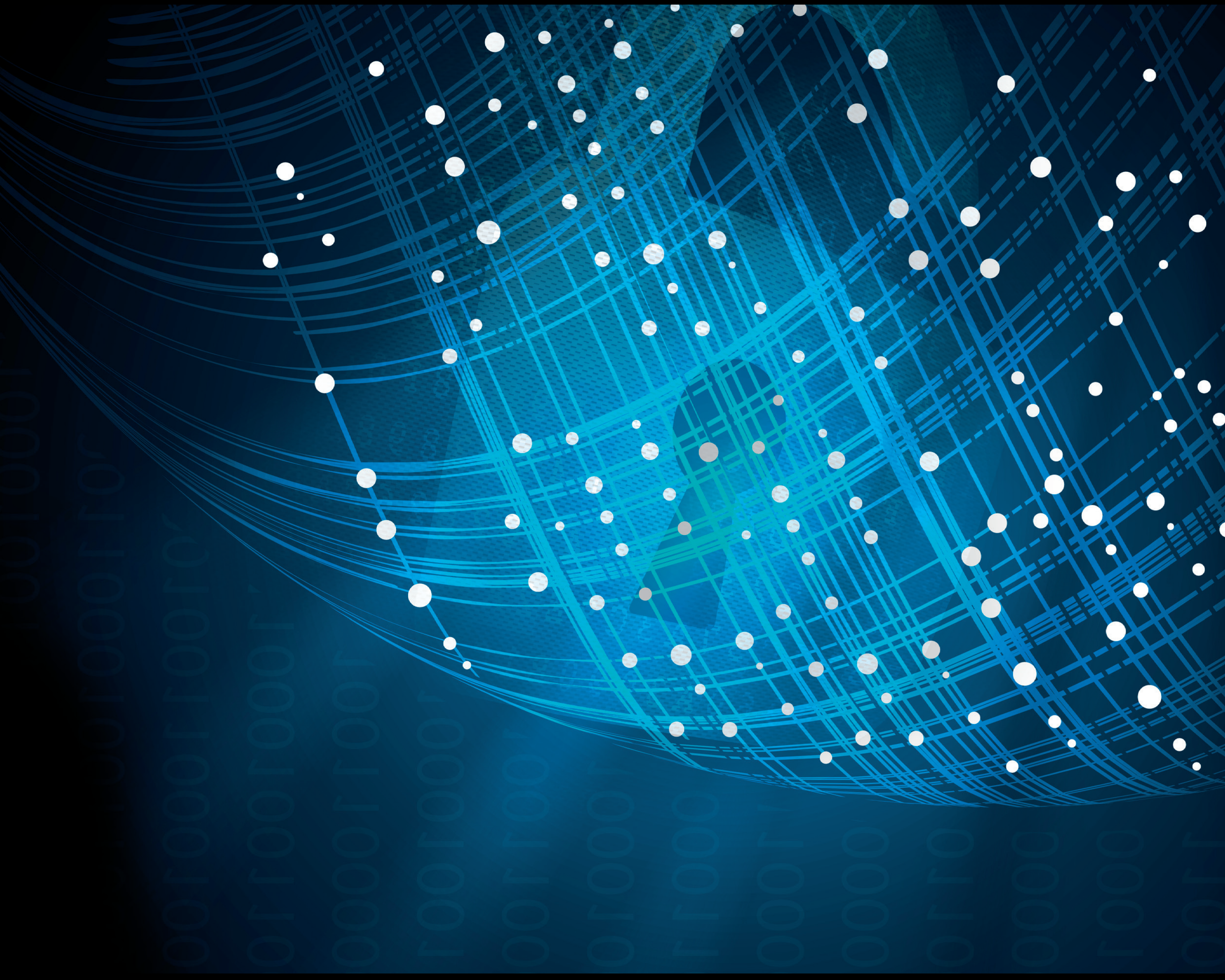
Security and Communication Networks

Malware Analysis and Vulnerability Detection Using Machine Learning 2022

Lead Guest Editor: Farrukh A. Khan
Guest Editors: Muhammad Faisal Amjad and Hammad Afzal



Malware Analysis and Vulnerability Detection Using Machine Learning 2022

Security and Communication Networks

Malware Analysis and Vulnerability Detection Using Machine Learning 2022

Lead Guest Editor: Farrukh A. Khan
Guest Editors: Muhammad Faisal Amjad and Hammad Afzal

Chief Editor

Associate Editors

Academic Editors

Contents

Hindawi
Security and Communication Networks
Volume 2023, Article ID 1216192, 7 pages
https://doi.org/10.1155/2023/1216192

Research Article

An Improved Big Data Analytics Architecture for Intruder Classification Using Machine Learning

Muhammad Babar,[1] **Sarah Kaleem**,[2,3] **Adnan Sohail**,[2] **Muhammad Asim**,[3,4] **and Muhammad Usman Tariq**[5]

[1]*Robotics and Internet of Things Lab, Prince Sultan University, Riyadh 11586, Saudi Arabia*
[2]*Computing and Technology Department, Iqra University, Islamabad 44000, Pakistan*
[3]*EIAS Data Science Lab, Prince Sultan University, Riyadh 11586, Saudi Arabia*
[4]*School of Computer Science and Technology, Guangdong University of Technology, Guangzhou 510006, China*
[5]*Abu Dhabi University, Abu Dhabi 59911, UAE*

Correspondence should be addressed to Sarah Kaleem; sarahkaleem33887@iqraisb.edu.pk

Received 18 October 2022; Revised 21 January 2023; Accepted 11 April 2023; Published 4 December 2023

Academic Editor: Chien-Ming Chen

The approval of retrieving information on the Internet originates several network securities matters. Intrusion recognition is a critical study in network security to spot unauthorized admission or occurrences on protected networks. Intrusion detection has a fully-fledged reputation in the current era. Research emphasizes several datasets to upsurge system precision and lessen the false-positive proportion. This article proposes a new intrusion detection system using big data analytics and deep learning to address some of the misuse and irregularity detection limitations. The proposed method could identify any odd activities in a network to recognize malicious or unauthorized action and permit a response during a confidentiality break. The proposed system utilizes the big data analytics platform based on parallel and distributed mechanisms. The parallel and distributed platforms improve the training time along with the accuracy. The experimentation appropriately classifies the information as either normal or abnormal. The proposed system has a recognition proportion of 96.11% that pointedly expands overall recognition accuracy related to existing strategies.

1. Introduction

The eminence and number of cyberattacks have amplified pointedly as the services upsurge due to the Internet [1, 2]. The demand and the execution of safety actions are needed to avoid and comprehend the cyberattacks, i.e., many dangerous software packages such as trojans, malware, viruses, and other unknown hacking tactics [3, 4]. Hence, there is a need for a method that must drastically improve the security of the system's networks to avoid cyberattacks. Data integrity, confidentiality, and availability will not be limited as an upshot [5]. Every network may be considered a target, and all the systems are vulnerable. As a result, it is susceptible to illegal access and disclosing private and erogenous data [6, 7]. A firewall is a versatile and essential component of any security system. It configures the security policy but cannot protect us from malicious activities. Only the text of a packet's caption is examined in this firewall situation, whereas both materials and captions of a package are read in an intrusion detection system (IDS) [8]. The IDS is a more dynamic method for defending sensitive and private data.

According to the definition, an intrusion is any act that compromises data integrity, confidentiality, or availability [9]. IDS still needs to be fully formed and considered a comprehensive safeguard despite playing a vibrant role in scheming and safeguarding a secure structure. An IDS distinguishes intrusions and issues a warning in the form of an alert to guarantee that resources are not compromised [10]. An IDS acts as a deterrent to illegal access to information. It offers a user-friendly interface for nonexpert workers to activate the systems professionally. In current years, online fraud has become more predominant in China. However, a community clamor exploded the previous year after a phone scam targeted a college student. An IDS is a network security system formerly intended to distinguish vulnerability against a sole application. The IDS is out-of-band on the network structure and is not in the correct communication path between the transmitter and receiver of data [11]. Instead, IDS on edge would engage a TAP or SPAN port to check a copy of the inline traffic flow stream [12]. The IDS was built in this manner as the complexity of the analysis essential for intrusion detection could not be steered at a rate that could persevere the equipment on the network [13]. As previously noted, the IDS is also a listen-only device, as the attackers can exploit vulnerabilities quickly after gaining network access.

Deep learning (DL) is a class of ML methods considered by some computation layers that permit an algorithm to learn suitable predictive features [14, 15]. The advent of deep learning has squeezed many machine learning-based applications. The victory is based on two foremost benefits: (1) it affords the capability to learn nonlinear relationships between parameters and (2) it permits one to leverage information from unlabelled data that does not belong to the problem. Furthermore, DL practices could be used in cases, where massive or complex data processing challenges ML or conventional data analysis methods [16]. Currently, the researchers are relying upon the DL technique. Deep learning may be employed along with other automation techniques, e.g., rule- and heuristics-based and machine learning techniques [17].

Unauthorized movements on a computer network are referred to as intrusion. It was either passively or aggressively successful. Data collection and eavesdropping are used in a passive intrusion. In the event of full of life, however, intrusion occurs and is accomplished through destructive packet forwarding, packet dropping, and whole attacks [18]. An IDS aims to classify an intruder before it imposes actual harm to the system. Currently, the current security methods that are unrealistic are utilized to avoid security faults. Hence, the misuse recognition method cannot classify unidentified attacks. Anomaly-based detection is realized by increasing the accuracy rate of intrusion detection using deep learning methods. In this research, an improved system is proposed for addressing the precise recognition of the intrusion. The classical LSTM is modified to classify, investigate, and produce estimates of the intruders based on time series data.

The reaming of the paper is organized as follows. Section 2 elaborates the detailed literature review along with related work. Section 3 represents the proposed methodology. Afterward, Section 4 discusses the results and discusses on results. Finally, Section 5 concludes the manuscript.

2. Related Work

An IDS is observed as a hazardous element in shielding systems that store critical information, intellectual property, and other digital resources. The complete system could rapidly crumble and cast doubt on the long-term viability if a private party has access to the information [19]. An IDS has conventionally maintained administrators in detecting intrusions and managing risks [20]. On the other hand, the act of IDS is steadily endangered. The technology hackers use to hijack a network and the counter-technology administrators are deployed to combat these attacks. This has outstripped the opportunity and measure of IDS [21]. Soft computing is one of the policies that aid in reducing detection costs [22]. Because of their learning and flexibility, capabilities are used to construct utilities in the intrusion detection industry. The ability to detect threats and attacks is critical to their prevention, and accurate threat detection is vibrant. Numerous intrusion detection models have been proposed in the context of security. In the current research, the neural network method has become one of the most often utilized soft computing strategies [23].

IDSs have been presented based on anomaly detection using an unsupervised ANN [24]. A hybrid data mining method combined k-means clustering with the SMO classification algorithm for classifying network intrusions [25]. The NSL-KDD dataset was utilized with k-means clustering to reduce the dimensionality of the training dataset. SMO has completed the classification process to classify the intruders. The suggested approach (k-mean + SMO) achieved a 94.48 percent positive detection rate while lowering the false alarm rate to 0. A novel edge-up methodology called cluster center and nearest neighbor was created and implemented that computed two distances [26]. The KDD Cup 99 dataset was utilized in the tests. It trains the dataset before it is divided into multiple subsets. Normal and abnormal CANN profiles are created using SVM that do not outperform K-NN in classifying R2L and U2L attacks [27]. A new hybrid approach, DT-SVM, was presented, and a foundation classifier that uses two classifiers, including SVM and a decision tree was used. This mixed technique aimed to increase detection accuracy while minimizing computational complexity.

A hierarchical hybrid detection method was proposed that integrates misuse and anomaly detection algorithms [28]. This hybrid method outperforms the traditional models. Multicategory classifiers were employed to upsurge intrusion detection accuracy [29]. The study's main goal was to use all the classifiers' excellent features. The results of the tests show that the accuracy of detecting denial of service attacks has increased. A hybrid method with several metrics is proposed and applied for better accuracy [30]. Finally, the hybrid model's results were compared to other primary algorithms that better detect intrusion. The SVM algorithm accuracy is 94%, and the result also shows a significant percentage of (FP), (TP), (FN), and (TN) alarms when using a hybrid model. A novel IDS based on a data gain criterion and unique SVM was proposed for extracting noteworthy features from the network traffic archives domain to categorize and detect future intrusions [31]. Machine learning techniques have acknowledged considerable attention in the middle of the intrusion detection scholars to report facts-based weaknesses in detecting methods [32]. Unsupervised techniques like k-means, SOM, and one-class SVM outperformed the others, although their ability to correctly identify all attack types needed to be more consistent.

Deep learning has glimmered much attention in academia and industry as a newfangled hotspot in neural networks. Deep knowledge has formed excellent results in the field of intrusion detection. It is claimed and proved that an LSTM recurrent neural network could detect intrusions realistically. The suggested classifier successfully distinguished between DOS attacks and network probes, each with a time series of events. With a 93.82 percent accuracy rate, LSTM surpassed the winning entries in the KDD Cup 99 competition, according to experiments. Machine learning employs a compound architecture or a series of nonlinear operations to obtain high-level abstractions in data. LSTM is applied to an RNN in this article. The NSL-KDD dataset is used to train the model and measure its performance.

3. Proposed Methodology

The proposed system performs processing in a parallel fashion. It utilizes the big datasets using the big data analytics platform. The proposed approach utilizes the LSTM model of learning. It preprocesses information with an unsupervised filter and classifies it with the LSTM algorithm. The proposed system is fabricated from many processes that process data to classify irregularities in network traffic. The multiple functions are processed in a parallel manner. The proposed system adds various layers to address the limitation of intrusion detection accuracy. The block model of the proposed scheme is depicted in Figure 1, where the detailed structure is provided in Figure 2.

The proposed hybrid approach combines two leading deep learning convolution-based models. The proposed technique adds three convolutional layers to design the hybrid method. The endangered gradient delinquent, where the neural network's efficiency diminishes due to inadequate training, is one limitation of widespread RNNs. Usual RNNs with a gradient-based learning method reduce as their significance and intricacy rise. Amending the sceneries professionally at the initial stages is time consuming and computationally exhaustive. RNNs might practice the obligation to discover to remember the critical data and then loop back to the network if the data are not recognized. Although network traffic collection comprises several impractical, imprecise unrelated data, and noisy data that affect the result across normal and abnormal network traffic categorization. As a result, preprocessing is essential to expand the recognition capacities of classifiers.

Preprocessing permits standardizing and cleaning of the data; the preprocessing is carried out so that the inappropriate information does not encumber classifier accuracy, and redundant data are removed. The big datasets are preprocessed before being translated into the format required for the proposed improved modified LSTM model. The proposed modified LSTM model will be a multilayered sequential model, including two LSTM layers followed by a thick layer that predicts the detection rate. The sequential class came from the Keras Models library, while the Embedding, Masking, LSTM, Dropout, and Dense types came from the Keras Layers library. Initially, an instance of the Sequential class is created to utilize as a proposed modified model. In addition, the Embedding, Masking, LSTM, Dropout, and Dense layers are added to it.

The LSTM layer is added to the sequential model to start the modeling building. The embedding layer for data form follows it, and the masking layer for pretrained embeddings. The first parameter is the number of neurons or nodes required for the LSTM layer. A dropout layer is added to the proposed layer using the second parameter. Finally, a thick layer is added at the end to make the model more resilient. The convolutional layers accomplish the operation over the input K. The output of the coating can be calculated as equation (1). To compute the more diverse and rich representation of the input, multiple filters have been used.

$$\mathrm{CN}\left(Y_{i,j}\right) = \sum_{x=-a/2}^{a/2} \sum_{l=-n/2}^{n/2} F_z \ (l,m) Z_{i-l,j-m}. \tag{1}$$

Later the rectified linear unit is processed that is applied after convolution operation. ReLU can be calculated as follows:

$$\mathrm{RECT}\,(Z) = \max\,(O, Z). \tag{2}$$

ReLU provides faster convergence during training and performs better than other activation functions like Sigmoid because it overcomes the vanishing gradient problem since the gradient is linear function. Finally, the polling layer is processed. There are different types of polling such as average, maximum, and minimum polling. Maximum polling is the most popular pooling technique, which takes the largest value over the input x. Let p be the size of the polling filter, and the output of the polling operation is computed as follows:

$$M\left(Z_i\right) = \max\left\{Z_{i+n,i+m}\right\}. \tag{3}$$

4. Experimental Results and Discussion

The experimental results and discussion are discussed in this section. Big data processing is based on the parallel and distributed mechanism, where the big data are divided into various chunks (blocks), and each piece is loaded and processed in parallel. It is imperative to decide the number of parallelisms that how many blocks (nodes) are required to be loaded at a time for computations and processing. This concept is known as the level of equality. We utilized the Apache Spark big data open-source platform.

RNN can be used to solve sequence problems. LSTM is being exploited to manage sequence problems. The recommended techniques based on LSTM with selecting 41 features in an NSL-KDD dataset are adopted to increase the accuracy. The dataset is divided into 70-30 training and testing rats. The text labels in the target data are first encoded into an integer. The same preprocessing technique turns all text inputs in the training and test data into integers. Instead of memorizing the data displayed during training, the

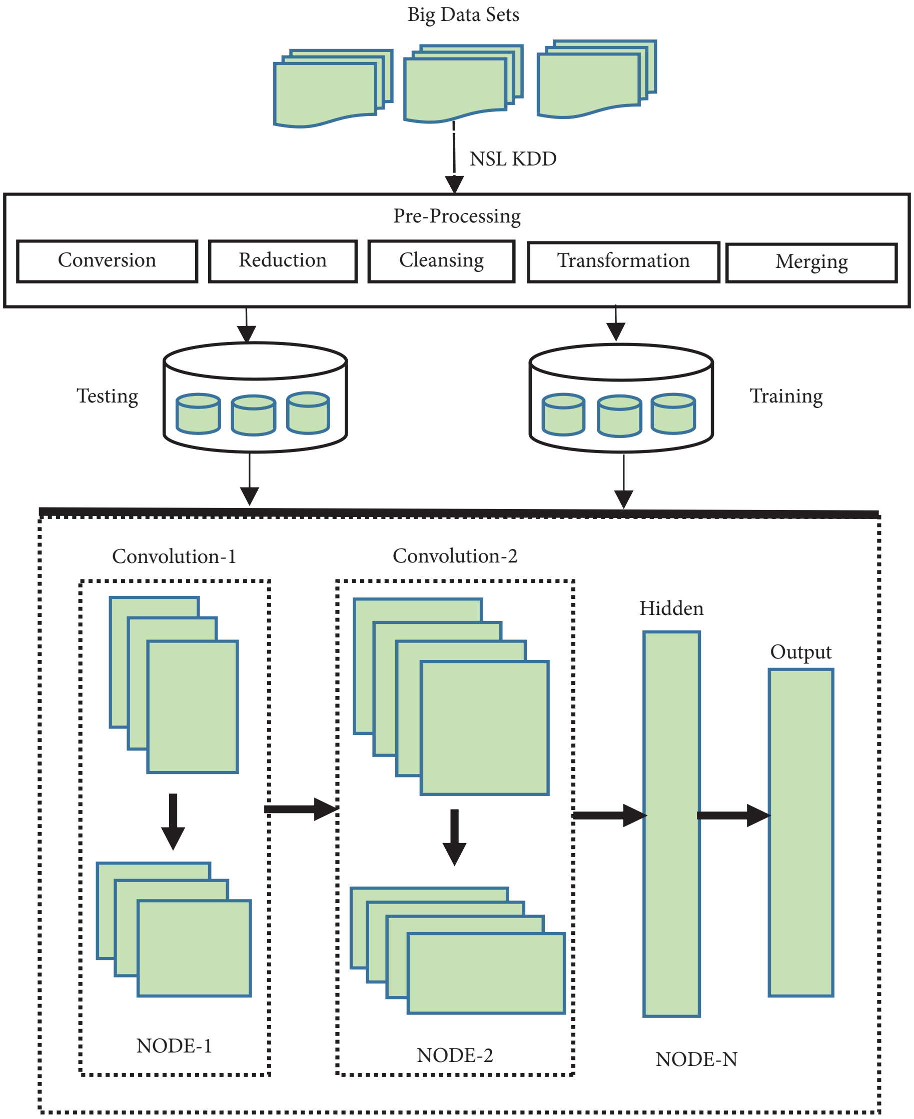


Figure 1: Proposed block model.

machine learning model aims to discover patterns that generalize well to new data. The proposed model performs well on unseen instances that were not used to train the model. Figure 3 shows the model prediction on the evaluation dataset (held-out data). Figure 4 depicts the loss, which is reasonably encouraging when compared to earlier traditional procedures.

It is evident from the graph that the proposed classifier accuracy is improved. The classifier accurately recognises incursions with 20 epochs.

The model loss is also reported for 20 epochs with a batch size of 64. Compared to previous strategies, the model converges after 20 cycles which is a level-headedly short time. Figure 5 demonstrates the model building time over several epochs. The accuracy analysis of different models is also shown in Figure 6.

The confusion matrix is used to generate most performance metrics. When making predictions, the classification model becomes perplexed. To obtain a logic of precise and error in prediction, the normalization process is carried out.

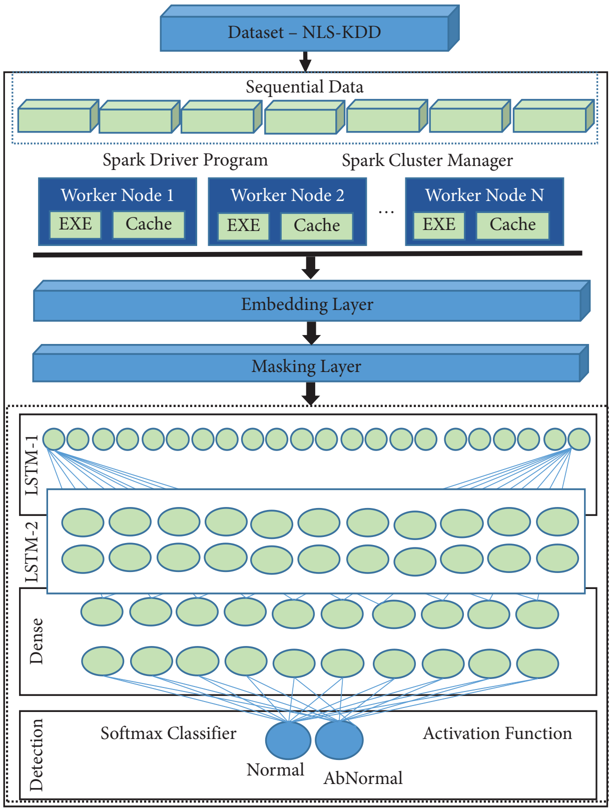


Figure 2: Proposed model structure.

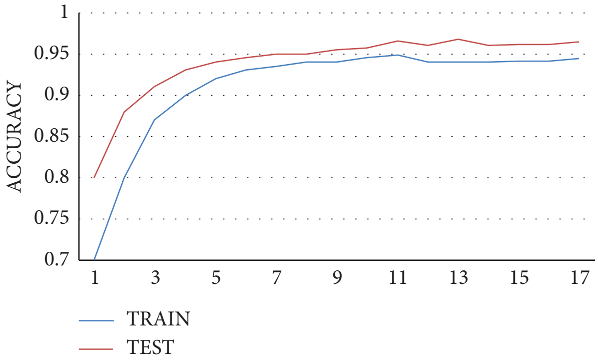


Figure 3: Proposed modified LSTM model accuracy rate.

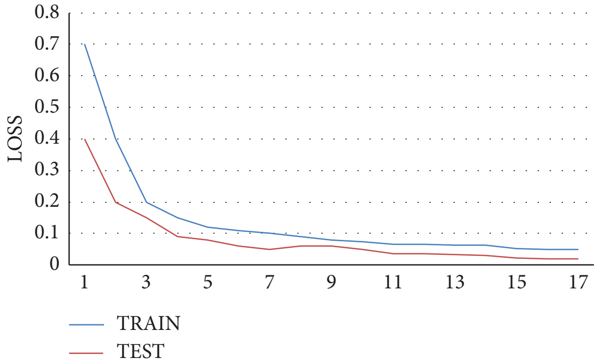


Figure 4: Proposed modified LSTM model loss rate.

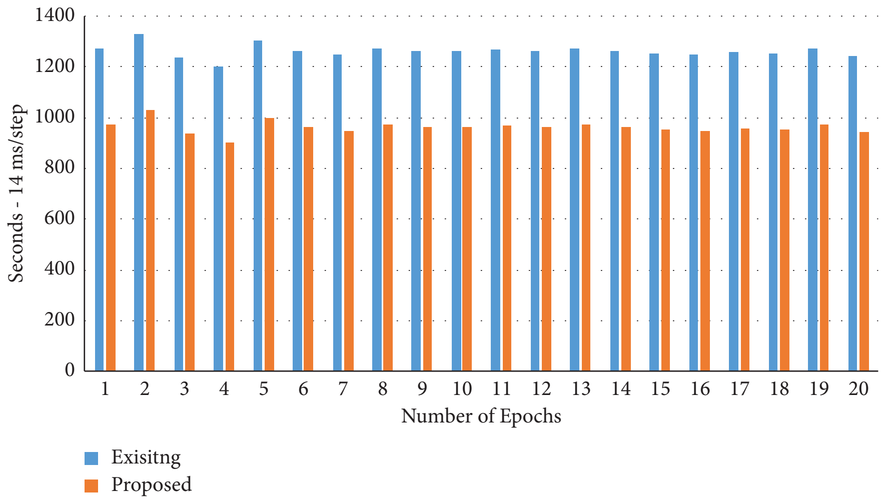


Figure 5: Model building time.

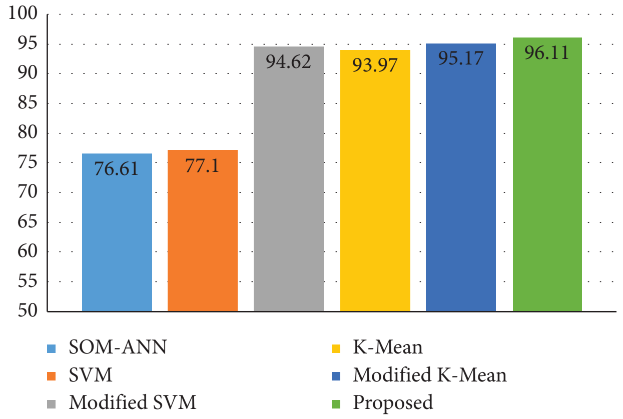


Figure 6: Accuracy analysis of different models.

5. Conclusion

The current security methods could be more efficient in avoiding security faults. An intrusion detection process has become an important part of network security. Hence, the misuse recognition method cannot classify unidentified attacks. The irregularity recognition practice is utilized to detect anomalies. Anomaly-based detection is realized by increasing the accuracy rate of intrusion detection using deep learning methods. In this article, an improved methodology is proposed for accurately detecting the intrusion using a modified LSTM algorithm. The proposed system utilizes the big data analytics platform based on parallel and distributed mechanisms. The parallel and distributed platform improves the training time along with the accuracy. The classical LSTM is modified to classify, investigate, and produce estimates of the intruders based on time series data. The proposed modified LSTM is compared with traditional algorithms and their modified versions. It is evident from the results that the proposed system outperforms the existing approaches. The proposed method works well when it comes to detecting assaults. In terms of detection rate, the proposed method knocks existing techniques. The proposed system accurately classifies data as normal or abnormal. The proposed method has a detection rate of 96.11 percent, which is implausible. Filter layers like dropout ten eliminate nonsensical, noisy, and irrelevant data from the source data.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Authors' Contributions

Muhammad Babar proposed the methodology, wrote the original draft, and validated the study. Sarah Kaleem proposed the methodology, wrote the original draft, and validated the study. Adnan Sohail reviewed and edited the manuscript and formally analysed the study. Muhammad Asim reviewed and edited the manuscript and acquired the funding. Muhammad Usman Tariq reviewed and edited the manuscript and visualized the study.

Acknowledgments

The authors would like to acknowledge the support of Prince Sultan University for paying the Article Processing Charges.

References

[1] M. K. Kagita, N. Thilakarathne, T. R. Gadekallu, P. K. R. Maddikunta, and S. Singh, "A review on cybercrimes on the Internet of Things," in *Deep Learning for Security and Privacy Preservation in IoT, Signals and Communication Technology*, pp. 83–98, Springer, Singapore, 2022.

[2] I. Almomani, M. Ahmed, and L. Maglaras, "Cybersecurity maturity assessment framework for higher education institutions in Saudi Arabia," *PeerJ Computer Science*, vol. 7, p. e703, 2021.

[3] A. Sedik, O. S. Faragallah, H. S. El-sayed et al., “An efficient cybersecurity framework for facial video forensics detection based on multimodal deep learning,” *Neural Computing and Applications*, vol. 34, no. 2, pp. 1251–1268, 2022.

[4] R. Deepalakshmi, R. Vijayalakshmi, C. Sam Ruben, R. Pandiya Rajan, and J. Pradeep, “Application of artificial intelligence in cybersecurity: a detailed survey on intrusion detection systems,” in *An Interdisciplinary Approach to Modern Network Security*, pp. 1–22, CRC Press, Boca Raton, FL, USA, 2022.

[5] E. N. Witanto, Y. E. Oktian, and S.-G. Lee, “Toward data integrity architecture for cloud-based AI systems,” *Symmetry*, vol. 14, no. 2, p. 273, 2022.

[6] M. I. Talukdar, R. Hassan, M. S. Hossen, K. Ahmad, F. Qamar, and A. S. Ahmed, “Performance improvements of AODV by black hole attack detection using IDS and digital signature,” *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6693316, 13 pages, 2021.

[7] S. Sanober, I. Alam, S. Pande et al., “An enhanced secure deep learning algorithm for fraud detection in wireless communication,” *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6079582, 14 pages, 2021.

[8] H. A. Hassan, E. E. Hemdan, W. El-Shafai, M. Shokair, and F. E. A. El-Samie, “Intrusion detection systems for the internet of thing: a survey study,” *Wireless Personal Communications*, vol. 128, no. 4, pp. 2753–2778, 2022.

[9] A. Thakkar and R. Lohiya, “A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions,” *Artificial Intelligence Review*, vol. 55, no. 1, pp. 453–563, 2022.

[10] A. Kim, M. Park, and D. H. Lee, “AI-IDS: application of deep learning to real-time Web intrusion detection,” *IEEE Access*, vol. 8, pp. 70245–70261, 2020.

[11] A. Palshikar, “What distinguishes binary from multi-class intrusion detection systems: observations from experiments,” *International Journal of Information Management Data Insights*, vol. 2, no. 2, Article ID 100125, 2022.

[12] D. A. Kumar and S. Venugopalan, “Intrusion detection systems: a review,” *International Journal of Advanced Research in Computer Science*, vol. 8, no. 8, pp. 356–370, 2017.

[13] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, A. Al-Nemrat, and S. Venkatraman, “Deep learning approach for intelligent intrusion detection system,” *IEEE Access*, vol. 7, pp. 41525–41550, 2019.

[14] I. Idrissi, M. Azizi, and O. Moussaoui, “Accelerating the update of a DL-based IDS for IoT using deep transfer learning,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 23, no. 2, pp. 1059–1067, 2021.

[15] Z. Ahmad, A. Shahid Khan, C. Wai Shiang, J. Abdullah, and F. Ahmad, “Network intrusion detection system: a systematic study of machine learning and deep learning approaches,” *Transactions on Emerging Telecommunications Technologies*, vol. 32, no. 1, Article ID e4150, 2021.

[16] S. E. Whang, Y. Roh, H. Song, and J.-G. Lee, “Data collection and quality challenges in deep learning: a data-centric ai perspective,” *The VLDB Journal*, vol. 32, no. 4, pp. 791–813, 2023.

[17] A. Khan, S. H. Khan, M. Saif, A. Batool, A. Sohail, and M. Waleed Khan, “A survey of deep learning techniques for the analysis of COVID-19 and their usability for detecting omicron,” *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 2023, pp. 1–43, 2023.

[18] S. Sharma and A. Kaul, “A survey on Intrusion Detection Systems and Honeypot based proactive security mechanisms in VANETs and VANET Cloud,” *Vehicular Communications*, vol. 12, pp. 138–164, 2018.

[19] R. Yang, R. Wakefield, S. Lyu et al., “Public and private blockchain in construction business process and information integration,” *Automation in Construction*, vol. 118, Article ID 103276, 2020.

[20] A. Stewart, *The Community Defense Approach: A Human Approach to Cybersecurity for Industrial and Manufacturing Systems*, University of Cincinnati, Cincinnati, OH, USA, 2019.

[21] L. Balke, “China’s new cybersecurity law and US-China cybersecurity issues,” *Santa Clara Law Review*, vol. 58, p. 137, 2018.

[22] A. H. Hamamoto, L. F. Carvalho, L. D. H. Sampaio, T. Abrão, and M. L. Proença, “Network anomaly detection system using genetic algorithm and fuzzy logic,” *Expert Systems with Applications*, vol. 92, pp. 390–402, 2018.

[23] V. Gowdhaman and R. Dhanapal, “An intrusion detection system for wireless sensor networks using deep neural network,” *Soft Computing*, vol. 26, no. 23, pp. 13059–13067, 2021.

[24] M. Ozkan-Okay, R. Samet, Ö. Aslan, and D. Gupta, “A comprehensive systematic literature review on intrusion detection systems,” *IEEE Access*, vol. 9, pp. 157727–157760, 2021.

[25] Gadal, S. M. Ali Mohamed, and R. A. Mokhtar, “Anomaly detection approach using hybrid algorithm of data mining technique,” in *Proceedings of the 2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, pp. 1–6, IEEE, Khartoum, Sudan, January, 2017.

[26] W.-C. Lin, S.-W. Ke, and C.-F. Tsai, “CANN: an intrusion detection system based on combining cluster centers and nearest neighbors,” *Knowledge-Based Systems*, vol. 78, pp. 13–21, 2015.

[27] S. Peddabachigari, A. Abraham, C. Grosan, and J. Thomas, “Modeling intrusion detection system using hybrid intelligent systems,” *Journal of Network and Computer Applications*, vol. 30, no. 1, pp. 114–132, 2007.

[28] V. Hajisalem and S. Babaie, “A hybrid intrusion detection system based on ABC-AFS algorithm for misuse and anomaly detection,” *Computer Networks*, vol. 136, pp. 37–50, 2018.

[29] L. S. Liu and H. Y. Fu, “Intrusion detection model based on multi-category feature fusion,” in *Proceedings of the International Conference on Frontiers of Electronics, Information and Computation Technologies*, pp. 1–5, Yangzhou, China, May, 2021.

[30] Y. Mirsky, D. Tomer, Y. Elovici, and A. Shabtai, “Kitsune: an ensemble of autoencoders for online network intrusion detection,” 2018, https://arxiv.org/abs/1802.09089.

[31] G. Bovenzi, G. Aceto, D. Ciuonzo, V. Persico, and A. Pescapé, “A hierarchical hybrid intrusion detection approach in IoT scenarios,” in *Proceedings of the GLOBECOM 2020-2020 IEEE Global Communications Conference*, pp. 1–7, IEEE, Taipei, Taiwan, December, 2020.

[32] I. Guarino, G. Bovenzi, D. Di Monda, G. Aceto, D. Ciuonzo, and A. Pescapé, “On the use of machine learning approaches for the early classification in network intrusion detection,” in *Proceeding sof the 2022 IEEE International Symposium on Measurements & Networking (M&N)*, Padua, Italy, July, 2022.

Hindawi
Security and Communication Networks
Volume 2023, Article ID 7827823, 20 pages
https://doi.org/10.1155/2023/7827823

Research Article

KTSDroid: A Framework for Android Malware Categorization Using the Kernel Task Structure

Saneeha Khalid, Khalid Imran, and Faisal Bashir Hussain

Bahria University, Islamabad, Pakistan

Correspondence should be addressed to Saneeha Khalid; saneeha.nust@gmail.com

Received 11 October 2022; Revised 3 November 2022; Accepted 24 November 2022; Published 13 May 2023

Academic Editor: Hammad Afzal

The penetration of malicious applications in the Android market has enhanced the significance of designing malware mitigation systems for Android. Malware detection systems are being developed by examining applications using static and dynamic analysis techniques. The use of code obfuscation has highlighted the importance of dynamic analysis as many static analysis schemes can be evaded by code obfuscation strategies. In order to record the true working of the application, a volatile memory-based solution for application analysis is presented in this study. Time-based memory dumps are collected after interactions with an application. Process-specific artifacts of the application under analysis are extracted by examining the kernel task structure of memory. The features in the kernel task structure belong to nine broad categories based on their semantics. An important contribution of the study is the analysis of the kernel task structure for determining the set of effective categories and features for Android malware categorization. Three of the most important categories and fourteen valuable features are reported. The proposed system categorizes the applications into five classes: adware, banking Trojans, riskware, SMS Trojans, and benign. The proposed system is able to categorize applications with an average F1-score of 0.984, which is the highest score reported so far for multiclass Android malware categorization with a minimum number of kernel task structure-based features.

1. Introduction

The tremendous rise in smartphone usage has transformed working patterns all over the world. Many business and personal tasks are performed using smartphones as they are considered more accessible and easier to use as compared to other devices. The adaptability of smartphones all over the world is due to the highly efficient and usable operating systems such as Android, iOS, and Windows. Android is the most used operating system for smartphones and holds a major market share of 71.45 percent (https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/). The success of Android can be attributed to the large number of Android-compatible user applications. These applications are frequently used by users and are considered reliable by a large population. The malicious application developers take advantage of the usage and popularity of Android and are developing a large number of malicious applications for the platform. According to statistics, 10.5 million Android malware infections were detected in 2019 and 0.48 million new Android malware infections per month were found in 2020 (https://www.statista.com/statistics/680705/global-android-malware-volume/).

The increasing rate of malware penetration in Android is a serious threat [1]. Therefore, many schemes have been proposed to mitigate this issue. Signature-based schemes have dominated malware detection techniques, but their major drawback is the inability to detect zero day malware. Signature-based schemes are also known to be less efficient against malware variants [2]. A more generic approach to malware detection and categorization is the creation of generic behavior patterns. Machine learning-based methods are used for creating generic patterns; however, the selection of useful and significant features is important for creating effective classification systems.

In order to analyze malicious Android applications using machine learning approaches, static and dynamic analysis techniques can be used [3]. Static analysis refers to analyzing an application by examining its structure and code without execution. Static analysis becomes less effective when the applications use code obfuscation techniques for hiding the code semantics [4].

Obfuscation refers to arranging the structure of code in a way that reverse engineering becomes difficult [5]. Common obfuscation schemes include class encryption, code reordering, reflection, junk code insertion, and control flow modification [6]. In order to minimize the effect of obfuscation, dynamic analysis techniques are widely being used at present. Dynamic analysis refers to analyzing the application by executing it in a sandbox. These techniques extract the runtime activity of applications; therefore, they are more resilient against obfuscation techniques [7].

Dynamic analysis helps analyze the runtime behavior of an application with the help of network activity [8], runtime API usage [9], and volatile memory usage [10]. The usage of volatile memory for extracting dynamic features has gained significant attention in recent past, as the true working of the application is visible by extracting memory artifacts. Also, code obfuscation schemes become ineffective, as memory-based artifacts show the actual essence of the executed code.

Many recent studies [11–13] have highlighted the importance of volatile memory-based artifacts for Android malware detection. Process metadata features present in the kernel task structure of memory represent an important source of information for malicious application detection [12, 14, 15]. The kernel task structure is used by the operating system for managing the running processes. It contains all the information about the running application which is needed by the kernel for managing the process. The information in the kernel task structure is contained in a number of features, which are grouped into nine categories: task_state, mem_info, scheduling_info, signal_info, process_credentials, I/O_statistics, openfile_info, CPU_specific_state, and others. These categories contain related features as per their semantics. In addition to direct features, the categories of the kernel task structure also contain a number of structures (structs) that can be traversed for extracting deep features. However, existing studies lack the in-depth working on these features in terms of extraction and analysis. It has been observed that only initial categories and structures are investigated for extraction of features. The analysis of all categories for the selection of most relevant features for malware identification and categorization needs to be thoroughly investigated. Additionally, existing studies have focused on the evaluation of process metadata features for the detection of malicious applications only. The evaluation of these features for categorizing malicious applications into respective classes is not performed.

In this study, KTSDroid, a malware mitigation framework based on process-specific artifacts from volatile memory, is proposed. The proposed framework captures volatile memory dumps while executing the application under analysis. The application's behavior profile is generated by analyzing process-specific artifacts (process metadata) from the kernel task structure in memory. KTSDroid extracts direct features from all nine categories of the kernel task structure. In addition to direct features, the nine categories of the kernel task structure are traversed up to a depth of six levels for the generation of a feature set. In order to ascertain that the extracted features contain useful information about the malicious behavior of the application, a time-based memory dump extraction process is conducted. In addition to this, random events are generated on the application before the capture of each dump to ensure interactions. As a result, four dumps with interactions are generated for each application in the dataset. Each dump is then utilized for the extraction of the kernel task structure for the process (application) under analysis. Overall, the contributions of the study are as follows:

(1) The study proposes a kernel task structure-based Android malware categorization framework by utilizing multiple time-based memory dumps with interactions.

(2) The kernel task structure of memory is utilized for the extraction of features. To the best of our knowledge, this is the first study to explore nine categories of the kernel task structure for the extraction of features w.r.t. the Android platform. In addition to this, traversal of each category to a depth of six levels is performed. A comprehensive feature set comprising 526 process specific features, grouped into nine distinct categories, is used for analysis.

(3) The effectiveness of kernel task structure features is reported against five distinct Android application classes, i.e., adware, banking Trojans, riskware, SMS Trojans, and benign.

The rest of the paper is organized as follows: details of the kernel task structure are discussed in Section 2. Section 3 presents the related work. Proposed methodology for feature extraction and selection is presented in Section 4. Results for the proposed methodology are reported in Section 5. Section 6 discusses the results, and finally, conclusion is presented in Section 7.

2. Overview of the Kernel Task Structure

KTSDroid utilizes the kernel task structure of memory for feature set extraction. In order to device an effective malware mitigation strategy, the understanding of the design and layout of the kernel task structure is important. This section briefly introduces the functions of this structure and highlights the categories in which features are organised within it.

The kernel task structure, also known as process control block, is a data structure in the kernel space of memory that contains important information about the running processes. The operating system uses this structure for managing all the running processes by dynamically allocating the structure to each process. In order to analyze the task structure of a particular process, the PID of the process can be used. The information in the kernel task structure is effective in identification and classification of malware because it describes the target application in running state and hence overcomes the problems caused by code obfuscation techniques. Information contained in the Android kernel task structure can be grouped into nine categories. Features from all these categories are utilized by KTSDroid for malware categorization. The information contained in each category is listed in Table 1.

3. Related Work

Recently, the design of memory-based schemes for Android malware categorization has gained substantial attention due to their strong resilience against various obfuscation schemes. Many studies [13, 16, 17] have associated the effectiveness of memory-based artifacts for malicious Android application detection with their ability to represent the runtime execution of the application. Different strategies are adapted by researchers to analyse memory for malicious application detection. Some of the studies [10, 18] have utilized volatile memory dumps against malicious applications in the form of images and classified them on the basis of differences in images. However, the complete memory dump contains a number of other processes as well, and the analysis is not specific to the process (application) under evaluation. Memory-based features are used by [19] for classifying Android applications into benign and malicious classes. Thirty two features using different plugins for volatility are extracted and used for analysis. The study has not incorporated time-based capturing of features, and the proposed dataset is built using only one memory snapshot. Another approach is the use of process metadata features, available in the kernel task structure of memory. These features contain useful information about the process behavior and can be used to create a malware detection system. This section highlights the memory-based frameworks that have used process metadata features from the kernel task structure for application analysis, as they are closer to the approach presented by this study.

Wang and Li [12] proposed a framework for the detection of malware on the Android platform using machine learning with a feature set from the kernel task structure. A total of 112 features grouped into 5 categories are extracted from the task struct using 1275 malware samples and 1275 benign samples. Principal component analysis, chi-squared statistic, correlation, and information gain are used as dimension reduction methods. The proposed framework is evaluated by using 4 different machine learning algorithms (Naive Bayes, decision tree, neural network, and K-nearest neighbors). It is shown that the proposed framework can achieve 94% to 98% accuracy and less than 10% false positive rate.

Alawneh et al. [14] proposed a malware detection system that can identify trojanized malware. The focus of the study is on improved detection time rather than the accuracy of the model. In the experiment, 112 fields were extracted from the kernel process control block and grouped into five categories, including mem_info, CPU_scheduling_info, signal_info, task_state, and others. For dataset creation, a total of 2400 apks (1200 benign and 1200 malware) were used. Features were recorded for 15 seconds for each apk. The study is evaluated by using a back propagation neural network (BPNN). The model is evaluated by considering the feature set, and it is reported that the best result is achieved by selecting 43 features out of 112, with 96.8% detection accuracy, which takes around 30 seconds for training the classifier and 73 μs for malware detection after 100 ms of information mining.

Shahzad et al. [15] proposed a real-time malware detection framework, namely, TstructDroid, for Android-based devices. 110 benign and 110 malicious applications are used for dataset creation. Out of 99 preliminary task structure fields, 32 fields are shortlisted for dataset creation using time series feature shortlisting techniques. After shortlisting features, time series blocks are created and then frequency information is calculated using the discrete cosine transform. The framework achieves a detection rate of 90-93.6% with a false alarm rate between 5.4% and 7.3%.

Kim and Choi [20] proposed a malware detection method for the Android platform. Features are extracted from the proc filesystem of the Linux platform. The proc filesystem is a virtual filesystem which provides an interface to the kernel structures, i.e., it permits communication between the user space and the Linux kernel. A total of 36 features out of 59 features are selected from three classes: memory, CPU, and network. Feature extraction was performed periodically every 10 seconds, against the applications being executed. Support vector machine (SVM) is used as a classifier for performance evaluation in the experimental setup. A TPR of 95.97%, an FPR of 0.67, a precision of 96.63%, and an accuracy of 98.85 were achieved after feature selection. The results are computed for six applications only.

A summary of the related work on kernel task structure-based malware detection is presented in Table 2. It is pertinent to highlight that the studies have only classified the applications into malicious and benign categories. Determining the category or nature of the malware is significant for understanding the criticality of threat and effective mitigation. Another important observation is the availability of a large number of kernel task structure-based features in Android, whereas previous works have focused on a limited set of features.

4. KTSDroid Android Malware Categorization Using the Kernel Task Structure

KTSDroid is an Android malware detection and categorization framework that uses dynamic memory information extracted from the programs' kernel task structure. The architecture of KTSDroid is shown in Figure 1 that has three components: feature extraction, feature selection, and classification. During feature extraction, initially, Android applications are executed and interactions are made to obtain process-specific memory dumps. These dumps are further analyzed for extracting process metadata features from the kernel task structure. In the second phase, a step-by-step approach is adapted for investigating significant categories and features. Finally, the selected features are used

Table 1: Information in categories of the kernel task structure.

KTS category	Information
task_struct	State of the process like exit code and process execution domain
mem_info	Major and minor page faults, heap address of the process, start and end address of code segment, and start and end address of data segment
scheduling_info	Priority of the process, scheduling state, scheduling policy, execution time, waiting time, snapshot of user, and system CPU time
signal_info	Signal sources, the signal handler, and timers related to the process
process_credentials	Ownership and process capabilities
I/O_statistics	Block I/O delay and I/O statistics like number of byte read, number of read system call, and number of write system calls
openfiles_info	Opened files related to the process like maximum number of file descriptor and opened file descriptor
CPU_specific_state	CPU state of the process, which includes different register states and fault info
Others	Miscellaneous information about the process like age of the process and tracer information

to classify the applications into respective benign or malware classes. In the remaining of this section, the aforementioned three core components of the KTSDroid framework are discussed in detail.

4.1. KTSDroid Feature Extraction. KTSDroid uses a dynamic analysis scheme based on volatile memory artifacts for application analysis. This section covers the details of the memory-based feature extraction process. The dataset under analysis consists of N apk files, and each apk belongs to a class C_i, where i ranges from 1 to 5. In order to formulate a malware detection system using these apks, each apk must be processed to extract volatile memory-based features.

The overall process of feature extraction can be divided into two major steps: memory dump extraction and process metadata extraction by traversing the kernel task structure. The details are provided in the subsections as follows, and an overall view of the feature extraction process is shown in Figure 2.

4.1.1. Memory Dump Extraction. The dynamic nature of the proposed system requires the applications to be executed in a controlled environment for evaluation. For this purpose, an AVD (Android virtual device) environment is chosen, where the AVD is configured for application installation and memory dump extraction. The AVD is created with the Nexus 6P hardware profile and Android 9.0 (Google APIs) system image with x86_64 architecture. The host system is Ubuntu 18.04, and communication between the host system and the virtual device is carried out by using ADB (Android debug bridge).

The features used by the system are based on memory; therefore, memory dump extraction is the first step in application analysis. In order to extract a memory dump, LiME (Linux Memory Extractor) (https://github.com/504ensicsLabs/LiME), a loadable kernel module needs to be compiled for the target kernel of AVD and loaded into the device. LiME is an open-source loadable kernel module (LKM) for Linux-based devices, which allows the acquisition of complete volatile memory dumps. For the LiME module to be loaded by the target kernel, the kernel needs to be compiled with loadable kernel module support for the target device. It is because Android does not have default support for loadable kernel modules. KTSDroid uses Goldfish kernel 4.4, compiled with loadable kernel module support.

KTSDroid uses a Python application for automating the process of scanning a local directory containing a dataset of apk files, installing the application (apk file) on the virtual device, simulating pseudorandom user input using monkey (https://developer.android.com/studio/test/other-testing-tools/monkey) against the installed application and capturing a memory dump of the virtual device. For analysis, a total of four volatile memory dumps were obtained. The first dump was taken right after the application installation. The sunsequent dumps were taken after event generation on the apk. The events were triggered using Monkey and included 150, 1500, and 4000 events. Acquiring volatile memory dumps during different states of the application adds variability to the dump and hence contributes to a rich data collection that comprises 10,000 memory dumps. Simulating user inputs is required to ensure code coverage and triggering of the malicious behavior. Installation and execution of applications (benign/malicious) are performed while the device is running in the read-only mode. Running the device in the read-only mode is required to keep changes made by the applications nonpersistent and ensure the device is in a clean state after every restart. Although process-specific features are considered during the study, but to ensure smooth execution of the application and the device, a single application is considered for installation and analysis at a time.

4.1.2. Process Metadata Extraction. The memory dump extraction process produces a set of four memory dumps for each apk in the data set. To extract digital artifacts from these volatile memory dumps, the Python application is extended to use the volatility framework (https://www.volatilityfoundation.org/) as a library. Volatility is an open-source collection of tools used for the analysis of volatile memory samples of Mac, Windows, Linux, and Android-based devices. In order to use volatility for the analysis of the memory dump of the target Android device,

Table 2: Related work on malware detection using kernel task structure features.

Study	Number of effective features	Feature selection techniques	Algorithm classification	Reported performance	Data set size	Multiclass classification	KTS category analysis
Wang and Li [12]	10–40 out of 112	PCA, correlation, IG, and chi-square	Naïve Bayes, decision trees, and neural network	ACC: 94%–98%	1275 malware, 1275 benign	×	Partial
Alawneh et al. [14]	43 out of 112	Logistic regression	Neural network	ACC: 96.80%	1200 malware, 1200 benign	×	×
Shahzad et al. [15]	32 out of 90	Correlation	Decision trees (J48)	ACC: 93%–96%	110 malicious, 110 benign	×	×
Kim and Choi [20]	36 out of 59	Manual	Support vector machines (SVM)	ACC: 98.85%,	6 malicious	×	×

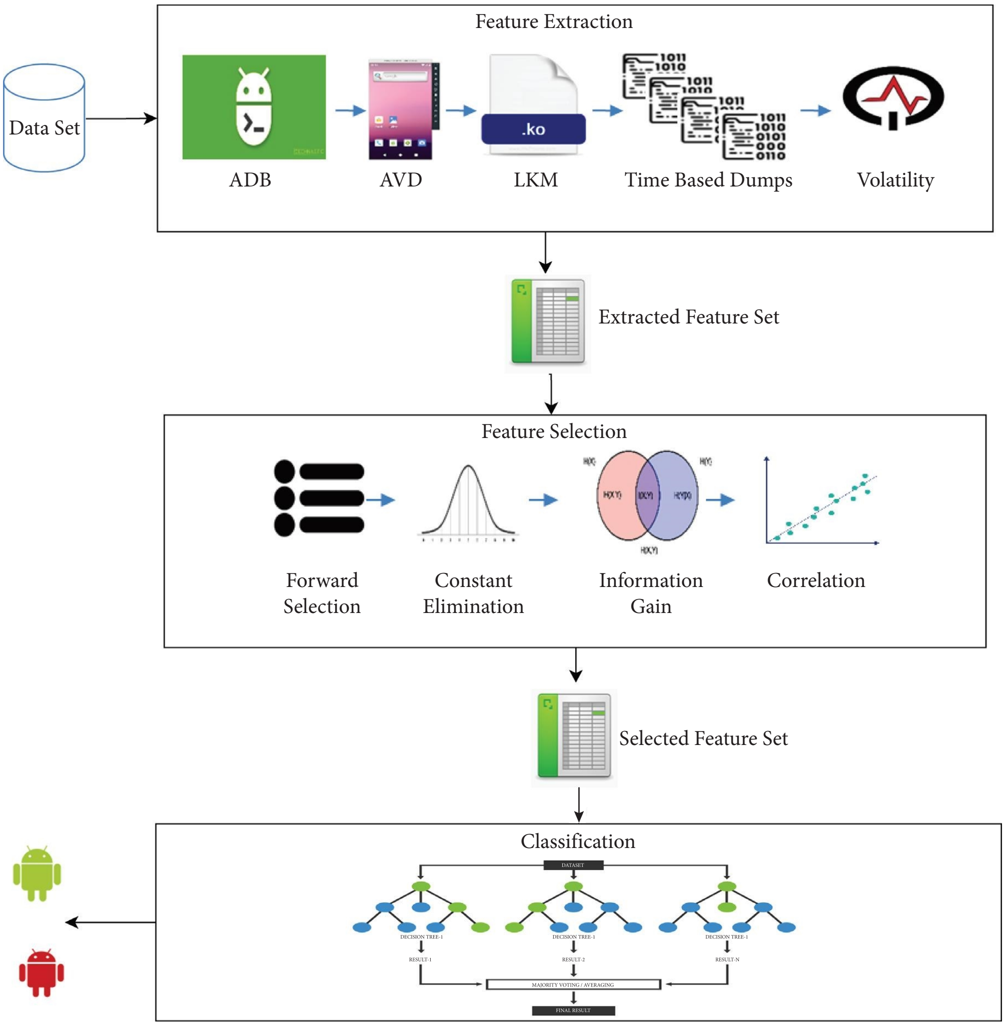


Figure 1: KTSDroid framework for feature extraction, feature selection, and classification.

a profile for the target kernel is generated. The profile is used by the volatility framework for locating and parsing information in the memory dump. The Linux_pslist plugin of volatility, which collects active tasks by walking through the kernel task structure, is utilized for extracting the features of the running application.

The kernel task structure is a combination of many structs, as each field in the structure may also be a struct containing other fields. This makes it a cascaded structure with a lot of information about the running process. The Python-based application iterates the kernel task structure six levels deep for the extraction of features. The depth of features explored in this study is shown in Figure 3. The extracted information is recorded into a csv file with columns representing the features and rows representing the memory dumps. Each record consists of 526 fields from the kernel task structure.

4.2. KTSDroid Feature Selection. The feature extraction process produces a rich set of features for each apk in the dataset. In order to design an effective malware categorization system, all of these features need to be analyzed for their significance in malware detection. For this purpose, a feature selection process is designed to gauge the importance of features. Feature selection is an important part of formulating a machine learning-based system, as using significant features positively impacts the performance of the classification system. Feature selection helps improve the training time, reduce the complexity of the model, and improve performance. Feature selection is also a useful way of handling overfitting which results in enhanced model generalization [21].

Feature selection methods can be divided into two broad categories: wrapper-based methods and filter-based methods. Wrapper-based methods use a classification algorithm to find the effectiveness of features, and filter-based methods use statistical techniques to find the importance of a feature in output prediction [22]. This study uses methods from both of these techniques for finalizing the set of important features.

The feature extraction process iterates through the kernel task structure to extract a rich set of 526 features. These features belong to nine categories as per the general categorization of the kernel task structure. As the number of

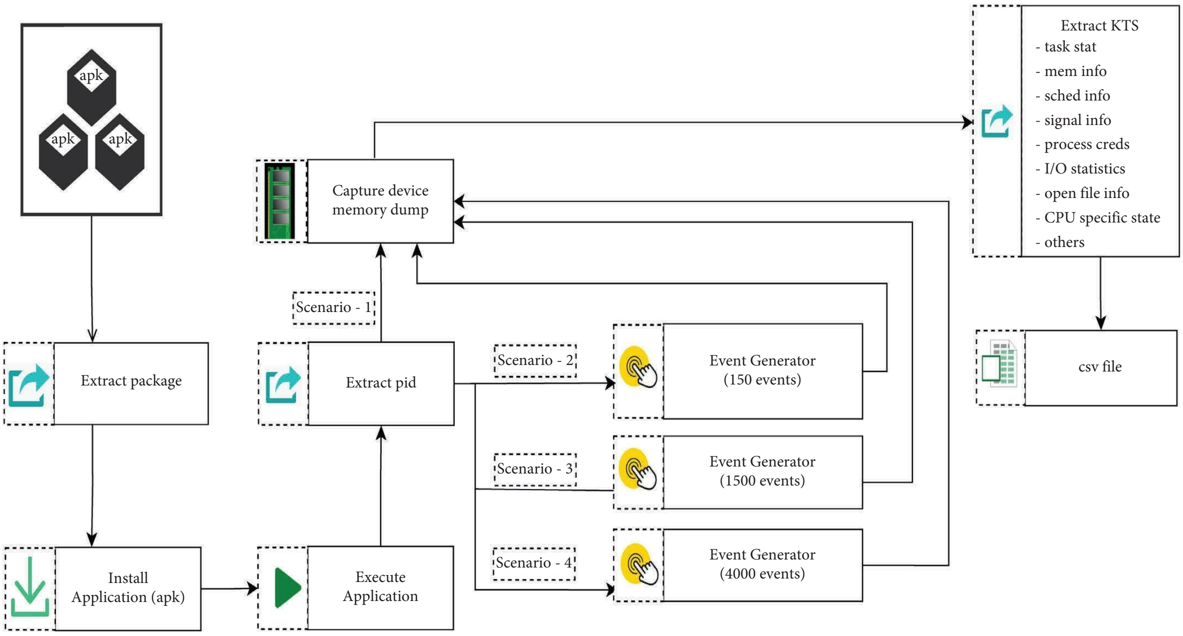


Figure 2: KTSDroid feature extraction process.

features is quite large, selecting the most effective features is significantly important. The feature selection approach used in this study comprises two steps. In the first step, all categories of the kernel task structure are analyzed to find the categories that are not significant for malware categorization. The reason for using this strategy is that all the features in a certain category are related to each other semantically. It may be possible that the information present in a certain category is not an effective identifier of malicious behavior. Therefore, finding the significant categories that contain information related to malware identification is important. Once important categories are identified, the selected categories are parsed to find the most significant features for malware categorization. Details of these steps are described in the subsections as follows.

4.2.1. Significant Kernel Task Structure Category Identification. The first step of feature selection is the identification of significant categories of the kernel task structure for malware categorization. For this purpose, the set of all features is grouped category-wise and labeled for each class. All categories are then represented by the set $\mathrm{CT}^{\mathrm{all}}$, where $\mathrm{CT}^{\mathrm{all}} = \{ct_1, ct_2, ct_3, ct_4, ct_5, ct_6, ct_7, ct_8, ct_9\}$. Each category, ct_i, contains a number of features, as shown in Table 3.

The initial process of feature selection focuses on evaluating the significance of a complete category instead of evaluating each feature individually because the features in each category are semantically related to each other. For example, mem_info contains features related to memory usage, and IO_statistics refer to features related to IO operations. As the features are semantically related, a broader landscape of feature importance can be extracted by looking into the significance of each category for classification.

In order to find the set of significant categories, $\mathrm{CT}^{\mathrm{sig}}$, where $\mathrm{CT}^{\mathrm{sig}} \subseteq \mathrm{CT}^{\mathrm{all}}$, a wrapper-based selection method, is used. Wrapper methods use the evaluation metrics of the classification model to find the best set of features. Features are supplied to the classification system, and performance is measured. The set of features that report the highest performance in optimal time is selected [22]. Many wrapper-based feature selection methods are available. For this study, the wrapper method of forward selection is used. It utilizes a model and threshold value of performance measures to find the best set of features. In forward selection, a classification model is created for each feature in the dataset. The model's performance is recorded, and the best performing feature is selected. In each step, the next best feature is added to the model and the process continues. This method is very resource-intensive, as there are many features in a dataset, and testing each of them one by one is a time- and resource-intensive task [23]. However, in this experiment, the complexity of forward selection is reduced to only nine features ($\mathrm{CT}^{\mathrm{all}} = \{ct_1, ct_2, ct_3, ct_4, ct_5, ct_6, ct_7, ct_8, ct_9\}$) as a complete category of features is considered at a time. This reduces time and resource complexity by a large amount.

While applying wrapper-based methods, the selection of a suitable classification algorithm according to the type of data is important. It should also be considered that in all wrapper methods, classification is performed a number of times on subsets of features; therefore, the process must be concluded when a certain threshold for performance is achieved. In this work, the model used for classification in forward selection is random forest. Random forest is an ensemble of decision trees. It is recommended as a classifier

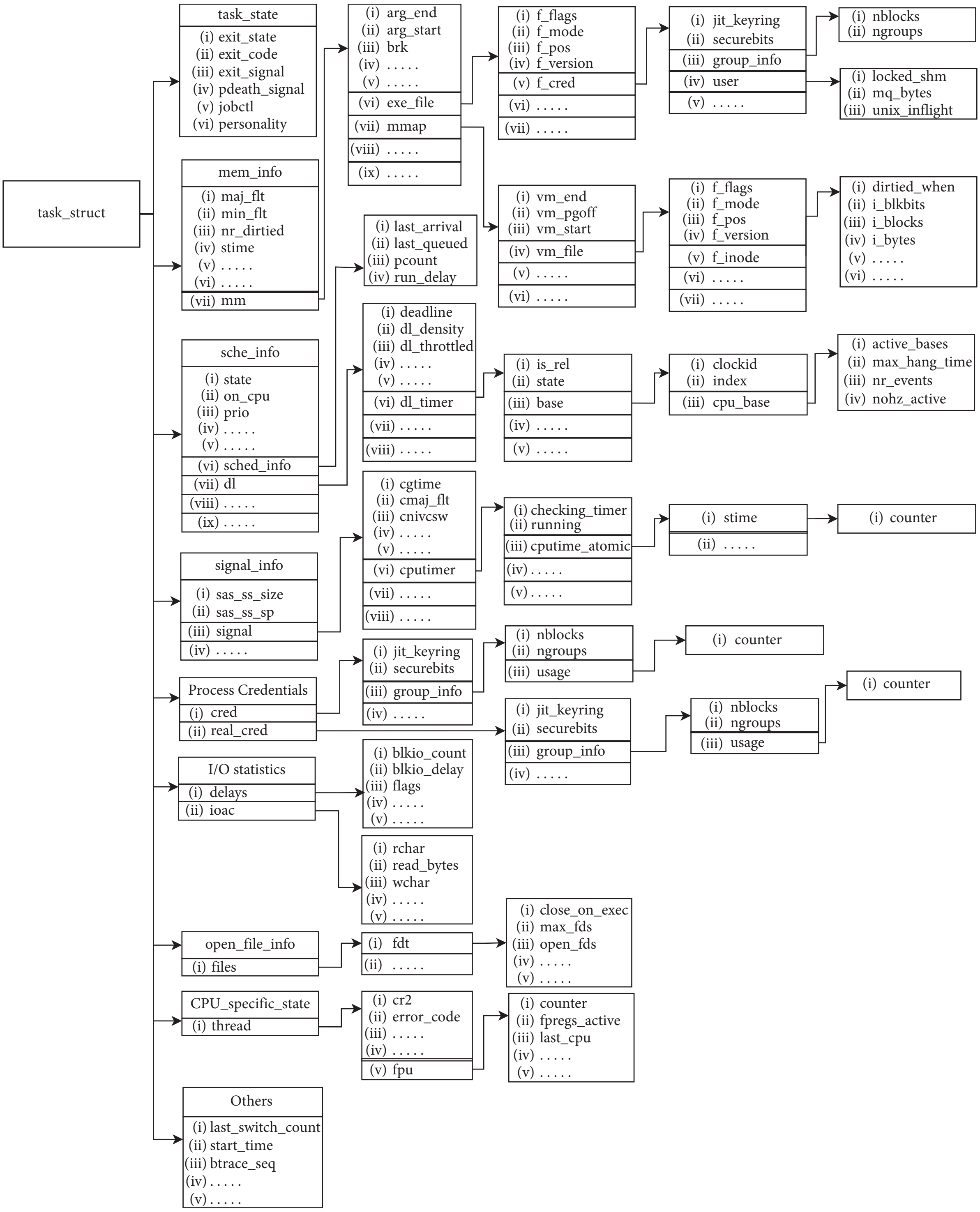


Figure 3: Kernel task structure-extracted categories and features.

for Android malware detection by a number of studies [24–26]. The iterations of measuring classification results are terminated when the performance metrics become constant and adding new categories does not add to the performance enhancement. After applying forward selection, the set of all categories $\mathrm{CT}^{\mathrm{all}}$ is reduced to a smaller set $\mathrm{CT}^{\mathrm{sig}}$, where $\mathrm{CT}^{\mathrm{sig}} = \{ct_i \ldots\ldots ct_m\}$ and m is the number of significant categories for malware classification.

4.2.2. Significant Feature Selection. After the identification of important categories of the kernel task structure, the next step is to find the most minimal and effective feature set for Android malware categorization. For this purpose, a three-phase process is used. In the first phase, features are analyzed to find the set of constant features. All constant features are identified and dropped from further analysis. In the second phase, the remaining set of features is evaluated for their

Table 3: Number of features in kernel task structure categories.

KTS category	Number of features
task_state	5
mem_info	212
scheduling_info	91
signal_info	83
process_credentials	75
I/O_statistics	18
openfile_info	12
CPU_specific_state	20
Others	17

mutual information (MI) values against the output class. Features with insignificant MI values are not considered for further analysis. Finally, in the last phase, the dimensions of the data are further reduced by removing the linearly correlated features. All phases of feature selection are applied to each category separately because of the following two reasons:

(1) Evaluating the features category wise makes the results manageable and easy to understand

(2) The final result of feature selection summarizes the contribution of each category of the kernel task structure in the final feature set.

(1) Phase 1: Constant Feature Elimination. The feature for which the values remain the same for all classes is referred to as a constant feature. These features increase the dimensionality of the feature set and can be a cause of the slow convergence of the training algorithm [22]. Using these features has no effect on the predictive power of the model; only the complexity of the model increases. Therefore, such features can be dropped from the dataset. In order to find constant features, a statistical measure of variance can be used [27]. Variance measures the variability in the values of a variable and can be used to check the constant nature of a feature. It measures the spread in the values of a variable by calculating the average squared distance from the mean. Variance is widely used in feature selection by setting a threshold value for the variability of values against a feature. In this study, we are interested in finding features that have no variance in values; therefore, the threshold value is 0. If the value for variance is zero, it indicates that the feature is constant and can be dropped from further analysis.

KTSDroid groups the extracted features category-wise; therefore, all features in a category, ct_i, where $ct_i \in \mathrm{CT}^{\mathrm{sig}}$ are evaluated for variance. If f_{ij} is the j^{th} feature in the i^{th} significant category, then equation (1) illustrates the process of feature selection through variance.

$$f_{ij}^{\mathrm{temp}} = \left\{ \begin{array}{ll} f_{ij} & \mathrm{var}\left(f_{ij}\right) > 0 \\ \phi & \text{otherwise} \end{array} \right\}, \qquad (1)$$

where i ranges from $1, \ldots, m$ and m is the number of selected categories; j ranges from $1, \ldots, n$ and n is the number of features in the i^{th} category.

In equation (1), the feature f_{ij} is tested using variance as given by equation (2). It is selected as f_{ij}^{temp} if the value for variance is greater than zero.

$$\mathrm{var} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(x_i - \overline{x}\right)^2}. \qquad (2)$$

Initially, F_i^{sel} is an empty set for i^{th} category. After testing all features in a category, the selected features are added to set F_i^{sel}, as shown in the following equation:

$$F_i^{sel} \leftarrow \bigcup_{j=1}^{n'} f_{ij}^{\mathrm{temp}}, \qquad (3)$$

where n' is the number of nonconstant features.

Constant feature elimination is the most basic step of feature selection. The feature set after constant feature elimination is stored for further analysis using mutual information in phase two of feature selection.

4.2.3. Phase 2: Mutual Information (MI). Mutual information, also referred to as information gain, is one of the most commonly used filter-based methods for feature selection [28]. Its working is based on entropy, and it measures the reduction in entropy after the transformation of a dataset. It estimates the dependency between a feature and the output. In this work, mutual information is chosen for feature significance evaluation as it estimates linear as well as nonlinear relationships between feature and output as compared to other univariate feature selection methods like F-test that only finds the linear dependencies between two variables [29]. Another important point to be considered is the size of the features to be evaluated. The features need to be tested individually; therefore, an efficient algorithm in terms of time and resource usage must be used. Mutual information can be easily applied to a large number of features because of its lower complexity and computation time [30].

Mutual information helps in finding significant features by estimating the dependency of output on a feature. Significant features are always related to the output in some way. However, features that are insignificant in the prediction of output always have low or no dependency on output. If the value of mutual information for a feature is zero, it indicates that the output is independent of the feature. Higher values show a higher dependency between the feature and the output.

The feature set produced by constant feature elimination F_i^{sel} for each category, as shown by equation (3), is evaluated to find the value of mutual information against all features. If f_{ij} is the j^{th} feature in the i^{th} category, then equation (4) represents the application of mutual information and the selection of significant features based on a threshold T.

$$f_{ij}^{\mathrm{temp}} = \left\{ \begin{array}{ll} f_{ij}^{sel} & \mathrm{MI}\left(f_{ij}^{sel}\right) > T \\ \phi & \text{otherwise} \end{array} \right\}, \qquad (4)$$

where i ranges from $1, \ldots, m$ and m is the number of selected categories; j ranges from $1, \ldots, n^{'}$ and $n^{'}$ is the number of nonconstant features in the i^{th} category. T is the threshold value.

$$\mathrm{MI}\left(f_{ij}^{sel}; C\right) = H\left(f_{ij}^{sel}\right) \check{} H\left(f_{ij}^{sel}|C\right). \tag{5}$$

In equation (4), f_{ij}^{sel} is tested for mutual information with respect to output class C by using equation (5). The feature, f_{ij}^{sel}, is selected as f_{ij}^{temp} if the value for mutual information is greater than threshold T. The set of all selected features (f_{ij}^{temp}) is moved to set F^{temp}, as shown in equation (6). Finally, the set of selected features, F_i^{sel}, is replaced by the set of selected features using mutual information as shown in the following equation:

$$F_i^{\mathrm{temp}} \leftarrow \bigcup_{j=1}^{n^{''}} f_{ij}^{\mathrm{temp}}, \tag{6}$$

where $n^{''}$ is the number of features with the MI score greater than threshold T

$$F_i^{temp} \leftarrow F_i^{\mathrm{temp}}. \tag{7}$$

In order to select the features based on mutual information, the selection of a threshold value (T) is extremely important, as features with MI scores greater than the threshold will be included in the F_i^{sel} set. In order to find the optimal value for T, the feature set is evaluated at different threshold values $(T = 0.0, 0.1, \ldots, t)$ against a performance metric. The threshold value starts from zero and is increased by 0.1 for feature set selection. The process stops for $T = t$, where t is a threshold value for which the performance becomes constant or starts decreasing. The complete algorithm for feature selection for all categories using mutual information is shown in Algorithm 1.

After the selection of features using mutual information, the updated feature set F_i^{sel} for each category is analyzed to find linearly correlated features.

4.2.4. Phase 3: Correlation. Correlation is a statistical measure for finding the linear dependency between two variables. For feature selection, correlation can be used to find features, which have a strong linear relationship [31] among themselves. It is beneficial, as two features having a strong linear dependency on each other will have almost the same effect on the output variable. Therefore, they may be replaced by any one of them. This helps reduce the dimensionality of the data and the complexity of the model [32].

KTSDroid computes correlation for all features within a selected category. F_i^{sel} is the set of features obtained after eliminating features with mutual information values less than the desired threshold. If f_{ij}^{sel} and f_{ik}^{sel} represent two features from the set of selected features for a category, then the application of correlation can be described by equation (8). Here, one of the features is selected as f_{ij}^{temp} if the two have a correlation value of 0.95.

$$f_{ij}^{\mathrm{temp}} = \left\{ \begin{array}{ll} f_{ij}^{sel} & \mathrm{corr}\left(f_{ij}^{sel}, f_{ik}^{sel}\right) > 0.95 \\ \phi & \mathrm{otherwise} \end{array} \right\}. \tag{8}$$

If f_{ij}^{sel} and f_{ik}^{sel} are represented by $f1$ and $f2$, then correlation between the two features can be defined by the following equation:

$$corr = \frac{\sum_{i=1}^{n^{''}}\left(f1_i - \overline{f1}\right)\left(f2_i - \overline{f2}\right)}{\sqrt{\sum_{i=1}^{n^{''}}\left(f1_i - \overline{f1}\right)^2\left(f2_i - \overline{f2}\right)^2}}, \tag{9}$$

where $n^{''}$ is the number of features with the MI score greater than threshold T

Equation (9) refers to the person coefficient for finding correlation. A value close to 1 indicates a positive linear relationship, and a value close to −1 indicates a negative linear relationship. A value of 0 indicates that both features are linearly independent. The set of linearly correlated features is grouped into a set F^{temp} by using equation (7). Finally, all correlated features are removed from the selected feature set using the following equation:

$$F_i^{sel} \leftarrow F_i^{sel} - F_i^{\mathrm{temp}}. \tag{10}$$

4.3. KTSDroid Classification. The selected set of features after applying a number of feature selection techniques can now be evaluated for performance using a classification model. Random forest (RF) is a classification model that uses decision trees as the underlying base classifier. It works by creating a number of trees and later accumulating the results from each. Each tree is generated using bagging and a bootstrap sample of data [33]. Given a training set D, bagging generates M new training sets D_i, where D_i is generated from D uniformly and with replacement. Sampling with replacement means that in each set D_i, some features will be unique and some will duplicate. After the generation of trees, the final result is obtained by either averaging, weighted averaging, or voting [34]. RF has a low bias as the trees are unpruned and fully grown. The correlation among the trees is also low; each tree is built independent of its peers [26].

KTSDroid used RF for the evaluation of the selected feature set. It has been chosen for classification as the detection of malware is a rule formation problem, and RF generates a number of rule sets in the form of trees. It is also not prone to overfitting and does not require retraining to fine-tune a large number of parameters. Overall, it is an efficient ensemble classification model with low bias and variance. Many malware analysis studies have also reported higher performance measures with RF as compared to other classifiers [24–26]. In this work, the final result of classification is obtained by voting on the results of candidate trees.

5. Experiments and Results

The dataset used for this study is CICMalDroid 2020 [35], developed by the Canadian Institute of Cyber Security. Applications belonging to five distinct classes: adware,

```
procedure Extracting_Significant_Features_using_MI
  for $i \longleftarrow 1$ to $m$ do
    Load the set of all features in the category $(F_i^{sel})$
    //$F_i^{sel}$ is the feature set after constant featureelimination
    $F_i^{\text{temp}} = []$
    $T = 0.0$ //$T$ is the threshold
    Previous_$F1$_Score = 0.94
    //$F1$_score after removing constant features
    Current_$F1$_Score = 0
    while Current_$F1$_Score < = Previous_$F1$_Score do
      for$j \longleftarrow 1$ to $F_i^{sel}$.len() do
        $f_{ij}^{\text{temp}} \longleftarrow f_{ij}^{sel}$
Score = Mutual_Information $(f_{ij}^{\text{temp}}, C)$
        if Score $< T$ then
          $F_i^{\text{temp}}$.append$(f_{ij}^{\text{temp}})$
        end if
      end for
      Current_$F1$_Score $\longleftarrow$ $F1$_Score$(F_i^{\text{temp}}, C)$
      if Current_$F1$_Score $\geq$ Previous_$F1$_Score then
        $T = T + 1$
      end if
    end while
    $F_i^{sel} \longleftarrow F_i^{\text{temp}}$
  end for
end procedure
```

Algorithm 1: Algorithm used by KTSDroid for feature selection using mutual information.

banking Trojans, riskware, SMS Trojans, and benign, are selected for analysis. The data set consists of 3000 applications, where each malware class has 500 samples and each benign class has 1000 samples. As the data set consists of apk files, each apk needs to be processed in order to extract useful artifacts. The feature extraction process executes each apk and extracts four memory dumps with interactions. The time-based memory dump extraction ensures the capture of malicious activity. Each dump is processed to traverse the kernel task structure for the extraction of features. Nine categories of the kernel task structure are traversed six levels deep to generate a comprehensive dataset of 526 features. Overall, the dataset used by the study consists of a total of 12,000 records, with 2000 records against each malware class and 4000 records against the benign class. A summary of dataset details is presented in Table 4.

The comprehensive analysis of kernel task structure results in a rich set of features. KTSDroid analyzes these features by using a number of feature selection techniques. The final set of selected features is then used for classification. The remaining part of this section discusses the results of all applied feature selection techniques and classification. The results are organised according to the sequence of applied techniques. Initially, the results for finding the most significant categories are reported. Afterwards, the results for finding the most important features from the significant categories using a three-phase process are shown. Finally, classification results on the final feature set for categorizing applications into five classes are presented.

5.1. Significant Kernel Task Structure Category Identification.

The feature extraction process results in the generation of a large feature set for each apk in the dataset. In order to identify important features, a feature selection approach depicted in Figure 4 is adopted. In the first step of feature selection, the set of all categories $(\mathrm{CT}^{\text{all}})$ is analyzed to find the set of significant categories $(\mathrm{CT}^{\text{sig}})$. For this purpose, a wrapper-based method of forward selection is used. The results of the first iteration of forward selection for selecting the most significant category are shown in Table 5.

It can be inferred from the results that the most significant category for malware categorization in terms of the F1-score for all malware classes is mem_info. Therefore, it is selected in the first iteration. All categories are then added for evaluation after sorting them by their individual F1-scores from the first iteration. The process of forward selection includes many iterations of feature combinations. The result of each step is not shown in the paper, as many results are intermediate and keep changing when new categories are added. An overall summary of important results of the forward selection process is shown in Figure 5. From the graph, it can be observed that the best performance is achieved by combining mem_info, process_credentials and signal_info categories. Combining these three categories helps in achieving an average F1-score of 0.95. It should also be noted that adding other categories does not significantly improve the performance of the system.

The application of forward selection for category analysis reduces the set of all categories CT^{all} to a set of three significant categories. This process reduces the overall size of

Table 4: Dataset details.

Application class	Number of samples	Number of memory dumps	Events generated	Total samples
Adware	500	4	0, 150, 1000, 4000	2,000
Banking Trojans	500	4	0, 150, 1000, 4000	2,000
Riskware	500	4	0, 150, 1000, 4000	2,000
SMS Trojans	500	4	0, 150, 1000, 4000	2,000
Benign	1000	4	0, 150, 1000, 4000	4,000

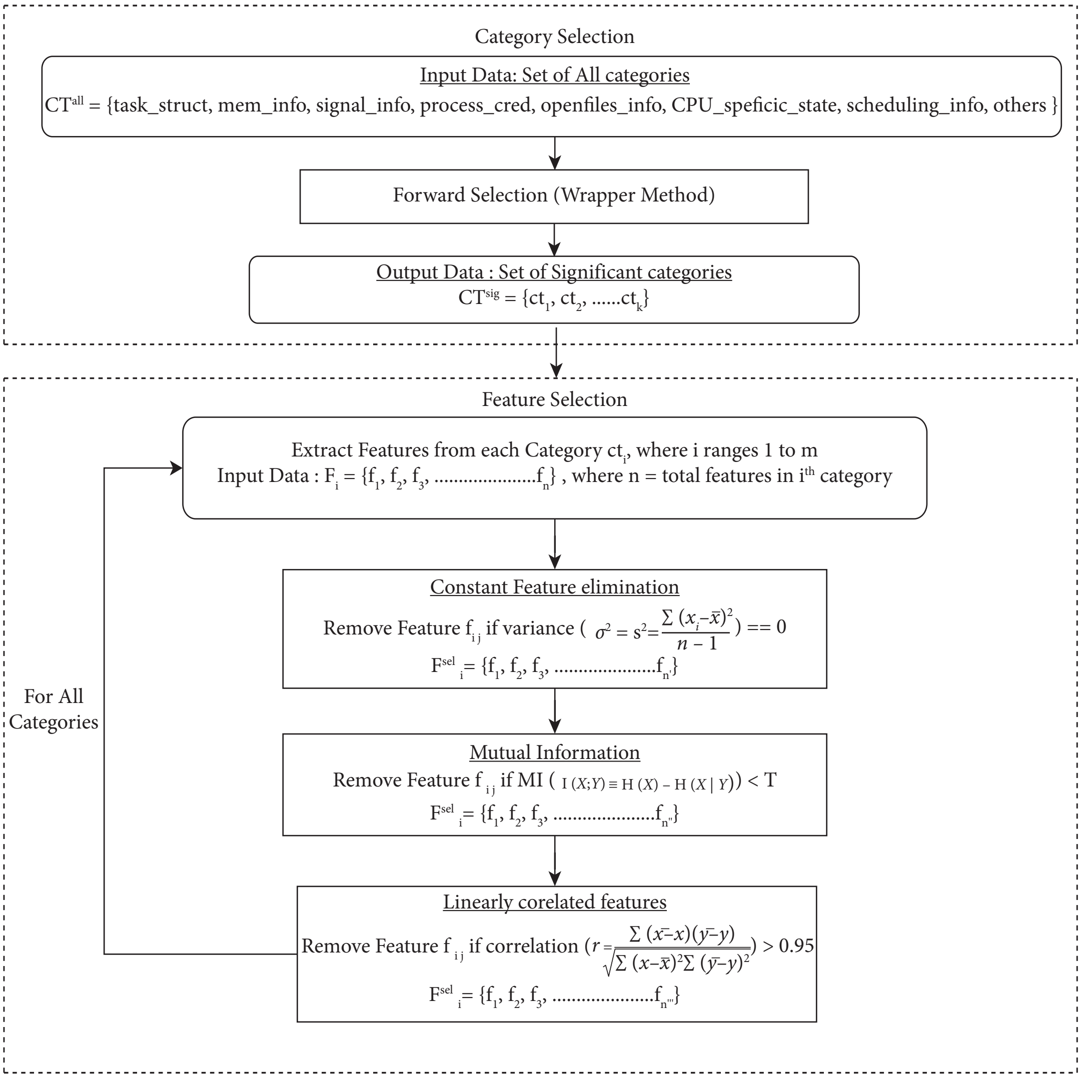
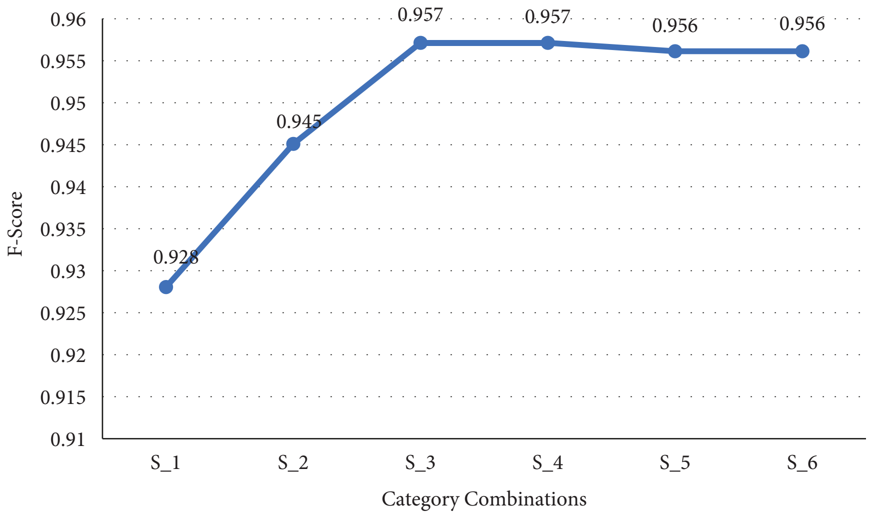

Category Selection
Input Data: Set of All categories
CT^all = {task_struct, mem_info, signal_info, process_cred, openfiles_info, CPU_speficic_state, scheduling_info, others }
Forward Selection (Wrapper Method)
Output Data : Set of Significant categories
CT^sig = {ct1, ct2, ...ctk}
Feature Selection
Extract Features from each Category cti, where i ranges 1 to m
Input Data : Fi = {f1, f2, f3, ...fn} , where n = total features in ith category
Constant Feature elimination
Remove Feature fij if variance (σ² = s² = Σ (xi−x̄)² / (n − 1)) == 0
F^sel_i = {f1, f2, f3, ...fn'}
Mutual Information
Remove Feature fij if MI (I (X;Y) ≡ H (X) − H (X | Y)) < T
F^sel_i = {f1, f2, f3, ...fn''}
Linearly corelated features
Remove Feature fij if correlation (r = Σ (x̄−x)(ȳ−y) / √(Σ (x−x̄)² Σ (ȳ−y)²)) > 0.95
F^sel_i = {f1, f2, f3, ...fn'''}
For All Categories

Figure 4: KTSDroid feature selection process.

Table 5: Results for finding the most significant KTS category using forward selection.

KTS category	F1-score				
	Adware	Banking Trojan	Riskware	Trojan SMS	Benign
task_state	0	0.3	0	0	0
mem_info	0.928	0.927	0.937	0.92	0.92
signal_info	0.768	0.891	0.775	0.834	0.761
process_credentials	0.926	0.912	0.939	0.904	0.963
Scheduling_info	0.597	0.797	0.503	0.70	0.55
IO_statistics	0.668	0.759	0.732	0.767	0.658
Openfile_info	0.531	0.665	0.616	0.758	0.686
CPU_state	0.843	0.756	0.699	0.789	0.702
Others	0.304	0.441	0.252	0.284	0.290



Combination Name	Categories in a combination
S-1	mem_info
S-2	mem_info and process_credentials,
S-3	mem_info, process_credentials and signal info,
S-4	mem_info, process_credentials, signal_info, and openfiles_info,
S-5	mem_info, process_credentials,signal_info, openfiles_info and IO_statsistics
S-6	mem_info, process_credentials, signal_info, openfiles_info and IO_statsistics

Figure 5: Forward selection results for significant category identification.

the dataset by 30 percent. The set of significant categories can now be defined as

$$\mathrm{CT}^{\mathrm{sig}} = \left\{ct_{\mathrm{mem\_info}}, ct_{\mathrm{process\_credentials}}, ct_{\mathrm{signal\_info}}\right\}. \quad (11)$$

Further improvement in performance can be achieved by finding the most important and effective features from the selected categories, which are discussed in the next section.

5.2. Significant Feature Selection. In order to find significant features, the features in each of the selected categories are evaluated using a three-phase process. The results of each phase are discussed as follows.

5.2.1. Phase 1: Constant Feature Elimination. The first phase of significant feature selection focuses on constant feature evaluation and removal from the selected categories of mem_info, process_credentials, and signal_info. The set of nonconstant features for a category F_i^{sel} is found by using equations (1) and (2). The results of applying constant feature elimination on each category are shown in Table 6.

Table 6: Constant feature elimination results.

KTS category	Total features	Constant features	Remaining features
Mem_info	212	131	81
Process_credentials	75	34	47
Signal_info	83	27	56

The results show that a large number of constant features are present in all selected categories. It indicates that a substantial number of features show the same behavior for malicious and benign applications. It can be inferred that these features are indicative of the general working of the application and are not specific to the malicious actions. After removing these constant features, the size of the dataset is further reduced by 49 percent. Now, the remaining features will be gauged based on their mutual information values against the output class.

5.2.2. Phase 2: Mutual Information (MI) for Feature Selection. Mutual information measures the significance of a feature by estimating the dependency of output on the feature. After constant feature elimination, the features in each category are evaluated for mutual information scores by using equations (4) and (5). The results of mutual information against all features in the selected categories are shown in Figures 6–8.

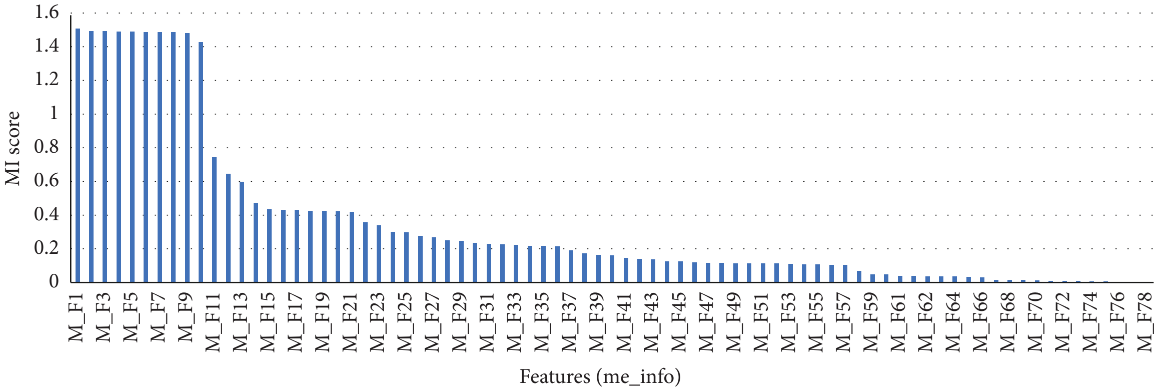


Figure 6: MI scores for mem_info features.

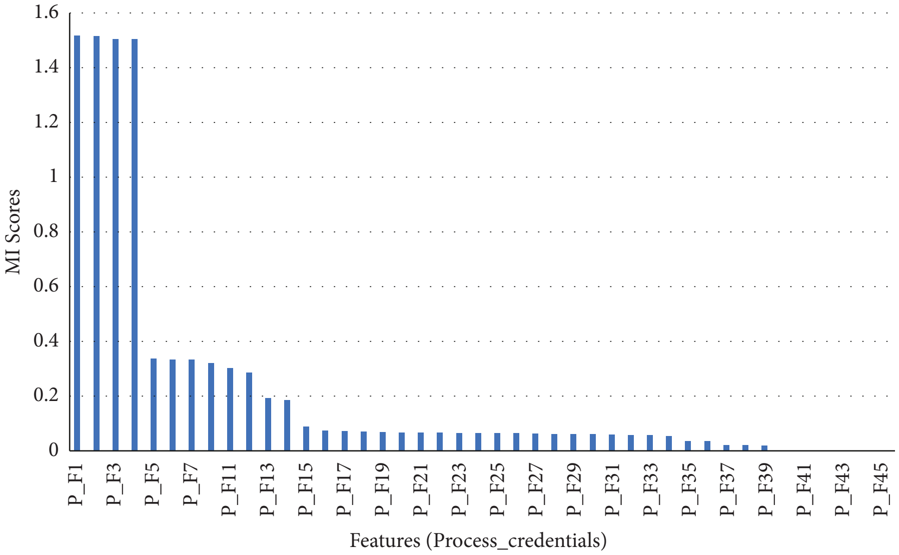


Figure 7: MI scores for process_credentials features.

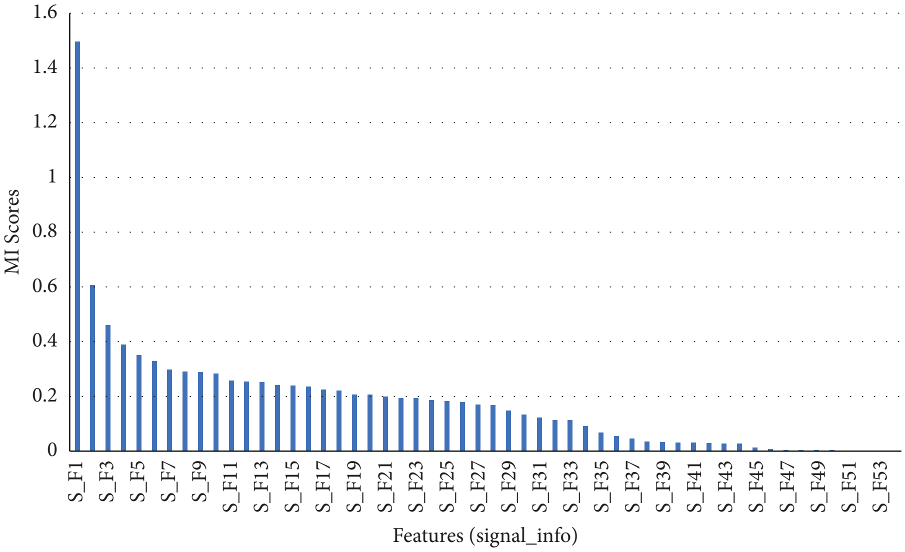


Figure 8: MI scores for signal_info features.

Table 7: Performance for MI thresholds.

Threshold for MI score (T)	Features in mem_info	Features in process_credentials	Features in signal_info	F1-score
0.0	81	47	47	0.942
0.1	56	14	33	0.956
0.2	35	12	19	0.964
0.3	24	11	6	0.974
0.4	21	4	3	0.987
0.5	13	4	2	0.980
0.6	12	4	1	0.980

From the results, it can be seen that many features have very low values for mutual information. It means that the dependency between these features and output is very less. In order to drop features with low MI scores, a step-by-step approach is used. Features are not dropped abruptly; instead, they are dropped based on a certain threshold value (T) of MI scores. Initially, the threshold (T) is set to 0.1. All features below the MI score of 0.1 are dropped, and performance in terms of the F1-score is measured. If there is improvement in performance, the threshold value T is updated by a factor of 0.1 and the process of performance measurement is repeated. The updation of the threshold value T is stopped until it reaches a value t at which the performance becomes constant.

The result of feature reduction at each threshold value and corresponding performance is shown in Table 7. It can be observed that at $T = 0.6$, the performance becomes constant, so the iterations for threshold updation can be stopped. Observing the performance against all thresholds reveals that the best performance is reported at a threshold value of 0.4. Therefore, the features at a threshold value of 0.4 are selected for further analysis. It can be observed that the number of features is now reduced to twenty-eight, of which twenty-one belong to mem_info, four belong to process_credentials, and three belong to signal_info. The names of selected features along with their MI scores are shown in Table 8.

5.3. Phase 3: Correlation for Feature Selection. The selection of features based on mutual information greatly reduces the size of the feature set. However, as two linearly correlated features have the same output, one of them can be dropped from analysis in order to reduce the dimensionality of the data. For this purpose, correlation is calculated for all features in a category. Two features are considered correlated if the value of the correlation coefficient as represented in Eqn. (9) is 0.95. The correlation matrix for all features in selected categories is shown in Figures 9–11. The number of correlated features and the size of the final set of features after the removal of correlated features is shown in Table 9.

The final set of features, after the removal of correlated features, now contains fourteen features in total. Table 10 shows the name, category, and depth of each feature in the kernel task structure.

5.4. Classification. The application of feature selection results in a reduced set of effective features. Features belonging to each category are now combined together and classified using random forest. In order to apply random forest, the length of each tree and the number of trees in the ensemble need to be assessed. KTSDroid uses unpurged trees to incorporate all features, as the feature set is now reduced to a manageable size. In order to estimate the number of trees inside the ensemble of random forest, the performance on the final feature set is evaluated for a different number of trees. The performance is evaluated for three, five, seven, nine, and eleven trees. It is observed that the highest average F1-score of 0.985 is reported for nine trees; therefore, the number of tress in the random forest ensemble is set to nine.

In order to ascertain the effectiveness of feature selection approaches, the performance is calculated at all steps of feature selection, i.e., forward selection (FS), constant feature elimination (CFE), mutual information (MI), and correlation (Corr). It can be observed from Figures 12 and 13 that each stage of feature selection results in the reduction of the feature set and the enhancement of performance. This highlights the preciseness of the proposed feature selection scheme represented in Figure 4.

One of the important contributions of the study was to evaluate the significance of kernel task structure features for multiclass classification for Android applications. For this purpose, the performance measures are analyzed for each class individually. Table 11 and Figure 14 show the performance of the final feature set for each class. The table shows that adware, riskware, and benign application types are classified by an F1-score of 0.99, and banking and SMS Trojans are classified by a 0.96 F1-score. The high rate of detection is proof of the effectiveness of memory-based solutions for Android malware categorization in general and kernel task structure-based features in particular.

KTSDroid is compared with two studies based on memory-based artifacts for Android malware analysis. Comparison is conducted in terms of the explored categories of the kernel task structure, number of features, number of output classes, and performance. The comparison shown in Table 12 highlights that KTSDroid has explored the maximum number of categories from the kernel task structure and reported an accuracy of 0.985 using the least number of features for five output classes.

Table 8: Selected features using mutual information (MI).

Rep	Feature name	MI score
M_F1	task -> mm -> mmap -> vm_file -> f_inode -> i_generation	1.51
M_F2	task -> mm -> mmap_base	1.50
M_F3	task -> mm -> brk	1.50
M_F4	task -> mm -> mmap_legacy_base	1.49
M_F5	task -> mm -> start_brk	1.49
M_F6	task -> mm -> end_data	1.49
M_F7	task -> mm -> start_code	1.49
M_F8	task -> mm -> start_data	1.49
M_F9	task -> mm -> end_code	1.48
M_F10	task -> mm -> mmap -> vm_file -> f_inode -> i_ino	1.43
M_F11	task -> mm -> shared_vm	0.74
M_F12	task -> mm -> total_vm	0.64
M_F13	task -> mm -> hiwater_vm	0.59
M_F14	task -> mm -> exec_vm	0.47
M_F15	task -> mm -> env_end	0.43
M_F16	task -> mm -> start_stack	0.43
M_F17	task -> mm -> arg_end	0.42
M_F18	task -> mm -> arg_start	0.42
M_F19	task -> mm -> env_start	0.42
M_F20	task -> mm -> highest_vm_end	0.41
M_F21	task -> mm -> mm_count -> counter	0.41
P_F1	task -> cred -> session_keyring -> last_used_at	1.51
P_F2	task -> real_cred -> session_keyring -> last_used_at	1.51
P_F3	task -> real_cred -> session_keyring -> serial	1.50
P_F4	task -> cred -> session_keyring -> serial	1.50
S_F1	task -> sas_ss_sp	1.49
S_F2	task -> signal -> ioac -> rchar	0.60
S_F3	task -> signal -> ioac -> wchar	0.46
S_F4	task -> signal -> real_timer -> base -> cpu_base -> clock_was_set_seq	0.38

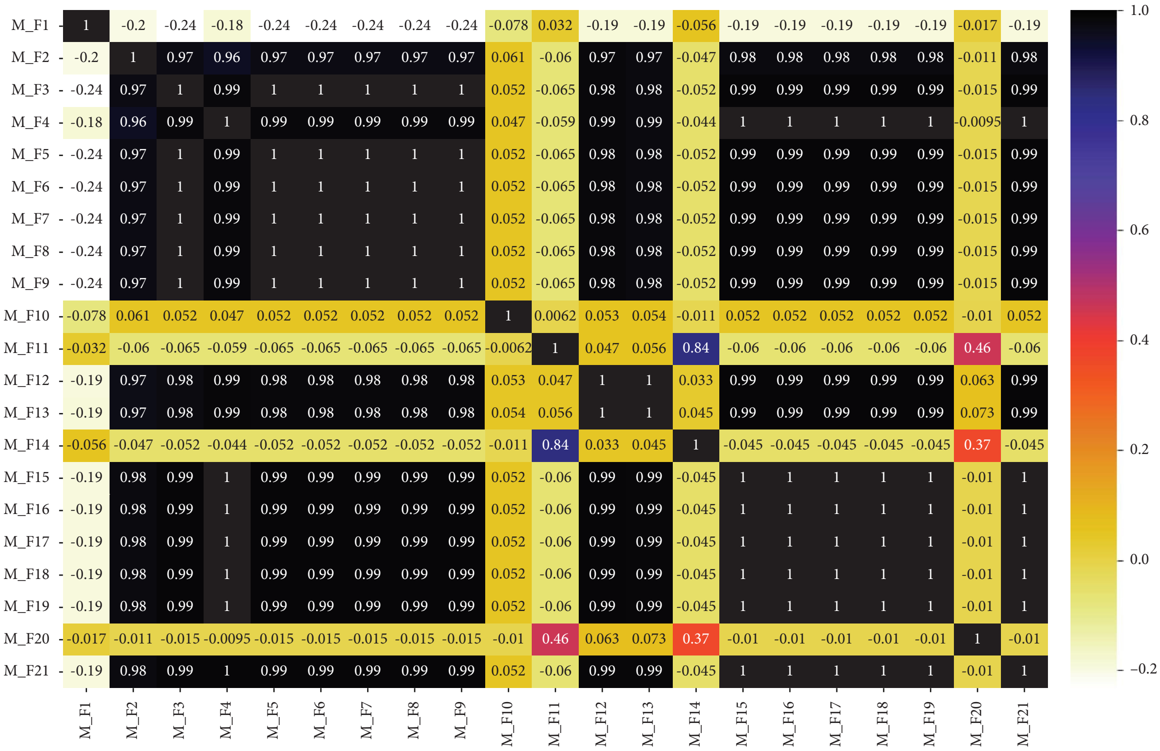


Figure 9: Correlation matrix of mem_info features.

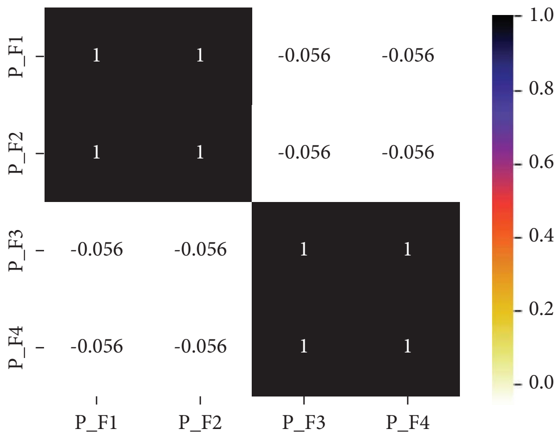


Figure 10: Correlation matrix of process_cred features.

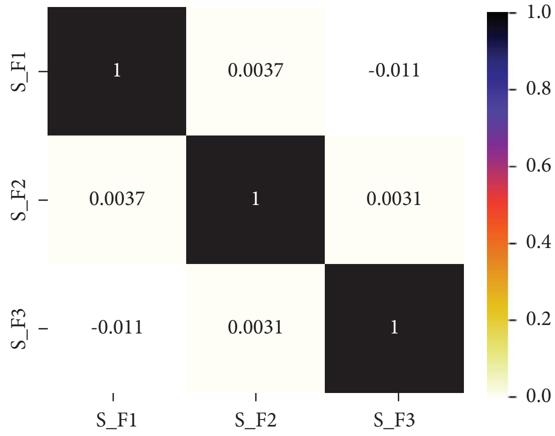


Figure 11: Correlation matrix of signal_info features.

Table 9: Correlation results.

KTS category	Original features	Correlated features	Reduced features
mem_info	21	12	9
process_credentials	2	4	2
signal_info	0	3	3

Table 10: Final feature set used by KTSDroid.

Rep	Feature name	Depth
M_F1	task -> mm -> mmap -> vm_file -> f_inode -> i_generation	6
M_F2	task -> mm -> mmap_legacy_base	3
M_F3	task -> mm -> end_data	4
M_F4	task -> mm -> mmap_base	3
M_F10	task -> mm -> mmap -> vm_file -> f_inode -> i_ino	6
M_F11	task -> mm -> shared_vm	3
M_F12	task -> mm -> total_vm	3
M_F14	task -> mm -> exec_vm	3
M_F20	task -> mm -> mm_count -> counter	4
P_F1	task -> real_cred -> session_keyring -> last_used_at	4
P_F3	task -> real_cred -> session_keyring -> serial	4
S_F1	task -> sas_ss_sp	2
S_F2	task -> signal -> ioac -> rchar	4
S_F3	task -> signal -> ioac -> wchar	4

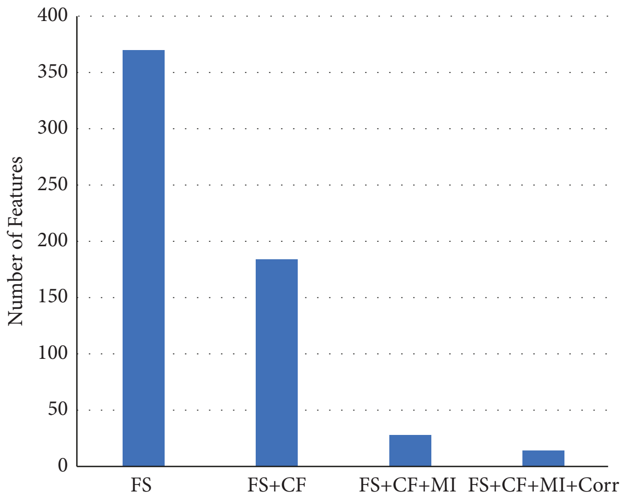


Figure 12: Dimensionality reduction by feature selection methods.

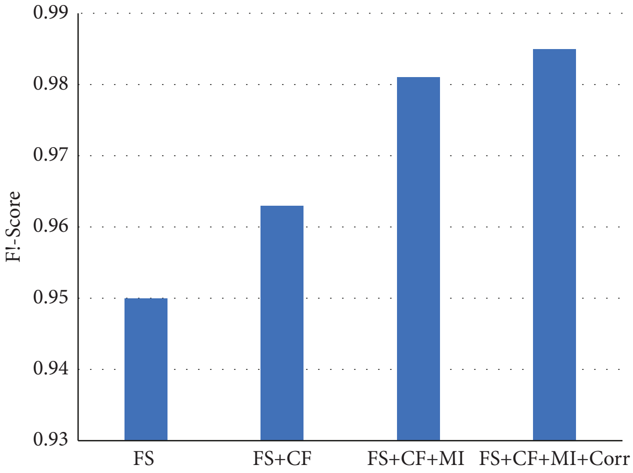


Figure 13: Performance improvements by feature selection methods.

Table 11: KTSDroid performance for malicious and benign classes.

Class	F1-score	Precision	Recall
Adware	0.992	0.988	0.996
Banking Trojans	0.967	0.972	0.962
Riskware	0.992	0.989	0.995
SMS Trojans	0.968	0.969	0.968
Benign	0.993	0.994	0.992

6. Discussion

KTSDroid analyzes the effect of memory-based features on Android malware categorization. The kernel task structure of memory is used for the extraction of process-specific features. It is thoroughly analyzed for nine categories up to a depth of six levels, as compared to existing studies that have worked with five categories for a depth of three. A large number of features are extracted, which are then evaluated for significance. KTSDroid uses a minimal set of fourteen features and is able to classify malicious applications with high performance. The high performance of KTSDroid can be attributed to the following important points:

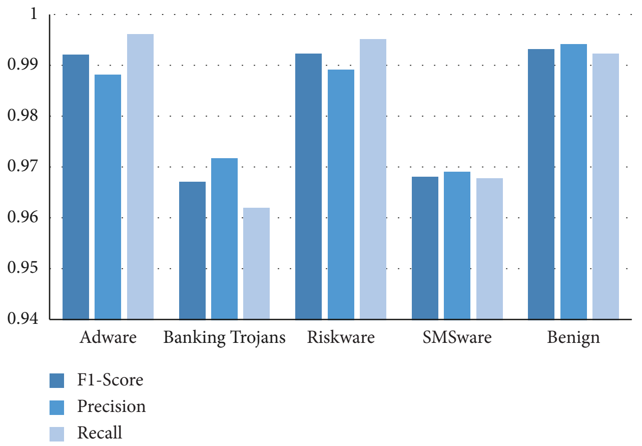


Figure 14: KTSDroid performance for multiclass application categorization.

Table 12: KTSDroid comparison with existing studies.

	Significant categories	Number of features	Number of output classes	Performance
Wang et al.	2 out of 5	40	2	0.98 (*F*1-score)
Tstructdroid	—	32	2	98 (accuracy)
KTSDroid	3 out of 9	14	5	0.985 (*F*1-score)

(1) Process-specific features from the kernel task structure are used for creating behavior profiles for applications. These features are better representative of the application's behavior as compared to general memory usage features, as they are shared by a number of processes.

(2) Five additional categories of the kernel task structure are explored for feature extraction by KTSDroid. Among these, the category of process_credentials is found to be the second most important for malware categorization by the forward selection method, as shown in Table 5. Features from the category of process_credentials are included in the final feature set.

(3) The deep exploration of the kernel task structure enables the extraction of features beyond the depth of three (as per previous studies). The features beyond the depth of three constitute seventy-one percent of the final feature set. The high percentage of features from deeper structures of the kernel task structure highlights the importance of traversing deeper levels of the kernel task structure.

7. Conclusion

Dynamic analysis-based solutions for malware analysis have replaced static analysis solutions due to the inability of static analysis to explore the runtime working of the application. This study has proposed a dynamic analysis-based malware categorization system that extracts volatile memory-based artifacts for malicious Android application detection and categorization. A time-based memory dump extraction process with interactions is conducted to ensure the capture of malicious actions of the applications. The kernel task structure from all memory dumps is analyzed for the extraction of process-specific features. A large number of process-specific features grouped into nine categories are extracted. A comprehensive analysis is conducted on the extracted set of features to find the most important categories of the kernel task structure for malware categorization. The most significant features of the selected categories are also reported in the study. The proposed system is able to classify malicious applications into five distinct classes by using a small number of features with high performance.

Data Availability

The data supporting the findings of this study are available on the following git repository: https://github.com/saneehaAmir/KTSDroid.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] P. Stirparo, I. N. Fovino, and I. Kounelis, "Data-in-use leakages from Android memory — test and analysis," in *Proceedings of the 2013 IEEE 9th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pp. 701–708, Lyon, France, October 2013.

[2] Z. Bazrafshan, H. Hashemi, S. M. H. Fard, and A. Hamzeh, "A survey on heuristic malware detection techniques," in *Proceedings of the The 5th Conference on Information and Knowledge Technology*, pp. 113–120, Shiraz, Iran, May 2013.

[3] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying Android malware using dynamically obtained features," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.

[4] Y. Ding, M. Naber, C. L. E. Paffen, J. H. Fabius, and S. Van der Stigchel, "Saccades reset the priority of visual information to access awareness," *Vision Research*, vol. 173, pp. 1–6, 2020.

[5] A. Aghamohammadi and F. Faghih, "Lightweight versus obfuscation-resilient malware detection in android applications," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 2, pp. 125–139, 2020.

[6] M. Hammad, J. Garcia, and S. Malek, "A large-scale empirical study on the effects of code obfuscations on android apps and anti-malware products," in *Proceedings of the Proceedings of the 40th International Conference on Software Engineering*, pp. 421–431, Association for Computing Machinery, New York, NY, USA, May 2018.

[7] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *Proceedings of the 2017 12th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 31–38, Fajardo, PR, USA, October 2017.

[8] M. Gohari, S. Hashemi, and L. Abdi, "Android malware detection and classification based on network traffic using deep learning," in *Proceedings of the 2021 7th International Conference on Web Research (ICWR)*, pp. 71–77, Tehran, Iran, May 2021.

[9] H. Gao, S. Cheng, and W. G. D. Zhang, "GDroid: android malware detection and classification with graph convolutional network," *Computers & Security*, vol. 106, Article ID 102264, 2021.

[10] A. S. Bozkir, E. Tahillioglu, M. Aydos, and I. Kara, "Catch them alive: a malware detection approach through memory forensics, manifold learning and computer vision," *Computers & Security*, vol. 103, Article ID 102166, 2021.

[11] H. Wang, H. He, and W. Zhang, "Demadroid: object reference graph-based malware detection in Android," *Security and Communication Networks*, vol. 2018, Article ID 7064131, 16 pages, 2018.

[12] X. Wang and C. Li, "Android malware detection through machine learning on kernel task structures," *Neurocomputing*, vol. 435, pp. 126–150, 2021.

[13] W. Zhang, H. Wang, H. He, and P. Liu, "DAMBA: detecting android malware by ORGB analysis," *IEEE Transactions on Reliability*, vol. 69, no. 1, pp. 55–69, 2020.

[14] H. Alawneh, D. Umphress, and A. Skjellum, "Android malware detection using neural networks & process control block information," in *Proceedings of the 2019 14th International Conference on Malicious and Unwanted Software (MALWARE)*, pp. 3–12, Nantucket, MA, USA, August 2019.

[15] F. Shahzad, M. Akbar, S. Khan, and M. Farooq, "Tstructdroid: realtime malware detection using in-execution dynamic analysis of kernel process control blocks on android," Technical Report, National University of Computer & Emerging Sciences, Islamabad, Pakistan, 2013.

[16] A. Ali-Gombe, A. Tambaoan, A. Gurfolino, and G. G. Richard III, "App-agnostic post-execution semantic analysis of Android in-memory forensics artifacts," in *Proceedings of the Annual Computer Security Applications Conference*, pp. 28–41, Austin, TX, USA, December 2020.

[17] Y. Dai, H. Li, Y. Qian, and X. Lu, "A malware classification method based on memory dump grayscale image," *Digital Investigation*, vol. 27, pp. 30–37, 2018.

[18] A. De Lorenzo, F. Martinelli, E. Medvet, F. Mercaldo, and A. Santone, "Visualizing the outcome of dynamic analysis of Android malware with VizMal," *Journal of Information Security and Applications*, vol. 50, Article ID 102423, 2020.

[19] A. H. Lashkari, B. Li, T. L. Carrier, and G. V. Kaur, "Volatile memory analyzer for malware classification using feature engineering," in *Proceedings of the 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, pp. 1–8, IEEE, Hamilton, Canada, May 2021.

[20] H. H. Kim and M. J. Choi, "Linux kernel-based feature selection for Android malware detection," in *Proceedings of the The 16th Asia-Pacific Network Operations and Management Symposium*, pp. 1–4, Hsinchu, Taiwan, September 2014.

[21] J. Abawajy, A. Darem, and A. A. Alhashmi, "Feature subset selection for malware detection in smart IoT platforms," *Sensors*, vol. 21, no. 4, p. 1374, 2021.

[22] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digital Investigation*, vol. 13, pp. 22–37, 2015.

[23] N. Maleki and H. Rastegari, "An improved method for packed malware detection using PE header and section table

information," *International Journal of Computer Network and Information Security*, vol. 11, no. 9, 2019.

[24] J. Jung, H. Kim, D. Shin et al., "Android malware detection based on useful API calls and machine learning," in *Proceedings of the 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 175–178, IEEE, Laguna Hills, CA, USA, September 2018.

[25] P. Agrawal and B. Trivedi, "Machine learning classifiers for Android malware detection," in *Data Management, Analytics and Innovation*, pp. 311–322, Springer, Berlin, Germany, 2021.

[26] H. J. Zhu, T. H. Jiang, B. Ma, Z. H. You, W. L. Shi, and L. Cheng, "HEMD: a highly efficient random forest-based malware detection framework for Android," *Neural Computing & Applications*, vol. 30, no. 11, pp. 3353–3361, 2018.

[27] S. Khalid and F. B. Hussain, "Evaluating dynamic analysis features for android malware categorization," in *Proceedings of the 2022 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 401–406, IEEE, Dubrovnik, Croatia, May 2022.

[28] T. A. Alhaj, M. M. Siraj, A. Zainal, H. T. Elshoush, and F. Elhaj, "Feature selection using information gain for improved structural-based alert correlation," *PLoS One*, vol. 11, 2016.

[29] A. Salah, E. Shalabi, and W. Khedr, "A lightweight android malware classifier using novel feature selection methods," *Symmetry*, vol. 12, no. 5, p. 858, 2020.

[30] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, vol. 2, Springer, Berlin, Germany, 2009.

[31] A. M. Kowshalya, R. Madhumathi, and N. Gopika, "Correlation based feature selection algorithms for varying datasets of different dimensionality," *Wireless Personal Communications*, vol. 108, no. 3, pp. 1977–1993, 2019.

[32] X. F. Song, Y. Zhang, D. W. Gong, and X. Z. Gao, "A fast hybrid feature selection based on correlation-guided clustering and particle swarm optimization for high-dimensional data," *IEEE Transactions on Cybernetics*, vol. 52, no. 9, pp. 9573–9586, 2022.

[33] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[34] I. Ahmad, M. Basheri, M. J. Iqbal, and A. Rahim, "Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection," *IEEE Access*, vol. 6, pp. 33789–33795, 2018.

[35] S. Mahdavifar, D. Alhadidi, and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of Network and Systems Management*, vol. 30, pp. 22–34, 2022.

Hindawi
Security and Communication Networks
Volume 2023, Article ID 5834434, 15 pages
https://doi.org/10.1155/2023/5834434

Research Article

DQfD-AIPT: An Intelligent Penetration Testing Framework Incorporating Expert Demonstration Data

Yongjie Wang,[1,2] Yang Li ,[1,2] Xinli Xiong,[1,2] Jingye Zhang,[1,2] Qian Yao,[1,2] and Chuanxin Shen[1,2]

[1]*College of Electronic Engineering, National University of Defense Technology, Hefei 230037, China*
[2]*Anhui Province Key Laboratory of Cyberspace Security Situation Awareness and Evaluation, Hefei 230037, China*

Correspondence should be addressed to Yang Li; ly_20d@163.com

Received 9 August 2022; Accepted 7 October 2022; Published 4 May 2023

Academic Editor: Muhammad Faisal Amjad

The application of reinforcement learning (RL) methods of artificial intelligence for penetration testing (PT) provides a solution to the current problems of high labour costs and high reliance on expert knowledge for manual PT. In order to improve the efficiency of RL algorithms for PT, existing research has considered bringing in the knowledge of PT experts and combining it with the use of imitative learning methods to guide the agent in its decision-making. However, the disadvantage of using imitation learning is also obvious; that is, the performance of the strategies learned by the agent hardly exceeds the demonstrated behaviour of the expert and it can also cause expert knowledge overfitting. At the same time, the expert knowledge in the currently proposed method is poorly interpretable and highly scenario-dependent. The expert knowledge used in these methods is not universal. To address these issues, we propose an intelligent PT framework named DQfD-AIPT. The framework encompasses the process of collecting and using expert knowledge and provides a rational definition of the structure of expert knowledge. To solve the overfitting problem, we perform PT path planning based on the deep Q-learning from demonstrations (DQfD) algorithm. DQfD combines the benefits of RL and imitation learning to effectively improve the PT strategy and performance of agents while avoiding overfitting. Finally, we conducted experiments in a simulated network scenario containing honeypots. The experimental results proved the effectiveness of expert knowledge incorporation. In addition, the DQfD algorithm can improve the efficiency of penetration testing more effectively than that by the classical deep reinforcement learning (DRL) method and can obtain a higher cumulative reward. Not only that, due to the incorporation of expert knowledge, in scenarios with honeypots, the DQfD method can effectively reduce the probability of interacting with honeypots compared to the classical DRL method.

1. Introduction

With the development of the Internet, the network environment is increasingly complex, and cyber security threats that we face are increasing day by day. Globally, protecting modern systems and infrastructure is becoming a challenge in the field of computer security. The traditional approach to system security assessment takes a defender's perspective by solidifying and enhancing system security against attackers [1]. Penetration testing is used as a positive method to attack and test a target system against authorised networks from the attacker's point of view. We can conduct vulnerability detection and security assessment through potential threat paths [2]. However, with the increase in network size and system complexity, the number of hosts in the network, and the complexity of configuration information, the efficiency of performing penetration testing will be affected by the AoI (age of information) [3, 4]. Performing PT manually involves a lot of repetitive actions and procedures [5]. As a result, automated and intelligent penetration testing was born out of this need.

Early research included automated penetration testing tools and related theoretical studies. Automated penetration testing tools integrate modules for scanning, penetration attacks, and payload selection, such as Metasploit [6]. However, human intervention is still required for the critical target identification and load selection in this tool. In terms of theoretical research, attack trees, attack graphs, and planning domain programming languages (PDDLs) are representative. Methods such as attack graphs plan attack paths through formal representation of the target network configuration information and state transfer analysis [6, 7]. However, all these methods require in-depth knowledge of the target network's information in advance and cannot model the uncertainty in the penetration testing process. Recent advances in artificial intelligence have provided a new approach to the current research on automated and intelligent penetration testing. In particular, RL methods are proving to be a general and effective approach. RL can be used to solve the problem of optimal performance of an agent in a given environment. A model-free approach to RL, for example, allows an agent to learn a strategy by interacting with the environment, with little reliance on prior knowledge of the environment. The method is analogous to a human player interacting with a game in order to complete the game objectives and learn relevant solution strategies [8].

The Markov decision process (MDP) is a paradigm of RL. Schwartz and Hanna [9] and Zhou et al. [10] trained the agent for path planning by formalising PT as MDP and using the DRL algorithm in a constructed penetration test simulation environment. The problem is that the training process takes a long time to converge and that the simulation environment is poorly simulated. Zennaro et al. [11] combine RL and imitation learning approaches and classify PT problems according to scenarios. They provide a priori knowledge to the agent for a specific network scenario structure, which better guides the agent to explore their problem space and thus obtain a better solution. On the downside, its expert knowledge is limited by the constructed penetration test scenarios and is less interpretable and generalisable. Chen [12] proposed an intelligent PT framework named GAIL-PT. GAIL-PT collects expert knowledge via Metasploit and uses GAIL (generative adversarial imitation learning) in imitation learning for path planning. However, the complexity of the A3C-GAIL and DPPO-GAIL algorithms used in the experiments is still high. The above research takes full account of the characteristics of the penetration testing problem and uses imitative learning methods for intelligent penetration testing while incorporating expert knowledge. The use of expert knowledge with decision aids as inputs can go some way to improving the PT strategy of the agent, allowing it to be trained in a direction close to the behaviour of the expert [13]. However, the aim of imitation learning methods for training models is to fit the trajectory distribution of model-generated strategies to the trajectory distribution of the input. Therefore, using imitation learning in combination with RL makes it difficult to make policy enhancements to that part of the environment that has not been explored. In conclusion, the following challenges need to be stressed:

(i) Challenge 1: the penetration testing expert knowledge provided to the agent is poorly interpretable, usually dependent on specific network scenarios, and not universally applicable.

(ii) Challenge 2: the use of imitation learning is tending to produce an overfitting of expert knowledge, making it difficult to balance exploration and exploitation of the environment. The higher complexity of the algorithms used for intelligent penetration testing leads to slower convergence and lower efficiency.

To address these challenges, we first propose an intelligent PT framework called DQfD-AIPT that incorporates expert knowledge. DQfD-AIPT contains mainly the collection and exploitation of expert knowledge and the training of agents. At the same time, we have rationalised expert knowledge. Second, we use the DQfD algorithm based on the reinforcement learning with expert demonstrations (RLED) framework, combining both supervised and unsupervised methods to construct the loss function. The algorithm also uses prioritized experience replay (PER) for experience sampling to balance expert data and interaction data to prevent overfitting. Ultimately, experiments were conducted in a simulated network scenario containing honeypots to verify the effectiveness of expert knowledge incorporation and to test the performance of the DQfD algorithm. The experiments show that the DQfD algorithm has better penetration testing performance than the classical DRL method. For the experimental platform, we selected the CyberBattleSim (CBS) platform developed by Microsoft, which features high simulation and support for RL algorithms and is currently a more well accepted intelligent PT simulation platform.

The main contributions of this paper are as follows:

(i) To address the problems of poor interpretability of expert knowledge and dependence on specific scenarios, we propose an intelligent PT framework named DQfD-AIPT that incorporates expert knowledge. The construction of an expert knowledge base is carried out through two methods: the transformation of abstract expert knowledge and the collection of PT traces in different network scenarios. At the same time, we also define the form and structure of expert knowledge.

(ii) To address the problem of overfitting of expert behaviour due to imitation learning, we use the DQfD algorithm incorporating expert demonstration data, together with a PER mechanism for sampling of expert data and interaction data. With the guidance of the expert demonstration data and interaction data, the efficiency and overall performance of the training process of the agent are effectively improved.

The organization of this paper is as follows: In Section 2, we provide an overview of research advances in PT, intelligent PT, and use of expert knowledge. In Section 3, we explain and outline the RL covered in this paper. In Section 4, we describe the method that we used and the specific implementation details of the method. In Section 5, we describe the procedure and hyperparameter settings of the experiments, while the results are analysed and evaluated.

Finally, in Section 6, we summarise our work and further articulate future research directions.

2. Related Work

In this section, we mainly introduce the basic concepts of PT, the research progress of intelligent PT, and the importance of expert knowledge in the PT process. In the end, we conclude with a summary of the problems in the current study.

2.1. PT and Its Automation. PT is a method of simulating real attacks with the aim of assessing the security of computer systems and networks. PT assesses information security from the attacker's perspective. Through PT of companies, organisations, or departments, we understand their information security policies and network vulnerabilities and give possible solutions and remedies to improve network security [14]. As network equipment and defence detection systems continue to upgrade, the complexity of performing the PT process has increased dramatically. The entire testing process involves skilled cyber security experts generating attack plans to discover and exploit vulnerabilities in networks and applications. A team of highly experienced testers is therefore essential, who must control all tasks manually. In addition to this, there are a large number of repetitive actions and deterministic steps in the PT process, which will lead to the problem of the high time cost of conducting PT manually. In summary, PT currently appears to be a costly means of assessing the vulnerability of network systems [15, 16].

To address the high cost and reliance on manual PT, methods and tools to implement automated PT have been proposed. In terms of theoretical research, early studies such as attack trees, attack graphs, and PDDL are representative. These methods plan attack paths through formal representations of target network configuration information and state transfer analysis [3, 4]. On the one hand, these abovementioned methods require full knowledge of the network topology and the configuration of each machine, which is unrealistic from the attacker's point of view. On the other hand, these methods focus on a regular representation of known information and then find the path of attack by means of planning. The uncertainty of a real PT process is not well modeled. That is, the uncertainty of system knowledge must be obtained using remote tools before a planned attack can be executed.

In terms of the development of automated PT tools, mature automated PT tools include APT2 PT suite, Autosploit PT tool, and Awesome-Hacking tools [2]. These PT tools have significantly improved the efficiency of PT and simplified the process of conducting PT manually. However, there is still the problem of not being able to intelligently select the attack payload and of targeting only a single host. There is still a need to base the correct choice of attack methods and means on the decisions of PT experts. The use of intelligent planning techniques to improve the automation of attack path discovery is still the key to achieving automated PT [17].

2.2. Intelligent PT Using RL Methods. With the development of algorithms in the field of artificial intelligence, there have been new advances in the study of automated PT. Artificial intelligence-driven PT methods are able to intelligently select attack targets and attack payloads based on the current state of the target network. The RL approach learns how to map the current state to an action and provides an idea how to do so. The agent learns the PT strategy by interacting with the environment composed of the target network and based on the feedback from the interaction. The process by which we build a simulated environment for PT to train an agent is similar to how a player interacts with a game to discover its solution [9].

A precondition for applying RL for intelligent PT is the need to formalise the PT process into the RL paradigm. Sarraute et al. [18], Schwartz et al. [19], Hu et al. [20], and Zennaro [11] et al. formalised the penetration testing process as a partially observable Markov decision process (POMDP). They incorporated the attacker's observations of the environment into the attack process. However, as the size of the network scenario expands, the computational complexity increases and is still not applicable to large-scale networks. Durkota and Lisý [21] proposed the model penetration test as an MDP, in which the action space consists of specific vulnerabilities and the state space consists of the results of attack actions. The goal of the whole model is to minimize the expected loss value. Hoffmann [22], based on Durkota, ignores the structure of the target network and relies instead on expressing the uncertainty of PT in the form of possible action outcomes. This is essentially a model-free approach [9] that requires minimal prior knowledge of the environment. However, POMDP is more realistic in most cases. However, considering the computational complexity and the efficiency of the reinforcement learning algorithm, the MDP model is still a better scheme to balance the computational efficiency and modeling rationality.

In recent years, a variety of RL algorithms have been used extensively in addressing intelligent PT. Schwartz and Hanna [9] constructed the network attack simulator (Nasim) and used known network configurations as states, available scans and exploits as actions, and used table-based Q-learning methods and neural network-based DQN methods to achieve intelligent discovery of attack paths. Zhou et al. [10] combined various improvements with the DQN algorithm and proposed the NDSPI-DQN algorithm to optimise PT path discovery. The algorithm effectively reduces the action space of the agent and is experimentally validated based on Nasim. Zhang [2] introduced the multidomain action selection module on the basis of intelligent PT. This module can effectively identify the actions that can be used according to a specific state, reducing unnecessary exploration by an agent. Finally, this method combined with the deep deterministic policy gradient (DDPG) algorithm is verified in a simulated environment.

2.3. Use of Expert Knowledge. Expert guidance often plays a key role in solving real-world problems. As a method highly dependent on expert knowledge, reference to experienced expert knowledge is often helpful to exploit the vulnerability of the target system, so as to achieve the target at a lower cost. From the perspective of research status, the current research is mainly focused on solving the problems of state space explosion, action space explosion, and sparse reward caused by penetration testing using the reinforcement learning algorithm. Most of them focus on the algorithm itself, often ignoring the characteristics of expert decision-making in the penetration test process and the analysis of specific network scene structures.

Zennaro et al. [11] simplified the penetration testing problem with different structures in the form of a capture flag challenge and demonstrated how the performance of an agent can be improved by relying on different forms of prior knowledge provided to the agent. The experiments show that by incorporating prior knowledge, the agent can better explore the space of their problems and thus effectively obtain solutions. However, the CTF scenarios constructed for the experiments were only simplified versions and were not experimented on relatively complex scenarios. Chen [12] first proposed a generic intelligent PT framework based on GAIL. GAIL-PT addresses the problem of high labour costs due to the intervention of security experts and high-dimensional discrete action spaces. The study used a variety of algorithms for experiments, but the results showed that the complexity of the A3C-GAIL and DPPO-GAIL algorithms, which combine GAIL, is still relatively high.

The main idea of imitation learning is to match the behavioural strategies of an agent with the behaviour of an expert by means of training. Imitation learning can be divided into behavioural cloning [23], inverse reinforcement learning [24], and generative adversarial imitation learning [25]. However, imitation learning tends to focus on imitating the behaviour and trajectories of experts, making it difficult to enhance and contribute to the performance of the agent in the environment when combined with RL methods. This method is essentially an exploration and exploitation of the environment without enhancement for the strategy in the application of RL methods. Recently, demonstration data have been shown to help solve difficult exploration problems in RL. Subsequently, a framework known as reinforcement learning with expert demonstration (RLED) was proposed. This framework is suitable for scenarios where rewards are provided by the environment. Todd Hester et al.of the Google DeepMind team [26] propose the DQfD algorithm based on the RLED framework. The method is pretrained on presentation data while combining the features of supervised and unsupervised learning to construct a loss function and using PER in order to achieve a balanced amount of demonstration data in the training data. By training deep neural networks in this way, the results show that DQfD outperforms imitation learning, which only imitates expert trajectories, as well as classical deep Q networks in terms of average overall performance.

2.4. Brief Summary. We have summarised the current progress in intelligent penetration testing research in the previous section. Traditional penetration testing has high reliance on expert knowledge and high labour costs. In the process of changing from manual to intelligent PT, the cost of manual time is effectively reduced. This goes some way to solving one of the major dilemmas of current manual PT. However, as the complexity of the target system increases, the performance of penetration testing using classical RL methods also encounters bottlenecks. The problem is that real-world PT relies on expert experience and the proficiency of the penetration tester. Knowledge reasoning between successive states during real PT is missing in the training of the agent, but in the course of a real penetration test, this is quite important. Considering the integration of expert knowledge into intelligent PT can effectively solve these problems.

In the exploration of incorporating expert knowledge in PT, previous studies have mainly used imitative learning methods. These studies have been conducted by processing collected expert knowledge and combining it with imitative learning methods for PT path planning. On the one hand, the expert knowledge collected is usually highly relevant to the target network scenario and not universally applicable. The interpretability of this expert knowledge is relatively poor. On the other hand, the use of a combination of imitative and RL was effective in guiding the agent to fit the expert knowledge trajectory to some extent. However, imitative learning is more concerned with imitating the behaviour of the expert than the strategies of the testing expert. This will lead to problems of expert knowledge overfitting. As a result, agents trained using imitation learning often struggle to outperform experts. In conclusion, the current research in the field of intelligent penetration testing incorporating expert knowledge can be further enhanced with regard to the interpretability of expert knowledge and the RL methods used.

3. Preliminaries

3.1. Classical RL Method and Its Improvements

3.1.1. Q-Learning. Q-learning is a value-based RL algorithm. Q is $Q(s, a)$, which is the expected gain from taking an action $a (a \in A)$ in a state $s (s \in S)$ at a given time. The main idea of the algorithm is to construct a Q table of states and actions to store Q values and then select the action that yields the greatest benefit based on the Q value. Q-learning uses temporal difference (TD) to update Q values, with the updated formula shown in the following equation:

$$Q(s_t, a_t) \longleftarrow Q(s_t, a_t) + \alpha \left[r_t + \gamma \max_{a_{t+1}} (s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right], \tag{1}$$

where α is the learning rate, γ is the discount factor, a_t and s_t are the action and state at moment t, respectively, s_{t+1} is the next state after performing action a_t, a_{t+1} is the possible action to be performed in the state s_{t+1}, and r_t is the immediate reward obtained.

3.1.2. Deep Q-Learning. Q-learning takes a tabular approach to storing Q values. Therefore, when facing the RL assignment with a high-dimensional state space and action space, the limited space of the table cannot store all states and actions, which limits the performance of the algorithm [27]. Algorithms that combine the advantages of deep learning give a better solution to this problem. Minh [28] proposed the deep Q-network (DQN), which is an extension of Q-Learning. The algorithm replaces the Q value table in Q-learning with a neural network. This transforms the original problem of convergence of action value function solving into a function fitting problem for neural networks.

During the exploration and exploitation of the environment, the transition data are stored in a replay buffer. DQN uses randomly sampled data from the replay buffer to train the neural network to approximate the Q function. This method breaks the correlation between the training data and makes the training process stable. The DQN algorithm uses the square of the error between the target Q value and the estimated Q value as the loss function when updating the parameters of the neural network, where the target Q value y_i at the i iteration is calculated as follows:

$$y_i = E_{s_{i+1}\sim\varepsilon}\left[r_i + \gamma \max_{a_{i+1}} Q'\left(s_{i+1}, a_{i+1} \mid \theta_i'\right) | s_i, a_i\right], \tag{2}$$

where $\theta_i^{'}$ are the parameters of the target Q network. The goal of strategy learning is to update the parameters of the strategy Q network by the mean square error between the target Q value and the current Q value, where the loss function of the algorithm at the i iteration is calculated as follows:

$$L_i\left(\theta_i\right) = E_{s_i,a_i\sim\rho(\cdot)}\left[\left(y_i - Q\left(s_i, a_i \mid \theta_i\right)\right)^2\right], \tag{3}$$

where $\rho(s_i, a_i)$ is the probability distribution of s_i and a_i. θ_i are the parameters of the strategy network. The parameters of the strategy network are assigned to the target network in fixed steps of intervals. When optimizing the loss function, the parameters of the target network $\theta_i^{'}$ are not updated.

3.1.3. Prioritized Experience Replay. Prioritized experience replay is a technique for prioritizing experiences, whereby important state transition experiences are replayed more frequently. This method is effective because some transition data contain more information that is worth learning. Giving these transitions more opportunities to be played back helps accelerate the overall learning process. The core idea of PER is to measure the importance of different transition data through the TD error δ. The larger the error of the sample, the larger the value of the sample. The sampling probability of the state transition i is calculated as follows:

$$P(i) = \frac{p_i^{\alpha}}{\sum_k p_k^{\alpha}}, \tag{4}$$

where p_i refers to the priority of the state transition i, denoted as $p_i = |\delta_i| + \in$, and $\in$ is a positive number used for numerical stability so that $p_i \rangle 0$. ∂ is an exponential hyperparameter representing the degree of the impact of the TD error on the sample.

It is worth noting that the purpose of using experience replay is to eliminate sample correlation, but the use of prioritized sampling certainly forgoes random sampling. Therefore, it is also necessary to reduce the training weights of the high-priority state transition data. The PER method uses importance sampling weights to correct for deviations in the state transition i. The weights are calculated as follows:

$$\omega_i = [NP(i)]^{-\beta}, \tag{5}$$

where N refers to the capacity size of the replay buffer and β is the annealing hyperparameter of the training process. To implement the above method efficiently, we store the priorities in an efficient query line tree data structure and sample the range of line segments during the training process. We call this efficient query data structure a sum tree.

3.2. RL Formalisation for PT. An RL agent must be able to perceive the state of the environment, have one or more goals related to the state of the environment, and then take action and influence the state of the environment. In order to implement intelligent PT in conjunction with an RL method, we begin by modeling the penetration testing process as an RL paradigm. MDP is a theoretical framework for achieving goals through interactive learning. It is the classic formal expression of sequential decision-making. The actions of an agent in an MDP affect not only the current immediate reward but also the subsequent state and the future benefit. Thus, the MDP is a mathematically idealised form of the reinforcement learning problem. The interaction process between the agent and the environment in the MDP is shown in Figure 1.

The machines that learn and implement decisions in the MDP are called agents. Everything that interacts with it outside of the agent is referred to as the environment. Taking the penetration testing process as an example, if the target network is considered as a state variable environment, the feedback from the environment to the actions of the agent is considered as a reward. The whole penetration testing process can then be represented in the form of a 4-tuple$\langle S, A, R, T\rangle$, where S represents the state space, A represents the action space, R represents the reward function, and T represents the transfer function. Detailed definitions for specific penetration testing questions are given in Section 4.3.

4. Methods

In this section, we first present an intelligent PT framework incorporating expert knowledge and explain the details and processes involved in this framework. We then detail the collection and use of expert knowledge and present the RL algorithm that we use that incorporates demonstration data.

4.1. DQfD-AIPT Framework. Manual PT relies on the experience and knowledge of experts. Expert knowledge and the way of making decisions are also of great importance for intelligent PT. By analysing the PT process and summarising

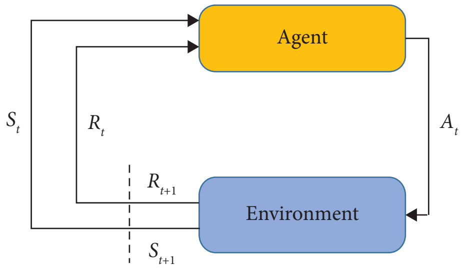


Figure 1: The agent interacts with the environment.

the characteristics of expert knowledge and RL, we propose DQfD-AIPT for intelligent PT that incorporates expert knowledge as shown in Figure 2. DQfD-AIPT consists of three main phases: expert knowledge collection, input and use of demonstration data, and interaction and training of the agent. Our proposed framework is suitable for intelligent PT in simulated network scenarios and is characterised by simplicity of use and generality. In the following, we will explain the specifics of each phase of the framework.

4.1.1. Stage 1: Collection of Expert Knowledge. The first step in combining expert knowledge for intelligent PT is the collection and acquisition of expert knowledge. The acquisition of expert knowledge is an abstract and difficult matter. The difficulty lies in the fact that the generation and design of expert knowledge are often based on the experience and rules of experts. As the process of PT is usually strongly correlated with the structure and vulnerability distribution of the target network scenario, the actions taken by penetration testers facing a specific target network are somewhat unpredictable. Considering the interpretability and validity of expert knowledge, we propose two ways of collecting expert knowledge:

(i) Method 1: As shown in Figure 3, first, we transform the abstract experience of the penetration tester into an executable action that can interact with the simulated environment. This executable action is usually mapped to a specific environmental state. For example, a penetration tester decides to take a certain penetration exploit action based on the current state of the target network. We can abstract this expert knowledge and represent it in the form of a state-action pair. Afterwards, the penetration test simulation environment executes the action and processes the result of the action and gives feedback. Finally, the reward value R resulting from the execution of action A is integrated into the state S together with the new state S'. We obtain a complete set of expert transition data that can be used by the training of the agent and stored in an expert knowledge base.

(ii) Method 2: We collect valid paths and traces of agents completing PT objectives in multiple different simulated network scenarios (scenarios that are within a fixed order of magnitude due to the uniformity of expert knowledge). These trajectories consist of multiple transition data, and the transition data are obtained from tests of the agent against different network scenarios. However, due to the structural specificity of the expert data that we designed, real-world common open ports and services, for example, have specific bitmasks in the transition data. These predefined bitmasks often override the configuration of our simulated network environment. For example, the agent collects expert data in multiple network scenarios of size less than N. We can apply this expert knowledge to the training of network scenarios of a size less than N. In addition to this, we can take valid transition data (reward values > 0 for action execution) and store them in an expert knowledge database.

4.1.2. Stage 2: Input of Expert Knowledge. The transitional data collected in the expert knowledge base can be used as demonstration data to ensure that agents can learn the PT strategy of experts through pretraining. The demonstration data input to the agent conform to the standard transition data form of reinforcement learning algorithm and are used for training and processing. The detailed representation and structure of transition data are shown in Figure 3.

The transition data consist of a four-tuple: $\tau(S, A, R, S')$, where S is a valid observation of the current target network state by the agent and also serves as the state input to the neural network in the RL algorithm, A is the vector of actions performed, consisting of the number of the agents' actions, R is the reward value for executing action A under the state S, and S' is the new state to which the transition is made after the execution of action A under the state S. Each transition in the expert knowledge base has the same structure and can be applied to the training of the agent as demonstration data. The observed state contains the agent's perception of the target network environment, which contains statistically tractable information that has an impact on the action decisions of the PT process. This information includes, among other things, the number of hosts the agent has discovered, the number of hosts it controls, the number of open services it has scanned, the number of connection credentials it has obtained, and the ports it has discovered.

4.1.3. Stage 3: Interaction and Training of the Agent. As the agent interacts with the penetration test simulation environment, the agent's actions will change the state of the environment, while the environment will give feedback to the agent on rewards and penalties. The agent adjusts the PT strategy and actions based on the rewards. At this point, the demonstration data extracted from the expert knowledge base serve to assist the agent in making decisions and to influence the agent's tendency to perform actions. The better transition data generated by the agent as it interacts with the environment are also recorded and stored in the expert knowledge base. In this way, the expert knowledge base is continuously expanded with valid transition data. More

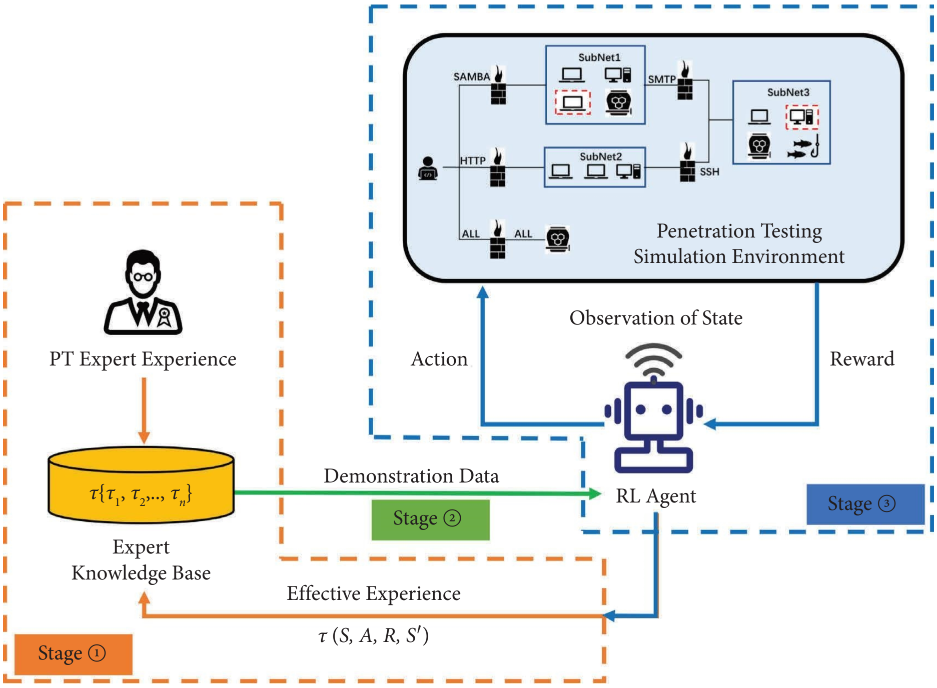


Figure 2: Framework structure of DQfD-AIPT.

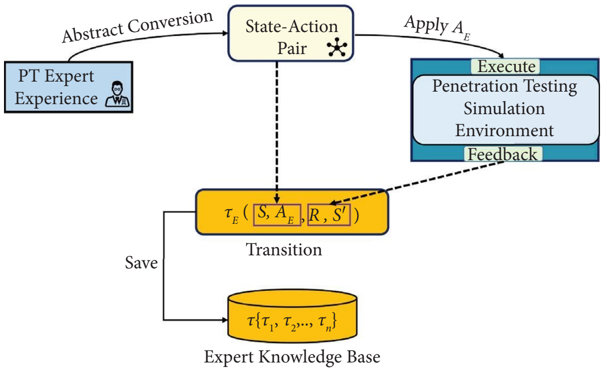


Figure 3: The process of converting abstract expert knowledge into transition data.

comprehensive and effective guidance is provided for the training of the agent.

4.2. DQfD Training. The previous section described the basic process of implementing our DQfD-AIPT framework incorporating expert knowledge. In particular, we highlighted the process of collecting expert knowledge data. This section will describe the DQfD RL algorithm and the specific implementation details of how the algorithm combined with expert knowledge data can guide the agent in PT.

Based on the algorithm structure of DQN and combined with the framework of DQfD, we implement the DQfD algorithm, whose algorithm structure can be expressed as shown in Figure 4. First, based on the algorithmic structure of the DQN, we build a policy network and a target network with the same structure. Each network consists of a multilayer structured neural network: an input layer, three fully connected hidden layers, and an output layer. At the same time, we implement PER through the tree storage structure of the sum tree. PER draws those samples that are more valuable at higher frequency than random samples, and PER takes into account the importance of the different state transition data by means of the TD error. This approach is used to balance the proportion of demonstration data and interaction data contained in the small batch of transition data sampled. On this basis, we combine the demonstration data from the expert knowledge base to improve algorithm performance and learning efficiency.

The detailed process of the DQfD algorithm can be described as follows.

4.2.1. Pretraining Stage. State transition data from the constructed expert knowledge base are prepositioned in the demonstration data area of the sum tree. In particular, it is important to emphasise that the size of the demonstration data area and the data filled by the sum tree are fixed. Throughout the training process, the data in the demonstration area are not overwritten as new state transition data are added to the sum tree. After the expert knowledge preset was completed, the policy network was pretrained by sampling batch-sized state transition data from the demonstration data areas several times. The pretraining process updates the parameters of the Q network using a $J(Q)$ loss combining the three losses, where the $J(Q)$ error is calculated as follows:

$$J(Q)_s = J_Q(Q) + \lambda_1 J_n(Q) + \lambda_2 J_E(Q) + \lambda_3 J_{L2}(Q), \tag{6}$$

where $J(Q)_s$ is the joint loss containing the supervised loss, λ_1, λ_2, and λ_3 represent the constants, respectively, $J_Q(Q)$ is the loss of DQN, $J_n(Q)$ is the N-step return loss, $J_E(Q)$ is

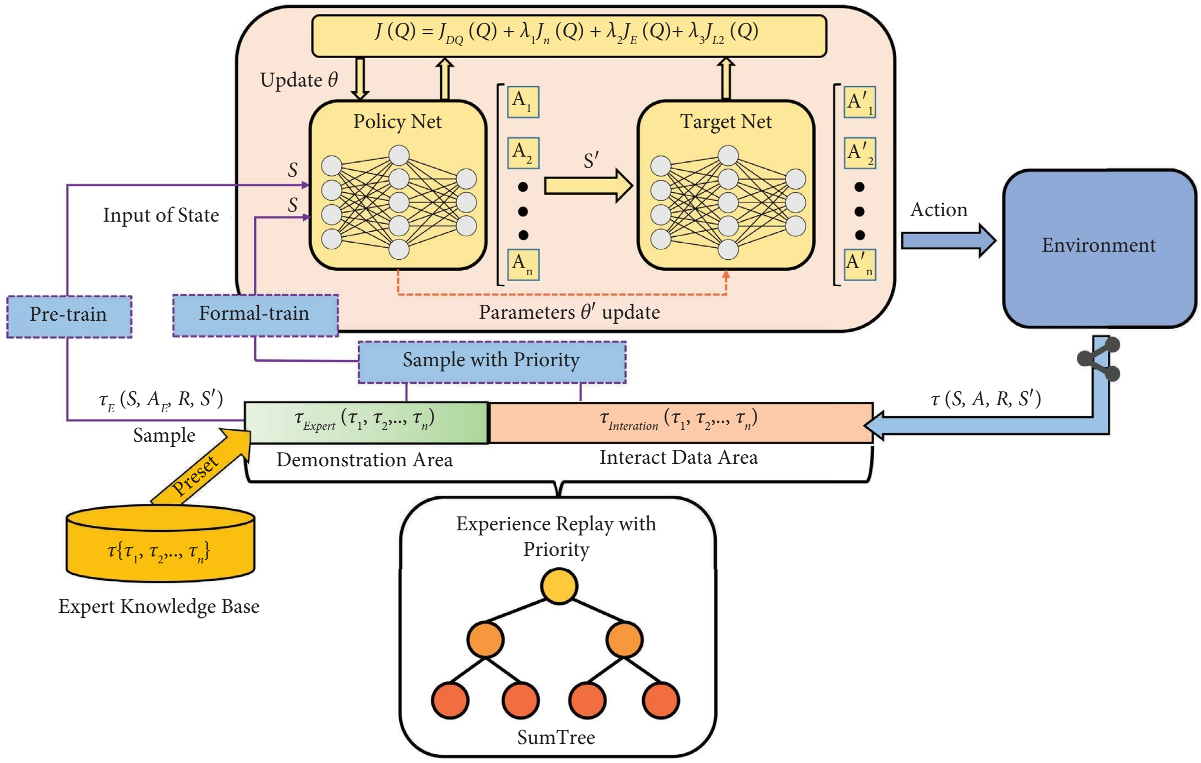


Figure 4: DQfD algorithm training process.

a supervised loss, and $J_{L2}(Q)$ is a regularisation term applied to the neural network to alleviate overfitting to the presentation data and to prevent the strategy from overfitting to a small fraction of the experience in the expert data. A detailed explanation of the $J_n(Q)$ and $J_E(Q)$ is given below.

For the $J_n(Q)$ loss, the agent updates its Q-network with a mixture target of 1-step and n-step return. The incorporation of the N-step loss can help propagate the value of the expert data to an earlier state, greatly enhancing learning from the limited demo dataset. It also ensures that the pretrained learned neural network value function estimates satisfy the Bellman equation. The calculation of $J_n(Q)$ can be expressed as follows:

$$r_t + \gamma r_{t+1} + \ldots + \gamma^{n-1} r_{t+n-1} + \max_{\alpha} \gamma^n Q(s_{t+n}, a). \tag{7}$$

$J_E(Q)$ is a supervised loss. The incorporation of the supervised loss is the key to the pretraining process. It can be expressed as follows:

$$J_E(Q) = \max_{a \in A}[Q(s,a) + l(a_E, a)] - Q(s, a_E), \tag{8}$$

where a_E is the action corresponding to the expert demonstration data in the state S. $l(a_E, a)$ is a margin function, which is a measure of how well the currently executed action matches the action demonstrated by the expert and can be expressed as follows:

$$l(a_E, a) = \begin{cases} 0, & a_E = a, \\ k, & a_E \neq a. \end{cases} \tag{9}$$

In this way, the value of any action that differs from the expert action a_E is less than the value of the action a_E. With the supervised loss, the values of actions outside the range of the demonstration data also become reasonable values, resulting in a value-driven ε-greedy strategy that effectively imitates the expert's actions. Pretraining is a good starting point for learning the task. Once the agent begins to interact with the task, it continues to learn by sampling from its own generated and demonstrated data.

4.2.2. Training Stage. After the pretraining phase, we get an agent with expert experience. However, the agent does not interact with the environment throughout the pretraining phase. During the formal training phase, the agent first interacts with the environment to generate transition data, and each transition data is stored in the interaction data area of the sum tree structure. In addition, to avoid overfitting of the expert transition data early in the training process, the interaction data area of the sum tree needs to be filled up by the interaction of the agent with the environment before the formal learning begins. After the maximum storage capacity is reached, the agent-generated data will continuously cover the interaction data area of the sum tree structure. The flow of the PER algorithm is presented in Algorithm 1.

In the pretraining phase, only the expert transition data are extracted from the demonstration data areas for training. In the formal training phase, the transition data from both the demonstration data region and the interaction data area could be extracted from the sum tree according to the PER method. The difference is that the supervised loss $J_E(Q)$ is

```
Input: $k$: batch size, $N$: capacity of the sum tree, and $n$: the amount of transition currently stored by the sum tree, initialised the sum tree structure
Output: Updated the sum tree structure after sampling the transitions
if $n < N$ then
    push the transition $\tau(S, A, R, S')$ into the sum tree with maximal priority
end if
Start sampling batch-size transitions from the sum tree
Calculate $\sigma \leftarrow$SumTreeMaxPriority/$k$
for steps $t \in \{1, 2, \ldots k\}$ do
    Sample $k$ transition data with priority from the sum tree
    $a \leftarrow t^* \sigma$, $b \leftarrow (t+1)^* \sigma$,
    $v \leftarrow$ generate a random number between $a$ and $b$
    Transition $\tau_t$ and corresponding priority are obtained according to a random number $v$
    Compute importance sampling weight for each transition $\tau_t$
end for
Train this batch size of transition and compute the TD error according to the weight
Update transition priority according to the TD error
```

Algorithm 1: Implements PER with the sum tree structure.

removed from the calculation of the joint loss $J(Q)_s$, which indicates that $\lambda_2 = 0$.

In addition to this, the network update process is more computationally expensive due to the forward propagation process compared to the forward propagation process. The purpose of this form is to ensure that the replay buffer is closer to the state distribution of the current policy and to prevent network overfitting. Therefore, the update frequency of the target network in the pretraining phase is measured in steps, and the step setting interval should not be too small. In the formal training stage, the update frequency of the target network is in episodes. In summary, the DQfD algorithm combined with expert knowledge is presented in Algorithm 2.

4.3. RL Settings for PT. In this paper, we model the PT process as an MDP. We use the RL agent as an attacker who penetrates the target system. The target system constitutes the environment in which the agent interacts with each other. We take the formalisation of MDP as a basis and consider the characteristics of the intelligent PT simulation environment used for the experiments. We give the relevant settings for the necessary elements required for RL. The relevant elements in the PT formalisation into an MDP can be represented separately as follows:

(i) State space: A state space is a finite set of states with a nonfixed structure. In PT problems, the state space covers the range of changeable states of the target network. In this paper, we take the awareness of the target network environment by an agent through observation as the state. The representation of the specific states is shown in Figure 5.

(ii) Action space: The action space contains all the executable actions of the agent and does not change with the current state of the environment. This means that the output dimension of the neural network is always fixed during the state-to-action mapping process. In this paper, there are three main types of actions for an agent:

① Local exploit: the local exploit action is the process of exploiting the local resources of a target host after taking control of that host. The outcome of this action exploitation is privilege elevation, credential information leakage, suspicious link leakage, etc.

② Remote exploit: the remote exploit uses the current controlled host as a springboard to execute malicious commands by submitting them in the local browser. The outcome of the exploit is to gain control of the target host, leaking a suspicious link, etc.

③ Connect: the connect action acts on the discovered hosts and connects to the target host by means of the host credential information acquired during the lateral movement. The action outcome is to control the target host.

(iii) Reward: the reward function is feedback from the environment for the action taken by the agent, and the calculation of the reward during the penetration test can be expressed as follows:

$$R = \mathrm{Eval}\left(\mathrm{Outcome}_A^S\right) - \mathrm{Cost}(\mathrm{Action}). \tag{10}$$

In the equation, $\mathrm{Eval}(\mathrm{Outcome}_A^S)$ is an evaluation of the outcome of the execution of an action by the agent. $\mathrm{Cost}(\mathrm{Action})$ is the cost of performing the action. The classification of the exploit outcomes of the actions and the corresponding values are shown in Table 1.

(iv) Transfer function: The transfer function is a description of the probability of the environment to make a state transfer under certain conditions. For the model-free approach, learning is performed from the experience generated during the interaction, as the agent cannot directly rate the merit

```
Input: $D^{\text{replay}}$: the experience replay area built by the sum tree, $D^{\text{demo}}$: expert demonstration data area in $D^{\text{replay}}$, $D^{\text{interact}}$: interactive data area in $D^{\text{replay}}$, $\theta$: weights for the policy network (randomly generated), $\theta'$: weights for the target network (randomly generated), $f_p$: update target network frequency of pretraining, $f_f$: update target network frequency of formal training, $k$: batch size, $j$: number of pretraining gradient updates, $E$: episode number of training, and $S$: max steps per episode
Output: An agent trained with expert knowledge
Push expert transition data into $D^{\text{demo}}$ and initialize their priority
for steps $t \in \{1, 2, \ldots j\}$ do
  Sample a batch size of $k$ transitions from $D^{\text{demo}}$ with prioritization
  Calculate loss $J(Q)_s$ using the target network
  Perform a gradient descent step to update the weights for the policy network $\theta$
  if $t \bmod f_p = 0$ then $\theta' \leftarrow \theta$ end if
end for
for episode $u \in \{1, 2, \ldots E\}$ do
  for step $v \in \{1, 2, \ldots S\}$ do
    Sample action $A$ from the behaviour policy
    The environment performs $A$ and gives back $R$(reward), and the agent observes $(S, A, R, S')$
    Push the transition $\tau(S, A, R, S')$ into $D^{\text{interact}}$, overwriting oldest interaction transition if over capacity of $D^{\text{interact}}$
    Sample a batch size of $k$ transitions from $D^{\text{replay}}$ with prioritization
    Calculate loss $J(Q)$ using the target network
    Perform a gradient descent step to update the weights for the policy network $\theta$
    $S' \leftarrow S$, the state transitions from $S$ to $S'$
  end for
  if $u \bmod f_f = 0$ then $\theta' \leftarrow \theta$ end if
end for
```

ALGORITHM 2: DQfD.

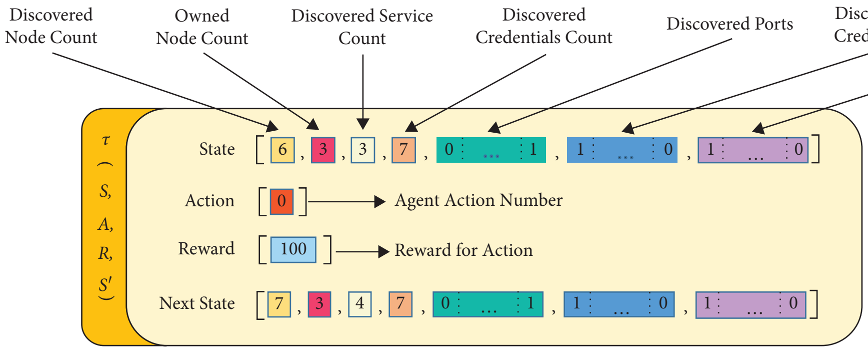


FIGURE 5: Structure of transition data.

TABLE 1: Action execution outcome and evaluation.

Outcomes	Evaluation
Leak new nodes	10
Leak credentials	5
Successful connection	20
Privilege promotion	10
Failed to exploit	−10
Repetitive action	−5

of the transformed state. The transfer function is unknown when formalising the PT process as an MDP.

5. Experiment

First, we build PT simulation network scenarios on the CBS platform developed by Microsoft. Second, we build an expert knowledge base containing transition data for multiple network scenarios by using the expert knowledge collection method introduced in Section 3. Finally, to validate the effectiveness of our proposed method, we use the DQfD algorithm and the DQN algorithm to perform PT path planning under scenarios containing elements of network defence deception (equipped with honeypots), respectively, and the performance of the algorithms is evaluated by specific metrics.

5.1. Main Experimental Procedures

5.1.1. Experimental Platform. Our experiments were conducted on the CyberBattleSim (CBS) platform developed by the Microsoft security team. CBS is an experimentation research platform to investigate the interaction of automated agents operating in a simulated abstract enterprise network environment. The simulation provides high-level abstraction of computer networks and cyber security concepts. We construct a simulated network scene through CBS and encapsulate this network scene into a gym environment where we can interact with an agent and further combine it with RL-related algorithms for our experiments.

5.1.2. Network Scenario Building. We abstracted a real-world enterprise network scenario and built a simulated network scenario containing a honeypot based on the CBS platform. The topology of the network scenario is shown in Figure 6.

The simulated enterprise network scenario consists of a DMZ zone, Trust-1 zone, and Trust-2 zone. The Trust-1 and Trust-2 zones provide web services and database services in the enterprise network. We have deployed honeypots in two separate trust areas. Honeypots are replicas of sensitive servers and hosts and provide some common services, open more sensitive ports, and contain some invalid resources and information. Honeypots are set up to consume the attacker's resources and to mitigate the impact of the attacker's actions on the enterprise network. This is a high fidelity construction of a real-world network scenario. The firewall between each zone controls the access policy between zones. The host configuration information and firewall access policies and vulnerability information for the simulated enterprise network scenario are shown in Tables 2–4, respectively.

5.1.3. Penetration Testing Goals. In the simulated enterprise network scenario, the attacker has initially gained control of SpringBoot in the DMZ zone. The attacker uses this as a springboard machine for further lateral move. The goal of the PT process is to gain access to the control commands of the database server in the Trust-2 area in order to get further access to sensitive data and critical information. At the same time, the agent needs to avoid getting caught in the honeypot in the trust area. That is, the agent expects to obtain as many cumulative rewards as possible at the least cost of consumption.

5.1.4. Expert Knowledge Collection. For the construction of the expert knowledge base, we convert the artificial experience of successfully conducting PT into demonstration transition data that can be understood and learnt by the agent. In our experiments, we collected 1000 expert demonstration data from each of 10 different structured network scenarios and pushed them into the expert knowledge base. The scale of these network scenarios is all within a certain size range. We preplace these expert demonstration data in the demonstration data areas of the sum tree before pretraining.

5.1.5. Evaluation Metrics

(i) Average cumulative reward: In the process of applying the RL algorithm to train an agent for penetration testing, the cumulative reward earned in each round can directly indicate the training of the current round as the number of steps increases. Therefore, the average cumulative reward over multiple rounds can effectively show the average performance of the agent throughout the whole training process.

(ii) Probability of attacking honeypots: The honeypots in the simulated network scenario are hosts or servers with a cyber deception defence role. We calculate the number of times the honeypot is attacked in each round as a percentage of the number of actions performed by the agent. This metric assesses the effectiveness of expert demonstration data for policy training by the agent.

5.2. Experimental Results and Analysis. We trained the agent to perform PT using two algorithms, DQfD and DQN, respectively. The hyperparameter settings for the algorithms and the DQfD-specific parameter settings are shown in Tables 5 and 6, respectively.

The average of the cumulative rewards obtained by the agent over the 200 round episodes of training was counted to compare the performance of DQfD and DQN. Second, to verify the effectiveness of the incorporated expert knowledge, we counted the probability of attacking the honeypot for Honeypot-1 and Honeypot-2, respectively. To ensure the credibility of the experimental results, we conducted 10 experiments in the same network scenario. We plotted the average results of the 10 experiments as graphs as shown in Figures 7–9.

As can be seen from the experimental results in Figure 7, the DQfD algorithm incorporating expert knowledge is able to achieve the PT goal in fewer steps compared to the DQN algorithm (DQfD within 500 steps and DQN within 3000 steps). Redundant repetitions, exploitation failures, and actions that fall into the honeypot during the penetration test often result in penalties. Therefore, the larger the reward value accumulated in each round, the more it reflects the superiority of the agent's PT path and action selection strategy. The DQfD algorithm accomplishes the goal faster while earning more cumulative rewards in each round. The experimental results indicate that the DQfd algorithm improves the performance of penetration testing to a certain extent while demonstrating the superiority of fusing expert knowledge.

The results in Figures 8 and 9 show that intelligence trained using the DQfD algorithm has a significantly lower probability of attacking the honeypot hosts in the Trust-1 region and Trust-2 region in each episode. Compared to DQN, DQfD maintains a lower probability of attack throughout the training convergence, always below 0.1. The reason for fluctuations in DQfD in the early stages is due to the fact that in the early stages, the exploration rate ϵ is in the process of decaying. However, there is still a high probability of random exploration of actions. DQN has a high probability of attacking the honeypot in the early stages. As training progresses, trial-and-error experience is learned into the network model though. However, due to the lack of guidance from expert knowledge, its ability to avoid

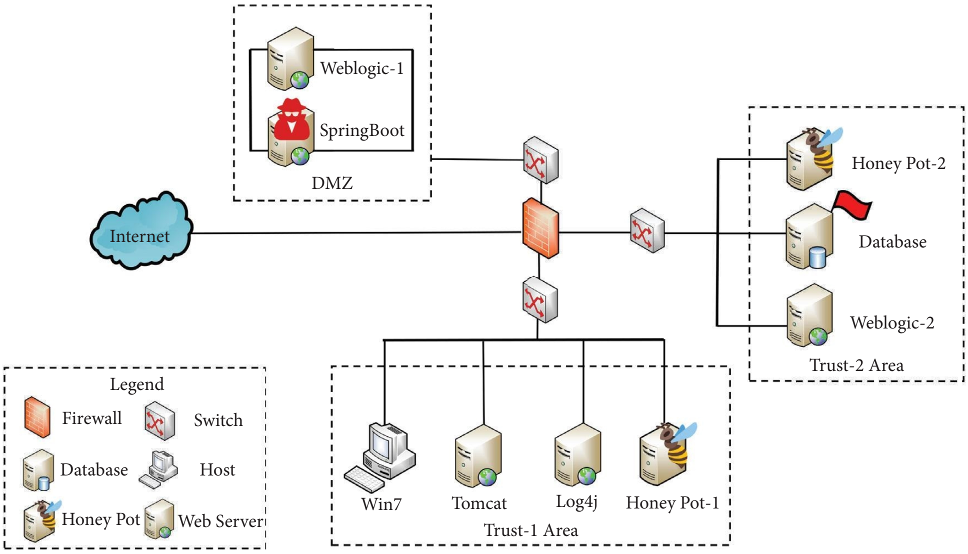


FIGURE 6: Enterprise network scenarios with honeypots.

TABLE 2: Host configuration information of the enterprise network.

Host ID	Vulnerability	Service	Open port	Value
Weblogic-1	CVE-2015-4852	HTTP	7001,5556	10
SpringBoot	Scancachehistory	HTTPS	8082	50
Windows7	CVE-2021-42287, scancachehistory	RDP, HTTPS, SMTP, FTP	8080, 3389, 25, 21	100
Tomcat	CVE-2017-12615	HTTP, SSH	8088, 22	100
Log4j	CVE-2021-44228, scancredhistory	HTTPS, RDP	3389, 8080	100
Honeypot-1	CVE-2016-10009	HTTPS, MySQL, FTP, SMTP, SSH	8080, 3306, 21, 25, 22, 2222	−200
Honeypot-2	CVE-2016-10009	HTTPS, MySQL, FTP, SMTP, SSH	8080, 3306, 21, 25, 22, 2222	−200
Database	CVE-2017-4971	HTTPS, MySQL, FTP	3306, 8080, 21	200
Weblogic-2	CVE-2015-4852	HTTP	7001, 5556	100

TABLE 3: Firewall policies for the enterprise network.

Sources	Destination	Service	Rule
SpringBoot	Windows7	HTTPS, RDP	Permit
SpringBoot	Tomcat	HTTP, SSH	Permit
SpringBoot	Log4j	HTTPS, RDP	Permit
SpringBoot	Honeypot-1		All permit
SpringBoot	Trust-2		All deny
Trust-1	Trust-2	HTTPS, MySQL, FTP, HTTP, SSH	Permit
Trust-1	DMZ		All permit
Trust-2	Windows7	HTTPS, RDP	Permit
Trust-2	Tomcat	HTTP, SSH	Permit
Trust-2	Log4j	HTTPS, RDP	Permit
Trust-2	Honeypot-1		All permit

deception defences was not significantly improved compared to DQfD.

The experimental results effectively reflect the role of incorporating expert knowledge in the identification and evasion of the deception defence components of the scenario during the PT performed by the agent. The less the intelligence interacts with the honeypot in each episode, the less the cost of completing PT will be consumed. This will greatly weaken the role of honeypot deployments from another perspective, where expert knowledge can guide and modify the agent's PT strategy.

Table 4: Vulnerability information and exploitation results.

Vulnerability	Type	Outcome	Cost
CVE-2015-4852	Remote exploit	Privilege promotion	10
CVE-2016-10009	Remote exploit	Privilege promotion	10
CVE-2017-12615	Remote exploit	Privilege promotion	10
CVE-2021-44228	Remote exploit	Privilege promotion	10
CVE-2021-42287	Remote exploit	Privilege promotion	10
Scancachehistory	Local exploit	Live nodes leaked	5
Scancredhistory	Local exploit	Credential leaked	5

Table 5: Hyperparameter setting of the algorithm.

Hyperparameter	DQfD	DQN
Batch size	512	512
Epsilon	0.9	0.9
Discount factor	0.015	0.015
Epsilon exponential decay	5000	5000
Epsilon minimum	0.1	0.1
Learning rate	0.01	0.01
Demonstration memory size	1000	*
Replay memory size	5000	5000
Target network update frequency	6	6
Max steps per episode	3000	3000
Training episode	200	200

Table 6: Special hyperparameter setting of the DQfD.

Hyperparameter	Value
Pretrain step	1000
N-step return weight λ_1	1.0
Supervised loss weight λ_2	1.0
L_2 regularisation weight λ_3	1.0
Expert margin $l(a_E, a)(a \neq a_E)$	0.8
N of N-step return	10
Prioritized replay exponent α	0.4
Prioritized replay constants ϵ_a	0.001
Prioritized replay constants ϵ_d	1.0
Prioritized replay importance sampling exponent β_0	0.6

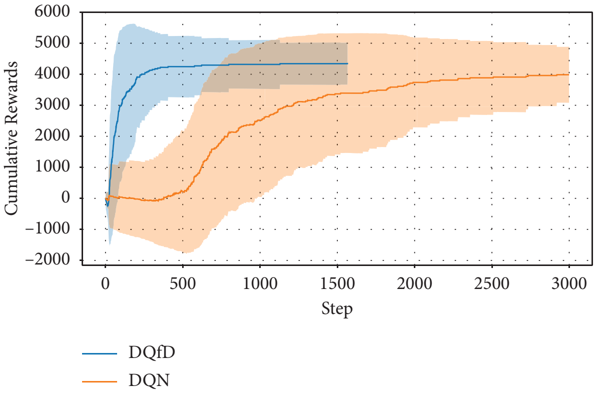


Figure 7: Changes in the average cumulative reward value.

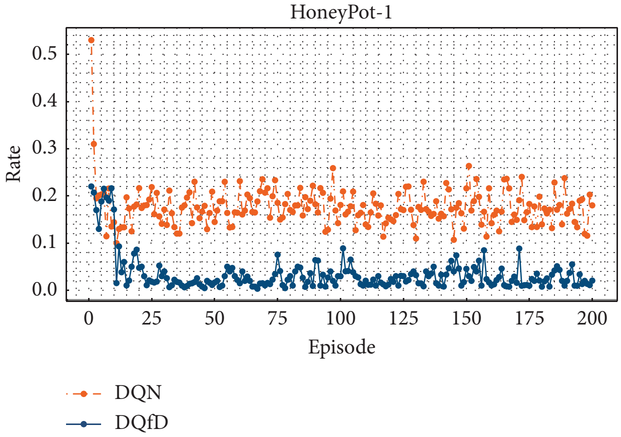


Figure 8: Probability comparison of attacking Honeypot-1.

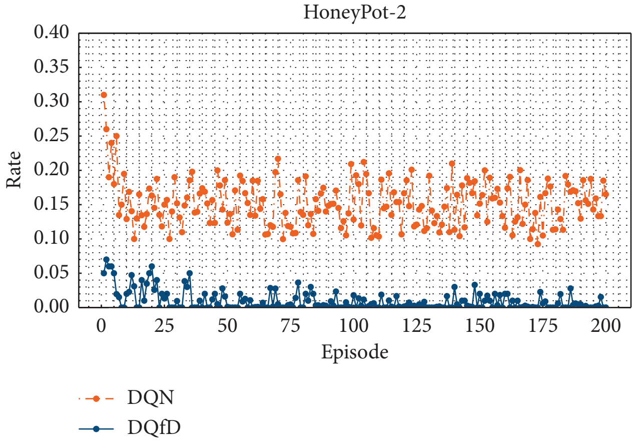


Figure 9: Probability comparison of attacking Honeypot-2.

6. Conclusion and Future Work

In this study, we focus on the effective use of expert knowledge in the process of intelligent PT. In order to address the poor interpretability of expert knowledge and the overfitting of algorithms to expert knowledge that exists in the current study, we propose an intelligent PT framework named DQfD-AIPT. DQfD-AIPT incorporates the collection of expert knowledge and guides the agent in the enhancement of PT strategies. The expert knowledge that we use is interpretable and generalisable for PT processes in network scenarios at a fixed scale. At the same time, we provide a detailed description of the process of transforming and collecting expert knowledge and define the structure of expert knowledge. Our proposed DqfD-AIP framework is generic and feasible. In terms of algorithms for combining expert knowledge, we use the DQfD method based on the RLED framework instead of traditional imitation learning.

The advantage of using the DQfD algorithm is that it combines the advantages of supervised and unsupervised learning. The DQfD algorithm makes reasonable use of the transition data from experience replay and avoids the phenomenon of overfitting of expert data. The results of experiments conducted in a network scenario with honeypots also indirectly indicate the validity of the expert demonstration data. Experimental results also show that the DQfD algorithm not only achieves higher cumulative reward values faster in network scenarios with honeypots but also makes better use of the expert demonstration data to avoid getting trapped in honeypots compared to DQN.

The efficiency and difficulty of PT depends on the complexity of the target network structure. Most of the expert knowledge that we currently collect is gathered through training in network scenarios of a specified size range. The coverage of expert knowledge is therefore relatively small and limited by the representational form of the transition data. In future research, we consider improving the interpretability of expert knowledge by better converting human PT experience into knowledge that can be accepted and learned by an agent. [15, 27, 28].

Data Availability

As a result, the data for this experiment were generated through training on the CBS platform, and no previous data were used. The transition data used in this article are generated through self-constructed scenarios.

Additional Points

Our experiment was carried out on the basis of Microsoft's open source CBS platform (CyberBattleSim). CBS highly abstracts the process of penetration testing from the network scene in the real world and builds simulated network scenarios. The agent algorithm interface is provided in CBS, and the agent uses the interaction with the environment and the reinforcement learning algorithm to obtain the penetration test strategy. The reinforcement learning method is an unsupervised machine learning method and difficult to repeat, and the data for simultaneous training are generated in the process of interaction between the agent and the concrete environment. The difficulty of penetration testing depends on the complexity of network scenarios. In reality, network scenarios are diverse, and the generation of transition data is also related to the structure of network scenarios.

Conflicts of Interest

There are no conflicts of interest regarding the publication of this paper.

References

[1] A. Chowdhary, D. Huang, J. S. Mahendran, D. Romo, Y. Deng, and A. Sabur, "Autonomous security analysis and penetration testing," in *Proceedings of the 2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, IEEE, Tokyo, Japan, December 2020.

[2] Y. Zhang, "Domain-independent intelligent planning technology and its application to automated penetration testing oriented attack path discovery," *Electron. Inf. Technol*, vol. 42, pp. 2095–2107, 2020.

[3] M. A. Abd-Elmagid, N. Pappas, H. S. Dhillon, and H. S. Dhillon, "On the role of age of information in the Internet of Things," *IEEE Communications Magazine*, vol. 57, pp. 72–77, 2019.

[4] Z. Fang, J. Wang, Y. Ren, Z. Han, H. V. Poor, and L. Hanzo, "Age of information in energy harvesting aided massive multiple access networks," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 5, pp. 1441–1456, 2022.

[5] F. Baiardi, "Avoiding the weaknesses of a penetration test," *Computer Fraud & Security*, vol. 2019, Article ID 10.1016/s1361-3723(19)30041-7, pp. 11–15, 2019.

[6] F. Holik, "Effective penetration testing with Metasploit framework and methodologies," in *Proceedings of the 2014 IEEE 15th International Symposium on Computational Intelligence and Informatics (CINTI)*, IEEE, budapest, Hungary, November 2014.

[7] N. Polatidis, E. Pimenidis, M. Pavlidis, S. Papastergiou, and H. Mouratidis, "From product recommendation to cyber-attack prediction: generating attack graphs and predicting future attacks," *Evolving Systems*, vol. 11, pp. 479–490, 2020.

[8] M. Sultana, A. Taylor, and L. Li, "Autonomous network cyber offence strategy through deep reinforcement learning," *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications III*, SPIE, vol. 11746, , 2021.

[9] J. Schwartz and K. Hanna, "Autonomous penetration testing using reinforcement learning," 2019, https://arxiv.org/abs/1905.05965.

[10] S. Zhou, J. Liu, D. Hou, X. Zhong, and Y. Zhang, "Autonomous penetration testing based on improved deep q-network," *Applied Sciences*, vol. 118823 pages, 2021.

[11] Zennaro, F. Massimo, and L. Erdodi, "Modeling penetration testing with reinforcement learning using capture-the-flag challenges: trade-offs between model-free learning and a priori knowledge," 2020, https://arxiv.org/abs/2005.14165, Article ID 12632.

[12] J. Chen, "GAIL-PT: a generic intelligent penetration testing framework with generative adversarial imitation learning," 2022, https://arxiv.org/.

[13] J. A. Bland, M. D. Petty, T. S. Whitaker, K. P. Maxwell, and W. A. Cantrell, "Machine learning cyberattack and defense strategies," *Computers & Security*, vol. 92, Article ID 101738, 2020.

[14] K. Qian, "Ontology and reinforcement learning based intelligent agent automatic penetration test," in *Proceedings of the IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, IEEE, Dalian, China, June 2021.

[15] D. Stiawan, "Cyber-attack penetration test and vulnerability analysis," *International Journal of Online and Biomedical Engineering*, vol. 13, 2017.

[16] H. M. Z. A. Shebli and B. D. Beheshti, "A study on penetration testing process and tools," in *Proceedings of the 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT)*, pp. 1–7, May 2018.

[17] M. C. Ghanem and T. M. Chen, "Reinforcement learning for efficient network penetration testing," *Information*, vol. 11, no. 1, 6 pages, 2019.

[18] C. Sarraute, O. Buffet, and J. Hoffmann, "Penetration testing==POMDP solving?," 2013, https://arxiv.org/abs/1306.4714.

[19] J. Schwartz, H. Kurniawati, and E. El-Mahassni, "Pomdp+ information-decay: incorporating defender's behaviour in autonomous penetration testing," in *Proceedings of the International Conference on Automated Planning and Scheduling*, pp. 235–243, Singapore, June 2020.

[20] Z. Hu, R. Beuran, and Y. Tan, "Automated penetration testing using deep reinforcement learning," in *Proceedings of the 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, September 2020.

[21] K. Durkota and V. Lisý, *Computing Optimal Policies for Attack Graphs with Action Failures and Costs*, STAIRS, Geneva,Switzerland, 2014.

[22] J. Hoffmann, "Simulated penetration testing: from "dijkstra" to "turing test++," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 25, pp. 364–372, Jerusalem Israel, June 2015.

[23] A. Kanervisto, J. Pussinen, and V. Hautamäki, "Benchmarking end-to-end behavioural cloning on video games," in *Proceedings of the 2020 IEEE conference on games (CoG)*, IEEE, Osaka, Japan, August 2020.

[24] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: challenges, methods and progress," *Artificial Intelligence*, vol. 297, Article ID 103500, 2021.

[25] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: a survey of learning methods," *ACM Computing Surveys*, vol. 50, pp. 1–35, 2018.

[26] T. Hester, M. Vecerik, O. Pietquin et al., "Deep q-learning from demonstrations," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, New Orleans LA USA, Febraury 2018.

[27] Y. Fenjiro and H. Benbrahim, "Deep reinforcement learning overview of the state of the art," *Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 12, no. 3, pp. 20–39, 2018.

[28] V. Mnih, *Playing Atari with Deep Reinforcement Learning*, https://arxiv.org/abs/1312.5602, 2013.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 7611741, 10 pages
https://doi.org/10.1155/2022/7611741

Research Article

Design and Analysis of Machine Learning Based Technique for Malware Identification and Classification of Portable Document Format Files

Sultan S. Alshamrani

Department of Information Technology, College of Computer and Information Technology, Taif University, P.O. Box 11099, Taif 21944, Saudi Arabia

Correspondence should be addressed to Sultan S. Alshamrani; susamash@tu.edu.sa

Received 5 July 2022; Revised 26 August 2022; Accepted 7 September 2022; Published 21 September 2022

Academic Editor: Muhammad Faisal Amjad

Modern day antivirus software, which is available commercially, is incapable of providing the protection from the malicious portable document format (PDF) files and thus considered as a threat to system security. In order to mitigate the same to some extent, a new PDF malware classification system based on machine learning (ML) is introduced in this paper. The novelty of this system is that it will be inspecting the given PDF file both statistically and dynamically, which in turn will increase the accuracy of finding the correct nature of the document. This method is nonsignature-based and hence can possibly distinguish obscure and zero-day malware. The experiment is carried out for this system by deploying five different classifier algorithms to find out the best fit for the system. The best fit approach is analyzed by calculating the true positive rate (TPR), precision, false positive rate (FPR), false negative rate (FNR), and F1-score for each of these classifier algorithms. Comparison of this work is carried out with previously existing PDF classification systems. A malicious attack on to the proposed system is also implemented, which will in turn obfuscate the malicious code inside the PDF file by making it hidden during the parsing phase by the PDF parser. It has been inferred that the proposed approach achieved F1-measure of 0.986 by using the random forest (RF) classifier in comparison to state-of-the-art where F1-measure was 0.978. Thus, our approach is quite effective in the identification of the malwares when embedded in the PDF file in comparison to the existing systems.

1. Introduction

In the current generation of the digital world, most of the activities are centric towards the usage of the Internet, and thus, it becomes more important to safeguard our applications, data, and information in the presence of various attackers who are always trying to devise new malicious codes and attacks to compromise the resources. Hence, malware analysis becomes one of the prime concerns today as various malwares are generated by attackers and even their properties are changing very rapidly day by day. Nowadays, malware is not the same one that was there before as they change its signatures with time and thus difficult to trace. So, identification and classification of the latest malware is one of the most sought-after areas of research. There are majorly two ways for malware identification: one is a signature-based detection technique and the another one is behavior-based. The signature-based technique is quick and efficient only for identifying known malware and the behavior-based technique is able to identify unknown and complex malware to some extent using machine intelligence and other approaches, but the behavior-based technique is a complex one. None of the methods can detect all kinds of malware, especially when the count of malware is increasing day by day. In the signature-based approach, unique signature is created by using the attributes of the underlying object. The presence of digital signature is efficiently detected by scanning the object by the algorithm. In the behavioral-based approach, intended actions of objects are evaluated before such actions are carried out. This approach analyses the potential behavior of any actions carried by objects before the actual execution of behavior. The older malware was easy to detect as they were able to hide their features, but the malware uses different techniques like obfuscation [1–4] to hide their identity for a longer span and even they can bypass the firewall and other security checks present in the network or system. Also, multiple types of malwares are used to launch the attack, so the effects are more devastating.

There are majorly two ways for analyzing the malware: static and dynamic. In the static approach, malware is inspected without running the code it embeds, whereas in the dynamic approach, it is inspected by running its code [5, 6]. Thus, malware identification is one of the foremost steps in the malware analysis process. The static analysis does not require executing any malware samples and is very simple. There is no need to cover each phase of the process while performing static malware analysis. Dynamic analysis involves the detailed analysis for malware detection. A complete behavior of actions is thoroughly analyzed, while the process remains under execution. This analysis requires detailed monitoring for processes. Classification of malware is also important as identification is. The various categories of malware are viruses, trojan horse, worms, rootkits, ransomware, and key logger. There are various ways by which the classification can be carried out for malware such as feature-based techniques [7, 8] and image classification where the binary values are transformed to image. So, for better classification, more information will be fruitful [9, 10]. For better classification, good classifiers are required to be developed for the better and accurate classification of malware using some latest machine intelligence techniques.

One of the most widely used document formats is PDF. Despite the general public's ignorance, it quickly emerged as a critical attack vector for computers. Hackers may take over a victim's computer using dozens of flaws found in adobe reader. In addition, antivirus software developers have a difficult time protecting PDF files from assaults because of the file's complex internal structure and the vast variety of obfuscation techniques already in use [11]. Most of us send attachments in the PDF format since it is recognized for its mobility and small weight. However, we have no idea what kinds of assaults these files may be used for or propagated to. The three primary forms of PDF malware are vulnerabilities, phishing, and exploitation of PDF features. Vulnerability in the PDF reader's API is exploited by exploit kits, allowing the attacker to run an arbitrary code on the compromised machine. In most cases, JavaScript code is included in the file to do this. However, in phishing assaults, an unsuspecting file is used to trick the user into clicking on an infected link. These campaigns have just lately been uncovered, and they are significantly more difficult to recognize. A malicious program may be downloaded or a website's login credentials may be stolen by any of these assaults.

The static analysis makes use of some techniques for identification such as file format inspection, string extraction, fingerprinting, which primarily used hash code values, antivirus scanning, and disassembly where the machine code is changed to assembly language [12]. Static methods are the time taking ones and also more based on behavior analysis of malware. But in the dynamic approach, the behavior is monitored, while the file is executing for any malware identification. So, it has more leverage to identify the malware. Similarly, detection of malware is primarily done through two ways: one is signature-based where the predefined signatures, if there are any, of malware are used for detection and the other one is a heuristic-based approach where multiple factors are used that contrasts the malicious behavior. One of the challenges in the signature-based approach is that attackers develop the malware by changing their signatures so many times that they are hard to trace. Hence, the heuristic approach is more favorable owing to its capacity to identify polymorphic and some latest attacks.

Using heuristics, sequences of code, and string comparison, signature-based algorithms may determine if a PDF is benign or malicious. However, this has not been demonstrated to be effective against stealth assaults of the present day. If one wants to find the hidden malicious behaviors of a particular file, dynamic approaches are more successful since they run the file in a supported environment and analyze the process it goes through as well as the API calls it makes and build a thorough record of its activities. One may learn a lot about a file's characteristics by looking at the execution log. Because the attributes that are employed to identify malware vary depending on the approach, this is true for all methods of malware detection. Like in a signature-based byte sequence, all methods of malware detection,such as Dynamic link libraries (DLL), behavior-based API and system calls, heuristic used operation code, context-free grammars, and some new techniques such as mobile-based used android permissions and system calls are used [13].

The novelty of the proposed approach given in this paper is signature-less driven criteria. The suggested model will evaluate the API calls processes inside the PDF file and will thus look for the activities that will be performed throughout the file's processing. The detection may be dependent on the system calls and JavaScript files inside the PDF file that have been evaluated. The data mining technique is used in this system to collect information from API requests. It is possible to categorise a particular file as being either "Ordinary (O)" or a "Potentially Malicious (PM)" based on the retrieved characteristics and statistics. Finally, these results are sent via the classification block which maps the gathered information with the algorithm's findings and classifies the file as "correct" or a "malicious" file.

The main highlights of this paper are as follows:

(1) It provides a novel ML-based malware identification approach for the PDF files

(2) It provides the training and testing implementation of the proposed model under the various ML approaches

(3) It also highlights the efficiency of the proposed approach under the simulated malicious attack

The paper is divided into the sections as per their relevance. The work already done in the context of malware identification using the ML and PDF file based has been elaborated in Section 2. The proposed techniques have been defined in Section 3, which is followed by the dataset details under Section 4. Results and the inference drawn have been defined in Section 5. Also, the comparative analysis with existing models has been done in Section 6. Conclusion is highlighted in Section 7.

2. Related Work

In this part of the paper, the researchers' main efforts are in detecting and classifying malware using machine learning (ML) and other approaches. Also, the work done pertaining to the file types that are utilized for the malware identification has also been expressed.

2.1. Malware Identification Related Work. Malware analysis focuses on finding the operation modus of malware and how it affects the programs and systems. Historically, signature-based identification approaches were widely used. This technique works against known malware quickly and effectively but does not work with respect to the zero-day malware properly [14, 15]. A malware identification framework oriented on the genetic algorithm (GA) and signature generators [16] was proposed by authors. While the authors claim that this methodology may identify unknown malware, the paper does not include significant information for the proposed framework, such as testing results, the amount of malware studied, and a comparison to other current studies. Fukushima et al. have defined [17] a behavior-based detection method. New and encrypted malware may be detected using the proposed approach on Windows OS. In [18], a supervised ML method is suggested. The model utilized an SVM kernel basis that weighs the frequency of each library call for the detection of Mac OS X malware.

Recently, with the advent of intelligence techniques, ML has also become one prominent way in malware analysis. Deep learning is an ML subcomponent, which is a heritage from artificial neural networks (ANNs). It is a novel method and is widely utilized for the analysis of images and autonomous cars, but is not enough for virus detection. Although it quite effectively and significantly decreases the area for features, it does not prevent assaults from evasion. Shabtai et al. [19] proposed taxonomy for malware identification by reporting certain sorts of functions and selecting features in the literature, using ML methods. They focus largely on the selection of features. In [20], author has provided a detailed survey of ML for malware analysis. They have mentioned the challenges of datasets and the ways to overcome them. Image transformation with ML is used for malware identification by the author where the convolution neural network (CNN) is utilized [21]. Similarly, the work in the direction of tools usage and framework representation for the malware analysis has been carried out by the researchers recently [22–25].

2.2. File-Based Malware Identification Related Work. In [26], authors examined PDF design and JavaScript information included in PDFs from top to bottom. With regard to design and metadata, they created an extensive set of capabilities, such as the count of bytes per second, the encoding scheme, object names, catchphrases, and comprehensible strings in JavaScript. Also, when the characteristics vary, it is difficult to create antagonistic models since little changes are strong for AI calculations. They built up a classification model utilizing discovery type models keeping structures and data features to limit the danger of ill-disposed assaults. To approve the proposed model, they fabricated an adversarial attack. In [27], authors have presented an outline of the PDF; also, the current assaults are used to be carried out on PDF malware through solid assault models gathered in nature. They depicted how to play out a measurable examination of a PDF record to discover the proof of implanted malware utilizing programming strategies. They examined some of the new PDF malware detection apparatuses dependent on AI that can uphold computerized scientific examinations; recognizing dubious documents before a more profound, a more definite statistical evaluation is released. They examined the PDF constraints and other open issues, particularly regarding the misuse of their weaknesses to possibly misdirect resulting measurable investigations. At last, they recommended tips for improving the exhibition of such frameworks enduring an onslaught and sketch promising analysis. In [28], authors have focused on the malware implanted in PDF files as a delegate instance of modern-day cyber-attacks. They started by giving a scientific classification of the various methodologies used to produce PDF malware. To combat PDF malware classifiers based on learning, they have utilized an adversarial AI structure that has been shown effective. For example, this method enables us to identify existing flaws in learning-oriented PDF malware locators and to identify fresh attacks that may jeopardize such frameworks, along with the possibility of protective measures. In [29], authors have planned and executed a novel framework called AIMED, utilizing hereditary calculations to sidestep malware classifiers. Their tests proved that an opportunity to accomplish ill-disposed malware tests can be diminished up to half, contrasted with exemplary arbitrary approaches. Also, they carried out AIMED to create ill-disposed models utilizing individual malware scanners as target and tried the adversarial documents against additional classifiers from both examination and industry. The created models accomplished up to 82% of cross-avoidance rates among the classifiers.

In [30], authors have exhibited how the most pessimistic scenario conduct of a malware classifier regarding explicit vigor properties can be evaluated. Besides, they found that preparation of classifiers that fulfill officially checked vigor properties can build the avoidance cost of unbounded assailants by dispensing with straightforward assaults avoidances. They proposed another distance metric that works on the PDF tree structure and determined two classes of strength properties including subtree inclusions and erasures. They used the best in class irrefutably vigorous for preparing a strategy to construct strong PDF malware classifiers. A PDF malware classifier, PDFrate, is used by the authors later in [31] to evaluate their methods. Using data from a real network, they demonstrate that high quality classifier arrangements can make the majority of predictions. It is clear that the classifier cannot reliably predict the outcomes of most avoidance efforts, including nine focusing on imitation scenarios from two current projects. Over 100,000 PDF files as well as 100,000 Android apps are part of their evaluation. In [32], authors presented "Hidost," the primary static AI based malware discovery framework intended to work on various file extensions. Broadening a formerly distributed and profoundly viable strategy, it consolidates the coherent design of documents with their substance for better identification precision. On account to its specific plan and general list of capabilities, it is extended to differentiate organizations whose coherent design is co-ordinated as a chain of command.

In [33], authors presented a novel AI framework for automating the discovery of malicious PDF files. Both of the structure and data in the PDF are extracted, and a sophisticated parsing mechanism is included. As a result, a broad range of malware may be distinguished, comprising parsing-based and non-JavaScript malware. Additionally, with a cautious decision of the learning calculation, their methodology has given an altogether higher exactness contrasted with other static examination methods, particularly within the sight of ill-disposed malware control.

To identify JavaScript-induced malware, the authors of [34] employed AI algorithms to get a sample of API references that depict the malicious code. An important application area was examined in this investigation, namely, the placement of the malicious JavaScript code in PDF files. Although their training data contained instances of malware, they demonstrated that their strategy has been able to identify new malware even when it was introduced into an existing system that had not previously been exposed to such malicious code. In [35], authors built up a framework that utilizes various feature selection and AI-induced techniques to set up the attributes of typical JavaScript code.

3. The Proposed Approach

PDF documents include a header, body, cross-reference table (CRT), and a trailer. Components in the body include information about the document itself, while the header provides the information about the document's current version. Tables used to connect to objects are included in the CRT. The root object and the table locations of the objects in the body area are included in the trailer part.

The proposed ML-based malware categorization technique is explained here. For the most part, this system is designed to scan the PDF file being inspected, sort out its internal code, and determine if it is good or dangerous. The hacker's attempts to obfuscate file headers have also been found to be blocked by the mechanism in place. This technique does not identify the malware family contained inside a particular file, but it does accurately classify the file's type [36]. System's categorization procedure of the proposed work is shown in Figure 1. Even if this is a high-level system design, it provides a better idea of how the classifier is implemented. To begin the inspection procedure, the PDF file document must be uploaded to the system. After the document is submitted, it is first analyzed for its information and structure. It is tagged for additional assessment if it follows the pattern of known harmful files. This saves a lot of time and improves speed. In other circumstances, when it does not fit the pattern of hostile instances, the feature extraction module analyses the whole file structure and derives the features from it. It is then given to the classifier component for evaluation once characteristics have been extracted from the PDF. The main ML algorithm is located in the classifier component, and it is this algorithm that is responsible for thoroughly examining the information provided by the feature extractor component [37]. The classifier will categorise the PDF file as either a "correct" or an "infected" file after doing the necessary data analysis.

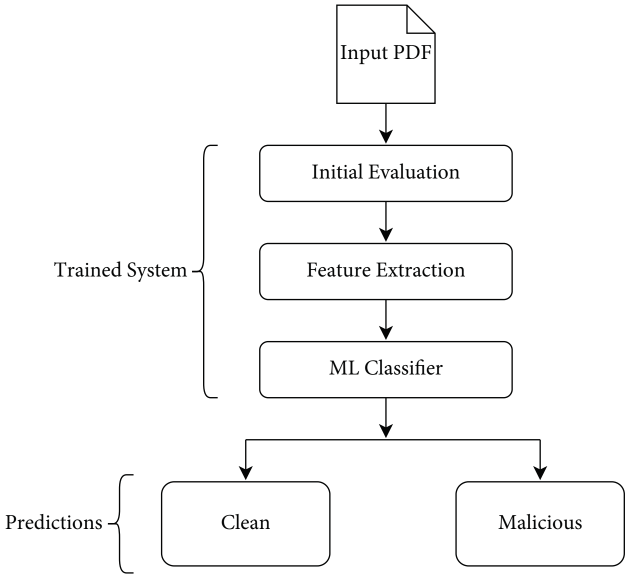


Figure 1: Proposed layout working of the model.

There may be some suspicious API references in the code that can only be discovered via the dynamic code assessment, which can only be done through a new static analysis. The static and dynamic code inspection both employed the same monitoring method as with the PhoneyPDF to keep an eye out for any API references. SpiderMonkey and Rhino are two examples of open-source facilitators that have been used to conduct dynamic investigations in the past. The JavaScript ECMA standard is seen by these translators, but they are unable to comprehend JavaScript connections to the Acrobat PDF format, unless Adobe DOM duplication occurs [38].

A reference design is developed by selecting a subset of API reference that depicts the harmful JavaScript code. Using a collection of PDF files that are either clean or malicious, our system can automatically build a specific set. Acrobat PDF API perceives all JavaScript objects, strategy, and capacity constants as part of the "H" arrangement. This enables us to define "Φ" as the arrangement of all JavaScript objects, strategy, and capacity constants as well as constants. The total number of harmful and nonmalicious files is equal to M. The following equation depicts the characteristic set provided by all of the references.

$$\sum_{i=0}^{M} (\Phi(H, i)) > \text{threshold}. \tag{1}$$

Also, it is to be noticed that $\Phi(H, i)$ may be holding two values and signifies −1 and +1. Also, if result comes out +1, then it signifies malicious PDF. If the value is −1, then it signifies that the file is a safe one.

4. Dataset Details

An overview of the dataset is provided in this section that is used in this proposed research. Following the benign set, the malicious dataset that we analyzed was provided. A total of 1200 PDF samples, both malicious and safe, have been obtained for the investigation. An 800-sample training set has been employed, and 400 samples have been used for testing as depicted in Table 1. It must have been important to have a ratio of good files to malicious files in the training and testing sets of 1 : 1. The majority of the samples are based on genuine cyber-attacks that have been made public. Samples are gathered from a variety of locations over the Internet.

Because the JavaScript code is included in many of these PDF files, some classifiers believe they are all malicious because of the file's large size. The approach in this work, on the other hand, does not use file size as a criterion for determining whether or not a file is harmful. To demonstrate this, the harmless JavaScript code is purposely inserted into nondangerous PDF files in order to make them seem as though they included the malicious JavaScript code. All dangerous and safe PDFs have been analyzed independently and the average size of safe and malicious files was of only approximately 800 kB difference, as shown in Table 2.

5. Implementation and Results

In this section, the various approaches that have been executed for the analysis and implementation of the proposed model are described. Here, the training and testing part is done consequently and the results inferred are discussed. A variety of methods have been used to study and implement this suggested approach, and they are all discussed in this section. This system has been trained on 800 PDFs using several ML classification techniques. With five alternative algorithms, including stochastic gradient boosting (SGB), random forest (RF), decision tree (DT), support vector classifier (SVC), and logistic regression (LR), a comparison is carried out to check how well the system performs under these algorithms.

The effectiveness of the proposed work is reflected by confusion matrix parameters obtained after classification. The confusion matrix comprises of training, testing, validation, and a combined matrix that reflects $\text{true}_{\text{positive}}$, $\text{true}_{\text{negative}}$, $\text{false}_{\text{positive}}$, and $\text{false}_{\text{negative}}$ outcomes. These parameters are further used to calculate the performance parameters like precision, recall, and F1-score using the following equations, respectively.

$$\text{Precision} = \frac{\text{true}_{\text{positive}}}{\text{true}_{\text{positive}} + \text{false}_{\text{positive}}}, \tag{2}$$

$$\text{Recall} = \frac{\text{true}_{\text{positive}}}{\text{true}_{\text{positive}} + \text{false}_{\text{negative}}}. \tag{3}$$

$$\text{F1 score} = 2 * \frac{(\text{precision} * \text{recall})}{(\text{precision} + \text{recall})}. \tag{4}$$

Figure 2 signifies that deploying RF, LR, and DT takes the least amount of time possible, and thus, they perform the classification in a faster manner. In comparison, the SGB's efficiency is average, whereas the SVC's is poor.

The "True Positive Rate (TPR)" is computed by placing the various classifiers during testing, and the trend line is produced. The system should have a higher TPR score in order to be the optimal match. Figure 3 shows that the RF approach has the highest TPR score among all the options. Moreover, the SVC seems inappropriate for file functional testing since it has the lowest TFR score value.

The same holds true when this suggested system's "Precision Score (PS)" under various classifiers was tested. Using Figure 4, it can be concluded that the RF is providing an average PS ratio of approximately 96 percent and that the SVC is providing the lowest PS at roughly 72 percent.

When calculating the "False Positive Rate (FPR)" when testing the system, it is deduced from Figure 5 that the RF has the lowest FPR score, which is preferred, and the SVC has the highest FPR score, which demonstrates its ineffectiveness.

During the calculation of the "False Negative Ratio (FNR)" score, Figure 6 shows that the RF, DT, and SGB all have the lowest FNR scores; hence, they come strongly recommended. SVC and LR, at the other hand, have a high FNR score.

In addition, RF has shown the best overall F1-score on the dataset when compared to the SGB. The DT's F1-score remained similarly moderate. RF has shown to be the best fit for our proposed system, whereas SVC had the worst results when tested with our system.

A number of other PDF malware classification techniques, created by a variety of authors, have been tested. It is evident that our system has the best fit when utilizing the RF classifier based on previous parts. Extraction of features relies on API calls performed by the document as well as the JavaScript code included inside its contents. The F1-score of the various classifiers is determined, and it is inferred that for the proposed classification method, it is higher in contrast to other classifiers as mentioned in Table 3. The numbers (F1-score) shown in the table are derived utilizing the same dataset, through which the testing was executed earlier.

6. System Analysis under Attacks

Malicious samples are developed to resist our system after it had been built, and it is supported by developing a

Table 1: File count for testing and training.

	Safe files	Malicious files	Total files
Count for training	396	404	800
Count for testing	175	225	400
Total count			1200

Table 2: File size (kB) details of dataset for evaluation.

Category type of file	Maximum size	Minimum size	Average size
Safe	21,365	2	10,657
Malicious	22,061	2	11,321

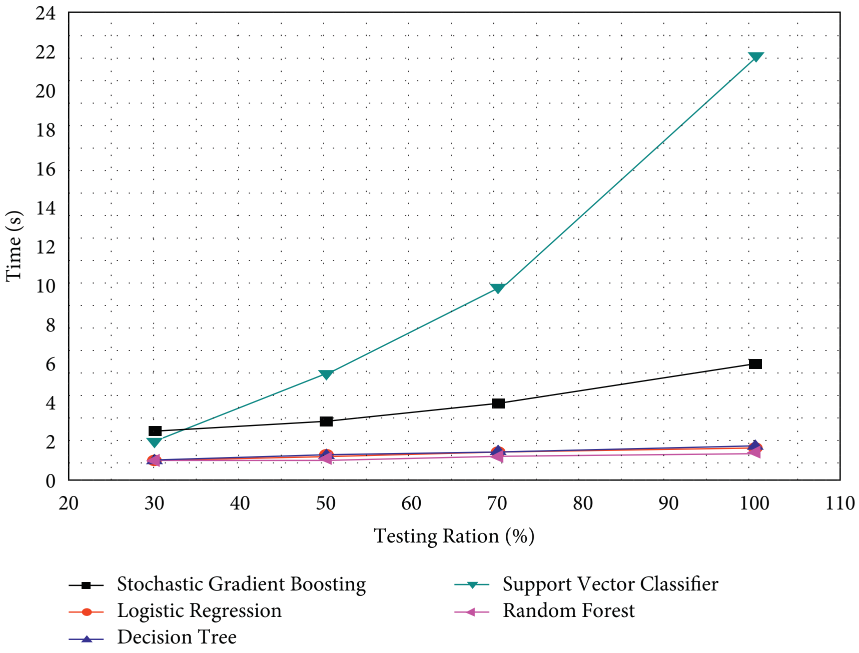


Figure 2: Performance evaluation under various classifiers.

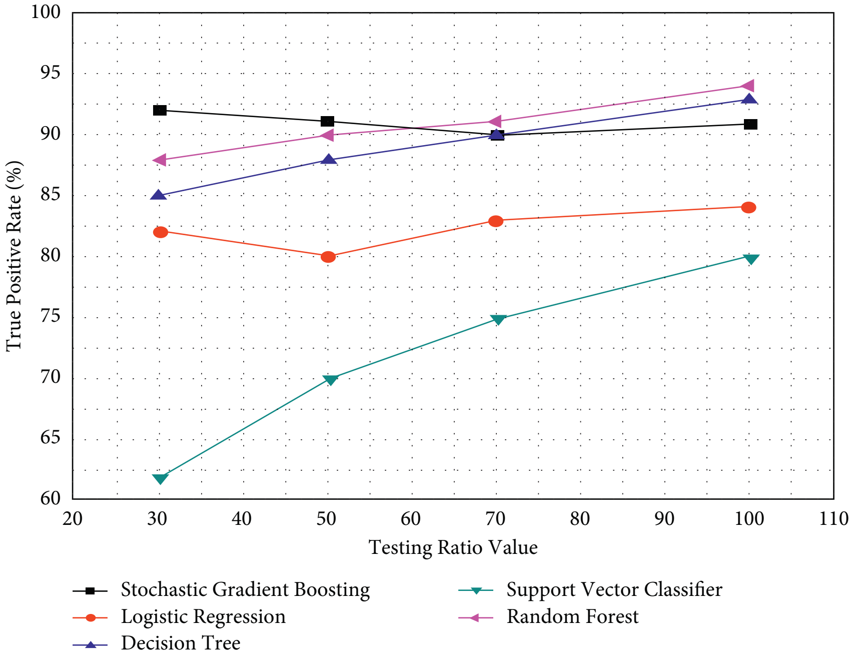


Figure 3: Analysis of TPR under the testing phase.

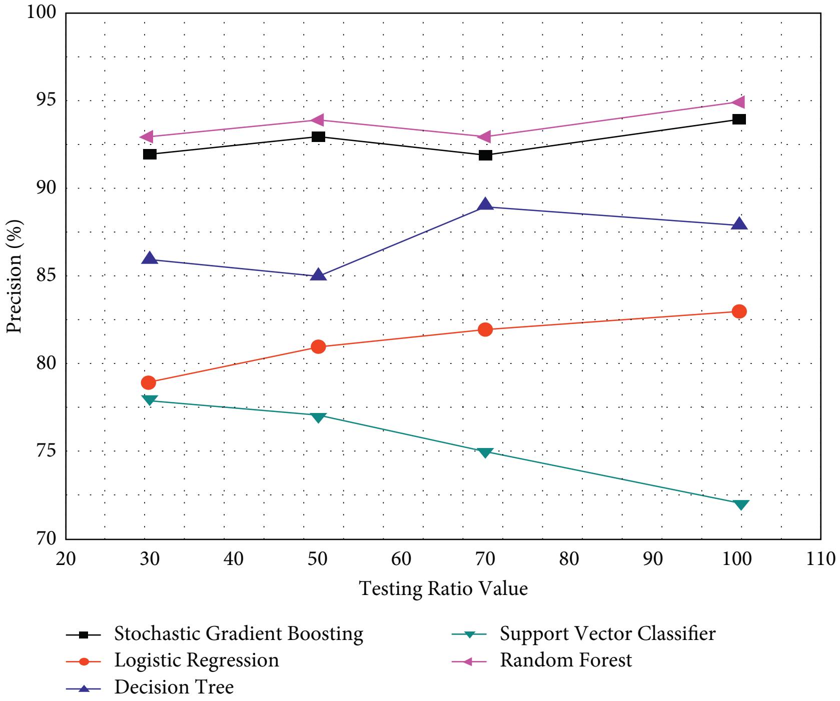


Figure 4: Precision evaluation under various classifiers.

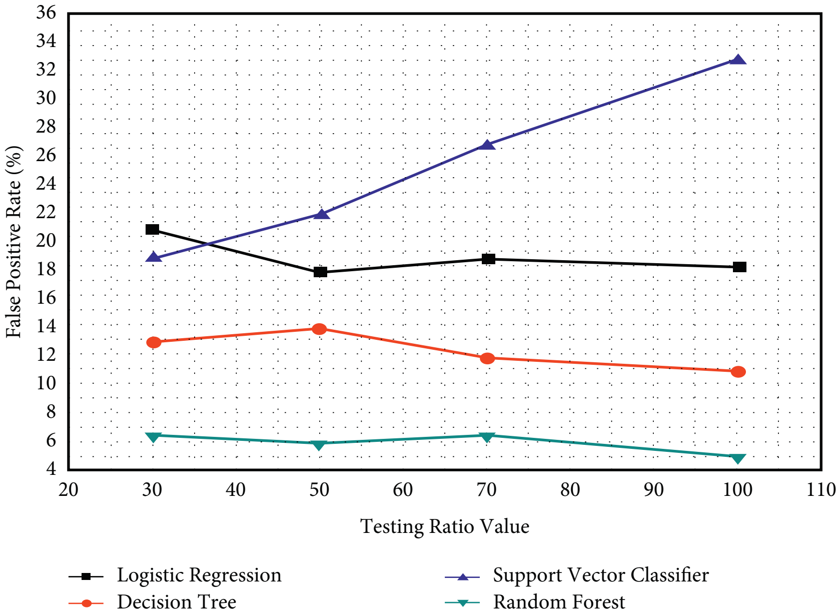


Figure 5: FPR evaluation under various classifiers.

mechanism to recognize those types of attackers during the testing step. As a general rule, while parsing PDF files, the parser first travels to the trailer and retrieves the location of the first item in the list of items in the body. When the first object has been entirely parsed, the program returns to the cross-reference table (CRT) and receives the second item's address. Since the harmful code is not processed or read when a PDF reader is requested, this work deleted the references to the body section objects that contain the dangerous code. Because of this, we may fool the parser into thinking that the file is secure, even if it has a harmful code inside it. If one wants to deceive the system into thinking a malicious file is safe, one may use this method. This is despite the fact that it has been tested using dynamic classifiers, which means that it can be inspected throughout the course of its execution. This code does not execute because it does not include any references to the portions of the body mentioned above. As a result, we may also send the

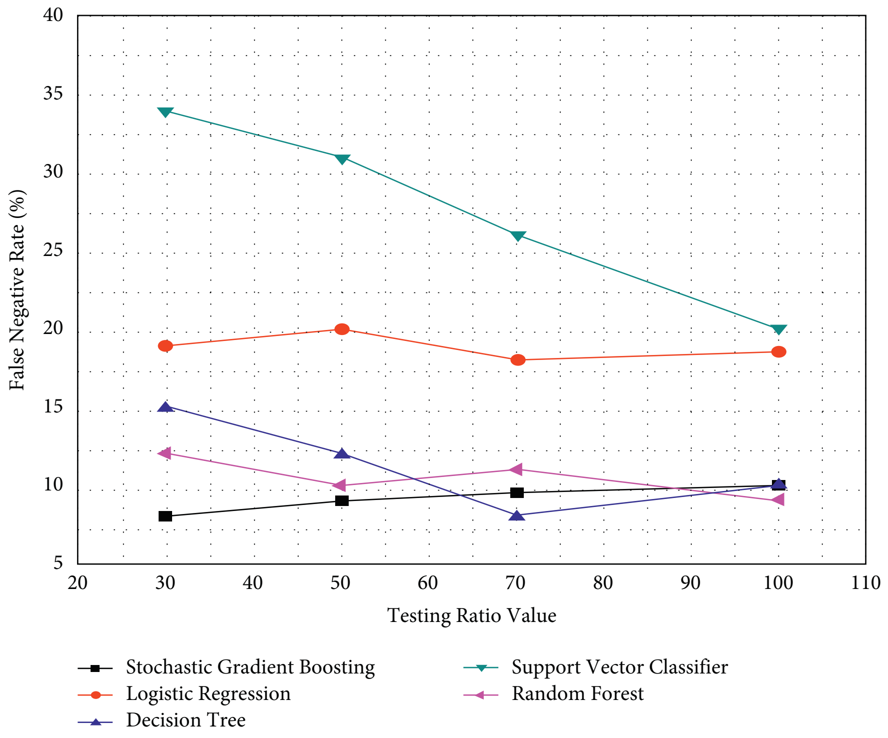


FIGURE 6: Analysis of FNR under the testing phase.

TABLE 3: Comparative analysis of the proposed model with existing based on F1-score.

Tool reference details	Classifier used	F1-score
[39]	SVM	0.828
[40]	RF	0.982
[41]	RF	0.778
[32]	RF	0.818
[34]	RF	0.980
[33]	AdaBoost	0.960
[31]	DT	0.658
[35]	Bayesian	0.978
Proposed model	RF	0.986

TABLE 4: Document classification under the malicious attack on the proposed system.

Type of file	Normal stage		Executing the malicious attack	
	Classified	Not classified	Classified	Not classified
Safe	109	16	119	6
Malicious	122	53	170	5

malicious code-infected PDF file during dynamic analysis. The classification of documents under the malicious attacks is given in Table 4.

7. Conclusion and Future Scope

A ML model that can identify JavaScript and malicious API calls attacks in PDF files is provided in this paper. This work also tried out a number of alternative classifiers, including DT, RF, LR, SVC, and SGB, on the dataset to see how they performed. The RF classifiers within this work have produced the best results. A comparison of this approach with other PDF classifiers revealed that this proposed approach has a high F1-score of 0.986, making it 4 percent more efficient than the other most recent PDF classifiers. To further enhance the system's defense against malicious code obfuscation methods, functionality is included to run an object scanner within the PDF document to identify any objects that are not being processed. Unparsed objects containing the malicious code may be easily identified and removed using this approach. Future plans include adding support for other file formats. Use an advanced data mining approach for more detailed insights of documents. The use of ML during the detection and classification phase of malware is highly useful, but it fails against evasion attacks; thus, it must be explored in the future.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The author declares that there are no conflicts of interest.

Acknowledgments

This research was supported by Taif University Researchers supporting Project number (TURSP-2020/215), Taif University, Taif, Saudi Arabia.

References

[1] K. M. A. Alzarooni, "Malware Variant Detection," Doctoral Dissertation, UCL (University College London), London, England, 2012.

[2] W. Stallings, L. Brown, M. D. Bauer, and A. K. Bhattacharjee, *Computer Security: Principles and Practice*, pp. 978–980, Pearson Education, Upper Saddle River, NJ, USA, 2012.

[3] S. Alam, R. N. Horspool, I. Traore, and I. Sogukpinar, "A framework for metamorphic malware analysis and real-time detection," *Computers & Security*, vol. 48, pp. 212–233, 2015.

[4] A. Mehtab, W. B. Shahid, T. Yaqoob et al., "AdDroid: rule-based machine learning framework for android malware analysis," *Mobile Networks and Applications*, vol. 25, no. 1, pp. 180–192, 2020.

[5] Y. Alosefer, *Analysing Web-Based Malware Behaviour through Client Honeypots*, Doctoral dissertation PhD Thesis, Cardiff University, Cardiff, Wales, 2012.

[6] N. Idika and A. P. Mathur, "A survey of malware detection techniques," Technical Report, Purdue University, West Lafayette, IN, USA, 2007.

[7] R. Islam, R. Tian, L. M. Batten, and S. Versteeg, "Classification of malware based on integrated static and dynamic features," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 646–656, 2013.

[8] J. Saxe and K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *Proceedings of the 2015 10th International Conference on Malicious And Unwanted Software (MALWARE)*, pp. 11–20, IEEE, Fajardo, PR, USA, October 2015.

[9] L. Nataraj and B. S. Manjunath, "SPAM: signal processing to analyze malware [applications corner]," *IEEE Signal Processing Magazine*, vol. 33, no. 2, pp. 105–117, 2016.

[10] K. S. Han, J. H. Lim, B. Kang, and E. G. Im, "Malware analysis using visualized images and entropy graphs," *International Journal of Information Security*, vol. 14, no. 1, pp. 1–14, 2015.

[11] D. Liu, H. Wang, and A. Stavrou, "Detecting malicious JavaScript in pdf through document instrumentation," in *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pp. 100–111, IEEE, Atlanta, Georgia, June 2014.

[12] K. Chumachenko, *Machine Learning Methods for Malware Detection and Classification*, Bachelor's Thesis Information Technology, Xamk Kouvolan kampus, Kouvola, Finland, 2017.

[13] Ö. A. Aslan and R. Samet, "A comprehensive review on malware detection approaches," *IEEE Access*, vol. 8, pp. 6249–6271, 2020.

[14] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Human-centric Computing and Information Sciences*, vol. 8, no. 1, pp. 3–22, 2018.

[15] A. R. Javed, M. O. Beg, M. Asim, T. Baker, and A. H. Al-Bayatti, "Alphalogger: detecting motion-based side-channel attack using smartphone keystrokes," *Journal of Ambient Intelligence and Humanized Computing*, vol. 2020, pp. 1–14, 2020.

[16] M. F. Zolkipli and A. Jantan, "A framework for malware detection using combination technique and signature generation," in *Proceedings of the 2010 Second International Conference on Computer Research and Development*, pp. 196–199, IEEE, Kuala Lumpur, Malaysia, May 2010.

[17] Y. Fukushima, A. Sakai, Y. Hori, and K. Sakurai, "A behavior-based malware detection scheme for avoiding false positive," in *Proceedings of the 2010 6th IEEE Workshop on Secure Network Protocols*, pp. 79–84, IEEE, Kyoto, Japan, October 2010.

[18] H. H. Pajouh, A. Dehghantanha, R. Khayami, and K. K. R. Choo, "Intelligent OS X malware threat detection with code inspection," *Journal of Computer Virology and Hacking Techniques*, vol. 14, no. 3, pp. 213–223, 2018.

[19] A. Shabtai, R. Moskovitch, C. Feher, S. Dolev, and Y. Elovici, "Detecting unknown malicious code by applying classification techniques on opcode patterns," *Security Informatics*, vol. 1, no. 1, pp. 1–22, 2012.

[20] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[21] D. L. Vu, T. K. Nguyen, T. V. Nguyen, T. N. Nguyen, F. Massacci, and P. H. Phung, "A convolutional transformation network for malware classification," in *Proceedings of the 2019 6th NAFOSTED Conference on Information And Computer Science (NICS)*, pp. 234–239, IEEE, Hanoi, Vietnam, December 2019.

[22] S. K. Sasidharan and C. Thomas, "ProDroid—an Android malware detection framework based on profile hidden Markov model," *Pervasive and Mobile Computing*, vol. 72, Article ID 101336, 2021.

[23] Y. Jian, H. Kuang, C. Ren, Z. Ma, and H. Wang, "A novel framework for image-based malware detection with a deep neural network," *Computers & Security*, vol. 109, Article ID 102400, 2021.

[24] Y. Li, X. Wang, Z. Shi, R. Zhang, J. Xue, and Z. Wang, "Boosting training for PDF malware classifier via active learning," *International Journal of Intelligent Systems*, vol. 37, no. 4, pp. 2803–2821, 2022.

[25] A. R. Javed, W. Ahmed, M. Alazab, Z. Jalil, K. Kifayat, and T. R. Gadekallu, "A comprehensive survey on computer forensics: state-of-the-art, tools, techniques, challenges, and Future Directions," *IEEE Access*, vol. 10, pp. 11065–11089, 2022.

[26] A. R. Kang, Y. S. Jeong, S. L. Kim, and J. Woo, "Malicious PDF detection model against adversarial attack built from benign PDF containing javascript," *Applied Sciences*, vol. 9, no. 22, p. 4764, 2019.

[27] D. Maiorca and B. Biggio, "Digital investigation of pdf files: unveiling traces of embedded malware," *IEEE Security & Privacy*, vol. 17, no. 1, pp. 63–71, 2019.

[28] Y. Chen, S. Wang, D. She, and S. Jana, "On training robust {PDF} malware classifiers," in *Proceedings of the 29th USENIX Security Symposium (USENIX Security 20*, pp. 2343–2360, Berkeley CA. USA, August 2020.

[29] C. Smutz and A. Stavrou, "When a Tree Falls: Using Diversity in Ensemble Classifiers to Identify Evasion in Malware Detectors," in *Proceedings of the 23rd Annual Network and Distributed System Security Symposium, NDSS 2016*, San Diego, CA, USA, February 2016.

[30] N. Šrndić and P. Laskov, "Hidost: a static machine-learning-based detector of malicious files," *EURASIP Journal on Information Security*, vol. 2016, no. 1, pp. 22–20, 2016.

[31] D. Maiorca, D. Ariu, I. Corona, and G. Giacinto, "A structural and content-based approach for a precise and robust detection of malicious PDF files," in *Proceedings of the 2015 International Conference on Information Systems Security and Privacy (Icissp)*, pp. 27–36, IEEE, Angers, France, February 2015.

[32] I. Corona, D. Maiorca, D. Ariu, and G. Giacinto, "Lux0r: detection of malicious pdf-embedded javascript code through

discriminant analysis of api references," in *Proceedings of the 2014 Workshop on Artificial Intelligent and Security Workshop*, pp. 47–57, Scottsdale, ARI, USA, November 2014.

[33] M. Cova, C. Kruegel, and G. Vigna, "Detection and analysis of drive-by-download attacks and malicious JavaScript code," in *Proceedings of the 19th International Conference on World Wide Web*, pp. 281–290, Raleigh North, CAR, USA, April 2010.

[34] A. Demontis, M. Melis, B. Biggio et al., "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 4, pp. 711–724, 2019.

[35] A. Pektaş and T. Acarman, "Malware Classification Based on API Calls and Behaviour Analysis," *IET Information Security*, vol. 12, no. 2, 2017.

[36] P. Panda, I. Chakraborty, and K. Roy, "Discretization based solutions for secure machine learning against adversarial attacks," *IEEE Access*, vol. 7, pp. 70157–70168, 2019.

[37] B. Chen, Z. Ren, C. Yu, I. Hussain, and J. Liu, "Adversarial examples for cnn-based malware detectors," *IEEE Access*, vol. 7, pp. 54360–54371, 2019.

[38] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[39] P. Laskov and N. Šrndić, "Static detection of malicious JavaScript-bearing PDF documents," in *Proceedings of the 27th Annual Computer Security Applications Conference*, pp. 373–382, Orlando, FL, USA, December 2011.

[40] D. Maiorca, G. Giacinto, and I. Corona, "A pattern recognition system for malicious pdf files detection," in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pp. 510–524, Springer, Berlin, Heidelberg, 2012.

[41] C. Smutz and A. Stavrou, "Malicious PDF detection using metadata and structural features," in *Proceedings of the 28th Annual Computer Security Applications Conference*, pp. 239–248, Orlando, FL, USA, December 2012.