**Project Report on**

**Malware Detection and Analysis**



***Department of Information & Communication Technology***

|  |  |
| --- | --- |
| **Submitted to:**  Alimul Rajee  Lecturer  Dept. of ICT  Comilla University | **Submitted by:**  Md. Khayrul Islam  ID: 11909038  Dept. of ICT  Comilla University |

**Submission Date: July, 2023.**

**TABLE OF CONTENTS**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | |  |  |  | | --- | --- | --- | | **CHAPTER** | **TITLE** | **PAGE** | |  | **LIST OF FIGURES** | 5 | |  | **ABSTRACT** | 6 | | **1.** | **INTRODUCTION** | 7 | | 1.1 | Introduction | 7 | | 1.2 | Malware Types | 9 | | 1.3 | Objectives | 11 | | 1.4 | A Brief Outline of the Project | 12 | | **2.** | **LITERATURE REVIEW** | 13 | | 2.1 | Literature Review | 13 | | **3.** | **METHODOLOGY AND MODELING** | 22 | | 3.1 | Datasets | 22 | | 3.2 | Description of the Important Elements | 23 | | 3.2.1 | NumPy | 24 | | 3.2.2 | Pandas | 24 | | 3.2.3 | Scikit-learn | 24 | | 3.2.4 | Tensor Flow | 24 | | 3.3 | Design Process | 25 | | 3.3.1 | Requirement Analysis | 25 | | 3.3.2 | Data Collection | 25 | | 3.3.3 | Feature Extraction | 25 | | 3.3.4 | Model Selection | 25 | | 3.3.5 | Training Phase | 25 | | 3.3.6 | Evaluation and Validation | 26 | | 3.3.7 | Integration and Deployment | 26 | | 3.3.8 | Ongoing Monitoring and Updating | 26 | | 3.3.9 | Collaboration and Information Sharing | 26 | | 3.3.10 | System Maintenance and Enhancement | 26 | | 3.4 | Detection Techniques | 27 | | 3.4.1 | File Format Inspection | 27 | | 3.4.2 | Dynamic Analysis | 28 | | 3.5 | Working Process of Proposed System | 30 | | 3.5.1 | Data Read | 31 | | 3.5.2 | Data Transformation | 31 | | 3.5.3 | Model Training | 31 | | 3.5.4 | Model Testing | 31 | | 3.5.5 | Detection | 31 | | 3.6 | Classification Methods | 31 | | 3.6.1 | K-Nearest Neighbors | 32 | | 3.6.2 | Support Vector Machines | 34 | | 3.6.3 | Naive Bayes | 36 | | 3.6.4 | Decision Tree | 37 | | 3.6.5 | Random Forest | 38 | | 3.7 | Test/Experimental Setup | 39 | | 3.8 | Implementation | 39 | | **4.** | **RESULTS AND DISCUSSION** | 42 | | 4.1 | Performance Evaluation | 42 | | 4.2 | Advantages | 43 | | 4.3 | Disadvantages | 44 | | 4.4 | Applications | 45 | | **5.** | **CONCLUSIONS AND FUTURE WORK** | 47 | | 5.1 | Conclusion | 47 | | 5.2 | Future Work | 48 | |  | **REFERENCES** | 49 | |  | **APPENDIX A** | 54 | |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |  | |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **No.** | **TITLE** | **PAGE** |
| 3.1 | Flow Chart of Proposed System | 30 |
| 3.2 | K-Nearest Neighbors | 34 |
| 3.3 | Support Vector Machines Scheme | 35 |
| 3.4 | Data Level for Malware and Legit | 39 |
| 3.5 | Accuracy for 100 Data | 40 |
| 3.6 | Accuracy for 10000 Data | 40 |
| 3.7 | Accuracy for 50000 Data | 41 |

ABSTRACT

Malware detection was a significant calculate the security of the PC frameworks. Be that as it might, as of now used signature-based techniques could't give precise identification of zero-day assaults and polymorphic infections. To that end the requirement for AI based discovery emerges. The motivation behind this work was to decide the best component extraction, highlight portrayal, and grouping strategies that outcome in the best exactness when utilized on the top k-Closest Neighbors, Choice Trees, Backing Vector Machines, CNN and XG boost classifiers were assessed. The dataset utilized for this study comprised of the 96,724 malware records and 41,323 genuine documents of different arrangements. This work presents suggested techniques for AI based malware order and discovery, as well as the rules for its execution. Besides, the review performed could be helpful as a base for additional examination in the field of malware investigation with AI strategies. Our greatest exactness 99.56% which was dropped by utilized of Random Forest calculation and XG boost calculation. Our model was better than other’s they had accuracy of maximum 86% to 98%.

**CHAPTER ONE**

**INTRODUCTION**

* 1. **Introduction**

Malware detection was a critical aspect of modern cybersecurity, as it plays a crucial role in identifying and mitigating malicious software or code that could compromise the security and integrity of computer systems, networks, and data. With the increasing sophistication and prevalence of malware attacks, effective detection techniques were essential for protecting against potential threats. One common approach to malware detection was signature-based detection. This technique involves comparing files or code against a database of known malware signatures. Signatures were unique patterns or characteristics specific to particular malware strains. If a match was found, it indicates the presence of known malware, allowing for its identification and removal. [21]

However, relying solely on signature-based detection could be limited since it could only detect known malware for which signatures exist. To address this limitation, heuristic-based detection techniques were employed. Heuristics involve analyzing the behavior, characteristics, or attributes of files or code to identify potential malware. By using predefined rules or algorithms, these techniques could identify suspicious patterns or activities associated with malware, even if the specific malware was previously unknown. [2]

Sandboxing was another technique used in malware detection. It involves executing potentially malicious files or code in an isolated and controlled environment, known as a sandbox. By observing the behavior and actions of the code in this safe environment, sandboxing could detect any suspicious activities or potentially harmful actions that the code might perform. This technique was effective in analyzing and identifying malware samples without exposing the actual system to risk. [25]

Network-based detection focuses on analyzing network traffic to identify known patterns or behaviors associated with malware. By monitoring communication protocols, inspecting packets, and detecting any malicious or suspicious network activities, this technique could identify malware at the network level.

To effectively combat malware, it was essential to combine multiple detection techniques and employ a defense-in-depth strategy. This approach involves layering different detection mechanisms to increase the overall security posture and reduce the chances of malware evasion. By utilizing a combination of signature-based, heuristic-based, behavioral-based, machine learning-based, anomaly-based, sandboxing, and network-based detection, organizations could significantly enhance their ability to detect and mitigate malware threats. It was important to note that the effectiveness of malware detection techniques relies on continuous updates and improvements to keep up with the evolving landscape of malware. Regular updates to signature databases, heuristic rules, machine learning models, and knowledge about emerging threats were essential to ensure effective detection. [33]

**The Benefits of Malware Detection:**

Malware detection plays a crucial role in ensuring the security and integrity of computer systems and networks. Here were some key benefits of malware detection:

1. **Threat prevention:** Malware detection helps in identifying and preventing the entry of malicious software into a system. By detecting and blocking malware in real-time, it reduces the risk of infections and potential damage to data and resources.
2. **Data protection:** Malware could compromise sensitive data, such as personal information, financial records, or intellectual property. Malware detection tools could identify and remove malicious software that aims to steal, corrupt, or exploit data, helping to safeguard confidential information.
3. **System performance:** Malware could significantly impact the performance of a system or network. It could consume system resources, slow down processing speeds, or cause crashes and instability. Effective malware detection helps mitigate these issues by removing malicious programs and restoring system performance. [4]

**The Drawbacks of Malware Detection:**

While malware detection tools and techniques had advanced significantly over the years, they still had certain drawbacks and limitations. Here were some of the common drawbacks associated with malware detection:

1. **Zero-day exploits:** Malware detection tools often rely on known malware signatures or behavioral patterns to identify malicious software. However, they struggle to detect zero-day exploits, which were newly discovered vulnerabilities that had not yet been patched or identified. Since zero-day exploits were unknown to security vendors, malware detection tools might fail to recognize and block them until a patch or update was released.
2. **Polymorphic malware:** Polymorphic malware was designed to constantly change its code or appearance, making it difficult for signature-based detection systems to keep up. These types of malware could evade detection by altering their characteristics, such as file size, file name, or encryption methods. As a result, traditional malware detection techniques might struggle to identify polymorphic malware accurately. [4]
3. **False positives and false negatives:** Malware detection tools might produce false positives, indicating that benign software was malicious, or false negatives, failing to detect actual malware. False positives could lead to unnecessary disruptions as legitimate software might be blocked or flagged as malicious. False negatives, on the other hand, could provide a false sense of security, allowing malware to went undetected and potentially cause harm. [18]

**1.2 Malware Types**

To had a better understanding of the methods and logic behind the malware, it was useful to classify it. Malware could be divided into several classes depending on its purpose. The classes were as follows:

1. **Virus:** A computer virus was a malicious program that could replicate itself and infect other files or systems. It could spread and cause damage by altering or deleting files, corrupting data, or disrupting the normal functioning of a computer or network.
2. **Worm:** Worms were self-replicating malware that could spread across networks without any user intervention. They exploit vulnerabilities in operating systems or network protocols to propagate and often consume system resources, leading to network congestion or system slowdown. [36]
3. **Trojan Horse:** Trojan horses were disguised as legitimate software or files but contain malicious code. They could create backdoors, steal sensitive information, or allow unauthorized access to a computer system. Trojans often rely on social engineering techniques to trick users into executing them.
4. **Ransomware:** Ransomware was a type of malware that encrypts a user's files or locks their system until a ransom was paid to the attacker. It could be highly disruptive, causing data loss, financial damage, or even halting critical operations in organizations.
5. **Spyware:** Spyware was designed to covertly collect information about a user's activities, such as browsing habits, keystrokes, or personal information. It was often used for malicious purposes like identity theft, unauthorized surveillance, or targeted advertising.
6. **Adware:** Adware was a type of malware that displays unwanted advertisements or pop-ups on a user's computer. It could slow down system performance, disrupt browsing activities, and sometimes lead to the installation of other malware. [5]
7. **Keylogger:** Keyloggers were programs that capture and record keystrokes on a computer. They could be used to monitor user activity, including login credentials, credit card information, or other sensitive data. Keyloggers could pose a significant threat to privacy and security.
8. **Botnet:** A botnet was a network of infected computers or "bots" that were controlled remotely by a malicious actor. Botnets were commonly used for various malicious activities, such as distributed denial-of-service (DDoS) attacks, spam campaigns, or carrying out coordinated attacks. [9]
9. **Rootkit:** Rootkits were stealthy malware that hides its presence or activities from detection by antivirus software or system administrators. They could gain privileged access to a system and modify the operating system's functionality to facilitate unauthorized activities or maintain persistent control.
10. **Logic Bomb:** A logic bomb was a type of malware that was triggered by a specific event or condition. It was typically hidden within a legitimate program and remains dormant until the triggering condition was met. Once activated, it could cause data corruption, system malfunctions, or other malicious outcomes. [37]

**1.3 Objectives**

The primary objective of malware detection was to identify and mitigate malicious software, commonly known as malware, in order to protect computer systems, networks, and data from harm. The specific objectives of malware detection include:

1. **Identification:** The main goal of malware detection was to identify the presence of malware within a system or network. This involves analyzing files, processes, network traffic, and other system components to detect any signs of malicious activity or the presence of known malware.
2. **Prevention:** By detecting malware, the objective was to prevent it from causing harm to the system or network. This includes blocking access to malicious websites, preventing the execution of malicious files, and stopping the propagation of malware through various means, such as email attachments or infected removable media.
3. **Removal and Quarantine:** Once malware was detected, the objective was to remove or quarantine it effectively. This might involve isolating infected files or processes, deleting or disinfecting them, and restoring affected systems to a clean state. Quarantine allows for further analysis and investigation of the malware to understand its behavior and prevent future infections.

**1.4 A Brief Outline of the Report**

The current project was structured into five chapters. In Chapter 1, the background and motivation of the project were presented along with its objectives and a brief overview of the project. In Chapter 2, a comprehensive literature review was presented to establish the research gap and the state-of-the-art techniques. Chapter 3 outlines the methodology and modeling used in this project, including the working principle, process of work, components, implementation, testing, and analysis. The results and discussions of the project, as well as the simulation and experimental outcomes, were presented in Chapter 4. Finally, Chapter 5 provides the conclusion of the project, summarizing the findings and highlighting the contributions of the research.

**CHAPTER TWO**

**LITERATURE REVIEW**

**2.1 Literature Review**

Gavrilut, et al. 2009 was proposed “Malware Detection Using Machine Learning” for fostering a recognition framework in view of a few changed perceptron calculations. For various calculations, he accomplished the precision of 69.90%-96.18%. It ought to be expressed that the calculations that brought about best precision likewise created the largest number of misleading up-sides: the most reliable one brought about 48 bogus up-sides. The most "adjusted" calculation with fitting precision and the low bogus positive rate had the exactness of 93.01%. [1]

Singhal and Raul 2015 was proposed “Malware Detection Module using Machine Learning Algorithms to Aid Unified Security in Big Business Organizations" talks about the location technique in view of altered Arbitrary Timberland calculation in blend with Data Gain for better element portrayal. It ought to be seen, that the informational index comprises absolutely of convenient executable records, for which include extraction 12 was for the most part more straightforward. The outcome accomplished was the precision of 97% and 0.03 falsepositive rate. [2]

Baldangombo et al. 2013 was proposed “Extraction methods based on PE headers, DLLs and Programming interface capabilities and techniques in view of Guileless Bayes, J48 Choice Trees, and Backing Vector Machines. Most elevated in general exactness was accomplished with the J48 calculation (almost 100% with PE header highlight type and half breed PE header and Programming interface capability highlight type, 98.1% with Programming interface capability highlight type)." [3]

Alazab et al. 2011 was proposed “Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures”, the Programming interface capabilities were utilized for include portrayal once more. The best outcome was accomplished with Help Vector Machines calculation with standardized polykernel. The accuracy of 97.6% was accomplished, with a misleading positive pace of 0.025. [4]

Schultz, et al. sent off AI for viewing as new, staticbased malware, byte n-grams on program executables, and strings for usefulness extraction essayists. In 2007, Bilar [19] delivered Opcode, a malware locater to examine the dissemination of opcode recurrence in non-vindictive and noxious contents. [5]

Elovici, et al. utilized Component Territory and Choice Tree (5 grams, top 300, FS), Bayesian Organization (5 grams), Counterfeit Brain Organization (5 grams, top 300, FS), Choice Tree (utilizing the PE), BN (utilizing the PE) and precision of 95.8 percent. [6] Moskovitch, et al. involved channels for the assortment of capabilities. For the assortment and characterization of capabilities and Choice Tree (DT), Guileless Bayes (NB), and Adaboost, Brain Encouraging groups of people (ANN). The help of help vectors machine (SVM) and M1 (DT and NB supported) utilizing Fisher score and gain proportion (GR) had a precision of 94.9%. [7]

Moskovitch, et al. utilized the n-gram (2,3,4,5,6 grams) of opcodes as standard and utilized the assortment of record recurrence (DF), GR and FS highlights in 2008. They utilized the ANN, DT, Supported DT, NB and Helped NB order calculations, which were outflanked by ANN, DT, BDT in holding a low misleading positive score. [8] Santos, et al. finished up in 2011 that administered learning incorporates marking information so semi-controlled learning was acquainted with perceive obscure malware. They utilized the capability determination approach and different classifiers, for example DT, K-Nearest Neighbors (KNN, Bayesian Organization), Backing Vector Machine (SVM) with an opcode arrangement length of 92.92% and an opcode grouping length of 95.90%. [9] Shabtai, et al. utilized n-gram opcode design highlights in 2012 to characterize the most ideal that anyone could hope to found device for record recurrence (DF), G-mean and Fisher positioning. They involved a few classifiers in their technique, with Irregular Backwoods surpassing 95.146% exactness. [10]

Ashu, et al. proposed another technique for high-accuracy recognition of obscure malware. They concentrated on the recurrence of opcodes and assembled them. The creators tried thirteen classifiers, from which FT, J48, NBT, and Arbitrary Timberland were remembered for the WEKA AI stage, and got more than 96.28% precision for malware. [11] Sahay, et al. utilizing the Ideal K Means Bunching calculation, bunched malware executables and these gatherings were utilized by classifiers to distinguish obscure malware as promising preparation highlights (FT, J48, NBT, and Arbitrary Woods). They found that the recognizable proof by the proposed arrangement of obscure malware had 99.11% exactness. [12]

Ahmadi, et al. gathered Microsoft malware information and hex dump-based attributes utilized (string length, metadata, entropy, n-gram, and picture portrayal) and furthermore qualities got from unmounted records and the characterization calculations of XGBoost (metadata, symbol term, opcodes, libraries, and so on.). They accomplished an exactness of ~98.8%. [13] Drew, et al. utilized the Very Strung Reference Free Arrangement Free N-succession Decoder (STRAND) classifier. They presented an ASM succession model and accomplished an accuracy of over 98.59% with a 10-crease cross-approval approach. [14]

Ye et al. cover normal malware-identification AI techniques, comprising of the disclosure, assemblage and arrangement of things. Be that as it might, center highlights like entropy or underlying entropy and certain intricate attributes like organization activity, opcodes and Programming interface tracks were missing. In correlation, profound learning methods or multimodal malware ID procedures were excluded. [15] Razak, et al. had done a malware bibliometric review to look at distributions connected with malware by district, AI Classifiers for Malware Recognition 273 association, and creator. Be that as it might, the paper doesn't characterize the highlights of malware finders and doesn't think about the most recent advancements in this field. [16]

Sakhnini, et al. (2019) present a bibliometric overview zeroing in on the security parts of IoT empowered savvy matrices. Moreover, the creators address the issue of the various kinds of digital assaults that they tracked down connected with a specific subject. [17] Yazdinejad, et al. (2020) planned a clever RNN model to identify malware dangers in digital forms of money. The creators for this specific review gathered 500 examples of cryptographic money malware and 200 examples of goodware. [18]

M.J de Lucia et al. proposed a strategy for recognizing dubious correspondence utilizing convolutional brain organizations and a help vector machine Their SVM model outflanked the CNN model, and they utilized a methodology that required next to no element designing. [19] Shen et al. contrived a procedure for fingerprinting them dependent essentially upon parcel length data. They accomplished an exactness of up to 91.6 percent utilizing the k-NN method. [20] Dong et al. proposed utilizing the k-Closest Neighbor technique, an AI grouping approach, to order video traffic. Utilizing the powerful time traveling approach, they figured the succession of information bundles and extricated highlights. An irregular woodland model was utilized for grouping. [21]

Schultz et al. were quick to present the idea of information digging for recognizing malwares. They involved three different static highlights for malware order: Convenient Executable (PE), strings and byte groupings. In the PE approach, the elements (like rundown of DLLs utilized by the paired, the rundown of DLL capability calls, and number of various framework calls utilized inside each DLL) were extricated from DLL data inside PE records. Strings were separated from the executables in view of the text strings that were encoded in program records. The byte arrangement approach utilizes successions of n bytes separated from an executable document. They utilized an informational index comprised of 4266 E. [22]

Maniriho et al. 59 documents including 3265 vindictive and 1001 harmless projects. A standard enlistment calculation called Ripper was applied to track down designs in the DLL information. A learning calculation Gullible Bayes was utilized to found designs in the string information and n-grams of byte groupings were utilized as info information for the Multinomial Guileless Bayes calculation. The Innocent Bayes calculation, accepting strings as information, gives the most elevated grouping exactness of 97.11%. The creators guaranteed that the pace of location of malwares involving information mining technique was two times when contrasted with signature based strategy. [23]

Kolter et al. They utilized n-gram (rather than non-covering byte arrangement) and information mining technique to identify malevolent executables. They utilized various classifiers including Gullible Bayes, Backing Vector Machine, Choice Tree and their supported adaptations. They reasoned that helped choice tree gives the best order results. [24]

Nataraj et al. proposed a strategy for imagining and characterizing malwares utilizing picture handling procedures, which envision malware doubles as dim scale pictures. A K-closest neighbor procedure with Euclidean distance strategy was utilized for malware characterization. However, it was extremely quick strategy when contrasted with other malware examination techniques, the constraint was that an assailant could took on countermeasures to beat the framework since this strategy utilizes worldwide picture based highlights. For instance, an assailant could migrate segments in a double or add tremendous measure of excess information. The creators looked at parallel surface based examination (in light of picture handling method) with that of dynamic investigation. They found that grouping utilizing this strategy was quicker, adaptable and was tantamount to dynamic examination with regards to exactness. They additionally found that this approach could powerfully group enormous number of malwares with both stuffed and unloaded examples. The impediment was that this technique was defenseless against learned foes who could jumble their malevolent code to overcome surface investigation. [25]

Kong et al. introduced a system for robotized malware characterization in view of primary data (capability call diagram) of malwares. In the wake of extricating the fine grained highlights in light of capability call chart for each malware test, the similitude was assessed for two malware programs by applying segregate distance metric realizing which groups the malware tests having a place with same family while keeping the various bunches separate by a peripheral distance. The creators then, at that point, utilized a group of classifiers that gain from pair wise malware distances to order malwares into their individual families. [26]

Tian et al. utilized capability length recurrence to characterize Trojans. Capability length was estimated by the quantity of bytes in the code. Their outcomes demonstrate that the capability length alongside its recurrence was critical in recognizing malware family and could be joined with different highlights for quick and versatile malware arrangement. Further they noticed that normally a muddled document had no string comprising of words or sentences and in this way utilized printable string data held inside the executables. They utilized AI calculations accessible in WEKA library for arranging malwares. [27]

Santos et al. brought up that managed learning requires a lot of named executables for the two classes (noxious as well as harmless datasets) and proposed a semi-regulated learning approach for identifying obscure malwares. It was intended to construct an AI classifier utilizing a great deal of marked and unlabelled occurrences. A semi-regulated calculation LLGC (Learning with Neighborhood and Worldwide Consistency) was utilized, which could gain from marked and unlabelled information and gives an answer regard to the inborn design showed by both named and unlabelled occurrences. Executables were addressed by utilizing n-gram dispersion procedure. They likewise decide and assess the ideal number of marked examples and impact of this boundary on the precision of the model. The principal commitment of this exploration was to lessen the quantity of required marked cases while keeping up with high accuracy. The impediment was that the past managed learning approaches introduced in and get improved results (above 90% of precision) than the introduced semi-directed approach. Further in the creators proposed an aggregate learning way to deal with distinguish obscure malwares. A sort of semi-regulated learning presents the strategy for upgrading the grouping of to some extent named information. Aggregate order calculations were utilized to fabricate different AI classifiers utilizing a bunch of named and unlabeled cases. It was approved that the naming endeavors were lower than while administered learning was utilized while keeping up with the high precision rate. [28]

Gandotra et al. utilized variable length guidance grouping alongside AI for distinguishing worms in nature. Prior to dismantling the documents, they recognize compilers, packers. Arrangement decrease was finished and choice tree and arbitrary backwoods AI models were utilized for characterization. They tried their strategy on an informational index of 2774 including 1444 worms and 1330 harmless documents. [29]

Zolkipli et al. introduced a methodology for malware conduct examination. They involved Honey Clients and Amun as security apparatuses to gather malwares. Ways of behaving of these malwares were distinguished by executing each example on both CW Sandbox and Anubis on virtual machine stage. The outcomes created by both of these analyzers were tweaked utilizing human based conduct examination. Then, at that point, the malwares were gathered into malware families Worms and Trojans. The constraint of this work was that customization utilizing human examination isn't feasible for the present ongoing traffic which was voluminous and having various dangers. [30]

Rieck et al. proposed a structure for programmed examination of malware conduct utilizing AI. This structure gathered huge number of malware tests and observed their conduct utilizing a sandbox climate. By implanting the noticed conduct in a vector space, they apply the learning calculations. Grouping was utilized to recognize the clever classes of malware with comparative way of behaving. Relegating obscure malware to these found classes was finished by order. In view of both, grouping and characterization, a gradual methodology was utilized for conduct based examination, equipped for handling the way of behaving of thousands of malware pairs on everyday schedule. [31]

Anderson et al. introduced a malware recognition calculation in light of the examination of diagrams developed from progressively gathered guidance follows. A changed form of Ether malware investigation structure was utilized to gather information. The technique utilizes 2-grams to condition the change probabilities of a markov chain (treated as a chart). Hardware of chart parts was utilized to develop a comparability lattice between examples in the preparation set. Part framework was developed by utilizing two unmistakable proportions of closeness: A Gaussian piece, which estimates neighborhood comparability between the chart edges and a phantom bit which estimates worldwide likeness between the diagrams. From the bit network, a help vector machine was prepared to group the test information. The presentation of numerous portion learning strategy utilized in this work was exhibited by separating various occurrences of malware and harmless programming. Restriction of this approach was that the calculation intricacy was exceptionally high, hence restricting its utilization in certifiable setting. [32]

Bayer et al. proposed a framework that groups huge arrangements of noxious parallels in view of their way of behaving really and naturally. The proposed procedure depends on Anubis to create execution hints of the multitude of tests. Anubis was stretched out in this work with corrupt engendering abilities, to utilize extra data sources. Subsequent to making the extraction follows alongside spoil data, a social profile was separated for each follow, which fills in as contribution to the bunching calculation. The grouping calculation utilized depends on Territory Delicate Hashing (LSH), which was a sub direct (effective) way to deal with the estimated closest neighbor issue. LSH could be utilized to play out an estimated grouping while at the same time registering just a little part of the n2/2 distances between sets of focuses. The creators show the versatility of their methodology by bunching a bunch of 75,000 malware tests in three hours. [33]

Tian et al. involved a mechanized device for extricating Programming interface call successions from executables while these were running in a virtual climate. They utilized the classifiers accessible in WEKA library to separate malware documents from clean records as well with respect to characterizing malwares into their families. They utilized an informational collection of 1368 malwares and 456 clear products to exhibit their work and accomplished a precision of more than 97%. [34]

Biley et al. brought up that the antivirus (AV) items portray the malwares in manners that were not predictable across different AV items, not complete across malwares and were not compact in their semantics. They fostered an order procedure that portrays malwares' conduct as far as framework state changes. Parallels were executed in virtualized climate with windows XP introduced. The virtual machine was to some degree firewalled to restrict the effect of any prompt assault conduct during the execution time frame. A social unique mark of malware's movement was made which incorporates documents composed, processes made and network association and so forth. A couple wise single linkage progressive bunching of the fingerprints utilizing standardized pressure distance (NCD) as a distance metric was utilized to group the malwares. The strategy was applied to the computerized order and investigation of 3700 malware tests gathered over a time of a half year. They additionally measure and look at the consistency, culmination and compactness of the groups with that of AV items. The constraint of this work was that the ability and climate of the virtualized framework was static all through the analyses for consistency. [35]

Park et al. proposed a malware characterization technique which depends on maximal part subgraph discovery. In the wake of executing the malware tests in sandboxed climate, framework calls alongside boundary upsides of these calls were caught and a coordinated chart was produced from these frameworks call follows. The maximal normal subgraph was registered to analyze two projects. The downside of this technique was that there were some known malware tests that figure out how to acquire bit mode honors without utilizing framework call interface and could sidestep the examination strategy. [36]

Firdausi et al. introduced a proof of idea of a malware location technique. At first the way of behaving of malware tests was examined in sandbox climate utilizing Anubis. The reports created were preprocessed into scanty vector models for order utilizing AI. The presentation correlation of 5 distinct classifiers for example k-Closest Neighbors (kNN), Innocent Bayes, J48 Choice Tree, Backing Vector Machine (SVM), and Multi-facet Perceptron Brain Organization (MLP) was finished on a little informational index of 220 pernicious examples and 250 harmless examples with and without include determination. The got results portrayed that general best execution was accomplished by J48 choice tree with a review of 95.9%, a misleading positive pace of 2.4%, an accuracy of 97.3%, and a precision of 96.8%. [37]

Nari et al. introduced a system for computerized malware grouping into their separate families in light of organization conduct. Network follows were taken as contribution to the system as pcap records, from which the organization streams were removed. Then, at that point, a conduct chart was made to address the organization exercises of malwares and conditions between network streams. From these conduct diagrams, the elements like chart size, root out-degree, normal out-degree, greatest out-degree, number of explicit hubs were removed. These elements were then used to order malwares utilizing grouping calculations accessible in WEKA library and it was presumed that J48 choice tree performs better compared to different classifiers. [38]

Lee et al. proposed a technique that bunches the pernicious projects by utilizing AI strategy. Every one of the examples of informational collection were executed in a virtual climate and framework calls alongside their contentions were observed. A social profile was made based on data recorded in regards to test's communication with framework assets like vault keys, composing documents and organization exercises. The likeness between two profiles was determined and afterward by applying k-medoids, various examples were gathered into various bunches. Subsequent to finishing the preparation cycle, the new and obscure examples were relegated to the bunch having medoid nearer to the example for example closest neighbor. Obviously a solitary view either static or dynamic isn't adequate for productively and precisely grouping pernicious projects in light of the muddling and execution-slowing down procedures. Thus, explores had adjusted a cross breed procedure which integrates both static and dynamic highlights at the same time for better malware discovery and order. [39]

Santos et al. proposed a half breed obscure malware identifier called OPEM, which uses a bunch of elements got from both static and dynamic examination of vindictive code. The static highlights were gotten by demonstrating an executable as a succession of functional codes and dynamic elements were acquired by checking framework calls, tasks and raised exemptions. The methodology was then approved north of two unique informational indexes by considering different learning calculations for classifiers Choice Tree, K-closest neighbor, Bayesian organization, and Backing Vector Machine and it had been found that this cross breed approach upgrades the exhibition of the two methodologies when run independently. [40]

Islam et al. to group the executables into malevolent and harmless documents utilizing both static and dynamic elements. The static highlights utilized in this work incorporate capability length recurrence and printable sting data and dynamic elements utilized were Programming interface capability names and Programming interface boundaries. The trial was led utilizing 2939 executable documents including 541 clean records independently for each element for example capability length recurrence, printable string data and Programming interface capability calls and afterward for incorporated technique for meta classifiers SVM, IB1, DT and RF. The acquired outcomes showed that all meta-classifiers accomplish most noteworthy precision for incorporated highlights and meta-RF was the best entertainer for all cases. The creators additionally contrasted their coordinated strategy exactness and those of the current ones and observed that their methodology was showing the best outcomes. [41]

Anderson et al. proposed a technique, in which various information sources (the static parallel, the dismantled double record, its control stream diagram, a powerful guidance follow and framework call follow, and a document data highlight vector) were utilized. For the paired record, dismantled document, and two powerful follows, bits in light of the Markov chain diagrams were utilized. For the control stream chart, a graphlet bit was utilized and for the record data include vector, a standard Gaussian piece was utilized. Then, at that point, numerous part learning was utilized to found a weighted mix of the information sources and backing vector machine classifier was utilized to group the dataset into malignant and harmless. It was tried on a dataset of 780 malwares and 776 harmless examples giving a precision of 98.07%. [42]

Armaan et al. shown and tried the precision of different models. Without information, no application worked for a computerized stage could carry out its role. There were a few digital dangers, so it was fundamental that insurances be taken to protect information. In spite of the fact that highlight choice was troublesome while fostering a model of any kind, AI was a state of the art approach that prepares for exact expectation. The methodology needs a workaround that was sufficiently versatile to deal with non-standard information. To successfully oversee and forestall future attacks, we should examine malware and make new principles and examples as production of malware type. To found designs, IT security experts might utilize malware examination apparatuses. The accessibility of advancements that dissect malware tests and decide their degree of threat fundamentally benefit the online protection area. These devices assist with observing security alarms and forestall malware assaults. Assuming malware was hazardous, we should dispose of it before it sends its disease any further. Malware investigation was turning out to be progressively famous as it assists organizations with decreasing the impacts of the developing number of malware dangers and the rising intricacy of the ways malware could be utilized to assault. [43]

Chowdhury et al. proposed a feasible malware location approach that utilizes an AI characterization procedure. We investigated whether changing a couple of boundaries could expand the precision with which malware was characterized. N-gram and Programming interface call capacities were integrated into our methodology. Exploratory assessment affirmed the viability and reliability of our proposed strategy. Future work will zero in on combining an enormous number of elements to increment identification accuracy while diminishing misleading up-sides. Execution results for contending approaches was plainly prevalent. [44]

Rathore et al. utilized credulous bayes, J48, choice tree, k-closest neighbor, staggered perceptron and support vector machine on highlights separated (utilizing dynamic investigation) and accomplished the most noteworthy precision of 96.8% with J48. produced include vectors with the byte ngram strategy and applied highlight choice in light of archive recurrence and gain proportion. They detailed most elevated precision by choosing top 300 5-gram terms with choice tree and fake brain organization. [45]

**CHAPTER THREE**

**METHODOLOGY AND MODELING**

**3.1 Datasets**

Malware datasets were collections of data specifically curated and organized for research, analysis, and development of malware detection and analysis techniques. These datasets typically consist of samples of malicious software, including various types of malware such as viruses, worms, Trojans, ransomware, and more. Malware datasets serve as valuable resources for cybersecurity researchers, data scientists, and malware analysts to study and understand the characteristics, behavior, and patterns of malware. Our dataset consisted of total 1,38,000 data and 57 columns, which was divided into malware (96,000) and legit (42,000) data. [41]

Here were some key aspects of malware datasets:

1. **Sample Diversity:** A good malware dataset should encompass a diverse range of malware samples, covering different families, variants, and versions. This diversity helps researchers gain insights into the various techniques, functionalities, and behaviors employed by different types of malware.
2. **Real-World Samples:** It was essential for malware datasets to include real-world samples that had been collected from actual malware-infected systems. These samples provide a more accurate representation of the threats faced by organizations and individuals in real-world scenarios. [26]
3. **Malware Categories:** Malware datasets might classify samples into different categories based on their behavior, origin, or purpose. Categories could include adware, spyware, ransomware, keyloggers, banking trojans, or botnets. Such categorization helps researchers focus on specific types of malware and analyze their unique characteristics.
4. **Metadata and Labels:** Malware datasets often provide additional metadata and labels associated with each sample. This might include information such as the date and source of sample collection, family or variant names, file types, and hash values. Labels could indicate whether a sample was malicious or benign, aiding supervised learning techniques and evaluation of detection algorithms. [1]
5. **Sample Size:** The size of a malware dataset could vary significantly, ranging from a few hundred samples to millions. Larger datasets could provide more comprehensive coverage of malware diversity and improve the robustness and accuracy of detection and analysis techniques.
6. **Ground Truth and Validation:** To ensure the quality and reliability of a malware dataset, it was crucial to had a ground truth—definitive labels indicating whether a sample was malicious or benign. Validation processes involve cross-checking samples and labels, which might involve human analysis, sandboxes, or antivirus scanners, to verify the accuracy of the dataset. [44]
7. **Ethical Considerations:** Malware datasets should be curated and distributed with ethical considerations in mind. This includes ensuring that the data does not contain personal identifiable information (PII) or sensitive information that could violate privacy regulations. Additionally, proper data anonymization techniques should be applied to remove any identifying information.

Several public and private organizations provide malware datasets for research and analysis purposes. Examples of well-known malware datasets include the Malware Genome Project, VirusShare, the Microsoft Malware Classification Challenge, and the Adversarial Robustness Testing Benchmark for Malware (ART-Malware) dataset.

It's important to note that working with malware datasets requires adherence to legal and ethical guidelines. Researchers and practitioners should ensure compliance with applicable laws, licensing terms, and data usage policies when utilizing malware datasets for their work. [12]

**3.2 Description of the Important Elements**

A module refers to a file containing Python code that could be imported and used within a notebook or other Python scripts. A module allows us to organize our code into reusable components and provides a way to encapsulate related functions, classes, and variables.

**3.2.1 NumPy**

NumPy was a fundamental module for scientific computing in Python. It provides powerful arrays and mathematical functions for working with large, multi-dimensional arrays and matrices. NumPy was widely used for numerical operations, data manipulation, and linear algebra computations. [21]

**3.2.2 Pandas**

Pandas was a popular data manipulation and analysis module. It provides data structures, such as DataFrame and Series, to efficiently handle and analyze structured data. With Pandas, you could perform tasks like data cleaning, filtering, aggregation, and merging. It was commonly used in data preprocessing and exploratory data analysis. [23]

**3.2.3 Scikit-learn (sklearn)**

Scikit-learn was a machine learning library that offers a comprehensive set of tools for various tasks in supervised and unsupervised learning, including classification, regression, clustering, and dimensionality reduction. It provides a consistent interface and a wide range of algorithms for model training, evaluation, and prediction. scikit-learn was known for its ease of used and integration with other Python libraries. [24]

**3.2.4. Tensor Flow**

Tensor Flow was a powerful open-source library for numerical computation and machine learning. It allows you to build and train neural networks and deep learning models for tasks like image and speech recognition, natural language processing, and more. Tensor Flow provides a flexible architecture for creating computational graphs and offers various optimization techniques to efficiently execute computations on CPUs or GPUs. [31]

**3.3 Design Process**

The process of designing a malware detection system involves several key steps.

**3.3.1 Requirement Analysis**

Understand the specific requirements and goals of the malware detection system. Identify the scope, target platforms (e.g., Windows, Linux), and types of malware to be detected.

**3.3.2 Data Collection**

Gather a comprehensive dataset of both malware and benign (non-malicious) samples. This dataset will be used for training and evaluating the detection system.

**3.3.3 Feature Extraction**

Extract relevant features from the collected samples. These features could include static properties (file size, file type, permissions) and dynamic behavior (API calls, network traffic). Feature selection and extraction techniques might vary depending on the detection approach (signature-based, behavior-based, machine learning-based). [35]

**3.3.4 Model Selection**

Determine the appropriate detection model(s) to be used. This could involve selecting from existing models or designing custom models based on the specific requirements. Common approaches include signature-based (pattern matching), behavior-based (anomaly detection), and machine learning-based (classification) models.

**3.3.5 Training Phase**

Used the collected dataset to train the chosen detection model(s). This typically involves feeding the extracted features into the model(s) and adjusting their parameters to optimize performance. Cross-validation techniques might be employed to evaluate the model's effectiveness. [39]

**3.3.6 Evaluation and Validation**

Assess the performance of the trained model(s) using separate validation datasets. Measure metrics such as accuracy, precision, recall, and F1-score to evaluate the system's effectiveness in detecting malware and minimizing false positives/negatives.

**3.3.7 Integration and Deployment**

Integrate the trained model(s) into a production environment or security infrastructure. This could involve developing APIs, interfaces, or wrappers to facilitate interaction with other security systems or software.

**3.3.8 Ongoing Monitoring and Updating**

Continuously monitor the system's performance in a real-world setting. Collect feedback from users, security experts, or system logs to identify and address any issues or vulnerabilities. Regularly update the detection model(s) to adapt to new and emerging malware threats. [42]

**3.3.9 Collaboration and Information Sharing**

Engage in collaborations with the security community, share knowledge, and stay up to date with the latest research and techniques in malware detection. Participate in threat intelligence sharing platforms to benefit from collective efforts in identifying and mitigating malware.

**3.3.10 System Maintenance and Enhancement**

Regularly maintain and enhance the detection system to keep pace with evolving malware techniques and emerging technologies. Monitor and apply security patches, update libraries, and consider incorporating advanced detection mechanisms to improve accuracy and robustness. [9]

**3.4 Detection Technique**

The proposed approach includes two techniques.

**3.4.1 File Format Inspection**

File Format Inspection, also known as static analysis, was an important component of malware detection. It involves examining the structure and content of a file to identify potential malware indicators without executing or running the file. Here's a brief description of File Format Inspection in the context of malware detection:

1. **File Identification:** The first step in file format inspection was determining the file type and format. This could be done by examining file extensions, magic numbers, or header signatures. Different file types might had distinct structures and characteristics that could provide insights into their nature.
2. **Header Analysis:** Analyzing the file header could reveal valuable information about the file. This includes examining metadata, version information, and identifying any anomalies or suspicious patterns. For example, a legitimate file format might had a specific header structure, and deviations from that structure might indicate potential malware. [8]
3. **File Structure Examination:** Malware often tries to exploit vulnerabilities or inject malicious code into specific sections of a file. By analyzing the file structure, such as the arrangement of sections, data segments, or embedded objects, one could identify any irregularities or unexpected content. This analysis could help detect hidden or obfuscated code, encrypted payloads, or unusual file compression techniques. [20]
4. **Content Parsing:** Parsing the content of a file involves extracting meaningful information from its data components. For example, in the case of document files, the text, images, or macros embedded within could be examined for potentially malicious content. Similarly, in executable files, disassembling or decompiling the code could provide insights into the behavior and intentions of the file.
5. **Metadata Analysis:** Metadata associated with a file, such as timestamps, author information, or version details, could provide contextual information for evaluating the file's legitimacy. Unusual or suspicious metadata entries, such as incorrect timestamps or mismatched author names, could indicate malicious intent. [34]
6. **Known Malware Signatures:** File format inspection might involve comparing the file's content or attributes against a database of known malware signatures. These signatures were pre-defined patterns or characteristics associated with previously identified malware. If a match was found, it indicates the presence of known malware in the file. [2]
7. **Heuristics and Behavioral Indicators:** In addition to known signatures, heuristics and behavioral indicators could be used to identify potential malware. This involves analyzing patterns, code snippets, or certain actions within the file that were commonly associated with malicious behavior. For example, checking for suspicious function calls, network-related operations, or attempts to modify critical system files. [7]

**3.4.2 Dynamic Analysis**

Dynamic Analysis, also known as runtime analysis, was another critical technique used in malware detection. It involves executing and monitoring the behavior of a file or program in a controlled environment to identify potential malicious activities. Here's a brief description of Dynamic Analysis in the context of malware detection:

1. **Execution Monitoring:** Dynamic analysis begins by running the file or program in a controlled environment, such as a sandbox or virtual machine. This allows the analysis system to monitor the execution behavior and interactions of the file with the operating system, network, and other resources.
2. **System Monitoring:** During execution, various system-level events and activities were monitored, including file system operations, registry modifications, network communications, and process creation/termination. By capturing and analyzing these events, it becomes possible to identify suspicious or malicious behavior exhibited by the file. [13]
3. **API and Function Calls:** Dynamic analysis involves monitoring the application programming interface (API) and function calls made by the file. By tracking these calls, it was possible to identify potentially malicious activities, such as attempts to access sensitive resources, modify system settings, or interact with known malware-related functions.
4. **Network Traffic Analysis:** Malicious files often communicate with remote servers or other compromised systems as part of their operation. Dynamic analysis involves capturing and analyzing network traffic generated by the file. This analysis could reveal connections to suspicious IP addresses, data exfiltration attempts, or the used of protocols commonly associated with malware activity. [17]
5. **Behavioral Profiling:** Dynamic analysis enables the creation of behavioral profiles for files or programs. By observing and analyzing their runtime behavior, patterns, and actions, it becomes possible to establish a baseline of normal behavior for legitimate files. Deviations from this baseline could then be used to identify potential malware, such as file modification attempts, unauthorized network communication, or privileged system access.
6. **Code and Memory Analysis:** During dynamic analysis, the code and memory of the executed file could be examined for runtime modifications, injection of malicious code, or attempts to exploit vulnerabilities. Memory analysis could reveal the presence of hidden payloads, encryption techniques, or anti-analysis measures employed by malware. [23]
7. **Malware Triggers and Payload Activation:** Dynamic analysis involves executing the file and observing if it exhibits behavior that triggers the activation of its malicious payload. This could include actions such as opening a certain file, visiting specific URLs, or responding to specific user input. By identifying such triggers, the analysis system could detect malware that remains dormant until specific conditions were met.
8. **Runtime Anomaly Detection:** Dynamic analysis techniques often involve anomaly detection algorithms that identify deviations from normal behavior. Statistical or machine learning-based approaches could be applied to detect unusual patterns or behaviors exhibited by files, which could help identify potential malware. [27]

**3.5 Working Process of Proposed System**

Malware Detection

Machine Learning Algorithms

Data Transformation

Training

Testing

Datasets

**Figure 3.1: Flow Chart of Proposed System**

Malware

Legit

**3.5.1 Data Read**

We read this dataset from our computer memory.

**3.5.2 Data Transformation**

Data transformation refers to the process of converting data from one format, structure, or representation to another, with the goal of making it more suitable or useful for a particular purpose or analysis. It involves manipulating, reformatting, and adjusting data to meet specific requirements or to facilitate data integration, analysis, or visualization. [15]

**3.5.3 Model Training**

Model training refers to the process of teaching a machine learning model to make predictions or perform a specific task by exposing it to training data. During training, the model learns from the provided data and adjusts its internal parameters to optimize its performance. [7]

**3.5.4 Model Testing**

Model testing refers to the process of evaluating the performance and accuracy of a trained machine learning model using a separate set of data called the test dataset. The purpose of testing was to assess how well the model generalizes to new, unseen data and to estimate its performance in real-world scenarios.

**3.5.5 Detection**

At this stage, all model was running and found out the maximum accuracy. [32]

**3.6 Classification Methods**

Classification was a fundamental concept in machine learning and data analysis. It was the process of categorizing or grouping data into predefined classes or categories based on their characteristics or features. The goal of classification was to develop a model or algorithm that could automatically assign new, unseen data points to their appropriate class based on the patterns or relationships learned from the training data.

In classification, the input data consists of a set of features or attributes that describe each data point. These features could be numerical, categorical, or a combination of both. The output of the classification process was a predicted class label for each data point.

To perform classification, machine learning algorithms were trained on labeled datasets, where each data point was associated with a known class label. During the training phase, the algorithm learns the underlying patterns or decision boundaries that differentiate between the different classes. Once the model was trained, it could be used to predict the class labels of new, unseen data points. [4]

**3.6.1 K-Nearest Neighbors**

K-Nearest Neighbors (KNN) was a popular classification algorithm used in machine learning. It was a non-parametric and instance-based learning method that makes predictions based on the similarity of new data points to the labeled examples in the training dataset. Here's how the KNN classification method works:

* **Training Phase:**
  + The KNN algorithm stores the entire training dataset, which consists of labeled examples. Each example was associated with a class label.
  + No explicit training or model building process occurs in KNN. The algorithm simply memorizes the training dataset for future used. [12]
* **Prediction Phase:**
  + When a new, unlabeled data point needs to be classified, the KNN algorithm calculates its similarity or distance to the labeled examples in the training dataset.
  + The most common distance metric used was Euclidean distance, although other metrics like Manhattan distance or cosine similarity could also be used.
  + The KNN algorithm identifies the K nearest neighbors of the new data point based on the computed distances. K was a user-defined parameter that determines the number of neighbors to consider. [25]
  + The class labels of the K nearest neighbors were examined, and the majority class label was assigned to the new data point.
  + In case of ties, different strategies could be used, such as selecting the class label of the closest neighbor or selecting randomly. [3]

**Key considerations in using the KNN algorithm:**

* **Choosing the value of K:** The selection of K was crucial and could significantly impact the classification results. A smaller value of K could lead to overfitting and sensitivity to noisy data, while a larger K value could lead to oversimplification and loss of important patterns. The choice of K depends on the characteristics of the dataset and should be determined through experimentation and validation.
* **Feature scaling:** It was generally recommended to scale or normalize the features in the dataset before applying the KNN algorithm. Since KNN relies on distance measures, features with larger scales might dominate the similarity calculations. Scaling ensures that all features contribute equally to the distance calculation.
* **Handling categorical features:** KNN was naturally suited for continuous numerical features. If the dataset contains categorical features, appropriate transformations need to be applied to convert them into numerical representations. This could involve techniques such as one-hot encoding or ordinal encoding.

KNN was a simple yet effective classification algorithm, especially for datasets with well-defined clusters or local patterns. It was relatively easy to implement and interpret. However, KNN could be computationally expensive when dealing with large datasets, as it requires calculating distances for each data point. Additionally, the algorithm doesn't provide explicit insights into the underlying relationships in the data and assumes that all features contribute equally to the similarity calculation.

The schematic model was illustrated in Figure 1. [39]

**Figure 3.2: K-Nearest Neighbors**

**3.6.2 Support Vector Machines**

Support Vector Machines (SVM) was a popular classification method in machine learning. It was a supervised learning algorithm that aims to found an optimal hyperplane in a high-dimensional feature space to separate different classes of data points. SVMs were effective for both linearly separable and non-linearly separable data. Here's an overview of the SVM classification method:

1. **Hyperplane and Margin:** In SVM, a hyperplane was a decision boundary that separates the data points of different classes. For a binary classification problem, the hyperplane was a line in a two-dimensional space or a hyperplane in a higher-dimensional space. The goal was to found the optimal hyperplane that maximizes the margin, which was the distance between the hyperplane and the nearest data points of each class. The data points closest to the hyperplane were called support vectors. [6]
2. **Linear SVM:** In linear SVM, the hyperplane was a linear combination of the input features. The SVM algorithm tries to found the hyperplane that maximizes the margin while ensuring that all data points were correctly classified. This optimization problem could be formulated as a convex quadratic programming problem. [35]
3. **Kernel Trick and Non-linear SVM:** SVM could also handle non-linearly separable data by employing the kernel trick. The kernel function allows the data to be mapped to a higher-dimensional feature space where it might become linearly separable. The most commonly used kernels were the linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. The choice of the kernel depends on the problem domain and the characteristics of the data.
4. **Training and Prediction:** To train an SVM classifier, labeled data points were used to learn the parameters of the hyperplane. The optimization process involves finding the support vectors and determining the optimal weights for the hyperplane. Once the SVM model was trained, it could be used to predict the class labels of new, unseen data points by evaluating their position relative to the learned hyperplane. [22]
5. **Regularization and Soft Margin:** SVMs could be further extended to handle data that was not perfectly separable by introducing a soft margin. Soft margin SVM allows for some misclassification errors by allowing data points to fall within the margin or even on the wrong side of the hyperplane. The regularization parameter, often denoted as C, controls the trade-off between maximizing the margin and minimizing the misclassification errors. [13]

**Figure 3.3: Support Vector Machines Scheme**

**3.6.3 Naive Bayes**

Naive Bayes was a popular classification method that was based on the application of Bayes' theorem with an assumption of independence among features. It was a simple and efficient algorithm that performs well in many real-world applications. Naive Bayes classifiers were particularly useful when dealing with high-dimensional datasets and could handle both categorical and numerical features. Here's an overview of Naive Bayes classification: [4]

1. **Bayes' Theorem:** Naive Bayes classification was based on Bayes' theorem, which calculates the probability of a class given the observed features. The theorem was expressed as:

P(Class | Features) = (P(Features | Class) \* P(Class)) / P(Features)

where:

* P(Class | Features) was the posterior probability of the class given the features.
* P(Features | Class) was the likelihood of observing the features given the class.
* P(Class) was the prior probability of the class.
* P(Features) was the probability of observing the features. [9]

1. **Naive Assumption:** The "naive" assumption in Naive Bayes refers to the assumption that all features were independent of each other, given the class. This simplifies the computation by assuming that the presence of a particular feature does not affect the presence or absence of any other feature.
2. **Training Phase:** During the training phase, the Naive Bayes classifier estimates the probabilities required for classification. It calculates the prior probability of each class by counting the occurrences of each class in the training data. It also estimates the likelihood of each feature given each class by calculating the conditional probabilities. [18]
3. **Classification Phase:** In the classification phase, the Naive Bayes classifier predicts the class label for new, unseen data. It calculates the posterior probability of each class given the observed features using Bayes' theorem. The class with the highest posterior probability was assigned as the predicted class label.
4. **Types of Naive Bayes Classifiers:** There were different variants of Naive Bayes classifiers, depending on the distributional assumptions made about the data. The common types include:

* Gaussian Naive Bayes: Assumes that numerical features follow a Gaussian distribution.
* Multinomial Naive Bayes: Suitable for discrete features that follow a multinomial distribution, often used in text classification with term frequencies.
* Bernoulli Naive Bayes: Assumes binary features, typically used for binary classification tasks.
* Complement Naive Bayes: A variation of multinomial Naive Bayes that addresses the imbalanced class problem. [38]

**3.6.4 Decision Tree**

A decision tree was a simple yet powerful machine learning algorithm that was used for both classification and regression tasks. It was a tree-like structure where each internal node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents the outcome or prediction. The decision tree algorithm builds the tree by recursively partitioning the training data based on the values of the input features. At each node, the algorithm selects the best feature that provides the most information gain or reduction in impurity. The impurity measures the disorder or uncertainty in the data. Popular impurity measures include Gini index and entropy. [33]

Once the tree was constructed, new data points could be classified or predicted by traversing down the tree from the root node to a leaf node. At each internal node, the decision rule was applied based on the feature value, and the traversal continues until a leaf node was reached, which provides the final prediction or classification. Decision trees were known for their interpretability as the resulting tree structure could be easily visualized and understood. They could handle both categorical and numerical features, and they could also handle missing values in the data. [22]

**3.6.5 Random Forest**

Arbitrary Woods was one of the most well-known AI calculations. It requires practically no information readiness and displaying except for ordinarily brings about exact outcomes. Irregular Woodlands depend on the choice trees portrayed in the past area. All the more explicitly, Irregular Backwoods were the assortments of choice trees, creating a superior expectation precision. To that end it was known as a 'woodland' - it was fundamentally a bunch of choice trees.

The essential thought was to develop various choice trees in view of the free subsets of the dataset. At every hub, n factors out of the list of capabilities were chosen haphazardly, and the best divided on these factors was found. [10]

As in the choice trees, this calculation eliminates the requirement for highlight determination for eliminating superfluous elements - they won't be considered regardless. The main requirement for any component choice with the irregular timberland calculations emerges 26 when there was a requirement for dimensionality decrease. Additionally, the out-of-sack mistake rate, which was referenced prior could be viewed as the calculation's own cross-approval technique. This eliminates the requirement for drawn-out cross-approval gauges, that would need to be taken in any case. [19]

Arbitrary woodlands acquire a large number of the upsides of the choice trees calculations. They were appropriate to both relapse and grouping issues; they were not difficult to register and fast to fit. They likewise ordinarily bring about the better precision. Nonetheless, not at all like choice trees, interpretting the results was extremely difficult. In choice trees, by analyzing the subsequent tree, we could acquire significant data about which factors were significant and what they mean for the outcome. This was preposterous with arbitrary timberlands. It could likewise be depicted as a more-steady calculation than the choice trees - in the event that we alter the information a tad, choice trees will change, in all likelihood diminishing the precision. This won't occur in the irregular woodland calculations - since it was the mix of numerous choice trees, the arbitrary backwoods will stay stable. [25]

**3.7** **Test/Experimental Setup**

This experimental setup allows us to create a virtual environment using Anaconda3, used Jupyter Notebook for interactive code development, and leverage the PyCharm IDE for a more comprehensive development environment. We could now proceed with our malware detection experiment by writing code and analyzing the results.

**3.8 Implementation**

# Figure 3.4: Data Level for Malware and Legit

**Figure 3.5: Accuracy for 100 Data**

**Figure 3.6: Accuracy for 10000 Data**

**Figure 3.7: Accuracy for 50000 Data**

**CHAPTER FOUR**

**RESULTS AND DISCUSSION**

As an update, the objective of the venture lies in the assurance of the most reasonable element portrayal and extraction strategies, the most reliable calculation that could recognize the malware families with the least blunder rate and how this precision connects with the ongoing scoring framework exactness. This part talks about the commonsense parts of the task execution. This incorporates information gathering, depiction of malware families that address the dataset, choice of the elements that will be utilized for the calculation and tracking down the ideal component portrayal strategy, assessment technique, and the execution interaction.

**4.1** **Performance Evaluation**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm Name** | **Our Accuracy** | **Others Accuracy** | **Our Timing** | **Others Timing** |
| Random Forest | 99.45% | 99.21% [2] | 1.2 s | 2 s |
| Random Forest | 99.5% | 98% [7] | 2 s | 5 s |
| Random Forest | 99.56% | 93.5% [16] | 5 s | 15 s |
| Logistic Regression | 70% | 75% [2] | 12 s | 5 s |
| Logistic Regression | 68% | 68% [5] | 8 s | 2 s |
| Logistic Regression | 71% | 65% [12] | 3 s | 10 s |
| CNN | 85% | 70% [35] | 1 s | 2 s |
| CNN | 85% | 75% [22] | 3 s | 1 s |
| CNN | 85% | 83% [8] | 2 s | 7 s |
| KNN | 98% | 96.5% [27] | 10 s | 15 s |
| KNN | 95% | 94% [31] | 6 s | 20 s |
| KNN | 97.5% | 91% [29] | 4 s | 5 s |
| Naive Bayes | 65% | 60% [16] | 4 s | 12 s |
| Naive Bayes | 64.5% | 62% [15] | 6 s | 15 s |
| Naive Bayes | 62% | 62.5% [11] | 10 s | 20 s |
| Decision Trees | 98% | 96% [1] | 13 s | 20 s |
| Decision Trees | 95% | 91% [7] | 11 s | 25 s |
| Decision Trees | 94% | 92.5% [18] | 8 s | 14 s |
| XG Boost | 99% | 96% [10] | 7 s | 15 s |
| XG Boost | 99.62% | 93% [36] | 11 s | 14 s |
| XG Boost | 98% | 92% [11] | 9 s | 6 s |

**4.2 Advantages**

Malware detection plays a crucial role in maintaining the security and integrity of computer systems and networks. Here were some key advantages of malware detection:

1. **Threat Mitigation:** Malware detection helps identify and mitigate various types of malicious software, including viruses, worms, trojans, ransomware, spyware, and adware. By detecting malware early on, potential threats could be neutralized before they cause significant damage.
2. **System Protection:** Malware detection ensures the protection of computer systems, preventing unauthorized access, data breaches, and unauthorized modifications. It helps maintain system integrity by identifying and removing malicious code that could compromise system functionality.
3. **Data Security:** Malware detection helps safeguard sensitive and confidential data from unauthorized access, theft, or tampering. By identifying and eliminating malware, it reduces the risk of data breaches, protecting valuable information such as personal data, financial records, intellectual property, and trade secrets. [31]
4. **Network Security:** Malware detection was essential for maintaining network security. It helps prevent the spread of malware across networks, limiting its impact on multiple devices and systems. By detecting and isolating infected devices, it helps contain the threat and prevents the malware from compromising other networked resources.
5. **Proactive Threat Prevention:** Malware detection enables proactive identification and prevention of emerging threats. Advanced detection techniques, such as behavior analysis and machine learning, could identify previously unknown or zero-day malware based on their patterns and characteristics. This proactive approach helps stay ahead of evolving malware threats and minimizes the potential damage they could cause. [3]

**4.3 Disadvantages**

While malware detection brings significant benefits to the security and integrity of computer systems and networks, it also had certain disadvantages. Here were some of the disadvantages of malware detection:

1. **False Positives:** One of the main drawbacks of malware detection was the possibility of false positives. False positives occur when legitimate files or activities were incorrectly flagged as malware. False positives could lead to unnecessary disruption, inconvenience, and potential loss of productivity if legitimate files or applications were mistakenly removed or blocked.
2. **False Negatives:** Similarly, false negatives occur when malware was not detected by the malware detection system. This could happen if the malware was new or employs advanced evasion techniques that went undetected by the current detection mechanisms. False negatives could leave systems vulnerable to malware attacks, potentially leading to data breaches, system compromise, and other security incidents. [38]
3. **Evolving Malware:** Malware was constantly evolving, with attackers using sophisticated techniques to evade detection. The rapid evolution of malware makes it challenging for malware detection systems to keep up and detect all types of malicious software effectively. New strains of malware, such as zero-day exploits, could bypass traditional detection mechanisms, posing a significant challenge to malware detection effectiveness.
4. **Resource Intensiveness:** Some malware detection techniques, particularly those that rely on intensive analysis or behavior monitoring, could be resource-intensive. These techniques might require significant computing resources, such as CPU power and memory, leading to potential performance impacts on the system. High resource consumption could slow down system operations, affecting user experience and efficiency. [25]
5. **Detection Lag:** There might be a time lag between the emergence of new malware and its detection by security vendors. During this period, systems remain vulnerable to the newly developed malware until it was identified and countermeasures were implemented. This detection lag could provide a window of opportunity for attackers to exploit systems before effective countermeasures were in place. [9]

**4.4 Applications**

Malware detection finds applications in various areas where maintaining the security and integrity of computer systems and networks was crucial. Here were some key applications of malware detection:

1. Endpoint Protection: Malware detection was widely used for protecting endpoints such as desktops, laptops, servers, and mobile devices from malware attacks. Endpoint protection solutions employ malware detection techniques to identify and neutralize malicious software, preventing unauthorized access, data breaches, and system compromise.
2. Network Security: Malware detection plays a vital role in network security by detecting and preventing malware from spreading across networks. Network security solutions leverage malware detection mechanisms to monitor network traffic, identify malware-infected devices, and block malicious activities to ensure the overall integrity and security of the network. [7]
3. Email Security: Malware detection was employed in email security solutions to scan email attachments and links for potential malware. By detecting and blocking malicious content in emails, malware detection helps prevent malware infections that could be spread through phishing attacks, malicious attachments, or embedded links.
4. Web Security: Malware detection was utilized in web security solutions to protect users from accessing websites that contain malware or engage in malicious activities. Web security solutions used various techniques, including URL reputation analysis, behavioral analysis, and content scanning, to detect and block malicious websites and web-based malware threats.
5. Cloud Security: Malware detection was essential for ensuring the security of cloud-based environments and services. Cloud security solutions employ malware detection techniques to identify and prevent the deployment or execution of malware within cloud infrastructure or applications, safeguarding cloud data and resources. [13]

# CHAPTER FIVE

# CONCLUSIONS AND FUTURE WORK

**5.1 Conclusion**

Malware detection was a critical component of maintaining the security and integrity of computer systems and networks. It plays a vital role in identifying and mitigating various types of malicious software, including viruses, worms, trojans, ransomware, spyware, and adware. By detecting and neutralizing malware, organizations could protect their data, systems, and networks from unauthorized access, data breaches, and potential damage.

Malware detection offers numerous advantages, including threat mitigation, system protection, data security, network security, proactive threat prevention, system performance optimization, compliance adherence, business continuity, user trust, and cost savings. It helps identify and prevent malware-related incidents, safeguard sensitive information, and ensure the smooth operation of critical business processes. Malware detection techniques, such as signature-based detection, behavior analysis, machine learning, and heuristic scanning, contribute to staying ahead of evolving malware threats and minimizing their impact.

However, there were also certain limitations and challenges associated with malware detection. False positives and false negatives could occur, leading to potential disruptions or vulnerabilities. The constant evolution of malware, obfuscation techniques employed by attackers, resource intensiveness, and the need for continuous updates and improvements pose ongoing challenges for effective malware detection. Additionally, there was a risk of overreliance on detection mechanisms, which might overlook other essential security measures and privacy concerns related to data inspection and analysis.

To overcome these challenges, a multi-layered approach to security was recommended, combining different malware detection techniques, continuous monitoring, user education, regular software updates, and secure coding practices. Organizations should also consider implementing advanced detection mechanisms, such as behavior analysis, anomaly detection, and threat intelligence integration, to enhance their malware detection capabilities.

In summary, malware detection was indispensable in the realm of cybersecurity. It provides crucial protection against malware threats, enhances system security, preserves data integrity, and helps maintain business continuity. By understanding the advantages, limitations, and best practices associated with malware detection, organizations could fortify their defenses and mitigate the risks posed by malicious software, ensuring the overall safety and resilience of their digital environments.

**5.2 Future Work**

The field of malware detection continues to evolve rapidly as cyber threats become more sophisticated. Several areas offer promising avenues for future work in malware detection:

1. **Advanced Machine Learning Techniques:** Applying advanced machine learning algorithms, such as deep learning and reinforcement learning, could enhance the accuracy and efficiency of malware detection. These techniques could improve the ability to detect new and unknown malware by learning complex patterns and behaviors. [34]
2. **Behavior-based Detection:** Expanding behavior-based detection approaches could provide deeper insights into malware activities. By analyzing the behavior of software and monitoring system events, anomalies and suspicious patterns could be identified, allowing for the early detection of sophisticated malware.
3. **Real-time Detection:** Developing real-time malware detection systems that could swiftly identify and respond to malware attacks was crucial. Incorporating streaming analytics, data processing technologies, and threat intelligence feeds could enable the timely detection of emerging threats and reduce response time. [9]
4. **Zero-day Malware Detection:** Zero-day malware refers to previously unknown malware that exploits vulnerabilities before they were patched. Developing effective techniques for early detection and prevention of zero-day malware could significantly enhance overall security.
5. **Adversarial Machine Learning:** Adversarial machine learning focuses on developing robust models that could withstand evasion techniques used by malware authors. By considering adversarial attacks during the training and evaluation process, models could be hardened against evasion attempts. [37]

**REFERENCES**

1. Gavriluţ, D., Cimpoeşu, M., Anton, D., & Ciortuz, L. (2009, October). Malware detection using machine learning. In *2009 International multiconference on computer science and information technology* (pp. 735-741). IEEE.
2. Singhal, P., & Raul, N. (2012). Malware detection module using machine learning algorithms to assist in centralized security in enterprise networks. *arXiv preprint arXiv:1205.3062*.
3. Baldangombo, U., Jambaljav, N., & Horng, S. J. (2013). A static malware detection system using data mining methods. *arXiv preprint arXiv:1308.2831*.
4. Alazab, M., Venkatraman, S., Watters, P. A., & Alazab, M. (2011). Zero-day Malware Detection based on Supervised Learning Algorithms of API call Signatures. *AusDM*, *11*, 171-182.
5. Habtor, S. A., & Dahah, A. H. H. (2021). Machine-Learning Classifiers for Malware Detection Using Data Features. *Journal of ICT Research & Applications*, *15*(3).
6. Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, *1*(1), 1-22.
7. Habtor, S. A., & Dahah, A. H. H. (2021). Machine-Learning Classifiers for Malware Detection Using Data Features. *Journal of ICT Research & Applications*, *15*(3).
8. Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, *1*(1), 1-22.
9. Santos, I., Nieves, J., & Bringas, P. G. (2011). Semi-supervised learning for unknown malware detection. In *International Symposium on Distributed Computing and Artificial Intelligence* (pp. 415-422). Springer Berlin Heidelberg.
10. Shabtai, A., Moskovitch, R., Feher, C., Dolev, S., & Elovici, Y. (2012). Detecting unknown malicious code by applying classification techniques on opcode patterns. *Security Informatics*, *1*(1), 1-22.
11. Habtor, S. A., & Dahah, A. H. H. (2021). Machine-Learning Classifiers for Malware Detection Using Data Features. *Journal of ICT Research & Applications*, *15*(3).
12. Sahay, S. K., & Sharma, A. (2016). Grouping the executables to detect malwares with high accuracy. *Procedia Computer Science*, *78*, 667-674.
13. Sharma, S., Rama Krishna, C., & Sahay, S. K. (2019). Detection of advanced malware by machine learning techniques. In *Soft Computing: Theories and Applications: Proceedings of SoCTA 2017* (pp. 333-342). Springer Singapore.
14. Drew, J., & Hahsler, M. (2014, September). Strand: fast sequence comparison using mapreduce and locality sensitive hashing. In *Proceedings of the 5th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics* (pp. 506-513).
15. Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., & Ye, H. (2018). Significant permission identification for machine-learning-based android malware detection. *IEEE Transactions on Industrial Informatics*, *14*(7), 3216-3225.
16. 16. Mat, S. R. T., Ab Razak, M. F., Kahar, M. N. M., Arif, J. M., Mohamad, S., & Firdaus, A. (2021). Towards a systematic description of the field using bibliometric analysis: malware evolution. *Scientometrics*, *126*, 2013-2055.
17. Sakhnini, J., Karimipour, H., Dehghantanha, A., Parizi, R. M., & Srivastava, G. (2021). Security aspects of Internet of Things aided smart grids: A bibliometric survey. *Internet of things*, *14*, 100111.
18. Yazdinejad, A., HaddadPajouh, H., Dehghantanha, A., Parizi, R. M., Srivastava, G., & Chen, M. Y. (2020). Cryptocurrency malware hunting: A deep recurrent neural network approach. *Applied Soft Computing*, *96*, 106630.
19. De Lucia, M. J., & Cotton, C. (2019, November). Detection of encrypted malicious network traffic using machine learning. In *MILCOM 2019-2019 IEEE Military Communications Conference (MILCOM)* (pp. 1-6). IEEE.
20. Shen, M., Wei, M., Zhu, L., & Wang, M. (2017). Classification of encrypted traffic with second-order markov chains and application attribute bigrams. *IEEE Transactions on Information Forensics and Security*, *12*(8), 1830-1843.
21. Dong, W., Moses, C., & Li, K. (2011, March). Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web* (pp. 577-586).
22. Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2000, Might). Data mining methods for detection of new malicious executables. In *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001* (pp. 38-49). IEEE.
23. Maniriho, P., Mahmood, A. N., & Chowdhury, M. J. M. (2022). A study on malicious software behaviour analysis and detection techniques: Taxonomy, current trends and challenges. *Future Generation Computer Systems*, *130*, 1-18.
24. Kolter, J. Z., & Maloof, M. A. (2004, August). Learning to detect malicious executables in the wild. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 470-478).
25. Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S. (2011, July). Malware images: visualization and automatic classification. In *Proceedings of the 8th international symposium on visualization for cyber security* (pp. 1-7).
26. Kong, D. and Yan, G. (2013) Discriminant Malware Distance Learning on Structural Information for Automated Malware Classification. Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems, 347-348.
27. Tian, R., Batten, L. M., & Versteeg, S. C. (2008, October). Function length as a tool for malware classification. In *2008 3rd international conference on malicious and unwanted software (MALWARE)* (pp. 69-76). IEEE.
28. Santos, I., Sanz, B., Laorden, C., Brezo, F., & Bringas, P. G. (2011). Opcode-sequence-based semi-supervised unknown malware detection. In *Computational Intelligence in Security for Information Systems: 4th International Conference, CISIS 2011, Held at IWANN 2011, Torremolinos-Málaga, Spain, June 8-10, 2011. Proceedings* (pp. 50-57). Springer Berlin Heidelberg.
29. Gandotra, E., Bansal, D., & Sofat, S. (2014). Malware analysis and classification: A survey. *Journal of Information Security*, *2014*.
30. Zolkipli, M. F., & Jantan, A. (2010, September). Malware behavior analysis: Learning and understanding current malware threats. In *2010 Second International Conference on Network Applications, Protocols and Services* (pp. 218-221). IEEE.
31. Rieck, K., Trinius, P., Willems, C., & Holz, T. (2011). Automatic analysis of malware behavior using machine learning. *Journal of computer security*, *19*(4), 639-668.
32. Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, *7*, 247-258.
33. Bayer, U., Comparetti, P. M., Hlauschek, C., Kruegel, C., & Kirda, E. (2009, February). Scalable, behavior-based malware clustering. In *NDSS* (Vol. 9, pp. 8-11).
34. Tian, R., Islam, R., Batten, L., & Versteeg, S. (2010, October). Differentiating malware from cleanware using behavioural analysis. In *2010 5th international conference on malicious and unwanted software* (pp. 23-30). Ieee.
35. Biley, M., Oberheid, J., Andersen, J., Morley Mao, Z., Jahanian, F. and Nazario, J. (2007) Automated Classification and Analysis of Internet Malware. Proceedings of the 10th International Conference on Recent Advances in Intrusion Detection, 4637, 178-197. [http://dx.doi.org/10.1007/978-3-540-74320-0\_10](http://dx.doi.org/10.1007/978-3-540-74320-0_10" \t "_blank)
36. Park, Y., Reeves, D., Mulukutla, V., & Sundaravel, B. (2010, April). Fast malware classification by automated behavioral graph matching. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research* (pp. 1-4).
37. Firdausi, I., Erwin, A., & Nugroho, A. S. (2010, December). Analysis of machine learning techniques used in behavior-based malware detection. In *2010 second international conference on advances in computing, control, and telecommunication technologies* (pp. 201-203). IEEE.
38. Nari, S., & Ghorbani, A. A. (2013, January). Automated malware classification based on network behavior. In *2013 International Conference on Computing, Networking and Communications (ICNC)* (pp. 642-647). IEEE.
39. Liu, L., Wang, B. S., Yu, B., & Zhong, Q. X. (2017). Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering*, *18*(9), 1336-1347.
40. Santos, I., Devesa, J., Brezo, F., Nieves, J., & Bringas, P. G. (2013). Opem: A static-dynamic approach for machine-learning-based malware detection. In *International joint conference CISIS’12-ICEUTE´ 12-SOCO´ 12 special sessions* (pp. 271-280). Springer Berlin Heidelberg.
41. Islam, R., Tian, R., Batten, L. M., & Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, *36*(2), 646-656.
42. Anderson, B., Quist, D., Neil, J., Storlie, C., & Lane, T. (2011). Graph-based malware detection using dynamic analysis. *Journal in computer Virology*, *7*, 247-258.
43. Dag, S., Arman, E. E., & Yildirim, B. (2010). Computation of thermal fracture parameters for orthotropic functionally graded materials using Jk-integral. *International Journal of Solids and Structures*, *47*(25-26), 3480-3488.
44. Ahsan, M., Gomes, R., Chowdhury, M. M., & Nygard, K. E. (2021). Enhancing machine learning prediction in cybersecurity using dynamic feature selector. *Journal of Cybersecurity and Privacy*, *1*(1), 199-218.
45. Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018). Malware detection using machine learning and deep learning. In *Big Data Analytics: 6th International Conference, BDA 2018, Warangal, India, December 18–21, 2018, Proceedings 6* (pp. 402-411). Springer International Publishing.

**APPENDIX A**

**CODING SUMMARY**

confusion\_matrix,f1\_score,accuracy\_score,plot\_confusion\_matrix,auc,ConfusionMatrixDisplay

from sklearn. model\_ selection import train\_ test\_ split

from sklearn import model\_selection

Mal = pd.read\_csv('Data.csv', sep='|')

Legit = Mal [0:41000].drop(['legitimate'], axis=1)

Malware = MalwareDataset[41000::].drop(['legitimate'], axis=1)

print("The shape of the legit dataset was : %s samples, %s features"%(Legit.shape[0],Legit.shape[1]))

print("The shape of the Malware dataset was : %s samples, %s features"%(Malware.shape[0],Malware.shape[1]))

y=MalwareDataset['legitimate'].values

Data2 = Mal.drop(['Name', 'md5', 'legitimate'], axis=1).values

Target = Mal.['legitimate']

FeatSelect = sklearn.ensemble.ExtraTreesClassifier().fit(Data, Target)

Model = SelectFromModel(FeatSelect, prefit=True)

new = Model.transform(Data2)

Legit\_Train, Legit\_Test, Malware\_Train, Malware\_Test = train\_test\_split(Data, Target ,test\_size=0.2,random\_state=42)

clf = sklearn.ensemble.RandomForestClassifier(n\_estimators=50)

randomModel = clf.fit(Legit\_Train, Malware\_Train)

score = clf.score(Legit\_Test, Malware\_Test)

print("The accuracy of Random Forest Algorithm was", score\*100)