Digital Innovations
& Contemporary Research in SCIENCE, ENGINEERING & TECHNOLOGY
Vol. 12. No. 2, 2024 Series

Article Citation Format
Christopher, L. U. & Ayorinde, I. T. (2024): Malware Detection Using Hidden Markov Model. Journal of Digital Innovations & Contemporary Research in Science, Engineering & Technology. Vol. 12, No. 2. Pp 37-46. www.isteams.net/digitaljournal. dx.doi.org/10.22624/AIMS/DIGITAL/V11N2P4

Article Progress Time Stamps

Article Type: Research Article
Manuscript Received: 21st March, 2024
Review Type: Blind Peer
Final Acceptance:15th April, 2024

Malware Detection Using Hidden Markov Model

[1]Christopher, L. U. & [2]Ayorinde, I. T.
[1&2]Department of Computer Science
University of Ibadan
Ibadan, Nigeria.
Emails: [1]xtoluck@gmail.com; [2]temiayorinde@yahoo.com

ABSTRACT

Malware is a broad term for harmful software that poses significant threats by damaging computer systems and spreading across networks. Traditional detection methods include signature-based and heuristic-based techniques, which are effective against known malware but struggle with new, unknown variants, particularly sophisticated ones like metamorphic, encrypted, and polymorphic viruses. Hence, this research aims at improving malware detection, specifically targeting metamorphic malware that can evade traditional detection methods. The study shows the effectiveness of dynamic analysis over static analysis for detecting metamorphic malware due to its ability to adapt to the malware's constant changes. The dynamic analysis involves examining malware behavior during execution, using dynamic software birthmarks and the Hidden Markov Model to identify malicious activities. It therefore recommends the use of dynamic analysis in the detection of patterns of metamorphic malware.

Keywords: Hidden markov models, Malware detection, Dynamic analysis and Static analysis

1. INTRODUCTION

Malware is a software designed to infiltrate or damage a computer system without the owner's informed consent, encompassing all kinds of computer threats. It can be classified into file infectors and stand-alone malware, or by specific actions. They are viruses, worms, backdoors, Trojans, rootkits, spyware, and adware. Malware is becoming more sophisticated, making detection and analysis challenging due to improved anti-analysis techniques like obfuscation and packing, and the dynamic and heterogeneous nature of current computing infrastructure [1].

Developing quick detection and eradication techniques for malware is critical for both businesses and consumers. This can be done with the use of software birthmarks which are unique identifiers of software for malware detection, employing both static and dynamic birthmarks and machine learning techniques such as Hidden Markov Models (HMMs) and Profile Hidden Markov Models (PHMMs).

Viruses, a type of malware, are self-replicating and can modify or halt a computer's functioning by infecting various system areas like the boot sector, hard drives, and programs. An example is the Stuxnet virus, which targeted industrial control systems in power plants [2,3,4]. Viruses can infect system sectors, files, macros, companion files, directories, batch files, source code, screensavers, and vulnerabilities. Its categories include polymorphic viruses, stealth viruses and rootkits, fast and slow infectors, sparse infectors, armored viruses, multipartite viruses, spacefiller (cavity) viruses, tunneling viruses, camouflage viruses, metamorphic viruses, buffer overflow, and botnets among others.

Signature-based malware detection technique involves maintaining a database of malware signatures and comparing patterns against this database. While effective for known malware, it struggles with new, unknown malware due to evolving techniques like obfuscation and polymorphism [5,6]. Signature-based detection techniques are limited by the need for frequent updates and their ineffectiveness against sophisticated malware like metamorphic, encrypted, and polymorphic viruses. Heuristic-based detection, also known as behaviour or anomaly-based detection, analyzes the behaviour of applications to detect malware, focusing on source or destination addresses, attachments, and other statistical features [7]. Specification-based detection is a derivative of behaviour-based detection that tries to monitor program executions against manually developed specifications of legitimate system behavior to detect deviations, aiming to reduce false alarms associated with behavior-based detection [8].

Encrypted viruses encrypt their bodies to evade signature-based detection by antivirus software. They decrypt themselves before executing their malicious code. Polymorphic viruses are similar to encrypted viruses, but their decryption code changes with each infection, making them harder to detect. Metamorphic viruses change their code structure in each generation without encryption, using techniques like dead code insertion, instruction reordering, instruction substitution, and register swapping [3]. Worms are self-replicating like viruses but spread across networks. They can exploit technical weaknesses or use social engineering to infect machines. Early examples include the Creeper and Reaper worms, and the notorious Internet Worm of 1988 [9, 10]. Signature-based detection matches files against a database of known malware patterns but struggles with sophisticated malware like metamorphic, encrypted, and polymorphic viruses. Static Heuristics Analysis identifies malware by analyzing behavior and structure, useful for zero-day threats but can yield false positives. Behavior Blocker monitors programs for suspicious activities during execution and stops them if detected [3]. Hidden Markov Model (HMM) based detection utilizes statistical pattern analysis to model malware behavior. HMMs are trained on observation sequences to detect malware based on the likelihood of matching trained models [2] and it is the technique adopted in this study. These techniques highlight the ongoing battle between evolving malware strategies and adaptive detection methods.

2. RELATED WORKS

Numerous research papers focus on malware detection using hybrid methods, yet many existing reviews do not specifically examine these approaches. Reviews by [11, 12, 13] discussed malware detection using static, dynamic, and hybrid approaches but lack focus on the unique aspects of hybrid methods. Hybrid approaches address specific factors such as the working environment, detection order, and detection integrity, which other methods might overlook. [14] reviewed articles on hybrid approaches but covered only five articles, leading to limited scope and depth.

Extensive research exists on static and dynamic birthmarks for malware detection. Static birthmarks, such as the k-gram technique proposed by [15] and dynamic birthmarks, like those proposed by [16], offer insights into opcode sequences and executable traces, respectively. [17] introduced a graph-based technique using Markov chains and Support Vector Machines for improved malware classification. [18] discussed API call sequences and frequencies as robust dynamic birthmarks.

In [19], a lightweight Convolution Neural Network based model extracting relevant features using call graph, n-gram, and image transformations is proposed for malware detection and shows that the federated approach achieves the accuracy closer to centralized training while preserving data privacy at an individual organization. [20] explored the use of deep neural networks in the defence against malware, looking specifically at how recent advances in Image Classification can be used to detect and classify malware, using off-the-shelf free or low-cost software, or cloud services. In [21], Hidden Markov Models (HMM) and Principal Component Analysis (PCA) are used to obtain embedding vectors in an approach referred to as HMM2Vec and generate vector embeddings based on principal component analysis.

The work in [22] explored malware detection and classification elements using modern machine learning (ML) approaches, including K-Nearest Neighbors (KNN), Extra Tree (ET), Random Forest (RF), Logistic Regression (LR), Decision Tree (DT), and neural network Multilayer Perceptron (NNMLP). The study used the publicly available dataset UNSWNB15 and feature encoding method was applied to convert their dataset into purely numeric values. After that, a feature selection method named Term Frequency-Inverse Document Frequency (TFIDF) based on entropy for the best feature selection was applied. The dataset was then balanced and provided to the ML models for classification. The study concluded that Random Forest, out of all the tested ML models, yielded the best accuracy of 97.68 %.

3. METHODOLOGY

Figure 1 shows the flow of the process of the methodology used in this study. This is explained as follows:

Step 1: Data collection involving malware dataset
Step 2: A controlled Virtual environment was setup where the dataset collected at step 1 was extracted and executed. This was to prevent the host machine from malware attack.
Step 3: Extract the features based on API call logs information.
Step 4: Generate string list based on the whole dataset using Octave 4.0 script.
Step 5: For each sample, check the occurrence of each string from the string list, mapping each sample into a vector.
Step 6: Obtain a model from the HMM
Step 7: Call Matlab libraries to train the model using the training dataset.
Step 8: Evaluate the test data.
Step 9: Iterate for a number of times for different malware samples.
Step 10: Check the accuracy level and plot the graph.

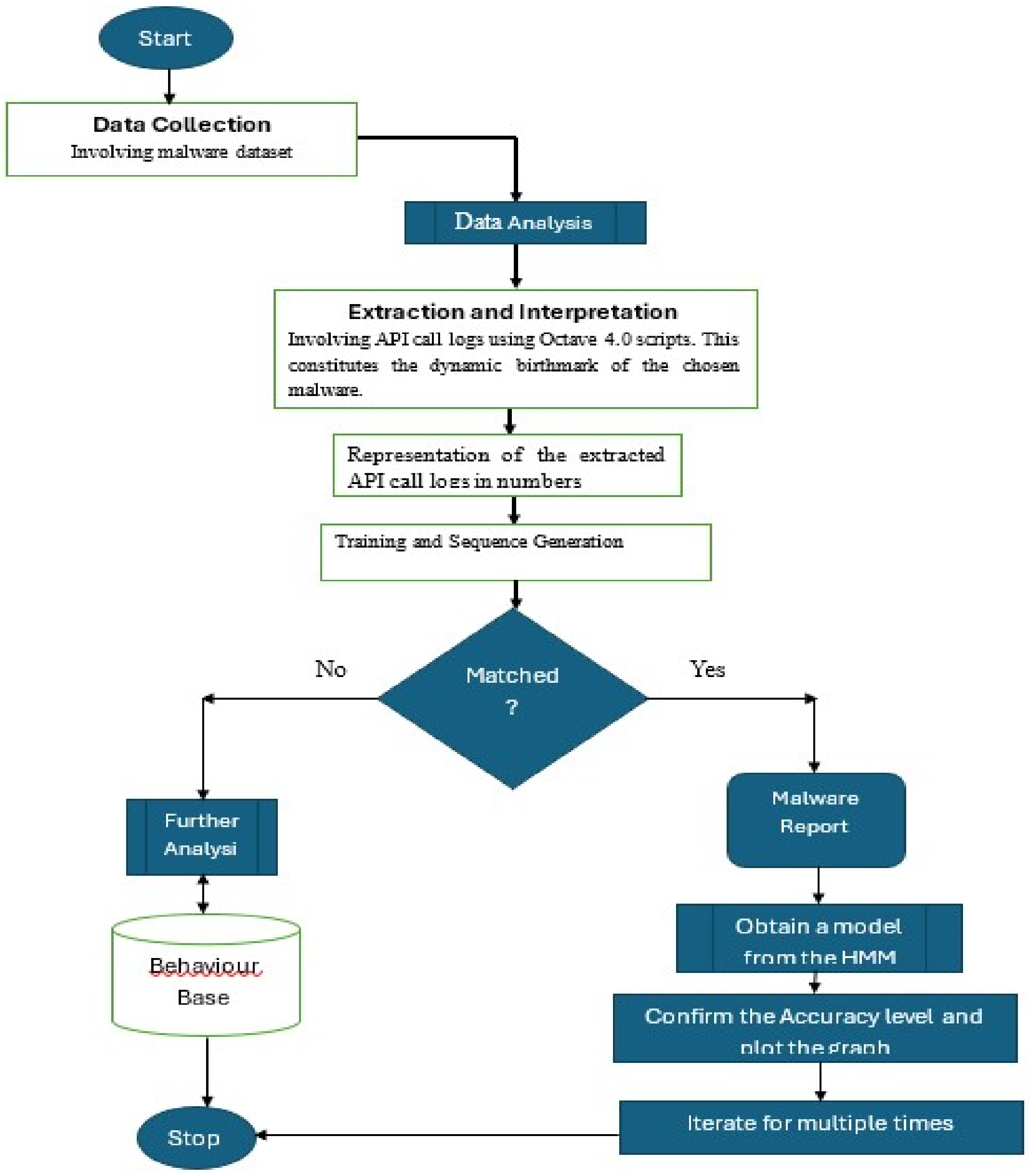
Start
Data Collection
Involving malware dataset
Data Analysis
Extraction and Interpretation
Involving API call logs using Octave 4.0 scripts. This constitutes the dynamic birthmark of the chosen malware.
Representation of the extracted API call logs in numbers
Training and Sequence Generation
Matched ?
No
Yes
Further Analysi
Behaviour Base
Malware Report
Obtain a model from the HMM
Confirm the Accuracy level and plot the graph
Iterate for multiple times
Stop

Figure 1: Flow of Process for the Methodology

3.1 Data Collection

The malware samples used in this research were collected from vxheaven.org (https://github.com/opsxcq/mirror-vxheaven.org). Interactive Disassembler (IDA) Pro V5.1 was used for disassembling the malware samples. The collection has many data files that include log data for various types of malware. These recovered log features were used to train a broad variety of models. Approximately 10 distinct malware families were found in the samples. About 409 files from 10 distinct malware families were used in addition to 89 cleanware, all totaling 498 files. This is shown in Table 1.

Table 1: Experimental Dataset

	Family	Number of files
Malware Files classified as families	Clagger	97
	Robknot	19
	Alureon	6
	Bambo	7
	Beovens	14
	Boxed	36
	Emerleox	97
	Looked	22
	Robzip	71
	Agobot	40
	Sub total	409
Cleanware files	Cleanware	89
	Total	498

3.2 Setup of A Controlled Virtual Environment

This was where the dataset collected was extracted and executed. This was to prevent the host machine from malware attack. Data were stored in the file system as binary code, and the files themselves were unprocessed executables. Unpacking the executables required a protected environment, or virtual machine (VM). A sandboxed environment was used to monitor the malware binaries as they were executed after automated unpacking of compressed executables within the protected environment.

The change of state in Application Programming Interface (API) function calls were the basis for the generation of behaviour based analysis report. Features reflecting behavioural patterns, for example, opening a file, locking and unlocking of a mutex and locating a registry key, were extracted with the help of analysis reports which were useful in embedding the malware behaviour into a high dimensional vector space. An HMM was implemented for detecting malware binaries and applied on different test data. Many results were generated when we analyzed the program by IDA Pro in the Sandbox. The one that was useful to us was the API calls and the API call frequency amongst others.

3.3. Feature Extraction

In this phase, the files were executed one by one in the VMWare virtual environment and execution logs were collected.

3.3.1 Static Analysis Script

Static malware analysis and detection techniques habitually fall short and fail to sense many new, unknown malware samples. This involved the following process:
Step 1: Save the pre-configured VM state by creating a snapshot.
Step 2: Revert to snapshot.
Step 3: Copy one binary file into the VM
Step 4: Execute the file using sandbox which will generate a trace report
Step 5: Kill the VM
Step 6: Copy the trace report off the VM
Step 7: Repeat steps 2 to 6 for all files

Step 8: Collate the trace report for further analysis.
All the trace reports from the malware files were collected and then used in the trace information to extract features.

3.3.2 Dynamic Analysis Script

For each malware execution, the following steps were taken automatically in the Dynamic Analysis Script:

Step 1. Revert the virtual machine to its original snapshot. It actually overwrites the current virtual machine state with the original virtual machine state, so that all the changes made by the malware during its execution were lost.
Step 2: Start the virtual machine to set up the virtual runtime environment.
Step 3: Copy the executable file from Host to VM.
Step 4: Run IDA Pro tool to monitor and trace the execution of malware. IDA Pro monitors the state changes in the system and generates a log file.
Step 5: In our experiments, we run each executable file for 30 seconds.
Step 6: Stop the virtual machine.
Step 7: Restart the virtual machine and then copy the generated log file from virtual machine to Host.

4. IMPLEMENTATION AND DISCUSSION OF RESULTS

The host machine that was used had an Intel(R) Pentium (R) CPU B960 @2.20Ghz processor, 4GB RAM, and Windows 10 Pro 64-bit operating system. The guest machine was a VMWare Virtual Machines and it consisted of a base memory of 2210MB, 2 GB RAM, 32 bit operating system and Windows Server 2008 operating system. The training and testing of the HMMs are done in the host machine whereas the API calls and opcode extraction are done in the guest machine.

The client operating system used was Microsoft Windows Server 2008 R2. A virtual machine environment was setup by installing and configuring VMware Server 2.0.2. Then there was a creation of a new virtual machine installing Windows Server 2008 as guest operating system and disabled networking. Before the start of dynamic analysis, a snapshot of the virtual machine was taken and then this snapshot was reverted to for every execution. In such a way, it was assured that the virtual machine was rehabilitated every time.

To automatically run the executables, online resource was resorted to via the url, https://malwr.com/ which carried out the executions in the VMware environment. This took 30 seconds per file. Then after which a report of the analysis with the screen shot was sent via email with a detailed description of the behavioural and other characteristics of the analysed malware. MATLAB script was written to extract the API call logs and a global list was created which contained the dynamic birthmarks of the various malware samples that were dynamically analysed. Then the individual file string was compared against the general list. This formed the feature vector matrix with which the HMM was trained for efficient detection of malware.

Baum Welch algorithm of the HMM was performed on different malware families using both static and dynamic analysis just to compare their performance. We then plotted the *receiver operating characteristic (ROC) curve* for each case and calculate the area under the curve (AUC). Figures 2 and 3 show the scattered plot for dynamic analysis and its ROC Curve. From the scattered plot the separation between the malware and benign scores were observed to determine the separation. It can be seen that there is a clear separation between the malware and benign scores.

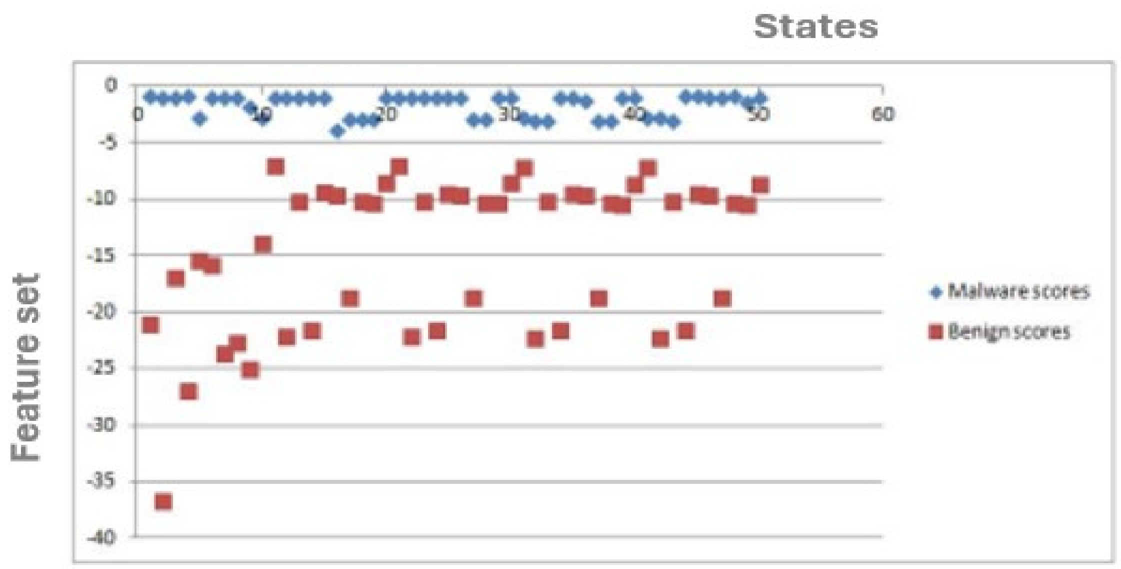

Figure 2: The Scattered Plot for Dynamic Analysis

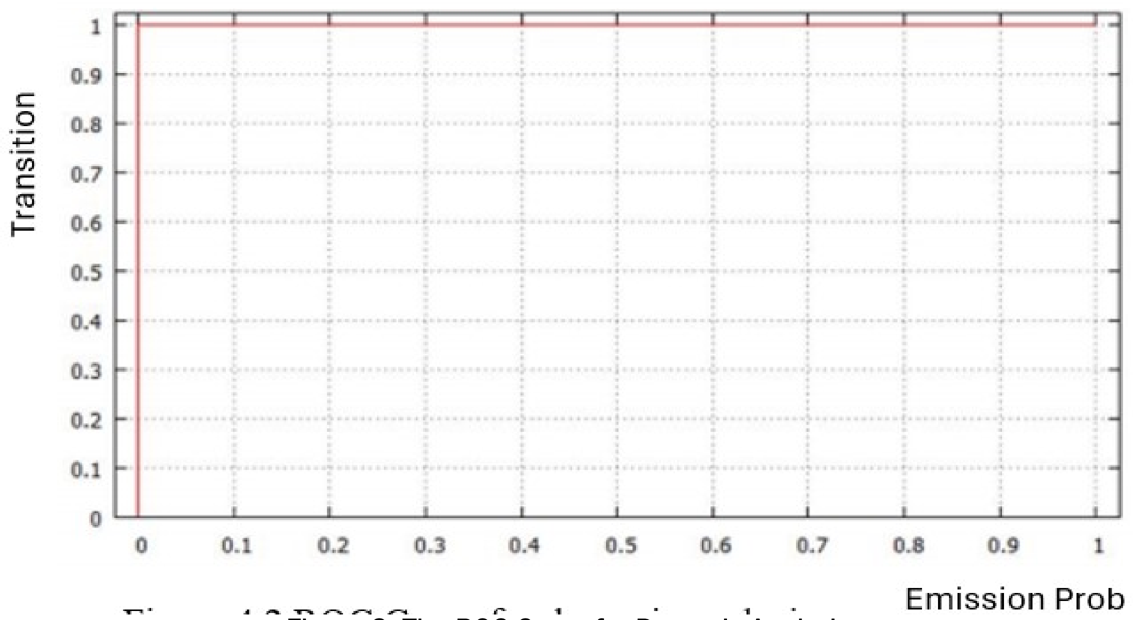

Figure 3: The ROC Curve for Dynamic Analysis

From Figure 3, AUC = 1 with dynamic analysis. This implies that it is a perfect classifier and does not have any false positives or false negatives. Hence, the dynamic analysis technique using Baum Welch algorithm of the HMM works effectively for detecting malware.

Figures 4 and 5 show the scattered plot for static analysis and its ROC Curve.

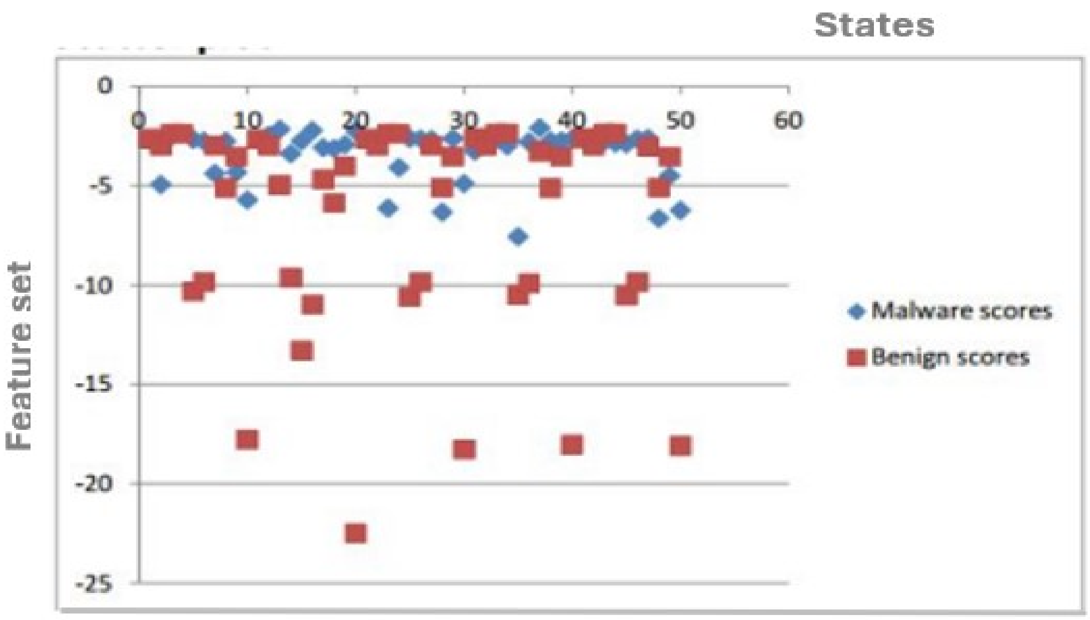

Figure 4: The Scattered Plot for Static Analysis

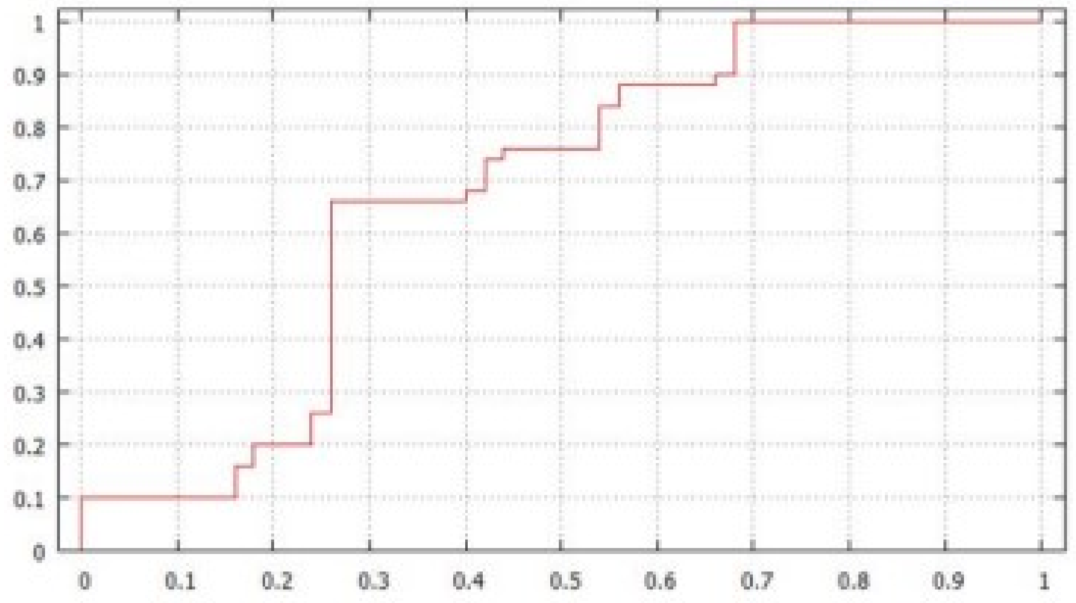

Figure 5: The ROC Curve for Dynamic Analysis

It can be seen from the scattered plot in Figure 4 that there is a lot of misclassification of the malware and benign files and there is no clear separation between the scores unlike in the dynamic analysis case. Also, from Figure 5, the AUC = 0.753, which indicates that it is a poor classifier. It can be succinctly seen that dynamic analysis using Baum Welch algorithm in HMM based detection has outperformed static analysis. This was due to the fact that the real behaviour of the malware at runtime cannot be seen in static analysis..

5. CONCLUSION

This study has presented a novel technique to reliably identify metamorphic malware. The technique relied on behavioural analysis and HMM technique to characterize the high-level behaviour of a malware executable. By abstracting from the concrete implementation of a malware, such as API calls and API frequency, the technique supported the identification of different mutations of a metamorphic malware using the dynamic analysis. This can be used as the basis for the implementation of a system that is resilient to a number of code transformation techniques.

REFERENCES

[1]. Anil K.D., Das S.K. (2021). NB2M–Mechanism for Magnifying Micro Level Bugs for Secure Software System. 2021;3:26–33.

[2]. Annachhatre,C., Austin, T. H. and Stamp, M. (2014)."Hidden Markov models for malware classification," Journal in Computer Virology Hack Tech:Springer, 2014.

[3]. Aycock,J.(2006)"Computer Viruses and Malware," Advances In Information Security:Springer, 2006

[4]. Falliere, N.,Murchu,L. O. anien, E. (2010). W32.Stuxnet Dossier (Version 1.3). Symantec Security Response Blog.

[5]. Alshamrani, A. Myneni, S., Chowdhary, A., and Huang, D. (2019). A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities IEEE Communications Surveys & Tutorials, 21 (2) (2019), pp. 1851-1877

[6]. Aslan, O., Yilmaz, A. A. (2021). A new malware classification framework based on deep learning algorithms IEEE Access, 9 (2021), pp. 87936-87951

[7]. Global Threat Landscape (GTL) Report 2H (2023). What is heuristic analysis? A Blog retrieved on 11 July, 2024 from: https://www.fortinet.com/resources/cyberglossary/heuristic-analysis

[8]. Robiah, Y. & Selamat, Siti Rahayu & Masud, Zaki & Sahib, Shahrin & Abdollah, Mohd & Ramly, Marliza. (2009). A New Generic Taxonomy on Hybrid Malware Detection Technique. International Journal of Computer Science and Information Security. 5.

[9]. Erickson, J. (2008) . Hacking: The Art of Exploitation. 2nd Edition.No Starch Press, 2008.

[10]. One, A. (1996). Smashing the stack for fun and profit. Phrack, 7(49), 1996.

[11]. Shu, L., Dong, S., Su, H. and Huang, J. (2023). ''Android malware detection methods based on convolutional neural network: A survey,'' IEEE Trans. Emerg. Topics Comput. Intell., vol. 7, no. 5, pp. 1330–1350, Oct. 2023, doi: 10.1109/TETCI.2023.3281833.

[12]. Koushki, M. M., Abualhaol, I., Raju, A. D., Zhou, Y., Giagone, R. S. and Shengqiang, H. (2022). ''OnbuildingmachinelearningpipelinesforAndroid malware detection: A procedural survey of practices, challenges and opportunities,'' Cybersecurity, vol. 5, no. 1, pp. 1–37, Dec. 2022, doi: 10.1186/s42400-022-00119-8.

[13]. Meijin, L., Zhiyang, F., Junfeng, W., Luyu, C., Qi, Z., Tao, Y., Yinwei, W. and Jiaxuan, G. (2022). ''A systematic overview of Android malware detection,'' Appl. Artif. Intell., Vol. 36, No. 1, pp. 497–524, Dec. 2022, doi: 10.1080/08839514.2021.2007327.

[14]. Shabir A. G. and Sabahat, N. (2020). ''Areviewofhybridmalwaredetectiontechniques in Android,'' in Proc. IEEE 23rd Int. Multitopic Conf. (INMIC), Nov. 2020, pp. 1–6, doi: 10.1109/INMIC50486.2020.9318117.

[15]. Myles, G. & Collberg, C. (2004). Detecting Software Theft via Whole Program Path Birthmarks. 3225. 404-415. 10.1007/978-3-540-30144-8_34.

[16]. Lu, B., Liu, F., Ge, X., Liu, B. and Luo, X. (2007). A Software Birthmark Based on Dynamic Opcode n-gram. 37-44. 10.1109/ICOSC.2007.4338330.

[17]. Anderson, B., Quist, D., Neil, J., Storlie, C. & Lane, T. (2011). Graph-based malware detection using dynamic analysis. Journal in Computer Virology. 7. 247-258. 10.1007/s11416-011-0152-x.

[18]. Tamada, H., Okamoto, K., Nakamura, M., Monden, A. & Matsumoto, K. (2007). Design and Evaluation of Dynamic Software Birthmarks Based on API Calls. Information Science Technical Report, NAIST-IS-TR2007011, ISSN 0919-9527, Graduate School of Information Science, Nara Institute of Science and Technology.

[19]. Singh, N., Kasyap, H., & Somanath, T. (2020). Collaborative Learning Based Effective Malware Detection System. 10.1007/978-3-030-65965-3_13.

[20]. Thambi-Rajah, Tr and Jahankhani, H.. (2021). The Role of Deep Neural Network in the Detection of Malware and APTs. 10.1007/978-3-030-87166-6_7.

[21]. Chandak, A., Lee, W. and Stamp, M. (2021). A Comparison of Word2Vec, HMM2Vec, and PCA2Vec for Malware Classification.

[22]. Azeem M, Khan D, Iftikhar S, Bawazeer S, Alzahrani M.(2023).Analyzing and comparing the effectiveness of malware detection: A study of machine learning approaches. Heliyon. 2023 Dec 12;10(1):e23574. doi: 10.1016/j.heliyon.2023.e23574. PMID: 38187275; PMCID: PMC10770453.