

INTRUSION DETECTION SYSTEM FOR FILE-LESS MALWARE

Submitted in partial fulfillment of the requirements for the
award of
Bachelor of Technology degree in Information Technology
By

SAKTHI VINAYAK (Reg.No - 40120077)

REEYA B VARGHESE (Reg.No - 40120074)

THIRISHA A (Reg.No - 40120098)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

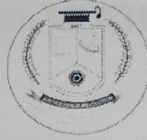
**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Category-1 University by UGC

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by
AICTE**

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600119**

APRIL - 2024



SATHYABAMA

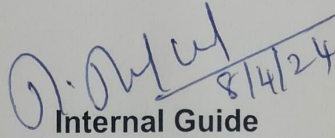
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

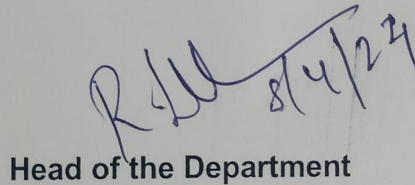
www.sathyabama.ac.in

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **REEYA B VARGHESE**(Reg.No - 40120074), **SAKTHI VINAYAK** (Reg.No - 40120077) and **THIRISHA A**(Reg.No - 40120098) who carried out the Project Phase-2 entitled **"INTRUSION DETECTION SYSTEM FOR FILE-LESS MALWARE"** under my supervision from January 2024 to April 2024.


Internal Guide

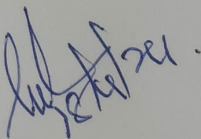
Ms.S. SAGEENGRANA M.E.,


Head of the Department

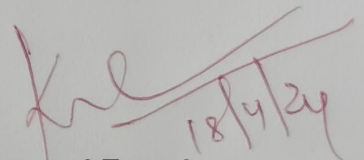
Dr.R. SUBHASHINI, M.E., Ph.D.

Submitted for Viva voce Examination held on

18/4/2024



Internal Examiner



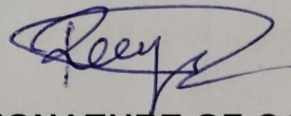
External Examiner

DECLARATION

I, **REEYA B VARGHESE** (Reg.No - 40120074), hereby declare that the Project Phase-2 Report entitled "**INTRUSION DETECTION SYSTEM FOR FILE-LESS MALWARE**" done by me under the guidance of **MS. S.Sageengrana, M.E.**, is submitted in partial fulfillment of the requirements for the award of Bachelor of Technology degree in **Information Technology**.

DATE: 18/04/2024

PLACE: Chennai



SIGNATURE OF CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D., Dean**, School of Computing, **Dr. R.Subhashini M.E., Ph.D.**, Head of the Department of Information technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. S.Sageengrana, M.E.**, for her valuable guidance, suggestions and constant encouragement paved the way for the successful completion of my phase-2 project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

This work addresses the problem of file-less malware using event logs and memory dump data. Fileless malware differs from traditional malware as it doesn't need to install malicious software to infect the victim's machine. Instead, it takes advantage of existing vulnerabilities on your machine. In this work, We are developing a comprehensive method to detect the presence of file-less malware in the system using event log data and memory dump data. Additionally, we perform experimental analysis using various machine learning and deep learning algorithms and find the optimal methodology which gives the highest accuracy. The event logs dataset and the memory dump dataset undergo various processes to remove unwanted elements in the data like outliers, and missing values, and transform it into a standard format that is compatible with chosen ML methods. We have chosen two ML models to be trained on the event log data namely Multinomial naive Bayes and random forest classifier. The multinomial naive Bayes model is trained on the train data and produces an accuracy of 99.39% on test data. This study uses Deep neural networks (DNN) for training the memory dump dataset. This model produced an accuracy of 99% but the loss was high. To reduce the loss and prevent the model from overfitting we have added 3 dropout layers with 20%, 30% and 40% dropout rate to each layer. This has significantly reduced the loss rate and produced the final accuracy of 99.93% compacting overfitting. We have compared the results of the ML and DL models used for event logs dataset and memory dump dataset using various metrics like confusion matrix, F1- score, Precision, recall and Accuracy score. This work introduces two pipelines that use machine learning and deep learning classifiers to detect abnormal activities signaling malware in a system. Our utilization of deep neural networks for malware detection has demonstrated superior performance compared to existing models outlined in the literature. Integrating this technology into existing Intrusion Detection Systems offers an opportunity to strengthen overall protection, providing a comprehensive defense mechanism for critical systems and data.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	IV
	LIST OF FIGURES	VIII
	LIST OF ABBREVIATIONS	X
1	INTRODUCTION	1
	1.1 Background and Motivation	3
	1.2 Problem Statement	3
	1.3 Objectives and Scope	3
	1.4 Applications of Intrusion Detection System	4
	1.5 Organization of Thesis	6
2	LITERATURE SURVEY	7
	2.1 Overview of Existing Research and Literature	7
	2.2 Inferences from Literature Survey	13
	2.3 Open Problems in Existing System	14
3	REQUIREMENTS ANALYSIS	15
	3.1 Feasibility Studies of the Project	15
	3.1.1 Economic Feasibility	15
	3.1.2 Technical Feasibility	16

3.2	Requirements Specification	16
3.2.1	Hardware Requirements	16
3.2.2	Software Requirements	16
3.2.3	Description of the Requirements	17
3.2.4	Algorithms Used	18
3.3	System Use Case	21
3.3.1	System Description	21
3.3.2	System Use Case Scenario	21
4	DESCRIPTION OF PROPOSED SYSTEM	22
4.1	Description of Research Approach and Methods	22
4.1.1	Dataset	23
4.1.2	Pre-processing	23
4.1.3	Machine Learning Models	25
4.1.4	Training and Testing	27
4.1.5	Evaluation Metrics	27
4.2	Architecture of Proposed System	28
4.3	Description of Software for Implementation and Testing Plan of the Proposed System	30
4.4	Project Management Plan	31
4.5	Financial Report on Estimated Costing	32
4.6	Software to Operations Plan	33

5	IMPLEMENTATION DETAILS	35
	5.1 Development and Deployment Setup	35
	5.2 Testing	36
6	OUTCOMES AND DISCUSSIONS	37
	6.1 Introduction	37
	6.2 IEEE Standards Followed in the Project	38
	6.3 Constraints	38
	6.4 Tradeoff in the Project	39
7	RESULTS AND DISCUSSIONS	41
	7.1 Presentation of Findings	41
	7.2 Data Analysis and Interpretation	41
8	CONCLUSION	51
	8.1 Conclusion	51
	8.2 Recommendations for Future Research	51
	REFERENCES	53
	APPENDIX	56
	A. SOURCE CODE	56
	B. SCREENSHOTS	61
	C. RESEARCH PAPER	68

LIST OF FIGURES

CHAPTER NO.	FIGURE NAME	PAGE NO.
3.1	Structure of Random Forest Algorithm	19
3.2	Structure of deep neural networks	20
4.1	Architecture of the proposed system	22
4.2	Selecting the best value of alpha for Multinomial Naive Bayes using hyper-parameter tuning	25
4.3	Structure of the deep neural network	26
4.4	Flow diagram of the proposed system	28
7.2.1	Memory Dump Dataset	41
7.2.2	Event logs Dataset	42
7.2.3	ROC curve of Multinomial NB and Random forest	42
7.2.4	Confusion Matrix of Multinomial NB Model	43
7.2.5	Unit testing of Multinomial Naive Bayes Model	43
7.2.6	Memory dump data category	44
7.2.7	Creation of DNN Model	45
7.2.8	Total Loss vs Total Validation Loss of DNN Model	46
7.2.9	Total Accuracy vs Total Validation Accuracy of LRCN Model	46
7.2.10	Accuracy of LRCN Model	47
7.2.11	Picture showing the collection event log data in real time	47
7.2.12	The flask application page where the user can view the detected activities	48

7.2.13	The Alert notification appearing to the user	48
7.2.14	The Page to upload the input file which contains the behavioral patterns of the malware	49
7.2.15	Prediction Page which shows the result that the model detected	49
7.2.16	Email alert notification sent to the Admin user	50
B.1	Applying PCA dimensionality reduction	61
B.2	Converting Log data into TF-IDF vector	61
B.3	Importing the packages in Flask	62
B.4	Using NLP on the event logs	62
B.5	Real time extraction of Event logs	63
B.6	Deploying DNN model	63
B.7	Creating the DNN model	64
B.8	Training the Model	64
B.9	Plotting Accuracy Curves of DNN Model	65
B.10	Plotting Loss Curves of DNN Model	65
B.11	Plotting ROC curve	66
B.12	Plotting the Model	66
B.13	Training the multinomial Naive Bayes model	67
B.14	Unit Testing of Event Logs model	67

ABBREVIATIONS

AVT	Advanced Volatile Threat
APT	Advanced Persistent Threat
WMI	Windows Management Instrumentation
DDOS	Distributed Denial Of Service
MEME	Microsoft Enhanced Mitigation Experience
DLL	Dynamic Link Library
VBA	Visual Basics For Applications
ASLR	Address Space Layout Randomization
DEP	Data Execution Prevention
SMTP	Simple Mail Transfer Protocol
PCA	Principal Component Analysis
TFIDF	Term Frequency Inverse Document Frequency

CHAPTER 1

INTRODUCTION

Fileless malware is classified as AVT (Advanced Volatile Threat), a type of malware that is designed to leave little to no footprint on a system, making it difficult to detect. This type of malware is specifically designed to exfiltrate data from the compromised system. The malware operates within the volatile memory, Random Access Memory (RAM), by disguising itself as a legitimate program while engaging in harmful actions. Given its exclusive operation within the volatile memory, the functionality of this particular form of malware ceases upon the user's system shutdown. In this particular situation, the malware may appear to possess a lesser degree of threat compared to an Advanced Persistent Threat (APT), which is specifically engineered to maintain a prolonged presence within a network while evading detection. The APT malware was formerly regarded as the most perilous among several types of malwares. However, it has been surpassed by AVT, which has become widely dreaded. This is mostly due to the fact that threat actors have begun employing persistence techniques to evade detection and remain within the system even after the user has shut it down.

Persistence is accomplished by the malware through the manipulation of the operating system's registry. The Registry is an integral component of the operating system responsible for managing and storing configuration settings. This is achieved by injecting malicious code directly into the Registry using a dropper program. The malware possesses the capability to generate or alter registry keys in a manner that initiates the malware's execution upon each instance of the operating system being powered on and operational. In this particular scenario, the presence of a malicious file is absent, as the harmful code is instead immediately inserted into a legitimate application. Consequently, the virus is able to evade detection for an extended duration.

The fileless threat typically initiates its lifecycle with the delivery stage, during which the hacker attempts to transmit the malicious script to the target machine by diverse methodologies. Phishing, a widely employed tactic observed in numerous instances of malware infections, enjoys significant popularity and prevalence. The perpetrator sends

an infected email to the target individual, containing harmful scripts, or employs deceptive tactics to entice the user into accessing a malicious website. This website then proceeds to download an adverse JavaScript code or an infected Microsoft Office document, which utilizes social engineering techniques to deceive the user into enabling malicious macros. Upon infiltrating the system of the targeted individual, the malware endeavors to modify the registry keys in order to attain persistence. Additionally, it creates a connection with the command-and-control server for the purpose of downloading the malicious script. The payload operates within a legitimate tool, such as PowerShell, and executes pre-programmed malicious actions on the targeted machine, such as the unauthorized extraction of data.

Some of the recorded fileless attack techniques are the presence of a malicious script within the Windows Management Instrumentation subscription (WMI), the direct inclusion of a malicious script as a command line parameter in PowerShell, the storage of a malicious script within the registry, and/or the operating system scheduler task, and the execution of the said script by the operating system scheduler, as well as the extraction and direct execution of a malicious executable in memory without being saved on disc via the use of a .Net reflection technique, among other methods, is observed in academic research.

1.1 BACKGROUND AND MOTIVATION

The efficacy of traditional anti-virus software in detecting and mitigating fileless threats has been called into question. These software programs are designed to scan and monitor the hard disc of a system, rendering them ineffective in identifying fileless malware. This is primarily due to the fact that fileless malware does not leave any discernible trace or footprint within the system, making it impervious to detection by signature-based antivirus software. In addition, fileless malware capitalizes on the utilization of established, authorized programs within the operating system that typically evade detection from anti-virus software. This project introduces two pipelines that use machine learning and deep learning classifiers to detect abnormal activities signaling malware in a system. Our utilization of deep neural networks for malware detection has demonstrated superior performance compared to existing models outlined in the literature. Integrating this technology into existing IDS offers an opportunity to strengthen overall protection, providing a comprehensive defense mechanism for critical systems and data.

1.2 PROBLEM STATEMENT

We propose a novel method which makes use of windows event logs and memory dump data of windows processes to detect the presence of file-less malware. Event logs are structured files which contain information about the events or processes running in the system. Memory dump is a memory forensic technique that can extract all the information from the RAM of our computer and make a copy out of it in our local disk. This study uses machine learning algorithms to capture these features and detect the presence of the malware in the system. Through thorough experimental analysis, the proposed methodology achieved high accuracy of 99.93% by combining event logs and memory dump data which have also significantly contributed to the reduction in the false positive rates.

1.3 OBJECTIVE AND SCOPE

Objective

This project tackles the growing threat of fileless malware, malicious software that operates within a system's memory without leaving traditional file-based traces. Existing detection methods often struggle against such adversaries. To address this

challenge, the project proposes a novel approach that combines data fusion and machine learning. The system analyzes both event logs records of system activities and memory dumps snapshots of the system's memory to gain a comprehensive view of system behavior. Machine learning algorithms are then employed to automatically learn patterns and identify subtle indicators of malicious activity within this combined data. To improve detection accuracy and efficiency, the project utilizes feature engineering techniques like Principal Component Analysis (PCA) to optimize the data for the machine learning models. Additionally, hyperparameter tuning is performed to specifically tailor the models to the unique characteristics of the combined event log and memory dump data. Through this innovative approach, the project aims to achieve a highly accurate and automated system for detecting fileless malware. This can significantly enhance system security by proactively identifying and mitigating these stealthy threats that might evade traditional methods.

Scope

The real-world applications of this project extend to various areas where robust fileless malware detection is crucial. Firstly, it can be integrated into endpoint security software, providing real-time protection for individual user devices like laptops and desktops. This bolsters individual user security and reduces the risk of malware infections on personal machines. Secondly, the project's findings can be implemented in network security systems. By analyzing network traffic alongside event logs and memory dumps of connected devices, these systems can identify potential malware infiltration attempts across the network, offering an additional layer of protection for entire organizations. Finally, this method has applications in cloud security platforms. Cloud environments can leverage this approach to monitor virtual machines and containerized applications for signs of fileless malware activity. By incorporating this project's methodology, organizations across various sectors can significantly strengthen their overall security posture and proactively combat the growing threat of fileless malware attacks.

1.4 APPLICATIONS OF INTRUSION DETECTION SYSTEM

Intrusion Detection Systems(IDS) play a crucial role in detecting and preventing malware attacks within computer networks. There are some key applications of IDS specifically for malware detection:

1. **Signature-Based Detection:** IDS can use signature-based detection to identify known malware by comparing network traffic or system activity against a database of predefined signatures. When a match is found, the IDS raises an alert, enabling administrators to take appropriate action such as quarantining or removing the infected file.
2. **Anomaly-Based Detection:** Anomaly-based detection techniques are employed by IDS to identify deviations from normal network or system behavior. Malware often exhibits anomalous behavior such as excessive network traffic, unusual file access patterns, or unexpected system modifications. Anomaly-based IDS can detect these deviations and flag them as potential malware infections.
3. **Behavioral Analysis:** IDS can analyze the behavior of software programs and processes to detect suspicious or malicious activity indicative of malware. This can include monitoring processes for actions such as attempts to modify system files, establish unauthorized network connections, or execute potentially harmful commands. By identifying such behaviors, IDS can detect malware that may not be detected by signature-based methods.
4. **Heuristic Analysis:** Heuristic analysis involves identifying potentially malicious patterns or characteristics in network traffic or system activity that may indicate the presence of malware. IDS employing heuristic analysis techniques can detect polymorphic malware variants or previously unseen malware by analyzing code structures, execution paths, or other behavioral attributes.
5. **Traffic Inspection:** IDS can inspect incoming and outgoing network traffic to identify patterns associated with malware propagation, such as suspicious URLs, command and control communications, or malicious payloads. By analyzing network packets in real-time, IDS can detect and block malware before it infiltrates the network or compromises systems.
6. **Integration with Anti-Malware Solutions:** IDS can complement traditional anti-malware solutions by providing an additional layer of defense against sophisticated threats. By integrating with antivirus software or endpoint protection platforms, IDS can enhance malware detection capabilities and provide comprehensive coverage across network endpoints.

Incident Response and Forensics: IDS plays a vital role in incident response and forensic investigations following a malware attack. By logging and analyzing security events, IDS can provide valuable insights into the timeline of an attack,

the methods used by the malware, and the extent of the compromise. This information is essential for remediation efforts and strengthening defenses against future attacks.

1.5 ORGANIZATION OF THESIS

The organization includes an introduction that provides an overview of the research problem, objectives, and significance. The literature review section discusses the existing research on File-less malware using KNN, CNN models and the Limitations of the existing system. The methodology section describes the data collection process, pre-processing, and the machine learning and deep learning models used. The results section presents the findings of the study, including the accuracy of the models. The discussion interprets the results, discusses the limitations of the study, and suggests research directions. Finally, the conclusion section summarizes the main findings and contributions of the study. The thesis also includes an abstract, acknowledgments, and references.

CHAPTER 2

LITERATURE SURVEY

Several studies on file-less malware detection have been conducted in recent years. This section summarizes all of the previous work on detection of file-less malware.

2.1 OVERVIEW OF EXISTING RESEARCH AND LITERATURE

A. Afreen et al [1] provides a view regarding threats and emphasizes that there is so much work yet to be done in this field of file-less malware. It is extremely challenging to capture file-less malware since there are so little file-based artifacts that can be used to identify the threat. Detecting exfiltration and malicious actions can be challenging due to the usage of legitimate and multipurpose tools like PS, Macros, and WMI in these attacks. A multilevel approach is required to detect and halt file-less assaults over the course of the whole file-less threat lifecycle. With the widespread availability of multipurpose technologies, fileless malware will become much more deadly. The goal of this study is to examine the factors that contribute to the proliferation of file-less malware and to the difficulty with which it may be detected and protected. Malware researchers can use the discussed issues and recommended techniques to improve detection systems and sample collecting for file-less malware.

Sudhakar et al [2] presents a novel investigative framework that aims to facilitate the breakdown of analysis into a series of simple processes. The objective is to enhance the efficiency of detecting and analyzing the underlying causes of these attacks. Fileless malware can be identified by a range of techniques, such as manual inspection methods, system behavior monitoring, rule-based detection, and analysis of attack behavior. The paper stresses the importance of monitoring critical aspects, including remote command execution, elevation of standard user privileges to administrative privileges, and the execution of harmful programs in the primary memory. The suggested process model for incident response is specifically tailored to address occurrences linked to malware, encompassing both real-time attacks and post-attack scenarios. The paper proposes an initial five-phase procedure that entails the identification and examination of the incident, followed by the study of evidence and artifacts. The regular training of individual incident handlers is necessary in order to maintain their preparedness for emerging security risks and the associated

technologies leveraged by more sophisticated malicious programs, such as fileless malware, botnets, Advanced Persistent risks (APT), and Distributed Denial of Service (DDoS) attacks. The research paper additionally proposes that in order to thwart fileless attacks, the detection system should take into account various attack vectors and sources of evidence. It should also utilize machine learning algorithms to examine command lines, scrutinize internet connections, observe process behavior, and safeguard the memory space of the active process.

B.N.Sanjay et al [3] conducted in "An Approach to Detect Fileless Malware and Defend its Evasive Mechanisms" categorizes fileless malware into two primary classifications: This discussion focuses on two types of malwares: RAM-resident Fileless Malware and Script-Based Fileless Malware. RAM-resident malware exhibits enhanced stealthiness due to the fact that antivirus solutions primarily scrutinize newly initiated processes, rendering them less likely to trigger suspicion. The script-based fileless malware involves the exploitation of vulnerabilities present in Microsoft Office, several Windows apps, and Windows PowerShell. The research paper necessitates the implementation of a comprehensive and cohesive strategy that encompasses all stages of the threat lifecycle for effective mitigation of fileless malware threats. The mitigation of fileless malware entails the identification and monitoring of Windows PowerShell, white-listed apps, scripts, and settings as a means of safeguarding against fileless attacks. It is recommended to regularly monitor advisories pertaining to the latest updates addressing vulnerabilities in the program and to conduct frequent security checks. Regularly apply operating system patches and upgrades, while utilizing additional tools like the Microsoft Enhanced Mitigation Experience Toolkit (MEME) to identify any deviations from a predetermined baseline. The implementation of a policy to restrict the usage of PowerShell involves the imposition of limited access to execute scripts via the Windows Group Policy. The objective is to observe and track operations within Windows PowerShell or similar engines that interact with thread data, with the purpose of obtaining a comprehensive understanding of any potentially harmful actions.

P.Borana et al [4] Presents a tool to assist in the detection of fileless malware. The tool is developed using Java and makes use of the Windows operating system's runtime system and process information. The purpose of the tool's numerous modules is to collect process information and present it to users so that system administrators can make informed decisions based on the data. The application possesses the following features: Process Monitoring, Network Monitoring, Detection of processes operating

with high priority, Process Hierarchy, DLL(Dynamic Link Library), and command-line arguments information. The application hosts a module that transmits the information of the system's process dump to an online database, such as Virus Total, to determine whether or not the application is malicious.

O. Khalid et al [5] Describes the lifecycle and infection chain of fileless malware and proposes detection techniques based on machine learning and memory forensics. This paper briefs about the methodology of feature extraction and analysis of memory dumps of the virtual machine in which both malicious and non-malicious samples are executed. Following the sequential execution of malicious and benign samples in the virtual machine, a snapshot of the virtual machine is captured. Volatility, a memory forensics tool, extracts the necessary features for classification from the snapshot and memory dump of an infected system. The features are meticulously chosen to aid in the training of machine learning-based classification models that classify instances as malicious or not. This paper utilized multiple machine learning algorithms to compare and evaluate the models with the highest accuracy score. Among these algorithms, Random Forest outperformed the competition with an accuracy score of 93.33%.

S. Mele Pottaraikkal et al [6] presents an extended version of memory image analysis, which is one of the most widely used techniques for detecting file-less malware. Memory image analysis has thus far focused on obtaining a memory snapshot of the infected system at a specific timestamp to detect the presence of fileless malware. This paper proposes an MMI (multiple memory-image) analysis in which the memory image is retrieved at various timestamps and is used collectively to detect the presence of fileless malware. This technique has proven effective in achieving accuracy and significantly reducing the number of false positives. Collecting malware samples of several types and retrieving the VMEM(Virtual memory) of the infected system constitutes the research. These memory images are then supplied to a Python script that performs heuristics, and anomaly detection based on process dumps, and labels each instance as either positive or negative. This method has been proven to reduce the number of false positives and has successfully detected the hidden fileless malware samples used during the analysis with the aid of the MMI technique.

S.Zhang et al [7] explains the research experiment that is similar to the one presented in reference [5]. The experiment involves executing a set of static samples, some of which are malicious, and others are non-malicious. Subsequently, the memory dump

of the system is obtained to facilitate a detailed analysis of the process flow and memory usage. The objective of this analysis is to accurately distinguish and categorize the observed events as either malicious or non-malicious. This study utilizes binary fragments of a process, as it does not load all the contents of the PE files into the RAM. The study involves the conversion of retrieved memory PE files into binary material, followed by preprocessing and the selection of an appropriate deep-learning model. In this research investigation, the utilization of LSTM (Long short-term memory) and CNN1D (Convolutional Neural Networks- 1 dimensional) is employed to get a heightened level of accuracy in the detection of malicious code. The accuracy of the model was 97.48%, with the PE file header fragment of 4096 bytes achieving the maximum accuracy.

M.H.Tsai et al [8] mentions that Fileless attacks commonly occur when malicious actors exploit legitimate and built-in programs, such as living off-the-land binaries and PowerShell commands. When confronted with malicious PowerShell scripts, the threat actors employ several techniques of obfuscation and encryption to impede the analysis procedure and masquerade as a genuine script, hence rendering detection challenging. This study introduces a novel hybrid framework that combines deep learning techniques with behavior profiling. The PowerDP methodology comprises two distinct steps. The initial phase focuses on the de-obfuscation of malicious PowerShell instructions, while the subsequent phase involves behavioral profiling, wherein both phases utilize the multi-label classifier. The framework has been evaluated and has demonstrated an impressive accuracy rate of 99.82%.

S.Saad et al [9] provides an overview of the ongoing evolution and increasing sophistication of malware attacks. Threat actors employ diverse strategies to compromise computer systems, including the exploitation of inherent functionalities inside programming languages and dynamic link libraries (DLLs), which are sometimes referred to as off-the-shelf software libraries. Fileless malware is a type of malicious software that utilizes the exploit mechanism stated above. Saad et al. have created fileless malware using the inherent functionalities of JavaScript and HTML5 in order to investigate the life cycle and attack patterns of fileless malware in web apps. The occurrence of a malware infection scenario commences with the establishment of a WebSocket connection when the users visit a malicious web server. Subsequently, the WebSocket channel is utilized to transmit a harmful payload by establishing a connection with a command and control (C&C) server. In addition, the payload may

potentially be transmitted using HTML injection on online applications that are susceptible to such vulnerabilities. The aforementioned virus demonstrated the ability to effectively elude detection by several existing tools for detection, such as VirusTotal, Sucuri, and Rescan.Pro, SiteGuarding, WebInspector, Quettra, and AI-Bolit. The malware successfully evaded detection by employing a direct injection technique to introduce obfuscated malicious code across the web socket. By leveraging a benign and dynamic injection approach, both static and dynamic analysis methods were unable to identify the presence of the malware.

R.Bearden et al [10] presents a method for classifying whether the macros in MS Office documents are malicious or not. The researchers utilize p-code, an assembly language code native to Visual Basics for Applications (VBA) that is typically generated when a VBA script is compiled to enable the code to operate on multiple processor types. They implemented the Term Frequency Document Frequency (TFIDF) in order to standardize the relationship between a term's frequency and its relative frequency in the document. The DF selection method and the Fisher Selection method were employed to choose significant and efficient model features. The K-Nearest Neighbors algorithm was used to train the model with a training dataset of 90% and a testing dataset of 10%, achieving 96.3% accuracy. The most difficult aspect of conducting the research was locating sufficient samples for testing; the team was only able to locate MS Office Word files for malicious samples and MS Excel files for benign samples. This is due to the fact that malicious documents are sent to organizations for testing and analysis, whereas benign documents are difficult to obtain due to the presence of confidential corporate information that could potentially cause damage to the corporation.

S.Kim et al [11] emphasizes the growing prevalence of VBA macro attacks and the necessity for additional study into methods of detecting obfuscation in VBA macros. Preprocessing VBA macros, getting rid of unnecessary ones, and labeling training samples are all part of this paper's approach to identifying obfuscation. The writers get VBA macro codes out of MS Office documents with the help of oletools, a free Python library.

Information on the correlation between malice and obfuscation is provided by dividing the dataset into benign and malicious subsets. In order to evaluate the efficacy of the approach for malicious files, 2,537 VBA-enabled MS Office documents were collected for the experiment. Malicious files were found to be smaller on average, and the system was able to efficiently detect obfuscation tactics. The team has categorized four unique VBA obfuscation approaches, and features have been chosen for the detection process based on this classification. The study employed five different machine learning algorithms, all of which were measured for their effectiveness. The experiment was shown to have an F2 score improvement of over 23% when compared to similar investigations

A. More et al [12] presents a novel simulation framework for analyzing and detecting fileless malware which is the primary focus of this research. The framework's ability to detect and analyze fileless malware is demonstrated by its thorough overview of the attack scenario, execution mode, logging entry, static code analysis, and dynamic behavior analysis. The investigation process for finding and understanding fileless malware must begin on the ground floor. There are three distinct types of analysis: static analysis, dynamic analysis, and log analysis. Software updates, antivirus and anti-malware programs, behavioral-based protection, firewalls, network segmentation, limiting administrator rights, and frequent security awareness training are all methods for protecting against fileless malware. Insights gained from these methods improve security professionals' capacity to counteract fileless malware. By enabling auditing and logging, you can track everything that happens on your system, spot any questionable activity, and collect evidence for a forensic investigation. Protecting RAM from fileless malware requires the deployment of memory security mechanisms such as Address Space Layout Randomization (ASLR) and Data Execution Prevention (DEP). This simulation methodology is a major step forward in our ability to counter fileless malware and is likely to be a valuable resource for both security experts and academics.

M. U. Rana et al [13] investigates several techniques employed by malware authors to establish persistence within a dynamic Windows operating system environment, with specific emphasis on the solar wind attack. Various techniques, including Registry Persistence, Task Scheduler Persistence, Startup, Basic Input/Output System (BIOS) endurance, and PowerShell, enable malicious actors to exercise command and control at their discretion. The objective of this study is to make a scholarly contribution to

contemporary methods of identifying malware, with a specific focus on its application in the realms of cyber espionage, e-crime for financial motives, and hacktivism disruption. The executions are performed using Microsoft Visual Studio, and the experimental findings are presented through C# code. The experimental findings indicate that the persistence of malware plays a critical role in the activation of viruses. The findings were categorized into three distinct sections: 40% of the results were attributed to the utilization of startup location, 60% were attributed to the implementation of the task scheduler approach, and 80% were attributed to the utilization of registry strategies for achieving persistence. Based on the findings, it can be deduced that the utilization of persistence through registry proves to be significantly more effective than alternative methods for obfuscating malware within the system. confirm that you have the correct template for your paper size. This template has been tailored for output on the US-letter paper size. If you are using A4-sized paper, please close this file and download the file "MSW_A4_format".

2.2 INFERENCES FROM LITERATURE SURVEY

The primary objective of the literature survey was to compile a comprehensive collection of strategies employed for the detection and mitigation of fileless malware attacks. The techniques commonly include a combination of methodologies such as Machine Learning, Deep Learning, Memory Forensics, and Malware Analysis. While all the approaches discussed in the survey article were effective in detecting and mitigating fileless malware, it is important to note that no single strategy can be deemed superior due to the varying attack patterns employed by fileless threats. Among the techniques mentioned in the survey paper, it was found that the combination of memory forensics and machine learning algorithms showed greater efficacy in accurately detecting fileless malware, which typically resides in the computer's memory. This approach outperformed alternative methods in terms of accuracy. The use of Memory forensic tools and Machine learning algorithms to monitor behavioural information of the malware in the operating system is proven to work well with memory resident malware. Script based malwares require techniques that can extract and represent the contents of the code which are hidden. These features are critical for their detection using Machine learning and deep learning algorithms.

2.3 OPEN PROBLEMS IN EXISTING SYSTEM

Fileless malware detection remains a significant challenge for security researchers and practitioners. While traditional signature-based methods struggle against these ever-evolving threats, existing approaches still face several open problems that hinder their effectiveness.

- **Scarcity of open-source fileless malware samples-** Security researchers rely on access to a diverse range of malware samples to train and test detection models. However, due to the inherent stealthy nature of fileless malware, publicly available samples are scarce. This limited dataset size can lead to overfitting in machine learning models, where the model performs well on the training data but fails to generalize to unseen malware variants.
- **Feature Extraction -** Many existing techniques for extracting features indicative of fileless malware behavior are not automated. This often involves manual analysis of memory dumps or event logs, a time-consuming and resource-intensive process that requires a high level of expertise. Additionally, these techniques might not be readily scalable to handle the ever-growing volume of data generated by modern systems.
- **Evasion Techniques -** Fileless malware authors are constantly developing new techniques to evade detection. This includes techniques to obfuscate code, manipulate memory dumps, and exploit vulnerabilities in analysis tools. Detection systems need to be able to adapt to these evolving techniques and identify anomalies even when the malicious behavior is disguised.
- **Limited Visibility -** Traditional security tools often lack complete visibility into system activity, especially within virtualized environments and cloud platforms. Fileless malware can exploit these limitations to remain hidden. Developing detection methods that provide deeper visibility into system processes and memory activity is crucial.
- **Integration and Automation -** Integrating disparate security tools and automating the detection and response process is an ongoing challenge. This includes streamlining the analysis of event logs, memory dumps, and network traffic data to provide a unified view of potential fileless malware activity.
- **Privacy Concerns -** advanced detection techniques that involve deep inspection of memory and system activity might raise privacy concerns. Striking a balance between effective detection and user privacy is critical.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 FEASIBILITY STUDIES OF THE PROJECT

We have evaluated our project based on the IEEE 12207.2-1997 [25] Industry Implementation of International Standard ISO/IEC 12207:1995 (ISO/IEC 12207) Standard for Information Technology – Software Life Cycle Processes – Implementation Considerations. The 2 key parameters used for analysis are:

- Economical Feasibility
- Technical Feasibility

3.1.1 Economic Feasibility

- **Computing resources** - real time detection of malware memory snapshots require high computing resources and storage which may increase the cost of implementation.
- **Maintenance and Upkeep** - The constant updation of the snapshots and the extraction of the features to a csv format contributes to the overall cost of implementation.
- **Cost of Development** - Developing and implementing the IDS requires skilled professionals, in both cybersecurity and Artificial intelligence fields and the cost may vary based on the size and complexity of the project.

3.1.2 Technical Feasibility

- **Availability of data** - The availability of File-less malware data is very scarce. It plays an important role to find out the behavioral pattern of the malware.
- **Model architecture** - The architecture of our system and integration of event logs and memory dump model contributes to the accuracy of our prediction. Hyperparameter tuning and feature selection are also important factors to consider to increase the quality of precision.

3.2 REQUIREMENTS SPECIFICATION

Based on the reference of IEEE Std. 830 – 1998 [19] Recommended Practice for Software Requirements, The Hardware and Software Requirements are described in this part.

3.2.1 Hardware Requirements

- RAM : 16GB or above
- Processor of Frequency : 1.5GHz or above
- Processor : Intel Corei7 7th Generation or higher
- Ethernet Connection : (LAN) or a wireless adapter (Wi-Fi)

3.2.2 Software Requirements

- Operating System : Windows 10 and above
- Languages : Python, HTML, CSS, FLask
- Tools used : Visual Studio Code, Jupyter Notebook, Anaconda env
- Packages : Sckitlearn, Tensorflow, win32log, Sendgrid

3.2.3 Description of the Requirements.

Python is a high-level general-purpose and a very popular programming language. Python programming language is being used in web development and machine learning applications, along with all cutting-edge technology in the Software industry.

Machine learning (ML), is the scientific study of algorithms and statistical models that computer systems use to perform a specific task without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of artificial intelligence. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions and to uncover key insights in data mining projects. These insights subsequently drive decision-making within applications and businesses, ideally impacting key growth metrics

Flask, is a popular Framework that is used for building web applications in python. It provides a quick and easy way to create RESTFUL APIs. It is a simple and independent framework. It is faster and a full stack backend framework with almost everything built-in.

SendGrid, is a cloud-based email platform that allows users to send and manage emails through SMTP (Simple Mail Transfer Protocol) servers. We use this platform to alert notification through email to the admin users.

PowerShell, is a task automation and configuration management framework from Microsoft, consisting of a command-line shell, a scripting language, and a set of tools. PowerShell can be used to control and automate the administration of Windows systems, and other Microsoft products and technologies. It can also be used to automate the configuration of Windows systems, and other Microsoft products and technologies. It can also be used to automate the deployment of software, and to manage the configuration of software.

NumPy, which stands for numerical python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

Pandas is a one-dimensional labeled array capable of holding data of any type (integer,

string, float, Python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an Excel sheet. Labels need not be unique but must be a hashable type. The object supports both integer and label-based indexing and provides a host of methods for performing operations involving the index.

Scikit-learn, commonly known as sklearn, is a library in Python that is used to implement machine learning algorithms. It is an open-source library hence it can be used free of cost. Powerful and robust, since it provides a wide variety of tools to perform statistical modeling. This includes classification, regression, clustering, dimensionality reduction, and much more with the help of a powerful, and stable interface in Python. This library is built on NumPy, SciPy, and Matplotlib libraries.

Keras is an open-source high-level Neural Network library, which is written in Python and is capable enough to run on Theano, TensorFlow, or CNTK. It is made user-friendly, extensible, and modular for facilitating faster experimentation with deep neural networks. It has huge community support in the market as most AI companies are keen on using it. It supports Data parallelism, which means Keras can be trained on multiple GPUs at an instance to speed up the training time and processing a huge amount of data.

TensorFlow is a popular framework for machine learning and deep learning. It is a free and open-source library. It is entirely based on the Python programming language and used for numerical computation and data flow, which makes machine learning faster and easier. TensorFlow can train and run deep neural networks for image recognition, handwritten digit classification, recurrent neural networks, word embedding, natural language processing, video detection, and many more. TensorFlow is run on multiple CPUs or GPUs and also mobile operating systems.

3.2.4 ALGORITHMS USED

Multinomial Naive Bayes: Multinomial Naive Bayes is a popular classification algorithm, especially for text classification tasks. It's based on Bayes' theorem with the "naive" assumption of independence between features. The term "multinomial" indicates that the features are assumed to be generated from a multinomial distribution. This algorithm is particularly well-suited for classification with discrete features, such as word counts in text classification. It is based on the Bayes theorem, which estimates the likelihood of occurrence based on prior knowledge of the event's conditions. When

predictor B itself is available, we calculate the likelihood of class A. It's based on the formula below: $P(A|B) = P(A) * P(B|A)/P(B)$.

Random Forest: Random Forest is a popular ensemble learning technique used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the class that is the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It is a popular machine learning algorithm that belongs to the supervised learning technique. There are two assumptions in the random forest algorithm:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

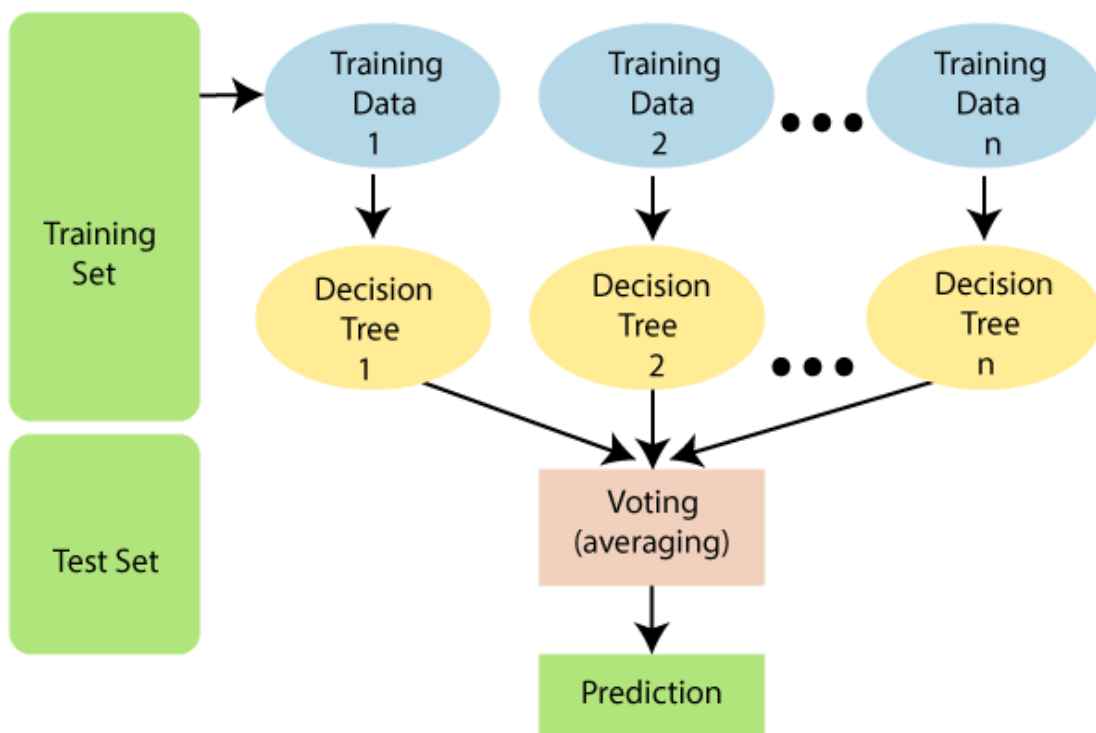


Fig 3.1 Structure of Random forest algorithm

Deep Neural Networks: A deep neural network (DNN) is a group of layers that perform

a specific task. DNNs are inspired by biological neural networks and are used in computer vision, natural language processing, and transfer learning. DNNs are a subset of machine learning that uses neural networks to cluster data and make predictions. Neural networks with three or more layers attempt to simulate the human brain by learning from large amounts of data. DNNs have become popular due to their high efficiency in achieving a variety of deep learning projects. Some popular types of DNNs include: Multi-Layer Perceptrons (MLP), Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). A deep neural network (DNN) is an ANN with multiple hidden layers between the input and output layers. Similar to shallow ANNs, DNNs can model complex non-linear relationships. The main purpose of a neural network is to receive a set of inputs, perform progressively complex calculations on them, and give output to solve real world problems like classification.

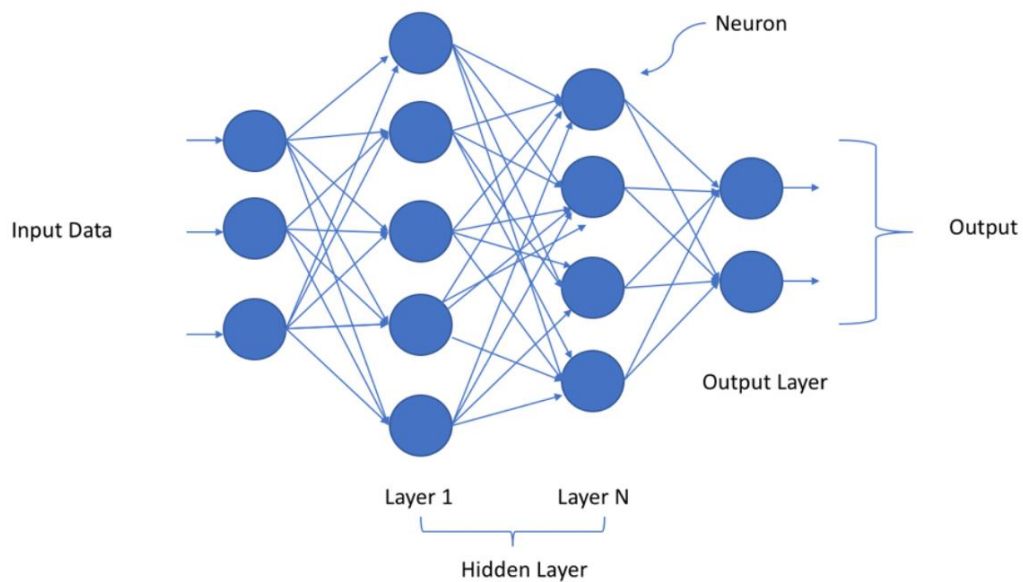


Fig 3.2 Structure of deep neural networks.

3.3 SYSTEM USE CASE

By following the IEEE Std. 1016-1998 [22] Recommended Practice for Software Design Descriptions.

3.3.1 System Description

The system is designed to Detect the presence of any malicious activity in the user system. It takes memory dump data and event logs data as the input. The system utilizes two models, Multinomial naive bayes for extracting features from event logs data and DNN deep learning model to extract features from memory dump data , to extract the behavioral features and to detect the presence of malware effectively.

3.3.2 System Use Case Scenario

- User uploads a CSV file to the system containing the behavioral memory data extracted through memory forensics to the system.
- The system constantly monitors the event logs to collect the security log data.
- The system stores both the data in a csv file.
- The system preprocesses the data by normalizing it and converting it into TF-IDF vector format.
- The Multinomial Naive Bayes model is used to classify the event logs data into the appropriate binary class as normal event or abnormal event.
- The DNN model is used to extract the features from the memory dump data and detect if it reflects the behavior of a Malware.
- The system displays the prediction to the user.
- The system also raises a windows notification alert when the abnormal activity occurs. followed by a detailed email notification alert to the admin.

CHAPTER 4

DESCRIPTION OF PROPOSED SYSTEM

After analyzing the drawbacks of the previous work, we have proposed a new system in an attempt to address the problems in the existing system. In our proposed system we plan to combine Memory forensics and ML algorithms, two of the most effective methods to detect file-less malware. Along with this, we have integrated Event log analysis to mitigate false positives.

Advantages of Proposed System

This synergy between memory forensics, machine learning, and event log analysis offers a promising path forward. The system has the potential to achieve higher detection accuracy for fileless malware by leveraging machine learning's pattern recognition capabilities. Additionally, by reducing false positives with event logs, the workload on security analysts is lessened, allowing them to focus on genuine threats.

4.1 DESCRIPTION OF RESEARCH APPROACH AND METHODS

This section describes the various methodologies implemented in the study. We have made use of two datasets: Windows event logs dataset and “CIC-Malmem-2022” Malware memory dump dataset by the Canadian institute for cybersecurity, to achieve our objectives. The two datasets undergo the phases of pre-processing, training, testing, and evaluation of ML algorithms separately and finally the best performing ML models are integrated in the deployment phase.

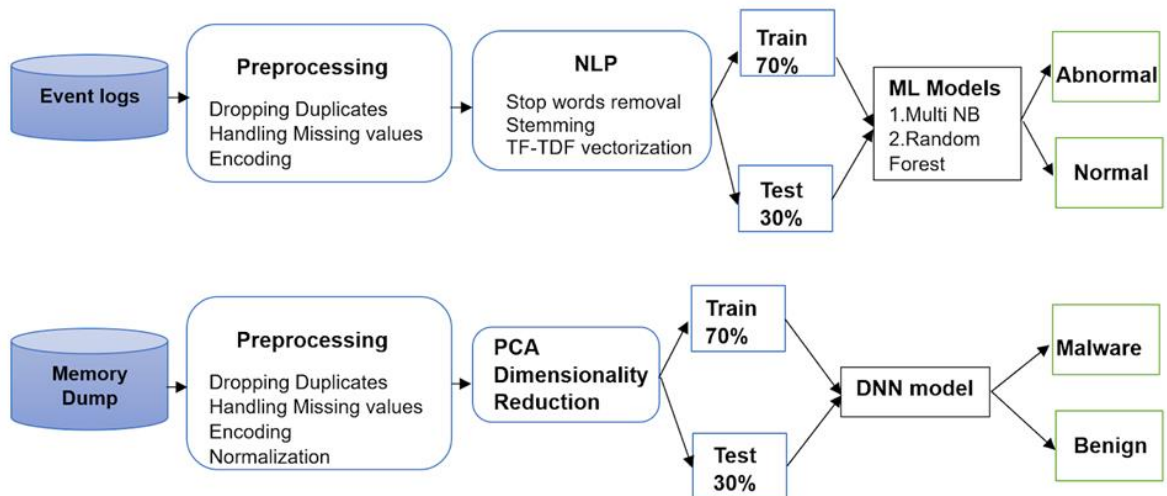


Fig: 4.1: Architecture of the proposed system

4.1.1 Dataset

This study uses two datasets for the proposed methodology. Firstly, Windows event logs data which is a record of important hardware and software actions. This information helps users, network administrators to troubleshoot issues and also forecast future issues. As we are dealing with malware activities which cause escalation of privileges in the system, we have used Security event logs data. Benign files were collected from our windows system using the shell command Get-WinEvent in the Microsoft PowerShell and these records were exported into a csv file. The malicious activity event logs were taken from the , Both the benign and malicious event logs were combined into a single csv file. The dataset contains the following features: Event ID, Timestamp, Level, Task category, and message summary.

We created a balanced dataset with 50% malicious activity records and 50% benign activity records. The exact records are 500 benign and 500 malicious, this dataset will be referred to as the event logs dataset henceforth. Secondly, an obfuscated malware dataset from CIC-MalMem-2022 was used for this study. This dataset uses debug mode for the memory dump process. The dataset is designed to detect malware through memory forensic analysis. This dataset consists of data from three malware families: a) Ransomware b) Spyware and c) Trojan Horse. There are 57 features in this dataset which outlines the details of various processes in the memory. This dataset is also balanced with 50% malicious memory dumps and 50% benign memory dumps. The dataset contains a total of 58,596 records with 29,292 benign and 29,298 malicious samples, this dataset would be referred as memory dump dataset henceforth.

4.1.2 Pre- Processing

The event logs dataset and the memory dump dataset undergo various processes in this phase to remove unwanted elements in the data like outliers, and missing values, and transform it into a standard format that is compatible with chosen ML methods.

a) Event Logs dataset preprocessing:

The event log dataset has 4 features: event ID, which is an identification number of the specific event type, subcategory indicating the recorded event type, and message summary describing the event that has been flagged. The benign records were appended with the label as normal and the malicious files were appended with the label abnormal in the dataset. The dataset is checked for any

missing values or duplicate values and they are eliminated from the dataset. In the next step dataset independent feature which is the “label” is encoded as 1 for abnormal and 0 for normal using the `df.replace()` command in pandas. Following this, we implement natural language processing (NLP) to extract valuable features.

The message summary column in the dataset is a textual description of the event. Valuable information can be extracted from this by converting it into a vector format that the machine learning models can understand. Firstly the punctuations and stopwords in the text data are removed using `nltk.stopwords` package in Python. Secondly, we use stemming techniques to reduce the words in the text to their root word so that the ML models can treat them as a single entity and reduce the complexity. Finally, the textual data is transformed into a vector format using the Term Frequency - Inverse Document Frequency (TF-IDF). The array contains 405 vector features to which the event ID, subcategory, and label columns are appended contributing to 408 features and 822 columns after the preprocessing phase.

b) Memory Dump dataset preprocessing:

The memory dump dataset contains 57 features which gives us a detailed overview of the various processes running in the RAM of the system. The benign records were encoded as 0 and the malicious records as 1 in the dataset. Missing values and duplicate values are removed from the dataset using the pandas library. The numerical features are normalized using the standard scaler function from the sklearn preprocessing library of python. It scales each value to unit variance. Owing to the complexity of the features in the dataset, we have employed Principle Component Analysis (PCA) to achieve dimensionality reduction. It is extensively used to find the highly correlated features in the dataset and remove them, finally producing a new set of features with least correlation between them. Through this we can select the features that contribute the most to the learning of the model and thereby increase the efficiency of the model. We have reduced the features set to 20 features through PCA which have proved to improve the model's efficiency.

4.1.3 Machine Learning Models

a) Event Log ML Pipeline:

We have chosen two ML models to be trained on the event log data namely Multinomial naive Bayes and random forest classifier. It predicts the tag of the text based on conditional probability and the assumption that the features are independent of each other. The dataset is split into 2 sets; 70% of the data were to be utilized for the training phase and the rest 30% data were to be utilized for the testing phase. Hyper-parameter tuning on the hyper-parameter alpha is performed on the multinomial naive Bayes model using 4-fold cross-validation using the scoring metric negative log loss. The best value for the hyperparameter alpha is found to be 0.0406. The multinomial naive Bayes model is trained on the train data and produces an accuracy of 99.39% on test data. The other ML model used in this study is the Random forest classifier model. The Gini index is used to find the purity of the classes and the number of decision trees is decided by hyperparameter tuning using 3-fold cross-validation. The best-performing hyperparameter is 75 decision trees and we train the model with training data and the accuracy achieved is 98.08%.

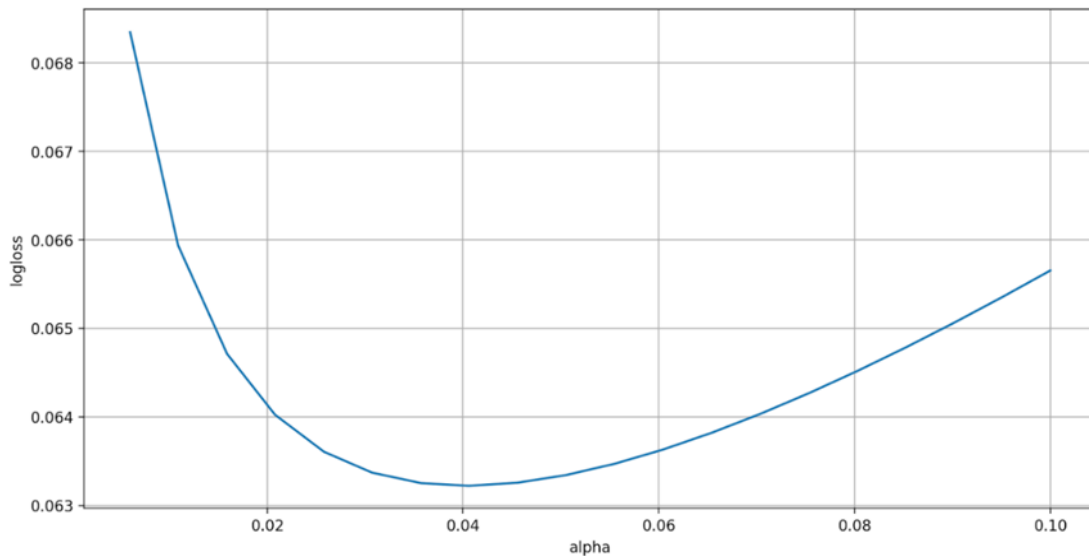


Fig 4.2: Selecting the best value of alpha for Multinomial Naive Bayes using hyper-parameter tuning.

b) Memory Dump DNN pipeline:

This study uses Deep neural networks (DNN) for training the memory dump dataset. DNN is a neural network with stacked layers. We have implemented a sequential model with 1 input layer of size 128 with activation Relu, and 2 hidden layers of size 64 and size 32 with activation Relu, and final output layer with size 1 and activation Sigmoid. This model produced an accuracy of 99% but the loss was high. A high loss indicates that the model might not be generalizing well to unseen data.

This could lead to overfitting, where the model performs well on the training data but fails to accurately classify new memory dumps. To reduce the loss and prevent the model from overfitting we have added 3 dropout layers with 20%, 30% 40% dropout rate to each layer. By employing dropouts with varying rates, we introduce a form of ensemble learning within the model itself. This helps the model learn diverse representations of the data, leading to improved generalization and reduced overfitting. This has significantly reduced the loss rate and produced the final accuracy of 99.93% compacting overfitting. This substantial improvement highlights the effectiveness of dropout regularization in mitigating overfitting and enhancing the model's ability to accurately classify unseen memory dumps.

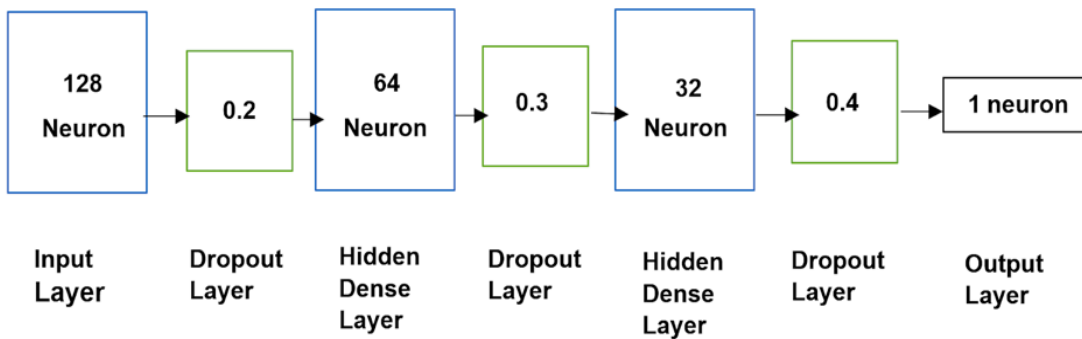


Fig 4.3: Structure of the deep neural network.

4.1.4 Training and Testing

Once the models have been developed, the next step is to train them on the labeled dataset of physical activities. The development products of the given activity confirm the requirement of that activity. The software is validated for its intended use and user need IEEE Std. 1012-1998 [21] IEEE Standard for Software Verification and Validation. The labeled dataset typically includes a set of physical activities such as walking, running, and cycling. The dataset is divided into training and testing sets, with a larger portion of the dataset used for training the models. The models are trained using backpropagation and stochastic gradient descent. Once the models have been trained, they are tested on the testing set to evaluate their accuracy.

4.1.5 Evaluation Metrics

The accuracy of the models is evaluated using various evaluation metrics such as accuracy, precision, recall, and F1-score. The accuracy of the models is the proportion of correctly classified physical activities. The precision is the ratio of correctly classified physical activities to the total number of physical activities classified as positive. The recall is the ratio of correctly classified physical activities to the total number of actual positive physical activities. The F1-score is the harmonic mean of precision and recall.

4.2 ARCHITECTURE OF PROPOSED SYSTEM

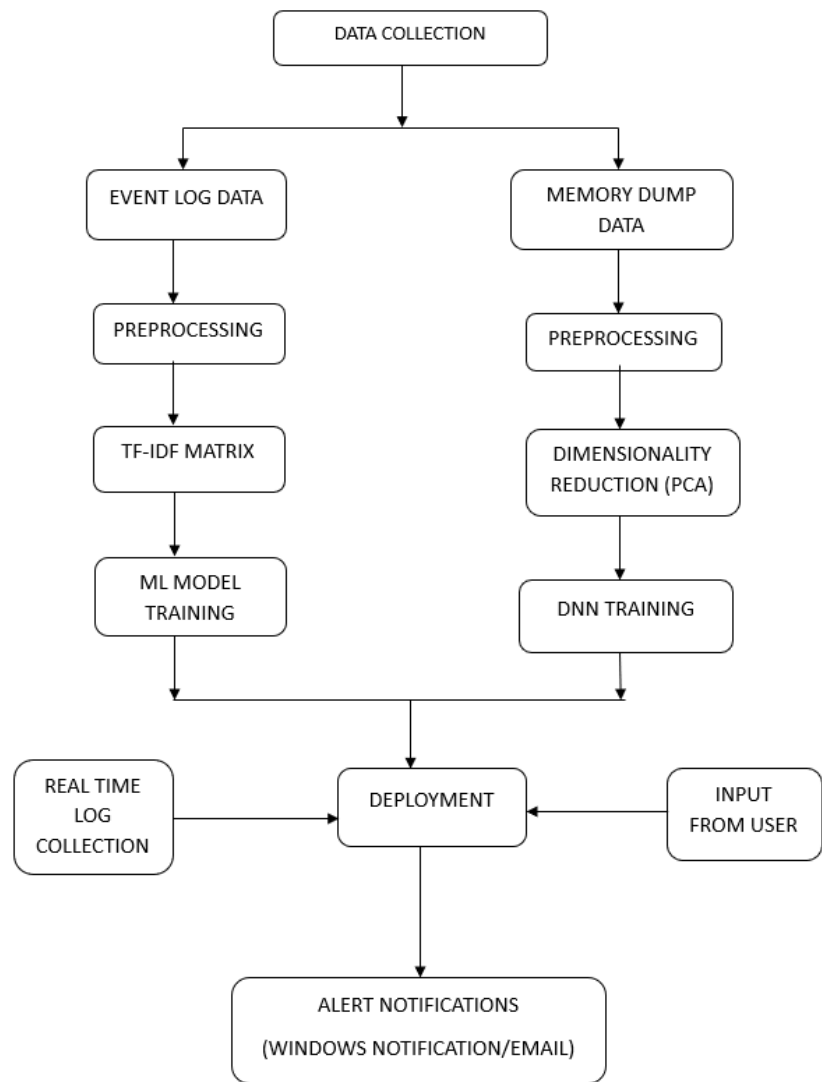


Fig. 4.4: Flow diagram of the proposed system.

The architecture of the Intrusion Detection system uses two pipelines namely, Event logs and Memory dump analysis. The event log data undergoes preprocessing steps like handling missing values, removing duplicates, and potentially encoding categorical data. This prepares the data for further analysis. Similar preprocessing steps are applied to the memory dump data. Additionally, Dimensionality Reduction using Principal Component Analysis (PCA) might be employed to reduce the number of features in the memory dump data while preserving the most important information for malware detection.

The preprocessed event log data is used to train a machine learning model, possibly a Multinomial Naive Bayes or Random Forest classifier. This model learns to classify event log entries as malicious or benign based on the patterns and features it identifies within the data. The preprocessed memory dump data is used to train a Deep Neural Network (DNN) model. DNNs are powerful for complex pattern recognition tasks. In this case, the DNN model is likely trained to identify patterns indicative of malicious behavior within the memory dump data. The system splits the preprocessed data from both sources (event logs and memory dumps) into training and testing sets.

The training data is used to train the machine learning models, while the testing data is used to evaluate their performance on unseen examples. During training, the models learn to identify patterns within the data that differentiate malicious activity from benign behavior. This involves adjusting the internal parameters of the models (weights and biases) to minimize prediction errors on the training data.

Once trained, the models can analyze new, incoming event log data and memory dumps. The event log model classifies these entries as malicious or benign, while the DNN model analyzes memory dumps for signs of malicious activity. The system likely combines the results from both models. This might involve assigning weights to the predictions from each model based on their individual performance or using techniques like ensemble learning to leverage the strengths of both models. The system employs a threshold-based approach for flagging suspicious activity. If the model's prediction score (probability of being malicious) exceeds a predefined threshold, the system raises an alert. If either model detects suspicious activity, the system generates an alert or report for further investigation by security analysts.

4.3 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED SYSTEM

Taking the reference of IEEE Std. 1012-1998 [21] Standard for Verification and Validation. The software implementation and testing plan for the proposed fileless malware detection system utilizing memory forensics and machine learning algorithms.

- **Data collection and preprocessing** - This module handles collecting data from memory dumps acquired through tools like volatility or vendor-specific solutions. Preprocessing steps like filtering, normalization, and potentially dimensionality reduction using PCA will prepare the data for analysis. Python libraries like Pandas, NumPy, and Scikit-learn can be leveraged for this purpose.
- **Model Development and Training** - This module focuses on building and training the machine learning models. We propose exploring different algorithms like Support Vector Machines (SVM) or Random Forests for analyzing event log data, while Deep Neural Networks (DNNs) are suitable for memory dump analysis. Frameworks like TensorFlow or PyTorch can be used for model development.
- **Model Evaluation and Validation** - This module assesses the performance of the trained models. Techniques like k-fold cross-validation will be employed to split data into training and testing sets, ensuring robust evaluation. Performance metrics like accuracy, precision, recall, and F1 score will be utilized to gauge model effectiveness.
- **Result Combination and Alerting** - This module focuses on combining the results from both event log and memory dump models. Techniques like weighted scores or ensemble learning approaches can be explored to create a more robust and accurate overall classifier. Triggers for generating alerts will be established based on model confidence scores.
- **Deployment** - While not the initial focus, future considerations include deploying the trained model into a production environment. Flask or Django frameworks could facilitate development of a web interface or API for integrating the system with existing security infrastructure.

The testing plan for the proposed intrusion detection system can include several stages, such as:

- **Unit testing** - This involves testing the individual components of the system, such as the data collection and pre-processing, model development and training, and model evaluation and validation, to ensure that they are working as expected.
- **Integration testing** - This involves testing the integration of the different components of the system to ensure that they are working together as expected.
- **Performance testing** - This involves testing the performance of the system, such as the accuracy, precision, and recall of the model, to ensure that it meets the required performance metric.
- **Acceptance Testing** - Real-world scenarios will be simulated to assess the system's effectiveness in detecting fileless malware and minimizing false positives. End-user feedback might be incorporated during this stage.

4.4 PROJECT MANAGEMENT PLAN

This plan outlines the management approach for developing an Intrusion Detection System (IDS) utilizing a combination of machine learning models (DNN, Multinomial Naive Bayes, Random Forest) to identify fileless malware. It adheres to the principles set forth in IEEE Std. 1058-1998 [23] and IEEE Std. 1540-2001 [24].

- **Project Planning and Management Module** - This module will include the overall planning and management of the project, including setting project goals and timelines, creating and assigning tasks, tracking progress, and communicating with team members.
- **Data Collection and Preprocessing Module** - As specified in Article IEEE Std. 1058-1998 [23] IEEE Standard for Software Project Management Plans, IEEE Std. 1540-2001 [24] IEEE Standard for Software Life Cycle Processes – Event log and memory dump dataset collection for this project is implemented and the data is preprocessed. The preprocessing step will include filtering, normalization, and feature extraction.
- **Multinomial Naive Bayes Model Module** - The MNB model will be trained using the labeled event log data. It will include designing the model architecture,

training the model on the preprocessed data, and evaluating the performance of the model on a validation dataset.

- **Random Forest Module** - The Random Forest model will be trained using the labeled event logs data. It will include designing the model architecture, training the model on the preprocessed data, and evaluating the performance of the model on a validation dataset.
- **Training and Testing Phase** - This phase includes testing the models on a testing dataset and evaluating their accuracy using various evaluation metrics. The testing phase is expected to take two weeks

Developing an Intrusion Detection System (IDS) with machine learning necessitates a well-defined project management plan with distinct modules. This modular approach fosters a structured development process. Throughout the development process, focusing on individual modules facilitates regular evaluation against project goals and user requirements. This ensures the final IDS effectively addresses the intended security objectives.

4.5 FINANCIAL REPORT ON ESTIMATED COSTING

Deep learning and Machine Learning techniques such as Multinomial Naive Bayes , Random Forest and DNN have greatly improved the accuracy of the Intrusion detection system. However, developing an IDS system using ML and DNN models can be a complex and resource-intensive process that requires careful planning and budgeting. In this financial report, we will provide an estimate of the cost of developing a IDS system using ML and DNN

- **Cost Estimate** - Hardware and Software Costs: The hardware and software cost for developing an IDS using ML and DNN can be significant. This includes the cost of purchasing and maintaining high-performance GPUs, cloud computing services, and specialized software such as TensorFlow or Keras.
- **Data Acquisition and Pre-processing Costs** - Acquiring pre-labeled datasets for training and testing machine learning models can be expensive. Alternatively, efforts might be required to curate and label data internally, consuming time and personnel resources. Storing large datasets, particularly memory dumps, might necessitate dedicated storage infrastructure, leading to additional costs.

- **Model Development Costs** - Developing and optimizing the ML and DNN models can be a time-consuming and resource-intensive process. This may involve hiring machine learning experts or outsourcing the development work to a third-party provider. Additionally, the cost of testing and validating the models can be significant, as this requires access to large datasets and specialized software tools.
- **Deployment and Maintenance Costs** - Once the models have been developed, they need to be deployed and integrated into the target system. This may involve additional costs for server infrastructure, API development, and ongoing maintenance and support. The cost of ongoing maintenance and support can be significant, especially if the system is deployed in a complex environment with multiple users or if the models need to be updated frequently.

4.6 SOFTWARE TO OPERATIONS PLAN

Using IEEE Std. 1016-1998 [22] Recommended Practice for Software Design Descriptions.

An IDS developed with ML and Deep Neural Networks (DNNs) for fileless malware detection is a complex endeavor. Developing a IDS involves various stages such as data collection, model development, testing, and deployment. Once the system is developed and tested, it needs to be transitioned to operations to ensure its long-term sustainability and effectiveness. This requires a well-planned project transition/ software to operations plan. In this document, we will outline the project transition/ software to operations plan for our intrusion detection system.

- **Monitoring** - The intrusion detection system should be monitored for performance, errors, and other issues on a regular basis. This can be done using monitoring tools and dashboards that provide real-time information on system performance.
- **Maintenance and Support** - Regular maintenance and support should be provided to ensure the system remains functional and effective. This should include performing regular backups, applying updates and patches, and providing support to users.
- **Security** - The intrusion detection system should be regularly audited for security vulnerabilities, and measures should be taken to address any issues that are identified. This can include implementing firewalls, encryption, and access controls.

- **Upgrades and Enhancements** - The intrusion detection system should be periodically upgraded and enhanced to ensure that it remains up-to-date with the latest technology and requirements. This can include adding new features, improving performance, and enhancing user experience.
- **Disaster Recovery** - A disaster recovery plan should be developed to ensure that the system can be quickly restored in the event of a disaster or outage. This should include regular backups and a plan for restoring the system in the event of a catastrophic failure.

Developing an intrusion detection system is a complex process that requires careful planning and execution. Once the system is developed and tested, it needs to be transitioned to operations to ensure its long-term sustainability and effectiveness. This requires a well-planned project transition/ software to operations plan. The plan should include establishing a project transition team, defining transition requirements, developing a deployment plan, developing a maintenance and backup plan.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 DEVELOPMENT AND DEPLOYMENT SETUP

To develop and deploy a Intrusion Detection system using Machine learning and deep learning models , you can follow the steps below:

- **Data Collection and Preprocessing** - Collect data from various sources and pre-process it to make it suitable for machine learning. You can use various techniques for data pre-processing, such as normalization, data augmentation, and feature extraction. In addition, We have converted the textual data into a Tf-Idf vector. We have used the PCA algorithm to reduce the dimensionality of the features.
- **Model Architecture Design** - Design the architecture of the DNN model. This architecture should include sequential input layers for feature extraction from the input data, dropout layers and hidden layers, and fully connected dense output layers with Relu activation for detection.
- **Hyperparameter tuning** - We have used 3 fold cross validation to conduct fine tuning of the parameters alpha in multinomial naive bayes and the parameter criterion which indicates the depth in Random forest.
- **Model Evaluation** - Evaluate the performance of the trained model using various metrics, such as accuracy, precision, recall, and F1 score. You can also use visualization techniques, such as confusion matrices or ROC curves, to analyze the performance of the model.
- **Deployment** - Deploy the model in a production environment. You can use various frameworks, such as TensorFlow or PyTorch, to create a deployable model. We have used the flask web application framework. You can also use cloud platforms, such as AWS or Azure, to deploy the model in a scalable and cost-effective way.
- **Continuous Improvement** - Continuously improve the model by retraining it on new data and fine-tuning the hyperparameters. You can also use techniques such as transfer learning to improve the model's performance.

Overall, the development and deployment setup for Intrusion Detection System for fileless malware using Machine learning involves several steps, including, data

collection, data preprocessing, model training and testing, deployment and continuous improvement. With proper planning and execution, a robust and secure Intrusion detection system for fileless malware can be developed and deployed in a real-world environment.

5.2 TESTING

By using IEEE 829 [18], IEEE 1008 [20] and IEEE 1012 [21], the testing plan is formulated. We have used the following testing techniques can be used to evaluate the performance of our ML and DNN models:

- **Hold-Out Testing** - In this technique, the dataset is divided into two parts: a training set and a testing set. The model is trained on the training set and then evaluated on the testing set to determine its accuracy.
- **Cross-Validation** - We have used 3 fold cross validation to test and evaluate the performance of the models.
- **Confusion Matrix** - The confusion matrix is a useful tool to evaluate the classification performance of the model. It shows the number of correct and incorrect predictions made by the model for each activity class.
- **F1 Score** - The F1 score is a measure of the model's accuracy, calculated as the harmonic mean of precision and recall. It is a useful metric for evaluating the overall performance of the model.

In addition to these techniques, other measures can be used to evaluate the performance of the model, such as accuracy, precision, recall, and the area under the ROC curve. These measures can be used to compare the performance of all the 3 models used for the experimental analysis and to select the best model with the highest accuracy for deployment.

CHAPTER 6

OUTCOMES AND DISCUSSION

6.1 INTRODUCTION

Fileless malware, a growing threat in the cybersecurity landscape, utilizes legitimate system processes to avoid traditional signature-based detection methods. This necessitates advanced techniques for robust intrusion detection. Machine learning (ML) models, particularly Deep Neural Networks (DNNs) and other complementary algorithms, offer promising solutions for identifying these elusive threats. SVMs effectively analyze event log data, identifying patterns that differentiate malicious activity from benign behavior within logs. Random Forest can further enhance detection by introducing diversity and potentially improving overall model robustness.

The outcomes of IDS using DNN models can vary depending on various factors such as the dataset used, the architecture of the model, and the preprocessing techniques applied to the data. However, some common outcomes of IDS using ML and DNN models include:

- **Improved Detection Rates** - By combining the strengths of DNNs and other models, the system can achieve high accuracy in detecting fileless malware, leading to a more secure environment. Studies suggest that well-designed machine learning models can achieve significant improvements in detection rates compared to traditional methods.
- **Robustness** - DNN models have been shown to be robust to noise and variations in sensor data. This is important because memory dump data can be noisy and contain variations due to various factors.
- **Reduced False Positives** - Fine-tuning model parameters and utilizing techniques like weighted scoring or ensemble learning methods can help minimize false positives, reducing unnecessary system alerts and operational overhead.
- **Adaptability**- The machine learning models can be continuously trained on new data, allowing them to adapt to evolving malware tactics and maintain effectiveness against emerging threats. This adaptability is crucial in the ever-changing cybersecurity landscape.

6.2 IEEE STANDARDS FOLLOWED IN THE PROJECT

- **IEEE 1012-1998** - This standard provides a process framework for verifying and validating software systems. It can be applied throughout the development lifecycle of the IDS to ensure it meets the specified requirements and effectively detects fileless malware.
- **IEEE 1058-1998** - This standard outlines the elements and structure of a comprehensive project management plan. Following this standard helps establish a clear roadmap for the IDS project, including defining goals, timelines, tasks, resource allocation, and risk management strategies.
- **IEEE 1540-2001** - Standard for Software Life Cycle Processes - Risk Management: This standard details a risk management process specifically applicable to software development lifecycles.
- **IEEE 2795-2014** - This standard provides a framework for defining and implementing security requirements for software applications. While the IDS itself is a security tool, following this standard can help ensure the IDS itself is secure and not vulnerable to exploitation.
- **IEEE 1063-2015** - This standard provides guidelines for the software life cycle processes, which can be useful in guiding the development, testing, and maintenance of the IDS.
- **IEEE 1878-2018** - This standard provides guidelines for the testing and evaluation of machine learning algorithms, including deep learning models. It can be useful in ensuring the accuracy and reliability of the IDS.

6.3 CONSTRAINTS

While machine learning models offer significant promise for Intrusion Detection Systems (IDS), there are several constraints that can impact their performance and effectiveness:

- **Limited Data Availability:** Training effective machine learning models, especially DNNs, requires a vast amount of labeled data containing both malicious and benign network activity. Obtaining high-quality, labeled data specifically for fileless malware detection can be challenging and expensive.

- **Data Imbalance** - Real-world network traffic often exhibits a class imbalance, with malicious activity being a much smaller portion compared to benign traffic. This imbalance can lead to models that prioritize classifying everything as benign, potentially missing actual threats.
- **Data Privacy Concerns:** Collecting and storing network traffic data, especially user data, raises privacy concerns. Regulations like GDPR and CCPA might restrict data collection practices, limiting the availability of training data.
- **Overfitting and Generalizability:** There's a risk of overfitting the model to the training data, leading to poor performance on unseen malicious activity. Techniques like regularization and data augmentation are crucial to improve model generalizability.
- **High Computational Demands:** Training DNNs requires significant computational resources like GPUs or high-performance computing clusters. This can be expensive to acquire and maintain, impacting project budgets.
- **Evolving Threats:** The landscape of cyber threats is constantly evolving. Machine learning models need to be continuously updated with new data to maintain effectiveness against emerging fileless malware tactics.
- **Operational Cost** - Beyond development costs, deploying and maintaining an IDS involves ongoing costs for infrastructure, system administration, and potential software licensing fees.
- **Expertise** - Developing and maintaining an ML-based IDS requires expertise in machine learning, network security, and system administration. Hiring or training personnel with these skill sets can be challenging for some organizations.

6.4 TRADEOFF IN THE PROJECT

Following the risk management principles outlined in IEEE Std. 1540-2001 [24], here's a detailed analysis of key tradeoffs to consider when developing an IDS with ML and DNNs for fileless malware detection:

- **Detection Accuracy vs. False Positives** - The primary goal is achieving a high detection rate for malicious activity. DNNs excel at pattern recognition, potentially leading to a significant improvement in identifying fileless malware compared to traditional signature-based methods.

complexity. Thus, Striking a balance between high accuracy and minimizing false positives is crucial.

- **Generalizability vs. Overfitting** - The IDS needs to effectively detect novel and unseen malware variants. DNNs trained on a diverse dataset can learn generalizable patterns that are applicable to broader threat landscapes. With limited or overly specific training data, DNNs can become overfitted, performing well on the training data but failing to generalize to unseen attacks.
- **Computational Resources vs. Real-Time Performance**- Training and running complex DNN models require significant computational power, especially for high-speed network traffic analysis. Resource limitations might necessitate using less complex models. For effective threat detection, the IDS needs to analyze data and generate alerts with minimal latency. Latency due to model complexity can hinder real-time response capabilities
- **Interpretability vs. Performance** - DNN and ML models can be complex and difficult to interpret. However, interpretability is important for understanding the factors that contribute to human activities and improving the accuracy of the system. There may be a tradeoff between interpretability and performance.
- **Data Privacy vs. Security Effectiveness**- Collecting and analyzing network traffic, especially memory dumps, can raise privacy concerns. Regulations like GDPR and CCPA might place restrictions on data collection and usage. Access to rich data sources like memory dumps can provide valuable insights for training DNNs to identify complex attack patterns. Limited data access might hinder the effectiveness of the IDS.

Developing an effective IDS with ML and DNNs for fileless malware detection requires careful consideration of these tradeoffs. By understanding the competing interests and implementing appropriate mitigation strategies, project teams can achieve a robust system that balances accuracy, efficiency, interpretability, data privacy, and overall security effectiveness.

	A	B	C	D	E	F
1	Keywords	Date and Time	Source	Event ID	Task Category	Message Summary
						A user's local group membership was enumerated. Subject: Security ID:DESKTOP-GU3MJG8\reeya Account Name:reeya Account Domain:DESKTOP-GU3MJG8 Logon ID:0x32CE6 User: Security ID:DESKTOP-GU3MJG8\reeya Account Name:reeya Account Domain:DESKTOP-GU3MJG8 Process Information: Process ID:0x1298 Process Name:C:\Windows\System32\mmc.exe
2	Audit Success	9/29/2023 15:41	Microsoft-Windows-Security-Auditing	4798	User Account Management	

Fig. 7.2.2: Event logs Dataset

We have used event logs dataset to monitor the security event logs and memory dump data to detect the fileless malware activity present in the memory.

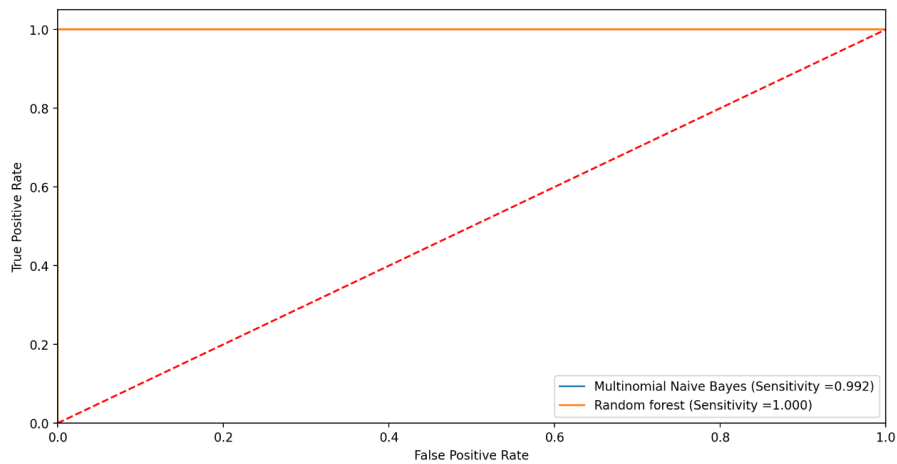


Fig. 7.2.3: ROC curve of Multinomial NB and Random forest

Here we have our Multinomial NB Model's confusion matrix from our Training and Testing Steps.

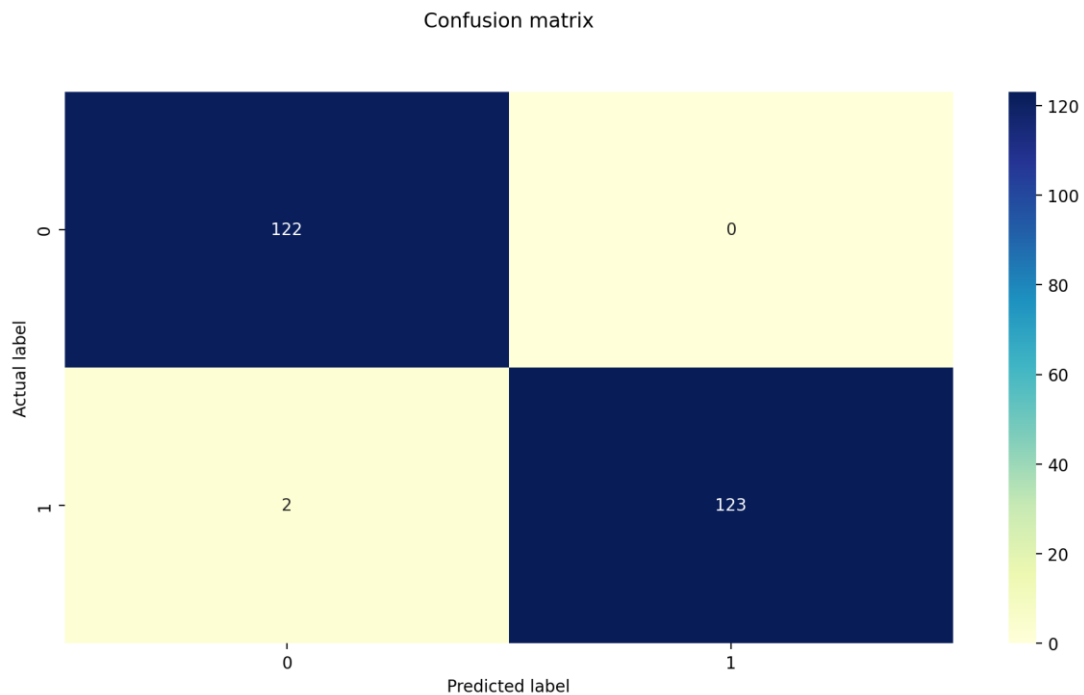


Fig. 7.2.4: Confusion Matrix of Multinomial NB Model

```
[ ] array1.shape

[ ] eventID_array = np.array([[eventID]], dtype='float64')
    features= np.concatenate([array1, eventID_array], axis=1)

[ ] df=pd.DataFrame(features)
    df

   0  1  2  3  4  5  6  7  8  9  ...  397 398 399 400 401 402 403 404 405 406
0  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0  ...  0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 4616.0
1 rows x 407 columns

[ ] pred=model_NB.predict(df)

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MultinomialNB was fitted with feature names
  warnings.warn(

[ ] if pred== 0:
    print("normal activity")
    else:
    print("abnormal activity")

abnormal activity
```

Fig. 7.2.5: Unit testing of Multinomial Naive Bayes Model

We obtained an accuracy of 99.1% after training and testing the Multinomial Naive Bayes model.

The total number of benign and malware classes

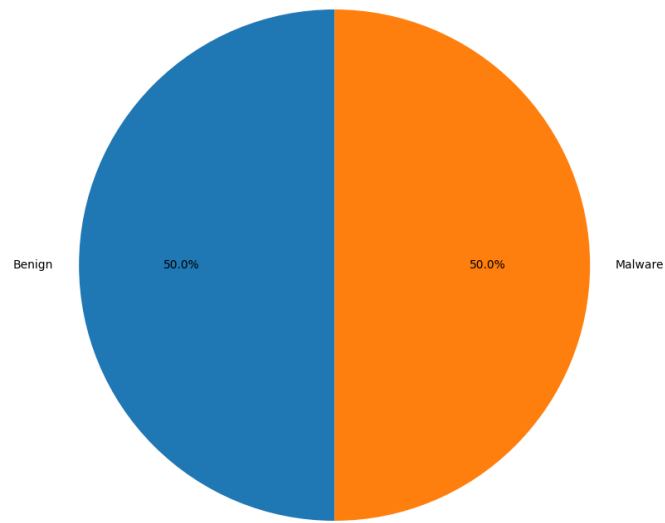


Fig: 7.2.6: Memory dump data category

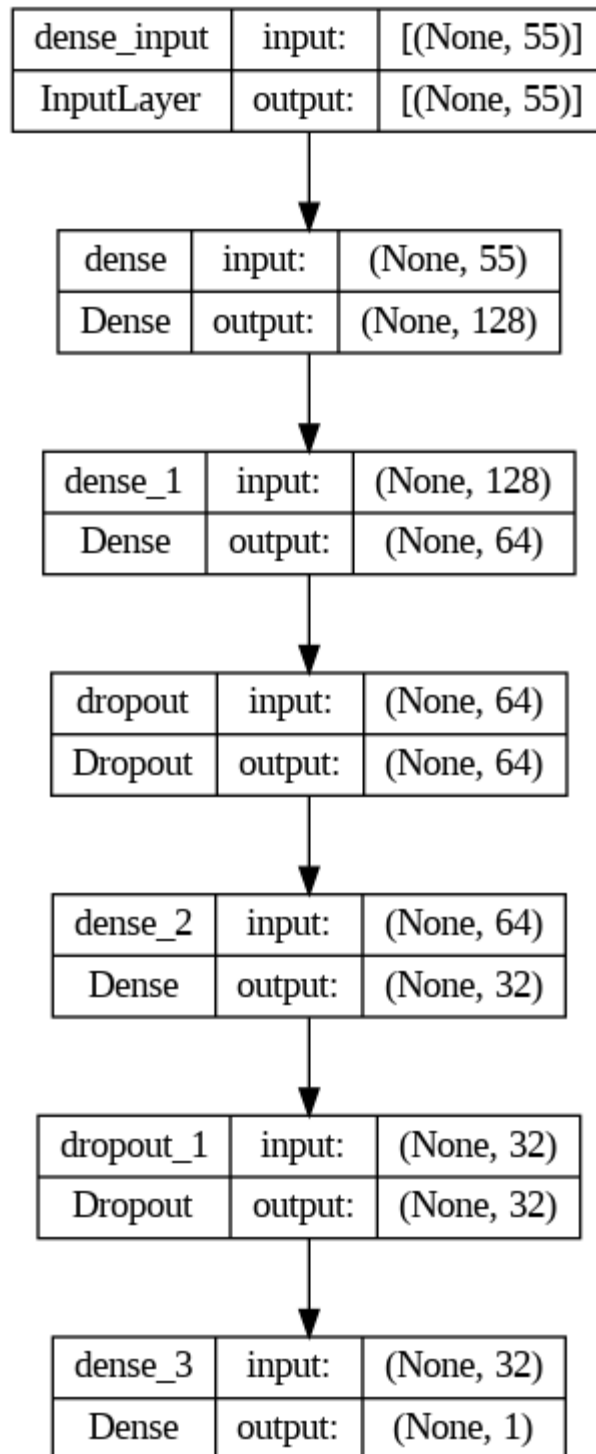


Fig. 7.2.7: Creation of DNN Model

The above picture shows the creation of the DNN Model.

Here we have our DNN Model's Loss and Accuracy Curves from our Training and Testing Steps.

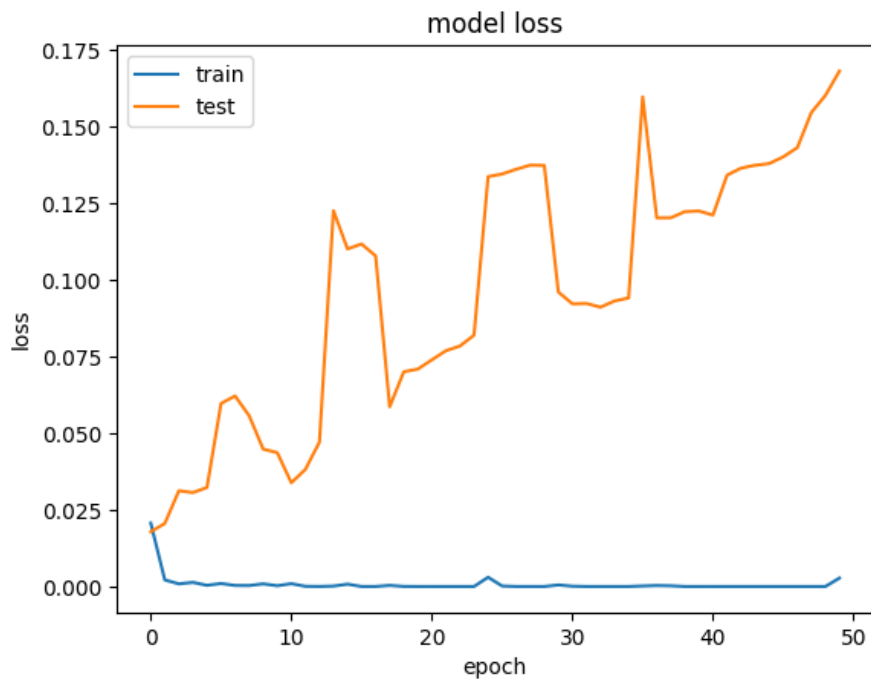


Fig. 7.2.8: Total Loss vs Total Validation Loss of DNN Model

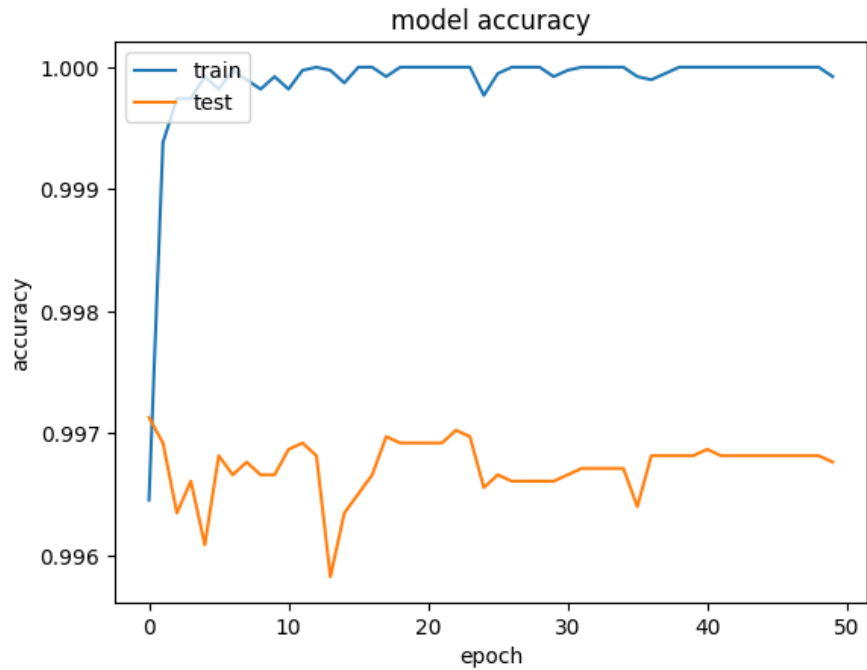


Fig. 7.2.9: Total Accuracy vs Total Validation Accuracy of LRCN Model

```
[ ] y_pred_probs = model.predict(x_test)
y_pred = (y_pred_probs > 0.5).astype(int)
```

545/545 [=====] - 1s 1ms/step

```
loss, accuracy = model.evaluate(x_test, y_test)
print('Accuracy:', accuracy)
```

545/545 [=====] - 1s 2ms/step - loss: 0.0591 - accuracy: 0.9991
Accuracy: 0.9990814924240112

Fig. 7.2.10: Accuracy of LRCN Model

After training and testing the DNN model produced 99.9% accuracy

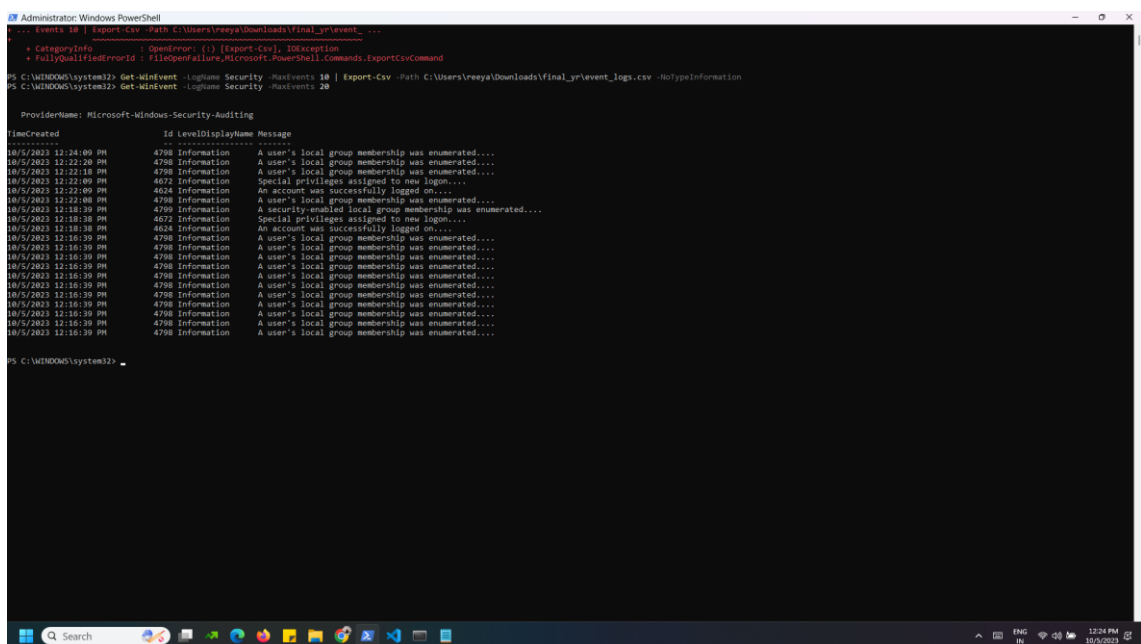


Fig. 7.2.11: Picture showing the collection event log data in real time

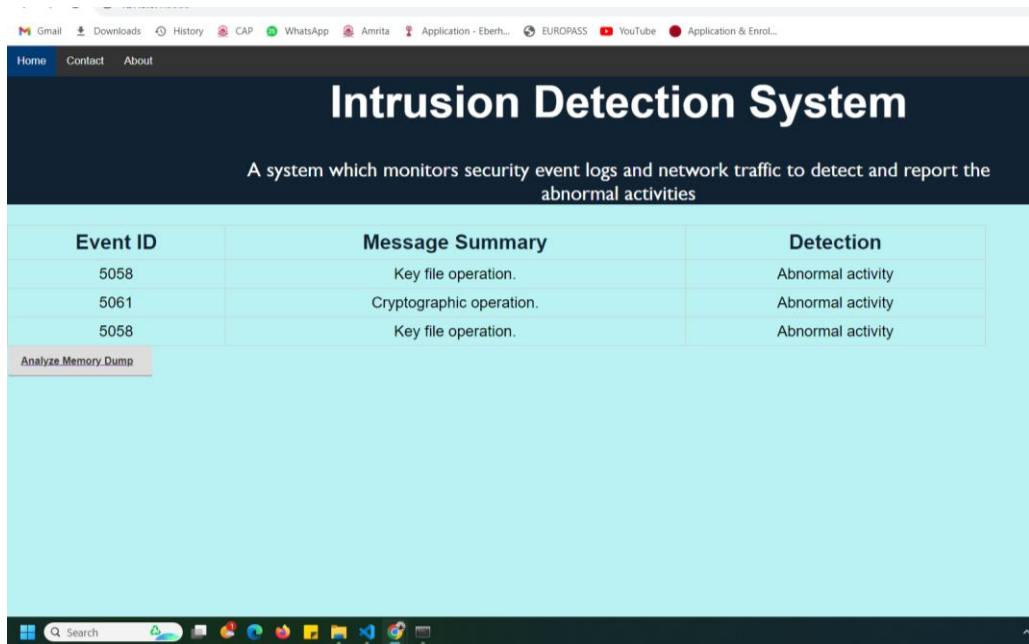


Fig. 7.2.12: The flask application page where the user can view the detected activities.

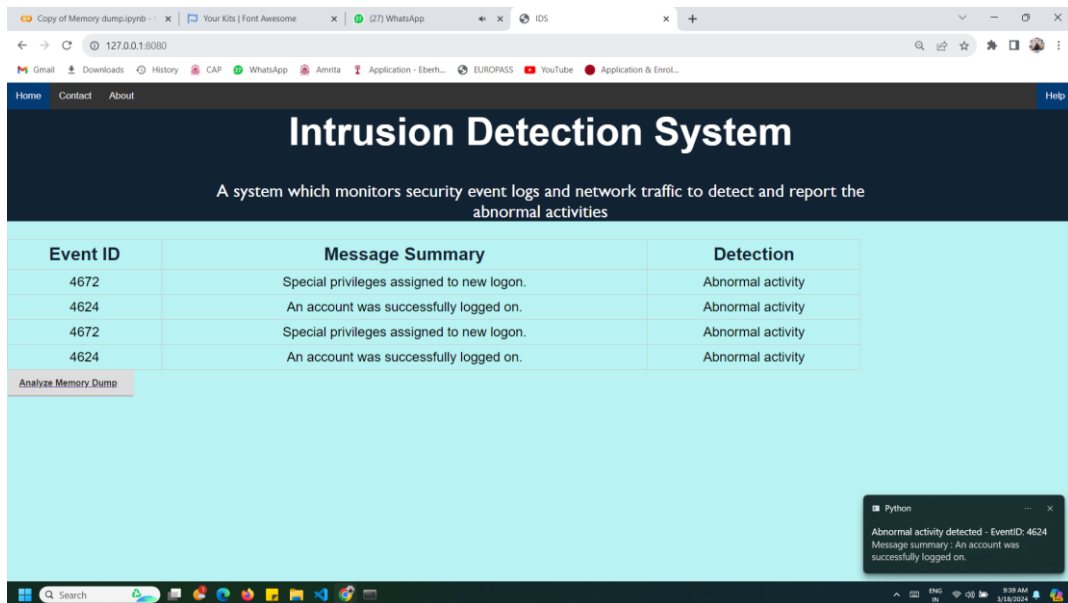


Fig. 7.2.13: The Alert notification appearing to the user.

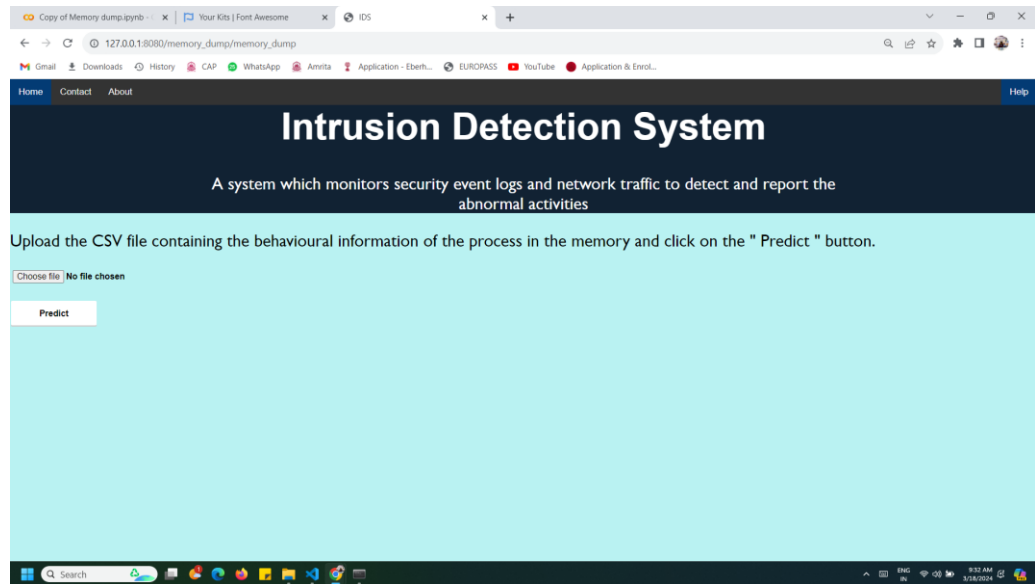


Fig. 7.2.14: The Page to upload the input file which contains the behavioral patterns of the malware.

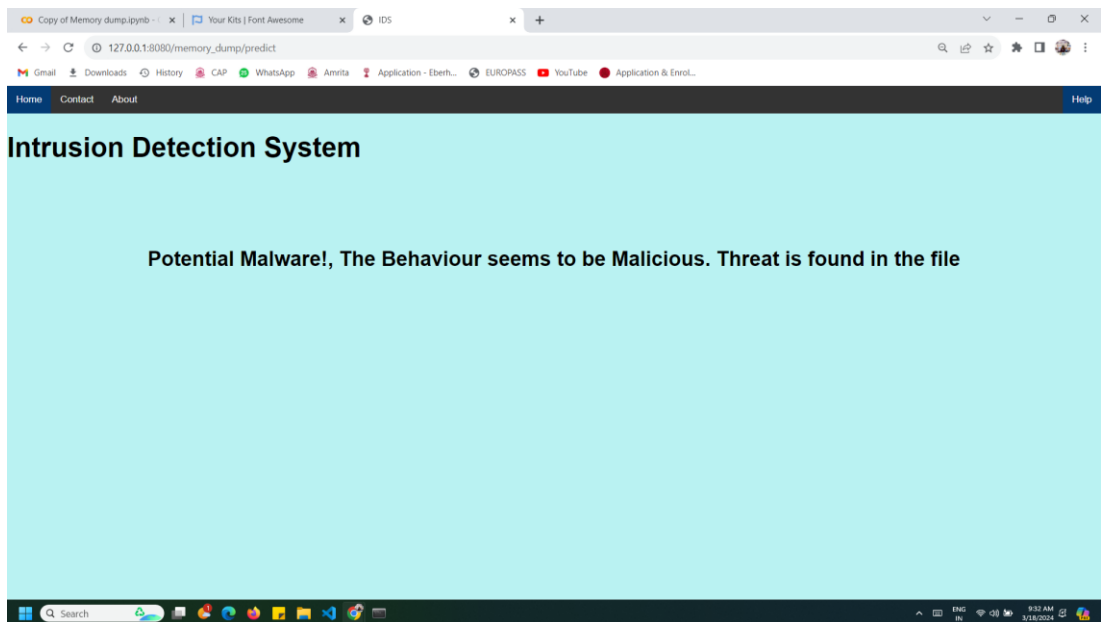


Fig. 7.2.15: Prediction Page which shows the result that the model detected.

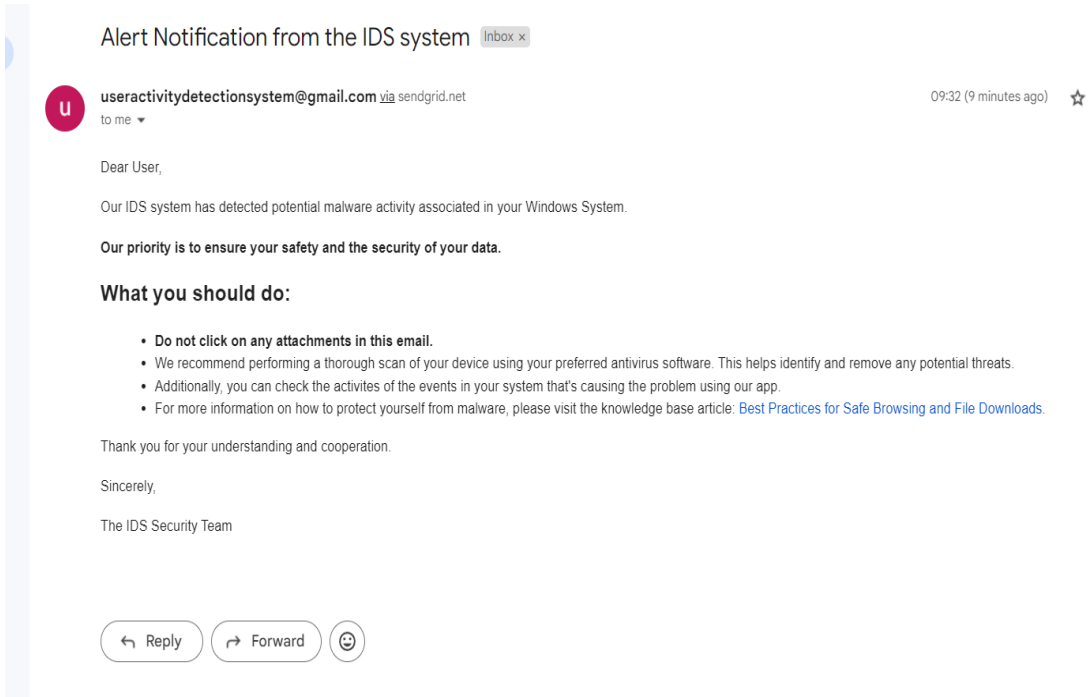


Fig. 7.2.16: Email alert notification sent to the Admin user.

CHAPTER 8

CONCLUSION

8.1 CONCLUSION

This paper presents two pipelines leveraging machine learning and deep learning classifiers to detect abnormal activities indicative of malware presence within a system. By analyzing Windows event logs and memory dump data, these pipelines offer a robust approach to identifying and mitigating potential threats. Our utilization of deep neural networks for malware detection has demonstrated superior performance compared to existing models outlined in the literature.

Additionally, employing the Random Forest model to identify event activities leading to privilege escalation has further enhanced the efficacy of our detection mechanisms. The results highlight the effectiveness of these pipelines in identifying and mitigating malware activity, offering a valuable contribution to the field of cybersecurity. Moving forward, the findings of this study could be leveraged to develop standalone software solutions dedicated to monitoring file-less malware activity, thereby enhancing system security.

8.2 RECOMMENDATIONS FOR FUTURE RESEARCH

The current state of IDS with ML and DNNs offers significant potential for fileless malware detection. However, there's always room for improvement. Here are some key areas for future exploration to further strengthen this technology:

- **Improved architectures** - Implementing attention mechanisms within DNNs can enable the model to focus on critical aspects of network traffic data (e.g., specific system calls, memory access patterns) during analysis, potentially leading to more accurate anomaly detection.
- **Multi-source data fusion** - Enriching the IDS with real-time threat intelligence feeds containing information on the latest malware variants and attack methods can further enhance detection capabilities and proactively identify emerging threats.
- **Adversarial learning** - Training the IDS with adversarial samples (crafted to evade detection) can improve its robustness against novel and sophisticated attacks. This proactive approach can help the IDS stay ahead of the curve in the

face of continuously evolving threats.

- **Human-in-the-Loop Design-** Developing a human-centric approach where the IDS flags suspicious activity and presents clear, interpretable information to security analysts empowers them to make informed decisions and prioritize critical threats.
- **Privacy and security -** As intrusion detection systems become more prevalent, there is a need to address privacy and security concerns. Future work can explore methods to ensure that these systems protect user privacy and prevent malicious attacks.

By actively exploring these promising avenues for improvement, future IDS solutions with ML and DNNs can become even more effective in safeguarding systems against the ever-growing threat of fileless malware.

REFERENCES

- [1] Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," 2020 International Conference on Cyber Warfare and Security (ICCWS), Islamabad, Pakistan, 2020, pp. 1-8.
- [2] Sudhakar, Kumar, S. An emerging threat Fileless malware: a survey and research challenges. *Cybersecur* 3, 1 (2020).
- [3] B. N. Sanjay, D. C. Rakshith, R. B. Akash and D. V. V. Hegde, "An Approach to Detect Fileless Malware and Defend its Evasive mechanisms," 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 2018, pp. 234-239.
- [4] P. Borana, V. Sihag, G. Choudhary, M. Vardhan and P. Singh, "An Assistive Tool For Fileless Malware Detection," 2021 World Automation Congress (WAC), Taipei, Taiwan, 2021, pp. 21-25.
- [5] O. Khalid et al., "An Insight into the Machine-Learning-Based Fileless Malware Detection," *Sensors*, vol. 23, no. 2, p. 612, Jan. 2023
- [6] S. Mele Pottaraikkal and A. Sujeer Sugatha, "Effectiveness of Multiple Memory-Images in detecting Fileless Malware," 2023 11th International Symposium on Digital Forensics and Security (ISDFS), Chattanooga, TN, USA, 2023, pp. 1-5'
- [7] S. Zhang, C. Hu, L. Wang, M. J. Mihaljevic, S. Xu, and T. Lan, "A Malware Detection Approach Based on Deep Learning and Memory Forensics," *Symmetry*, vol. 15, no. 3, p. 758, Mar. 2023, doi: 10.3390/sym15030758.
- [8] M. -H. Tsai, C. -C. Lin, Z. -G. He, W. -C. Yang and C. -L. Lei, "PowerDP: De-Obfuscating and Profiling Malicious PowerShell Commands with Multi-Label Classifiers," in *IEEE Access*, vol. 11, pp. 256-270, 2023.
- [9] "JSLess: A Tale of a Fileless JavaScript Memory-Resident Malware," [springerprofessional.de](https://www.springerprofessional.de/en/jsless-a-tale-of-a-fileless-javascript-memory-resident-malware/17402116). Available: <https://www.springerprofessional.de/en/jsless-a-tale-of-a-fileless-javascript-memory-resident-malware/17402116>
- [10] R. Bearden and D. C. -T. Lo, "Automated Microsoft Office macro malware detection using machine learning," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 4448-4452.
- [11] S. Kim, S. Hong, J. Oh and H. Lee, "Obfuscated VBA Macro Detection Using Machine Learning," 2018 48th Annual IEEE/IFIP International Conference on

- Dependable Systems and Networks (DSN), Luxembourg, Luxembourg, 2018, pp. 490-501,
- [12] A. More, K. Joshi, and K. Kumar, "Simulation Framework for Fileless Malware Detection and Analysis," *Simulation Framework for Fileless Malware Detection and Analysis* by Aditya More, Kashyap Joshi, Kapil Kumar :: SSRN, Jul. 12, 2023.
 - [13] M. U. Rana, M. Ali Shah and O. Ellahi, "Malware Persistence and Obfuscation: An Analysis on Concealed Strategies," 2021 26th International Conference on Automation and Computing (ICAC), Portsmouth, United Kingdom, 2021, pp. 1-6,
 - [14] Botacin, M., Grégio, A., & Alves, M. A. Z. (2020, September). Near-memory & in-memory detection of fileless malware. In *The International Symposium on Memory Systems* (pp. 23-38).
 - [15] Nuangpookka, P., Mengistu, Z., & Bafail, G. (2020). Fileless malware and programmatic method of detection. *Journal of Computing Sciences in Colleges*, 36(3), 161-162.
 - [16] Krishna, B. L. (2020). Comparative study of fileless ransomware. *Int. J. Trend Sci. Res. Develop.*, 4, 608-616.
 - [17] P. Chen, F. Li and J. Li, "Research on Intrusion Detection Model Based on Bagged Tree," 2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), Shenyang, China, 2021, pp. 579-582,
 - [18] D. P. Gaikwad and R. C. Thool, "Intrusion Detection System Using Bagging Ensemble Method of Machine Learning," 2015 International Conference on Computing Communication Control and Automation, Pune, India, 2015, pp. 291-295,
 - [19] S. J. Kamal, G. Karishma, S. T, A. Rajesh, V. Muthiah-Nakarajan and B. S, "ANN-LSTM Assisted Intrusion Detection for Next Generation Core Networks," 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-5,
 - [20] C. Liu, Y. Liu, Y. Yan and J. Wang, "An Intrusion Detection Model With Hierarchical Attention Mechanism," in *IEEE Access*, vol. 8, pp. 67542-67554, 2020.
 - [21] R. Koniki, M. D. Ampapurapu and P. K. Kollu, "An Anomaly Based Network Intrusion Detection System Using LSTM and GRU," 2022 International Conference on Electronic Systems and Intelligent Computing (ICESIC), Chennai, India, 2022, pp. 79-84.
 - [22] P. Chen, Y. Guo, J. Zhang, Y. Wang and H. Hu, "A Novel Preprocessing Methodology for DNN-Based Intrusion Detection," 2020 IEEE 6th International

Conference on Computer and Communications (ICCC), Chengdu, China, 2020, pp. 2059-2064,

- [23] IEEE Std. 829-1998 IEEE Standard for Software Test Documentation
- [24] IEEE Std. 830-1998 IEEE Recommended Practice for Software Requirements
- [25] IEEE Std. 1008-1997 IEEE Standard for Software Unit Testing
- [26] IEEE Std. 1012-1998 IEEE Standard for Software Verification and Validation
- [27] IEEE Std. 1016-1998 IEEE Recommended Practice for Software Design Descriptions
- [28] IEEE Std. 1058-1998 IEEE Standard for Software Project Management Plans
- [29] IEEE Std. 1540-2001 IEEE Standard for Software Life Cycle Processes – Risk Management
- [30] IEEE 12207.2-1997 Industry Implementation of International Standard ISO/IEC 12207:1995 (ISO/IEC 12207) Standard for Information Technology – Software Life Cycle Processes – Implementation Consideration.

APPENDIX

A. SOURCE CODE

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
#!pip install --upgrade scikit-learn==1.4.1.post1
df=pd.read_csv("/content/drive/MyDrive/Obfuscated-MalMem2022.csv")
df.head()
df["Class"].value_counts()
df2=df.dropna(subset=['Class'])
data_pie=df["Class"].value_counts()
data_pie
plt.figure(figsize=(10,10))
plt.pie(data_pie, labels=data_pie.index, startangle=90, autopct="%.1f%%")
plt.title("The total number of benign and malware classes")
plt.show()
df = df.drop_duplicates()
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,StandardScaler
label_encoder = LabelEncoder()
df['Category'] = label_encoder.fit_transform(df['Category'])
df['Class'] = label_encoder.fit_transform(df['Class'])
import pickle
with open('label_encoder.pkl', 'wb') as f:
    pickle.dump(label_encoder, f)
X = df.drop(['Class','Category'], axis=1)
y = df['Class']
scaler = StandardScaler()
X_scaled=scaler.fit_transform(X)
X_scaled
n= 20
from sklearn.decomposition import PCA
pca = PCA(n_components=n)
X1p = pca.fit_transform(X)
```

```

pca.explained_variance_ratio_
scaler = StandardScaler()
Xp=scaler.fit_transform(Xp)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3,
random_state=100)
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_scaled.shape[1],))) #
Input layer
#model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(32, activation='relu'))
model.add(Dropout(0.2)) # Hidden layer
model.add(Dense(1, activation='sigmoid'))
anomaly_data= pd.read_csv("/content/drive/MyDrive/Malware
data/WindowsSecurityAuditEvents.csv")
anomaly_data['Label'] = "anomaly"
anomaly_data.head()
anomaly_data.shape
normal_data=pd.read_csv("/content/drive/MyDrive/Malware data/logs.csv")
normal_data=normal_data[:411]
normal_data['Label'] = 'normal'
normal_data.head()
normal_data =normal_data.rename(columns={'Id':'Event ID','Message':'Message
Summary','TaskDisplayName':'Subcategory'})
normal_data["Message Summary"] = normal_data["Message
Summary"].str.split('Subject').str[0].str.strip()
normal_data.head()
features_n=normal_data[['Event ID','Subcategory','Message Summary','Label']]
features_a = anomaly_data[['Event ID', 'Subcategory','Message Summary','Label']]
data=pd.concat([features_n, features_a], ignore_index=True)
data.shape
data.isnull().values.any()

```

```

data.info()
data["Label"].value_counts()
data["Subcategory"].value_counts()
data.replace(to_replace="normal", value=0, inplace=True)
data.replace(to_replace="anomaly", value=1, inplace=True)
def remove_punctuation(text):
    import string
    translator = str.maketrans("", "", string.punctuation)
    return text.translate(translator)
data["Message Summary"] = data["Message Summary"].apply(remove_punctuation)
data
import nltk
nltk.download("stopwords")
sw = stopwords.words('english')
np.array(sw)
def stopwords(text):
    text = [word.lower() for word in text.split() if word.lower() not in sw]
    return " ".join(text)
data["Message Summary"] = data["Message Summary"].apply(stopwords)
data
stemmer = SnowballStemmer("english")
def stemming(text):
    """a function which stems each word in the given text"""
    text = [stemmer.stem(word) for word in text.split()]
    return " ".join(text)
data["Message Summary"] = data["Message Summary"].apply(stemming)
data
tfidf_vectorizer = TfidfVectorizer(stop_words="english")
tfidf_vectorizer.fit(data["Message Summary"])
tfidf_matrix = tfidf_vectorizer.transform(data["Message Summary"])
array = tfidf_matrix.todense()
df = pd.DataFrame(array)
df.head()

```

```

df['ID'] = data['Event ID']
df['Label'] = data['Label']
df["Subcategory"] = data["Subcategory"]
df.head()
df.to_csv('Tfid.csv')
tf_data=pd.read_csv("/content/Tfid.csv")
features = tf_data.iloc[:,0:408]
Label=tf_data["Label"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, Label, test_size=0.3,
random_state=1)
print(X_train.shape)
print(y_train.shape)
X_train
X_train=X_train.drop('Unnamed: 0', axis=1)
X_test=X_test.drop('Unnamed: 0', axis=1)
X_train
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, log_loss
from sklearn.model_selection import GridSearchCV
*****Multinomial Naive bayes*****
alpha_list1 = np.linspace(0.006, 0.1, 20)
alpha_list1 = np.around(alpha_list1, decimals=4)
alpha_list1
parameter_grid = [{"alpha":alpha_list1}]
classifier1 = MultinomialNB()
gridsearch1 = GridSearchCV(classifier1,parameter_grid, scoring = 'neg_log_loss',
cv = 4)
gridsearch1.fit(X_train, y_train)
results1 = pd.DataFrame()
# collect alpha list
results1['alpha'] = gridsearch1.cv_results_['param_alpha'].data

```

```

# collect test scores
results1['neglogloss'] = gridsearch1.cv_results_['mean_test_score'].data
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
plt.plot(results1['alpha'], -results1['neglogloss'])
plt.xlabel('alpha')
plt.ylabel('logloss')
plt.grid()
print("Best parameter: ",gridsearch1.best_params_)
print("Best score: ",gridsearch1.best_score_)
model_MNB=gridsearch1.best_estimator_
model_MNB
MNBpreds=model_MNB.predict(X_test)
print(accuracy_score(y_test, MNBpreds))
print(classification_report(y_test,MNBpreds, digits=4))

```

B. SCREENSHOTS

```
[ ] scaler = StandardScaler()
X_scaled=scaler.fit_transform(X)

[ ] X_scaled

array([[ 0.62176076,  0.8606912, -0.50070606, ...,  0.02794122,
        -0.02935797,  0.01098068],
       [ 0.96790907,  1.61223949,  0.11425751, ...,  0.02794122,
        -0.02935797,  0.01098068],
       [-0.24361003, -0.26663124,  2.1254341, ...,  0.34893904,
        -0.02935797,  0.01098068],
       ...,
       [-0.58975834,  0.1091429, -0.9500754, ...,  0.34893904,
        -0.02935797,  0.01098068],
       [-0.7628325,  0.1091429, -0.69741714, ...,  0.02794122,
        -0.02935797,  0.01098068],
       [-0.58975834,  0.1091429, -0.9335003, ..., -0.29305661,
        -0.02935797,  0.01098068]])

[ ] n= 20
from sklearn.decomposition import PCA

pca = PCA(n_components=n)
X1p = pca.fit_transform(X)
pca.explained_variance_ratio_
scaler = StandardScaler()
Xp=scaler.fit_transform(X1p)
```

Fig. B.1: Applying PCA dimensionality reduction.

```
[ ] X_train
```

Unnamed: 0	0	1	2	3	4	5	6	7	8	...	397	398	399	400	401	402	403	404	405	ID
274	274	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5379
814	814	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	6417
160	160	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5379
766	766	0.0	0.0	0.0	0.227429	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5462
642	642	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.334092	0.0	0.0	0.0	4948
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
715	715	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5125
767	767	0.0	0.0	0.0	0.320926	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5463
72	72	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5379
235	235	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5379
37	37	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0.0	5379

575 rows x 408 columns

```
[ ] X_train=X_train.drop('Unnamed: 0', axis=1)
```

Fig. B.2: Converting Log data into TF-IDF vector.

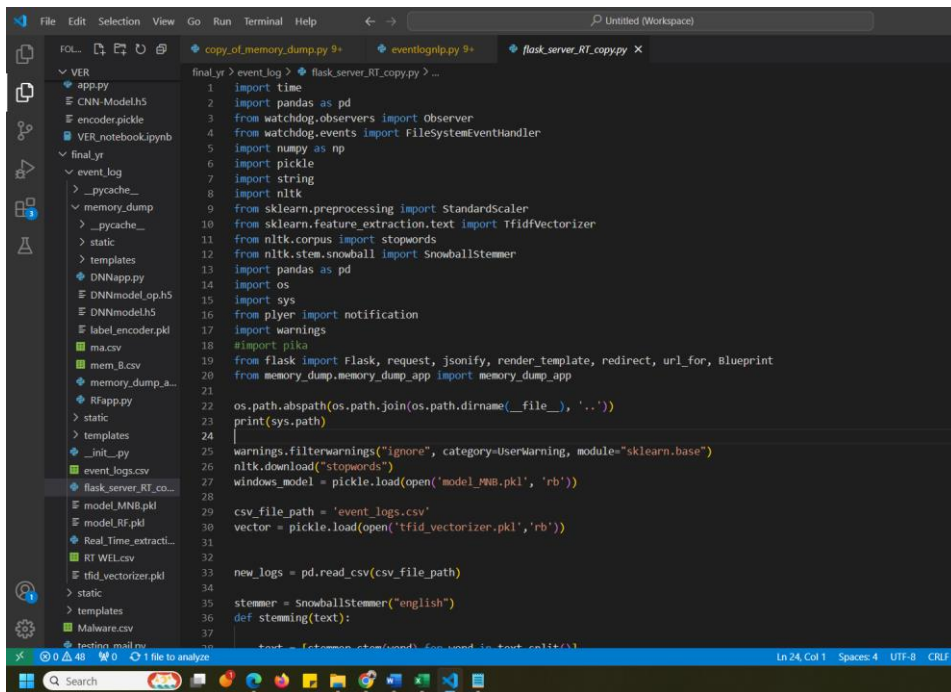


Fig. B.3: Importing the packages in Flask

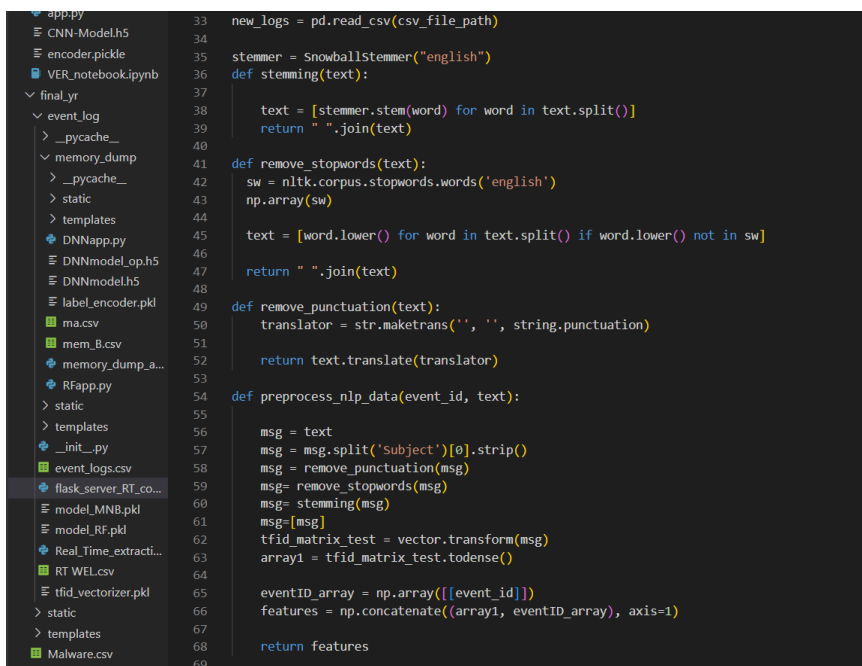


Fig. B.4: using NLP on the event logs.

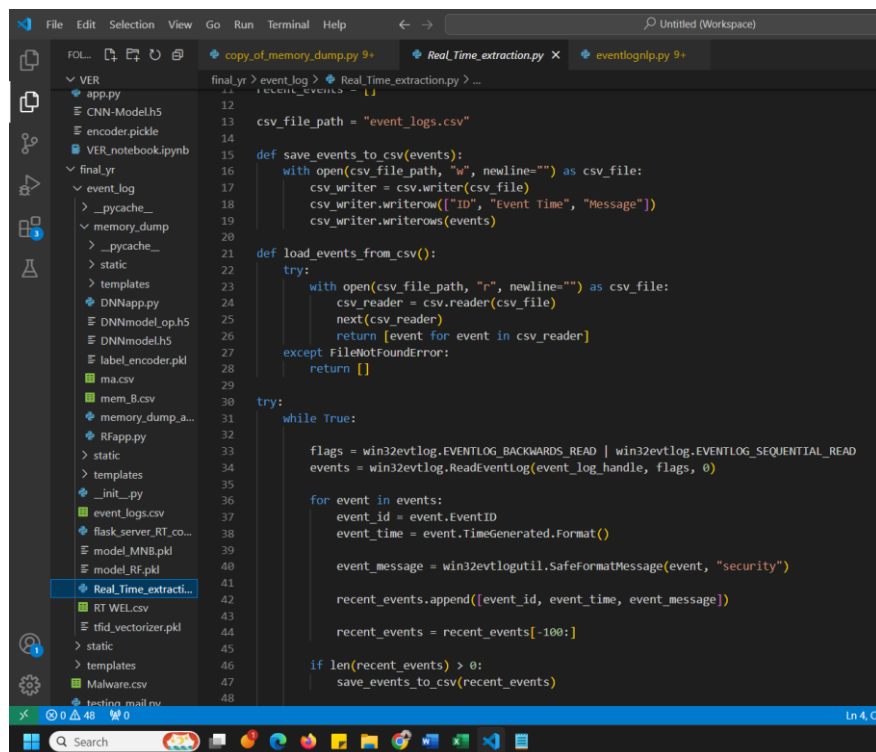


Fig. B.5: Real time extraction of Event logs

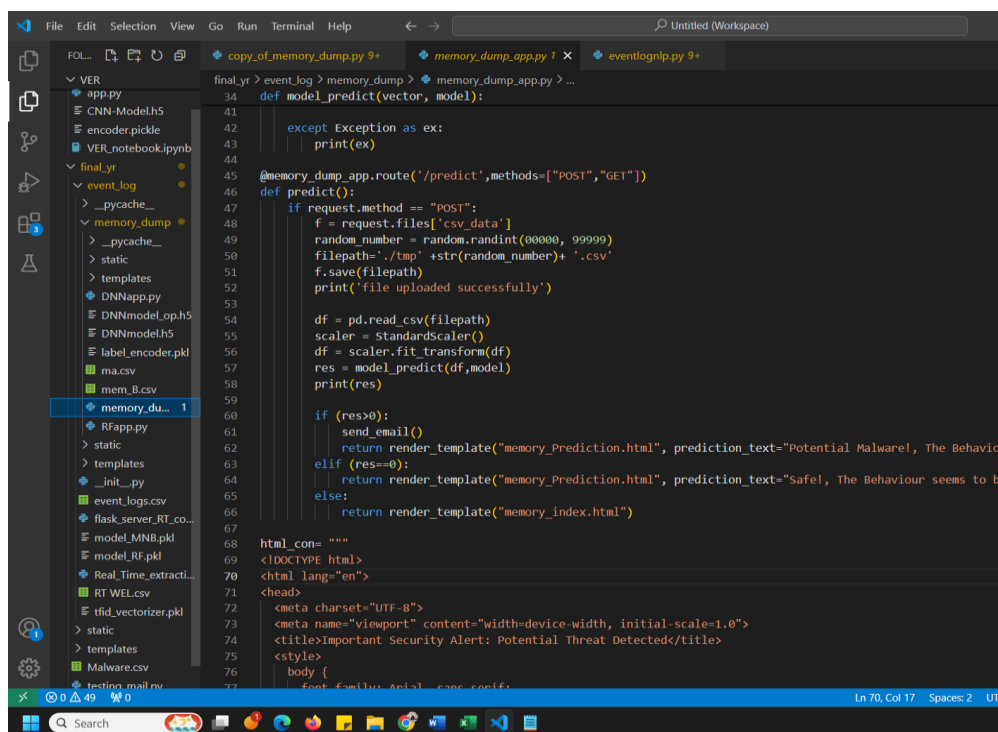


Fig. B.6: Deploying DNN model.

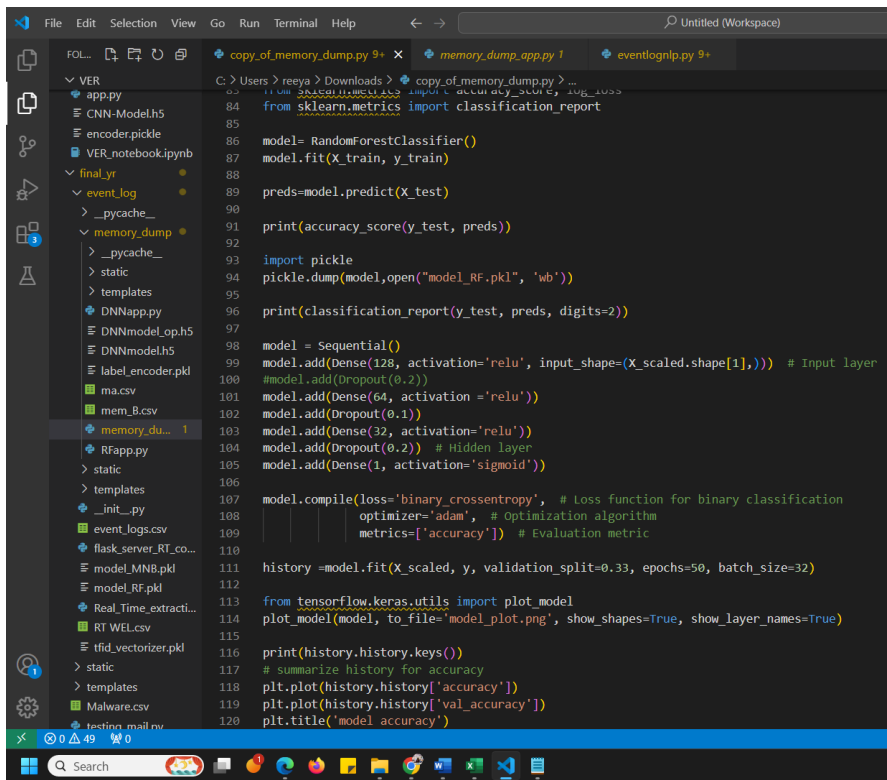


Fig. B.7: creating the DNN model

<pre> model.add(Dense(32, activation='relu')) model.add(Dropout(0.2)) # Hidden layer model.add(Dense(1, activation='sigmoid')) </pre>	
<pre> model.compile(loss='binary_crossentropy', # Loss function for binary classification optimizer='adam', # Optimization algorithm metrics=['accuracy']) # Evaluation metric </pre>	
<pre> history =model.fit(X_scaled, y, validation_split=0.33, epochs=50, batch_size=32) </pre>	
<pre> 1216/1216 [=====] - 5s 4ms/step - loss: 0.0013 - accuracy: 0.9997 - val_loss: 0.0306 - val_accuracy: 0.9966 Epoch 5/50 1216/1216 [=====] - 4s 3ms/step - loss: 3.8489e-04 - accuracy: 0.9999 - val_loss: 0.0323 - val_accuracy: 0.9961 Epoch 6/50 1216/1216 [=====] - 4s 4ms/step - loss: 9.7278e-04 - accuracy: 0.9998 - val_loss: 0.0597 - val_accuracy: 0.9968 Epoch 7/50 1216/1216 [=====] - 5s 4ms/step - loss: 3.4563e-04 - accuracy: 1.0000 - val_loss: 0.0621 - val_accuracy: 0.9967 Epoch 8/50 1216/1216 [=====] - 5s 4ms/step - loss: 3.1483e-04 - accuracy: 0.9999 - val_loss: 0.0557 - val_accuracy: 0.9968 Epoch 9/50 1216/1216 [=====] - 4s 3ms/step - loss: 8.5995e-04 - accuracy: 0.9998 - val_loss: 0.0448 - val_accuracy: 0.9967 Epoch 10/50 1216/1216 [=====] - 4s 3ms/step - loss: 2.5236e-04 - accuracy: 0.9999 - val_loss: 0.0437 - val_accuracy: 0.9967 Epoch 11/50 1216/1216 [=====] - 5s 4ms/step - loss: 9.2380e-04 - accuracy: 0.9998 - val_loss: 0.0338 - val_accuracy: 0.9969 Epoch 12/50 1216/1216 [=====] - 4s 3ms/step - loss: 6.3716e-05 - accuracy: 1.0000 - val_loss: 0.0382 - val_accuracy: 0.9969 Epoch 13/50 1216/1216 [=====] - 3s 3ms/step - loss: 1.0385e-05 - accuracy: 1.0000 - val_loss: 0.0471 - val_accuracy: 0.9968 Epoch 14/50 1216/1216 [=====] - 3s 3ms/step - loss: 1.2671e-04 - accuracy: 1.0000 - val_loss: 0.0336 - val_accuracy: 0.9969 </pre>	

Fig. B.8: Training the Model



Fig. B.9: Plotting Accuracy Curves of DNN Model

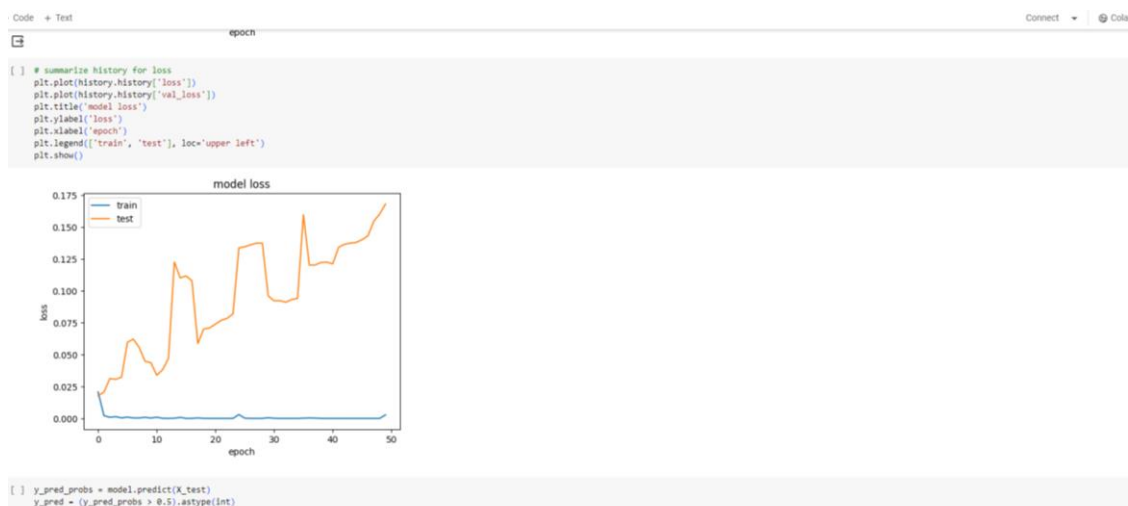


Fig. B.10: Plotting Loss Curves of DNN Model

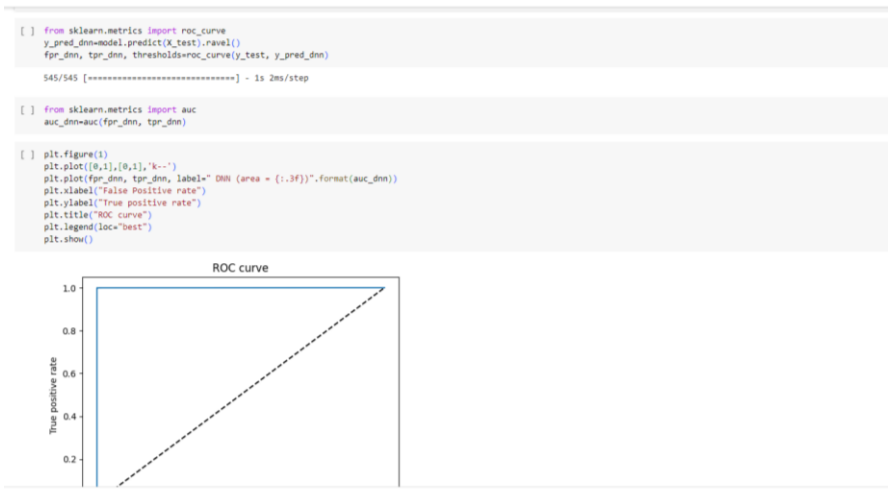


Fig. B.11: Plotting ROC curve

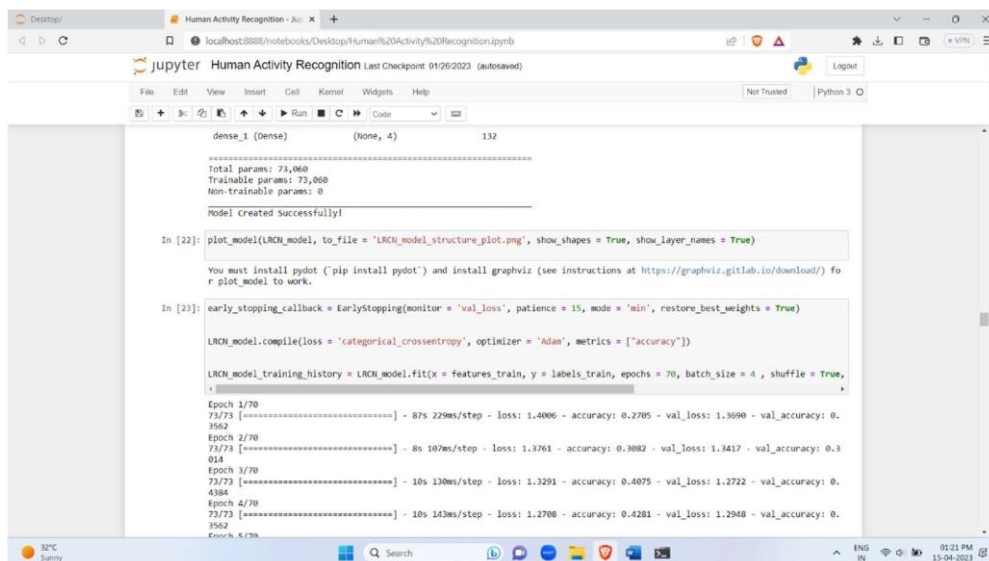


Fig. B.12: Plotting the Model

```

+ Code + Text
[] from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, log_loss
from sklearn.model_selection import GridSearchCV

Multinomial Naive Bayes

[] alpha_list1 = np.linspace(0.000, 0.1, 20)
alpha_list1 = np.around(alpha_list1, decimals=4)
alpha_list1

array([0.000, 0.0100, 0.0150, 0.0200, 0.0250, 0.0300, 0.0350, 0.0400,
       0.0450, 0.0500, 0.0550, 0.0600, 0.0650, 0.0700, 0.0750, 0.0800,
       0.0850, 0.0900, 0.0950, 0.1    ])

[] parameter_grid = [{"alpha":alpha_list1}]

classifer1 = MultinomialNB()
gridsearch1 = GridSearchCV(classifer1,parameter_grid, scoring = 'neg_log_loss', cv = 4)
gridsearch1.fit(X_train, y_train)

GridSearchCV
- estimator: MultinomialNB
- MultinomialNB

[] results = pd.DataFrame()
# collect alpha list
results['alpha'] = gridsearch1.cv_results_['param_alpha'].data
# collect test scores
results['neglogloss'] = gridsearch1.cv_results_['mean_test_score'].data

```

Fig. B.13: Training the multinomial Naive Bayes model.

```

+ Code + Text
1 frames
/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in _input_request(self, prompt, ident, parent, password)
893     except KeyboardInterrupt:
894         # re-raise KeyboardInterrupt, to truncate traceback
895         raise KeyboardInterrupt("Interrupted by user") from None
896     except Exception as e:
897         self.log.warning("Invalid Message:", exc_info=True)

KeyboardInterrupt: Interrupted by user

[] array1.shape

[] eventID_array = np.array([eventID]), dtype='float64')
features= np.concatenate((array1, eventID_array), axis=1)

[] df=pd.DataFrame(features)
df

0 1 2 3 4 5 6 7 8 9 ... 397 398 399 400 401 402 403 404 405 406
0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 ... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 4616.0
1 rows x 407 columns

[] pred=model_NB.predict(df)

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but MultinomialNB was fitted with feature names
warnings.warn(

[] if pred== 0:
    print("normal activity")
else:
    print("abnormal activity")

abnormal activity

```

Fig. B.14: Unit Testing of Event Logs model.

C. RESEARCH PAPER

Event Logs and Memory Dump Analysis for File-less Malware Detection using Deep Neural Networks

Reeya B Varghese

Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
reeyavar28@gmail.com

Thirisha Arul

Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
thirishaarul03@gmail.com

Sakthi Vinayak

Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
sakthi9800@gmail.com

S. Sageengrana

Department of Information Technology
Sathyabama Institute of Science and Technology
Chennai, India
sageengranadhas@gmail.com

Abstract—Fileless malware, a form of stealthy malicious software that has emerged since 2017, has gained notoriety due to its ability to evade detection by anti-virus software through the utilization of various persistence and evasion strategies. This paper proposes a novel method which makes use of windows event logs and memory dump data of windows processes to detect the presence of file-less malware. Event logs are structured files which contain information about the events or processes running in the system. Memory dump is a memory forensic technique that can extract all the information from the RAM of our computer and make a copy out of it in our local disk. This study uses machine learning algorithms to capture these features and detect the presence of the malware in the system. Through thorough experimental analysis, the proposed methodology achieved high accuracy of 99.93% by combining event logs and memory dump data which have also significantly contributed to the reduction in the false positive rates.

Keywords—Malware analysis, Fileless malware, Event logs, memory dump, memory forensics, Machine learning, deep learning, DNN, NLP, Principal Component Analysis (PCA).

I. INTRODUCTION

The sophistication of detection and prevention systems has been steadily rising, and threat actors are always looking for new ways to sneak into systems undetected. One approach was to make use of the legitimate and safe software that came with the OS. Attackers using this technique mostly operate on the premise of taking advantage of the trust model, which allows them to run applications with the help of a predetermined set of trusted and permitted programs. Malicious software that does not rely on an executable file stored in the endpoint system's file system is known as fileless malware.

A. Fileless Malware Threats

Fileless malware is categorized as AVT (Advanced Volatile Threat), a form of malicious software specifically engineered to minimize or eliminate any trace on a computer system, hence posing challenges for detection. This malware variant is specifically engineered to surreptitiously extract data from the affected system. The malware functions within the volatile memory, specifically the Random Access Memory (RAM), by masquerading as legitimate software while carrying out malicious activities. Due to its limited operation within the volatile memory, this specific type of malware stops functioning when the user's system is turned down. In this specific scenario, the malware may seem less threatening than an Advanced Persistent Threat (APT), which is designed to remain undetected in a network for an extended period of time. The APT malware was previously considered the most hazardous among several sorts of malwares. Nevertheless, AVT has already overtaken it and has gained a reputation for being universally feared. This primarily occurs because malicious actors have started utilizing persistence strategies to avoid being detected and to stay on the system even after the user has powered it down.

B. Persistence in Fileless Threats

The malware achieves persistence by manipulating the registry of the operating system. The registry is a vital element of the operating system that is responsible for the management and storage of configuration settings. This is accomplished by injecting malevolent code directly into the Registry using a dropper program. The malware possesses the capability to generate or alter registry keys in a manner that initiates the malware's execution upon each instance of the

operating system being powered on and operational. In this specific situation, there is no presence of a malicious file. Instead, the harmful code is directly inserted into a legitimate application. Consequently, the virus is able to evade detection for an extended duration.

C. Lifecycle of Fileless Malware

The fileless threat usually begins with the delivery step, in which the hacker attempts to transmit the malicious script to the target machine via various methods. Phishing, a commonly used strategy seen in many cases of malware outbreaks, is highly popular and widespread. The attacker transmits an infected email to the intended victim, which includes malicious scripts, or adopts deceitful strategies to lure the user into visiting a hazardous website. Subsequently, this website initiates the download of a detrimental JavaScript code or a contaminated Microsoft Office document, employing social engineering tactics to trick the user into allowing malicious macros. After gaining unauthorized access to the targeted individual's PC, the virus attempts to alter the registry keys in order to attain persistence. Additionally, it creates a connection with the command-and-control server for the purpose of downloading the malicious script. The payload operates within a legitimate tool, such as PowerShell, and executes pre-programmed malicious actions on the targeted machine, such as the unauthorized extraction of data.

D. Fileless Threats Attack Pattern

Some of the recorded fileless attack techniques are the presence of a malicious script within the Windows Management Instrumentation subscription (WMI), the direct inclusion of a malicious script as a command line parameter in PowerShell, the storage of a malicious script within the registry, and/or the operating system scheduler task, and the execution of the said script by the operating system scheduler, as well as the extraction and direct execution of a malicious executable in memory without being saved on disc via the use of a .Net reflection technique, among other methods, is observed in academic research.

E. Need for IDS specially for Fileless Threats

The efficacy of traditional anti-virus software in detecting and mitigating fileless threats has been called into question. These software programs are designed to scan and monitor the hard disc of a system, rendering them ineffective in identifying fileless malware. This is primarily due to the fact that fileless malware does not leave any discernible trace or footprint within the system, making it impervious to detection by signature based antivirus software. In addition, fileless malware capitalizes on the utilization of established, authorized programs within the operating system that typically evade detection from anti-virus software. The reduced footprint and absence of "foreign" executables to scan provide challenges for conventional antivirus (AV) and other endpoint security solutions in detecting or thwarting such forms of assaults.

II. LITERATURE SURVEY

Peng Chen et al [22] introduced an innovative preprocessing mechanism for Deep Neural Networks (DNNs) used in intrusion detection. The research highlights that the primary challenge in Intrusion Detection Systems (IDS) lies in effectively detecting advanced persistent threats (APTs), which are now inadequately recognized. The authors present a DNN model for intrusion detection and introduce an innovative preprocessing technique to enhance the effectiveness of detection techniques. The preprocessing methodology takes into account the statistical properties of the training set, test set, and detection set. The suggested preprocessing methodology comprises three essential steps: numeralization, transformation, and normalization. Numeralization is employed to handle symbolic data that cannot be directly processed by a deep neural network (DNN). Transformation entails the process of assigning a unique integer value to each value within a range of 1 to the total number of distinct values. Normalization improves the performance of the DNN structure by utilizing predetermined minimum and maximum values as global minimum and maximum values. The preprocessed data is given as input into the DNN model for detection. Upon detecting an attack, an alert is promptly dispatched. The proposed preprocessing methodology can consistently produce superior accuracy, irrespective of the number of Deep Neural Network (DNN) layers. Through a comparative analysis with prior research, the researchers demonstrated that their methodology has the capability to enhance accuracy, memory, and the F1 score while maintaining precision at a satisfactory level. Their methods exhibited superior performance compared to standard machine learning algorithms, as evidenced by the comparisons. To ensure the efficacy of their preprocessing methods, it is recommended to test additional datasets in future work. Additionally, the identification process should take into account the occurrence of idea drift.

Rachana Koniki et al [21] emphasize on the need of promptly detecting cyber-attacks to minimize network disruptions and protect user privacy. It specifically examines the difficulties involved in implementing intrusion detection systems on a wide scale, particularly the lack of labeled data for training and validating models. The deep learning model was trained and validated using the NSL-KDD dataset. The dataset is divided into training and validation subsets, with 67% and 33% of the data allocated to each subset, respectively. The proposed methodology consists of three main sections: data discovery, label aggregation, and training and testing. The initial module involves comprehensive data exploration and label aggregation, entailing the assignment of column names to both training and testing sets. Subsequently, the second module incorporates Autoencoder for data preparation and feature extraction, utilizing 'MinMaxScaler' and 'OneHotEncoder' for data pre-processing. One Hot Encoding is deployed to transform categorical data into numerical form via binary variables, while the MinMax scaler standardizes features by mapping them to predetermined values or within specified ranges. The study specifically investigates the utilization of LSTM and GRU in sequence modeling techniques, with a particular focus on

their application in network intrusion detection. Notably, the Probe attack demonstrates the highest precision among all attack forms, achieving an accuracy rate of 95%. R2L assaults closely follow with an accuracy rate of 87%, while conventional attacks achieve a rate of 77%. Furthermore, the study underscores potential avenues for future research, suggesting the exploration of diverse balancing algorithms to enhance existing methodologies and evaluating their impact on model acquisition. Additionally, the incorporation of more sophisticated iterations of the Gated Recurrent Unit and other advanced deep learning methods is recommended for further refinement of the intrusion detection system.

Chang Liu et al [20] introduced a novel approach by integrating an attention mechanism into the HLSTM, thereby proposing the HLSTM+Attention algorithm for intrusion detection. The architecture of HLSTM+Attention consists of two layers: the initial LSTM layer, is responsible for encoding the input sequence's column vector into a feature vector, and the subsequent LSTM layer, which encodes the derived feature vectors into scaled feature vectors. Notably, the size of weight values reflects the significance of features in elucidating the traffic category. Attention weights are assigned to sequence features, and the resultant feature vector is subsequently inputted into a fully connected layer for prediction. Experimental validation entails the allocation of attention weights to sequence characteristics and training the model. The outcomes affirm that the HLSTM+Attention technique effectively mitigates record redundancy and enhances the rationality of partitioning training and testing sets. Comparative analyses of HLSTM+Attention against various intrusion detection methodologies, including CNN with two convolutional layers and LSTM with two hidden layers, reveal the superior prediction accuracy of HLSTM+Attention, attaining 84.55% on the KDDTest+ dataset. Moreover, the study assesses the time complexity of HLSTM+Attention relative to the original HLSTM model. Findings indicate that HLSTM+Attention necessitates 11.04 minutes for training on the KDDTrain+ dataset, whereas the conventional HLSTM model consumes 10.25 minutes. Future research endeavors are slated to delve into and analyze the inherent data imbalance within the dataset, thus addressing an essential aspect of model refinement and performance enhancement. In summary, the incorporation of attention mechanisms into the HLSTM framework for intrusion detection presents a promising avenue, offering notable improvements in prediction accuracy and computational efficiency. The study underscores the significance of attention mechanisms in refining model performance and acknowledges the imperative of addressing data imbalances for robust intrusion detection systems.

S.Javeed Kamal et al [19] delves into the current landscape of Intrusion Detection Systems (IDS) by adopting a deep learning approach, with a specific focus on datasets, DL techniques, and evaluation metrics. The intrusion detection model is crafted using the NSL-KDD dataset, renowned for its standardized collection of diverse assault types categorized as DoS, Probe, U2R, and R2L. To ensure equal proficiency across all attack classes, a label encoder and random sampling technique are employed during model

training. The IDS model comprises three distinct neural network architectures: CNN, ANN, and LSTM. ANNs are hailed as universal function approximators due to their ability to learn and approximate any nonlinear function. Meanwhile, the LSTM model incorporates a recurrent link, facilitating information transmission across different time steps with shared parameters. The project's primary objective is to bolster accuracy and minimize errors in intrusion detection through the utilization of ANN and LSTM models. Notably, the LSTM-IDS model outshines other classifiers by achieving maximum performance with a reduced error rate, leveraging feature extraction techniques from the dataset. With an impressive accuracy of 97.36%, a detection rate of 98.64%, and a notably low false positive rate of 6.83%, the LSTM-IDS model demonstrates superior performance compared to the ANN. While the ANN achieves a commendable accuracy of 94.13% and a detection rate of 97.24%, it also exhibits a relatively high false-positive rate of 16.47%. In overall assessment, the LSTM surpasses the current CNN approach in terms of effectiveness and operational efficiency, particularly for real-time detection scenarios. The empirical findings on the dataset underscore the efficacy of the LSTM-based IDS model. Future research endeavors will prioritize enhancing precision through the integration of hybrid models and genetic algorithms, aiming to achieve real time identification and prevention of intrusions while maintaining high accuracy levels.

D.P. Gaikwad et al [18] have designed an innovative intrusion detection system leveraging a machine learning ensemble method known as bagging. The system aims to enhance accuracy, diminish false positive rates, and streamline development time for the model through the implementation of the Bagging approach. It consists of five integral modules: Feature selection, REPTree design, Primary Classifier Construction, Packet sniffer, and Detector. By utilizing the preprocessed NSL_KDD dataset and focusing on 29 pertinent features for training the Bagging model, the system effectively reduces false positives and improves categorization accuracy, thanks to the REPTree algorithm. Operating in distinct training and testing modes, the system constructs the model during the training phase and identifies online packets during testing. Through preprocessing, captured packets are refined to select relevant features before input into the model. Upon detecting intruder packets, the system promptly notifies the network administrator while allowing regular packets to pass through. Any unidentified packets are meticulously logged for subsequent analysis. Evaluation of the approach entails rigorous 10-fold cross-validation and comparison against established machine learning techniques. Notably, the ensemble bagging technique achieves an impressive accuracy of 99.67% in cross-validation and 81.29% on the test-dataset. Important factors revolve around the time it takes to build the model and the occurrence of incorrect positive results. In summary, the proposed ensemble approach showcases superior efficiency and attains high classification accuracy while effectively minimizing false positives. By leveraging the Bagging method and prioritizing feature selection and algorithm design, the system emerges as a robust solution for intrusion detection in modern network environments.

Pengtian Chen et al [17] developed an intrusion detection model employing bagged trees, a method demonstrated through empirical analysis to yield a decreased false alarm rate and heightened accuracy. The research leverages the UNSW-NB15 dataset, renowned for its comprehensive coverage of various attack methods, effectively capturing the intricacies of modern network environments. Encompassing 49 dimensions, the dataset encompasses nine distinct categories of attacks. The bagging algorithm, employed as a classification technique, utilizes Bootstrap sampling on the original training set to generate multiple sample sets. These sets are subsequently utilized to train diverse base classifiers, following which a combination strategy is applied to integrate these classifiers into a unified model. Notably, the algorithm adopts a voting mechanism to determine the final category, predicated on the principle that the majority prevails or when votes surpass fifty percent. To ensure the accuracy and consistency of the intrusion detection model, the study employs cross-validation to partition the data into distinct training and testing sets. Utilizing a 10-fold cross-validation approach, the mean of the ten outcomes serves as a surrogate for the algorithm's accuracy. The experimental assessment compares the proposed method with alternative network intrusion detection models, employing an evaluation index to gauge performance. Results demonstrate that the intrusion detection model based on bagged trees showcases superior accuracy and a decreased false alarm rate in comparison to other machine learning classifier algorithms utilized in intrusion detection. Furthermore, it ranks second in terms of both detection rate and the rate of missed alarms, underscoring its efficacy in detecting and mitigating network intrusions. In summary, the study underscores the efficacy of employing bagged trees in intrusion detection, highlighting its superiority over alternative methods and its potential ways to enhance network security in modern environments.

III. PROPOSED METHODOLOGY

This section describes the various methodologies implemented in the study. We have made use of two datasets: Windows event logs dataset and “CIC-Malmem-2022” Malware memory dump dataset by the Canadian institute for cybersecurity, to achieve our objectives. The two datasets undergo the phases of pre-processing, training, testing, and evaluation of ML algorithms separately and finally the best performing ML models are integrated in the deployment phase.

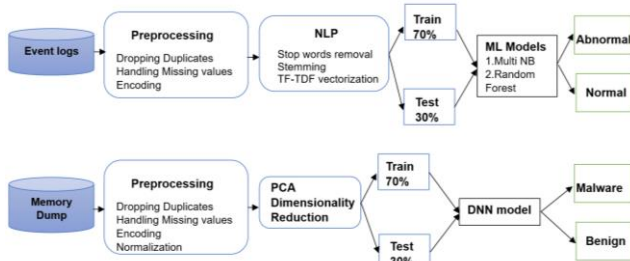


Fig 1: Architecture of the proposed methodology

A. Dataset

This study uses two datasets for the proposed methodology. Firstly, Windows event logs data which is a record of important hardware and software actions. This information helps users, network administrators to troubleshoot issues and also forecast future issues. As we are dealing with malware activities which cause escalation of privileges in the system, we have used Security event logs data. Benign files were collected from our windows system using the shell command Get-WinEvent in the Microsoft PowerShell and these records were exported into a csv file. The malicious activity event logs were taken from the , Both the benign and malicious event logs were combined into a single csv file. The dataset contains the following features: Event ID, Timestamp, Level, Task category, and message summary. We created a balanced dataset with 50% malicious activity records and 50% benign activity records. The exact records are 500 benign and 500 malicious, this dataset will be referred to as the event logs dataset henceforth. Secondly, an obfuscated malware dataset from CIC-Malmem-2022 was used for this study. This dataset uses debug mode for the memory dump process. The dataset is designed to detect malware through memory forensic analysis. This dataset consists of data from three malware families: a) Ransomware b) Spyware and c) Trojan Horse. There are 57 features in this dataset which outlines the details of various processes in the memory. This dataset is also balanced with 50% malicious memory dumps and 50% benign memory dumps. The dataset contains a total of 58,596 records with 29,2928 benign and 29,298 malicious samples, this dataset would be referred as memory dump dataset henceforth.

B. Pre- Processing

The event logs dataset and the memory dump dataset undergo various processes in this phase to remove unwanted elements in the data like outliers, and missing values, and transform it into a standard format that is compatible with chosen ML methods.

a) Event Logs dataset preprocessing:

The event log dataset has 4 features: event ID, which is an identification number of the specific event type, subcategory indicating the recorded event type, and message summary describing the event that has been flagged. The benign records were appended with the label as normal and the malicious files were appended with the label abnormal in the dataset. The dataset is checked for any missing values or duplicate values and they are eliminated from the dataset. In the next step dataset independent feature which is the “label” is encoded as 1 for abnormal and 0 for normal using the `df.replace()` command in pandas. Following this, we implement natural language processing (NLP) to extract valuable features. The message summary column in the dataset is a textual description of the event. Valuable information can be extracted from this by converting it into a vector format that the machine learning models can understand. Firstly the punctuations and stopwords in the text data are removed using `nlTK.stopwords` package in Python. Secondly, we use stemming techniques to reduce the words in the text to their root word so that the ML models can treat

them as a single entity and rescue the complexity. Finally, the textual data is transformed into a vector format using the Term Frequency - Inverse Document Frequency (TF-IDF). The array contains 405 vector features to which the event ID, subcategory, and label columns are appended contributing to 408 features and 822 columns after the preprocessing phase.

b) Memory Dump dataset preprocessing :

The memory dump dataset contains 57 features which gives us a detailed overview of the various processes running in the RAM of the system. The records are encoded using label encoding of the categorical column. The benign records were encoded as 0 and the malicious records as 1 in the dataset. Missing values and duplicate values are removed from the dataset using the pandas library. The numerical features are normalized using the standard scaler function from the sklearn preprocessing library of python. It scales each value to unit variance. Owing to the complexity of the features in the dataset, we have employed Principle Component Analysis (PCA) to achieve dimensionality reduction. It is extensively used to find the highly correlated features in the dataset and remove them, finally producing a new set of features with least correlation between them. Through this we can select the features that contribute the most to the learning of the model and thereby increase the efficiency of the model. We have reduced the features set to 20 features through PCA which have proved to improve the efficiency of the model.

C. Machine Learning Model Training

a) Event log ML pipeline :

We have chosen two ML models to be trained on the event log data namely Multinomial naive Bayes and random forest classifier. It predicts the tag of the text based on conditional probability and the assumption that the features are independent of each other. The dataset is split into 2 sets; 70% of the data were to be utilized for the training phase and the rest 30% data were to be utilized for the testing phase. Hyperparameter tuning on the hyper-parameter alpha is performed on the multinomial naive Bayes model using 4-fold cross-validation using the scoring metric negative log loss. The best value for the hyperparameter alpha is found to be 0.0406. The multinomial naive Bayes model is trained on the train data and produces an accuracy of 99.39% on test data. The other ML model used in this study is the Random forest classifier model. The Gini index is used to find the purity of the classes and the number of decision trees is decided by hyperparameter tuning using 3-fold cross-validation. The best-performing hyperparameter is 75 decision trees and we train the model with training data and the accuracy achieved is 98.08%.

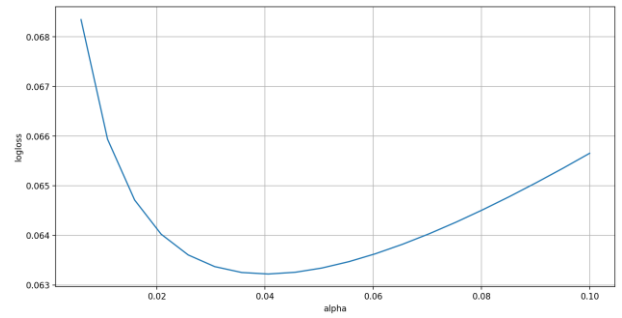


Fig 2: Selecting the best value of alpha for Multinomial Naive Bayes using hyper-parameter tuning.

b) Memory Dump DNN pipeline :

This study uses Deep neural networks (DNN) for training the memory dump dataset. DNN is a neural network with stacked layers. We have implemented a sequential model with 1 input layer of size 128 with activation Relu, and 2 hidden layers of size 64 and size 32 with activation Relu, and final output layer with size 1 and activation Sigmoid. This model produced an accuracy of 99% but the loss was high . To reduce the loss and prevent the model from overfitting we have added 3 dropout layers with 20%, 30% 40% dropout rate to each layer. This has significantly reduced the loss rate and produced the final accuracy of 99.93% compacting overfitting.

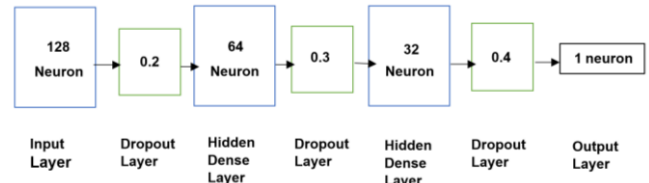


Fig 3: Structure of the deep neural network

IV. RESULT AND EVALUATION

We have compared the results of the ML and DL models used for event logs dataset and memory dump dataset using various metrics like confusion matrix, F1- score, Precision, recall and Accuracy score. It is a matrix consisting of four variables, which provides us the detection capacity of the model as True negative (TN) which are the correctly detected benign records, True positives (TP) which are the correctly detected malware records, False positives (FP) which are incorrectly detected benign records, and False negatives (FN) which are incorrectly recognised malware records. The aim is to reduce the false negative rates (FNRs) in the models. Multinomial naive bayes and DNN with PCA have been found to be reducing the false negative rates. The recall value measures The sensitivity of the model which is also found to be high 98% in Multinomial NB and 99% in DNN - PCA. We use F1 score as an important metric to assess the efficiency of classification of the model. It uses both precision and recall which is found to be 99% in Multinomial NB and 100% in DNN - PCA. We finally calculated the accuracy rate by adding the correctly predicted events and dividing it by the

total events. The accuracy of Multinomial NB is 99.1% and DNN - PCAis found to be 99.8%.

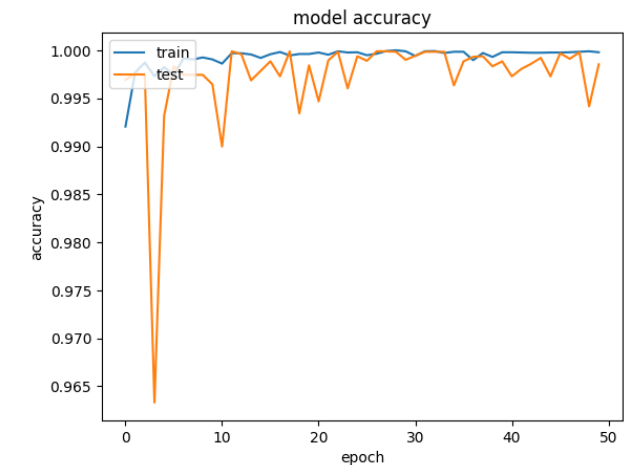


Fig 4: DNN accuracy graph before adding the dropout layers.

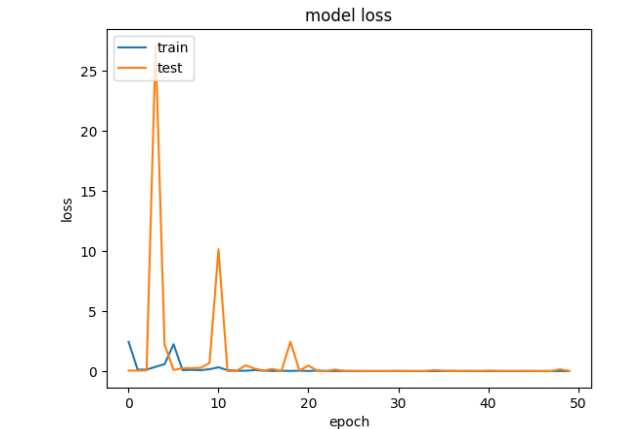


Fig 5: DNN loss graph before adding the dropout layers.

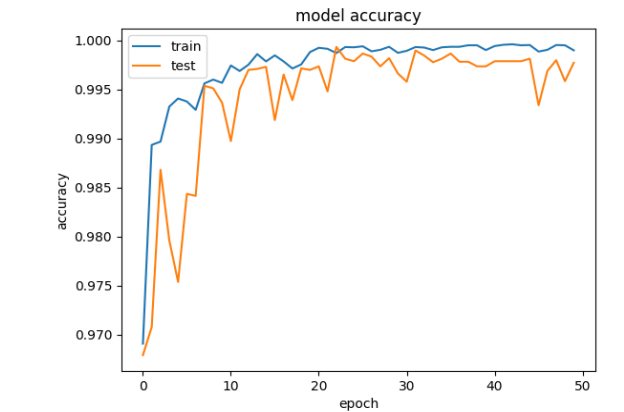


Fig 6: DNN accuracy after adding the dropout layers.

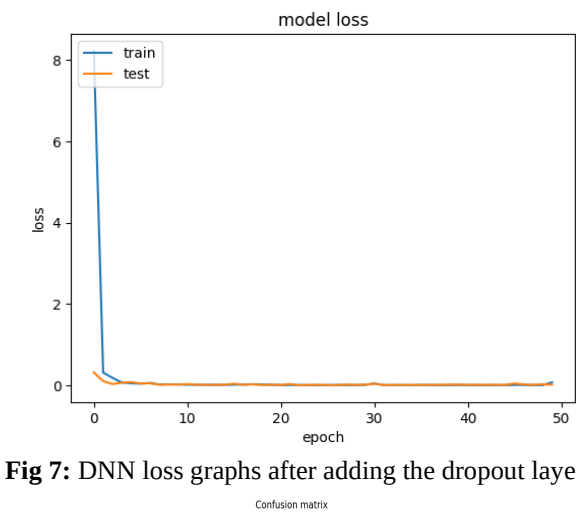


Fig 7: DNN loss graphs after adding the dropout layers.

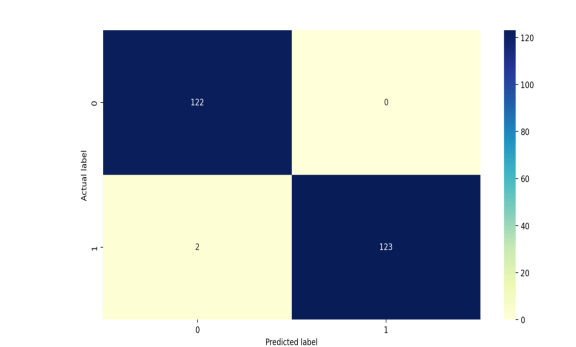


Fig 8: Confusion Matrix of Multinomial Naive Bayes

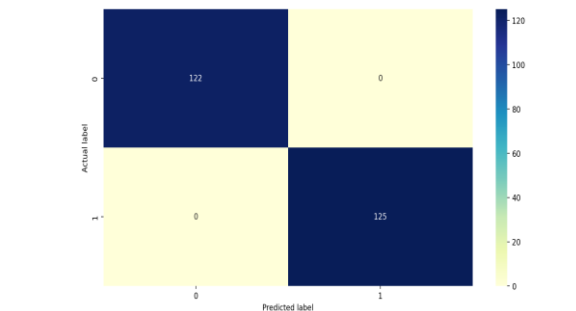


Fig 9: Confusion Matrix of Random Forest.

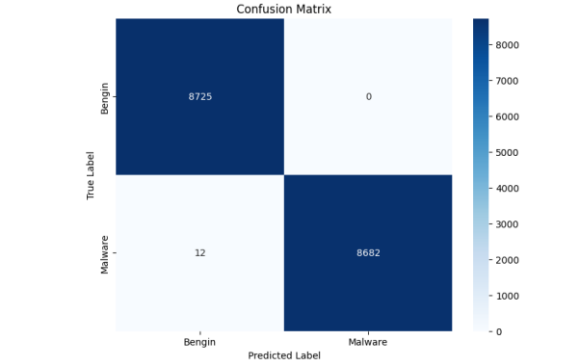


Fig 10: Confusion Matrix of Deep Neural Network (DNN - PCA).

TABLE I. Models Results in Comparison.

<i>Model</i>	<i>Precision</i>	<i>Recall</i>	<i>F1 Score</i>	<i>Accuracy</i>
Multinomial Naive Bayes - EVT	98.3%	98.4%	99.1%	99.1%
Random Forest - EVT	99.4%	99.93%	100%	100%
DNN- PCA - memory dump	98%	99.1%	100%	99.93%

V. CONCLUSION AND FUTURE WORK

This paper presents two pipelines leveraging machine learning and deep learning classifiers to detect abnormal activities indicative of malware presence within a system. By analyzing Windows event logs and memory dump data, these pipelines offer a robust approach to identifying and mitigating potential threats. Our utilization of deep neural networks for malware detection has demonstrated superior performance compared to existing models outlined in the literature. Additionally, employing the Random Forest model to identify event activities leading to privilege escalation has further enhanced the efficacy of our detection mechanisms. The results highlight the effectiveness of these pipelines in identifying and mitigating malware activity, offering a valuable contribution to the field of cybersecurity. Moving forward, the findings of this study could be leveraged to develop standalone software solutions dedicated to monitoring file-less malware activity, thereby enhancing system security. Moreover, integration into existing Intrusion Detection Systems presents an opportunity to bolster overall protection against malicious threats, providing a comprehensive defense mechanism for safeguarding critical systems and data.

REFERENCES

- [1] Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," 2020 International Conference on Cyber Warfare and Security (ICWS), Islamabad, Pakistan, 2020, pp. 1-8, doi: 10.1109/ICWS48432.2020.9292376.
- [2] Sudhakar, Kumar, S. An emerging threat Fileless malware: a survey and research challenges. *Cybersecur* 3, 1 (2020). <https://doi.org/10.1186/s42400-019-0043-x>
- [3] B. N. Sanjay, D. C. Rakshith, R. B. Akash and D. V. V. Hegde, "An Approach to Detect Fileless Malware and Defend its Evasive mechanisms," 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 2018, pp. 234-239, doi: 10.1109/CSITSS.2018.8768769.
- [4] P. Borana, V. Sihag, G. Choudhary, M. Vardhan and P. Singh, "An Assistive Tool For Fileless Malware Detection," 2021 World Automation Congress (WAC), Taipei, Taiwan, 2021, pp. 21-25, doi: 10.23919/WAC50355.2021.9559449
- [5] O. Khalid et al., "An Insight into the Machine-Learning-Based Fileless Malware Detection," *Sensors*, vol. 23, no. 2, p. 612, Jan. 2023, doi:10.3390/s23020612.[Online].Available:<http://dx.doi.org/10.3390/s23020612>

- [6] S. Mele Pottaraikkal and A. Sujeer Sugatha, "Effectiveness of Multiple Memory-Images in detecting Fileless Malware," 2023 11th International Symposium on Digital Forensics and Security (ISDFS), Chattanooga, TN, USA, 2023, pp. 1-5, doi: 10.1109/ISDFS58141.2023.10131728.
- [7] S. Zhang, C. Hu, L. Wang, M. J. Mihaljevic, S. Xu, and T. Lan, "A Malware Detection Approach Based on Deep Learning and Memory Forensics," *Symmetry*, vol. 15, no. 3, p. 758, Mar. 2023, doi: 10.3390/sym15030758. [Online]. Available: <http://dx.doi.org/10.3390/sym15030758>
- [8] M. -H. Tsai, C. -C. Lin, Z. -G. He, W. -C. Yang and C. -L. Lei, "PowerDP: De-Obfuscating and Profiling Malicious PowerShell Commands with Multi-Label Classifiers," in *IEEE Access*, vol. 11, pp. 256-270, 2023, doi: 10.1109/ACCESS.2022.3232505
- [9] "JSLess: A Tale of a Fileless JavaScript Memory-Resident Malware," *springerprofessional.de*.Available:<https://www.springerprofessional.de/en/jsless-a-tale-of-a-fileless-javascript-memory-resident-malware/17402116>
- [10] R. Bearden and D. C. -T. Lo, "Automated Microsoft Office macro malware detection using machine learning," 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 2017, pp. 4448-4452, doi: 10.1109/BigData.2017.8258483.
- [11] S. Kim, S. Hong, J. Oh and H. Lee, "Obfuscated VBA Macro Detection Using Machine Learning," 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Luxembourg, Luxembourg, 2018, pp. 490-501, doi: 10.1109/DSN.2018.00057.
- [12] A. More, K. Joshi, and K. Kumar, "Simulation Framework for Fileless Malware Detection and Analysis," *Simulation Framework for Fileless Malware Detection and Analysis by Aditya More, Kashyap Joshi, Kapil Kumar :: SSRN*, Jul. 12, 2023. Available: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4488146
- [13] M. U. Rana, M. Ali Shah and O. Ellahi, "Malware Persistence and Obfuscation: An Analysis on Concealed Strategies," 2021 26th International Conference on Automation and Computing (ICAC), Portsmouth, United Kingdom, 2021, pp. 1-6, doi: 10.23919/ICAC50006.2021.9594197
- [14] Botacin, M., Grégio, A., & Alves, M. A. Z. (2020, September). Near-memory & in-memory detection of fileless malware. In *The International Symposium on Memory Systems* (pp. 23-38).
- [15] Nuangpookka, P., Mengistu, Z., & Bafail, G. (2020). Fileless malware and programmatic method of detection. *Journal of Computing Sciences in Colleges*, 36(3), 161-162.
- [16] Krishna, B. L. (2020). Comparative study of fileless ransomware. *Int. J. Trend Sci. Res. Develop.*, 4, 608-616.
- [17] P. Chen, F. Li and J. Li, "Research on Intrusion Detection Model Based on Bagged Tree," 2021 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS), Shenyang, China, 2021, pp. 579-582, doi: 10.1109/TOCS53301.2021.9688618.
- [18] D. P. Gaikwad and R. C. Thool, "Intrusion Detection System Using Bagging Ensemble Method of Machine Learning," 2015 International Conference on Computing Communication Control and Automation, Pune, India, 2015, pp. 291-295, doi: 10.1109/ICCUBEA.2015.61.
- [19] S. J. Kamal, G. Karishma, S. T. A. Rajesh, V. Muthiah-Nakarajan and B. S, "ANN-LSTM Assisted Intrusion Detection for Next Generation Core Networks," 2023 2nd International Conference on Vision Towards Emerging Trends in Communication and Networking Technologies (ViTECoN), Vellore, India, 2023, pp. 1-5, doi: 10.1109/ViTECoN58111.2023.10157036.
- [20] C. Liu, Y. Liu, Y. Yan and J. Wang, "An Intrusion Detection Model With Hierarchical Attention Mechanism," in *IEEE Access*, vol. 8, pp. 67542-67554, 2020, doi: 10.1109/ACCESS.2020.2983568.

C.1 Certificate of Presentation

