Malware Detection and Next-Action Prediction using Learning-Based Methods

Zahra Jamadi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Electrical and Computer Engineering) at

Concordia University

Montréal, Québec, Canada

January 2024

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Zahra Jamadi**

Entitled: **Malware Detection and Next-Action Prediction using Learning-Based Methods**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Paula Lago

_____ External Examiner
Dr. Ramin Sedaghati

_____ Supervisor
Dr. Amir G. Aghdam

Approved by _____
Dr. Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

_____ 2024

_____
Dr. Mourad Debbabi, Dean
Faculty of Engineering and Computer Science

Abstract

Malware Detection and Next-Action Prediction using Learning-Based Methods

Zahra Jamadi

In this thesis, we introduce a comprehensive framework that combines natural language processing (NLP) techniques and machine learning (ML) algorithms for the early detection and prediction of malware activities. The core contribution of our research is the innovative application of text classification methods, particularly Bi-LSTM neural networks and Bayesian neural networks (BNN), to interpret application programming interface (API) call sequences as natural language inputs. This novel approach enables us to predict upcoming malware actions, facilitating proactive threat identification and mitigation. Our first framework employs a Bi-LSTM model to predict the next API call, treating consecutive API calls as 2-gram and 3-gram strings. These are then processed using a Bagging-XGBoost algorithm, enhancing the model's ability to detect malware presence in its early stages. The second framework advances this concept by utilizing a Bayesian Bi-LSTM neural network. This model not only forecasts the future actions of running malware but also quantifies the uncertainty associated with each prediction, providing a probabilistic insight into potential malware actions. By providing the second and third most probable predictions, we significantly improve the reliability and performance of the decision-making process. Both frameworks are rigorously evaluated through simulations, demonstrating their effectiveness in malware detection and action prediction. Integrating these two approaches within a single thesis represents a significant step in applying NLP principles to cybersecurity, particularly in understanding and countering malware threats more effectively and efficiently.

Acknowledgments

I extend my heartfelt gratitude to Dr. Amir Aghdam, my thesis supervisor, whose guidance and unwavering support were instrumental in this research. I am profoundly thankful to my family, especially my mother, for their endless love and encouragement. Their belief in me made this achievement possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Literature Review

In today's fast-paced digital world, the threat of cyber attacks, especially from malware, is a growing concern [1]. Malware, which includes various types of harmful software, is created to disrupt or damage computers, networks, and data [2]. With the widespread use of the internet and the increasing reliance on digital technology, these threats have become more common and sophisticated [3]. This has made strong cybersecurity measures more important than ever [1].

Traditional methods for fighting malware, such as signature-based detection, struggle to keep up with these advanced threats [4]. On the other hand, some existing results in the control literature use the notion of *structural controllability* to address failure and robustness in networked systems (a conceptually similar problem) using a different approach [5], [6], [7]. These older techniques may still be useful to some extent, but as new types of sophisticated malware emerge, they fail to detect them as effectively. This challenge calls for new and innovative approaches, e.g., machine learning-based methods [2], [4]. In this work, application programming interface (API) calls made by malware are monitored and analyzed dynamically. Additionally, sequences of API calls have similarities to language structure, e.g., following certain patterns, and contextual relationships between tokens [8]. Therefore, the existing methods in natural language processing (NLP) to correlate the API call sequences.

Bidirectional long short-term memory (Bi-LSTM) neural networks have marked a significant

advancement in various scenarios, such as text classification [11], sentiment analysis [12], and even specialized applications like malware classification [13]. These networks are designed to process sequences in both forward and backward directions, effectively increasing the amount of information available to the network and enhancing the context understood by the algorithm [9]. This dual-direction processing allows Bi-LSTMs to capture long-term dependencies and local features of text, leading to state-of-the-art results on several benchmark datasets [10].

Detecting the malware at its early stages is of great importance and the capability of any malware detection method can be significantly enhanced by the analysis of feature importance. Through the analysis of 2-gram and 3-gram strings of API call sequences, critical insights into the behavior and interactions of malware with systems are obtained. These N-gram sequences, representing pairs and triplets of API calls, provide a detailed view of potential malicious activities, making them essential features in the detection model. The analysis of these features aids in understanding the sequence and context of API calls, which are often key indicators of malware presence. The Bagging XGBoost classifier has been selected for early-stage malware detection due to its great classification performance and its ability to conduct feature importance analysis [14]. Consequently, it is an effective tool for the early detection of cybersecurity threats.

Deterministic neural networks, while effective in many applications, exhibit certain limitations that can be particularly challenging in complex and dynamic fields like cybersecurity. One significant limitation is their inability to quantify uncertainty in their predictions [15]. Deterministic models provide specific outputs for given inputs without an inherent measure of confidence or probability, which can be a drawback in scenarios where understanding the reliability of predictions is crucial. Furthermore, deterministic models are prone to overfitting, especially when dealing with limited or highly complex datasets [16]. This problem can lead to poor generalization of unseen data, reducing the model's effectiveness in real-world applications [16]. They also often require extensive data for training, which might not always be feasible, especially in domains like malware analysis where data collection is challenging or expensive e.g. malware analysis domain.

Bayesian neural networks (BNNs) address these limitations by offering a probabilistic approach to learning and prediction, to quantify the uncertainty in their outputs [17]. This feature is particularly valuable in applications demanding high confidence in decision-making, such as cybersecurity,

where the cost of incorrect predictions can be very high. The Bayesian approach integrates priors and regularization through a probabilistic framework, which not only helps in mitigating overfitting and reducing model complexity but also improves the model's data efficiency and robustness [18]. This makes BNNs particularly suitable for scenarios involving small or noisy datasets and those susceptible to adversarial attacks [18].

In cybersecurity, where adversaries continuously evolve their tactics and malware becomes more sophisticated, the ability to make more confident decisions about malware detection and mitigation is crucial [19]. Bayesian Bi-LSTM networks, by capturing uncertainty in the features and behaviors associated with malware, enable the development of more robust and adaptive detection systems [20]. Modeling uncertainty helps cybersecurity specialists establish a measure of confidence in their decisions, thereby improving the overall efficacy of malware detection and action prediction. Despite their computational intensity and the need for specialized training techniques, the benefits of Bayesian neural networks, especially in uncertain and dynamic environments like cyber-physical systems, make them an invaluable tool in this type of applications[21].

1.2 Thesis Contributions and Organization

The contributions of this thesis is summarized in the sequel. Due to the Bi-LSTM neural network's capability of handling sequences of tokens mentioned in the previous subsection, in this work, the Bi-LSTM neural network is used for predicting the upcoming actions of ongoing malware, thereby enabling the early mitigation of the malicious software. The first framework developed for malware detection and next-action prediction exploits a Bi-LSTM neural network tailored for predicting the future actions of malware. This network treats sequences of API calls as natural language inputs by transforming them into a new dataset using N-gram features. The model receives the sequences of API calls made by the malware up to that time and predicts the upcoming actions by anticipating the next API calls that are going to be made by the malware. The capability of the model to predict the next API call enhances the potential for early mitigation of malware threats, showcasing a novel way of addressing cybersecurity challenges. The study further progresses by adopting a method where consecutive API calls are modeled as 2-gram and 3-gram strings, allowing

for the extraction of new features. These features are then processed using a Bagging-XGBoost algorithm to detect the malware at its early stage and identify the most important malicious activities of the malware. Such a method is instrumental in suggesting the existence of malware, enhancing the framework's ability to detect and respond to cybersecurity threats. The integration of these advanced methodologies demonstrates a comprehensive approach to malware detection, combining the strengths of neural networks and machine learning algorithms. This phase of the research significantly contributes to the field by providing an effective tool for detecting and analyzing malware activities.

Building on the advantages of BNN outlined in the previous subsection, this work further develops two distinct architectures of Bayesian Bi-LSTM neural networks for malware detection and next-action prediction. These specialized architectures are designed to exploit the full potential of Bayesian neural networks in handling the uncertainties inherent in predictions. For malware detection, the Bayesian Bi-LSTM architecture is optimized to identify and analyze complex patterns in data, enabling it to detect a wide range of malware behaviors, including sophisticated and previously unseen variants. This capability is crucial in an ever-evolving threat landscape where traditional deterministic models might fall short. For next-action prediction, the Bayesian Bi-LSTM model is engineered to predict potential future actions of a detected threat. This predictive ability is vital for proactive cybersecurity measures, allowing for timely and effective responses to mitigate cyber threats.

The second framework proposed for enhanced malware detection and next-action prediction outperforms the first one in both detection and next-action prediction. The novelty of this method lies in incorporating probabilities, enhancing the robustness and reliability of the framework. For malware detection, a Bayesian Bi-LSTM neural network is developed which not only improves the accuracy of the framework but also offers a deeper interpretation of each result, providing valuable insights into the detection process. For next-action prediction, a more complicated Bayesian Bi-LSTM neural network is designed and implemented. This new framework outperforms its deterministic neural network counterpart and provides a more reliable method for predicting the upcoming actions of the malware by considering the probability values and associated uncertainty with the most probable prediction along with the second and third most probable predictions made by

the BNN. Probabilistic insights into the model's decisions provide cybersecurity experts with the necessary information to make more confident decisions.

The rest of this thesis is organized as follows. Chapter 2 presents the proposed early malware detection and next-action prediction model which has been published at the 13th IEEE International Conference on RFID Technology and Applications [22]. Chapter 3 introduces the enhanced malware prediction and containment using a BNN which outperforms the previous framework by incorporating a probabilistic nature to the detection and next-action prediction pipeline. The combination of the first and second methods will be submitted as one paper to the IEEE Journal of Radio Frequency Identification. Finally, Chapter 4 presents conclusions and possible future research directions.

Chapter 2

Early Malware Detection and Next-Action Prediction

2.1 Abstract

In this paper, we propose a framework for early-stage malware detection and mitigation by leveraging natural language processing (NLP) techniques and machine learning algorithms. Our primary contribution is presenting an approach for predicting the upcoming actions of malware by treating application programming interface (API) call sequences as natural language inputs and employing text classification methods, specifically a Bi-LSTM neural network, to predict the next API call. This enables proactive threat identification and mitigation, demonstrating the effectiveness of applying NLP principles to API call sequences. The Bi-LSTM model is evaluated using two datasets. Additionally, by modeling consecutive API calls as 2-gram and 3-gram strings, we extract new features to be further processed using a Bagging-XGBoost algorithm, effectively predicting malware presence at its early stages. The accuracy of the proposed framework is evaluated by simulations.

2.2 Introduction

Malware is a term describing a malicious program that is installed on a platform such as a personal computer, harming the user by damaging the system, stealing information, or hosting the system for blackmail purposes [23]. The number of reported cyberattacks continues to increase, and new malware is produced by attackers. According to 2021 SonicWall Cyber Threat Report, internet of things (IoT) malware attack volume in the first six months of 2021 increased by 59% compared to the previous year [24]. Additionally, the AV-TEST Institute indicates that every day, 450,000 new malicious programs (malware) and potentially unwanted applications (PUA) are registered [25].

Figure 1 shows the distribution of the malware and PAUs from 2008 to 2023.

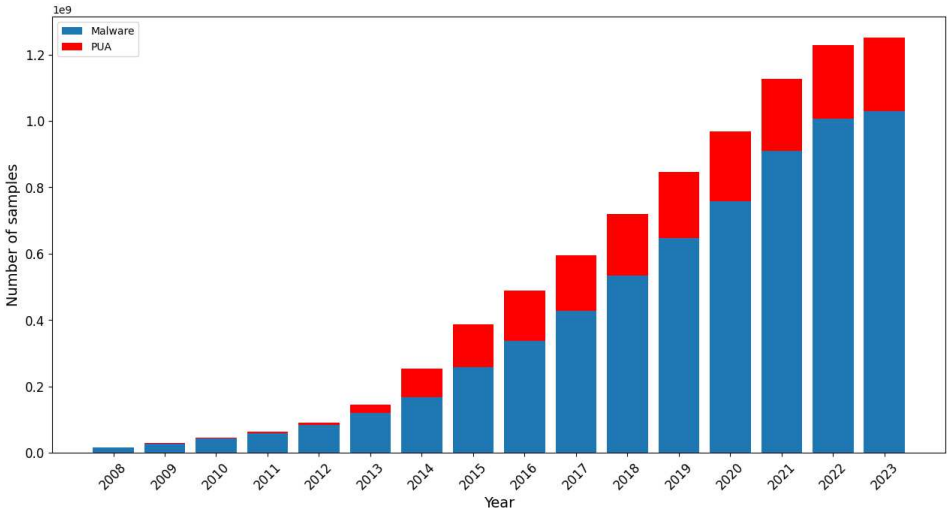

Figure 2.1: Malware and PUA samples from 2008 to 2023 [25]

Radio frequency identification (RFID) technology plays a crucial role in various industries, enabling automatic identification and tracking of objects using radio waves [41]. RFID systems typically consist of RFID tags attached to objects and RFID readers that communicate with the tags to exchange data [41]. To facilitate this communication, RFID middleware is used as an intermediary layer between RFID readers and the backend systems[42]. When malware infects the RFID middleware, it can modify the API calls made during the communication process, e.g., by manipulating the flow of data between the middleware and the backend systems [42].

As new families of malicious threats emerge and variants of malware continue to develop, conventional signature-based methods for detecting malware often fall short in identifying a large number of threats due to their reliance on known patterns [26]. Fortunately, learning-based methods are able to detect malware more efficiently since they can recognize and learn the patterns of previously unseen cyber-attacks [26].

Early malware detection and mitigation is an important task, especially for the types of malware that are costly to recover from [27]. It can save resources, minimize damage and protect sensitive information. One way to detect the malware at its early stage is to continuously monitor the application programming interface (API) calls made by the malware during its run-time and analyze them dynamically [28]. After early detection, it is desired to block the attack using a proper prediction strategy before it affects the other parts of the system.

A sequence of API calls can be modeled as a natural language processing (NLP) task because of their similarities, e.g., following a specific syntax and grammar, and the importance of context in understanding the meaning of a request [8]. Several studies have been conducted to detect malware by leveraging NLP techniques. Sundarkumar et al. [29] used text-mining and topic-mining techniques to use API call sequences for malware detection. Li et al. [29] built a joint representation of API calls to depict software behaviors and then implemented a Bi-LSTM model to learn the relationship between API calls in a sequence and performed malware detection. Liu et al. [30] employed several deep learning-based methods for malware detection based on API calls extracted from Cuckoo sandbox.

In this study, API call sequences are modeled as a natural language construct, and principles of NLP are utilized for early malware detection and next-step prediction. We first propose a framework for detecting malware at its early stage. For this purpose, sequences of API calls are modeled as 2-gram and 3-gram strings and used as new features. Bagging-XGBoost algorithm [31] is then used for malware detection and feature importance identification. In the second part of the work, we predict the malware's upcoming action(s). A bidirectional long-short term memory (Bi-LSTM) neural network which is a common method in text classification tasks is used to predict the next API call(s). The proposed method will help proactively detect and block the attack before it can cause any significant damage.

The rest of the paper is organized as follows. In Section II, we describe the datasets and the learning-based approaches used for malware detection and API call prediction. Section III presents the results and effectiveness of the proposed framework in detecting malware and predicting API calls. Finally, in Section IV, the contributions are summarized and potential directions for future research in this area are suggested.

2.3 Proposed Methodology

In this section, we first introduce the two datasets used in this study and will then describe the method used to detect the malware at its early stage and predict its upcoming action(s). The proposed framework in this study is represented in Figure 3.1.

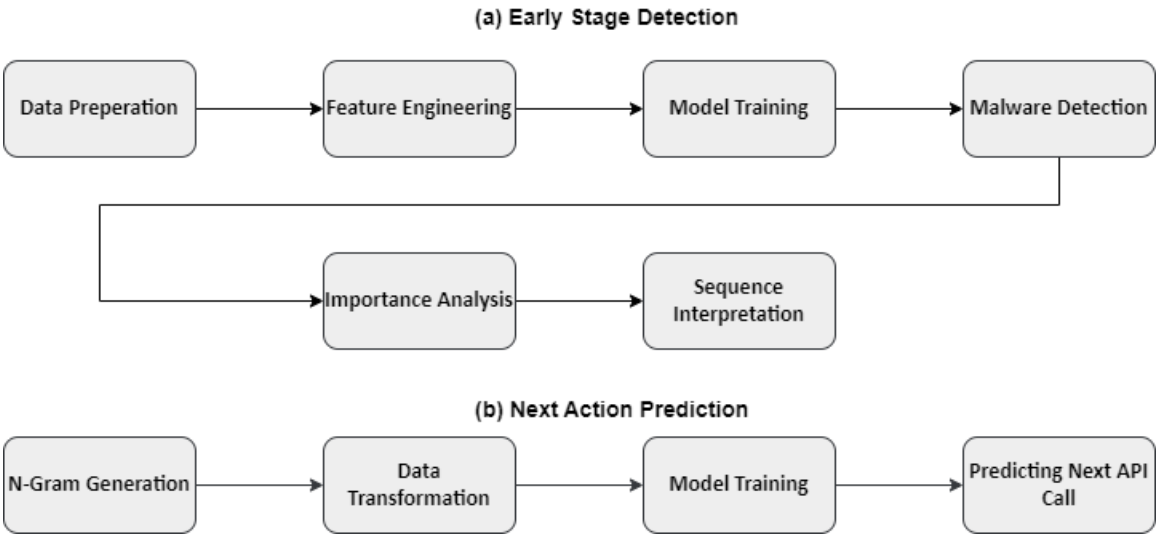

Figure 2.2: Proposed framework for a) malware detection and b) next-action prediction

2.3.1 Datasets

The first dataset used in this study contains 42,797 malware and 1,079 goodware API call sequences [32]. Each sequence contains the first 100 non-repeated consecutive API calls associated with the parent process, with each API call assigned a numerical integer value ranging from 0 to

306. The dataset's large number of malware samples provides a diverse range of malicious behaviors for the models to learn from, while the inclusion of goodware sequences allows the models to differentiate between benign and malicious behavior. This dataset is used for malware detection.

The second dataset comprises 7,107 malware samples from various families, with each malware API call sequence assigned a numerical integer value ranging from 0 to 341 [33]. This dataset's diversity of malware families enables a comprehensive analysis of the proposed method's effectiveness across different types of malware, while a large number of malware samples ensures that the models are trained on a broad range of malicious behaviors, enabling them to detect and predict various threats.

2.3.2 Early Malware Detection

We use the first dataset [32] for detecting the malware at its early stage. Due to the importance of early malware detection, API calls for this dataset are only extracted from the parent process, which is primarily responsible for initiating other processes. Furthermore, since the aforementioned dataset is not quantitatively balanced in terms of goodware and malware samples, the random oversampling method is used to increase the number of goodware and balance the dataset for both train and test sets. To ensure there is no data leakage, oversampling is performed after the dataset has been split into 75% for training, 10% for testing, and 15% for evaluation. This approach maintains the integrity of the testing and evaluation processes, as it prevents the model from being exposed to test or evaluation data during training. Our objective is to also identify malicious activities as the best indicators of the existence of malware. To this end, sequences of API calls, modeled as 2-gram and 3-gram strings of consecutive API calls, are tokenized and used as new features. The most important features are then identified.

Let extreme gradient boosting (XGBoost) algorithm [34] be used for the binary classification, and subsequently, early detection of the malware. XGBoost is a popular and powerful machine learning algorithm for classification, regression, and ranking tasks. It is an ensemble method that combines the predictions of multiple decision trees (DT) to make final predictions [14]. We then build XGBoost bagging to improve the accuracy and robustness of malware detection tasks. The bagging method in XGBoost, an important feature, involves the aggregation of multiple decision

trees, with each tree addressing the shortcomings of its predecessors. This iterative refinement is key to enhancing the model's predictive accuracy, especially in complex tasks like malware detection [34].

The XGBoost classifier is also used to rank the importance of 2-grams and 3-grams of API calls in terms of their prediction capabilities. In this context, the algorithm's ability to efficiently manage large feature sets, such as these n-grams, stands out. It assigns incremental weight to misclassified instances, focusing more on challenging segments of the data in subsequent trees [35]. This strategy is instrumental in improving the detection of sophisticated, previously undetected malware types. Three XGBoost classifiers with learning rates of $0.01, 0.05, 0.1$, maximum depth of $4, 3, 5$, and number of estimators equal to $100, 200, 300$, respectively, are used to perform the feature extraction and detection tasks. Hyperparameters for each classifier were selected using grid search.

2.3.3 Next Action Prediction

This subsection presents the main contribution of this work. To the best of the authors' knowledge, no prior research has been reported on predicting upcoming malware actions by predicting the next APIs. We address this problem by modeling the sequence of API calls as a natural language input. We then feed this sequence to Bi-LSTM model to predict the next APIs one by one, which are indicative of the malware's next actions. By predicting the next steps of the attack, proper mitigation techniques can be used to prevent the malware from affecting the other parts of the system.

Bi-LSTM is a type of recurrent neural network capable of capturing the dependencies between the elements of sequential data. Since it processes the input sequence in both forward and backward directions, it can efficiently identify the words before or after another word [28]. To ensure that the developed model is able to predict the next API calls of a given sequence efficiently, Bi-LSTM model is tested on both datasets. The N-gram method is applied to both datasets to convert the API calls sequence into a feature structure which helps the Bi-LSTM model learn the relationship of the API calls with each other [36]. N-gram generation is carried out after splitting the dataset into training, evaluation, and testing sets to ensure no data leakage.

To build N-gram features for a sequence of API calls, we consider a set of n consecutive API calls, where n ranges from 2 to the length of the sequence minus 1. For each subsequence, we use

the last API call as the label. For example, to generate the first data point, we take the first two API calls in the sequence and use the third API call as the label.

An example of N-gram features for a subsequence of one of the malware samples containing 7 API calls is given in Table 2.1.

Table 2.1: N-grams for a subsequence of 7 numerical API calls.

Data point	Corresponding label
$[220, 233]$	$[237]$
$[220, 233, 237]$	$[220]$
$[220, 233, 237, 220]$	$[233]$
$[220, 233, 237, 220, 233]$	$[290]$
$[220, 233, 237, 220, 233, 290]$	$[260]$

N-grams can capture the patterns in the sequence of API calls, indicating certain behaviors or outcomes. We can then use these N-gram features to train the Bi-LSTM model for predicting the next API call in a given sequence. The predicted API call is then added to the input sequence and fed to Bi-LSTM network. In this way, we are able to predict multiple API calls which indicates the future action of a malware.

The proposed Bi-LSTM neural network has four layers. The first one is an embedding layer for converting the input sequence into a dense vector capturing its semantic and contextual information. The second one is a Dropout layer with a 30% rate to prevent the model from overfitting. The third one is a Bi-LSTM layer with a size of 150 neurons. Finally, the last one is a dense layer which completes the classification task. Adam optimization algorithm with a learning rate of 0.01 is employed. The cost function used in this network is categorical cross-entropy, which is widely used in text classification problems. The aforementioned configuration and hyperparameter values are obtained after several experiments to achieve the highest level of performance.

2.4 Experimental Results

The proposed approach can anticipate the malware actions by predicting the next API calls that the malware makes one at a time. Figures 3.3 and 2.4 represent the performance of the Bi-LSTM model in predicting 10 subsequent API calls for two input sequences chosen randomly from the test

sets of the first and second dataset [32], [33]. In both figures, the sequence presented on the top is the predicted sequence, while the one below is the ground truth. The green solid and red dashed circles indicate the correct and wrong predictions, respectively. The results show that all but one API call are predicted correctly in both cases.

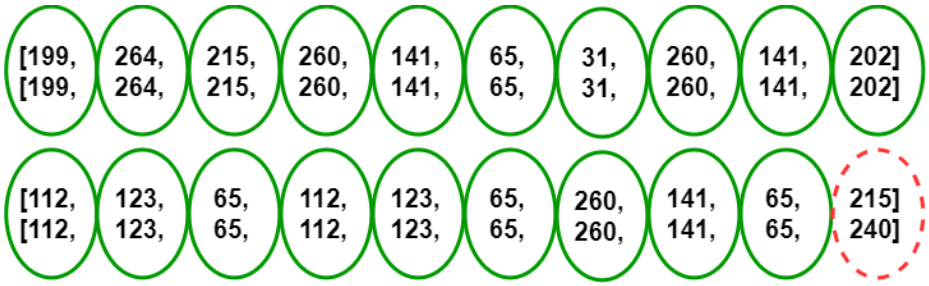

Figure 2.3: Next 10 API calls predicted by the Bi-LSTM model, along with the ground truth.

Figure 2.4: Next 10 API calls predicted by the Bi-LSTM model, along with the ground truth.

The performance of the Bi-LSTM model for each dataset is evaluated by measuring the accuracy, precision, recall and F1-score which are reported in Table 3.4. In this problem, label imbalance is a concern due to the unequal occurrence of different API calls when a process is running. Some API calls may be more common in both benign and malicious programs, while others might be rare. To overcome this challenge, the weighted average of all labels is reported, which takes the number of true instances for each label into account. Reported results show that the model is performing more accurately in predicting the next API call for the first dataset compared to the second one. This might be due to two factors. First, the first dataset [32] contains a more diverse repository of malware API call sequences (42797 samples) compared to the second dataset [33] (7107 samples). This diversity can help the model to generalize more efficiently to unseen patterns of API call sequences. The second reason could be related to the length of API sequence. The first dataset [32] contains the first 100 non-repeated consecutive API calls and the second dataset [33] contains all

the API calls made during the run-time. The Bi-LSTM model is performing better when trained on shorter sequences of API calls.

Table 2.2: Accuracy, precision, recall and F1-score of the Bi-LSTM model predicting the next API call using first and second dataset [32], [33]

	Dataset 1	**Dataset** 2
Accuracy	93.62%	88.80%
Precision	93.58%	88.50%
Recall	93.62%	88.80%
F1-score	93.52%	88.48%

To evaluate the prediction ability of the Bi-LSTM network, we obtained the ROC score for each API call label. This is a commonly used evaluation metric for classifiers, measuring the area under the ROC curve, which plots the true positive rate (TPR) against the false positive rate (FPR). We identified API calls for which the Bi-LSTM network was struggling to make correct predictions. We observe that for both datasets, these APIs are the ones that are not commonly called during the run-time of the samples, and the model struggles to predict them accurately when presented with new, unseen data samples. Table 2.3 displays a list of rarely used APIs that are present in both datasets.

Table 2.3: List of Rare API Calls

No.	**API Call**
1	CopyFileW
2	GetUserNameExA
3	GetDiskFreeSpaceExW
4	SHGetSpecialFolderLocation
5	HttpSendRequestA
6	InternetGetConnectedState
7	sendto
8	RtlDecompressBuffer

For early detection of malware, we generate new features containing two and three consecutive API calls extracted from the parents' process. We then extract 10 most important features identified by the XGBoost classifier and compared their occurrence in malware samples and benign samples. Table 2.4 indicates the most important features and their occurrence in goodware and malware

samples. Class 0 and class 1 correspond to bengin and malware samples respectively. The features that are more often in malware samples are then investigated to identify any potential malicious activities that occur during the run-time of malware. Table 2.5 presents these features ordered by their importance.

Table 2.4: Top 10 Important sequences of API call sequences and corresponding class frequencies

API Calls	**Class 0 Count**	**Class 1 Count**
[240, 117, 82]	1779	12287
[117, 297, 199]	0	6345
[35, 208, 240]	625	14365
[199, 264]	17349	16475
[215, 37]	549	5129
[114, 215]	14582	3064
[117, 215, 260]	12165	934
[215, 37, 158]	397	4993
[202, 260]	13248	8108
[117, 215, 89]	44	2276

Table 2.5: Top 5 important malicious API call sequences indicating the existence of malware

	API Call Sequence
1	LdrLoadDll, LdrGetProcedureAddress, CertOpenSystemStoreA
2	LdrGetProcedureAddress, NtCreateFile, SetFilePointer
3	GetSystemMetrics, NtAllocateVirtualMemory, LdrLoadDll
4	NtClose, NtOpenKeyEx, NtQueryValueKey
5	LdrGetProcedureAddress, NtClose, NtDuplicateObject

The malware corresponding to the first API call sequence specified in Table 2.5 is to load a malicious Dynamic Link Library (DLL) into memory, retrieve the address of a specific function within the DLL, and obtain a certificate from the system certificate store. The second suspicious API call sequence is when the malware locates a specific function within a loaded module, creates a new file on the system, and manipulates the file pointer to write data to a specific location within the file. The third sequence is extracting system information, allocating memory in the virtual address space, and loading a DLL into memory. The fourth is manipulating the Windows Registry, which is a hierarchical database storing configuration settings and other system information. The last one is locating and duplicating a handle to an object, such as a file, registry key, or process, in order to gain access to it and perform some malicious actions.

Bagging-XGboost is then used to classify an ongoing process as malware or goodware. The results show that the proposed malware detector framework is able to detect the malware at its early stage with high accuracy. The precision score of 92.70% shows that the model has a false alarm rate of 7.3% which is indicative of the well performance of our early malware detector. The full performance metrics of the proposed XGBoost bagging are presented in Table 2.6. To evaluate the performance of our early-stage malware detection technique, it is compared with other methods in the literature which use sequences of API calls. Table 2.7 reports the comparison between the accuracy of our model with other methodologies.

Table 2.6: Accuracy metrics of XGBoost bagging model

Accuracy	95.85%
Precision	92.70%
Recall	99.56%
F1-score	96.00%

Work	**Algorithm**	**Accuracy**
Sai et al. [37]	DT	89.89%
Xue et al. [38]	MLP	91.57%
Avci et al. [39]	Bi-LSTM	93.16%
Catak et al. [40]	Two layer LSTM	95.00%
Present work	XG-Boost Bagging	95.85%

Table 2.7: Comparison of Model Accuracies

2.5 Conclusions and Future Work

This paper presents a framework for early-stage malware detection and mitigation by treating API call sequences as natural language inputs and employing text classification methods, specifically a Bi-LSTM neural network, to predict the next API call. This study demonstrates that Bi-LSTM, a neural network architecture commonly used in NLP tasks, is an effective method for predicting API calls due to the similarities between the sequence of API calls and natural language structure [8]. The model is able to predict the next action of the malware by predicting the next API calls that are most likely to occur, one at a time, and allow early mitigation. Additionally, by modeling consecutive API calls as 2-gram and 3-gram strings, we extract new features to be further

processed using a Bagging-XGBoost algorithm. This enables us to identify the sequence of API calls and their corresponding activities that may suggest a malware exists.

For future work, one can investigate alternative NLP techniques, such as transformers and attention mechanisms, to enhance the malware detection and prediction capabilities of the framework. Moreover, evaluating the real-time performance of the proposed framework for online malware detection and mitigation could provide insights into its potential for practical deployment in real-world cybersecurity scenarios. Finally, this study only explored the prediction of API calls one step at a time. One direction could involve extending the framework to multistep-ahead prediction of API calls.

Chapter 3

Enhanced Malware Prediction and Containment using Bayesian Neural Networks

3.1 Abstract

In this paper, we present an innovative framework to enhance the accuracy and reliability of malware detection and the action prediction of the malicious software. Our approach leverages the power of natural language processing (NLP) techniques, coupled with the advantages of Bayesian neural networks (BNN). The main objective of this work is to forecast the future actions of running malware by predicting the next application programming interface (API) call given a sequence of API calls within the run time of malware. To this end, the sequence of API calls is processed as natural language inputs, transforming them into N-grams, and passing them to a Bayesian bidirectional long short-term memory (Bi-LSTM) neural network. This framework provides us with probabilistic insights into the actions an ongoing malware could take. The result is also extended to malware detection by employing a different Bayesian Bi-LSTM neural network architecture. The proposed Bayesian Bi-LSTM neural proves effective in making reasoned decisions by taking the probability

values and the uncertainty associated with each prediction. For both detection and next-action prediction, the second and third most probable predictions are also taken into account to improve the reliability and performance of the decision-making process. The accuracy of the proposed framework is evaluated by several simulations.

3.2 Introduction

In today's digital world, with the wide use of computers and the internet, cybersecurity is becoming increasingly critical in various sectors [1]. Malware is a term for "malicious software," which comes in many forms and is designed to disrupt, corrupt, or exploit computer systems, networks, and data [1]. As the digital world keeps changing, new variants of malware tend to rise [2]. The consequences of malware running on the user's system without being notified could be devastating. Such consequences include financial losses, data breaches, system disruptions, and even threats to national security [43]. Therefore, developing methodologies for early detection and mitigation of malware is absolutely crucial.

The ability to predict the malware's next actions is pivotal in effective mitigation. It can minimize potential damage and reduce the malware's impact on the system, resulting in a resilient defense against malware threats. This goal could be achieved by dynamically analyzing application programming interface (API) calls being made by the malware. Sequences of API calls share similarities with natural language input, as they can both be represented as sequences of tokens. Additionally, these sequences exhibit contextual and semantic relationships between their tokens, akin to the relationships found in words within language [8]. Therefore, natural language processing (NLP) methods can be applied for processing the sequences of API calls, and language models (LM) can be used for analysis and interpretation. LMs are exploited to find the next most probable word given a sequence of words by calculating the probability of a sequence of words [44]. Thus, the task of malware detection is similar to sentiment classification, where we classify the behavior of a program as either benign or malicious based on its API call sequence. We can also predict the upcoming action by predicting the next API call, which is similar to a multi-label text classification problem.

In the context of malware detection and its next action prediction, conventional methods such as signature-based detection and static file analysis have been used for many years [4]. However, these methods often fall short when it comes to detecting new and unknown malware [45]. Learning-based methods such as deep learning have emerged as a promising approach for malware detection and prediction [2]. Recent research has shown that deep learning methods can achieve better accuracy than traditional machine learning techniques and can learn to efficiently detect and classify new malware samples [2]. However, there are some limitations associated with deterministic neural networks, which could be addressed using Bayesian neural networks (BNN). Unlike deterministic networks, BNNs offer probabilistic predictions that quantify uncertainty, making them valuable in applications demanding confidence estimates [46]. They mitigate overfitting and model complexity by incorporating priors and regularization through a probabilistic framework. BNNs are data-efficient and robust, making them more suitable for small or noisy datasets and adversarial attack scenarios [18]. They also streamline hyperparameter tuning and offer richer model interpretability [18]. However, they do require specialized training techniques and can be computationally intensive [47]. Despite these challenges, BNNs are vital in fields where uncertainty modeling is critical [46].

In the field of cybersecurity, adversaries are continually evolving their tactics, and malware is becoming increasingly sophisticated [19]. According to an article published by the Information Systems Audit and Control Association (ISACA) [19], in 2022, 76% of organizations were targeted by a ransomware attack, of which 64% were infected. As a result, the ability to make more reliable malware detection and mitigation is advantageous. BNNs are capable of capturing uncertainty in the features and behaviors associated with malware, enabling more robust and adaptive detection systems. By modeling uncertainty, BNNs can help cybersecurity specialists establish a measure of confidence in the decisions they make.

To the best of the authors' knowledge, there are no results reported in the literature which treat the API calls made by malware at runtime as tokens and natural language inputs and analyze these inputs using BNNs for detecting malware and predicting its future actions. However, in prior research, there are a few works conducted using Bayesian neural networks for malware analysis. The authors in [48] present proposed an Android malware detection system using Bayesian classification based on permission features extracted via static analysis. The authors in [49] developed a

non-parametric Bayesian model for malware detection and classification allowing probabilistic data representation where a feature selection technique is given to improve the model's efficiency. This approach is not only used in the realm of cybersecurity but also exploited in other applications such as text classification. The authors in [50] categorize ransomware into nine distinct classes using artificial neural networks and Bayesian networks. They collect ransomware samples and create a learning database by extracting common strings from system calls. The work [51] presents a Bayesian network for malware detection, where optimized features are determined using a feature selection model. The features are then exploited to develop the predictive analytics model. The resultant Bayesian network outperforms traditional machine learning models, e.g. support vector machine (SVM) and K-nearest neighbors (KNN).

The contributions of this paper can be summarized as follows. In the first part of the study, a Bayesian Bi-LSTM neural network is designed and implemented for malware detection. Sequences of API calls are treated as a natural language input. Depending on the length of the input sequence used for the detection and uncertainty measurements, different results (including false alarm rates) can be achieved. Each different result can be interpreted according to the probabilistic nature of the model and uncertainty measurements. In the second part of the work, sequences of the API calls are transformed into a new dataset using N-gram features. Another architecture for a Bayesian Bi-LSTM neural network is developed to predict the upcoming action of a running malware by predicting the next API calls. For this part, the less probable predictions made by the network are also taken into account to improve the performance and the reliability of the framework.

The rest of the paper is organized as follows. Section II presents the details about the theory and the implementation of the developed Bayesian Bi-LSTM neural network. Simulations are given in Section III to validate the findings. Finally, in Section IV, the contributions are summarized, and directions for future research are suggested.

3.3 Bayesian Neural Network

Machine learning (ML) methods have been widely used in applications such as NLP, time-series analysis, and finance due to their ability to make accurate predictions and classifications [52]. However, they are highly prone to overfitting, making them poor in generalization and handling unseen data, especially when trained on the limited size of training data [52]. BNNs involve probability and uncertainty within the Bayesian framework [52]. In Bayesian statistics, probability is used to express beliefs about the likelihood of events or parameters, and these beliefs are updated as new evidence (data) becomes available [53]. Therefore, a BNN uses a prior distribution to formulate a posterior distribution over the network's parameters [52]. To this end, one can write:

$$p(\theta|D) = \frac{p(D|\theta) \cdot p(\theta)}{p(D)} \tag{1}$$

where $p(\theta|D)$ is the posterior probability, representing the probability of the parameter θ given the observed data D. The likelihood $p(D|\theta)$, on the other hand, describes the probability of observing the data D given a specific parameter value θ. Furthermore, $p(\theta)$ denotes the prior probability, which is the initial belief or probability distribution for the parameter θ before observing the data. Finally, $p(D)$ is the marginal likelihood or evidence, which is the overall probability of observing the data D, regardless of the specific parameter value.

There are several Bayesian inference algorithms to approximate the posterior distribution over the parameters of the Bayesian model, including Markov chain Monte Carlo (MCMC), variational inference, and Bayes by backpropagation [54]. For higher computational efficiency and scalability, variational inference is used in this research to estimate the posterior distribution of the network's parameters [54]. It is a powerful technique that treats the weights and biases of the network as random variables to capture the uncertainty associated with each prediction [54].

A BNN aims to model not just point estimates of parameters, but full probability distributions over them, allowing one to capture uncertainty. The posterior distribution over the network's parameters is inferred from the data. Variational inference approximates the true but often intractable posterior distribution with a simpler parameterized distribution, e.g. Gaussian, Cauchy and exponential distributions, to make this inference tractable [52]. In our study, the Gaussian distribution is

used for modeling the posterior distribution over the parameters of our network.

Training a BNN includes the following steps:

- Define a BNN architecture, specifying the prior distributions for the parameters.
- Specify the type of posterior distribution according to the nature of the problem and data characteristics.
- Use variational inference to approximate the posterior distribution of the parameters by optimizing the variational parameters to minimize the KL divergence. During this step, sampling from the approximate posterior allows for capturing the model's uncertainty in predictions.
- Use a loss function, including a likelihood term, to train the model on your data.

According to a study conducted by Google Brain [55], the integration of Bayesian layers within a neural network architecture enables the estimation of distributions over the layer's weights and biases. A Bayesian layer can easily be integrated with the deterministic layers inside a neural network architecture, enabling both deterministic and stochastic components [55]. Exploiting Bayesian layers incorporates uncertainty within the established semantics of deep learning [55]. In this research, we use a Bayesian layer as the output layer of a Bi-LSTM neural network. We develop the Bayesian layer using TensorFlow probability library. The optimization process in such a network is more complicated than in traditional neural networks, as it involves not only the usual back-propagation and weight adjustments seen in traditional neural networks but also the optimization of probabilistic parameters in the Bayesian layer.

Firstly, our Bi-LSTM processes the input data through both forward and backward layers during the forward pass, capturing complex dependencies in the sequence data. The output of this Bi-LSTM is then fed into the Bayesian layer. Instead of having fixed weights, however, the Bayesian layer possesses distributions of weights. This means that during training, we are not just learning specific weight values but rather the parameters (mean and variance) of the distributions from which these weights are drawn. The optimization process involves Bayesian inference, where we update our beliefs about the model parameters given the data. Typically, this consists of calculating the posterior distribution of the weights. However, exact Bayesian inference is often computationally

intractable for complex models like neural networks. Therefore, variational inference is used for posterior approximation.

In variational inference, we define a distribution with its own set of parameters. Then, we adjust these parameters to make the variational distribution as close as possible to the true posterior. The closeness is measured using the Kullback-Leibler (KL) divergence, for which the formulation is given as follows:

$$\min \mathrm{KL}(P \,||\, Q) = \int P(x) \log \left(\frac{P(x)}{Q(x)} \right) dx \tag{2}$$

where $P(x)$ is the true probability distribution, and $Q(x)$ is the estimated distribution. The training involves minimizing a loss function comprising not only the prediction error, which is binary and categorical cross entropy for malware detection and next-action prediction, but also this KL divergence term. This additional term acts as a regularizer, encouraging the model to maintain a balance between fitting the data and not being overly confident in its predictions.

The backpropagation process in this setup involves gradients not only with respect to the weights of the Bi-LSTM layers but also concerning the parameters of the variational distribution in the Bayesian layer. The optimization is carried out using the Adam optimizer, which is more challenging than traditional neural networks due to the involvement of probabilistic parameters. Ultimately, this advanced optimization process allows our Bayesian Bi-LSTM model to make reliable predictions while also providing a measure of uncertainty. This is crucial in many real-world applications where decisions are subject to uncertainty. The steps taken by the Bayesian layer in our implementation are demonstrated in Figure 3.1.

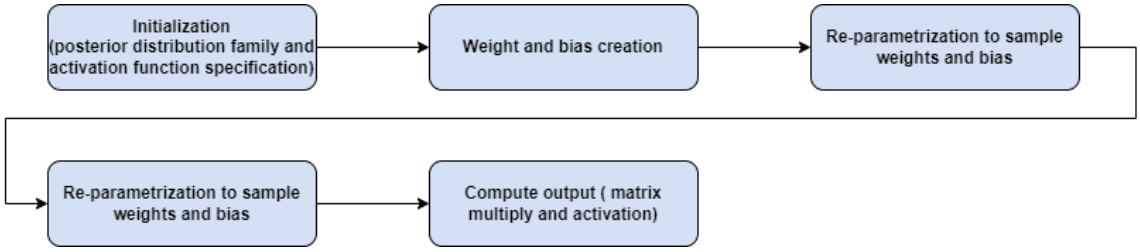

Figure 3.1: Bayesian dense layer algorithm flowchart

3.4 Proposed Methodology

In this section, we begin by introducing the datasets used for detection and next-action prediction, detailing the preprocessing steps applied to these datasets. Subsequently, we describe the Bayesian Bi-LSTM network architectures used for both detection and next-action prediction. The comprehensive flowchart of the proposed framework can be found in Figure 3.2.

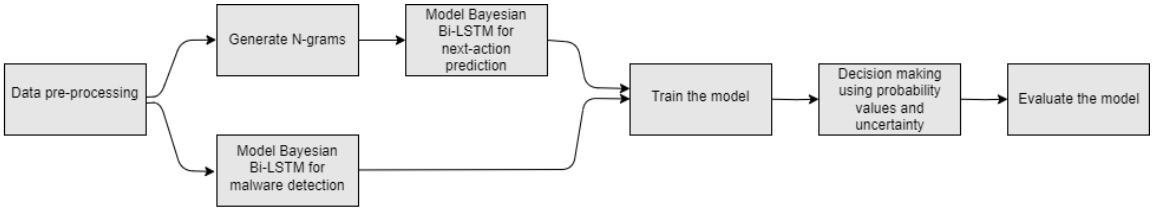

Figure 3.2: Overview of the implemented pipeline in this study for malware detection and predicting its upcoming action

3.4.1 Datasets

Two different datasets are used for detection and next-action prediction. The first one [56] was extracted from a larger dataset documented in [57]. It comprises 1,314 goodware and 2,626 malware API call sequences, with each API call assigned a numerical integer value ranging from 0 to 306. Due to the copyright restrictions related to API calls in goodware application runtime, publicly available datasets for malware behavioral analysis are scarce. However, the number of malware and goodware samples in this dataset is reasonable, eliminating the need for data augmentation or over-sampling to achieve dataset balance.

The second dataset, exclusively composed of API call sequences associated with malware, is employed in the latter part of our research, specifically for next-action prediction [58]. This dataset comprises 7,107 malware samples, with each API call mapped to a numerical integer ranging from 0 to 341, covering a diverse range of malware samples and providing the model with varied data for enhanced predictive capabilities.

3.4.2 Malware Detection

We use the first dataset [56] for probabilistic malware detection. Due to the importance of making probabilistic detection and uncertainty estimation when it comes to malware detection, we developed a Bayesian Bi-LSTM for this purpose by integrating a Bayesian dense layer as the output layer of a Bi-lSTM neural network. A Bi-LSTM neural network is exploited for processing sequential token data due to its ability to capture long-range dependencies and contextual information from both past and future, making it suitable for various natural language processing tasks [59]. Depending on the malware detection's time sensitivity, we can make decisions at various stages during the malware's runtime. Therefore, we can adjust the length of data collected during the malware's execution accordingly. Early-stage detection, for example, can be based on the initial API calls observed in the malware's runtime. Since we can measure probability values and uncertainty, we can interpret the results obtained for each decision. Cybersecurity specialists can decide about the timing thresholds and the amount of data collected during the runtime for malware detection based on different factors, including the potential number of infected systems, potential consequences of delayed mitigation, or the type of infrastructure being targeted.

The proposed Bayesian Bi-LSTM network for malware detection has four layers. The first is an embedding layer that captures the similarities among the API calls in the input sequence. The second layer is a Bi-LSTM layer with 200 LSTM units. The third layer is a global average pooling layer used for dimensionality reduction, and the fourth one is a Bayesian dense layer with one neuron and the sigmoid activation function to handle binary classification. The posterior distribution family considered for weights and bias is Gaussian. The choice of prior values for the parameters of the Bayesian layer is of great significance. A poor choice of initial values for the mean and standard deviation in the Gaussian distribution could lead to the divergence of the model due to high KL divergence; hence, the posterior distribution must closely match the prior values. In other cases, poor initial values could lead to slow convergence or a result not sufficiently close to the local minimum [18]. According to [18], the prior values for the weights and biases of the Bayesian layer in the case of language modeling and speech recognition can be chosen by using the average and standard deviation of the weights and biases of the last layer of a comparable neural network that

has fully or half converged. Thus, by considering the average and standard deviation of the weights and biases of the dense layer belonging to the last layer of a fully converged Bi-LSTM neural network trained on the same dataset, the initial values for the weights mean, bias mean, weights standard deviation and bias standard deviation are chosen as -0.004, 0.004, 0.08 and 0, respectively. Notably, we only have one bias value for the Bayesian layer since it only has one neuron for binary classification tasks.

The model's development involves systematically exploring architecture and hyperparameters to achieve optimal performance through multiple experiments. We employ the binary cross-entropy cost function throughout this iterative process, as commonly used in binary classification tasks. The optimization of our model leverages the Adam optimizer with an initial learning rate of 0.001. Additionally, we integrate a learning rate scheduler and an early stopping mechanism to ensure high accuracy. Additionally, we configure a batch size of 64 (an essential aspect of our optimization strategy) for a suitable balance between computational efficiency and model effectiveness.

3.4.3 Next Action Prediction

To the best of the authors' knowledge, aside from prior work of the same authors [22], there is no reported result on malware action prediction by analyzing the sequence of API calls. For a probabilistic prediction approach, which accounts for uncertainties, we design and implement a Bayesian Bi-LSTM network to tackle the underlying problem. The proposed architecture has five layers. Similar to the malware detection phase, the first layer of the proposed architecture is an embedding one. However, two Bi-LSTM layers are utilized instead of one, each with 150 LSTM units. The reason for considering two layers is the complexity of the task, similar to a multi-class classification. Given an input of API call sequences, predicting the following API call is like finding the next API call label among 342 different classes. It is more difficult for the Bayesian Bi-LSTM neural network to handle this task. Therefore, using two Bi-LSTM layers could improve performance and accuracy for the given task. The third layer is a global average pooling layer for dimensionality reduction. The last is a Bayesian layer with 342 neurons and a Gaussian distribution for each weight and bias in this layer. Similar to the approach used for the BNN for malware detection, the initial values of the Gaussian distribution parameters are set according to the values

of the weights and bias belonging to the last layer of a converged comparable Bi-LSTM neural network. Therefore, the initial values for weights mean, biases mean, weights standard deviation, and biases standard deviation are chosen as -0.1, -1.8, 0.4, and 0.8, respectively. The model is constructed through a series of experiments to optimize its architecture and hyperparameters, ensuring the most suitable results. In this process, we employ a categorical cross-entropy cost function, a common choice for text classification. We utilize the Adam optimizer with an initial learning rate of 0.001 for optimization. Furthermore, we implement a learning rate scheduler and early stopping technique to guarantee the highest accuracy. The batch size (a crucial part of this optimization process) is set to 128 to balance computational efficiency and model performance.

The dataset used in [57] is first split into training, evaluation, and testing and then converted to a new dataset by generating N-gram features built for a sequence of API calls by examining a series of consecutive API calls where the value of *N* varies from 2 to one less than the total length of the sequence. In this approach, considering each subsequence, the last API call within that subsequence is used as the label. For instance, when generating the initial data point, we take the first two API calls from the sequence and designate the third API call as the corresponding label. To understand N- gram feature extraction comprehensively, Table 3.1 provides an example of 7 API calls.

Table 3.1: N-grams for a subsequence of 7 numerical API calls

Data point	Corresponding label
$[50, 25]$	$[274]$
$[50, 25, 274]$	$[178]$
$[50, 25, 274, 178]$	$[117]$
$[50, 25, 274, 178, 117]$	$[16]$
$[50, 25, 274, 178, 117, 16]$	$[283]$

Generating N-gram features can help the Bayesian Bi-LSTM neural network effectively learn the dependencies and patterns between API calls. This technique aids in language modeling and text generation tasks for predicting the next token given a sequence of words [36]. Being able to predict the next API call to be made by the malware, given an observed sequence of API calls, allows us to predict the upcoming action of the malware. This is performed by adding the predicted API call to the input sequence and feeding it as new input to the model to predict the following API call further.

As a result, with an input sequence, we can predict the subsequent API calls one at a time until we obtain the necessary information regarding the future actions of the ongoing malware.

3.5 Experimental Results

In our research, the implemented Bayesian Bi-LSTM model can make stochastic predictions. Unlike conventional neural networks, we do not have a fixed set of parameters for different layers. Therefore, for a fixed input, the BNN samples the probability distributions of the parameters in the Bayesian layer for each prediction. This approach enables us to capture uncertainty in the output generated by the model when making predictions multiple times for the same input. Recognizing that there is no single set of parameters, our model provides a range of possible predictions for a given input, reflecting the inherent uncertainty in the model's learned parameters as well as in the processed data. Uncertainty is quantified as the standard deviation of the predicted probabilities for a given class, obtained through multiple evaluations of the model for the same input. This metric reflects the degree of variation or inconsistency in the model's predictions across different runs. A lower value of uncertainty (standard deviation) indicates that the model's predictions are more consistent and reliable for the given input, signifying higher confidence in the predicted class. Conversely, a higher uncertainty value suggests greater variability in the predictions, denoting lower confidence in the model's output. Accordingly, we run the models 20 times for a given input and average the predicted probabilities for each class. Considering the most certain class as the one with the highest average probability, we measure the uncertainty associated with this class represented by the standard deviation. Lower uncertainty values indicate higher confidence, implying that the model's predictions are consistently aligned across the runs. A high predicted probability and low uncertainty (standard deviation) for a particular class ensures high trust in the prediction. Table 3.2 showcases these results with specific examples of input API call sequences, the most certain class, and the actual label with the probability and uncertainty associated with the most certain class. As can be interpreted from this table, by observing the predicted probability and uncertainty values associated with the most certain class for the next API call, we can realize how much we can rely on the predictions made by the model.

Table 3.2: Next API call predictions along with the corresponding uncertainties and probabilities

Input Sequence	Most Probable Next API	True label	Probability	Uncertainty
[291, 291, 177, 291, 291]	294	294	1.0	0
[294, 289, 44, 289, 289]	44	44	0.99	0.01
[263, 34, 39, 39, 262]	199	34	0.13	0.03
[220, 220, 233, 233, 233]	237	220	0.38	0.11

When there is not a high probability value and small uncertainty associated with a made prediction, we need to doubt the outcome of the model. Therefore, we can look at the second and third most probable predictions made by the model and take those predictions into account as well. For this purpose, when the predicted probability value for the most probable prediction was less than 50%, we also considered the second most probable label. Additionally, when the predicted probability for the most probable label is less than 33%, we look at the second and third most probable labels. Cybersecurity specialists can rely on the less probable predictions based on their probability values and uncertainty measurements. Table 3.3 presents a few examples where the model is not confident about its predictions, and hence, less probable labels should also be provided.

Expanding on this, it is crucial to understand the value of these additional predictions in the context of cybersecurity, especially in the fight against malware. Malware attacks can be incredibly costly and complex, making it essential to consider every possible angle. In situations where the most probable prediction is not made with high confidence, second and third options can offer valuable insights. This methodology is crucial for a cybersecurity framework that aims to be both comprehensive and reliable. By including these lower probability predictions, the framework alerts human experts, who can then make confident decisions based on the ongoing context and their expertise. Such a system ensures a more contextual response to threats, particularly in environments where malware's behavior can be unpredictable. It enhances the reliability of the framework, ensuring that decisions are not solely based on automated predictions but are augmented by expert human judgment. This is a step towards creating a cybersecurity system that is robust, adaptable, and capable of responding to the evolving landscape of cyber threats.

As demonstrated in Table 3.3, when the primary prediction lacks an acceptable level of confidence, one can make more appropriate decisions by monitoring the corresponding probability and uncertainty. Cybersecurity experts occasionally choose less probable predictions out of experience.

Table 3.3: Comparison between first, second and third predictions sorted based on their probabilities and their associated uncertainties

Input Sequence	Prediction rank	Probability	Uncertainty	Prediction
[44, 252, 44, 107, 232]	Most certain	0.49	0.24	Wrong
[44, 252, 44, 107, 232]	Second most probable	0.44	0.26	Correct
[303, 316, 303, 316, 302]	Most certain	0.36	0.17	Wrong
[303, 316, 303, 316, 302]	Second most probable	0.23	0.14	Correct
[0, 0, 0, 0, 69]	Most certain	0.19	0.04	Wrong
[0, 0, 0, 0, 69]	Second most probable	0.11	0.05	Wrong
[0, 0, 0, 0, 69]	Third most probable	0.10	0.05	Correct

This method offers a valuable criterion for determining when professionals' expertise outweighs the analyzed pipeline's questionable reliability.

Figure 3.3 illustrates the performance of the Bayesian Bi-LSTM model for two randomly selected samples from the test set belonging to the second dataset [58] predicting the subsequent ten most probable API calls. The upper sequence is the most probable API call, and the sequence on the bottom is the ground truth. Table 3.4, on the other hand, presents the performance metrics of the Bayesian Bi-LSTM model, such as accuracy, precision, recall, and F1 score. This table compares the results with the deterministic Bi-LSTM neural network framework in [22] using the same dataset [22]. It also shows that the BNN can outperform its deterministic counterpart in terms of the above metrics. Note also that in this problem, the frequency of API calls among each label is different. It is also important to note that the problem involves API calls with various frequencies across different labels, as some API calls occur more frequently. Some API calls are significantly more common, while others are rare in malware and goodware samples. Consequently, the classification metrics for the Bayesian Bi-LSTM model are reported using weighted averages that consider the number of API call instances associated with each class, effectively treating them as support for each measurement.

By examining the classification report for each API call class, we can also estimate the reliability of the model predicting each API call. We gain valuable insights into the model's performance across various categories by analyzing precision, recall, and F1 scores for each class. We can decide if the model's prediction is trustworthy by combining the knowledge we gain with the predicted

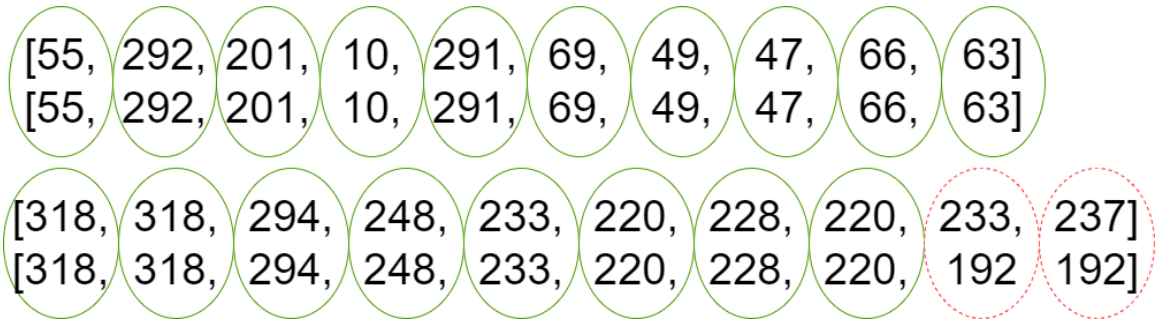

Figure 3.3: The next 10 API calls predicted by the proposed Bayesian Bi-LSTM model compared to the ground truth. The solid green and dashed red circles indicate the correct and wrong predictions, respectively.

Table 3.4: Performance Metrics of the Bayesian Bi-LSTM vs. Bi-LSTM Models for Predicting the Next API Call using the second dataset [58]

Metric	**Bayesian Bi-LSTM**	**Bi-LSTM**
Accuracy	89.53%	88.80%
Precision	89.25%	88.50%
Recall	89.52%	88.80%
F1 Score	89.22%	88.48%

probability and uncertainty when predicting the next API call. The model usually has a shortcoming when making predictions for API calls with low occurrence rates in the dataset. Table 3.5 presents some rarely occurred API calls with their associated metrics exhibiting the model's poor performance trying to predict them correctly.

Table 3.5: Rare API call performance metrics

API call	**Occurrences**	**Precision**	**Recall**	**F1 Score**
GetUserNameExW	19	0.36	0.42	0.39
CreateDirectoryExW	5	0.29	0.40	0.33
DrawTextExW	29	0.30	0.28	0.29
ReadCabinetState	9	0.20	0.11	0.14

The Bayesian Bi-LSTM model is used for malware detection to make probabilistic decisions. The first 50 API calls suffice for training the model and ensuring that the implemented model can detect the malware at its early stage. The performance metrics of the detection model are reported by making predictions ten times for each sample in the test set and averaging the probability values

and uncertainty measurements. Table 3.6 presents the performance metrics of the Bayesian Bi-LSTM model developed for early malware detection. The table reports a precision score of 97.26%, indicative of the false alarm rate of 2.74%, which shows the excellent performance of the proposed detector. The accuracy of our detection framework is compared to the existing malware detection results, including our previous work using XGBoost bagging in Table 3.7.

Table 3.6: Performance metrics of Bayesian Bi-LSTM neural network

Accuracy	96.44%
Precision	97.26%
Recall	99.12%
F1 score	98.18%

Reference	**Algorithm**	**Accuracy**
Sai et al. [37]	DT	89.89%
Xue et al. [38]	MLP	91.57%
Avci et al. [39]	Bi-LSTM	93.16%
Catak et al. [40]	Two layer LSTM	95.00%
Our previous work [22]	XG-Boost Bagging	95.85%
Present work	Bayesian Bi-LSTM	96.44%

Table 3.7: Comparison of Model Accuracies

Similar to the next-action prediction, by analyzing the probability values and uncertainty estimation associated with each prediction made by the model, one can evaluate the reliability of the corresponding output. This is where the advantage of employing a probabilistic (BNN) approach becomes evident. In contrast to a deterministic approach, which would require trusting every prediction made by the model, a probabilistic approach enables one to interpret the model's output. Consequently, we can identify and manage potentially erroneous predictions and avoid making assumptions of correctness for predictions with higher uncertainty. Table 3.8 showcases some wrong predictions made by the BNN and their associated probability values and uncertainties. As evident from the table, the wrong detection result is usually associated with a probability value less than the overall accuracy of the detection framework and high uncertainty.

Table 3.8: Predicted Labels with Probability and Uncertainty

Test data index	Predicted label	Probability	Uncertainty
39	malware	0.73	0.31
194	malware	0.89	0.13
386	Goodware	0.43	0.32

3.6 Conclusion and Future Work

This research presents an enhanced framework for malware detection and next-action prediction, combining NLP and BNN by incorporating probability and uncertainty into the model's outcomes. We treat sequences of API calls as natural language inputs, transform them into N-grams, and use a Bayesian Bi-LSTM network for probabilistic predictions and uncertainty estimations. We use BNN to overcome the limitations associated with deterministic neural networks, including lack of uncertainty quantification, being prone to overfitting, and suboptimal performance when dealing with limited datasets. A Bayesian layer is implemented as the output layer of a Bi-LSTM neural network, and variational inference is used to estimate the posterior probability distribution of the parameters of this layer. We show that using the Bayesian layer as a part of the neural network provides an enhanced probabilistic approach to predictions and presents valuable insights for each prediction. Measuring uncertainty and probability values associated with each outcome in detection and next-action prediction, on the other hand, equips cybersecurity experts with crucial information concerning the reliability of the outcomes. The proposed framework not only offers extensive insights into the reliability and trustworthiness of predictions but also outperforms the results of [22] in both detection and next-action prediction. For future work, one can improve the performance of malware analysis by using different architectures for the BNN and more effective algorithms for variational inference. In addition, one can investigate the application of the proposed framework for predicting specific threat scenarios or attack patterns to enhance its practical utility in cybersecurity. Finally, the real-time implementation of BNNs for immediate attack detection and early mitigation is another important direction for future research.

Chapter 4

Conclusion and Future Work

4.1 Conclusion

This thesis presents a novel framework that can be exploited for malware detection and mitigation strategies, focusing on the early stages of threat identification. The core methodology revolves around treating API call sequences similar to natural language inputs, leveraging the capabilities of neural network models, particularly Bi-LSTM and Bayesian Bi-LSTM. The first part of this research illustrates the effectiveness of a Bi-LSTM neural network in predicting API calls. Using the similarities between the sequence of API calls and the structure of natural language, we utilize text classification methods to anticipate malware activities.

The subsequent study builds upon this foundation by integrating a Bayesian approach into the Bi-LSTM framework. This enhanced method introduces probability and uncertainty assessments into the model's predictions, addressing key limitations of deterministic neural networks. By transforming API call sequences into N-grams and applying a Bayesian Bi-LSTM network, the research provides more meaningful probabilistic predictions along with uncertainty estimations. This advancement is particularly crucial in cybersecurity, where understanding the reliability and precision of predictions can significantly impact mitigation strategies.

Together, these studies offer a comprehensive framework for early-stage malware detection, combining the strengths of NLP methodologies and advanced neural network architectures. The incorporation of Bayesian principles into this framework not only enriches the predictive accuracy but

also equips cybersecurity experts with critical insights regarding the reliability of these predictions. The outcomes of this thesis open new avenues for future research in the domain of cybersecurity, emphasizing the importance of probabilistic approaches in the ever-evolving landscape of cyber threats.

4.2 Future Work

As a possible direction for future research, one can incorporate advanced natural language processing techniques, specifically transformer-based models like BERT or GPT, into the proposed framework. These models perform well in understanding complex contexts, which could enhance the way API call sequences are interpreted, leading to more precise malware predictions. This enhancement can also enable the framework to adapt to the evolving nature of malware threats more effectively.

As another related line of research in this field, broadening the scope and diversity of the utilized datasets is essential. Including a more comprehensive range of data, e.g., a more diverse family of malware samples, will enhance the model's capacity to detect a wider array of cyber threats. Collaborating with cybersecurity experts for access to real-world datasets associated with a large variety of malware would be beneficial in this effort.

Finally, the real-time implementation of these models is essential. In scenarios where API call sequences need to be monitored and processed with great speed, the ability to apply these advanced detection methods in real time becomes critical. This aspect of research would test the feasibility of the proposed models in environments where quick response times are essential for effective malware mitigation. Implementing these models in real-time scenarios will not only validate their efficacy under dynamic conditions but also provide insights into potential areas for further improvement. This would mark a significant step toward creating more responsive and robust cybersecurity systems.

Bibliography

[1] Frank Cremer, Pieter J. M. Jong, René M. Stulz, Jayanth R. Varma, and Gülnur Ülkü, "Cyber risk and cybersecurity: A systematic review of data availability," *The Geneva Papers on Risk and Insurance - Issues and Practice*, vol. 47, no. 3, pp. 698-736, July 2022.

[2] Roshan Ali, Aqeel Ali, Farrukh Iqbal, Malik Hussain, and Fazal Ullah, "Deep learning methods for malware and intrusion detection: A systematic literature review," *Security and Communication Networks*, pp. 2959222, 2022.

[3] Yuchong Li and Qinghui Liu, "A comprehensive review study of cyber-attacks and cyber security; Emerging trends and recent developments," *Energy Reports*, vol. 7, 2021.

[4] Amit Kumar Chakravarty, Aadesh Raj, Sagnik Paul, and Anirban Sarkar, "A study of signature-based and behaviour-based malware detection approaches," *International Journal of Advanced Research Ideas and Innovations in Technology*, vol. 5, no. 3, pp. 1509-1511, 2019.

[5] Mohammad Amin Rahimian and Amir G. Aghdam, "Structural controllability of multi-agent networks: Robustness against simultaneous failures," *Automatica*, vol. 49, no. 11, pp. 3149-3157, 2013.

[6] Saeid Jafari, Amir Ajorlou, and Amir G. Aghdam, "Leader localization in multi-agent systems subject to failure: A graph-theoretic approach," *Automatica*, vol. 47, no. 8, pp. 1744-1750, 2011.

[7] Saeid Jafari, Amir Ajorlou, Amir G. Aghdam, and S. Tafazoli, "On the structural controllability of multi-agent systems subject to failure: A graph-theoretic approach," in *Conference on Decision and Control (CDC)*, Atlanta, GA, USA, pp. 4565-4570, 2010.

[8] Nordic APIs, "Should You Design Natural Language First APIs?," *Nordic APIs*, October 18, 2018. [Online]. Available: https://nordicapis.com/should-you-design-natural-language-first-apis/. [Accessed: April 5, 2023].

[9] Kedir Zeberga, Muhammad Attique, Bushra Shah, Fazal Ali, Yacob Z. Jembre, and Tai-hoon Kim, "A Novel Text Mining Approach for Mental Health Prediction Using Bi-LSTM and Bert Model," *Computational Intelligence and Neuroscience*, pp. 1-18, 2022.

[10] Gang Liu and Jiabao Guo, "Bidirectional LSTM with attention mechanism and convolutional layer for text classification," *Neurocomputing*, vol. 337, pp. 325-338, 2019.

[11] Chengwei Li, Guan Zhan, and Zhen Li, "News Text Classification Based on Improved Bi-LSTM-CNN," in *International Conference on Information Technology in Medicine and Education*, Hangzhou, China, 2018.

[12] Kaiyao Ke, "A Comparative Study of CNN and Bi-LSTM in Text-Based Sentiment Analysis," in *International Conference on Networks, Communication and Computing*, Association for Computing Machinery, New York, NY, USA, 2022.

[13] Shamal Kashid, Divyansh Dixit, Nidhi Mishra, and Rohit Saluja, "Bi-RNN and Bi-LSTM Based Text Classification for Amazon Reviews," in *International Conference on Deep Learning, Artificial Intelligence and Robotics*, Cham: Springer International Publishing, 2022.

[14] Clément Bentéjac, Anna Csörgő, and Gonzalo Martínez-Muñoz, "A comparative analysis of gradient boosting algorithms," *Artificial Intelligence Review*, vol. 54, pp. 1937-1967, 2021.

[15] Yarin Gal and Zoubin Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *International Conference on Machine Learning*, PMLR, 2016.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*, MIT press, 2016.

[17] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra, "Weight Uncertainty in Neural Networks," in *International Conference on Machine Learning*, 2015.

[18] Boyang Xue, Shoukang Hu, Junhao Xu, Mengzhe Geng, Xunying Liu, and Helen Meng, "Bayesian Neural Network Language Modeling for Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 1-16, 2022.

[19] ISACA, "An Executive View of Key Cybersecurity Trends and Challenges in 2023," August 22, 2023. [Online]. Available: https://www.isaca.org/resources/news-and-trends/industry-news/2023/an-executive-view-of-key-cybersecurity-trends-and-challenges-in-2023. [Accessed: October 31, 2023].

[20] Meire Fortunato, Charles Blundell, and Oriol Vinyals, "Bayesian recurrent neural networks," *arXiv preprint arXiv:1704.02798*, 2017.

[21] Jianpeng Zhang, Yijing Zhou, Chen Wang, Jialiang Lu, and Xudong Zhu, "Advances in Bayesian deep learning: theory and applications," *IEEE Access*, vol. 7, pp. 85730-85748, 2019.

[22] Zahra Jamadi and Amir G. Aghdam, "Early Malware Detection and Next-Action Prediction," in *International Conference on RFID Technology and Applications (RFID-TA)*, Aveiro, Portugal, 2023.

[23] Tea Gržinić and Esteban Borges González, "Methods for automatic malware analysis and classification: a survey," *International Journal of Information and Computer Security*, vol. 17, no. 1-2, pp. 179-203, 2022.

[24] SonicWall, "SonicWall 2021 Cyber Threat Report," 2021. [Online]. Available: https://www.sonicwall.com/resources/2021-sonicwall-cyber-threat-report/. [Accessed: April 5, 2023].

[25] AV-TEST GmbH, "AV-TEST Statistics," 2023. [Online]. Available: https://www.av-test.org/en/statistics/malware/. [Accessed: April 5, 2023].

[26] Patrice Maniriho, Amin Mahmood, and M. Jashim Chowdhury, "Evaluation and survey of state of the art malware detection and classification techniques: Analysis and recommendation," *SSRN Electronic Journal*, 2022.

[27] Mathew Rhode, Pete Burnap, and Kevin Jones, "Early-stage malware prediction using recurrent neural networks," *Computers and Security*, vol. 77, pp. 578-594, 2018.

[28] Kai Chang, Nianyin Zhao, and Lin Kou, "A Survey on Malware Detection based on API Calls," in *International Conference on Dependable Systems and Their Applications (DSA)*, August 2022.

[29] Gopinath Ganapathy Sundarkumar, Hariharan Sundararajan, Srivatsan Ramanujam, Suresh Jagannathan, and Ananth Grama, "Malware detection via API calls, topic models and machine learning," in *2015 IEEE International Conference on Automation Science and Engineering (CASE)*, August 2015.

[30] Chengwei Li, Guan Zhan, Zhen Li, Shenghong Li, and Jianlong Tan, "A novel deep framework for dynamic malware detection based on API sequence intrinsic features," *Computers Security*, vol. 116, 2021.

[31] Xiaolin Deng, Weiwei Lin, Sheng Chen, and Junjie Tong, "Bagging-XGBoost algorithm based extreme weather identification and short-term load forecasting model," *Energy Reports*, vol. 8, pp. 8661-8674, 2022.

[32] Angelo Oliveira and Rodrigo Sassi, "Behavioral Malware Detection Using Deep Graph Convolutional Neural Networks," *TechRxiv, preprint*, 2019. [Online]. Available: https://doi.org/10.36227/techrxiv.12020416.v1. [Accessed: April 5, 2023].

[33] Furkan Oğuzhan Çatak and Adnan Fatih Yazı, "A Benchmark API Call Dataset for Windows PE Malware Classification," *arXiv preprint arXiv:1905.01999*, 2019.

[34] Tianqi Chen and Carlos Guestrin, "Xgboost: A scalable tree boosting system," in *International Conference on Knowledge Discovery and Data Mining*, 2016.

[35] Charles Elkan, “The Foundations of Cost-Sensitive Learning,” in *International Joint Conference on Artificial Intelligence*, 2001.

[36] Yinzhi Zhang and Zhaohui Rao, “n-BiLSTM: BiLSTM with n-gram Features for Text Classification,” in *Information Technology and Mechatronics Engineering Conference (ITOEC)*, Chongqing, China, 2020.

[37] Krishna Nand Sai, Biju Thanudas, Sreejith Sreelal, Anirban Chakraborty, and Biju Somanath Manoj, ”MACA-I: A Malware Detection Technique Using Memory Management API Call Mining,” in *2019 IEEE Region 10 Conference (TENCON)*, 2019, pp. 527-532.

[38] Jing Xue, Zhiyong Wang, and Rong Feng, ”Malicious Network Software Detection Based on API Call,” in *International Conference on Network and Information Systems for Computers (ICNISC)*, Hangzhou, China, 2022, pp. 105-110, doi: 10.1109/ICNISC57059.2022.00032.

[39] Cihan Avci, Bedir Tekinerdogan, and Cagatay Catal, ”Analyzing the Performance of Long Short-Term Memory Architectures for Malware Detection Models,” in *Concurrency and Computation: Practice and Experience*, vol. 35, no. 6, 2023, pp. 1-1.

[40] Ferhat Ozgur Catak, Ali Fahri Yazı, Ornela Elezaj, and Javed Ahmed, ”Deep Learning Based Sequential Model for Malware Analysis Using Windows EXE API Calls,” in *PeerJ Computer Science*, vol. 6, 2020, e285.

[41] Syed Md. Nazmus Hasnaeen and Alexander Chrysler, ”Detection of Malware in UHF RFID User Memory Bank using Random Forest Classifier on Signal Strength Data in the Frequency Domain,” in *2022 IEEE International Conference on RFID (RFID)*, Las Vegas, NV, USA, 2022, pp. 47-52.

[42] Melanie R. Rieback, Bruno Crispo, and Andrew S. Tanenbaum, ”RFID Malware: Design Principles and Examples,” in *Pervasive and Mobile Computing*, vol. 2, no. 4, pp. 405-426, 2006.

[43] Syed Saeed, Syed Abdulrahman Altamimi, Nourah Alkayyal, Ebtesam Alshehri, and Doaa Alabbad, "Digital transformation and cybersecurity challenges for businesses resilience: Issues and recommendations," *Sensors*, vol. 23, no. 15, pp. 6666, 2023.

[44] Boyang Xue, Shoukang Hu, Junhao Xu, Mengzhe Geng, Xunying Liu, and Helen Meng, "Bayesian Neural Network Language Modeling for Speech Recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 1-16, 2022.

[45] Nazri Zainal Gorment, Ali Selamat, and Oldřich Krejcar, "A Recent Research on Malware Detection Using Machine Learning Algorithm: Current Challenges and Future Works," in *Advances in Visual Informatics*, Springer International Publishing, 2021.

[46] Christian Leibig, Veronika Allken, Mehmet Sükrü Ayhan, Philipp Berens, and Siegfried Wahl, "Leveraging uncertainty information from deep neural networks for disease detection," *Scientific Reports*, vol. 7, no. 1, pp. 1-13, 2017.

[47] Mattia Magris and Alexandros Iosifidis, "Bayesian learning for neural networks: an algorithmic survey," *Artificial Intelligence Review*, vol. 56, pp. 11773-11823, 2023.

[48] Syahril Rizal Tuan Mat, Muhammad Faiz Abdul Razak, Mohd Nazri Mohd Kahar, Jazmi Mohd Arif, and Ahmad Firdaus, "A Bayesian probability model for Android malware detection," *ICT Express*, vol. 8, pp. 424-431, August 2022.

[49] José Antonio Perusquía Cortes, "Bayesian Nonparametric Methods for Cyber Security with Applications to Malware Detection and Classification," Doctor of Philosophy (PhD) thesis, University of Kent, 2022.

[50] Abdullah Mohammed AlZain, "A Bayesian Approach to Malware Classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 1-10, March 2018.

[51] Mohammed Abdullah AlZain, "A Bayesian Approach to Malware Classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 1-10, March 2018.

[52] Mattia Magris and Alexandros Iosifidis, "Bayesian learning for neural networks: an algorithmic survey," *Artificial Intelligence Review*, vol. 56, pp. 11773-11823, 2023.

[53] Peisong Zhao, Malay Ghosh, J. N. K. Rao, and Changbao Wu, "Bayesian Empirical Likelihood Inference with Complex Survey Data," in *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 82, no. 1, pp. 155-174, February 2020.

[54] Laurent Valentin Jospin, Hamid Laga, Ferdaous Boussaid, Wray Buntine, and Mohammed Bennamoun, "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users," in *IEEE Computational Intelligence Magazine*, vol. 17, no. 2, pp. 29-48, May 2022.

[55] Dustin Tran, Michael W. Dusenberry, Mark van der Wilk, and Dustin Hafner, "Bayesian Layers: A Module for Neural Network Uncertainty," in *Advances in Neural Information Processing Systems 32*, 2019.

[56] Gautam Karat, "API Call Sequences," *Kaggle*, October 2021. [Online]. Available: https://www.kaggle.com/datasets/gautamkarat/api-call-sequences

[57] Angelo Oliveira, "Malware Analysis Datasets: API Call Sequences," IEEE Dataport, October 23, 2019.

[58] Furkan Oğuzhan Çatak and Adnan Fatih Yazı, "A Benchmark API Call Dataset for Windows PE Malware Classification," *arXiv preprint arXiv:1905.01999*, 2019.

[59] Mostafa Nasraldeen, Ahmed Khleel, and Károly Nehéz, "Software defect prediction using a bidirectional LSTM network combined with oversampling techniques," *Cluster Computing*, vol. 26, no. 6, pp. 1-14, October 2023.