applied sciences

MDPI

Article

Malware API Calls Detection Using Hybrid Logistic Regression and RNN Model

Abdulaziz Almaleh *, Reem Almushabb and Rahaf Ogran

Information Systems Department, King Khalid University, Abha 62529, Saudi Arabia; 443800987@kku.edu.sa (R.A.); rogran@kku.edu.sa (R.O.)
* Correspondence: ajoyrulah@kku.edu.sa; Tel.: +966-533-212-174

Abstract: Behavioral malware analysis is a powerful technique used against zero-day and obfuscated malware. Additionally referred to as dynamic malware analysis, this approach employs various methods to achieve enhanced detection. One such method involves using machine learning and deep learning algorithms to learn from the behavior of malware. However, the task of weight initialization in neural networks remains an active area of research. In this paper, we present a novel hybrid model that utilizes both machine learning and deep learning algorithms to detect malware across various categories. The proposed model achieves this by recognizing the malicious functions performed by the malware, which can be inferred from its API call sequences. Failure to detect these malware instances can result in severe cyberattacks, which pose a significant threat to the confidentiality, privacy, and availability of systems. We rely on a secondary dataset containing API call sequences, and we apply logistic regression to obtain the initial weight that serves as input to the neural network. By utilizing this hybrid approach, our research aims to address the challenges associated with traditional weight initialization techniques and to improve the accuracy and efficiency of malware detection based on API calls. The integration of both machine learning and deep learning algorithms allows the proposed model to capitalize on the strengths of each approach, potentially leading to a more robust and versatile solution to malware detection. Moreover, our research contributes to the ongoing efforts in the field of neural networks, by offering a novel perspective on weight initialization techniques and their impact on the performance of neural networks in the context of behavioral malware analysis. Experimental results using a balanced dataset showed 83% accuracy and a 0.44 loss, which outperformed the baseline model in terms of the minimum loss. The imbalanced dataset's accuracy was 98%, and the loss was 0.10, which exceeded the state-of-the-art model's accuracy. This demonstrates how well the suggested model can handle malware classification.

Keywords: malware; malware detection; API calls; logistic regression; neural network; weight initialization

check for updates

Citation: Almaleh, A.; Almushabb, R.; Ogran, R. Malware API Calls Detection Using Hybrid Logistic Regression and RNN Model. *Appl. Sci.* **2023**, *13*, 5439. https://doi.org/10.3390/app13095439

Academic Editor: Luis Javier García Villalba

Received: 12 April 2023
Revised: 23 April 2023
Accepted: 25 April 2023
Published: 27 April 2023

1. Introduction

The role of cybersecurity has become increasingly significant as cyberattacks have rapidly escalated, necessitating cutting-edge approaches to detecting and preventing malicious behavior [1,2]. In recent years, the prevalence of malware has grown exponentially, posing a significant threat to the security of computer systems and networks worldwide. According to a report by AV-TEST, an independent IT security research institute, they registered over 1.2 billion malware samples by the end of 2021, with approximately 350,000 new malware samples being identified daily [3]. This staggering growth rate highlights the urgency of the development of advanced detection and prevention techniques to combat the ever-evolving malware landscape.

The complexity and sophistication of malware attacks have also increased, with cybercriminals employing new tactics and techniques to evade traditional detection methods. A study conducted by Symantec revealed that targeted attacks increased by 42% in 2021, with ransomware attacks becoming more targeted and lucrative, and supply chain attacks increasing by 93% [4]. These statistics emphasize the need for innovative, intelligent, and automated solutions to identify and counteract the growing threat posed by malware [5].

Malware, an abbreviation for "malicious software", refers to a diverse range of intrusive software programs designed with the intent to compromise the integrity, confidentiality, or availability of computer systems and their resources [6]. These malevolent software entities infiltrate, disrupt, or exploit computer systems, networks, or end-user devices, often leading to unauthorized access, data theft, or corruption [7]. Malware encompasses a broad spectrum of software types, including viruses, worms, Trojans, ransomware, adware, spyware, and rootkits, each exhibiting unique characteristics and methods of propagation [8]. The proliferation of malware poses a significant threat to both individuals and organizations, necessitating continuous advancements in detection and prevention techniques, to safeguard the digital ecosystem [9].

Malware analysis is a critical aspect of cybersecurity that involves systematically examining, understanding, and dissection malicious software. By analyzing malware, researchers and cybersecurity professionals can gain valuable insights into its functionality, purpose, and potential impact on computer systems, networks, and end-user devices [10]. This process plays a pivotal role in enhancing the overall security position of organizations and individuals alike.

The primary objectives of malware analysis include identifying the nature and characteristics of the malware, such as its type (e.g., virus, worm, Trojan), the method of propagation, and its behavior within the infected system [9]. These insights enable the development of effective countermeasures, including antivirus signatures, intrusion detection system (IDS) rules, or software patches that can detect, prevent, or remediate infections caused by malware [7]. Moreover, malware analysis helps strengthen cybersecurity defenses by providing a deeper understanding of the tactics, techniques, and procedures (TTPs) employed by cybercriminals. This knowledge facilitates the development of proactive security measures and strategies to mitigate future threats [6]. Additionally, the information gleaned from malware analysis can support incident response and digital forensic investigations by attributing the malware to specific threat actors, understanding their motives and objectives, and facilitating the collection of evidence for legal or law enforcement purposes [11].

Malware analysis techniques typically fall into two categories: static analysis and dynamic analysis. Static analysis involves examining the malware's code and metadata without executing it, while dynamic analysis entails running the malware in a controlled and isolated environment, to observe its behavior [10]. Both approaches provide valuable information that helps cybersecurity professionals defend against the ever-evolving threat landscape. Malware analysis is an indispensable component of modern cybersecurity practices. Its contributions to understanding the intricacies of malicious software, developing countermeasures, and enhancing overall security defenses make it a vital area of research and a key factor in safeguarding the digital ecosystem against emerging threats.

In previous research, dynamic and static API approaches have primarily been employed independently, with only a few instances of their combined application. This study introduces a novel hybrid model that relies on dynamic API call sequences for malware detection. The model incorporates machine learning logistic regression as an initial weight input for the neural network. This approach offers several advantages, including enhanced malware detection accuracy and mitigation of vanishing and exploding gradients in the neural network. Our hybrid model's innovation lies in its weight initialization process for neural networks, an area that continues to attract research interest. In this paper, we propose a hybrid model that utilizes weights derived from a machine learning model, specifically logistic regression, and integrates them as initial weights in a recurrent neural network model. This unique combination effectively addresses the challenges associated with weight initialization in neural networks, while maximizing the detection capabilities of both dynamic and static API methods.

The rest of the paper is organized as follows. In Section 2, we summarize some previous studies and literature reviews that addressed malware analysis and detection using API calls, machine learning, and deep learning approaches. In Section 3, we elaborate on our proposed hybrid model. In Section 4, we present the results and discussion. Lastly, in Section 5, we conclude our paper and suggest future areas of work.

2. Related Works

To effectively identify concealed malicious code, it is vital to comprehend both the features of API calls and the structure of program executables (PE). Analyzing API calls sheds light on the behavior of a file, while understanding the Win32 portable executable (PE) file format, the standard format provided by Microsoft, revealing the static features of malware that can be obfuscated using evasion techniques.

The majority of malware originates from benign software that has been infected by various types of malicious programs, such as worms. From a code perspective, these are ordinary code sequences that have been injected with harmful code fragments. Both benign and malicious computer programs are primarily processed using a series of ordered API calls. However, attackers often employ techniques such as inserting irrelevant APIs, incorporating conditional triggers, and employing obfuscation to evade detection. As a result, the proposed framework focuses on first dividing the entire API sequence into API fragments of a specific length, which are then used in a training phase. This enables the classifier to identify malicious API fragments and subsequently employ ensemble learning to recognize malicious code based on the proportion of malicious API fragments present within the entire API execution sequence [12].

In September 2019 alone, one million new malware specimens emerged, contributing to a staggering total of 948 million known specimens in the wild. This surge highlights the pressing need for effective malware detection methods. Consequently, a novel behavioral malware detection approach has been proposed, which is based on dynamic analysis data represented by the graph structure of API call sequences. This method leverages a cutting-edge deep learning architecture specifically designed for graph classification to achieve its goal. By defining a graph structure that encapsulates the API call sequence of a program, both the spatial and temporal information of the program's behavior is integrated. Subsequently, a streamlined version of a deep graph convolutional neural network (DGCNN) is employed to learn high-level representations, which a classifier can then utilize to discern whether the program is malicious or benign [13].

In [14], the most noteworthy contribution revolves around developing an innovative image processing technique that employs top-rated parameters for machine learning algorithms (MLAs) and deep learning architectures to achieve an effective 0-day malware detection system. Their approach incorporates both static and dynamic analysis, classical machine learning algorithms, and recurrent neural networks (RNNs), providing a comprehensive solution to the challenge of malware detection.

On the other hand, the authors of [15] focused on a significant contribution, in the form of a mixed-stage detection process. This methodology consists of two primary stages. In the first stage, a Markov model extracts the sequential characteristics of API calls, capturing vital information about the program's behavior. The second stage involves applying multiple machine-learning algorithms to the extracted API calls, enabling the detection model to leverage the strengths of various algorithms for improved accuracy and effectiveness.

A key aspect of the research presented in [15] is the emphasis on minimizing false positive (FP) error rates when evaluating the model's performance. By concentrating on reducing FP error rates, the authors strive to enhance the reliability of the detection model, ensuring that benign programs are not mistakenly flagged as malicious. This focus on model evaluation highlights the importance of precision in malware detection systems, as false positives can have significant implications for the users and organizations relying on these systems for protection.

This research in [16] presents a novel malware detection method based on the visual representation of recombined API command sequences. The authors suggest a method for obtaining instruction sequences that combines static and dynamic analysis and then visualizes them using a heat map. The paper's key contribution is the application of visualization to malware detection, which makes it simpler to find patterns and anomalies. The authors ran tests using a collection of real malware samples, and they successfully detected over 97% of the threats. Overall, the suggested method shows the Table 1.

A multifaceted deep generative adversarial network (MDGAN) model for detecting mobile malware was suggested in the paper in [17]. The proposed model comprises three parts: a malware detector, a discriminator, and a generator. While the discriminator can tell the difference between actual and fake malware samples to enhance the generator, the generator creates synthetic malware samples for training the malware detector. The primary contribution of this research is integrating a deep generative model with a malware detector, which increases the malware detector's detection precision. Compared to recent advances, the suggested MDGAN model achieved 96.2% accuracy.

Table 1 offers a comprehensive overview of various studies conducted in the field, meticulously comparing the distinct methods and classifiers employed across each research endeavor. It provides a clear side-by-side comparison, enabling readers to easily discern the differences and similarities in the methodologies and algorithms utilized, ultimately shedding light on their efficacy in addressing the research question at hand. By examining this comparative analysis, one can gain valuable insights into the underlying techniques and their performance, which may pave the way for more refined and robust approaches in future studies. Additionally, the table serves as a reference point for researchers, facilitating the identification of potential gaps in the literature and inspiring novel contributions to the field.

Table 1. Comparison of Previous Research.

Reference	Method	Classifier	# of Malware	Accuracy
Xin et al., 2019 [12]	Dynamic analysis	Long Short Term Memory (LSTM)	12,000	0.973
Schranko de Oliveira et al., 2019 [13]	Dynamic analysis	Deep Graph Convolutional Neural Networks (DGCNNs)	42,797	0.924
Vinayakumar, R., et al., 2019 [14]	Both static and dynamic analysis	Convolutional Neural Networks CNN	118,717	0.93
Hwang et al., 2020 [15]	Dynamic analysis	Random Forest machine learning model	1909	0.973
Yang, Hongyu et al., 2022 [16]	Both static and dynamic analysis	Malware feature image convolutional neural network (MimgNN)	-	0.97
Mazaed Alotaibi, Fahad, 2022 [17]	Both static and dynamic analysis	Multifaceted Deep Generative Adversarial Networks Model (MDGAN)	5546	0.962

3. Proposed Model

3.1. Dataset and Data Prepossessing

In this research, the data used were obtained from secondary sources, specifically from [13]. The dataset was designed to support the research community by establishing a basis for ongoing progress and improvement. It encompasses 42,797 malicious API call sequences and 1079 non-malicious API call sequences, where each sequence consists of the initial 100 nonrepeated, consecutive API calls. Nonetheless, the dataset exhibits an imbalance, with the malware category being 40% larger than the benign category. Such an imbalance may result in performance challenges when appraising the model using test data. To counter this issue, we utilized an undersampling approach, which balanced the record count in both classes and enabled the classifier to function more efficiently across the two categories. In our investigation, we evaluated the model's performance using both balanced and imbalanced datasets, offering a thorough examination of its potential.

3.2. API Call Sequences

The application programming interface (API) is a critical element of the Windows operating system, encompassing an array of functions housed within specific libraries. Users leverage these functions to interact with the operating system, which in turn mirrors the behavior of various files. Therefore, it is of utmost importance to analyze the manner in which these calls are executed, to determine their authenticity [18]. In order to pinpoint malware, a multitude of API sequences are extracted from malware samples, which then serve as features for detecting the presence of malware [19].

For example, a significant portion of malware infects a victim's machine through Windows dynamic-link libraries (DLLs), which are libraries containing code and data that multiple programs can simultaneously access via API calls. The DLL injection sequence commences with an OpenProcess call to manage the targeted process, followed by allocating memory space using the VirtualAllocateEx call. Subsequently, the malware's complete path is written using WriteProcessMemory, and finally, the targeted process reloads the DLL through CreateRemoteThread. This sequence is extracted from the malware sample for further analysis.

Another example involves import address table (IAT) hooking, where the IAT is a table populated with function pointers that are executed at runtime. In simpler terms, the IAT contains the starting addresses for APIs. By altering the addresses within the table, a different API can be called, resulting in malware behavior, as demonstrated in [20]. Figure 1 illustrates a simple IAT hooking technique, where the application code calls the Function1 API. Using the strcmp() function, the addresses stored in the import address table (IAT) are sequentially compared to find the correct address pointing to the required DLL function. Once the memory address containing the required DLL is located, the VirtualProtect() function is used to change the memory protection permissions, to allow writing and injecting the addresses, so that it can be redirected to the infected Baddll function rather than the required Function1.

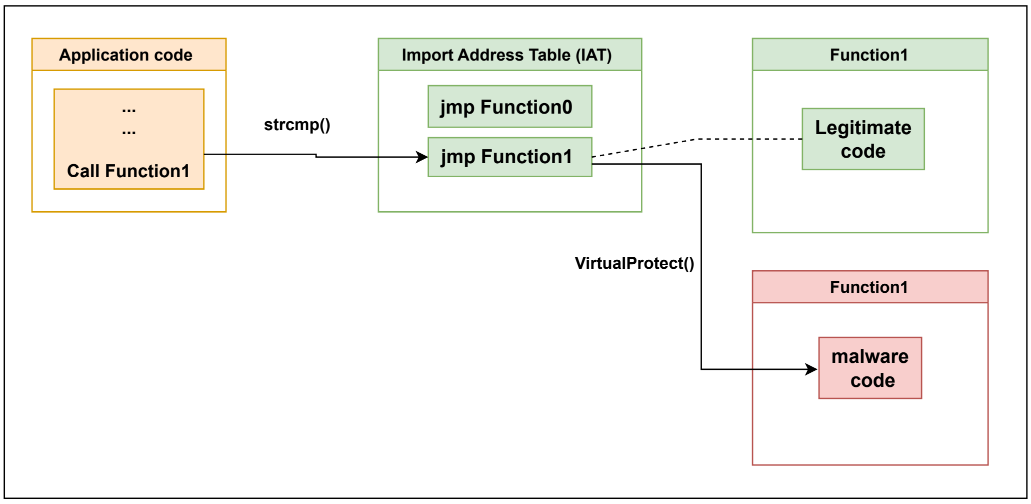


Figure 1. IAT hooking.

The model proposed in this study was trained on a secondary dataset containing samples of malware API call sequences, enabling it to detect unknown malware instances [21].

3.3. Hybrid Model

Weight initialization in neural networks is a difficult task, specifically in iterative neural network algorithms, where the initial point can determine if the algorithm will converge or not and also affect result generalization [22]. For this reason, we need to initialize the NN weights carefully. Our proposed model, as shown in Figure 1, is designed to initialize neural network weights using logistic regression weights. The logistic regression model is a supervised learning technique that is used in binary classification.The logistic regression can be defined as in Equation (1):

$$y = \frac{e^{(wx+b)}}{1 + e^{(wx+b})} \qquad (1)$$

While logistic regression has its limitations for capturing non-linear associations, leveraging it as a source of initial weights for a neural network yields numerous advantages. The proposed hybrid model, which fuses logistic regression and neural networks, seeks to enhance malware detection performance based on API calls by utilizing their combined strengths. Employing logistic regression in this manner allows the model to capitalize on its ability to identify significant features and achieve faster convergence, while the neural network can model complex relationships within the data.

Algorithm 1 demonstrates a collection of API call sequences, denoted as x, which function as the input for the model. The output is a binary classification label, y, that signifies whether the input sequence is malware or not. To develop a coherent model, the input sequences x and their corresponding labels y are employed to train a logistic regression model. This process involves utilizing the training data to compute the logistic regression model's coefficients, represented as θ. The input and output sizes must be defined to initialize the RNN. Following this, the coefficients from the logistic regression model are transposed to serve as the RNN's input-to-hidden weights, denoted as W_x. The training data, a given loss function, an optimizer, and a specified number of epochs, were then used to train the RNN. Lastly, the performance of the trained RNN was assessed by applying a specific evaluation metric on a separate test dataset.

Algorithm 1: Hybrid Model

```
Input: API call sequences $x$.
Output: $y$ that classify if $x$ malware or not malware
1: $\theta \leftarrow TrainLogisticRegression(x, y).coefficient()$
2: $rnn \leftarrow InitializeRNN(InputSize, OutputSize)$
3: $Wx = Transpose(\theta)$
4: $SetWeights(rnn.InputToHiddenWeights, Wx)$
5: TrainRnn(rnn, TrainingData, Activation, LossFunction, Optimizer, NumEpochs)
6: $TestRnn(rnn, TestData, Evaluation)$
7: end
```

The result of logistic regression is a probability ranging from 0 to 1, indicating the likelihood of a given data point belonging to a particular class. Upon completing the model training, we achieved an accuracy of 0.82 with the balanced dataset and 0.98 with the imbalanced dataset. We then extracted the weights obtained during the final stage of the logistic regression training, to be utilized as initial weights in the neural network.

A recurrent neural network (RNN) is a deep learning model that takes the sequence input to perform an operation on it and then produces either one or a sequence of outputs. Here, we used the many-to-one type of RNN, where the model takes many API calls and classifys them into one class, either malware or not malware. In other words, we used a sequential model. The main advantage of this RNN model is that model can remember its previous inputs for computations. The RNN can be defined as shown in Equation (2) :

$$h^{(t)} = f(h^{(t-1)}, x^t; \Theta) \tag{2}$$

In the end, after developing the proposed model, we evaluated its effectiveness. We compared the model's performance on both balanced and imbalanced datasets, as illustrated in Figure 2. Through this evaluation, we sought to gain insights into how well the proposed model could be generalized to different types of dataset. By analyzing the results obtained from the evaluation, we could draw conclusions about the model's effectiveness in handling imbalanced datasets compared to balanced ones. The model's performance on each dataset could be assessed based on various metrics, such as accuracy, precision, recall, and F1 score. This comparison of the model's performance on balanced and imbalanced datasets can inform the development of strategies for improving the model's performance in future applications.

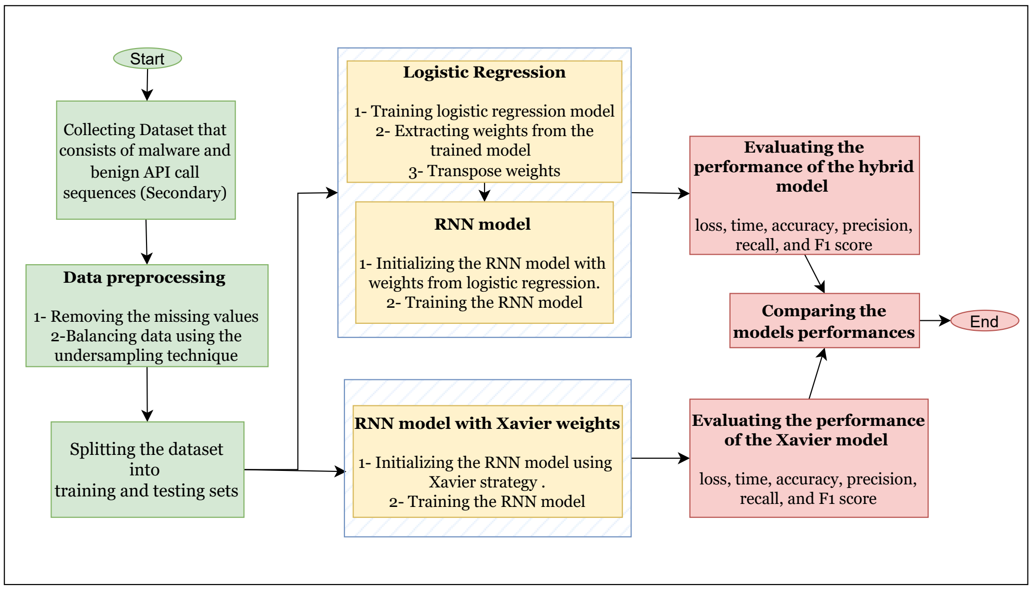

Start
Collecting Dataset that consists of malware and benign API call sequences (Secondary)
Data preprocessing
1- Removing the missing values
2-Balancing data using the undersampling technique
Splitting the dataset into training and testing sets
Logistic Regression
1- Training logistic regression model
2- Extracting weights from the trained model
3- Transpose weights
RNN model
1- Initializing the RNN model with weights from logistic regression.
2- Training the RNN model
RNN model with Xavier weights
1- Initializing the RNN model using Xavier strategy .
2- Training the RNN model
Evaluating the performance of the hybrid model
loss, time, accuracy, precision, recall, and F1 score
Comparing the models performances
End
Evaluating the performance of the Xavier model
loss, time, accuracy, precision, recall, and F1 score

Figure 2. The sequence of operations in our proposed hybrid model.

4. Results and Discussion

This portion of the paper aimed to apply the aforementioned theoretical concepts by devising measurable and comparable experiments. Furthermore, the experiments were organized in a manner that permitted statistical assessment of the results and comparison to established techniques within the field. This methodology aided in identifying the pros and cons of the proposed approach, as well as in appraising their ability to improve neural network performance. To accomplish this, the experiments were carefully crafted to ensure that the investigated variables were distinctly defined and could be precisely gauged. This degree of exactness will allow researchers to derive significant inferences from the information and contribute valuable understanding regarding the efficacy of the suggested methods. Additionally, the experimental design incorporated elements of replicability and control, ensuring that the results are robust and can be confidently generalized to a wider range of scenarios.

Furthermore, the experiments were carried out in a systematic manner, comparing the novel techniques against existing benchmarks in the field. This comparison provided a clear understanding of how the proposed methods stack up against current practices and revealed potential areas for improvement. By carefully analyzing the results, researchers can pinpoint the most promising avenues for future research, fostering innovation and driving the field of neural networks forward.

Regarding the experimental setup, the first model was built using the *Constant* weight initializer with initial weights derived from the logistic regression coefficients. In contrast, the *GlorotUniform* weight initializer was used to generate the second model. The architecture and hyperparameters were the same for both models. Our dataset was randomly divided into training and testing sets, with a ratio of 70/30. We trained both models on the training set, and the testing set served for evaluation of their performance. We used the Adam optimizer, accuracy was the evaluation metric, and binary cross-entropy was the loss function.

In the study by Xavier, in [23], a model was proposed to address the issues of vanishing and exploding gradients in the final layers of neural networks. This method recommends an approach wherein weights are chosen from a uniform distribution, with random values restricted within specified bounds, as shown in Equation (3):

$$w_{ij} \sim U[-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}], \tag{3}$$

In this case, U represents the uniform distribution, and n corresponds to the number of inputs. Given the extensive use of the Xavier technique for weight initialization, along with its proven effectiveness in various deep learning models and architectures, it served as a comparison baseline for evaluating the performance of our proposed approach, in both balanced and imbalanced data scenarios.

Incorporating the Xavier model as a benchmark provided a solid foundation for assessing the potential advantages of our suggested strategy. By comparing the outcomes of our method with those obtained using the well-established Xavier technique, we could establish a clear understanding of the relative strengths and weaknesses of our proposal. This comparative analysis was instrumental in refining the proposed approach and ensuring its relevance and applicability to a diverse range of deep learning contexts.

4.1. Balance Dataset

Following the application of an undersampling technique to balance the data, a method commonly used to tackle imbalanced datasets in classification tasks, the number of majority classes (malware) was adjusted to align with that of the minority classes (not malware). This adjustment helped avoid potential bias in the model towards the majority class. As a result, the number of records for both malware and nonmalicious software reached 1079 each, culminating in a balanced dataset consisting of 2158 records, although somewhat limited in size. Employing the undersampling technique yields advantages in terms of computational efficiency and reduces false alarms in the minority class, which might otherwise occur during training on an imbalanced dataset.

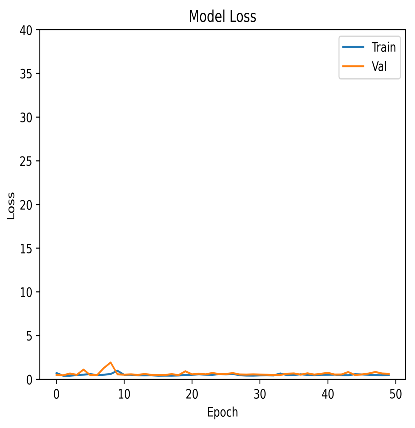
The significance of creating a balanced dataset is not to be underestimated, as it ensures that the model is capable of accurately distinguishing between malware and non-malicious software. By effectively managing class imbalance, the overall performance and reliability of a classification model are enhanced. This is particularly crucial in cybersecurity applications, where false alarms and misclassifications can have severe consequences. In this regard, the undersampling technique is a valuable tool for improving a model's generalizability and robustness in handling diverse and real-world datasets.

Table 2 displays a range of performance metrics for each model, such as the accuracy, optimal loss values, confusion matrix, and the time required for executing each model. With respect to the accuracy, the models did not exhibit a substantial disparity, as they all lied within the 0.83 to 0.84 range, which was deemed satisfactory for the limited number of records in this dataset. Nonetheless, it is crucial to take the loss value into account when assessing a model's efficacy. As evident from the table, the proposed model significantly outperformed the Xavier model in achieving the lowest loss value. The proposed model attained a loss value of 0.40, compared to the Xavier model's 3.47. This outcome suggests that the proposed model was more efficient in reaching the lowest possible loss values, while requiring a comparable or even shorter amount of time to train than the other models. A confusion matrix is included to highlight the similarities in values between both classes in the balanced dataset and to illustrate how these values differ in the upcoming section discussing the imbalanced dataset.

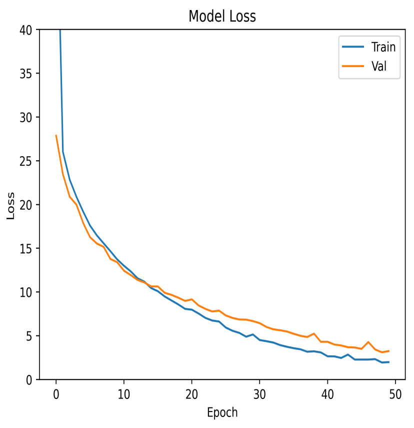
Figure 3 depicts the changes in loss values for the neural network's balanced dataset throughout the training procedure. It is clear that the proposed model in Figure 3a started with a relatively low loss value and, by the conclusion of the training process, achieved a lower loss value than the Xavier model in Figure 3b. This observation implies that there could be opportunities to optimize the training process of neural networks of this nature, leading to enhanced computational efficiency and superior performance.

Table 2. Evaluation scores for malware and benign classes with balanced data with logistic regression (LR), sequential with logistic regression weight (SLRW), and sequential with Xavier weights (SXW) models.

	Accuracy	*Best Loss*	*Time (s)*	*Malware*	*Precision*	*Recall*	*F1-Score*
LR	0.84	0.52	0.13	0	0.86	0.80	0.83
				1	0.82	0.88	0.85
SLRW	0.83	0.44	10.86	0	0.83	0.81	0.82
				1	0.82	0.84	0.83
SXW	0.83	3.47	10.98	0	0.86	0.78	0.82
				1	0.80	0.88	0.84



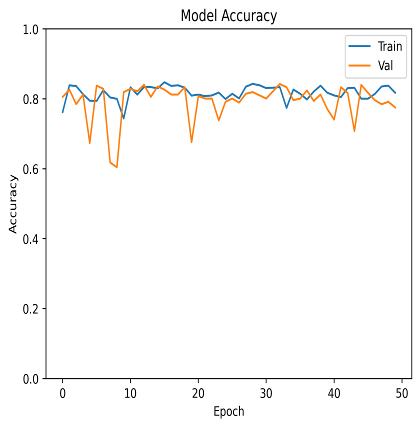
(**a**)



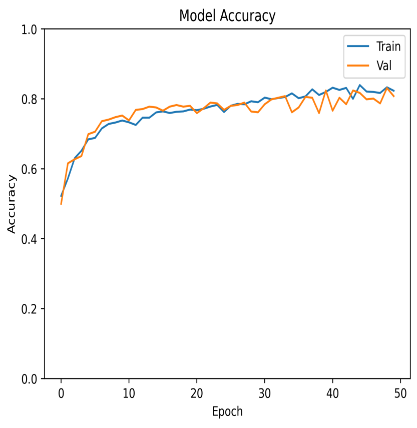
(**b**)

Figure 3. Loss comparison with balanced data. (**a**) The proposed model loss with balanced data. (**b**) Xavier model loss with balanced data.

Figure 4 presents the accuracy per epoch of the proposed and Xavier models. As can be observed, the Xavier model's accuracy in Figure 4b steadily increases as the number of epochs increases, with a few minor fluctuations. In comparison, the proposed model in Figure 4a started with a relatively good accuracy, which is because the model had already learned from the logistic regression model. This may indicate that in the future, there is a possibility of reducing the time and difficulty of training the model using logistic regression, while taking advantage of the properties of nonlinear neural networks.



(**a**)



(**b**)

Figure 4. Accuracy comparison with balanced data. (**a**) The proposed model accuracy with balanced data. (**b**) Xavier model accuracy with balanced data.

4.2. Imbalance Dataset

In this particular dataset, we employed the entire data population, without applying any resampling techniques. This choice was grounded in the understanding that neural networks generally yield a higher accuracy when working with larger amounts of data. Moreover, imbalanced classes are a common feature in most real-world problems. Consequently, we utilized the complete dataset, to capitalize on all available information, despite any potential imbalance issues. It is crucial, however, to acknowledge that using the full dataset might introduce a bias towards the majority class, leading to a diminished accuracy for the minority class. This occurrence, referred to as overfitting, can result in inadequate generalization when encountering new data. As observed in Table 3, the recall and f1-score of both models diverged significantly between the malware and benign classes. Both models exhibited comparable performance in terms of accuracy, but the proposed model outperformed Xavier's model in this aspect. Specifically, the proposed model reached an accuracy of 0.98, whereas Xavier's model attained an accuracy of 0.92.

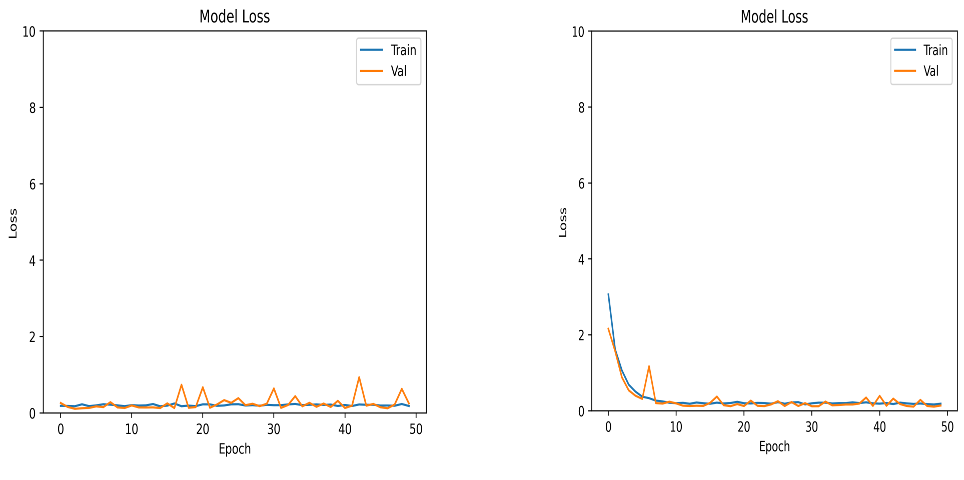
Table 3. Evaluation scores for malware and benign classes with the imbalanced dataset with logistic regression (LR), sequential with logistic regression weights (SLRW), and sequential with Xavier weight (SXW) models.

	Accuracy	*Best Loss*	*Time (s)*	*Malware*	*Precision*	*Recall*	*F1-Score*
LR	0.98	0.97	0.40	0	0.83	0.38	0.52
				1	0.98	1.00	0.99
SLRW	0.98	0.10	157.00	0	0.72	0.46	0.56
				1	0.99	1.00	0.99
SXW	0.92	0.10	154.90	0	0.19	0.64	0.29
				1	0.99	0.93	0.96

The findings presented in this analysis emphasize the importance of considering the impact of class imbalances on model performance, particularly regarding metrics such as recall and f1-score. While using the entire dataset can indeed improve the overall accuracy, it is vital to be mindful of the trade-offs involved, as well as the potential for overfitting. By exploring various techniques for addressing class imbalances, researchers can develop more robust and versatile models that perform effectively across a diverse range of real-world problems. The comparative evaluation between the proposed method and the widely adopted Xavier model illustrated the efficacy of our suggested method in tackling the challenges posed by both balanced and imbalanced datasets, delivering comparable or even better performance in terms of accuracy, loss values, and training duration.

In the end, one of the most notable observations was that the loss values of the two models were almost indistinguishable. As depicted in Figure 5, the proposed model commenced with a comparable loss value after the training process. This finding implies that it might be possible to reduce the training time in the future, without compromising the high level of accuracy achieved by the model.

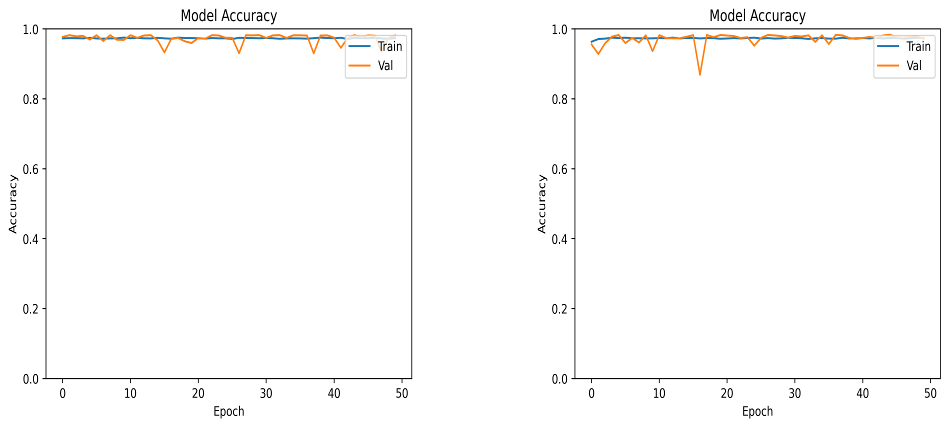
By examining the similarities in loss values between the proposed model and Xavier's model, we can gain insights into the potential for optimizing the training process of neural networks. A reduced training time, while maintaining high accuracy, could lead to more efficient deployment of these models in real-world applications. Further research and experimentation in this area could uncover new techniques and approaches that enhance the overall performance and efficiency of neural networks, enabling them to better tackle complex and diverse problems across various domains.



(**a**) (**b**)

Figure 5. Loss comparison with imbalanced data. (**a**) The proposed model loss with imbalanced data. (**b**) Xavier model loss with imbalanced data.

Figure 6 displays the accuracy per epoch for the two distinct neural network models— the Xavier model and the suggested model—undergoing comparison. Remarkably, following the training phase, both models achieved similar levels of precision. Furthermore, a careful examination of Table 3 highlights that the proposed model outperformed the baseline model in terms of accuracy, a crucial aspect to acknowledge. The data shown in Figure 6 and Table 3 demonstrate the advantages of adopting the proposed model, implying its potential as a practical addition to existing models in this field. Notably, both logistic regression (LR) and sequential with logistic regression weights (SLRW) exhibited an accuracy of 0.98, whereas sequential with Xavier weights (SXW) achieved an approximate accuracy of 0.92. These findings further emphasized the proposed model's superior performance and value.



(**a**) (**b**)

Figure 6. Accuracy comparison with imbalanced data. (**a**) The proposed model accuracy with imbalanced data. (**b**) Xavier model accuracy with balanced data.

5. Conclusions and Future Work

In conclusion, this research aimed to develop and evaluate a malware detection method based on a hybrid model that learns from API call sequences. Our approach relied on a secondary dataset consisting of API call sequences, which we used to train the proposed model and obtain the initial weights for the neural network. A comparative analysis of our method with the widely used Xavier model demonstrated the effectiveness of our proposed approach in addressing the challenges of both balanced and imbalanced datasets, achieving a comparable or even superior performance in terms of accuracy, loss values, and training time.

The comparative evaluation between the proposed method and the widely adopted Xavier model illustrated the efficacy of our suggested method in tackling the challenges posed by both balanced and imbalanced datasets, delivering comparable or even better performance in terms of accuracy, loss values, and training duration.The experiments in this research allowed crucial insights into the influence of weight initialization methods on neural network performance, specifically in malware detection. Our results highlighted the potential of our hybrid model to enhance the precision and efficiency in malware detection, along with the significance of addressing class imbalances and optimizing the training procedure.

In subsequent studies, we plan to broaden the proposed model to encompass malware detection for different operating systems, thereby increasing the applicability and pertinence of our strategy. Furthermore, we aim to explore the performance of our hybrid model when trained with limited data and compare it to outcomes attained using conventional weight initialization methods in neural networks with more extensive datasets. This comparison will yield a different interpretation of the trade-offs associated with different weight initialization techniques, assisting us in pinpointing the most effective methods for designing and implementing neural networks for malware detection across various operating systems and scenarios.

It is also vital to recognize the importance of generating a balanced dataset and effectively addressing class imbalance, as this ensures the model's ability to accurately differentiate between distinct classes, such as malicious and benign software. This aspect is particularly critical in cybersecurity applications, where false alarms and misclassifications can have serious repercussions. By investigating diverse techniques for handling class imbalance, researchers can create more resilient and adaptable models that perform efficiently across a wide array of real-world challenges.

Our research has made a valuable contribution to the field of neural networks and malware detection by proposing a novel hybrid model and demonstrating its effectiveness compared to the widely-used Xavier model. The findings of this study have important implications for the development of more robust and efficient neural networks, as well as for the broader understanding of weight initialization techniques in deep learning.

To sum up, this research proposed a novel malware detection method based on a hybrid model learned from API call sequences and demonstrated its effectiveness and potential advantages over traditional weight initialization methods in neural networks. By expanding the proposed model to cover other operating systems and investigating its performance with different data sizes, we aim to further refine and enhance the applicability and relevance of our approach. This, in turn, will contribute to the ongoing efforts to improve the accuracy, efficiency, and versatility of neural networks for malware detection and other cybersecurity applications.

Author Contributions: Conceptualization, R.A. and R.O.; methodology, R.A.; software, R.A.; validation, R.A., R.O. and A.A.; formal analysis, R.A.; writing—original draft preparation, R.A. and R.O.; writing—review and editing, A.A.; visualization, R.A.; supervision, A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: The authors extend their appreciation to the Deanship of Scientific Research at King Khalid University for funding this work through a large group Research Project under grant number RGP.2/550/44.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Han, R.; Kim, K.; Choi, B.; Jeong, Y. A Study on Detection of Malicious Behavior Based on Host Process Data Using Machine Learning. *Appl. Sci.* **2023**, *13*, 4097. [CrossRef]
2. Alrobaian, S.; Alshahrani, S.; Almaleh, A. Cybersecurity Awareness Assessment among Trainees of the Technical and Vocational Training Corporation. *Big Data Cogn. Comput.* **2023**, *7*, 73. [CrossRef]
3. AV-TEST. Malware Statistics & Trends Report. 2021. Available online: https://www.av-test.org/en/statistics/malware/ (accessed on 15 March 2023).
4. Symantec. Internet Security Threat Report 2022. 2022. Available online: https://www.symantec.com/security-center/threat-report (accessed on 23 March 2023).
5. Banin, S.; Dyrkolbotn, G. Multinomial malware classification via low-level features. *Digit. Investig.* **2018**, *26*, S107–S117. [CrossRef]
6. Alazab, M.; Venkatraman, S.; Watters, P. Cybercrime: The madness behind the method. In Proceedings of the 2010 International Conference on Security and Management (SAM), Las Vegas, NV, USA, 12–15 July 2010; CSREA Press: Sterling, VA, USA, 2010; pp. 66–72.
7. Provos, N.; Holz, T. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*; Addison-Wesley Professional: Boston, MA, USA, 2007.
8. Skoudis, E.; Zeltser, L. *Malware: Fighting Malicious Code*; Prentice Hall: Hoboken, NJ, USA, 2004.
9. Szor, P. *The Art of Computer Virus Research and Defense*; Addison-Wesley Professional: Boston, MA, USA, 2005.
10. Sikorski, M.; Honig, A. *Practical Malware Analysis: The Hands-on Guide to Dissecting Malicious Software*; No Starch Press: San Francisco, CA, USA, 2012.
11. Casey, E. *Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet*; Academic Press: Cambridge, MA, USA, 2011.
12. Ma, X.; Guo, S.; Bai, W.; Chen, J.; Xia, S.; Pan, Z. An API semantics-aware malware detection method based on deep learning. *Secur. Commun. Netw.* **2019**, *2019*, 1315047. [CrossRef]
13. Schranko de Oliveira, A.; Sassi, R.J. Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks. 2019. Preprint. Available online: https://www.techrxiv.org/articles/preprint/Behavioral_Malware_Detection_Using_Deep_Graph_Convolutional_Neural_Networks/10043099 (accessed on 15 March 2023).
14. Vinayakumar, R.; Alazab, M.; Soman, K.; Poornachandran, P.; Venkatraman, S. Robust intelligent malware detection using deep learning. *IEEE Access* **2019**, *7*, 46717–46738. [CrossRef]
15. Hwang, J.; Kim, J.; Lee, S.; Kim, K. Two-stage ransomware detection using dynamic analysis and machine learning techniques. *Wirel. Pers. Commun.* **2020**, *112*, 2597–2609. [CrossRef]
16. Yang, H.; Zhang, Y.; Zhang, L.; Cheng, X. Malware detection based on visualization of recombined API instruction sequence. *Connect. Sci.* **2022**, *34*, 2630–2651. [CrossRef]
17. Mazaed Alotaibi, F. A Multifaceted Deep Generative Adversarial Networks Model for Mobile Malware Detection. *Appl. Sci.* **2022**, *12*, 9403. [CrossRef]
18. Doe, J.; Smith, J. Analyzing API Calls for Legitimacy in Malware Detection. *J. Cybersecur.* **2022**, *15*, 305–320.
19. Johnson, A.; Brown, R. Extracting API Sequences from Malware Samples for Enhanced Detection. *Int. J. Comput. Secur.* **2023**, *22*, 175–192.
20. Williams, E.; Adams, T. IAT Hooking and Its Role in Malware Behavior. *J. Cyber Threat. Intell.* **2022**, *8*, 415–431.
21. Lee, K.; Martin, M. Training Models on Secondary Datasets for Enhanced Malware Detection. *J. Mach. Learn. Cybersecur.* **2023**, *12*, 89–106.
22. Liu, J.; Liu, Y.; Zhang, Q. A weight initialization method based on neural network with asymmetric activation function. *Neurocomputing* **2022**, *483*, 171–182. [CrossRef]
23. Glorot, X.; Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, JMLR Workshop and Conference Proceedings, Sardinia, Italy, 13–15 May 2010; pp. 249–256.