological approaches are focused on recurrent neural networks and attention mechanisms [8,10,11,21] or ensemble detectors, taking into account the observation window [1], which reflects the conceptualization of the underlying features as sequence-like structures. In contrast to them, in our approach, we step away from this conceptualization and apply the random forest model (as in Ref. [7]), which is based on the information gain of the underlying features rather than on their sequentiality.

## 3. Materials and Methods

One of the aims of this work is to produce a new dataset that meets the desiderata of representativeness, i.e., a dataset for dynamic malware analysis that contains a sufficient number of both benign and malware samples, represented by a more comprehensive set of features generated by the Cuckoo sandbox. The production of this dataset can be summarized in the following points:

(i) *Selection and safe management of malware files.* The collection of 450,000 malware file samples was acquired from the Virus Total database [28], which is considered referential in the professional community. From this file set, we programmatically selected 10,465 samples without subjective bias for inclusion in the dataset. Each malware sample taken from this database is provided in the form of a binary file without extension but with the accompanying JSON file that describes the given sample. In addition, we took measures to ensure that the management of the malware files does not compromise the lab environment set for the purpose of dataset production. The file samples were stored on the disk without extensions. Immediately before the submission of a sample to the sandbox, its extension was automatically extracted from a corresponding JSON file and added to the sample, making it suitable for execution. After collecting sandbox results, the extension was again removed.

(ii) *Selection of benign files.* We collected 74,000 real-life benign files representing correspondence documents and Windows 10 installation executables and libraries (e.g., .exe, .dll, .com, etc.), taken with permission from the archive of a technical company located in Serbia. The company was equipped with cyber-security protection and monitoring tools, and no malware activities have been detected over several years. From this file set, we programmatically selected 11,739 samples without subjective bias for inclusion in the dataset.

(iii) *Dataset comprehensiveness.* The dataset comprehensiveness was considered with respect to two separate but related aspects: diversity of the collected file types and representation of a file sample. The reported dataset contains samples (i.e., reports generated by the sandbox) related to 54 file types. The distribution of the samples with respect to file type is provided in Table 1 (cf. the “Full rep.” columns). In addition, for each file type, this table provides the number of samples that include API calls (cf. the “API calls” columns).

Table 1. File type distribution in the dataset.

File Type	# Benign Instances		# Malware Instances		File Type	# Benign Instances		# Malware Instances	
	Full rep.	API Calls	Full rep.	API Calls		Full rep.	API Calls	Full rep.	API Calls
exe	1395	344	7097	6597	Xlsm	0	0	5	5
dll	6574	3354	238	237	accdb	5	5	0	0
doc	1659	1659	947	947	Cat	5	5	0	0
xls	631	631	477	475	Gdl	5	5	0	0
txt	4	0	803	803	xml	5	5	0	0
fxp	558	558	0	0	<b>Db</b>	<b>4</b>	<b>0</b>	<b>0</b>	<b>0</b>
docx	0	0	269	269	Pptx	0	0	4	4
<b>pdf</b>	<b>118</b>	<b>0</b>	<b>120</b>	<b>0</b>	Sch	3	3	0	0
<b>cdx</b>	<b>223</b>	<b>0</b>	<b>0</b>	<b>0</b>	Ppt	0	0	3	3
dbf	222	222	0	0	Avi	2	2	0	0
xlsx	0	0	190	190	Bin	2	2	0	0
prg	138	138	0	0	<b>bmp</b>	<b>2</b>	<b>0</b>	<b>0</b>	<b>0</b>
html	0	0	128	128	Och	2	2	0	0
<b>none</b>	<b>0</b>	<b>0</b>	<b>60</b>	<b>59</b>	Tbk	2	2	0	0
ppd	52	52	0	0	Msg	0	0	2	2
docm	0	0	45	45	agcoc	1	1	0	0
fpt	35	35	0	0	Bak	1	1	0	0
fpx	0	0	33	33	Fmt	1	1	0	0
<b>zip</b>	<b>7</b>	<b>0</b>	<b>17</b>	<b>0</b>	<b>Ico</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>0</b>
<b>rar</b>	<b>0</b>	<b>0</b>	<b>24</b>	<b>23</b>	Mem	1	1	0	0
htm	19	19	0	0	Ms	1	1	0	0
crt	12	12	0	0	New	1	1	0	0
inf	10	10	0	0	pr1	1	1	0	0
bat	9	9	0	0	Ses	1	1	0	0
<b>dat</b>	<b>8</b>	<b>0</b>	<b>0</b>	<b>0</b>	Json	0	0	1	1
ini	8	8	0	0	mp3	0	0	1	1
lnk	7	7	0	0	Rtf	0	0	1	1
					Total:	11,735	7097	10,465	9823

The rows marked with grey highlight the file type classes containing samples that did not generate API calls, which is elaborated in more detail in Table 2.

For each file type, Table 1 provides information on:

- Number of benign files belonging to the given file type (cf. column “# Benign instances”—“Full rep.”).
- Number of benign files belonging to the given file type whose execution generates API calls (cf. column “# Benign instances”—“API calls”).
- Number of malware files belonging to the given file type (cf. column “# Malware instances”—“Full rep.”).
- Number of malware files belonging to the given file type whose execution generates API calls (cf. column “# Malware instances”—“API calls”).

From Table 1, it can be derived that 23.78 percent of the produced sandbox reports do not include API calls, i.e., there are 5280 such samples: 4638 benign and 642 malware. The distribution of samples that do not include API calls with respect to file type is shown in Table 2. It is common that benign files do not induce operating system API calls, e.g., in cases when they do not create files or access the registry, etc. On the other hand, it is

common that malware samples induce API calls, as they are designed to be lightweight (easily downloadable) and access the registry and files, etc. However, it can be observed from Table 2 that approximately 6 percent of malware samples do not induce API calls. We assume that these malware samples are intentionally developed not to apply operating system APIs in order to avoid antivirus behavioral analysis and possible sandbox detection based on API calls.

In contrast to the existing datasets, which represent file samples as sequences of API calls, we consider a more comprehensive set of features provided by the Cuckoo sandbox. Besides the API calls, we collect additional features relevant to dynamic malware analysis related to the following:

- Static and dynamic analysis based on YARA rules for code packing, obfuscation, etc.;
- Registry editing;
- File creation and modification;
- Network access;
- Checking user activities and other evasion techniques and behaviors of file samples, etc.

**Table 2.** The distribution of samples that do not include API calls with respect to file type.

File Type Extension	Instances	Benign Instances	Malware Instances
dll	3221	3220	1
exe	1551	1051	500
pdf	238	118	120
cdx	223	223	0
zip	24	7	17
dat	8	8	0
txt	4	4	0
db	4	4	0
xls	2	0	2
bmp	2	2	0
rar	1	0	1
ico	1	1	0
none	1	0	1
	5280	4638	642

Without omitting the basic static analysis, which is included in the Cuckoo sandbox, we do not perform additional analysis of static features (like in Taheri et al. [29,30]) or compare static and dynamic analyses. We focus our attention on dynamic malware analysis based on raw full-featured sandbox reports (without giving preferences to a particular feature subset), which serve as input for automatic feature extraction techniques and machine learning models.

The produced dataset, which we refer to as the full-featured dataset, is organized in two folders containing benign and malware samples. The size of the malware sample folder is 935 GB, and the size of the benign sample folder is 34 GB. Both folders are organized in subfolders whose names correspond to file type extensions (e.g., “dll”, “exe”, “doc”, etc.). The malware samples folder is of greater size because the malware files induced more activities in the sandbox environment than the benign files. Two short segments from JSON reports are given in Figure 1.

```

{
  "info": {
    "added": 1709568693.0,
    "started": 1709568693.0,
    "duration": 200,
    "ended": 1709568893.0,
    "owner": "",
    "score": 0.4,
    "id": 11382,
    "category": "file",
    "git": {
      "monitor": "2deb9ccd75d5a7a3fe05b2625b03a8639d6ee36b",
      "package": "",
      "route": "internet",
      "custom": "",
      "machine": {
        "platform": "",
        "version": "2.0.7",
        "options": ""
      }
    },
    "procmemory": [
      {
        "regions": [
          {
            "yara": [],
            "num": 1,
            "file": "/home/kruska/.cuckoo/storage/analyses/11382/procmem/1",
            "urls": [],
            "extracted": [
              {
                "pid": 1956
              }
            ]
          }
        ]
      }
    ],
    "target": {
      "category": "file",
      "file": {
        "yara": [],
        "sha1": "68c301701b660e41f47c69f91b1350295f6c69e1",
        "name": "2018 July Price Increase announcement.pdf",
        "type": "PDF document, version 1.5",
        "sha256": "31c07eef2e97dbeb6080157378a4c0781ecdbf",
        "urls": [
          "http://ns.adobe.com/pdf/1.3/",
          "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
          "http://purl.org/dc/elements/1.1/",
          "http://ns.adobe.com/xap/1.0/mm/",
          "http://ns.adobe.com/ix/1.0/",
          "http://ns.adobe.com/xap/1.0/"
        ]
      },
      "crc32": "12FC8238",
      "path": "/home/kruska/.cuckoo/storage/binaries/31c07eef2e97dbeb6080157378a4c0781ecdbf",
      "ssdeep": "6144:42m4tkxY0ks1ON5yu5vraQ7ne1cdedWgSI",
      "size": 226726
    }
  }
}

```

```

"signatures": [
  {
    "families": [],
    "description": "Creates (office) documents on",
    "severity": 2,
    "cttp": {},
    "markcount": 1,
    "references": [],
    "marks": [
      {
        "category": "file",
        "ioc": "C:\\Users\\Administrator\\AppD",
        "type": "ioc",
        "description": null
      }
    ],
    "name": "creates_doc"
  },
  {
    "families": [],
    "description": "Creates hidden or system file",
    "severity": 2,
    "cttp": {
      "T1158": {
        "short": "Hidden Files and Directories",
        "long": "To prevent normal users from"
      }
    },
    "markcount": 1,
    "references": [],
    "marks": [
      {
        "call": {
          "category": "file",
          "status": 1,
          "stacktrace": [],
          "api": "NtCreateFile",
          "return_value": 0,
          "arguments": {
            "create_disposition": 5,
            "file_handle": "0x000007b0",
            "filepath": "C:\\Users\\Admini",
            "desired_access": "0x40100080",
            "file_attributes": 2,
            "filepath_r": "\\?\\C:\\Users",
            "create_options": 4194400,
            "status_info": 2,
            "share_access": 0
          }
        },
        "time": 1707541564.181616,
        "tid": 2972
      }
    ]
  }
]

```

**Figure 1.** Example segments of benign (left) and malware (right) JSON reports generated by the Cuckoo sandbox.

The malware samples are randomly selected from the Virus Total database, which contains real-world malware examples. The manual inspection of the sandbox reports showed the following:

- Some reports are very big in size (over 2 GB).
- Some samples apply obfuscation techniques (discovered in the basic static file analysis conducted by the sandbox system).
- Some samples implement evasion techniques: checking the browser history, checking whether the running environment is virtual, checking whether a debugger is present, etc.

The benign samples are also real-world files produced over a several-year period by users who were oblivious to the goals of this research.

Thus, we believe that the reported dataset represents a comprehensive real-world representation of both malware and benign samples.

(iv) *Configuration of the sandbox system.* The Cuckoo sandbox version 2.0.7 was used (i.e., the latest release at the moment of this study) based on Python version 2. These particular settings required additional efforts to select correct program dependencies, i.e., adjusting dependency information among program variables, which is discussed in Ilić et al. [31]. Except for the Cuckoo sandbox system, all other codes used in the conduction of this research (i.e., codes for sample submission, reports collection, manipulation, training and validation of machine models, visualization of the results, etc.) were implemented in the latest Python 3 version.

(v) *Selection of the analysis virtual machine operating system.* We set a basic configuration of a Windows 7 virtual machine, which is more vulnerable with respect to malware attacks than more recent operating systems. Microsoft Office 2013 is installed, which supports a wide set of document formats present in the dataset.

(vi) *Configuration of anti-evasion techniques.* We introduced or enabled a set of anti-evasion techniques in the Cuckoo sandbox:

- We configured a Python agent, which monitors events while file samples are being executed and runs silently in the background.
- The analysis virtual machine is configured with usage history and data, a custom username, and software. Special attention was devoted to the browsers' history since we previously noticed frequent checks of Internet browser activities conducted by some malware samples.
- The VMware tools were intentionally omitted, and processing resources of the analysis virtual machine were unusually abundant for a sandbox in order to prevent sophisticated malware samples from recognizing and evading the sandbox techniques.
- The user action imitation technique (e.g., mouse move, click, etc.) integrated into the Cuckoo sandbox was enabled, and the execution time was increased to two minutes per file sample.
- The analysis virtual machine had limited Internet access with the firewall configured in front of the laboratory environment.

(vii) *Configuration of the Python runtime environment.* Some malware file samples, e.g., ransomware samples, perform a significant number of activities when executed in the sandbox system. Thus, the generated sandbox analysis reports were of significant size (i.e., multiple gigabytes). To process such large inputs during the generation of machine learning models, it was necessary to adjust the Python runtime environment by increasing the maximum recursion depth in order to allow for the loading of entire reports from JSON files. Without this change it would not be possible to deal with reports of significant size.

(viii) *Generation of the API call dataset for the feature ablation study.* One of the aims of this study is to demonstrate that the discriminative power of the full-featured sandbox reports discussed in (iii) is greater than the discriminative power of just API call sequences. Thus, for the purpose of the ablation study, we derive a subset that we refer to as the API call dataset. Samples in this dataset contain only information on API calls derived from samples in the full-featured dataset (cf. the "API calls" columns in Table 1). The generation of the API call dataset was performed automatically by applying a regular expression-based Python script. The size of the benign and malware portions of the API call dataset are 848 MB and 18 GB, respectively.

## 4. Results

### 4.1. Model and Metrics Selection

In Section 2.3, we briefly discussed the application of random forest models in our malware analysis system. For this purpose, we resort to the scikit-learn library (cf. Pedregosa et al. [32]). However, the training and validation of the underlying models have been performed in experimental settings. We apply a three-way factorial design, i.e., we consider three independent variables: the feature set, the feature representation technique, and the random forest model hyperparameter settings.

The first independent variable is related to the feature set and has two levels: we apply a dataset containing full-featured reports generated by the Cuckoo sandbox and a dataset whose samples contain only API call sequences. Both datasets described in the previous section (i.e., the full-featured and API call datasets) are transformed into pandas data structures (cf. McKinney et al. [33]) and stored as Python pickle files (i.e., byte streams). In addition, each sample is annotated: the label "0" is assigned to benign samples and "1" to malware samples. However, the full feature dataset was further processed. Since the malware part of this dataset was too big to fit into RAM memory, we divided it into three consecutive Python pickle files and then performed the subsampling process in the following steps:



- Loading the panda's data frames and shuffling them.
- Subsampling, i.e., we randomly selected 70 percent of each panda's data frame, keeping the original textual reports representing the selected samples.
- Storing, i.e., all subsample instances were stored in one data frame, which is additionally shuffled. This data frame was transformed into a pickle file.

After this subsampling phase, the selected part of the full features dataset contained 15,542 randomly selected samples (i.e., 8217 benign and 7325 malware).

The second independent variable is related to the feature representation technique and has two levels: we apply the count vectorizer (CV) and the term frequency-inverse document frequency vectorizer (TF-IDF), cf. Pedregosa et al. [32]. Using these techniques, we wanted to examine whether taking into account the importance of a feature to a sample (TF-IDF) will improve performance when compared to taking only term frequencies (CV) into account. For both feature representation techniques, we randomly divided each of the given datasets into a training set (80 percent) and a validation set (20 percent). After combining the first two independent variables (i.e., 2 feature sets  $\times$  2 feature representation techniques), we generated four pickle-based datasets:

- Full-featured samples represented by CV;
- Full-featured samples represented by TF-IDF;
- API calls samples represented by CV;
- API calls samples represented by TF-IDF;

which serve as points of departure for model training and validation.

The third independent variable relates to the random forest model hyperparameters. The first considered hyperparameter is the number of trees in a random forest. This parameter takes values ranging [1, 100]. Other hyperparameters of an RF model include the following: *max\_depth*, *min\_samples\_split*, *min\_samples\_leaf*, and *max\_features*, each of which takes three distinct values ([None, 100, 1000], [2, 100, 1000], [1, 100, 1000], and [sqrt, 0.1, 0.2], respectively). The hyperparameter values are discussed in more detail in Section 4.3.

We apply information gain to estimate the discriminative power of an explanatory feature with respect to the class feature (i.e., the "criterion" hyperparameter was set to the value "entropy"). Finally, we applied the accuracy measure as the dependent variable.

When the independent variables were combined, we obtained a  $2 \times 2 \times (100 \times 9)$  factorial design, i.e., we trained and tested 3600 random forest models.

#### 4.2. Experiment

With respect to the underlying dataset (i.e., feature set), these models can be divided into four groups, each of which contains 900 models. Table 3 provides additional information for the optimal model among each of the considered groups. For each of these models, a set of evaluation measure values have been provided (i.e., validation accuracy, precision, and recall for each class, and macro average precision, recall, and F1-score), and information of the underlying dataset (i.e., the number and distribution of samples and the number of features).

The first group of models is related to the experimental condition concerning the full-featured reports represented with CV. Figure 2(left) provides a graphical representation of the validation accuracy with respect to the number of decision trees in a random forest model. The maximum validation accuracy of 99.74 percent was obtained for the random forest model with seven trees. The confusion matrix obtained for this model is given in Figure 2(right), and the derived evaluation measures are provided in Table 3 (cf. the second column).

**Table 3.** Results, control mechanisms, and metrics in RF.

	CV Full	TF-IDF Full	CV API Calls	TF-IDF API Calls
Num. of trees in the optimal model	7	26	32	66
Validation accuracy (%)	99.74	99.68	95.56	95.4
Precision (benign)	0.997	0.997	0.935	0.933

Table 3. Cont.

	CV Full	TF-IDF Full	CV API Calls	TF-IDF API Calls
Precision (malware)	0.998	0.996	0.982	0.982
Macro average precision	0.997	0.998	0.959	0.957
Recall (benign)	0.998	0.997	0.985	0.985
Recall (malware)	0.996	0.998	0.921	0.918
Macro-average recall	0.997	0.995	0.953	0.952
Macro-average F1	0.997	0.997	0.954	0.954
Num. of samples	15,542	0.997	15,542	15,542
Num. of features	25,066,934	25,066,934	294	294
Benign samples	8217	8217	4968	4968
Malware samples	7325	7325	6891	6891

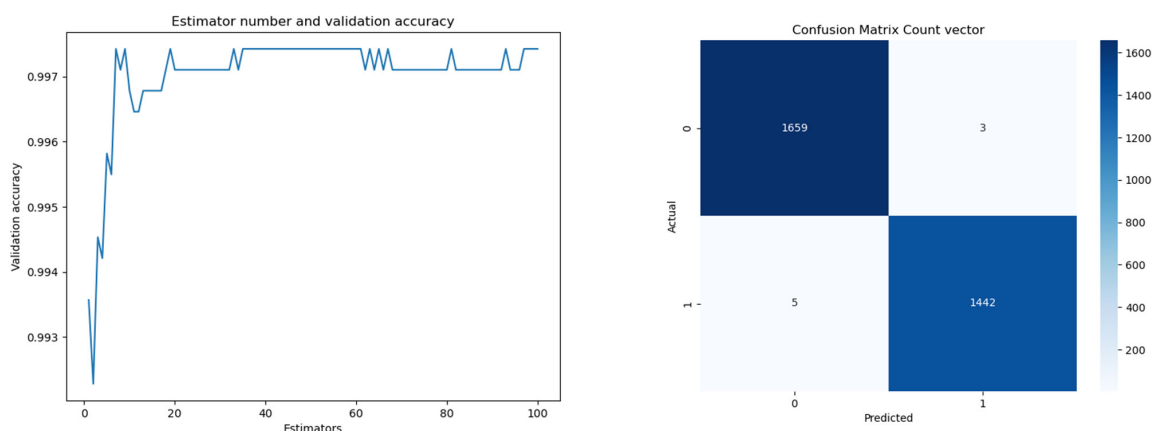


Figure 2. Full-featured reports represented with CV: (left) a graphical representation of the validation accuracy with respect to the number of decision trees in a random forest model; (right) the confusion matrix obtained for the optimal random forest model (0—benign, 1—malware).

The second group of models is related to the experimental condition concerning the full-featured reports represented with TF-IDF. A graphical representation of the evaluation results is provided in Figure 3(left). The maximum validation accuracy of 99.68 percent was obtained for the random forest model with 26 trees. The confusion matrix obtained for this model is given in Figure 3(right), and the derived evaluation measures are provided in Table 3 (cf. the third column).

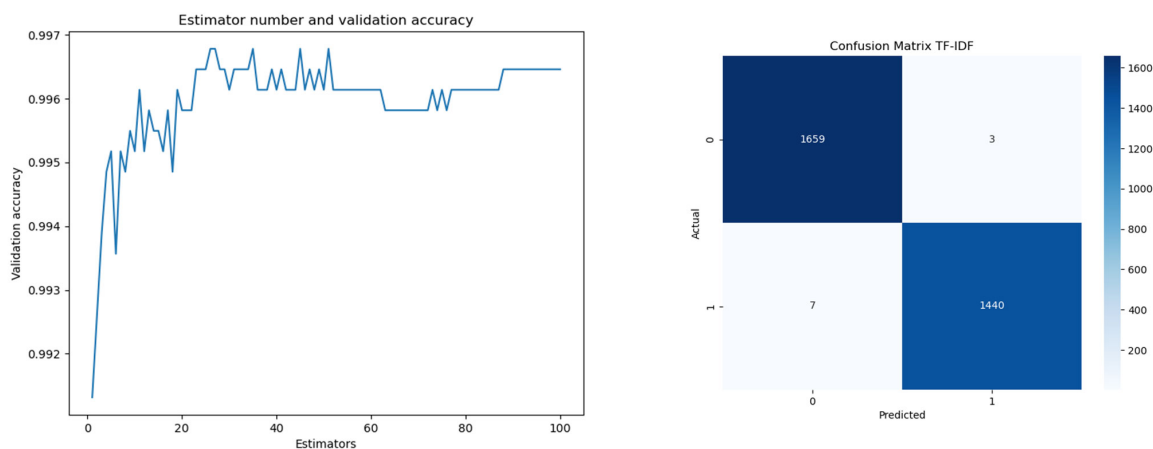
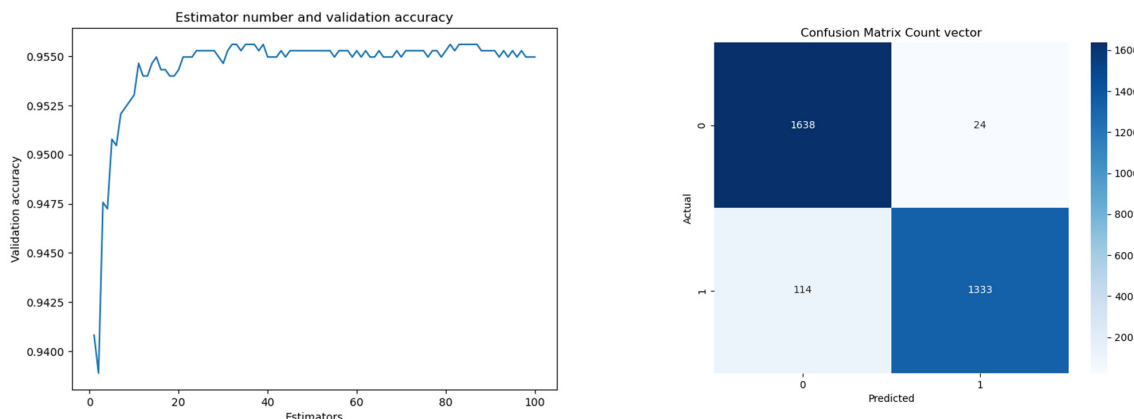


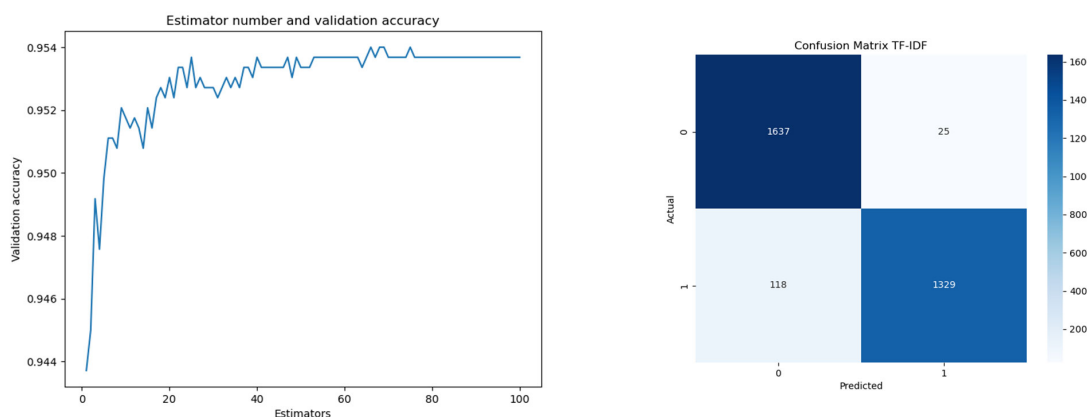
Figure 3. Full-featured reports represented with TF-IDF: (left) a graphical representation of the validation accuracy with respect to the number of decision trees in a random forest model; (right) the confusion matrix obtained for the optimal random forest model (0—benign, 1—malware).

The third group is related to the experimental condition concerning the API calls sequences dataset represented with CV. The evaluation results are provided in Figure 4. The maximum validation accuracy of 95.56 percent was obtained for the random forest model with 32 trees (cf. also fourth column of Table 3).



**Figure 4.** API call sequences represented with CV: (left) a graphical representation of the validation accuracy with respect to the number of decision trees in a random forest model, (right) the confusion matrix obtained for the optimal random forest model (0—benign, 1—malware).

Finally, the fourth group is related to the experimental condition concerning the API calls sequence dataset represented with TF-IDF. The evaluation results are provided in Figure 5. The maximum validation accuracy of 95.4 percent was obtained for the random forest model with 66 trees (cf. also the fifth column of Table 3).



**Figure 5.** API call sequences represented with TF-IDF: (left) a graphical representation of the validation accuracy with respect to the number of decision trees in a random forest model; (right) the confusion matrix obtained for the optimal random forest model (0—benign, 1—malware).

### 4.3. Computing Resources and Trade-Offs

The training and testing of the underlying models were performed on a Debian server VM, with resources of 16 virtual CPUs, approximately 900 GB of DDR5 memory, and a 2.5 TB virtual hard disk. During the feature extraction process (by applying the CV and TF-IDF techniques), the entire memory was used, which occasionally caused unexpected termination of the Python scripts until we performed the random subsampling of the malware part of the dataset to 70 percent of its initial size. The value of 70 percent was determined in a process of iterative subsampling in steps of 5 percent, aimed at determining the maximum dataset size, which will engage the entire memory but not cause an unexpected termination of the Python scripts.

The results of the further optimization process are shown in Table 4, where each row represents the group of 100 RF models, in which the hyperparameter representing the

number of trees (i.e.,  $n\_estimators$  parameter) takes values in the range [1, 100]. The columns are organized as follows:

- Random forest hyperparameters ( $max\_depth$ ,  $min\_samples\_split$ ,  $min\_samples\_leaf$ , and  $max\_features$ ) represent parameter values applied to each RF group of models in the optimization process.
- Computing resources (Exec. time, Max. RAM cons., Max. CPU cons.) describe the resources required to load the extracted features from the files stored on the disk (previously saved to the pickle file for both CV and TF-IDF) and to train and test the RF models.
- Results provide the maximum validation accuracy in the group and the number of trees in an RF model at which it was obtained.

**Table 4.** RF optimization and computing resources.

Datasets	Random Forest Hyperparameters				Computing Resources			Results	
	Max_depth	Min_samples_split	Min_samples_leaf	Max_features	Exec. Time (h)	Max. RAM Cons. (GB)	Max. CPU Cons. (%)	# Trees at Max. Acc.	Max. Validation Accuracy (%)
1	None	2	1	sqrt	7.00	19,3	6	89	99.067
2	100	2	1	sqrt	7.80	19.3	6.7	93	99.099
3	None	100	1	sqrt	6.25	17.8	6.78	25	98.874
4	None	2	100	sqrt	5.90	18	6.47	72	89.836
5	None	2	1	0.1	50.21	18.31	6.47	45	99.742
6	1000	2	1	sqrt	7.01	18.34	6.81	87	99.067
7	None	1000	1	sqrt	5.50	18	6.5	33	98.617
8	None	2	1000	sqrt	7.75	18	6.5	69	76.841
9	100	2	1	0.2	70.84	45	22	7	99.743
10	None	2	1	sqrt	6.11	20	7	86	99.035
11	100	2	1	sqrt	7.50	20	7	86	99.035
12	None	100	1	sqrt	5.34	18.8	7	79	98.617
13	None	2	100	sqrt	5.00	18	6.44	95	92.879
14	None	2	1	0.1	43.33	19	6.38	94	99.614
15	1000	2	1	sqrt	6.00	19.03	6.47	16	98.874
16	None	1000	1	sqrt	5.00	19.99	6.62	51	97.813
17	None	2	1000	sqrt	5.00	18.44	6,7	55	79.607
18	100	2	1	0.2	61.09	20.4	7	26	99.678
19	None	2	1	sqrt	0.06	18	6	33	95.529
20	100	2	1	sqrt	0.06	18	6	79	95.529
21	None	100	1	sqrt	0.06	18	6	9	94.854
22	None	2	100	sqrt	0.06	18	6	8	92.536
23	None	2	1	0.1	0.06	18	6	21	95.433
24	1000	2	1	sqrt	0.06	18	6	13	95.497
25	None	1000	1	sqrt	0.06	18	6	38	93.921
26	None	2	1000	sqrt	0.06	18	6	16	78.868
27	100	2	1	0.2	0.09	18	6	32	95.561
28	None	2	1	sqrt	0.07	18	6.5	43	95.24
29	100	2	1	sqrt	0.07	18	6.5	43	95.24
30	None	100	1	sqrt	0.05	18	6.5	22	94.757
31	None	2	100	sqrt	0.03	18	6.5	42	93.149
32	None	2	1	0.1	0.08	18	6.5	92	95.368
33	1000	2	1	sqrt	0.06	18	6.5	43	95.24
34	None	1000	1	sqrt	0.03	18	6.5	93	93.599
35	None	2	1000	sqrt	0.02	18	6.5	14	89.643
36	100	2	1	0.2	0.13	18	6.5	66	95.4

The values highlighted in red represent very long computation time, in blue the highest validation accuracy values, and in green the acceptable validation accuracy values with significantly less computational time.

The maximum accuracy values (cf. cells marked in blue in the last column) were obtained with the combination of a  $max\_depth$  value of 100 and  $max\_features$  of 0.2 but at a significant cost in terms of execution time (over 70 and 60 h for full reports dataset with CV and TF-IDF, respectively; cf. cells marked in light red). Additional experimental settings,

which obtained slightly lower but still satisfying validation accuracies at significantly lower execution times, are marked in green.

It can be observed that increasing the values of the *min\_samples\_split* and *min\_samples\_leaf* hyperparameters did not positively affect the validation accuracy. However, setting the hyperparameters *max\_depth* to 100 and *max\_features* (i.e., the number of the features considered at node split) to 0.1 and 0.2 (i.e., 10 and 20 percent of all features, respectively) increased the validation accuracy and computational time.

With the increase in the value of the *max\_features* hyperparameter, the number of trees at which the maximum validation accuracy was achieved decreased to only 7 for CV and 26 for TF-IDF. This indicates that, in practice, the number of trees in an RF model may be decreased to 30, which would significantly reduce the execution time required for the development of a model (i.e., from approximately 70 h to 20 h for CV and from approximately 61 h to 18 h for TF-IDF).

The RF models obtained from the API call dataset were less time demanding but also achieved lower validation accuracy. Such results are due to the significantly lower number of features considered in the API call dataset, i.e., 294 API-related features compared to over 25 million features present in the full report dataset.

## 5. Discussion

The main result given in Table 3 is that the RF models trained and evaluated on the full-featured datasets display higher validation accuracy than the RF models trained and evaluated on the API call dataset. This is in line with the starting assumption that the discriminative power of the full-featured sandbox reports discussed is greater than the discriminative power of just API call sequences.

Finally, the validation accuracy achieved on datasets containing CV features is slightly higher than the validation accuracy achieved on datasets containing TF-IDF features. This difference is not necessarily significant, but a possible underlying reason may be that the raw count of occurrences of tokens in CV is more indicative than the normalized values in TF-IDF.

The full-featured dataset contains samples that induce API calls, as well as samples that do not induce API calls. The validation accuracy obtained on this dataset is 99.74 percent. On the other hand, the API call dataset contains only samples that induce API calls, and the validation accuracy obtained on this dataset is 95.56 percent. Thus, we demonstrated that considering a more comprehensive feature set (i.e., going beyond API calls) improves the classification accuracy of malware and benign samples (i.e., the samples that do not induce API calls were classified with high accuracy).

The question that may arise here is why the second dataset contains only samples that induce API calls. First, if we also considered samples that do not induce API calls in the second dataset (i.e., if we added 6.13 percent of malware samples and 39.52 percent of benign samples), the accuracy result in the second case would be lower. However, this would not affect our conclusion: the validation accuracy obtained in the first case would still be higher than in the second case. Second, our samples were selected randomly to avoid subjective bias. Still, we do not claim that the share of samples that do not induce API calls is representative of a real-life distribution. Thus, if we included all samples in the second dataset, another question could arise: has the validation accuracy of models based only on API calls been artificially decreased by increasing the share of samples that do not induce API calls? To prevent this, we decided to evaluate these models on a dataset containing only samples that induce API calls, which maximizes the obtained accuracy. It is important to note that even in this “maximized” condition, the accuracy obtained when considering API calls is only lower than the accuracy obtained by the more comprehensive feature set—in line with our starting hypothesis.

## 6. Conclusions

In the research community, API call sequences are widely acknowledged as discriminative features in the context of dynamic malware analysis. However, a significant number

of benign samples do not use operating system API calls (e.g., reading of documents, etc.), and some malware instances are implemented in a way that does not leave any trace with respect to API calls. Only this observation would be enough to justify the extension of a feature set beyond API calls.

Still, even if we consider only binary files whose execution induces API calls, in this study, we demonstrated that extending the feature set beyond API calls may improve the malware detection performance. The accuracy of approximately 95.56 percent obtained for API call sequences was increased to 99.74 percent when full-featured sandbox reports were considered. This improvement comes with a cost: the number of features and the computational time for random forest models are significantly increased (cf. Table 4).

However, we believe that the significance of improvement justifies these costs. As part of future work, we intend to train and evaluate recurrent neural network-based models on the reported dataset and perform additional analyses of the features not related to API calls.

**Author Contributions:** Conceptualization, S.I. and M.G.; methodology, S.I. and M.G.; software, S.I.; validation, N.M., I.T., B.J. and M.G.B.; formal analysis, S.I. and M.G.; investigation, S.I.; resources, S.I.; data curation, S.I.; writing—original draft preparation, S.I. and M.G.; writing—review and editing, M.G., N.M., I.T., B.J. and M.G.B.; visualization, S.I.; supervision, M.G.; project administration, S.I. All authors have read and agreed to the published version of the manuscript.

**Funding:** The authors received no specific funding for this study.

**Data Availability Statement:** The reported dataset is available from authors upon request.

**Acknowledgments:** The company Tehnika KB doo (Belgrade, Serbia) contributed to this work with an archive of real-life benign files as a source for sandbox reports. Their support is gratefully acknowledged and highly appreciated.

**Conflicts of Interest:** The authors declare that they have no conflicts of interest to report regarding the presented work.

## References

1. Ficco, M. Malware Analysis by Combining Multiple Detectors and Observation Windows. *IEEE Trans. Comput.* **2022**, *71*, 1276–1290. [[CrossRef](#)]
2. Ho, T.K. Random decision forests. In Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, Canada, 14–16 August 1995; Volume 1, pp. 278–282. [[CrossRef](#)]
3. Mira, F. A Review Paper of Malware Detection Using API Call Sequences. In Proceedings of the 2019 2nd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh, Saudi Arabia, 1–3 May 2019; pp. 1–6. [[CrossRef](#)]
4. Deore, M.; Tarambale, M.; Ratna Raja Kumar, J.; Sakhare, S. GRASE: Granulometry Analysis with Semi Eager Classifier to Detect Malware. *Int. J. Interact. Multimed. Artif. Intell.* **2024**, *8*, 120–134. [[CrossRef](#)]
5. Düzgün, B.; Çayır, A.; Demirkıran, F.; Kahya, C.N.; Gençaydın, B.; Dağ, H. Benchmark Static API Call Datasets for Malware Family Classification. *arXiv* **2022**. [[CrossRef](#)]
6. Alsharni, A.; Alliheedi, M.A. Enhancing Malware Detection by Integrating Machine Learning with Cuckoo Sandbox. *arXiv* **2023**, arXiv:2311.04372. [[CrossRef](#)]
7. Syeda, D.; Asghar, M. Dynamic Malware Classification and API Categorization of Windows Portable Executable Files Using Machine Learning. *Appl. Sci.* **2024**, *14*, 1015. [[CrossRef](#)]
8. Zhang, S.; Wu, J.; Zhang, M.; Yang, W. Dynamic Malware Analysis Based on API Sequence Semantic Fusion. *Applied Sciences* **2023**, *13*, 6526. [[CrossRef](#)]
9. Huang, Y.; Chen, T.; Hsiao, S. Learning Dynamic Malware Representation from Common Behavior. *J. Inf. Sci. Eng.* **2022**, *38*, 1317–1334. [[CrossRef](#)]
10. Huang, Y.; Sun, Y.; Chen, M. TagSeq: Malicious behavior discovery using dynamic analysis. *PLoS ONE* **2022**, *17*, e0263644. [[CrossRef](#)]
11. Chen, L.; Yagemann, C.; Downing, E. To believe or not to believe: Validating explanation fidelity for dynamic malware analysis. *arXiv* **2019**, arXiv:1905.00122. [[CrossRef](#)]
12. Alhashmi, A.; Darem, A.; Alanazi, M.; Alashjaee, M.; Aldughayfiq, B.; Ghaleb, A.; Ebad, A.; Alanazi, A. Hybrid Malware Variant Detection Model with Extreme Gradient Boosting and Artificial Neural Network Classifiers. *Comput. Mater. Contin.* **2023**, *76*, 3483–3498. [[CrossRef](#)]
13. Lee, D.; Jeon, G.; Lee, S.; Cho, H. Deobfuscating Mobile Malware for Identifying Concealed Behaviors. *Comput. Mater. Contin.* **2022**, *72*, 5909–5923. [[CrossRef](#)]

14. Chen, T.; Zeng, H.; Lv, M.; Zhu, T. CTIMD: Cyber threat intelligence enhanced malware detection using API call sequences with parameters. *Comput. Secur.* **2024**, *136*, 103518. [[CrossRef](#)]
15. Yau, L.; Lam, Y.; Lokesh, A.; Gupta, P.; Lim, J.; Singh, I.; Loo, J.; Ngo, M.; Teo, S.; Truong-Huu, T. A Novel Feature Vector for AI-Assisted Windows Malware Detection. In Proceedings of the 2023 IEEE International Conference on Dependable, Autonomic and Secure Computing, International Conference on Pervasive Intelligence and Computing, International Conference on Cloud and Big Data Computing, International Conference on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech), Abu Dhabi, United Arab Emirates, 14–17 November 2023; pp. 0355–0361. [[CrossRef](#)]
16. Xu, Y.; Chen, Z. Family Classification based on Tree Representations for Malware. In Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems, Seoul, Republic of Korea, 24–25 August 2023; pp. 65–71. [[CrossRef](#)]
17. Li, C.; Cheng, C.; Zhu, H.; Wang, L.; Lv, Q.; Wang, Y.; Li, N.; Sun, D. DMalNet: Dynamic malware analysis based on API feature engineering and graph learning. *Comput. Secur.* **2022**, *122*, 102872. [[CrossRef](#)]
18. Li, S.; Wen, H.; Deng, L.; Zhou, Y.; Zhang, W.; Li, Z.; Sun, L. Denoising Network of Dynamic Features for Enhanced Malware Classification. In Proceedings of the 2023 IEEE International Performance, Computing, and Communications Conference (IPCCC), Anaheim, CA, USA, 17–19 November 2023; pp. 32–39. [[CrossRef](#)]
19. Nunes, M.; Burnap, P.; Rana, O.; Reinecke, P.; Lloyd, K. Getting to the root of the problem: A detailed comparison of kernel and user level data for dynamic malware analysis. *J. Inf. Secur. Appl.* **2019**, *48*, 102365. [[CrossRef](#)]
20. Li, N.; Lu, Z.; Ma, Y.; Chen, Y.; Dong, J. A Malicious Program Behavior Detection Model Based on API Call Sequences. *Electronics* **2024**, *13*, 1092. [[CrossRef](#)]
21. Jindal, C.; Salls, C.; Aghakhani, H.; Long, K.; Kruegel, C.; Vigna, G. Neurlux: Dynamic Malware Analysis Without Feature Engineering. *arXiv* **2019**, arXiv:1910.11376. [[CrossRef](#)]
22. Anderson, H.; Rothl, P. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *arXiv* **2018**, arXiv:1804.04637. [[CrossRef](#)]
23. Bosansky, B.; Kouba, D.; Manhal, O.; Sick, T.; Lisy, V.; Kroustek, J.; Somol, P. Avast-CTU Public CAPE Dataset. *arXiv* **2022**, arXiv:2209.03188. [[CrossRef](#)]
24. Herrera-Silva, J.; Hernández-Álvarez, M. Dynamic Feature Dataset for Ransomware Detection Using Machine Learning Algorithms. *Sensors* **2023**, *23*, 1053. [[CrossRef](#)]
25. Irshad, A.; Dutta, M. Identification of Windows-Based Malware by Dynamic Analysis Using Machine Learning Algorithm. In *Advances in Computational Intelligence and Communication Technology*; Gao, X.-Z., Tiwari, S., Trivedi, M.C., Mishra, K.K., Eds.; Springer: Singapore, 2021; Volume 1086, pp. 207–218. [[CrossRef](#)]
26. Sraw, J.; Kumar, K. Using Static and Dynamic Malware features to perform Malware Ascription. *ECS Trans.* **2022**, *107*, 3187–3198. [[CrossRef](#)]
27. Sethi, K.; Kumar, R.; Sethi, L.; Bera, P.; Patra, P. A Novel Machine Learning Based Malware Detection and Classification Framework. In Proceedings of the 2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security), Oxford, UK, 3–4 June 2019; pp. 1–4. [[CrossRef](#)]
28. Virus Total. Virustotal-Free Online Virus, Malware and Url Scanner. Available online: <https://www.virustotal.com/en> (accessed on 9 April 2024).
29. Taheri, R.; Javidan, R.; Shojafar MP, V.; Conti, M. Can Machine Learning Model with Static Features be Fooled: An Adversarial Machine Learning Approach. *arXiv* **2020**, arXiv:1904.09433. [[CrossRef](#)]
30. Taheri, R.; Ghahramani, M.; Javidan, R.; Shojafar, M.; Pooranian, Z.; Conti, M. Similarity-based Android malware detection using Hamming distance of static binary features. *Future Gener. Comput. Syst.* **2020**, *105*, 230–247. [[CrossRef](#)]
31. Ilić, S.; Gnjatović, M.; Popović, B.; Maček, N. A pilot comparative analysis of the Cuckoo and Drakvuf sandboxes: An end-user perspective. *Military Tech. Cour.* **2022**, *70*, 372–392. [[CrossRef](#)]
32. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830. Available online: <http://jmlr.org/papers/v12/pedregosa11a.html> (accessed on 3 September 2024).
33. McKinney, W. Data structures for statistical computing in python. In Proceedings of the 9th Python in Science Conference, Austin, TX, USA, 28 June–3 July 2010; PANDAS Conference Paper. Volume 445, pp. 51–56.

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.