Journal of Artificial Intelligence and Capsule Networks (ISSN: 2582-2012)
www.irojournals.com/aicn/

Malware Classification using Static Analysis Approaches

Dikshyant Dhungana[1], Ashish Sapkota[2], Sabigya Pokharel[3], Sudarshan Devkota[4], Bishnu Hari Paudel[5]

Department of Electronics and Computer Engineering, Institute of Engineering (IOE) Pashchimanchal Campus, Tribhuvan University, Pokhara, Gandaki, Nepal

Email: [1]dikshyantdhungana@gmail.com, [2]asissapkota42@gmail.com, [3]sabigyapokharel@gmail.com, [4]sudarshandevkota22@gmail.com, [5]bishnuhari@wrc.edu.np

Abstract

Malware threats are becoming increasingly complex, thereby posing greater challenges to effective mitigation efforts. It has become more essential than ever to address the malware, as it poses significant threats to individuals, organizations, and governments worldwide. Therefore, effective and more advanced malware classification techniques are necessary to address these malware threats. This proposed study presents an advanced approach to malware classification using static analysis which examines files without executing them. A structured framework was developed for systematic classification which involves gathering raw malware samples from various sources. Raw malware samples were deconstructed and information, such as the frequency of the opcode and the size of the section were collected. Experiments were carried out to assess the effectiveness of several classifiers in terms of accuracy, precision, recall, and F1 score across distinct malware classifications. Random Forest emerged as the best model in the examined dataset, with an accuracy of over 85.0%. These results demonstrate the effectiveness of Random Forest based on extracted datasets. The proposed research focuses on malware detection and classification, thereby enhancing cybersecurity in modern computing environments.

Keywords: Malware, Opcodes, Static Analysis, Unigram Analysis, Multiclass Classifier.

DOI: https://doi.org/10.36548/jaicn.2024.4.008
Received: 15.11.2024, received in revised form: 23.12.2024, accepted: 11.01.2025, published: 24.01.2025

1. Introduction

With rapid advancement in the field of computers, different technologies involved in the system have reached the horizon. Software is one of the examples. To begin with, it is the set of instructions used to perform specific tasks on the computer. Some software is useful, whereas some may affect the system. Those that affect the system are termed malware. Malware is a software program designed to gain access to a computer system without the owner's permission. The advancement in malware causes it to be smarter, that is, it can toy with our information without getting noticed and harm us in the worst way possible. Different malware, such as viruses, worms, Trojans, ransomware, spyware, and adware are classified based on payload, vulnerability exploitation method and propagation methods [1]. A study suggests that more than 100,000,000 malware were registered in 2022 alone [2]. In recent years, there has been the emergence of new malware such as WannaCry, a ransomware that uses encryption to hold victim data to demand ransom [3]. Similarly, REvil stands for Ransomware Evil, which appeared first in 2021, a ransomware, exploits systems using remote access vulnerabilities, turns off blacklisted processes, encrypts files, and infiltrates information [4]. These attacks highlight the sophistication and scale of modern malware campaigns. As cybercriminals constantly develop new techniques and obfuscation methods, the traditional approach fails to detect the malware. However, these approaches fail against the obfuscated malware, which changes its structure to avoid detection. Therefore, a completely new type of automated system is needed for the classification of new types of malware i.e., static analysis; particularly, opcode analysis. Static analysis is a process of analysing a program's binary code without executing it [5]. For this, the malware needs to be in a controlled sandbox environment where we can observe its behaviour. Opcode analysis focuses on the low-level machine instructions that drive program execution so it will be difficult for the malware to disguise itself and reveal its true behaviour. Compared to dynamic analysis, static analysis is faster but also more efficient for large-scale, real-time malware detection.

1.1 Information Related to Malware Families

With evolving cybersecurity and extensive digitization, malware is also evolving and posing new threats. The dataset that is taken into consideration in the study is relatively new, dated 2020. The in-depth study of this relatively new malware or its characteristics comparison has not been studied previously. This section includes a short description of the malware

families that were studied. These malware families are the only classes considered in this research.

Loki is a malware that steals information, which was first detected in February 2015. This spyware arrives as an attachment in emails. Loki does not have a downloading capability. Loki tries to steal usernames and passwords from web browsers such as Chromium, Apple Safari, Google Chrome, Internet Explorer, Opera, etc. Loki creates a backdoor that allows hackers to install additional software.[6] As its name suggests, Formbook malware is a type of spyware that steals data from HTML forms as well as from the autofill feature of browsers before the information reaches the secure server. [7] Amadey is a type of dropper malware that is capable of stealing sensitive information and installing additional malware after receiving the command from the attacker. The first appearance of this malware was in 2018. [8]

The njRAT malware family is a widespread remote access trojan (RAT). It allows attackers to steal passwords, log keystrokes, activate webcam and microphone, give access to the command line and allows attackers to remotely execute and manipulate files and system registry. [9] Smokeloader is a type of malicious program that can gain access to user's systems and install other malware programs. Its existence dates back to 2011, and has been active since then. It performs tasks such as stealing data and launching distributed denial of service attacks.[10]

Understanding the various research done by various researchers on the said topic, we get an overview of the use of machine learning to classify malware using various approaches. Moreover, analyzing the different malware families provided us with an in-depth insight into their characteristics and features. Taking into account all the factors, this research aims to use various static analysis approaches for feature selection and evaluate the performance and capabilities of various classification models.

2. Related Work

An appreciable amount of work has been done in the field of malware detection and classification, using various technologies and algorithms. Malware detection and classification can be carried out based on static and dynamic analysis. Here we have used the static analysis approach as it is more efficient, early detection of malware is possible, detection of dormant

malware is possible and can be scaled to analyze large numbers of files easily and efficiently than dynamic analysis and the process overhead is not possible in static approach [11].

The authors in [12] compared the different static feature extraction techniques for malware detection. They used Byte n-gram features, opcode n-grams and Portable Executables analysis to highlight the different malware signature capture and its impact on improving detection rates by providing statistical difference between legitimate and malicious files. However, this study did not address the dimensionality challenges that arise from high-dimensional data. In [11], Hassen et al. focused on opcode n-grams and control flow graphs to enhance malware accuracy. The authors explored control statement shingling, where the malware code is structured into function and control blocks, to provide additional context in feature representation as well as reduction techniques to tackle dimensionality issues and improve computational efficiency. However, this approach introduced computational overhead due to the complexity of control flow graph analysis. In [13], Islam et al. focus on static analysis methods using string and function features extracted from executable files. They serve as indicators of signatures that can help identify and differentiate malware from legitimate software. This study emphasizes the utility of text-based string matching to identify malware signatures. Although this study identified specific malware signatures, it struggles against obfuscated malwares. In [14], Schultz et al. extracted the features that represented features contained within each binary. They examined the subset of PE executables using LibBFD and extracted various characteristics of the file such as header information, file sections, import and export functions, code and instructions, entropy, and metadata.

The modelling of extracted features plays a pivotal role in malware detection and classification, with various authors exploring different techniques. In [14], Schultz et al. proposed a data mining framework for malware detection, utilizing algorithms such as Decision Trees, Random Forests, and Support Vector Machines (SVM) to analyze executable files and identify malicious patterns. In [15], Singh et al. expanded on this by evaluating the effectiveness of data mining methods like Naive Bayes and the rule-based RIPPER algorithm. Their approach involved extracting features such as program byte sequences, printed text, and DLL imports to classify new binaries as harmful or benign. However, byte sequence analysis faces challenges from binary packing techniques like UPX, which obscure raw code, complicating detection [16]. To address this limitation, opcode sequences and n-gram features have been proposed as alternatives. These techniques analyse disassembled binary instructions,

enabling robust clustering of malware samples and mitigating the impact of code obfuscation [17]. This evolution highlights the continuous refinement of feature modelling to counter sophisticated malware strategies effectively by focusing on binary-level unpacking techniques. However, this technique causes computational overhead, and sometimes the binary-level unpacking does not show the original code of malware when they are largely obfuscated or dynamically packed.

To fill the gap, we have utilized a dataset of over 200,000 malware samples from 870 unique families ensuring a thorough analysis of malware types unlike earlier studies. By using opcode unigram analysis and portable executables, we have extracted features that effectively captures the malware behaviour, even against obfuscation. We have made use of a custom count vectorizer to analyse specific sections of the PE headers of the executables and opcodes from the disassembled code, which increases the computational efficiency by almost 50%. Among the different classifiers evaluated, Random Forest emerges as the best-performing classifier achieving a validation accuracy of 85%. Furthermore, the static analysis approach used in this experiment offers a scalable and efficient approach to malware detection which is critical for real-time malware detection.

While earlier studies have contributed significantly to malware detection, there were limitations such as computational overhead, high dimensionality, and vulnerability to obfuscation. All these limitations are addressed by this study through robust datasets, feature extraction, and optimized classifier performance, increasing the efficiency of malware classification using static analysis approaches.

3. Proposed Model

The proposed model outlines a comprehensive approach to malware classification, detailing each step from data acquisition to performance evaluation. This includes the dataset, pre-processing, model description, and performance metrics.

3.1 Dataset

A well-structured and quantitative dataset is crucial for effective analysis and accurate predictions in any classification process. For this research, we utilized a dataset of malware samples sourced from Malware Bazaar [18], a widely respected repository known for its

extensive and diverse collection of up-to-date malware samples. The dataset acquired from Malware Bazaar was the raw executables that were in the form of password-protected zip.

We acquired 291 GB of malware samples from Malware Bazaar, covering the period from 2020-02-25 to 2022-08-20. The password-protected zip was extracted using a bash script to get the Windows executable files. Using the Malware Bazaar API, the samples were extracted, labelled, and classified into 231,015 samples, spanning 870 unique malware families. Out of these, 33 families contained more than 1,000 samples. For focused analysis, we selected five prominent malware families— Amadey, Loki, njrat, SmokeLoader, and Formbook—along with a non-malicious class. After preprocessing, the dataset size was reduced to 16 GB, containing 25,522 samples across the six classes. The class distribution of the dataset is shown in the Table.1 below.

Table 1. Class Distribution of the Dataset

Class	Sample Count
Amadey	2703
Formbook	6215
Loki	5686
Njrat	3763
SmokeLoader	5500
Non-malicious	1655

3.2 Data Pre-processing

After collecting the data from various sources, we have to perform a set of steps to make the data suitable for machine learning models.

1. **Data Cleaning**: We have removed the corrupted data, null values, and missing values and eliminated the redundant values as well.

2. **Class Balancing**: To address the class imbalance, we employed the Synthetic Minority Over-sampling Technique (SMOTE), a widely used method for oversampling. SMOTE ensures more balanced class distributions by interpolating between existing samples to

create synthetic samples for the minority classes. SMOTE was used for our dataset's underrepresented classes, Amadey, Njrat, and Non-malicious.

3. **Data Splitting**: The data was split into training and testing sets in the ratio of 80 to 20.

3.3 Feature Extraction

To prepare the data for modeling, the raw executable files were disassembled, and their static features were extracted. This research focused on static analysis to work on the properties of malware that do not require execution. The following steps were employed:

Disassembly using objdump: The executable files were disassembled into assembly language using the objdump tool to examine the structural and functional attributes.

Custom Feature Vectorization Using Count Vectorizer: To analyze specific sections of the PE headers of the executables and the opcodes from the disassembled code, we developed a custom Count Vectorizer. This technique, commonly used in NLP, provided us with the frequency of the text and section counts in the disassembled code.

The following static features were prioritized and extracted for this research:

1. File Size: The size of each executable was recorded, as variations in file size often provide valuable insights into malware characteristics, such as complexity and functionality.

2. PE Sections: Key sections of the Portable Executable (PE) format were analyzed, as these sections often reveal critical details about the structure of the malware. The following sections were examined.

.text contains the executable code. .idata stores import information, including external libraries. .data contains initialized data. .bss holds uninitialized data. .rdata contains read-only data, such as constants. .edata stores export information. .rsrc holds resources like icons, strings, and images. .tls Used for thread-local storage. .reloc contains relocation information for dynamic address adjustments.

3. Opcode Analysis using Unigrams: Opcode (Operation code) represents the instructions in the disassembled code of the executable. We conduct unigram analysis by counting the occurrences of individual opcodes. This provided a granular view of the malware's behaviour and functional pattern. Unlike other features, all unigrams were included

in the analysis as the opcode vocabulary was relatively small, making it computationally efficient to process.

3.4 Model Selection

In creating a robust malware classification system to determine which model offers the optimal performance for the classification problem, model selection is vital, which involves testing multiple machine learning algorithms. Given the multi-class nature of the problem, we experimented with a variety of classification techniques to classify malware into five families and a non-malicious class.

Naïve Bayes classifier is one of the supervised machine learning algorithms used for text classification that is based on Bayes Theorem. Logistic Regression (One-vs-Rest) is a discriminative classifier used for classification and predictive analysis. Logistic regression calculates the likelihood of an event, such as voting or not voting, based on a collection of independent variables. Support Vector Machine(SVM) is a supervised machine learning algorithm that is used for classification as well as regression. SVM algorithm classifies data by finding the maximum separating hyperplane between each class in the target space. Decision tree is a type of supervised learning algorithm that uses a tree-like model to make decisions based on input data. They break down complex decisions into a series of straightforward choices, making them easy to visualize and interpret. K-Nearest Neighbors(KNN) is a supervised learning algorithm that is used in classification. It works by finding the k nearest data points to a query point and making predictions based on their labels. Random forest is a commonly used machine learning algorithm that combines the output of multiple decision trees to reach a single result. When multiple decision trees form an ensemble in the random forest, they predict more accurate results. When using a random forest classifier, various hyperparameters can be used to alter the decision tree used and the ensemble. It is done through hyperparameter tuning, which is choosing the optimal hyperparameter for the model developed.

3.5 Performance Metrics

The methods used in evaluating the performance of the model are mentioned in this section. Accuracy is one of the performance metrics that measures the proportion of correctly predicted instances out of the total instances, and evaluates the overall performance of the model. The performance of a model does not depend on the accuracy alone, but also on other

metrics like precision, recall, and F1-score of each class to evaluate the hyperparameters. These metrics were derived from the confusion matrix, which provides a comprehensive view of the model's performance. Precision is a metric that measures how a model correctly predicts the positive class. The recall is an indicator of the model's ability to identify all the relevant classes. The F1-score provides the single value for the evaluation of the model and is the harmonic mean of precision and recall.

$$Accuracy = \frac{True\ Positive+True\ Negative}{True\ Positive+True\ Negative+False\ Positive+False\ Negative} \quad (1)$$

$$Precision = \frac{True\ Positive}{True\ Positive+False\ Positive} \quad (2)$$

$$Recall = \frac{True\ Positive}{True\ Positive+False\ Negative} \quad (3)$$

$$F1\ score = 2\ \frac{Precision\ x\ Recall}{Precision+Recall} \quad (4)$$

3.6 Tools Used

The machine learning models were implemented using Python, utilizing the following libraries:

Scikit-Learn: An open-source machine learning library used for building machine learning models like Random Forest, SVM, Logistic Regression, etc.

Pandas: A Python library used for analysing, cleaning, exploring, and manipulating data.

Numpy: A Python library used for performing optimized operations on large arrays.

Matplotlib: A library used for data visualization.

GridSearchCV: It is used for hyperparameter tuning.

Objdump: A tool used to disassemble the executable files into assembly language.

4. Results and Discussion

The result and analysis of the corresponding result are mentioned in this section.

4.1 Features Overview

After various pre-processing using the dataset and extraction of the features, those features were transformed into a structured dataset, enabling robust training and evaluation of the classification model. ANOVA analysis (Figure 1) further highlighted the relative importance of these features, as depicted in the following feature importance chart.

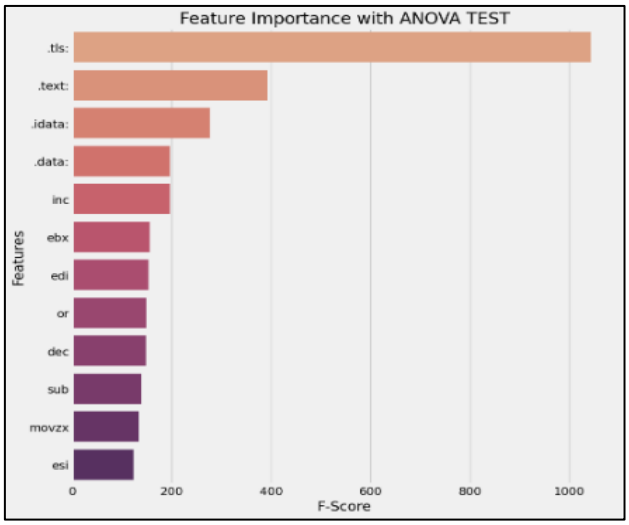

Figure 1. Feature Importance Score of Top 12 Features

As shown in the feature importance chart, the .tls section of the Portable Executable (PE) format, used for thread-local storage, was identified as the most important feature, achieving the highest F1 score. This indicates that the presence and characteristics of the .tls section are highly indicative of malicious behavior. Even if malware attempts to obfuscate its purpose or evade detection through thread-local storage, the machine-learning model effectively flags such anomalies.

The .text section, which contains the executable code, is also ranked as a critical feature, providing significant insights into the malware's functionality. Its high F1-score highlights its importance in distinguishing malicious activities embedded in the executable code. Other PE sections, such as .idata, .data, and .edata also contributed meaningfully to the classification process. These sections offer structural and behavioral details of malware that assist the model in accurately identifying and categorizing different malware families.

Additionally, the frequency of certain opcodes—instructions like INC, MOV, OR, SUB, and JMP—played a crucial role in determining malware behavior. These opcodes are

often associated with malicious operations, such as memory manipulation, control flow alteration, and data tampering, making them valuable features for the model to detect malicious intent.

By combining structural insights from PE sections and behavioural patterns from opcode frequencies, the model effectively distinguished malicious from non-malicious files and classified them into families, showcasing the robustness of the feature selection process.

4.2 Model Selection

The performance of various classifiers (Table 2) was assessed on the pre-processed dataset. Initially, we trained and evaluated models using standard hyperparameters to establish a baseline. We focused from the base level and upwards based on complexity to find the best-performing algorithm for the extracted dataset format.

Table 2. Experiment Results of Different Classifiers

Classifier	Scaled Dataset	Train Accuracy (%)	Val Accuracy (%)
Naïve Bayes	No	40	41
Logistic Regression (OVR)	Yes	61	60
Decision Tree	No	96	79
Support Vector Machines	Yes	66	65
KNN	Yes	96	84
Random Forest	No	98	85

Among the tested algorithms, Random Forest emerged as the best-performing model, achieving the highest validation accuracy of 85%. The high accuracy achieved by Random Forest can be ascribed by its capacity to handle non-linear datasets by combining the results of multiple decision trees. However, Random Forest has limitations, such as difficulties in generalizing to unseen datasets, especially when faced with smaller datasets. Additionally, Random Forest uses multiple decision trees to make predictions so it becomes harder to understand how the model is making decisions causing a lack of clarity. Furthermore, Random Forest's effectiveness is dependent upon hyperparameter tunings, such as the maximum

number of trees and maximum depth of the trees, which needs to be adjusted for various datasets.

K-Nearest Neighbors (KNN) and Decision Tree followed, achieving validation accuracies of 84% and 79%, respectively. KNN performed well but can be computationally expensive, especially with large datasets, as it relies on distance calculations for each instance. Decision Tree showed reasonable accuracy but is prone to overfitting without proper pruning techniques, which limits its generalizability. Among the lower-performing models, the Support Vector Machine (SVM) achieved 65% accuracy but struggled due to the large number of features (51 in this dataset). Logistic Regression achieved 60% accuracy, delayed by its inability to model non-linear relationships and overlapping feature distributions. Applying non-linear feature transformations could enhance its performance.

Naïve Bayes recorded the lowest accuracy at 41%, primarily due to its assumption of feature independence, which is not valid in malware classification where features like PE sections and opcode frequencies are interdependent. Feature selection and ensuring greater independence among features could improve its accuracy. While Random Forest displayed the highest accuracy, it is important to note that its performance may vary with different datasets, and other algorithms such as SVM or KNN could outperform it on more optimized or specific datasets.

To find the optimal performance of the classifier, hyperparameter tuning was performed using GridCV for each classifier. Each classifier has its own hyperparameter, so after exhaustive experimentation, the optimal hyperparameter for each classifier was determined. The classifier and its corresponding hyperparameters are mentioned in Table 3.

Table 3. Optimal Hyperparamter Values

Classifier	Hyperparamter	Value
Naïve Bayes	alpha	1.5
Logistic Regression	multi_class	ovr
Logistic Regression	c	100
Logistic Regression	class_weight	balanced

Logistic Regression	penalty	l2
Logistic Regression	solver	saga
Decision Tree	criterion	entropy
Decision Tree	max_depth	30
Decision Tree	min_samples_split	5
Support Vector Machine	kernel	rbf
Support Vector Machine	gamma	auto
Support Vector Machine	class_weight	balanced
KNN	n_neighbors	5
KNN	weights	distance
KNN	metric	manhattan
Random Forest	n_estimators	220
Random Forest	max_depth	30
Random Forest	min_samples_split	4
Random Forest	min_samples_leaf	2
Random Forest	max_features	10

To evaluate the performance of the classifiers, metrics other than accuracy should also be considered. Precision, recall, and F1-score—commonly used metrics for classification tasks—were calculated for each malware class. Table 4 summarizes the experimental results across different malware classes.

Table 4. Experiment Results of Different Classes

Class	Precision	Recall	F1-Score
Amadey	0.97	0.82	0.89
Formbook	0.71	0.82	0.76
Loki	0.76	0.66	0.70
SmokeLoader	0.92	0.96	0.945
njrat	0.88	0.88	0.88
Non-malicious	0.88	0.89	0.89

The overall result highlights that the model performs well across most classes, with SmokeLoader achieving the highest precision and recall. However, Loki showed relatively lower performance, which could be attributed to variations in its characteristics or insufficient representation in the dataset.

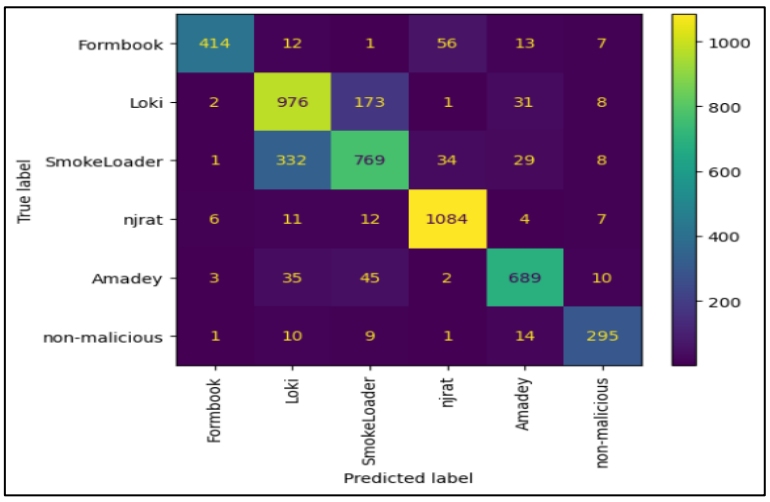

Figure 2. Confusion Matrix for Validation Set

The confusion matrix, shown in Figure 2, further validates the model's classification performance, highlighting its ability to accurately distinguish between malware families. The confusion matrix demonstrates that the model is effective in detecting and classifying malwares between different families. However, due to similarities in the characteristics of some malware

families, there are some misclassifications. To exemplify, Loki and njrat are identified with high accuracy (976 for Loki and 1084 for njrat). On the contrary, SmokeLoader is occasionally misclassified as Loki and njrat because of the overlapping characteristics of the malware families. However, Non-malicious files are generally classified correctly. Despite this misclassification, the high accuracy in the diagonals shows a strong overall accuracy.

Finally, the Random Forest classifier with hyperparameter tuning was identified as the most effective model for this classification task, offering a balanced trade-off between accuracy, interpretability, and computational efficiency.

5. Conclusion

The study, conducted in the malware classification using static analysis, classifies the malware by the family using machine learning. To sum up, the analysis revealed the significant variations in the accuracy among the algorithms due to their inability to handle non-linearity and overlapping characteristics in data. Although algorithms like Naïve Bayes and Logistic Regression struggled with accuracy because of huge noise and overlap, Random Forest demonstrated high accuracy, precision, and recall, making it suitable for such classification tasks. Its ability to handle complex patterns and reduce overfitting. This makes Random Forest the best fit for accurately classifying malware families, thereby mitigating potential threats in the user's system. In the future, the study intends to expand the data set, which was limited in this experimental analysis, with more malware samples due to the evolving nature of malware. Models trained on previous malware samples may not be able to detect new types of malware. Also, will explore more about bigram and trigram opcode analysis, and also use boosting methods and more complex algorithms, such as ANN.

References

[1] Namanya, Anitta Patience, Andrea Cullen, Irfan U. Awan, and Jules Pagna Disso. "The world of malware: An overview." In 2018 IEEE 6th international conference on future Internet of Things and cloud (FiCloud), Barcelona, Spain. IEEE, 2018. 420-427

[2] Dinh, Phai Vu, Nathan Shone, Phan Huy Dung, Qi Shi, Nguyen Viet Hung, and Tran Nguyen Ngoc. "Behaviour-aware malware classification: Dynamic feature selection." In 2019 11th International Conference on Knowledge and Systems Engineering (KSE). IEEE, 2019. 1-5

[3] Chen, Qian, and Robert A. Bridges. "Automated behavioral analysis of malware: A case study of wannacry ransomware." In 2017 16th IEEE International Conference on machine learning and applications (ICMLA),Cancun, Mexico. IEEE, 2017. 454-460.

[4] Jarjoui, Samir, Robert Murimi, and Renita Murimi. "Hold my beer: a case study of how ransomware affected an Australian beverage company." In 2021 International conference on cyber situational awareness, data analytics and assessment (cybersa) Dublin, Ireland. IEEE, 2021. 1-6.

[5] Vinayakumar, R., Mamoun Alazab, K. P. Soman, Prabaharan Poornachandran, and Sitalakshmi Venkatraman. "Robust intelligent malware detection using deep learning." IEEE access 7 (2019): 46717-46738.

[6] S. R. Ariani and R. Lumanto, “Study of lokibot infection against indonesian network,”

[7] N. Villeneuve, R. Eitzman, S. Nemes, and S. Dean, Formbook, https://cloud.google.com/blog/topics/threat-intelligence/formbook-malware-distribution-campaigns/, 2017.

[8] Z. Tilsiter, Amadey, https://darktrace.com/blog/amadey-info-stealer-exploiting-n-day-vulnerabilities, 2023.

[9] Rashid, Salar Jamal, Shatha A. Baker, Omar I. Alsaif, and Ali I. Ahmad. "Detecting Remote Access Trojan (RAT) Attacks based on Different LAN Analysis Methods." Engineering, Technology & Applied Science Research 14, no. 5 (2024): 17294-17301.

[10] THREATLABZ, Smoke loader, https://www.zscaler.com/blogs/security-research/brief-history-smokeloader-part-1, 2024.

[11] Hassen, Mehadi, Marco M. Carvalho, and Philip K. Chan. "Malware classification using static analysis based features." In 2017 IEEE symposium series on computational intelligence (SSCI), IEEE, 2017. 1-7.

[12] Ranveer, Smita, and Swapnaja Hiray. "Comparative analysis of feature extraction methods of malware detection." International Journal of Computer Applications 120, no. 5 (2015).

[13] Islam, Rafiqul, Ronghua Tian, Lynn Batten, and Steve Versteeg. "Classification of malware based on string and function feature selection." In 2010 Second Cybercrime and Trustworthy Computing Workshop, IEEE, 2010. 9-17

[14] Schultz, Matthew G., Eleazar Eskin, F. Zadok, and Salvatore J. Stolfo. "Data mining methods for detection of new malicious executables." In Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001,IEEE, 2000. 38-49.

[15] Singh, Prabhat K., and Arun Lakhotia. "Analysis and detection of computer viruses and worms: An annotated bibliography." ACM SIGPLAN Notices 37, no. 2 (2002): 29-35.

[16] M. Oberhumer, L. Molnár, and J. F. Reiser, Upx: The ultimate packer for executables-homepage, 2020.

[17] Hu, Xin, Kang G. Shin, Sandeep Bhatkar, and Kent Griffin. "{MutantX-S}: Scalable malware clustering based on static features." In 2013 USENIX Annual Technical Conference (USENIX ATC 13), 2013. 187-198

[18] M. Bazaar, Malware bazaar, https://bazaar.abuse.ch/browse.php, 2024.

Author's biography

Dikshyant Dhungana holds a Bachelor's Degree in Computer Engineering from Tribhuvan University. He has a strong interest in Distributed Systems, Cloud Computing, High Performance Computing and Artificial Intelligence, with a deep passion for advancing technology through research and innovation. He is interested in building scalable, fault-tolerant and high-performance systems.

Ashish Sapkota holds a Bachelor of Science in Computer Engineering. He is currently working as a Flutter developer, focusing on building cross-platform mobile applications. In addition to his professional role, he is passionate about contributing to advancements in the fields of Internet of Things (IoT) and Security.

Sabigya Pokharel is a Computer Engineering graduate with a strong interest in research and development. She is currently working as a Flutter developer, specializing in building cross-platform mobile applications. Her research interests include deep learning and convolutional neural networks.

Sudarshan Devkota is a computer engineering graduate with a strong interest in research and development. Currently working as a Java developer, he focuses on creating scalable and efficient software solutions while exploring advancements in technology and software engineering methodologies.

Bishnu Hari Paudel is an Assistant Professor in the Department of Electronics and Computer Engineering at Pashchimanchal Campus in Tribhuvan University, Nepal. He holds a dual Master's degree, having earned a Masters in Information Technology at University of Southern Queensland, Australia and Masters in Science in Communication & Knowledge Engineering at Tribhuvan University of Nepal. He also completed his Bachelor of Engineering in Computer Engineering from Pokhara University, Nepal. His research interests include Distributed Systems, Geographic Information Systems (GIS), and Artificial Intelligence (AI).