

10-8-2023

## MalFe—Malware Feature Engineering Generation Platform

Avinash Singh  
*University of Pretoria*

Richard Adeyemi Ikuesan  
*Zayed University, richard.ikuesan@zu.ac.ae*

Hein Venter  
*University of Pretoria*

Follow this and additional works at: <https://zuscholars.zu.ac.ae/works>



Part of the [Computer Sciences Commons](#)

---

### Recommended Citation

Singh, Avinash; Ikuesan, Richard Adeyemi; and Venter, Hein, "MalFe—Malware Feature Engineering Generation Platform" (2023). *All Works*. 6165.  
<https://zuscholars.zu.ac.ae/works/6165>

This Article is brought to you for free and open access by ZU Scholars. It has been accepted for inclusion in All Works by an authorized administrator of ZU Scholars. For more information, please contact [scholars@zu.ac.ae](mailto:scholars@zu.ac.ae).

Article

# MalFe—Malware Feature Engineering Generation Platform

Avinash Singh <sup>1,\*</sup> , Richard Adeyemi Ikuesan <sup>2</sup>  and Hein Venter <sup>1</sup> 

<sup>1</sup> Department of Computer Science, University of Pretoria, Pretoria 0002, South Africa; hventer@cs.up.ac.za

<sup>2</sup> College of Interdisciplinary Studies, Zayed University, Abu Dhabi P.O. Box 144534, United Arab Emirates; richard.ikuesan@zu.ac.ae

\* Correspondence: asingh@cs.up.ac.za

**Abstract:** The growing sophistication of malware has resulted in diverse challenges, especially among security researchers who are expected to develop mechanisms to thwart these malicious attacks. While security researchers have turned to machine learning to combat this surge in malware attacks and enhance detection and prevention methods, they often encounter limitations when it comes to sourcing malware binaries. This limitation places the burden on malware researchers to create context-specific datasets and detection mechanisms, a time-consuming and intricate process that involves a series of experiments. The lack of accessible analysis reports and a centralized platform for sharing and verifying findings has resulted in many research outputs that can neither be replicated nor validated. To address this critical gap, a malware analysis data curation platform was developed. This platform offers malware researchers a highly customizable feature generation process drawing from analysis data reports, particularly those generated in sandbox-based environments such as Cuckoo Sandbox. To evaluate the effectiveness of the platform, a replication of existing studies was conducted in the form of case studies. These studies revealed that the developed platform offers an effective approach that can aid malware detection research. Moreover, a real-world scenario involving over 3000 ransomware and benign samples for ransomware detection based on PE entropy was explored. This yielded an impressive accuracy score of 98.8% and an AUC of 0.97 when employing the decision tree algorithm, with a low latency of 1.51 ms. These results emphasize the necessity of the proposed platform while demonstrating its capacity to construct a comprehensive detection mechanism. By fostering community-driven interactive databanks, this platform enables the creation of datasets as well as the sharing of reports, both of which can substantially reduce experimentation time and enhance research repeatability.



check for updates

**Citation:** Singh, A.; Ikuesan, R.A.; Venter, H. MalFe—Malware Feature Engineering Generation Platform. *Computers* **2023**, *12*, 201. <https://doi.org/10.3390/computers12100201>

Academic Editors: Ömer Aslan and Refik Samet

Received: 15 August 2023

Revised: 3 October 2023

Accepted: 5 October 2023

Published: 8 October 2023



**Copyright:** © 2023 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/>).

**Keywords:** malware; malware feature engineering; malware datasets; malware detection; machine learning; artificial intelligence

## 1. Introduction

The escalating prevalence of cyberattacks presents a daunting and unrelenting challenge to organizations worldwide. This problem is further compounded by the exponential increase in the volume of malware [1]. According to AV-Test, cybersecurity researchers identified and catalogued over one billion new malware samples in 2022 alone [1]. Cyberattacks are often orchestrated by malicious actors with varying degrees of sophistication, and cast a long shadow over businesses and institutions. Not only do they jeopardize sensitive data and critical infrastructure, but they also disrupt regular business operations, potentially leading to reputational damage and substantial financial losses [2]. This growing threat landscape requires constant vigilance and innovative strategies to defend against malware attacks [3–5]. In this context, security researchers are tasked with creating robust detection and prevention mechanisms. These require solutions with the formidable capability of staying one step ahead of cybercriminals [6,7]. This imperative has spurred an environment of constant innovation and adaptation, pushing security researchers to think creatively and harness cutting-edge technologies to safeguard digital assets. Among the arsenal of

tools and approaches at their disposal, artificial intelligence, particularly machine learning, has emerged as a long-term solution to combat malware threats. With the ability of machine learning to discern intricate patterns and anomalies within vast datasets, security researchers have made significant strides in detecting and deterring a multitude of malware attacks [8–10]. The effectiveness of machine learning in identifying new and previously unseen attack vectors presents a promising and forward-looking approach to cyberdefense.

While machine learning holds great promise, its effectiveness is intricately tied to the quality and quantity of the data on which models are trained [4,11]. High-quality data are characterized by cleanliness and completeness along with a lack of missing or corrupted information. The integrity of the data is paramount, as even minor discrepancies or inaccuracies can propagate through the learning process, leading to skewed models and erroneous results. Moreover, the distribution and balance of data across different classes or categories are of critical importance. For instance, when training a machine learning model to distinguish between benign and malicious software, having a balanced dataset with a similar number of instances from each class is vital [12]. Imbalances can lead to biased models that overemphasize the majority class, potentially resulting in misleadingly high accuracy scores. Security researchers often focus on specific attributes or features that characterize malware or malicious behaviour. This attribute-based approach requires the execution of numerous malware samples within a controlled environment, often referred to as a sandbox, in order to capture the relevant data needed for machine learning [4,5,13]. While this process is essential for feature engineering, it is undeniably labour-intensive and susceptible to errors, potentially impacting the reliability and accuracy of the acquired data. A substantial portion of the raw data generated during these experiments often remains underutilized, and in most cases is undisclosed. This underutilization stems from the absence of platforms that can effectively manage, store, and make such data accessible to the broader research community, resulting in a considerable loss of valuable insights and research potential.

To address these challenges and bridge the gap between data availability and research needs, in this paper, we introduce a malware analysis data curation platform. This platform is designed to empower security researchers by offering a highly customizable feature generation process sourced from analysis data reports, particularly those generated in sandboxed environments such as Cuckoo Sandbox [14]. The platform aims to streamline the labour-intensive data acquisition process while providing a centralized repository for malware analysis data. In this way, it enables researchers to more efficiently build datasets, contribute reports, and collaborate with peers. Through community-driven interactive databanks, the platform facilitates a collective effort to reduce experimentation time and enhance the repeatability and verifiability of research outcomes.

The remainder of this paper is structured as follows. First, the background necessary to better understand the problem at hand is discussed. Thereafter, the proposed platform is presented and further discussed using a high-level process model. This is followed by an evaluation of the platform, which is achieved through the NIST validation cycle [15] while abiding by software engineering principles. To further highlight the usefulness of the platform, three case studies that involved malware detection with Cuckoo Sandbox reports are assessed to illustrate how the platform can aid researchers. Another method used to show the effectiveness of the platform is the reproduction of the identified case studies using the platform, followed by a comparison of the results. Furthermore, a literature search is conducted to determine the extent to which the platform can aid malware detection research, and a comparative analysis is conducted to determine whether any similar platforms currently exist and how the proposed platform compares to them. Finally, a real-world scenario is then used to illustrate how the platform can foster new novel research approaches using PE entropy for ransomware detection.

## 2. Background

The process of constructing concrete and reliable datasets for machine learning applications is a formidable undertaking [5,16]. Crafting a dataset that stands as a dependable foundation for robust machine learning models requires a comprehensive understanding of the data, mastery of data processing techniques, and the ability to interpret the data and extract meaningful insights. Fundamentally, the quality and relevance of a dataset hinges on its alignment with the intended purpose, making clarity of objectives a crucial starting point. The journey to create such datasets comprises numerous methodologies, each tailored to specific objectives, and often relies on meticulous domain expertise.

In the realm of machine learning, the availability of comprehensive datasets is paramount. These datasets serve as the fuel that powers the algorithms, enabling them to learn, generalize, and make predictions. The vast troves of data encountered in modern applications necessitate sophisticated approaches to processing and analysis. This is where machine learning algorithms come into play, providing the capacity to efficiently handle and derive valuable insights from large and complex datasets [9]. Machine learning algorithms can be broadly categorized into two main groups: supervised and unsupervised learning. In supervised learning, algorithms are trained on labelled datasets containing both positive and negative examples. This training equips them to recognize the distinguishing characteristics of positive and negative instances, making the resulting models suitable for classification tasks. On the other hand, unsupervised learning empowers algorithms to autonomously uncover patterns and relationships within data, effectively allowing the data to define what constitutes positive and negative instances. While unsupervised learning holds value in various contexts, it is generally discouraged for classification tasks due to the potential introduction of a bias, which can significantly impact outcomes.

Traditional techniques for malware detection have long revolved around static analysis [17–19]. Static analysis involves the examination of an executable or information prior to its execution, making it an invaluable tool to quickly identify malicious entities through signature-based pattern matching and key identifiers such as fingerprinting. However, the efficacy of static analysis is limited by the evolving tactics of modern malware, which often employ obfuscation techniques and embedding to evade detection [20,21]. Dynamic analysis involves the execution of malware, and has emerged as the most accurate method for detecting malware; yet, this approach comes with inherent risks, as executing malware can potentially inflict irreparable harm on a system [22]. To mitigate this risk, sandboxed environments have become essential in dynamic analysis. These controlled environments ensure that the host system remains unharmed, and snapshots can be employed to revert to a previous state after analysis. However, dynamic analysis generates copious amounts of data, rendering manual analysis impractical and inefficient [5,23,24].

This is precisely where machine learning emerges as a critical catalyst for progress. Machine learning algorithms possess the ability to sift through vast datasets, discern hidden patterns, and identify correlations that elude human analysts. By leveraging machine learning, the risk of human error is mitigated and the time required to formulate effective defences against malware is significantly reduced. However, a pivotal challenge arises in obtaining data, as malware binary samples must be sourced, executed, and analyzed before the requisite data become available for machine learning algorithms.

In previous work by Singh et al. [5], machine learning techniques were applied to a dataset comprising ransomware and benign samples in order to detect ransomware attacks using process memory. The dataset comprised 354 benign and 117 ransomware samples, producing a dataset of 937 samples after analysis. Building a balanced dataset was a challenging task, and resulted in laborious execution of these samples to extract process memory. Upon using the various memory access privileges, ransomware was detected with 96.28% accuracy with the XGBoost algorithm. In work by Ashraf et al. [13], a method was proposed to detect ransomware using transfer learning based on deep convolutional neural networks. Their approach tackled both static and dynamic analysis, using tools to extract the static information and Cuckoo Sandbox to extract the dynamic features. The dataset

was sourced from 3646 samples for the static part and 3444 samples for the dynamic part. As there were two datasets, the obtained results were 98% for static features using random forest and 92% for dynamic features using support vector machine. Using the transfer learning technique, an accuracy score of 94% was achieved from 300 features. However, no raw data were available for further validation of the results additional research.

The research by Faruki et al. [23] detected malware using API call-grams. Using Cuckoo Sandbox, 2510 benign and 2487 malicious samples were analysed to build their dataset. Faruki et al. [23] identified that the API call-grams generated an imbalanced dataset, with an induced bias towards API calls for certain classes. Using the voted perceptron algorithm from Rosenblatt and Frank [25], an accuracy score of 98.7% was achieved. Hansen et al. [26] evaluated behavioural traces of over 270,000 malware and 837 benign samples. Using Cuckoo Sandbox, the samples were analysed, and feature selection based on the information entropy and information gain ratio was used to obtain the feature set. For malware detection, Hansen et al. [26] decided to perform a dataset reduction of 88%, using only 1650 malware samples. Random forest was used to train the machine learning model on this reduced dataset, resulting in a weighted average score of 0.981. This dataset was heavily imbalanced; thus, dataset reduction can ensure that there is no over-representation of datapoints. Darshan et al. [27] used Cuckoo Sandbox for malware detection by the impact of exploring n-grams on the behaviour of the sample. With 3000 benign and 3000 malicious samples, and using an n-gram length of 3 and the Pegasos [28] algorithm, they obtained an accuracy of 90.03%. Poudyal and Dasgupta [29] created a ransomware detection framework that utilized Cuckoo Sandbox and Ghidra to perform detection using behaviour such as DLL and API calls, then performed NLP on the assembly code extracted from the samples. Using 550 ransomware and 540 benign samples, Poudyal and Dasgupta [29] achieved 99.54% accuracy using SVM and Adaboost with J48.

A common aspect across all of these research papers is that both static and dynamic analysis are needed for malware detection [5,13,23,26,27,29]. Each paper utilized Cuckoo Sandbox to obtain the raw data from the sample execution. These raw data were then processed using the necessary information (algorithm) based on the different approaches that the researchers developed. The extracted information was then used to build a robust dataset. The process of obtaining raw data is laborious, as it requires the execution of each sample. This makes the efforts of individual research teams unnecessarily complex, as there is no platform available to obtain raw Cuckoo reports. While platforms such as Kaggle [30], DataRobot [31], and Alteryx [32] exist, they depend solely on the processed dataset uploaded by the author and the willingness of authors to upload the code used to build machine learning models from the dataset. Considering that Kaggle is a general-purpose dataset platform, there is no way for raw data to be accessed and reused. This makes it difficult to evaluate the data quality as well as the information presented in the dataset. With DataRobot, both feature extraction and further processing of the data in a dataset are possible; however, it is limited to data uploaded for a project, and cannot be used for general purposes. Similarly, Alteryx [32] allows for the aggregation of data pipelines from multiple sources. However, these data sources are defined by the end user, and are not available 'out of the box' for the user base on the platform. Therefore, it cannot aid in promoting experimental repeatability.

To bridge this crucial gap, in this research we develop a novel solution in the form of a platform designed to facilitate the generation of datasets from Cuckoo Sandbox reports. By simplifying the process of dataset generation and compilation, the proposed platform empowers security researchers and data scientists to construct more accurate and effective machine learning models for malware detection. In the following sections, the intricacies of this platform are discussed, including its architectural composition, functionalities, capabilities, and corresponding pivotal role in fortifying the collective defense against the relentless tide of malware threats.

### 3. MalFe Platform

The Malware Feature Engineering (MalFe) platform aims to simplify the difficulty of obtaining and creating custom datasets from a databank through a community-driven platform. To ensure a scientific process, the design approach leverages software engineering principles to develop the processes and features of the platform. On a high level, MalFe contains five key components: Reports, Categories, User Code, Dataset Generator, and Datasets, as depicted in Figure 1. The reports consist of raw Cuckoo Sandbox reports which, are uploaded by the community from experiments run to analyze malware samples. The categories help with organizing these reports, allowing more targeted datasets to be built. User code is used to parse through this data repository of reports and extract the relevant information required to generate a dataset. The user code is then fed into a dataset generator, which runs on a queue basis and executes the user code, based on the selected categories of reports, resulting in a dataset being generated. After being approved by their creator(s), these datasets are then made available for public viewing.

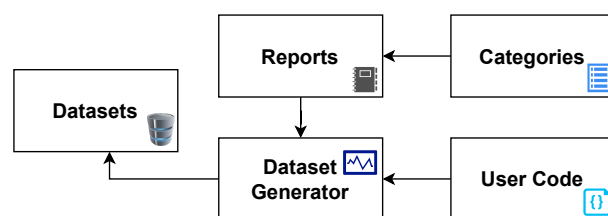
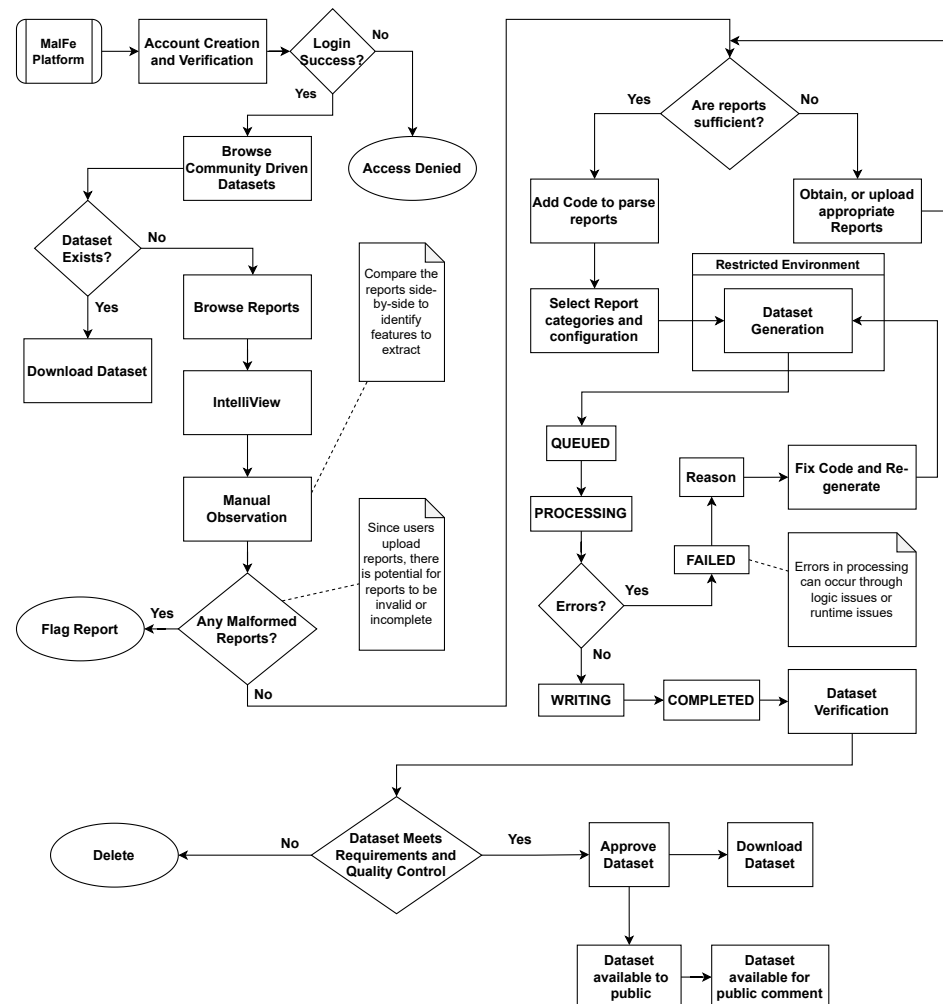


Figure 1. MalFe high-level model.

The MalFe process model is shown in Figure 2; in order to prevent misuse and to grow trust on the platform, this process starts with account creation and verification. After successful authentication, a user is able to browse the existing datasets that have been generated by the community. Thereafter, a user can search for a dataset that matches their need. In the event that the dataset does not exist on the platform yet, the user has the option to browse the uploaded Cuckoo Sandbox reports to see whether the information needed to build the dataset is available. If the information is not sufficient, the user needs to source more reports, either through experimentation or by obtaining them from an external source. Because such reports can be rather large, it is both difficult and typically not feasible to download all the reports in order to investigate the data.

To solve this problem, a lightweight comparison tool called IntelliView was created and added to the platform. It allows end users to browse sections of reports, making it easier to browse and compare reports. After manual observation conducted through IntelliView, if the user finds any malformed or erroneous reports they can flag these reports, ensuring that high-quality data are always available to the community and further enhancing the trust in the platform. At this stage, the reports are sufficient, and quality control has been conducted. The next step is for the user to add code to allow these reports to be parsed in order to extract the information needed to build the dataset. When this is complete, the metadata needed for the platform to build the dataset are requested from the user. This includes the category of reports for the dataset to be built off, as well as the name and description of the dataset. After the user code is checked for security vulnerabilities and syntax errors, the code is successfully queued on the platform to generate the dataset in a restricted environment. The process of dataset generation involves several status changes, keeping the user aware of what is happening. If the status is set to QUEUED, this means that the dataset has been added to a queue; the next generated dataset is taken from the queue when the previous dataset has been completed. This then sets the dataset generation status to PROCESSING. The processing stage of dataset generation is the core functionality of the platform, providing users with the ability to design the data according to their specific requirements. Error handling is integrated into the platform as well, providing an opportunity to understand any misconfiguration or other sources of errors during the data generation process. After dataset generation is completed, the data are written to

a CSV file and saved on the platform, and the status is set to COMPLETE. The owner is then prompted to verify the dataset and perform quality checking. When approved by the owner, the dataset is made publicly available. Finally, the dataset is available to be downloaded, shared, and even commented on for further research and networking with security professionals. The next section discusses the technical details behind the implementation of the platform.



**Figure 2.** MalFe process model.

### MalFe Implementation

The platform was developed following a modular approach and in line with agile principles to ensure adaptability and flexibility. Python, renowned for its versatility and wealth of built-in frameworks and libraries, was the language of choice for the implementation. To facilitate seamless web-based management, the Django web framework was chosen, which is a robust and widely respected choice in the field [33]. Security remains a paramount concern, and the platform integrates multiple layers of defense. Django's default security middleware forms the first line of protection, followed by custom sanitization middleware. This custom middleware diligently scrubs metadata of any special characters and tags, further fortifying the security posture of the platform. The critical middleware components in play are SecurityMiddleware, SessionMiddleware, CsrfViewMiddleware, AuthenticationMiddleware, XFrameOptionsMiddleware, OTPMiddleware, and SessionSecurityMiddleware. In the realm of authentication, Two-Factor Authentication (2FA) was implemented to enhance security. This robust 2FA mechanism is mandatory for logging in and is integrated seamlessly into the user experience. Users have the convenience of utiliz-

ing either a token generator or physical keys. Token generators leverage the Time-Based One-Time Pin (TOTP) algorithm [34], producing 6–8 unique digits based on the current time and a secret key established during account registration. As a security measure, these tokens regenerate every 30 s, thwarting brute force and phishing attacks. Additionally, backup tokens are accessible in case the primary 2FA devices are unavailable, ensuring both recovery and security. The default Django password field was used, which is fortified with PBKDF2 utilizing robust SHA256-bit hashing and a random salt [33]. Furthermore, email verification is enforced, fostering trust and ensuring the integrity of user accounts.

The process of adding a Cuckoo report to the platform is simplified, as the report undergoes rigorous scrutiny, with essential metadata such as the SHA256 hash, name, and description being extracted and validated. A further layer of scrutiny involves an assessment for any analysis errors before the report is deemed suitable for upload, ensuring the preservation of raw data integrity. For added convenience, the platform seamlessly integrates with Cuckoo Sandbox, enabling users to upload samples directly. These samples are then enqueued for execution within the sandbox environment, and the report is subsequently automatically integrated into the platform. The IntelliView feature represents a unique and innovative approach to manual observation, and was achieved through the implementation of asynchronous calls, with data being clipped to enhance performance and usability.

The way datasets are created is a novel feature in which a user provides the code to process an individual report, thereby extracting the data and features needed for building the dataset. A portion of the code responsible for executing user-generated code can be seen in Figure 3.

```
def process_user_code(self, user_code, user_func, *args, **kwargs):
    def apply(f, *a, **kw):
        return f(*a, **kw)

    try:
        restricted_locals = {
            "result": None,
            "args": args,
            "kwargs": kwargs,
        }

        builtins_extra = {**limited_builtins, **utility_builtins, **safe_builtins,
                          **safe_globals}

        restricted_globals = {
            "__builtins__": builtins_extra,
            "getitem": default_guarded_getitem,
            "apply": apply,
            'log': self.log,
            'getiter': default_guarded_getiter,
            "write": full_write_guard,
            "getattr": getattr,
        }

        # Add another line to user code that executes @user_func
        user_code += "\nresult = {0}(*args, **kwargs)".format(user_func)

        # Compile the user code
        byte_code = compile_restricted(
            user_code, filename="<user_code>", mode="exec")

        # Run it
        exec(byte_code, restricted_globals, restricted_locals)

        # User code has modified result inside restricted_locals. Return it.
        return restricted_locals["result"]
    except SyntaxError as e:
        # Do whatever you want if the user has code that does not compile
        raise
    except Exception as e:
        # The code did something that is not allowed.
        raise
```

Figure 3. User code execution snippet.

This is achieved by users implementing a stub function that is fed the Cuckoo report, filename, SHA256 hash, and category, as well as whether or not it is a malicious sample. Because the platform does not dictate the features, a user can have as many features as they require. Furthermore, because executing user code is a security risk, the code is first checked for syntactical errors using an abstract syntax tree. RestrictedPython [35], which



acts as a Python environment with limited imports and functions, is used to execute the user code. Safe built-ins and safe globals are used to ensure additional security and usability. To improve speed, concurrency is utilized with thread pools, allowing faster processing of Cuckoo reports and feature generation. A cron job is used to create the datasets, resulting in a queue system that limits potential resource abuse. Furthermore, each dataset has a maximum execution time.

The next section evaluates the platform with software engineering principles and an adaptation of the NIST Computer Forensics Tool Testing (CFTT) program [15], then presents a comparative analysis with existing malware analysis research.

#### 4. MalFe Evaluation

When evaluating the effectiveness of a platform's implementation, the requirements, specifications, and usability are the core fundamentals of any good software system. Therefore, these are considered in our evaluation. We followed the testing processes of the NIST Computer Forensics Tool Testing (CFTT) program [15]. Thereafter, case studies of existing research were explored to demonstrate the potential usefulness of the platform. Finally, a literature search was conducted to illustrate the impact that the platform can have on the research community.

##### 4.1. MalFe Validation

The NIST validation cycle is shown in Figure 4. The first step is to define requirements for the platform, followed by test assertions defining the behaviour of how the system should act. Next, the test cases which test the requirements are defined, and verification is performed to check whether the test assertions have been achieved. Finally, test methodology and validation are reached. The system requirements consist of two categories: Core Requirements (CR), which are listed in Table 1, and Optional Requirements (OR), which can be found in Table 2. For example, in Table 1, the label column provides a reference number that is used in the compliance matrix. The description, on the other hand, provides the requirements for the platform.

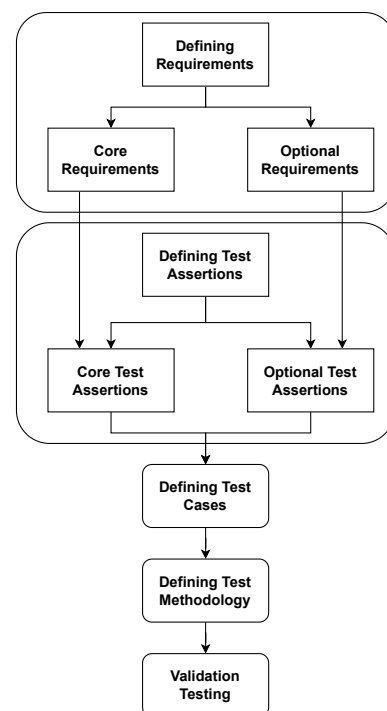


Figure 4. NIST validation cycle [15].

**Table 1.** Core requirements.

Label	Description
CR-01	The system should allow for secure access through accounts.
CR-02	Accounts should be verified and use 2-factor authentication for additional security.
CR-03	Users should be able to upload Cuckoo Reports.
CR-04	Duplicate Cuckoo Reports and Datasets should not exist.
CR-05	Users should be able to see the reports on the platform.
CR-06	Users can choose to upload private reports.
CR-07	Users can create datasets on the platform.
CR-08	Users can choose to create private datasets.
CR-09	Users should easily be able to select the report categories of reports for dataset generation.
CR-10	Users should be able to view datasets on the platform.
CR-11	User code should be checked for security vulnerabilities.
CR-12	User code should be executed in a restricted environment.
CR-13	User code should be syntax-checked to minimize runtime errors.
CR-14	Users should be able to download datasets.
CR-15	Users should be able to view a sample of the dataset.

**Table 2.** Optional requirements.

Label	Description
OR-01	Users should be able to see which datasets are used in publications.
OR-02	Users should be able to comment on datasets.
OR-03	Users should be able to see the status of dataset generation.
OR-04	Users should be informed why dataset generation failed and allowed to fix issues.
OR-05	Users should be able to flag reports for quality control.
OR-06	Users should be able to download reports and view them on Virus Total.
OR-07	Users should be able to search for reports and filter by category.
OR-08	Users should be able to see the reports used in a dataset.
OR-09	Users should be able to create versions of a dataset with different report categories.
OR-10	Users can turn off comments on their dataset.

The next step is to define the Test Assertions (TA), which are the post-conditions of a system's function. The assertions can be seen in Table 3. Thereafter, Test Cases (TC) are defined in order to ensure that the system has both met the standards and satisfied the test assertions. The test cases can be seen in Table 4.

**Table 3.** Test assertions.

Label	Description
TA-01	Restricted and secure access. Justification: To preserve trust and integrity on the platform; only authorized parties who are verified should be able to use the platform.
TA-02	Hash digests of the reports and datasets should be computed. Justification: To maintain integrity and eliminate any duplicates from occurring.
TA-03	Metadata sanitization of user data. Justification: To ensure good security practices and prevent potential system attacks. This reduces the attack vectors from injection attacks like XSS, SQL injection, and parsing attacks.
TA-04	The platform shall detect syntactical errors with the user code and/or malicious code. Justification: To eliminate any errors as well as improve the reliability and security of the system.
TA-05	The platform shall log the activity of users. Justification: To ensure security and prevent abuse.

**Table 4.** Test cases.

Label	Description
TC-01	Create an account and perform the verification.
TC-02	Turn on 2-factor authentication and verify that it works as expected.
TC-03	View the reports on the platform.
TC-04	View the datasets on the platform.
TC-05	Upload a report on the platform.
TC-06	Upload an existing report on the platform.
TC-07	Create a dataset with syntax errors in the code and malicious code.
TC-08	Create a dataset without errors.
TC-09	Download a dataset.
TC-10	View the code of the dataset.
TC-11	View the reports used in a dataset.
TC-12	Comment on a dataset.
TC-13	Add a publication entry.
TC-14	View user profile.
TC-15	Search for reports with NotPetya.
TC-16	Flag a report.

A compliance matrix simply maps the requirements to those test cases satisfying the test assertions. For example, if a core test assertion was met, that test assertion is specified in the result column. However, if a manual check was performed, this is indicated with ‘-check-’, indicating that the check result is compliant. Note that test assertions and manual checks can occur simultaneously in the result column. In Table 5, the compliance matrix confirms that the results from the test assertions have been fulfilled.

**Table 5.** Compliance matrix.

No.	Requirement	Test Case	Result
1	CR-01	TC-01	TA-01
2	CR-02	TC-02	TA-01, -check-
3	CR-03	TC-03, TC-05	TA-03, TA-05, -check-
4	CR-04	TC-06	TA-02
5	CR-05	TC-03	-check-
6	CR-06	TC-05	-check-
7	CR-07	TC-07, TC-08	TA-04, -check-
8	CR-08, CR-09	TC-08	TA-04, -check-
9	CR-10	TC-04, TC-10, TC-11	-check-
10	CR-11, CR-12, CR-13	TC-07	TA-03, TA-04, -check-
11	CR-14	TC-09	-check-
12	CR-15	TC-04	-check-
13	OR-01	TC-13	-check-
14	OR-02	TC-12	TA-03, -check-
15	OR-03, OR-04	TC-08	-check-
16	OR-05	TC-16	TA-05, -check-
17	OR-06	TC-03	-check-
18	OR-07	TC-15	-check-
19	OR-08	TC-11	-check-
20	OR-09, OR-10	TC-12	-check-

#### 4.2. Case Studies

To further evaluate the perceived usefulness of the MalFe platform, three existing research papers that utilize Cuckoo reports as a data source were explored.

##### 4.2.1. Early Detection Of Crypto-Ransomware Using Pre-Encryption Detection Algorithm [36]

The work by Kok et al. [22] explored ransomware detection using pre-encryption by exploring eleven API function calls. The authors explored a random forest classifier as well as their own custom algorithm. The study made use of 582 ransomware and 942 benignware samples. These samples were then run using Cuckoo Sandbox, and the reports were used for the detection mechanism. In order to show how this study could have benefited from using the MalFe platform, a replication of Kok et al. [36] using the eleven

identified API calls was conducted. The MalFe platform's 3084 samples (as of 12 June 2023) is already significantly more in comparison to the original study. Thereafter, a random forest classifier was used to perform machine learning on the newly formed dataset. The new dataset consisted of 11,430 records, of which 8774 were malicious. The random forest classifier had an accuracy of 90.2% and an Area Under the Curve (AUC) of 91.7%. In comparison, Kok et al. [36] achieved a 99.8% accuracy and an AUC of 99.9%, which may be inaccurate, as the paper represents the results in a graph as opposed to a table. This further illustrates the need for a platform such as MalFe that allows research findings to be easily validated. This dataset replication is available on the MalFe platform (created on 19 June 2023) (<https://malfe.cs.up.ac.za/datasets/view/CryptAPIStatisticsofRansomware/17>).

#### 4.2.2. A Novel Malware Analysis Framework for Malware Detection and Classification Using Machine Learning Approach [16]

This work by Sethi et al. [16] explored malware detection through Cuckoo Sandbox reports by extracting the API function calls and using these data as features for the machine learning algorithms. Both benign and malicious samples were executed in Cuckoo Sandbox, resulting in 220 samples. Sethi et al. [16] asserted that this was a laborious process, and represented a challenging task. With the adoption of the MalFe platform, Sethi et al. [16] would have been able to easily make use of the Cuckoo reports available on the platform and write a short script to extract the API calls to form a dataset with minimal effort, with no need to download these large reports or be concerned about storage. Another benefit would have been that the dataset and Cuckoo reports would be publicly available to anyone. A replication of this study was not possible, as Sethi et al. [16] did not enumerate all the features they explored and as such any replication would necessarily be based on inaccurate assumptions.

#### 4.2.3. Assessment of Supervised Machine Learning Algorithms Using Dynamic API Calls for Malware Detection [12]

This research by Singh and Singh [12] similarly explored malware detection using API calls originating from Cuckoo Sandbox reports. Singh and Singh indicated the difficulty of obtaining malware samples, as they had to source binaries from multiple locations. Furthermore, they identified several registry keys that could be investigated as features through a manual process. Singh and Singh would have benefited from using the MalFe platform in this research, as they would have already had access to benign and malicious reports. Even if there were insufficient reports, they could have run more analysis and uploaded the reports onto the MalFe platform, further driving the community-driven nature of the platform. This would have provided more data reports for future researchers while improving the platform and promoting its adoption over time. By analyzing the registry, Singh and Singh would have been able to benefit from the IntelliView feature on the MalFe platform, enabling quick and easy comparisons between samples to identifying good features to be used in machine learning algorithms.

#### 4.2.4. Literature Search

We conducted a non-exhaustive search of research repositories, including IEEE Xplore, Springer Link, Taylor and Francis, and Science Direct. These research repositories were explored to determine the number of papers that utilized Cuckoo Sandbox reports as their data source. From Table 6, it can be observed that IEEE Explore does not have very many articles on malware detection or Cuckoo Sandbox, with only 23 articles found. Springer Link and Science Direct, on the other hand, contain a large amount of research utilizing Cuckoo Sandbox. This further demonstrates the need for the MalFe platform as a mechanism to avoid unnecessary duplication of effort and experiments to obtain the data needed for machine learning purposes.

**Table 6.** Related literature findings (10 June 2023).

Keyword	IEEE Explore	Springer Link	Taylor and Francis	Science Direct
Machine learning + cuckoo sandbox	16	458	16	137
Malware detection + cuckoo sandbox	23	448	15	145
Cuckoo sandbox	32	519	19	167

#### 4.2.5. Comparative Analysis

While no other platform provides what MalFe can offer the research community, similar platforms are available, including Kaggle [30], DVC [37], Amazon SageMaker [38], IBM Watson Studio [39], DataRobot [31], and Alteryx [32]. However, they do not provide the ability to use the raw data for further research. These platforms only provide what authors make available in terms of processed datasets, and are quite restrictive with regard to future research prospects. In the security field, datasets are often kept private or made available only after a vetting process, making it difficult for researchers to gain access to valuable data for their research. This is usually because there is a monetary value associated with such datasets. Another similar platform is Google Datasets [40]; however, as with Kaggle, it is limited to the datasets that an author uploads on the platform, and does not allow for customization, tweaking, or building novel datasets from the raw data. The results after comparing the various platforms are presented in Table 7. From these results, it is apparent that while there is an increasing demand for AI pipelines and machine learning, there are no platforms that support raw data access and processing.

**Table 7.** Comparative analysis of similar platforms.

Features	Platform						
	MalFe	Kaggle	DVC	Amazon SageMaker	IBM Watson Studio	Data Robot	Alteryx
Access to raw data?	Yes	No	No	No	No	No	-
Allow Data Processing?	Yes	No	Yes	Yes	Yes	Yes	Yes
Community Driven?	Yes	Yes	No	No	No	No	No
Versioning?	Yes	No	Yes	No	-	Yes	-
Model Training?	Future Work	No	Yes	Yes	Yes	Yes	Yes
Model Code?	Future Work	Public	Private	Private	Private	Private	No
Public Comment?	Yes	Yes	No	No	No	No	No

The next section aims to provide a real-world use case for the MalFe platform as well as an example of the research that can be performed using the platform.

### 5. Using MalFe for Ransomware Detection Using PE Entropy

In this section of the paper, we explore how research is conducted using the MalFe platform. The Portable Execution (PE) header of executable files in the Windows Operating System was explored for ransomware detection. The PE information describes how the OS should execute the file and how to allocate memory. These PE sections, along with their entropies, are available in Cuckoo reports. Because the MalFe platform acts as a repository for Cuckoo reports, this case scenario can be used to demonstrate the platform's usefulness and ease of use. However, as the platform did not have enough samples, more benign and malicious samples from Cuckoo reports were uploaded from previous experimentation [5]. The process of uploading a sample to the platform is presented in Figure 5. Selected metadata are extracted from the selected report, such as the Sample SHA256 hash and Name fields, with the user required to select the category (ransomware) to which the report belongs.

**Figure 5.** Uploading Cuckoo reports on MalFe.

Creating a dataset was effortless, and involved providing a name and description followed by selecting the reports to be used in creating the dataset. In this case, all the benign and ransomware categories were selected, as seen in Figure 6. Finally, the code was added to parse the raw reports and enqueued in order for the dataset to be generated, as seen in Figure 7. The dataset generation process took approximately 26 min, spanning 100 + GB worth of data. The dataset consisted of 8599 rows and five columns, with 688 benign and 7911 malicious records sourced from 3084 Cuckoo reports. A snippet of the raw data can be seen in Table 8. The names map to the type of PE section information extracted from the executable along with the entropy of the data in that section. The category is displayed along with the label, where M stands for malicious and B for benign. The dataset is available on the MalFe platform (created on 12 January 2023) ([https://malfe.cs.up.ac.za/datasets/view/Entropy\\_of\\_PE\\_Sections/5#version1](https://malfe.cs.up.ac.za/datasets/view/Entropy_of_PE_Sections/5#version1)).

**Table 8.** Dataset features.

No.	Name	Entropy	Category	Label
0	.text	6.404235	Ransomware	M
1	.rdata	6.663571	Ransomware	M
...	...	...	...	...
8597	.reloc	6.669854	Ransomware	M

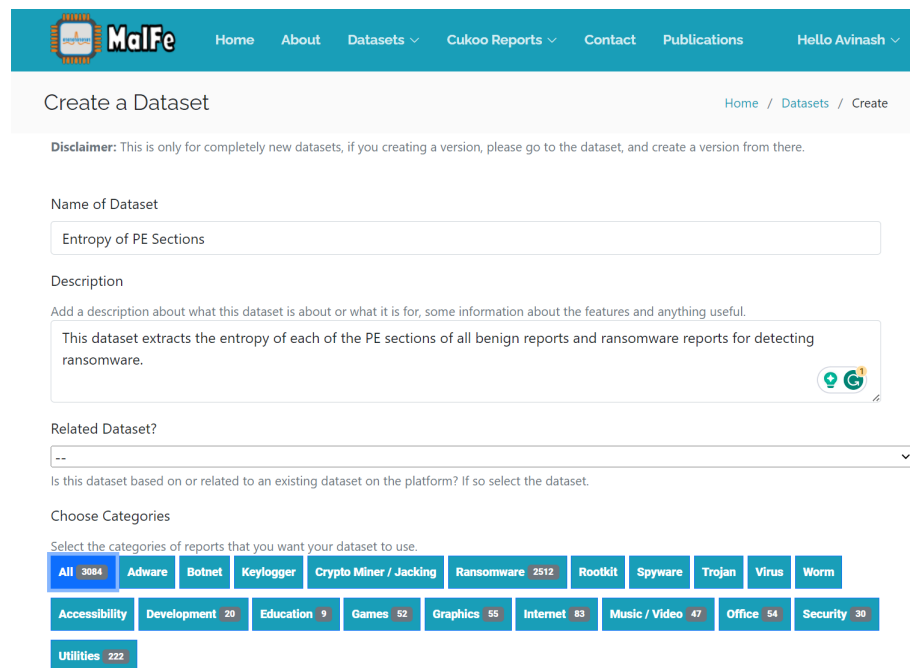


Figure 6. Creating a dataset on MalFe.

```

1
2 def process(data, filename, sha256, category, malicious):
3     try:
4         # Implement your data processing here, this function will get called with the data of each
5         # report in the category you have selected along with the filename of the report and hash.
6         # Do your processing and add your data to the features array, and return the features_labels,
7         # PLEASE NOTE: THE CODE BELOW SERVES AS AN EXAMPLE ONLY! This example extracts the entropy of the PE
8         # Sections. Which can be seen here: https://malfe.cs.up.ac.za/datasets/view/Entropy_of_PE_Sections/5
9         ###
10        features_labels = ["name", "entropy", "category", "label"]
11        features = []
12        label = "M" if malicious else "B"
13        if "static" in data:
14            if "pe_sections" in data["static"]:
15                for pe in data["static"]["pe_sections"]:
16                    features.append([pe["name"], pe["entropy"], category, label])
17        ###
18        return features_labels, features, sha256
19
20
21    except Exception as e:
22        log("|process error| " + filename + " -> " + sha256 + " (" + str(e) + ")")
23    pass
24
25

```

Public

Should this dataset be made public?

**Disclaimer:** Dataset generation might take time to generate. You will be notified once the dataset has been generated. Please continue to add Cuckoo Reports and build datasets, and also Please do not be malicious.

Submit

Figure 7. Dataset code on MalFe.

The next step is to perform quality control on the dataset and feed it into the machine learning algorithms. For this purpose, the Category and sha256 columns were dropped, leaving only the entropy of the sections for classification. Any null values were dropped as well, and manual checks were conducted. Tree classifiers such as Decision Tree (DT) and Random Forrest (RF) were chosen based on their ability to easily build logic paths that are human-readable and show better accuracy predictions related to computer security [5,27,41]. Boosted classifiers were explored as well, such as XGBoost (XGB), to determine the effects of boosted learning. Probabilistic classifiers such as Naïve Bayes (NB) were explored to check whether patterns in the name of the entropy section could provide good accuracy. Finally, clustering classifiers were explored using the K-nearest Neighbors (KNN) and Support Vector Machine (SVM) approaches. DT follows a rule-based approach, drawing patterns to build a tree structure from the data representing the outcomes. RF works similarly to

DT, building multiple trees in a single model to better map hierarchical patterns. SVM is designed for working with small amounts of data and finding associations between the data points using hyperplanes. XGB, on the other hand, is a form of boosted learning combined with a DT that evaluates multiple trees over several iterations until the best solution is found. Finally, NB draws correlations between individual features rather than introducing the relation with each feature as a whole. In order to remove any bias induced by the dataset, ten-fold cross-validation was performed when the models were trained. Furthermore, hyperparameter tuning was employed to determine the best parameters for each model; these parameters are shown in Table 9. For example, the criterion for DT model refers to the quality measure of how the data are split. The Gini impurity is a measure of how often a randomly chosen element is incorrectly classified. In the context of decision trees, the entropy measures the information gain achieved by a split, quantifying the disorder or randomness in a dataset. Using the log\_loss, the classifications are based on probability. The criterion parameter is used to determine which approaches mentioned above better suit the data at hand. For the splitter parameter, the best value uses the criterion to determine the best split, whereas the random value prevents overfitting. These parameters were used in all possible combinations in order to determine the parameters that yielded the best result. With RF, the number of estimators determines how many individual trees to construct and the max depth determines how many nodes the tree will have. Similarly, with KNN the number of neighbours is used to cluster datapoints together, and the distances between the clusters act as the weights. Lastly, with XGB, the type of boosting algorithm and the learning rate can be tuned to determine which configuration reveals the best result.

**Table 9.** Hyperparameter tuning.

Model	Parameters
DT	<i>'criterion'</i> : [ <i>'gini'</i> , <i>'entropy'</i> , <i>'log_loss'</i> ], <i>'splitter'</i> : [ <i>'best'</i> , <i>'random'</i> ]
RF	<i>'n_estimators'</i> : [5, 50, 250], <i>'max_depth'</i> : [2, 4, 8, 16, 32, None]
KNN	<i>'n_neighbors'</i> : [5, 50, 250], <i>'weights'</i> : [ <i>'uniform'</i> , <i>'distance'</i> ], <i>p'</i> : [1, 2], <i>'leaf_size'</i> : [5, 50, 250]
XGB	<i>'booster'</i> : [ <i>'gbtree'</i> , <i>'gblinear'</i> , <i>'dart'</i> ], <i>'learning_rate'</i> : [0.01, 0.1, 1, 10, 100]
NB	N/A

The results of the machine learning phase are further presented in Table 10. The precision measures the ratio of true positives and total positives predicted, where the goal is to reach a precision of 1. The recall is the ratio of true positives to all positives. The F-measure is the mean of the precision and recall. This is arguably the most important factor in determining a model's validity and accuracy. The sensitivity, on the other hand, measures the ability to detect positive instances whereas specificity measures the true negatives. Because there were only two features in this dataset, it would not have been feasible to explore SVM, as this algorithm works better with a larger number of features and fewer records. However, a brief test was conducted to measure the performance; with ten-fold cross-validation, SVM with a linear kernel did not finish training, and was terminated after running for two days. SVM had poor results when no cross-validation was performed, with an accuracy of only 68% even after a significant training and prediction time. In comparison, the other models' training times were within the range of 9 s to 2 min.

All the models (DT, RF, KNN, XGB, NB) presented in Table 10 achieved high levels of True Positives (TP) and True Negatives (TN), indicating their effectiveness in correctly classifying both positive and negative instances. The recall values for all models are generally high, suggesting that the models can identify a significant portion of the actual positive cases. The precision values are consistent with the accuracy, indicating that the models make relatively few false positive predictions. Log-loss values vary among the models, with the RF model achieving the lowest log-loss, indicating better probabilistic predictions. The AUC values are generally high for all models, particularly for XGB and



RF, indicating strong discriminative capability. The F-measure values are high as well, indicating a balance between precision and recall. Figure 8 shows the recall vs. the area under the curve vs. the F-measure vs. the accuracy, where NB performs the poorest among all the models. In Figure 9, the latency varies among the models, with KNN having the highest latency and NB having the lowest. A comparison between the obtained results and those of existing related works is shown in Table 11. From the comparison, the proposed PE entropy provides a good accuracy score, making it the second best among all the explored results from the literature. However, the work by Poudyal and Dasgupta [29] only explored 1090 samples, which could have introduced a bias in prediction. Furthermore, the use of multiple classifiers for detection presents a complexity challenge that would void the potential of pre-emptive detection and ransomware prevention. For instance, the NotPetya ransomware would have overtaken a system within the time required to compute and identify the ransomware using multiple classifiers. Preemptive detection relies on timely detection using salient attributes. This logic requires a knowledge base signature which is not achievable using multiple classifiers. In this regard, the results obtained in the current study can be readily deployed both for early detection and as a proactive preventive mechanism.

**Table 10.** Results of machine learning algorithms.

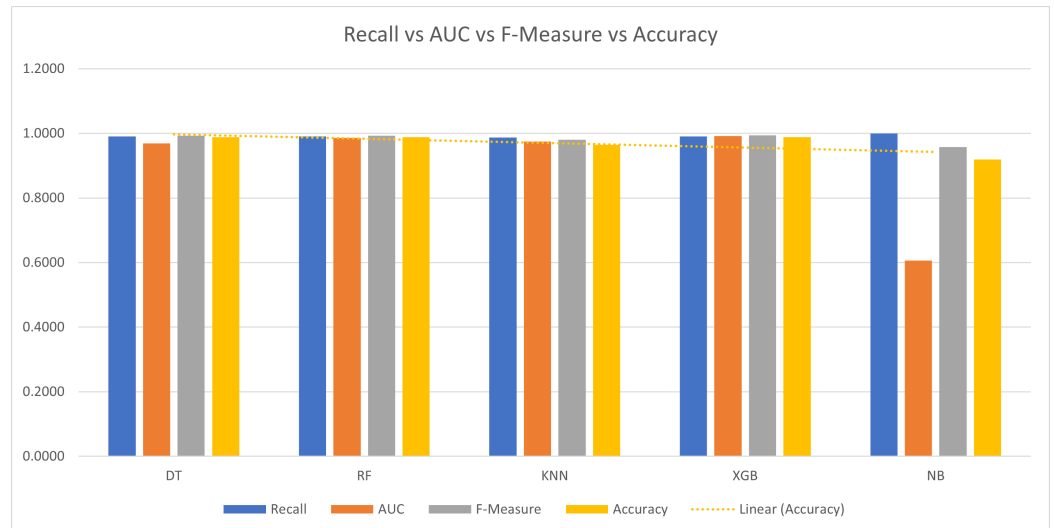
Machine Learning Algorithms					
Metrics of Evaluation	DT	RF	KNN	XGB	NB
True Positive	130	131	131	130	0
True Negative	1541	1540	1540	1542	1555
False Positive	14	15	15	13	0
False Negative	7	6	6	7	137
Recall	0.991	0.990	0.981	0.991	1.000
Precision	0.996	0.996	0.980	0.996	0.919
Log Loss	0.429	0.170	0.669	0.046	0.277
AUC	0.969	0.986	0.975	0.992	0.606
F-Measure	0.993	0.993	0.980	0.994	0.958
Latency (ms)	1.509	12.995	89.511	4.994	0.999
Accuracy	0.988	0.988	0.964	0.988	0.919

**Table 11.** Comparison of results.

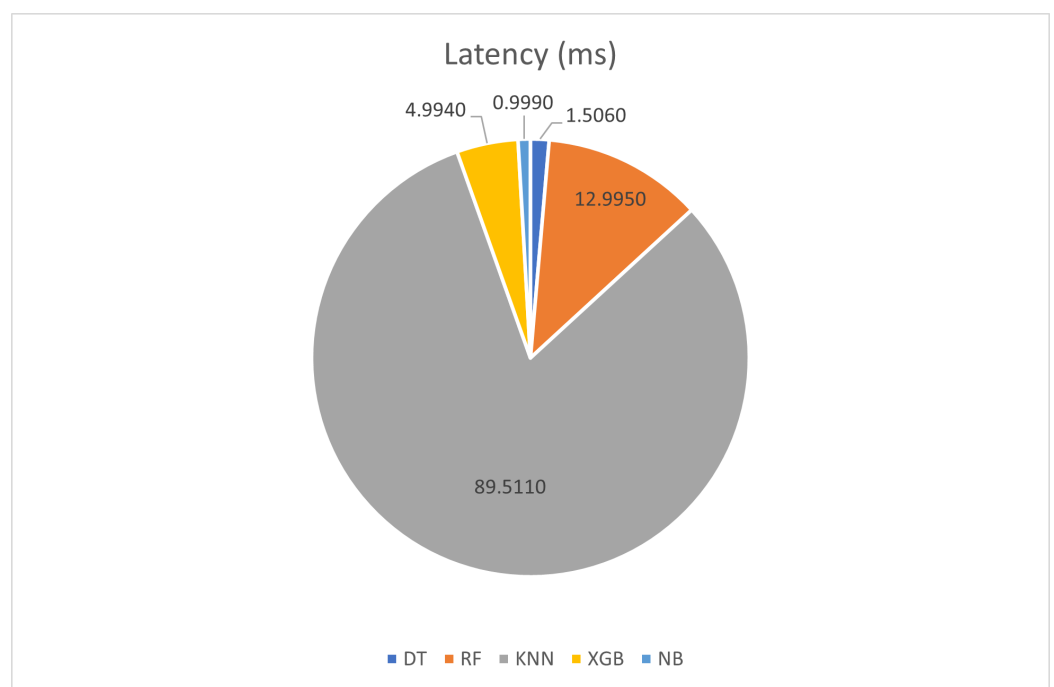
Reference	Dataset Samples	Model	Accuracy
[5]	471	XGBoost	96.28%
[13]	3444	ResNet-18	94%
[23]	4997	Voted Perceptron	98.7%
[26]	1650	Random Forest	98.1%
[27]	6000	Pegasos	90.03%
[29]	1090	SVM + Adaboost + J48	99.54%
Proposed Method	3084	Decision Tree	98.8 %

The Receiver Operating Characteristic (ROC) curve presents a true evaluation of a machine learning algorithm model which graphs the prediction confidence of the model. The confidence is a true reflection of the model's ability to distinguish the classes in a given feature space. The ROC curve for the machine learning algorithms is shown in Figure 10. The goal is to obtain a full right-angle curve (area under the curve = 1), which RF, XGB, and KNN are almost able to achieve. NB is able to predict extremely fast, with a 1 ms response time; however, the number of false negatives is fairly high in comparison to the other models. This means that although it is fast, it is not very confident in its predictions, as can be seen in Figure 10 and Table 10, with an AUC of 0.6, which is extremely poor. Because DT, RF, and XGB performed fairly similarly, with 98.8% accuracy, a validation test consisting of 1555 malicious and 138 benign unseen records was fed into the trained model to benchmark them against each other. The results of this benchmark resulted in 0.9829, 0.9858, and 0.9829 respectively. This leaves

the overall best model with the highest accuracy as Random Forest; however, as speed is more crucial in the domain of ransomware detection, the most suitable and accurate model for ransomware detection in this case is a Decision Tree.



**Figure 8.** Recall vs. AUC vs. F-measure.



**Figure 9.** Latency of prediction in milliseconds.

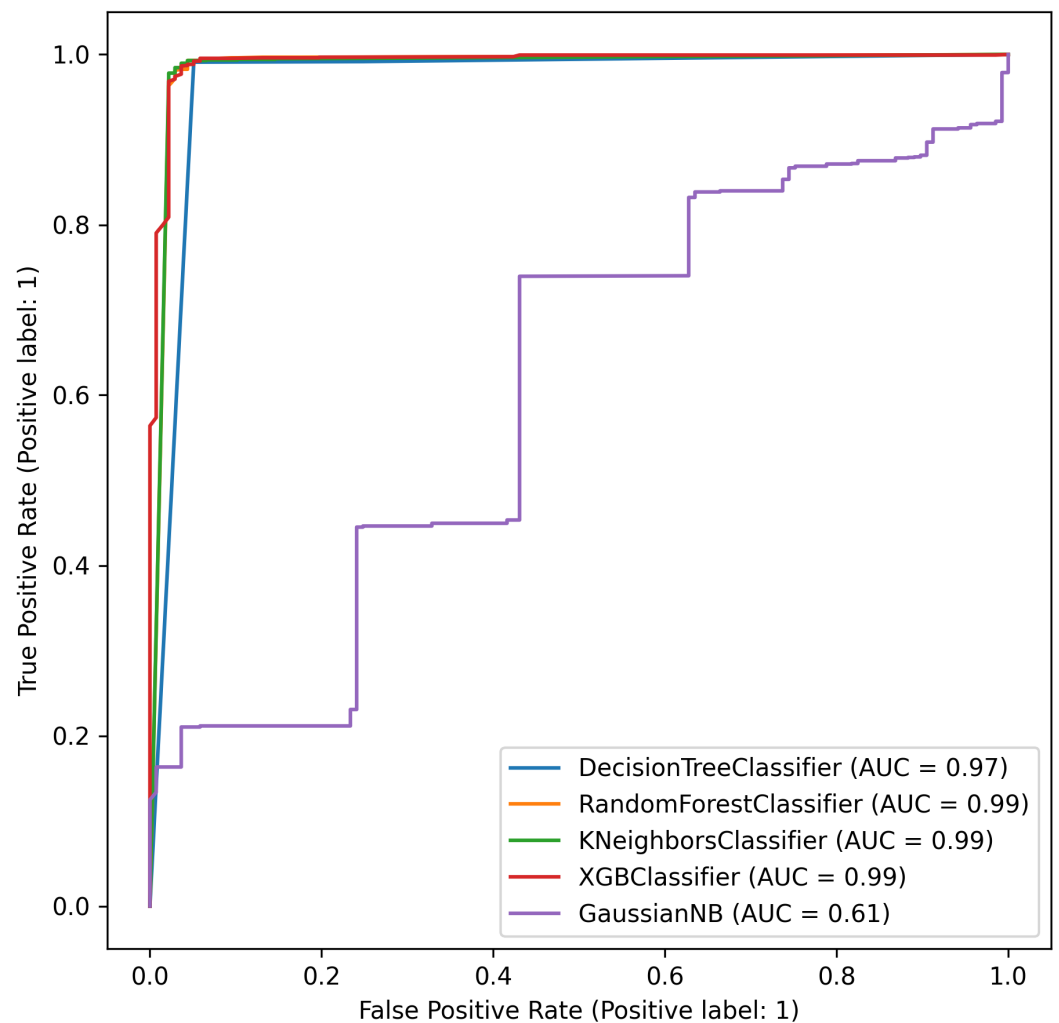


Figure 10. ROC curve.

## 6. Conclusions

The developed platform represents a valuable resource for malware researchers, streamlining the process of sample collection and execution within a sandbox environment and thereby facilitating the acquisition of essential data for machine learning purposes. This platform not only alleviates the need for redundant and labour-intensive experimentation, it can serve as a collaborative community-driven data repository housing Cuckoo reports. With this platform, researchers can create novel datasets without the need to download numerous large Cuckoo reports or deal with storage limitations. Additionally, the platform offers a unique approach to manual analysis and detection criteria by allowing users to compare Cuckoo reports side-by-side. Furthermore, the platform aids in building trust, as the code and reports used for dataset generation are available for public view. This allows research to be repeatable and verifiable, aiding in the dissemination of high-quality research.

To the best of the authors' knowledge, there are currently no available platforms that provide the research community with customizable raw data to build datasets for malware detection. Based on the evaluations conducted in this study, the MalFe platform holds the potential to significantly simplify the challenging task of obtaining high quality datasets for machine learning applications. Future enhancements to the platform may include automated machine learning model generation, further strengthening its utility and robustness. These developments would enhance the platform's adoption while expanding the array of tools available to security researchers.

**Author Contributions:** Conceptualization, methodology, software, validation, analysis, data curation, writing—original draft: A.S. Writing—review and editing: R.A.I., H.V. and A.S. Supervision: R.A.I. and H.V. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded in part by the National Research Foundation of South Africa (Grant Number 136239), and the APC was funded by a UCDP Grant.

**Data Availability Statement:** Dataset can be found at ([https://malfe.cs.up.ac.za/datasets/view/Entropy\\_of\\_PE\\_Sections/5](https://malfe.cs.up.ac.za/datasets/view/Entropy_of_PE_Sections/5), 14 August 2023).

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

API	Application Programming Interface
AUC	Area Under the Curve
CR	Core Requirements
DT	Decision Tree
KNN	k-Nearest Neighbors
NB	Naive Bayes
OR	Optional Requirements
PE	Portable Execution
RF	Random Forest
ROC	Receiver Operating Characteristic
SVM	Support Vector Machine
TA	Test Assertions
TC	Test Cases
XGB	XGBoost

## References

1. AV-Test. Malware Statistics. Available online: <https://www.av-test.org/en/statistics/malware/> (accessed on 21 September 2023).
2. Singh, A.; Venter, H.S. *Digital Forensic Readiness Framework for Ransomware Investigation*; Springer International Publishing: Cham, Switzerland, 2019; pp. 91–105. [CrossRef]
3. O'Brien, D. Internet Security Threat Report—Ransomware 2017. *Symantec* **2017**, *20*, 1–35.
4. Al-rimy, B.A.S.; Maarof, M.A.; Shaïd, S.Z.M. Ransomware threat success factors, taxonomy, and countermeasures: A survey and research directions. *Comput. Secur.* **2018**, *74*, 144–166. [CrossRef]
5. Singh, A.; Ikuesan, R.; Venter, H. Ransomware Detection using Process Memory. *Int. Conf. Cyber Warf. Secur.* **2022**, *17*, 413–422. [CrossRef]
6. Fan, Y.; Ye, Y.; Chen, L. Malicious sequential pattern mining for automatic malware detection. *Expert Syst. Appl.* **2016**, *52*, 16–25. [CrossRef]
7. Arshad, H.; Jantan, A.B.; Abiodun, O.I. Digital forensics: Review of issues in scientific validation of digital evidence. *J. Inf. Process. Syst.* **2018**, *14*, 346–376. [CrossRef]
8. Boulevard, S. A Universal Bypass Tricks Cylance AI Antivirus. Available online: <https://securityboulevard.com/2019/07/a-universal-bypass-tricks-cylance-ai-antivirus-into-accepting-all-top-10-malware-revealing-a-new-attack-surface-for-machine-learning-based-security/> (accessed on 5 October 2019).
9. Almashhadani, A.O.; Carlin, D.; Kaiiali, M.; Sezer, S. Computers & Security MFMCNS: A multi-feature and multi-classifier network-based system for ransomworm detection. *Comput. Secur.* **2022**, *121*, 102860. [CrossRef]
10. Dang, D.; Di Troia, F.; Stamp, M. Malware classification using long short-term memory models. In Proceedings of the ICISSP 2021—7th International Conference on Information Systems Security and Privacy 2021, Online, 11–13 February 2021; pp. 743–752. [CrossRef]
11. Jiang, Q.; Zhao, X.; Huang, K. A Feature Selection Method for Malware Detection. In Proceedings of the IEEE International Conference on Information and Automation, Shenzhen, China, 6–8 June 2011; pp. 890–895.
12. Singh, J.; Singh, J. Assessment of supervised machine learning algorithms using dynamic API calls for malware detection. *Int. J. Comput. Appl.* **2022**, *44*, 270–277. [CrossRef]
13. Ashraf, A.; Aziz, A.; Zahoor, U.; Khan, A. Ransomware Analysis using Feature Engineering and Deep Neural Networks. *arXiv* **2019**, arXiv:1910.00286.
14. Guarnieri, C. Cuckoo Sandbox. 2014. Available online: <https://cuckoosandbox.org/> (accessed on 4 October 2018).
15. NIST. Computer Forensics Tool Testing Program. 2014. Available online: <http://www.cftt.nist.gov/> (accessed on 12 October 2017).

16. Sethi, K.; Tripathy, B.K.; Chaudhary, S.K.; Bera, P. A Novel Malware Analysis for Malware Detection and Classification using Machine Learning Algorithms. In Proceedings of the ACM International Conference Proceeding Series, Jaipur, India, 13–15 October 2017; pp. 107–116. [CrossRef]
17. Gandotra, E.; Bansal, D.; Sofat, S. Malware Threat Assessment Using Fuzzy Logic Paradigm. *Cybern. Syst.* **2017**, *48*, 29–48. [CrossRef]
18. Bergeron, J.; Debbabi, M.; Desharnais, J.; Erhioui, M.M.; Lavoie, Y.; Tawbi, N.; Bergeron, J.; Debbabi, M.; Desharnais, J.; Erhioui, M.; et al. Static Detection of Malicious Code in Executable Programs. *Int. J. Req. Eng.* **2001**, *79*, 184–189.
19. Sihwail, R.; Omar, K.; Ariffin, K.A.Z. An Effective Memory Analysis for Malware Detection and Classification. *Comput. Mater. Contin.* **2021**, *67*, 2301–2320. [CrossRef]
20. Kiger, J.; Ho, S.S.; Heydari, V. Malware Binary Image Classification Using Convolutional Neural Networks. *Int. Conf. Cyber Warf. Secur.* **2022**, *17*, 469–478. [CrossRef]
21. Grégio, A.R.A.; De Geus, P.L.; Kruegel, C.; Vigna, G. Tracking memory writes for malware classification and code reuse identification. In *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7591 LNCS, pp. 134–143. [CrossRef]
22. Singh, J.; Singh, J. A survey on machine learning-based malware detection in executable files. *J. Syst. Archit.* **2021**, *112*, 101861. [CrossRef]
23. Faruki, P.; Laxmi, V.; Gaur, M.S.; Vinod, P. Behavioural detection with API call-grams to identify malicious PE files. In Proceedings of the International Conference on Security of Internet of Things, Kollam, India, 17–19 August 2012.
24. Singh, A.; Ikuesan, A.; Venter, H. A context-aware trigger mechanism for ransomware forensics. In Proceedings of the 14th International Conference on Cyber Warfare and Security, ICCWS 2019, Stellenbosch, South Africa, 28 February–1 March 2019; pp. 629–638.
25. Freund, Y.; Schapire, R.E. Large margin classification using the perceptron algorithm. In Proceedings of the Eleventh Annual Conference on Computational Learning Theory, Madison, WI, USA, 24–26 July 1998; pp. 209–217.
26. Hansen, S.S.; Larsen, T.M.T.; Stevanovic, M.; Pedersen, J.M. An approach for detection and family classification of malware based on behavioral analysis. In Proceedings of the 2016 International Conference on Computing, Networking and Communications, ICNC 2016, Kauai, HI, USA, 15–18 February 2016. [CrossRef]
27. Darshan, S.L.; Kumara, M.A.; Jaidhar, C.D. Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. In Proceedings of the 11th International Conference on Industrial and Information Systems, ICIIS 2016—Conference Proceedings, Roorkee, India, 3–4 December 2016; pp. 534–539. [CrossRef]
28. Shalev-Shwartz, S.; Singer, Y.; Srebro, N. Pegasos: Primal Estimated sub-GrAdient Solver for SVM. In Proceedings of the 24th International Conference on Machine Learning, Corvallis, OR, USA, 20–24 June 2007; pp. 807–814.
29. Poudyal, S.; Dasgupta, D. AI-Powered Ransomware Detection Framework. In Proceedings of the 2020 IEEE Symposium Series on Computational Intelligence (SSCI), Canberra, Australia, 1–4 December 2020; pp. 1154–1161. [CrossRef]
30. Kaggle: Your Machine Learning and Data Science Community. Available online: <https://www.kaggle.com/> (accessed on 18 June 2023).
31. Prepare Data for AI | DataRobot AI Platform. Available online: <https://www.datarobot.com/platform/prepare-modeling-data/> (accessed on 18 September 2023).
32. Data Preparation Tools—Alteryx. Available online: <https://www.alteryx.com/products/capabilities/data-preparation-tools> (accessed on 25 September 2023).
33. Django Auth. Available online: <https://docs.djangoproject.com/en/2.2/topics/auth/passwords/> (accessed on 4 October 2022).
34. M'Raihi, D.; Machani, S.; Pei, M.; Rydell, J. TOTP: Time-Based One-Time Password Algorithm. Available online: <https://www.rfc-editor.org/rfc/rfc6238.html> (accessed on 4 October 2022).
35. RestrictedPython. Available online: <https://pypi.org/project/RestrictedPython/> (accessed on 25 February 2023).
36. Kok, S.H.; Abdullah, A.; Jhanjhi, N.Z. Early detection of crypto-ransomware using pre-encryption detection algorithm. *J. King Saud Univ.-Comput. Inf. Sci.* **2022**, *34*, 1984–1999. [CrossRef]
37. AI, I. Data Version Control · DVC—dvc.org. Available online: <https://dvc.org/> (accessed on 18 September 2023).
38. Machine Learning—Amazon Web Services. Available online: <https://aws.amazon.com/sagemaker/> (accessed on 18 September 2023).
39. IBM Watson Studio | IBM. Available online: <https://www.ibm.com/products/watson-studio> (accessed on 18 September 2023).
40. Dataset Search. Available online: <https://datasetsearch.research.google.com/> (accessed on 18 June 2023).
41. Raghuraman, C.; Suresh, S.; Shivshankar, S.; Chapaneri, R. Static and dynamic malware analysis using machine learning. *Adv. Intell. Syst. Comput.* **2020**, *1045*, 793–806. [CrossRef]

**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.