MS Thesis

by Naveed Khurshid

Submission date: 09-Sep-2024 09:25AM (UTC+0500)
Submission ID: 2448736813
File name: Naveed_API_based_Malware_Detection-1_removed.pdf (883.84K)
Word count: 11290
Character count: 62957

Abstract

Malicious software penetrates and jeopardizes the security of computer systems and networks malicious software or malware poses a serious risk. Sophisticated polymorphic and emerging malware strains cannot be detected by conventional malware detection techniques based on static signatures. Techniques for dynamic analysis particularly those that look at the calls in an Application Programming Interface (API) seem to be a viable way to record and analyze virus behavior. In this work we present a novel approach to malware analysis that leverages static feature extraction and API call sequences. Part of our technique involves watching and recording the sequences of API calls that an executable generates while it runs in a controlled environment. After that we transform these sequences into a collection of static features that provide a comprehensive description of the malware operations. Numerous attributes including inter-call interdependence temporal patterns and API call frequency are covered by the generated static features. Next machine learning techniques are applied to evaluate this static feature set and categorize the executable as malicious or benign based on patterns of learned malicious behavior. By using deep analysis API calls and reducing false-positive occurrences, the suggested method shows substantial promise in accurately identifying malware, improving overall detection accuracy. This research offers useful insights into enhancing cyber security measures and protecting digital systems and networks against emerging and sophisticated malware attacks by contributing to a more accurate and effective malware detection procedure.

Chapter 1

Introduction

1.1 Introduction

The growth of harmful software, also known as malware, poses a danger to the security and integrity of computer systems and networks in an increasingly linked digital environment. The hostile actors take advantage of software flaws, compromise critical information, and interfere with regular computing activities, harming people, businesses, and even entire countries financially and reputationally. A crucial task in cybersecurity is identifying and reducing malware risks. Malware refers to any sort of software that is intentionally designed to infiltrate, harm, or compromise computer systems, networks, and devices without the user's consent or knowledge. In this thesis, we have developed a Hybrid Detection Logic to analyze and detect whether a binary is malware or benign. Malware is a broad category of malicious programs designed by cybercriminals for a variety of harmful reasons, such as stealing sensitive information, disrupting computer processes, or gaining illegal access to system [1].

1.2 Motivation

Cybersecurity experts are extremely concerned about the proliferation of malware and its detrimental effects on people businesses and governmental organizations. In addition to negatively impacting everyday users lives the growing threat posed by malware particularly ransomware has caused major financial and operational losses for businesses in a number of different industries [2]. An urgent need for thorough research to comprehend the mechanics of malware attacks create efficient defenses and enhance general cybersecurity techniques has arisen as a result of this growing threat.

As evidenced by the incredible harm caused by ransomware the financial consequences of malware attacks underscore the basic urgency of addressing this problem. Research published in Cybercrime Magazine projects that by 2031 ransomware attacks alone will have cost the world an astounding USD 265 billion in losses [3]. The aforementioned estimates underscore the pressing nature of the issue and present a persuasive argument for conducting comprehensive research to comprehend the intricacies involved in the dissemination exploitation and consequent economic ramifications of malware.

In addition the need for proactive and flexible measures is highlighted by the malware strains rapid evolution and diversification as demonstrated by the prolific production of over 91 million unique variations in a single year [4]. The new strategies that attackers are employing are too sophisticated for conventional security measures to handle. Therefore in order to inform the development of novel protective mechanisms and strategies a thorough investigation into the fundamental mechanics of malware production propagation and evasion tactics is imperative.

The thesis aims to explore the complex landscape of malware attacks with a particular emphasis on ransomware in the context of these urgent concerns. By carefully examining the techniques and tactics employed by malware developers this study seeks to identify malware vulnerabilities and workable countermeasures. This thesis uses an interdisciplinary approach involving computer science and behavioral analysis in an effort to contribute to the development of more resilient cybersecurity frameworks that can effectively address the growing threat of malware.

Ultimately the necessity to defend digital infrastructures and information assets against the growing wave of malware attacks serves as the thesis driving force. This research aims to provide cybersecurity practitioners policymakers and organizations with the knowledge and insights necessary to strengthen their defenses against an ever-evolving and expanding threat landscape by illuminating the complex dynamics of malware propagation and its far-reaching consequences.

Chapter 2

Background

Malware analysis is the process of figuring out what a dubious file or URL is trying to say and why. The goal is usually to stop or reduce cyberthreats by figuring out its operation source and potential impact. This process is essential to cybersecurity because it supports the identification understanding and prevention of malware or malicious software [1]. Lets dissect it step-by-step.

2.1 What is Malware?

Malicious software is what is meant to be referred to as malware. It is a class of software designed to harm exploit or compromise computer systems in some other way. Malware can take many different forms like these:

Viruses: Viruses bind to legitimate programs and propagate during their operation.
Worms: By taking advantage of weaknesses, worms propagate throughout networks.
Trojans: Trojans trick users into installing them by disguising themselves as trustworthy software.
Ransomware: Files encrypted by ransomware are demanded to be paid for the decryption key.
Spyware: Software that surreptitiously observes and gathers user data is known as spyware.
Adware: Unwanted advertisements are displayed by adware.
Rootkits: Attackers can take over a system by using rootkits.
Botnets: Unknown to their owners botnets are networks of malware-infected computers that can be remotely controlled by an attacker.
Keylogger: A type of malicious software known as a keylogger is one that surreptitiously logs user keystrokes on their keyboard.

2.2 Why is Malware Analysis Important?

Knowing how malware propagates enables security experts to:

Detect and Remove Threats: They can create tools and processes for identifying and getting rid of malware by knowing what it does.
Prevent Future Attacks: You can prevent similar attacks in the future by drawing lessons from past attacks.
Improve Security Systems: By updating systems to counter newly discovered threats this enhances overall security posture.
Forensic Analysis: It helps identify the perpetrators and assess the extent of a breach

2.3 Types of Malware Analysis

Following are the different malware analysis techniques and strategies.

2.3.1 Static Malware Analysis

Static malware analysis is a cybersecurity technique for analysing and comprehending dangerous software (malware) without running it. This form of analysis examines the malware's code and files' properties, code structure, and behaviour, frequently without running the virus on a real system or in a controlled environment. It is an essential strategy for comprehending and defending against many forms of cyber threats[1].

Number of advantages of Static Malware Analysis

- **Safe Analysis:** Because the malware is not executed, there is no danger of it causing harm to the analysis environment. Analysts can instantly assess the malware's possible capabilities and intentions without having to wait for it to execute.
- **Signature Generation:** Static analysis insights can be utilized to generate signatures for antivirus and intrusion detection systems.

Number of disadvantages of Static Malware Analysis

- **Insight into Limited Behaviour:** Because static analysis does not indicate how malware behaves in a real-world setting, some behavioural elements may be overlooked.

- **Obfuscation Difficulties:** Advanced obfuscation techniques might make static analysis of malware code challenging.
- **Failure to Recognize Runtime Behaviours:** Some malware only acts maliciously when executed, which static analysis may not uncover.

2.3.2 Dynamic Malware Analysis

Dynamic malware analysis is a technique used in cybersecurity to examine and comprehend the behaviour of malicious software (malware) in a controlled environment. Unlike static analysis, which includes inspecting the code or structure of malware without running it, dynamic analysis focuses on studying the real runtime behaviour of malware as it runs on a system or within a virtual environment. Dynamic malware analysis is critical for cybersecurity researchers, malware analysts, and incident responders to understand the strategies, methods, and procedures used by malware writers. It aids in the development of better detection systems, the creation of signatures for antivirus software, and the development of ways to mitigate the impact of malware infections. Its important to note though that some sophisticated malware can recognize analysis environments and react differently when run which can make dynamic analysis challenging in specific situations [1].

2.3.3 Hybrid Malware Analysis

Hybrid malware analysis is the process of analyzing and interpreting malicious software (malware) by combining a number of tools and analysis techniques. Learning as much as possible about the behavior capabilities and potential impact of the malware on users systems and networks is the aim of hybrid analysis. With the help of this technique malware behavior can be examined in greater detail by combining static and dynamic analysis methodologies. By attempting to get beyond the limitations of each technique separately hybrid malware analysis provides a more precise and comprehensive evaluation of the traits of the virus and its potential dangers [1].

2.4 Tools Used for Malware Analysis

Malware analysts employ diverse technologies to facilitate their work such as:

Hex Editors: Hex editors are used to examine a files raw data.

Disassemblers: Tools that convert executable files to assembly code are called disassemblers examples include IDA Pro.
Sandboxes: Cuckoo Sandbox is one example of a controlled environment where malware can be safely executed and observed.
Network analyzers: Network Analyzers are tools that monitor network traffic, such as Wireshark.

2.5 Challenges in Malware Analysis

Malware analysis can be challenging due to following:

Obfuscation: Obfuscation techniques used by writers to conceal the true nature of their code.
Polymorphism and metamorphism: Some malware can modify its code on the fly to avoid detection.
Anti-Analysis Techniques: Malware can detect when it is being analyzed in a sandbox and modify its behavior to escape detection.

2.6 Role of Machine Learning

Recently, machine learning has played an important role in malware analysis. Machine learning algorithms can examine massive databases of malware samples, detect patterns, and accurately anticipate the behavior of previously unknown malware. Here are some important characteristics of its role:

2.6.1 Detection and Identification

Signature-Based Detection:

Conventional antivirus programs use signatures that are already known to be active in order to identify malware. This can be made better by machine learning, which can identify trends and abnormalities that point to new, undiscovered malware.

Behavioral Analysis:

Unusual file access patterns, network communications, or modifications to system configurations are examples of suspicious activities that may point to malware. Real-time

evaluation of program behavior and identification of these actions are possible with machine learning models.

2.6.2 Classification

Family Classification:

By grouping malware into families according to common characteristics machine learning helps us comprehend the source and potential consequences of the malware.

File Type and Function Classification:

In order to facilitate faster response times and mitigation strategies it can also classify the different types of files or functions that are targeted by malware.

2.6.3 Feature Extraction

Automated Feature Extraction:

Large datasets of benign and malicious files can be used to extract important features by machine learning algorithms which improves malware detection efficiency and accuracy.

Static and Dynamic Analysis:

A more thorough analysis can be obtained by combining machine learnings capabilities with static analysis which involves analyzing code without actually running it and dynamic analysis which views execution behavior.

2.6.4 Anomaly detection

Network Traffic Analysis:

Network traffic can be scanned by machine learning models for unusual patterns that might point to the spread of malware or the exfiltration of data.

User Behavior Analytics:

Machine learning is able to identify changes in user behavior that may point to the presence of malware.

2.6.5 Threat Intelligence and Prediction

Predictive Analytics:

Using data from the past and present machine learning can predict future malware trends and possible threats.

Threat Intelligence Integration:

A more effective defense mechanism is produced by combining machine learning with threat intelligence streams to help correlate local observations with global threat data.

2.6.6 Evasion Techniques and Adaptation

Adversarial Machine Learning:

Machine learning can help detect and combat the evasion techniques used by malware such as polymorphism and code obfuscation.

Adaptive Learning:

Continuous learning models may adapt to new threats and progressively improve their detection capabilities by incorporating new data into their knowledge base.

2.6.7 Resource Optimisation

By giving priority to the examination of files or actions that are more likely to be dangerous, machine learning models can optimize the usage of computational resources and lessen the overall strain on security systems.

2.6.8 Integration with Security Tools

Integration of SIEM and EDR:

Machine learning enhances Security Information and Event Management (SIEM) and Endpoint Detection and Response (EDR) systems by providing advanced analytics and automated responses to detected threats.

Automation and Orchestration:

The overall effectiveness and efficiency of cybersecurity operations are increased by enabling the automation of tedious procedures and the coordination of complex security workflows. Machine learning in malware analysis enables a more proactive and dynamic approach to cybersecurity by facilitating faster and more accurate threat identification and response. One crucial element of cybersecurity is malware analysis. By understanding how malware functions analysts can develop techniques to recognize thwart and minimize cyber risks. Throughout the process a variety of tools and techniques are employed in both the static and dynamic analysis phases to ascertain the behavior and intent of malicious software. Even with the challenges brought on by advanced malware techniques research and machine learning continue to improve the effectiveness of malware analysis.

Chapter 3

Literature Review

As new malware variants are emerging so quickly and persistently it is becoming more and more obvious that the existing signature-based antivirus detection engines have limitations. These antiquated approaches find it difficult to keep up with the growth and evolution of cyber threats which calls for more sophisticated solutions. The study authors suggest a cutting-edge malware detection model that makes use of ensemble learning strategies in response to this expanding difficulty. This novel approach improves the efficacy and efficiency of the model by relying on minimal yet highly significant features that are extracted from file headers. In comparison to individual classification models the authors show through thorough experimental evaluations that their ensemble learning model performs better. Although it might seem like a slight improvement the model routinely beats its competitors by a significant amount. In particular the ensemble model that has been suggested demonstrates exceptional predictive accuracy attaining a noteworthy accuracy rate of 0. 998. Furthermore its robustness and dependability are highlighted by the exceptionally low false positive rate it maintains—just 0. 002—when identifying malware that has never been seen before. The study offers a comprehensive comparison of the suggested ensemble model with a range of alternative machine learning strategies emphasizing the benefits and promise of machine learning-based techniques in comparison to more conventional signature-based methods. In the field of antivirus detection the authors present a strong argument for the adoption of machine learning-based solutions by highlighting the advantages of their model. In the battle against new and sophisticated cyber threats this move towards sophisticated machine learning techniques is a major breakthrough. [5], [6].

According to the study metamorphic malware presents serious obstacles for conventional signature-based antivirus software underscoring the urgent need for creative and flexible solutions that can accurately identify and categorize these sophisticated threats. Static signature-heavy conventional methods face a significant challenge from

metamorphic malware which can change its code structure while maintaining its malicious capabilities. Because metamorphic malware is dynamic it requires a more advanced detection system that can comprehend and analyze malware behavior patterns instead of depending just on predefined signatures. In order to overcome these obstacles the researchers suggest a unique classification technique that centers on examining malware behavior using an extensive dataset made up of API calls made on the Windows operating system. Adware Backdoor Downloader Dropper Spyware Trojan Virus and Worm are just a few of the many malware kinds included in this dataset which offers a comprehensive and varied depiction of typical threats found in the wild. The study intends to use this dataset to capture the subtle behavioral differences between various malware families which are essential for creating efficient detection systems. Long Short-Term Memory (LSTM) networks have been selected as the classification method for this study. LSTM networks are a sophisticated technique that are well-known for their ability to handle sequential data and capture long-term dependencies within sequences. LSTM networks are especially well-suited for deciphering the malwares sequence of API calls which enables a more profound comprehension of the behavioral patterns connected to various malware kinds. With an F1-score of 0.83 and an accuracy rate of up to 95 percent the outcomes of using the LSTM classifier on the dataset are quite remarkable. These metrics demonstrate how well the suggested method works to reliably identify and categorize different types of malware according to their behavioral traits. Along with the LSTM models success the researchers used binary and multi-class malware datasets in a number of comprehensive experiments. Whether dealing with malware in binary form or as part of a multi-class classification problem this method shows the adaptability and resilience of the LSTM model. The LSTM models versatility in classifying data further highlights its potential as an effective malware detection tool. This study made a substantial contribution by creating a novel dataset intended only for the analysis of API calls made on Windows operating systems. Such a customized dataset has not been available for malware detection research before so this contribution will be especially helpful in expanding the field. The researchers have greatly improved accessibility for the larger research community by making the dataset publicly available on GitHub. This transparency promotes cooperation and creativity in the industry by stimulating additional study and advancement in malware detection. The studys thorough overview covers the difficulties in dealing with metamorphic malware the drawbacks of conventional detection techniques and the suggested remedy that combines the recently created dataset and LSTM-based classification. A significant advancement in malware detection research has been made possible by the high accuracy attained in the experiments and the datasets public accessibility. As a result the security environment as a whole is strengthened and more efficient and flexible ways to counteract changing cyberthreats are made possible [7].

By concentrating on API call sequences—a critical component of malwares system

interaction—the paper attempts to address the challenging task of comprehending and describing malware behavior. Because they are complex and require a lot of data traditional methods—which entail building comprehensive behavioral graphs for every malware instance—are frequently impracticable. Using word embedding techniques the paper presents a novel approach to overcome these limitations. With the help of this method API functions are represented in a high-dimensional space making it easier to understand their contextual relationships and to create behavioral graphs that are both straightforward and educational. The study also suggests a novel clustering technique that groups API functions according to comparable contextual characteristics. In addition to streamlining the analysis of malware call sequences this clustering technique improves the efficiency with which patterns and anomalies can be found. The research presents a Markov chain-based method for malware detection by examining the variations in call sequences between malware and genuine software (benign). This produces a semantic transition matrix that precisely depicts the connections and changes between API functions enabling a more precise and comprehensive understanding of malware behavior. As evidenced by their average detection precision of 0. 990 and low false positive rate of 0.010 the suggested models perform exceptionally well. The models high degree of accuracy suggests that they can reliably and effectively distinguish between malicious and non-malicious behavior. The study also presents a predictive methodology that looks ahead and recognizes potentially harmful API call sequences from their original functions. With an amazing average accuracy of 0. 997 this method demonstrates its capacity to foresee and identify threats before they materialize. This proactive prediction capability is a major breakthrough offering a tactical advantage over conventional post-execution detection techniques by enabling the preemptive blocking of malicious payloads. The research offers an extensive and innovative approach to malware analysis by addressing both detection and prevention. By doing so it offers advancements over traditional methods that strengthen cybersecurity defenses as a whole [8].

Cloud service providers (CSPs) have expanded quickly bringing with them a wide range of features and services that make it easy for businesses to move to cloud-based infrastructures. These solutions are flexible and scalable to meet their needs. Infrastructure as a Service (IaaS) stands out among these services because it enables customers to rent server access for processing and storing needs. Because of its affordability and scalability Infrastructure as a Service (IaaS) has gained popularity however it has also given rise to serious privacy and cybersecurity concerns. Modern techniques for malware detection in cloud environments are required because the use of malware to compromise data or interfere with cloud services has become a serious issue. Research on cloud-based malware detection has increased in response to these worries. Specifically this study uses process-level performance metrics from virtual machines in cloud settings to investigate the efficacy of online malware detection. The impact

of various machine learning models on these process-level features was used to evaluate the models ability to detect malware. Support Vector Classifier (SVC) Random Forest Classifier (RFC) K-Nearest Neighbor (KNN) Gradient Boosted Classifier (GBC) Gaussian Naive Bayes (GNB) and Convolutional Neural Networks (CNN) are among the models that were assessed. According to the studys findings neural network models—in particular Convolutional Neural Networks or CNNs—show superior accuracy when it comes to identifying malwares effects on the features of cloud-based virtual machines at the process level. CNNs which are well-known for their ability to handle intricate patterns and features demonstrated exceptional efficacy in differentiating between malicious and benign activity in cloud environments. Using a large dataset of 40680 samples—including both benign and malicious ones—the models were thoroughly tested validated and trained. This dataset was painstakingly created by running different malware families in a real-time cloud environment and gathering comprehensive process-level data. The outcomes of the study highlight the significance of utilizing cutting-edge machine learning methods to improve malware identification in cloud settings. Organizations may strengthen their capacity to identify and address malware threats more successfully by utilizing advanced models like CNNs. This will protect their cloud-based infrastructure and guarantee the integrity and security of their data and services [9].

The threat landscape has grown increasingly severe and complex due to the growing use of digital services and the spread of sophisticated malicious software. Since advanced malware can lead to serious consequences like data corruption identity theft and other cybercrimes it is critical that effective detection mechanisms be developed immediately. Preventing malware from causing extensive harm is crucial for preserving cybersecurity. Even though many malware detection techniques have been proposed by researchers there are still issues especially with correctly identifying a variety of malware types such as zero-day attacks. These difficulties are made worse by the sophisticated evasion and obfuscation strategies used by attackers as well as the quick and continuous creation of new malware variants with a variety of harmful traits. Although the review papers that are currently available provide important insights into these challenges they frequently do not provide a thorough overview that links various approaches to analysis and detection to the different types of data that are used. These relationships are essential for furthering research and creating practical mitigation plans. In order to close this gap the current survey offers a thorough analysis of malware detection models along with a comprehensive feature representation taxonomy that classifies different approaches and procedures utilized in malware analysis. By connecting each approach to widely used data types it expands the categorization of malware detection techniques and helps to clarify the ways in which various detection techniques interact with various kinds of data. Instead of concentrating only on analysis techniques the survey looks at feature extraction methods by analyzing

their underlying techniques. Knowing the advantages and disadvantages of various techniques in the context of diverse malware detection scenarios is made easier by this nuanced classification. Finally the paper highlights the need for creative solutions to counter evolving threats by providing a comprehensive overview of the fields current challenges and outlining possible future research directions. Through this survey we hope to close the gap between data types and analysis methods which will help direct future research efforts and increase the overall efficacy of malware detection and prevention strategies [10].

Malware sometimes known as malicious software is a serious cybersecurity threat that aims to damage computer systems including single machines servers and large networks as well as interfere with operations and obtain unauthorized access. Owing to their extensive use and built-in weaknesses Windows operating systems are especially vulnerable to malware attacks. Because of its extensive use malicious actors frequently target Windows in an effort to steal confidential data or cause harm. Because discovering new or unknown malware strains is frequently difficult for traditional signature-based detection techniques which rely on identifying known malware signatures combating the malware threat is a constant and changing challenge. Systems are exposed to novel and sophisticated attacks since these signature-based systems are only effective against threats that have already been detected. A recent paper proposes a novel approach to malware detection that aims to improve the identification of known and unknown threats in order to get around this limitation. The suggested approach makes use of a simple yet powerful model that uses characteristics taken from the Portable Executable (PE) headers of these files to distinguish between executable files that are malicious and those that are benign. Important information about the file structure and its properties can be found in the PE headers and this information may be suggestive of malicious activity. The model analyzes these features to find subtle anomalies that could be signs of malware. Support Vector Machine (SVM) Decision Tree Random Forest and Naive Bayes classifiers are among the machine learning methods used to categorize the files. Applying the Random Forest classifier to a dataset containing data from the file header optional header and section header has shown it to be the most accurate of these. Due to its capacity to handle complicated datasets and recognize nuanced patterns in the data this classifier has demonstrated superior performance. By focusing on features that are frequently missed by conventional methods this novel approach offers a significant advancement in malware detection. This method improves the capacity to recognize and react to new malware fortifying overall cybersecurity measures by focusing on hitherto unseen threats with a model that makes use of comprehensive PE header information. Enhancing detection accuracy and offering a strong framework to protect sensitive data from ever-changing cyber threats are two benefits of using machine learning to analyze these features [11].

3.1 Summary of Related Work

Table 3.1: Summary of Related Work.

Author - Year of Publication	Detection Techniques / Models	Strengths	Weaknesses	Category
Zelinki et al. 2019 [5]	SVM, NB, KNN, MLP, LDA, and DT	1. Improved Accuracy and Robustness 2. Efficient Use of Resources 3. Faster Training and Inference 4. Reduced Overfitting 5. Enhanced Interpretability 6. Simplification of Model Complexity 7. Cost-Effectiveness 8. Resilience Against Adversarial Attacks 9. Scalability and Deployment Flexibility: 10. High Performance on Limited Data	1. Minimum feature set was used (weakness) 2. Risk of Information Loss 3. Limited Representation 4. Inadequate Discrimination Ability 5. Reduced Model Flexibility 6. Less Resilience to Variability 7. Challenges in Feature Selection 8. Sensitivity to Feature Changes 9. Difficulty in Generalization 10. Potentially Increased False Positives/Negatives 11. Balancing Complexity and Performance	1. Static and Dynamic Analysis i.e., Hybrid Analysis. 2. Relied on Random Forest.

Table 3.1: Summary of Related Work.

Author - Year of Publication	Detection Techniques / Models	Strengths	Weaknesses	Category
J. C. Kimmell, et al. 2021, [12]	SVC, RFC, KNN, GBC, GNB, CNN	1. Real-time Threat Detection 2. Scalability 3. Adaptability to Evolving Threats 4. Enhanced Accuracy and Precision 5. Automated Response and Mitigation 6. Reduced Workload for Security Teams 7. Cost-effectiveness 8. Customization and Flexibility 9. Threat Intelligence and Pattern Recognition 10. Compliance and Regulatory Alignment	1. Small number of samples used (weakness) 2. Over-Reliance on Historical Data 3. Data Privacy and Sensitivity 4. False Positives and Negatives 5. Adversarial Attacks and Evasion Techniques 6. High Resource and Computational Requirements 7. Lack of Interpretability and Explainability 8. Model Bias and Fairness 9. Model Degradation over Time 10. Dependency on Skilled Personnel 11. Cost of Model Training and Maintenance	Dynamic Malware Analysis and Online Malware Detection.
S. Naz et al. [9]	KNN, GNB, RF, DT, BNB, ADB, LR, SVM	-	Only for Windows Executables (weakness)	Static Analysis
Daeef et al. [6]	SVM, KNN, RF	-	No deep analysis of API call (weakness)	Static and Dynamic i.e., Hybrid and API calls

Table 3.1: Summary of Related Work.

Author - Year of Publication	Detection Techniques / Models	Strengths	Weaknesses	Category
Namita et al. [13]	DT, SVM, LR, and KNN	1. Efficient Feature Representation 2. Dimensionality Reduction 3. Preservation of Information 4. Versatility in Model Choice 5. Adaptability to Various Malware Types 6. Learning from Large Datasets 7. Automation and Real-Time Detection 8. Continuous Improvement and Adaptation 9. Cost-Effective Detection	1. Semantic Information Loss with TF-IDF 2. Inability to Capture Context 3. Sensitivity to Feature Selection and Parameters 4. Limited to Bag-of-Words Representation 5. Data Imbalance Issues 6. Data Privacy and Sensitivity 7. Overfitting and Generalization 8. Adversarial Attacks 9. Resource Intensive Training and Inference	Dynamic Analysis + API Calls
Naman Aggarwal et al. [11]	SVM, Logistic Regression, Neural Network, Neural Network with Classes	1. Non-Intrusive Analysis 2. Rapid Initial Assessment 3. Identifying Suspicious Behaviour 4. Dependency Analysis 5. Signature Detection 6. Resource and Section Inspection 7. Metadata Analysis 8. Heuristic and Pattern Matching 9. Automated Processing 10. Risk Mitigation	1. Limited Behavioural Insight 2. Evasion Techniques 3. Dynamic Code Generation 4. Polymorphic Malware 5. Encrypted Payloads 6. Dependency on Known Signatures 7. False Positives and Negatives 8. Incomplete Analysis 9. Fileless Malware 10. Time Sensitivity 11. Obfuscated Code	Static Malware Analysis + API Calls

Table 3.1: Summary of Related Work.

Author - Year of Publication	Detection Techniques / Models	Strengths	Weaknesses	Category
Kai Lu et al. [14]	Decision Tree (DT), Random Forest (RF), Logistic Regression (LR), and K-Nearest Neighbour (KNN)	1. Improved Malware Detection Accuracy 2. Efficient Resource Utilization 3. Enhanced Performance and Speed 4. Reduced False Positives and Negatives 5. Cost-Effectiveness 6. Scalability 7. Robustness to Evolving Malware	1. Limited Applicability 2. Dependency on Feature Selection 3. Overfitting or Underfitting 4. Data Quality and Quantity 5. Scalability Challenges 6. Dependency on Correlation Information 7. Performance Trade-offs	Static Dynamic Analysis i.e., Hybrid. + Network traffic Analysis

3.2 Research Gap

APIs (Application Programming Interfaces) have been the backbone of modern software development in recent years, enabling seamless interaction across various software components, services, and platforms. While multiple studies have addressed various aspects of APIs, there is a substantial vacuum in the entire analysis of API calls, particularly in examining deeper and unknown aspects. Existing research focuses mostly on high-level usage patterns, security issues, and performance optimization, leaving other critical elements of API calls unexplored.

This study seeks to fill that void by going into the uncharted terrain of API calls to unearth hidden insights and potential issues that have not received appropriate consideration in the existing body of literature. The primary focus will be on identifying subtle patterns, behaviours, and implications of API requests that can have significant ramifications for software development, security, and overall system performance.

In essence, the problem statement identifies a requirement to comprehend the behavioural patterns, faults, semantics, resource impact, security threats, and evolving trends connected with API calls. Addressing these issues can improve the comprehension, reliability, efficiency, and security of API interactions in an ever-changing technological context.

Finally, the thesis aims to further the field of malware analysis by introducing a static analysis approach that focuses on API call sequences to detect malicious behaviour. This study aims to provide a more comprehensive and resilient method for identifying and classifying malware by leveraging static analysis and machine learning techniques, ultimately contributing to the improvement of cybersecurity measures in the face of evolving and sophisticated threats.

3.3 Research Questions

Current approaches to detect malware frequently encounter difficulties in accurately identifying malware activity The main issue is the limited effectiveness of existing malware analysis methods, which mostly rely on static characteristics. The current malware are dynamic and versatile, static analysis frequently misses these characteristics, producing false negatives and enabling hostile entities to avoid detection. Previously researcher did not investigate deep analysis behaviour of API calls. Based on the above issues, we proposed the following research questions:

1. "What are the key characteristics and patterns in API call sequences that can be indicative of malware behaviour, and how can these patterns be effectively extracted and analysed?"

2. "How does the diversity of malware families impact the development of a robust API call sequence-based malware detection system, and what strategies can be employed to address this challenge?"

3. "What role does static feature extraction play in augmenting the analysis of API call sequences for malware detection, and how does it contribute to improving the overall detection accuracy?"

4. "How do different types of malwares (e.g., ransomware, spyware, trojans) exhibit distinct API call sequence patterns, and how can these patterns be leveraged for more precise malware classification?"

5. "What are the potential limitations and challenges associated with using API call sequences for malware analysis, and how can these challenges be mitigated to ensure the reliability and effectiveness of detection methods?"

6. "In what ways can machine learning algorithms be applied to analyse API call sequences and static features for automated malware detection, and what are the trade-offs between different algorithmic approaches in terms of accuracy and computational efficiency?"

7. "How does the dynamic nature of malware evolution impact the adaptability and effectiveness of API call sequence-based detection methods, and what strategies

can be employed to maintain resilience against emerging threats?"

8. "What is the significance of incorporating contextual information, such as system calls and environmental factors, in conjunction with API call sequences for a more comprehensive understanding of malware behaviour and improved detection capabilities?"

9. "How do different operating systems influence the characteristics of API call sequences exhibited by malware, and how can cross-platform considerations be addressed in the development of a universal malware detection approach?"

10. "What are the ethical implications and privacy concerns associated with the collection and analysis of API call sequences for malware detection, and how can these concerns be balanced with the need for robust cybersecurity measures?"

3.4 Problem Statement

Cybersecurity is seriously threatened by the growing sophistication of malware, which calls for sophisticated methods for prompt identification and mitigation. Finding dangerous activity using API call sequences has become a promising technique in the field of malware investigation. Nevertheless, current approaches frequently encounter difficulties in accurately identifying malware activity, especially when depending exclusively on static feature extraction. The main issue is the limited effectiveness of existing malware analysis methods, which mostly rely on static characteristics. Because current malware is dynamic and versatile, static analysis frequently misses these characteristics, producing false negatives and enabling hostile entities to avoid detection. Evidently a comprehensive and dependable technique for analyzing malware through API call sequences is needed with an emphasis on enhancing detection efficiency and precision through dynamic behavioral analysis.

This study focuses on the following key issues:

1. Dynamic Malware Behaviour Detection: Methods that focus on dynamic components specifically API call sequences should be developed to improve the detection of malware behaviour. This means being aware of the ways in which malware interacts with system APIs during operation.
2. Reduce False Positives: You can decrease the possibility of false positives by refining the feature extraction process. Increase the accuracy of malware detection to reduce harm to legitimate software and reduce the workload of security personnel responding to false alarms.

3. Managing Polymorphic Malware: Make arrangements for the effective management of polymorphic malware. Examine strategies for managing the variation in API call sequences amongst instances of the same malware strain in order to guarantee a detection system that is both thorough and flexible.
4. Scalability and Resource Efficiency: Create a solution that is both scalable and resource-efficient so that it can cope with the increasing quantity and sophistication of malware samples. Take into account the computing burden linked to dynamic analysis and suggest optimization strategies while maintaining detection accuracy.
5. Adaptability to Changing Threat Landscape: Create a framework that is flexible enough to change with the strategies used by malware producers. Examine methods for updating the analysis model in light of the continually evolving malware landscapes in order to keep ahead of new threats and guarantee long-term relevance.

"The deep analysis of malware has not been done in the existing literature, therefore, it is difficult to identify and analyse refined API pattern and how to identify whether a binary is a benign or malicious".

The suggested study aims to improve malware analysis by tackling these issues and giving cybersecurity experts more dependable and effective tools to identify and counteract emerging malware threats.

3.5 Research Objective

1. Deeper Analysis of API Calls: Existing research tends to focus on high-level usage patterns, security issues, and performance optimization of APIs. However, there is a need for a more in-depth analysis of API calls, particularly examining deeper and unknown aspects such as subtle patterns, behaviours, and implications of API requests.

2. Understanding Behavioural Patterns and Faults: Research thoroughly examining the behavioural patterns faults and semantics related to API calls is scarce. Gaining an understanding of these factors is essential to enhancing the comprehension dependability and effectiveness of API interactions.

3. Resource Impact and Security Threats: The current body of literature mainly focuses on high-level aspects of APIs leaving room for resource impact and security threats associated with API calls to be under- or overly covered. To improve the security and effectiveness of API interactions research should focus on these important areas.

4. Evolving Trends in API Calls: Research frequently lags behind the quickly changing patterns in API utilization. Maintaining the relevance and currency of research

requires a thorough examination and analysis of the dynamic API call landscape.

5. **Static Analysis of API Call Sequences for Malware Detection:** The purpose of the thesis is to advance the field of malware analysis by introducing a static analysis approach that focuses on API call sequences for the purpose of detecting malicious behavior. This specific aspect represents a gap in the existing research concerning API calls and malware detection.

Addressing these research objectives would lead to a more comprehensive understanding of API calls, which is crucial for improving software development, security, and system performance in the rapidly evolving technological context. Additionally, the integration of static analysis and machine learning techniques for malware detection can significantly advance cybersecurity measures in the face of evolving and sophisticated threats.

Chapter 4

Proposed Technique or Methodology

4.1 Proposed Model / Methodology

4.1.1 Theoretical Analysis of Malware

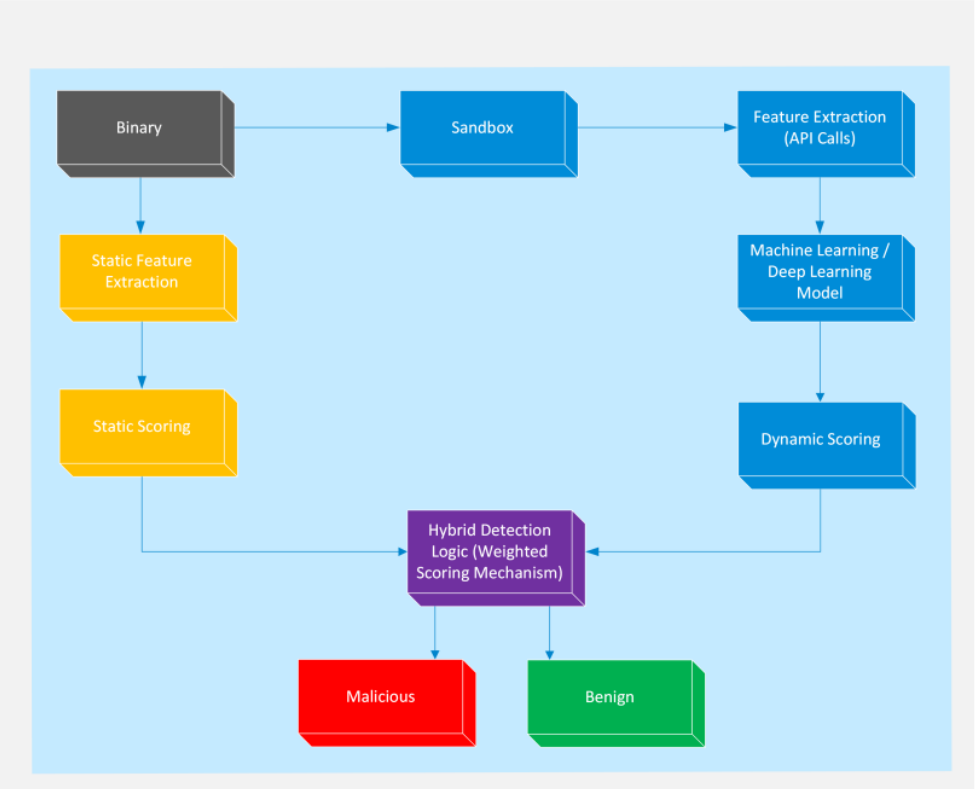
A detailed theoretical analysis of malware frequently looks at a number of factors, such as its behavior, modes of dissemination, and evasion strategies. Understanding how the malware uses API (Application Programming Interface) calls is one important factor.

4.1.2 Proposed Model

Our proposed model is a Hybrid model, which uses a Machine Learning Model together with a combination of Static and Dynamic Analysis techniques. An executable file(s) can be run in a sandbox environment which also submits it to a trained Machine Learning Model. The Sandbox also checks the number of API Calls and also notes a unique pattern of API Call Sequence. After that a Hybrid Detection Logic (HDC) was used to determine whether the executable file that was submitted was malicious or not.

Research and analysis of malwares API-related actions must be ongoing in order to detect and mitigate it effectively. It is well known that malware is always changing. This study gives a broad overview of the theoretical characteristics of malware with a focus on API calls though the specifics can vary based on the type and intent of the infection. Utilizing specialized tools and cybersecurity expertise practical analysis comprises dissecting and understanding the intricacies of each malicious entity.

Figure 4.1: Flow/Block Diagram



Understanding the functioning and possible threats of malware requires a thorough analysis of its behavior. In order to thoroughly examine malware behavior this chapter presents a practical method that integrates dynamic analysis static feature extraction and machine learning-based classification.

4.2 Observation and Recording of API Call Sequences

The first part of the technique is to run the malicious executable in a controlled environment and watch and record the API call sequences that are produced. Because of this controlled environment the behavior of the malware can be observed without endangering the host system. The series of API calls the executable makes during execution allow it to record the exchanges between the malware and the operating system or other software components. These sequences are essential for additional analysis and provide insightful information about the malwares behavior.

4.3 Conversion into Static Features

The API call sequences are converted into a set of static features after they are recorded. Features that are derived from the API call sequences and provide information about the malwares behavior are called static features. Through this transformation multiple attributes are extracted from the sequences including frequency of API calls temporal patterns and inter-call interdependence. A thorough description of the malwares actions can be obtained by examining these static characteristics. This step allows for a deeper understanding of the functionality of the malware by bridging the gap between the static analysis of extracted features and the dynamic analysis of real-time behavior.

4.4 Machine Learning-Based Classification

To classify the executable as malicious or benign the final step of the technique uses machine learning techniques to analyze the set of static features. An executables known benign or malicious status is indicated by the labels on the machine learning model that was trained on this data to create this classification. The static features that are taken out of the API call sequences teach the model patterns of malicious behavior during the training phase. Once trained the model evaluates static features of new executables and recognizes traits that point to malware. This allows the model to classify new executables with accuracy. This step improves malware classification accuracy and efficiency by utilizing machine learning which makes it possible to identify sophisticated and new threats. Using a combination of machine learning-based classification static feature extraction and dynamic analysis the presented method efficiently examines malware behavior. This approach offers a comprehensive framework for malware identification and classification by extracting static features capturing API call sequences and applying machine learning algorithms. It is a useful tool in the

ongoing fight against malicious software because of its capacity to identify complex and emerging threats.

4.5 Scoring on the basis of Static Feature

The following are involved in scoring based on static features:

4.5.1 Define the Features:

We identify and define the features that are pertinent to the scoring in this section. These characteristics are static which means they dont alter over time or in various situations. To define and identify the features we make use of the following:

1. Signatures
2. Strings
3. Entropy
4. Header Info
5. Impacts
6. Obfuscation Indicators

1. Signatures

Signatures provide a quick and reliable means of identifying threats making them an essential part of the static malware analysis toolbox. These are the different kinds of signatures;

a. Byte Signatures: Particular byte sequences that are exclusive to a certain malware infection.

b. Hash Signatures: Example of known malware files hash values like MD5, SHA-1, SHA-256. These provide a quick way to identify exact matches.

c. Code Signatures: Specific code sequences or patterns that suggest malevolent intent..

d. File Structure Signatures: Features of the file structure like sections or strange headers that are typical of malware.

2. Strings

One common technique used in the context of static malware detection to identify potentially dangerous software is the analysis of strings included in an executable or binary file. This process involves taking information about the binarys behavior and intended use by extracting and analyzing human-readable strings from the binary. The use of strings in static malware detection involves the following important features.

a. Signature-based Detection
b. Heuristic Analysis
c. Contextual Analysis
d. Yara Rules
e. Localized Indicators
f. Static Analysis Tools

3. Entropy

Entropy is a key component of static malware detection and is commonly employed to identify suspicious files based on their complexity and randomness. To measure the unpredictability or randomness of a dataset entropy is used in cybersecurity more especially in static malware analysis. By recognizing methods that malicious software frequently employs to evade detection such as obfuscation or compression entropy can be used to detect malware. Given its ability to disclose information regarding a files complexity and potential for concealment entropy is a valuable metric for detecting static malware. Security researchers can identify files with abnormally high entropy and target those that are more likely to be harmful.

4. Header Info

The header information can offer crucial information about an executable files potential maliciousness when looking at one (such as a Windows Portable Executable or PE file). This involves analyzing the metadata and structural components of the file without actually running it. Important header elements that are frequently looked at are as follows:

a. DOS Header (IMAGE_DOS_HEADER)
b. PE Header (IMAGE_NT_HEADERS)
c. Optional Header (IMAGE_OPTIONAL_HEADER)
d. Section Headers (IMAGE_SECTION_HEADER)
e. Import Table

f. Export Table
g. Resource Table
h. Debug Information
i. TLS (Thread Local Storage) Callbacks

5. Imports

Analyzing malicious software (malware) without really running the code is known as static malware analysis. It involves analyzing the structure, content, and metadata of the malware to understand its behavior, identify its components, and determine its potential impact.

6. Obfuscation Indicators

Obfuscation techniques are used by malware developers to hide malicious code and evade detection by security software. Recognizing indicators of obfuscation can be crucial for identifying malware. Here are some common obfuscation indicators in static malware detection:

a. Packing and Compression
b. Encryption and Encoding
c. Code Obfuscation
d. Resource Obfuscation
e. Anti-Analysis Techniques
f. Polymorphism and Metamorphism
g. Suspicious File Attributes
h. Suspicious Imports and APIs

4.5.2 Extract Features:

Implement methods to extract these features from your data. This can involve:
Textual Data: Extract features like word frequency, TF-IDF scores, sentiment scores, etc.
Numerical Data: Use statistical measures like mean, median, standard deviation, etc
Categorical Data: Convert categories into numerical values using techniques like one-hot encoding.

4.5.3 Feature Normalization:

Normalize or standardize the features to ensure they are on a similar scale. This can help in improving the performance of the scoring model.

4.5.4 Feature Weighting:

We can assign weights to each feature based on their importance. This can be done manually by domain experts or automatically using techniques like correlation analysis or feature importance from machine learning models.

4.5.5 Score Calculation:

We have calculated the score based on the weighted sum of the features.
Static Score = $\sum$(Normalized Factors)
Dynamic Score = Score generated by ML model
Score = $T_0 \cdot$ Static Score $+ T_1 \cdot$ Dynamic Score

4.5.6 Thresholds and Binning:

Set thresholds or create bins to categorize the scores into different levels (e.g., low, medium, high). This can help in interpreting the scores more effectively.

4.6 Extracting API Call Function Names from a Binary Executable

Extracting API call function names with corresponding DLLs and addresses from a binary is crucial to the malware detection phase. That is all done with automation i.e. we may give the name of the folder containing the binary executables and it returns the required results. The explanation of the algorithm that can extract the name of the API function calls with its corresponding Dlls and show addresses is mentioned below.

4.6.1 Algorithm Explanation

The algorithm described in the provided Python script focuses on extracting and displaying detailed information about API calls (imported functions) from a Portable Executable (PE) file using the pefile library. Here's a comprehensive explanation of the algorithm, broken down into key steps:

Step 1: Load the PE File The algorithm starts by loading and parsing the PE file. This is done using the pefile library, which simplifies the task of handling the complex structure of PE files.

Details:

Initialize PE Object: The pefile.PE(pe_path) function is called to load the PE file located at the given pe_path. **Parse PE Structure:** The pefile.PE object automatically parses the various headers and sections of the PE file, making it ready for analysis.

Code Snippet:

pe = pefile.PE(pe_path)

Step 2: Extract API Calls Next, the algorithm extracts the details of imported functions from the PE file's import table. The import table lists all functions that the executable imports from external DLLs.

Details:

1. Iterate Over DLL Entries: The algorithm iterates over pe.DIRECTORY_ENTRY_IMPORT, which contains entries for each DLL from which functions are imported.
2. Extract DLL Name: The DLL name is extracted for each DLL entry.
3. Iterate Over Imported Functions: To access each imported function the algorithm loops over the imports attribute within each DLL entry.
4. Extract Function Details: For each imported function, the algorithm extracts:

function_name: The name of the function, if available.

address: The memory address where the function will be loaded.

ordinal: The ordinal number (API ID) of the function.

```python
import pefile

def extract_api_calls(pe_path):
    # Load the PE file
    pe = pefile.PE(pe_path)

    api_calls = []

    # Iterate through the imported DLLs and functions
    for entry in pe.DIRECTORY_ENTRY_IMPORT:
        dll_name = entry.dll.decode('utf-8')
        for imp in entry.imports:
            function_name = imp.name.decode('utf-8') if imp.name else None
            api_calls.append({
                'dll': dll_name,
                'function': function_name,
                'address': imp.address,
                'id': imp.ordinal  # Always include the ordinal (API id)
            })

    return api_calls

# File Input
pe_path = input("Enter filename to extract API Function Calls with corresponding DLLs: ")
api_calls = extract_api_calls(pe_path)

# Output
for api_call in api_calls:
    function_info = f"{api_call['function']} (Ordinal {api_call['id']})" if api_call['function'] else f"Ordinal {api_call['id']}"
    print(f"API Call: {function_info} from {api_call['dll']} at address {hex(api_call['address'])}")
```

API Call Extraction Pseudo code

Step 3: Store API Call Details The extracted details of each API call are stored in a list of dictionaries. Each dictionary represents an API call with the following keys: 'dll', 'function', 'address', and 'id'.

Details:

1. Initialize List: An empty list api_calls is initialized to store the details of all API calls.

2. Create Dictionary for Each API Call: For each imported function, a dictionary is created with the extracted details.

3. Append Dictionary to List: The dictionary is appended to the api_calls list.

Step 4: Display API Call Details Finally, the algorithm prints out the collected details of each API call in a readable format.

Details:

1. Iterate Over API Calls: The algorithm iterates over the api_calls list.

2. Check for Function Name: For each API call, it checks if the function$_{n}ameisavailable.Ifnot,itus$

3. Print Details:$Itprintsthe'functionname(orordinalifthenameisnotavailable)',' DLLname',' address$

4.6.2 Summary of Algorithm

In order to extract and present comprehensive information about imported functions the algorithm efficiently parses a PE file. In order to extract pertinent information

(DLL name function name address and ordinal) load the PE file loop through the import table store the information in an organized manner and print the outcomes. This procedure is necessary to comprehend an executables dependencies which helps with malware detection security analysis and reverse engineering. The PE file format is otherwise complicated and detailed but using the pefile library makes handling it much simpler.

Chapter 5

Experimental Setup

In this section, we talk about the experiments we ran to get the outcomes we did.We use five distinct models for experimentation: Decision Tree, Random Forest, Naive Bayes, K-Nearest Neighbor, and Support Vector Machine. Support Vector Machines, also known as support vector networks, are supervised max-margin models used in machine learning that analyze data for regression and classification along with corresponding learning techniques [15]. Regression and classification can be accomplished with the non-parametric supervised learning technique known as the K-Nearest Neighbors algorithm (k-NN) [16]. A popular ensemble learning technique for classification, regression, and other problems is called Random Forests or Random Decision Forests. It works by building a large number of decision trees during the training phase [17]. A family of linear "probabilistic classifiers" known as Naive Bayes classifiers makes the assumption that, given the target class, the features are conditionally independent [18]. The Decision Tree model is a computation paradigm where an algorithm is essentially viewed as a decision tree, that is, as a series of adaptively executed queries or tests, where the results of one test might affect the results of subsequent tests [19].

We performed the experiments on a balanced and an unbalanced dataset, as the original dataset was naturally unbalanced. In this section, we also explain the evaluation measures that we use to analyze our results. Firstly, we discuss all the experiments and then proceed to analyze their results.

5.1 Experiment

To initiate the process, a comprehensive publicly available dataset [20] comprising labeled samples of both benign and malicious executables is amassed. This dataset encompasses a diverse range of malware samples spanning various API call sequences,

alongside file hashes. This dataset consists of 43,876 API call sequences divided into two classes—malware, with 42,797 samples, and legitimate, with 1079 samples.

5.1.1 Summary of Dataset

Type	No of Samples
Benign	1079
Malicious	42,797
Total	43,876

Table 5.1: Summary of Dataset

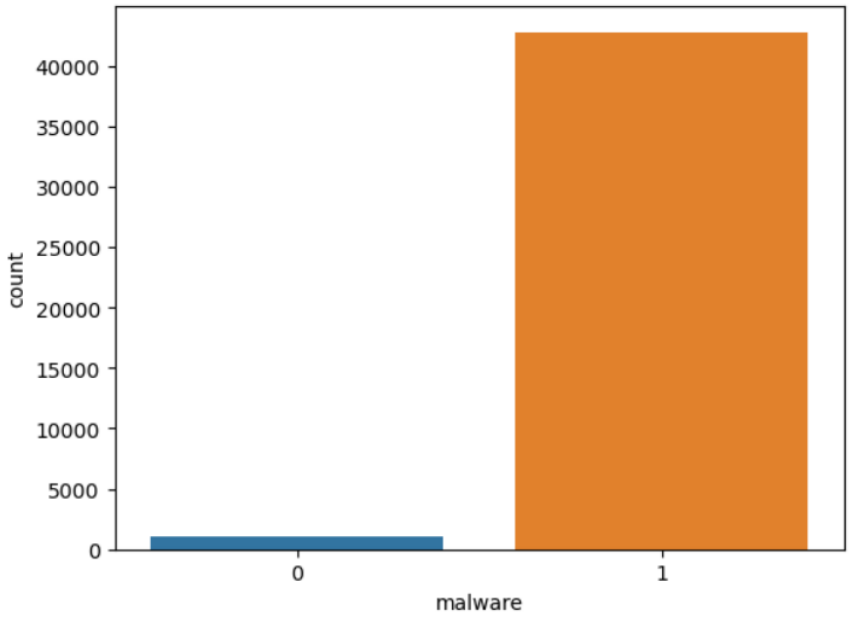


Figure 5.1: Unbalanced Dataset Summary

The dataset was made after running the executable in a controlled environment of Cuckoo Sandbox.

Class	No of API Calls
Benign	200
Malicious	252

Table 5.2: Total number of distinct API calls per class

5.1.2 Experiment using Support Vector Machine - Unbalanced Dateset

To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. Following results were obtained;

Type	Count
True Positive (TP)	153
True Negative (TN)	12819
False Positive (FP)	184
False Negative (FN)	7

Table 5.3: Confusion Matrix of SVM - Unbalanced Dataset

Discussion on SVM - Unbalanced Dataset Results

The confusion matrix shows the following;

- that 153 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12819 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 184 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 7 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.2: Confusion Matrix - SVM - Unbalanced Dataset

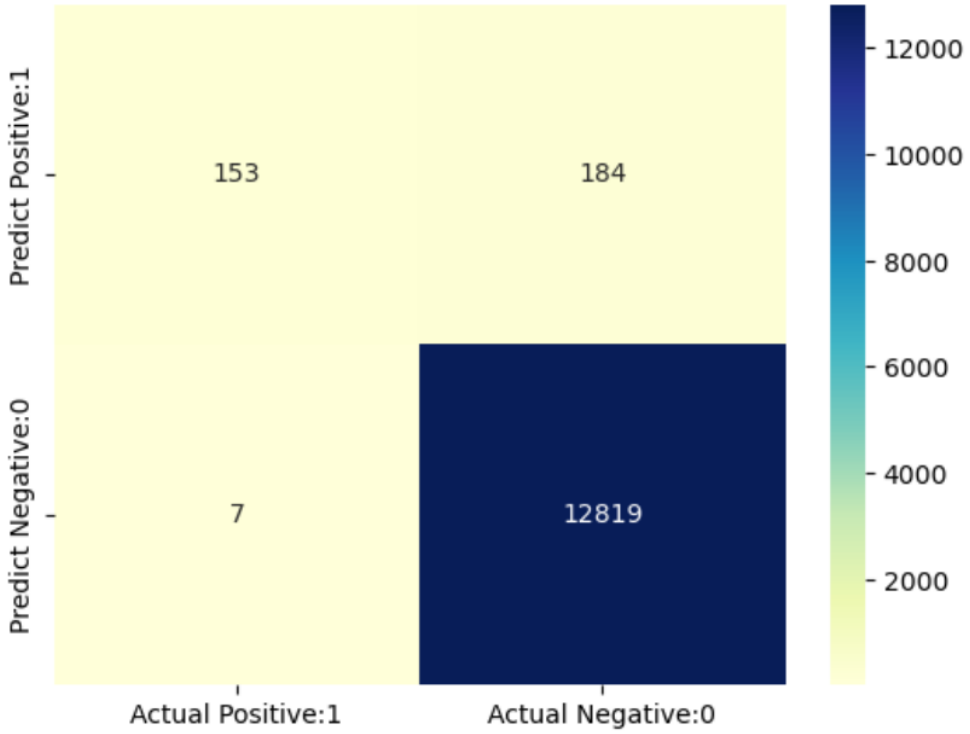


Figure 5.3: Accuracy - SVM - Unbalanced Dataset

	precision	recall	f1-score	support
0	0.96	0.45	0.62	337
1	0.99	1.00	0.99	12826
accuracy			0.99	13163
macro avg	0.97	0.73	0.80	13163
weighted avg	0.99	0.99	0.98	13163

5.1.3 Experiment using Random Forest - Unbalanced Dateset

To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.4: Confusion Matrix of Random Forest - Unbalanced Dataset

Type	Count
True Positive (TP)	200
True Negative (TN)	12812
False Positive (FP)	137
False Negative (FN)	14

Discussion on RF - Unbalanced Dataset Results

The confusion matrix shows the following;

- that 200 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12812 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 137 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 14 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.4: Confusion Matrix - RF - Unbalanced Dataset

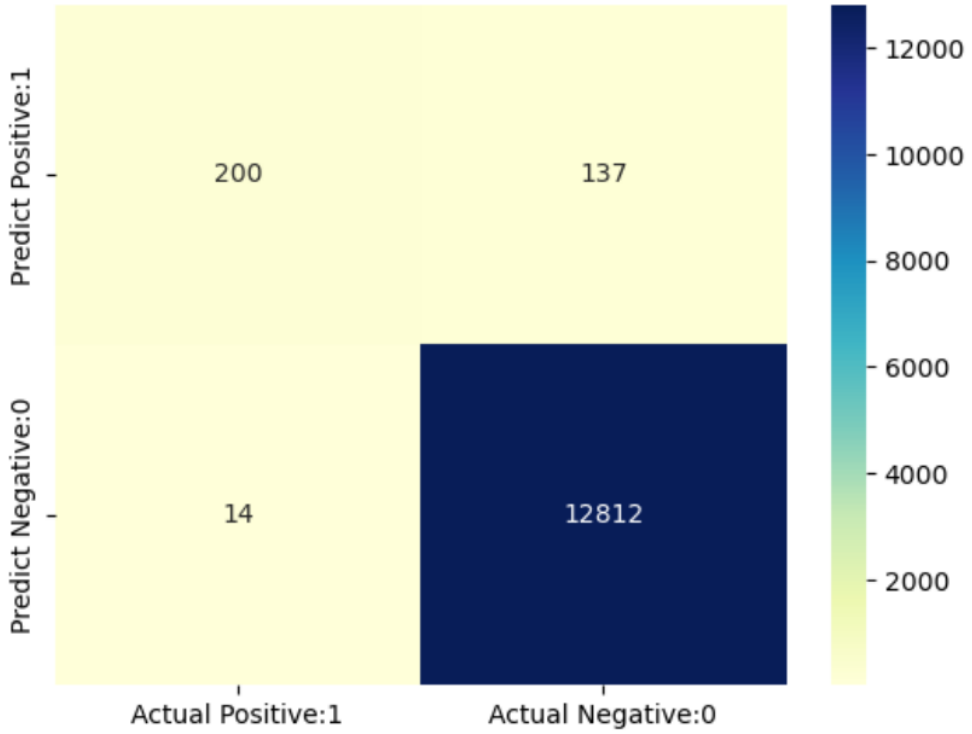


Figure 5.5: Accuracy - RF - Unbalanced Dataset

	precision	recall	f1-score	support
0	0.93	0.59	0.73	337
1	0.99	1.00	0.99	12826
accuracy			0.99	13163
macro avg	0.96	0.80	0.86	13163
weighted avg	0.99	0.99	0.99	13163

5.1.4 Experiment using KNN - Unbalanced Dateset

To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.5: Confusion Matrix of KNN - Unbalanced Dataset

Type	Count
True Positive (TP)	160
True Negative (TN)	12815
False Positive (FP)	177
False Negative (FN)	11

Discussion on KNN - Unbalanced Dataset Results

The confusion matrix shows the following;

- that 160 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12815 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 177 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 11 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.6: Confusion Matrix - KNN - Unbalanced Dataset

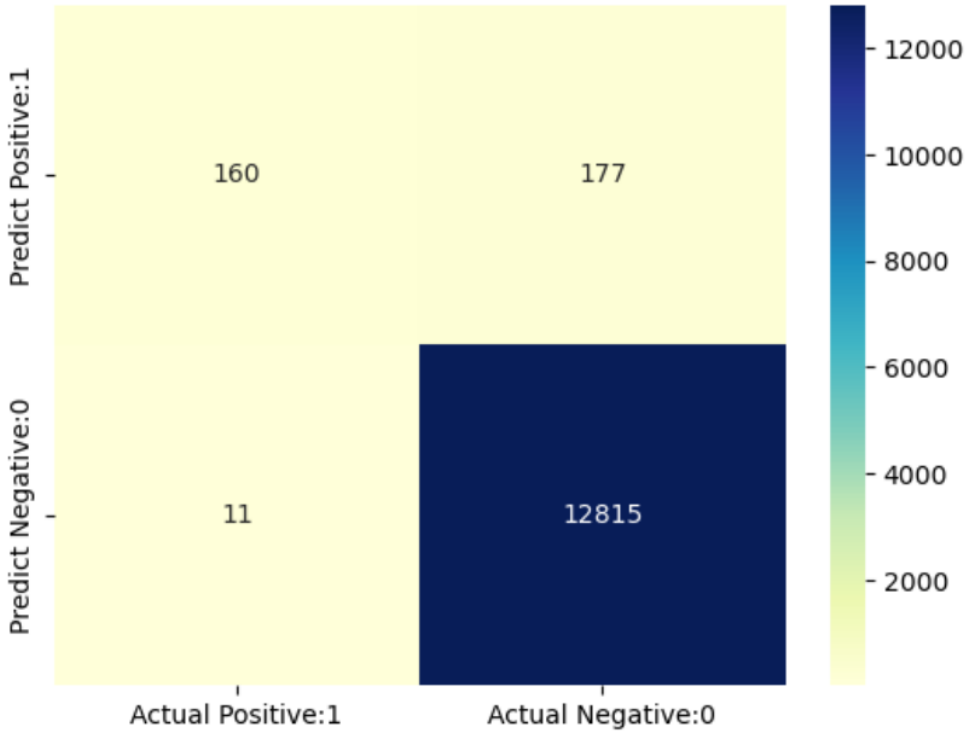


Figure 5.7: Accuracy - KNN - Unbalanced Dataset

	precision	recall	f1-score	support
0	0.94	0.47	0.63	337
1	0.99	1.00	0.99	12826
accuracy			0.99	13163
macro avg	0.96	0.74	0.81	13163
weighted avg	0.99	0.99	0.98	13163

5.1.5 Experiment using Decision Tree - Unbalanced Dateset

To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.6: Confusion Matrix of Decision Tree - Unbalanced Dataset

Type	Count
True Positive (TP)	128
True Negative (TN)	12821
False Positive (FP)	209
False Negative (FN)	5

Discussion on DT - Unbalanced Dataset Results

The confusion matrix shows the following;

- that 128 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12821 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 209 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 5 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.8: Confusion Matrix - DT - Unbalanced Dataset

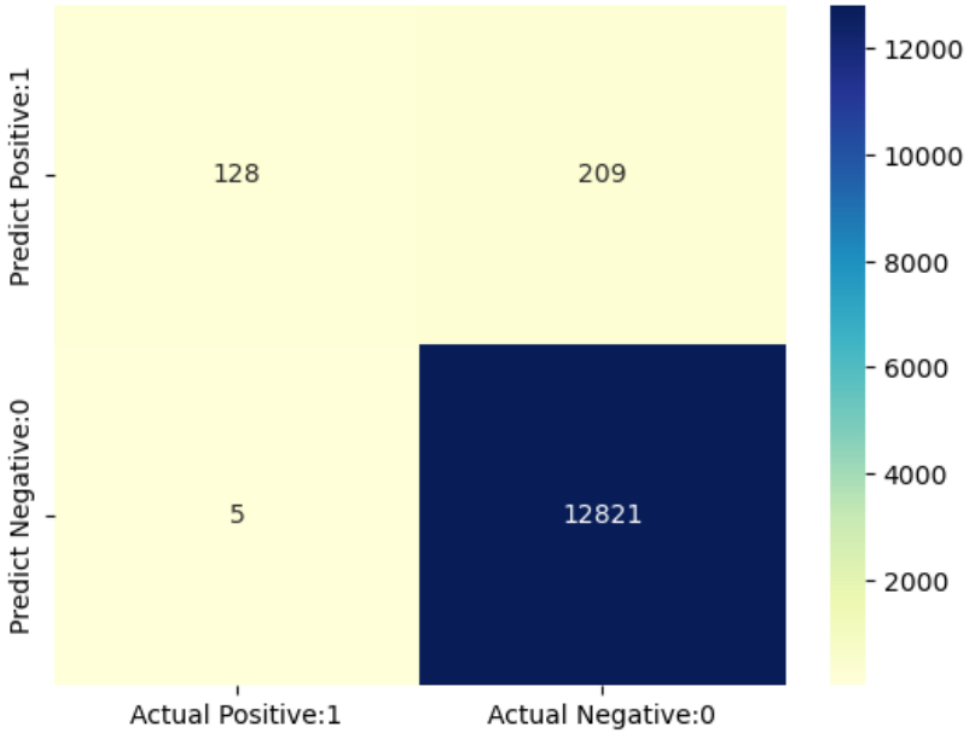


Figure 5.9: Accuracy - DT - Unbalanced Dataset

	precision	recall	f1-score	support
0	0.96	0.38	0.54	337
1	0.98	1.00	0.99	12826
accuracy			0.98	13163
macro avg	0.97	0.69	0.77	13163
weighted avg	0.98	0.98	0.98	13163

5.1.6 Experiment using Naive Bayes - Unbalanced Dateset

To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.7: Confusion Matrix of Naive Bayes - Unbalanced Dataset

Type	Count
True Positive (TP)	158
True Negative (TN)	11560
False Positive (FP)	179
False Negative (FN)	1266

Discussion on NB - Unbalanced Dataset Results

The confusion matrix shows the following;

- that 158 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 11560 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 179 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 1266 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.10: Confusion Matrix - NB - Unbalanced Dataset

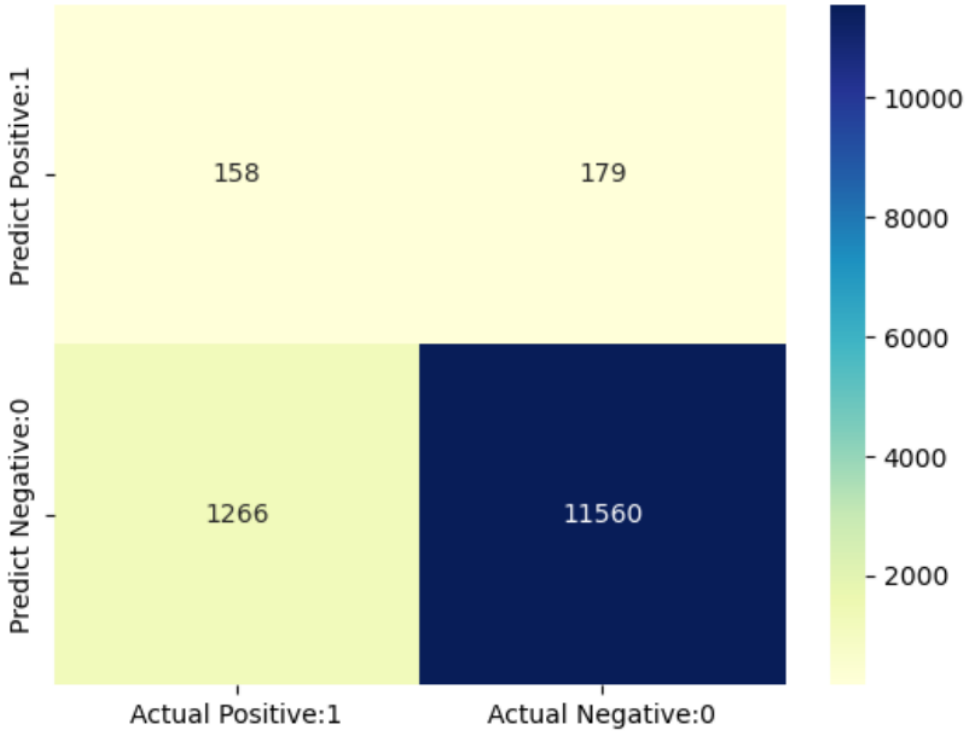
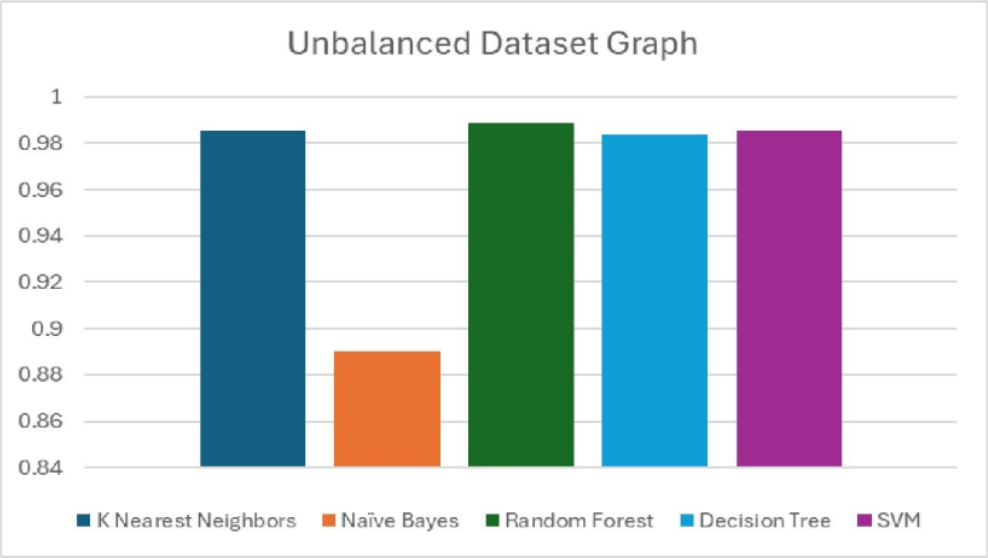


Figure 5.11: Accuracy - NB - Unbalanced Dataset

	precision	recall	f1-score	support
0	0.11	0.47	0.18	337
1	0.98	0.90	0.94	12826
accuracy			0.89	13163
macro avg	0.55	0.69	0.56	13163
weighted avg	0.96	0.89	0.92	13163

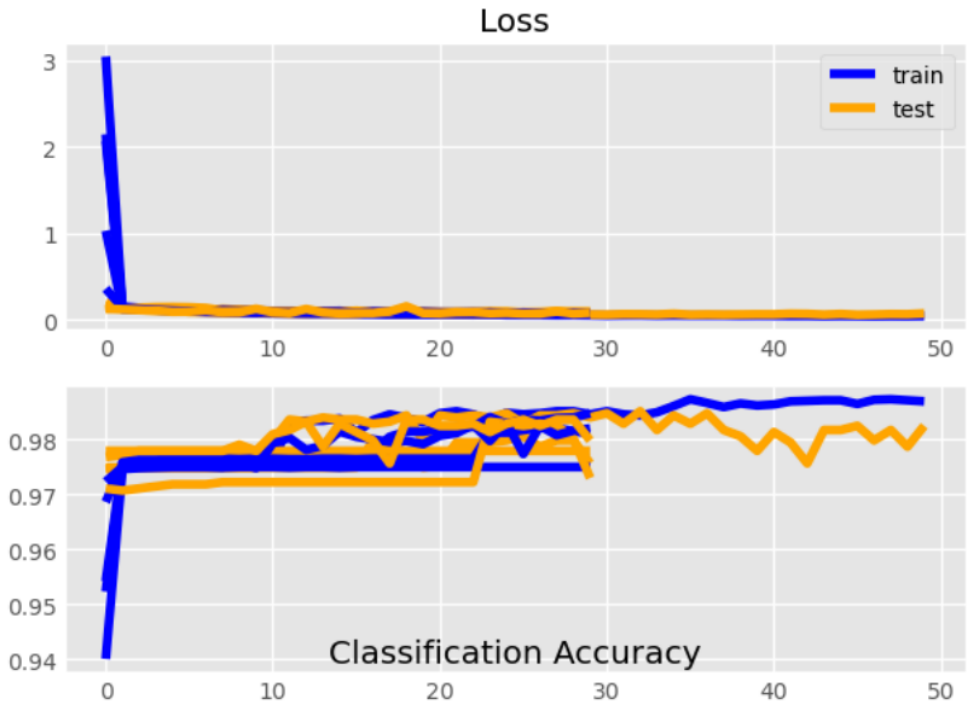
5.2 Accuracy Comparison of The Unbalanced Dataset Using Different Models

Figure 5.12: Accuracy Comparison - Unbalanced Dataset



5.3 Loss & Accuracy of The Unbalanced Dataset Using ANN Deep Learning Model

Figure 5.13: Loss & Accuracy Graph of ANN - Unbalanced Dataset



The above graph shows the loss and accuracy of the Artificial Neural Network (ANN) - Deep Learning Model with two hidden layers (Unbalanced Dataset).

The base paper did not perform experiments on balanced dataset, however, we did it.

5.3.1 Experiment using SVM - Balanced Dateset

For balanced dataset, upgrading technique was applied to unbalanced dataset, To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.8: Confusion Matrix of SVM - Balanced Dataset

Type	Count
True Positive (TP)	12661
True Negative (TN)	12887
False Positive (FP)	51
False Negative (FN)	80

Discussion on SVM - Balanced Dataset Results

The confusion matrix shows the following;

- that 12661 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12887 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 51 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 80 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.14: Confusion Matrix - SVM - Balanced Dataset

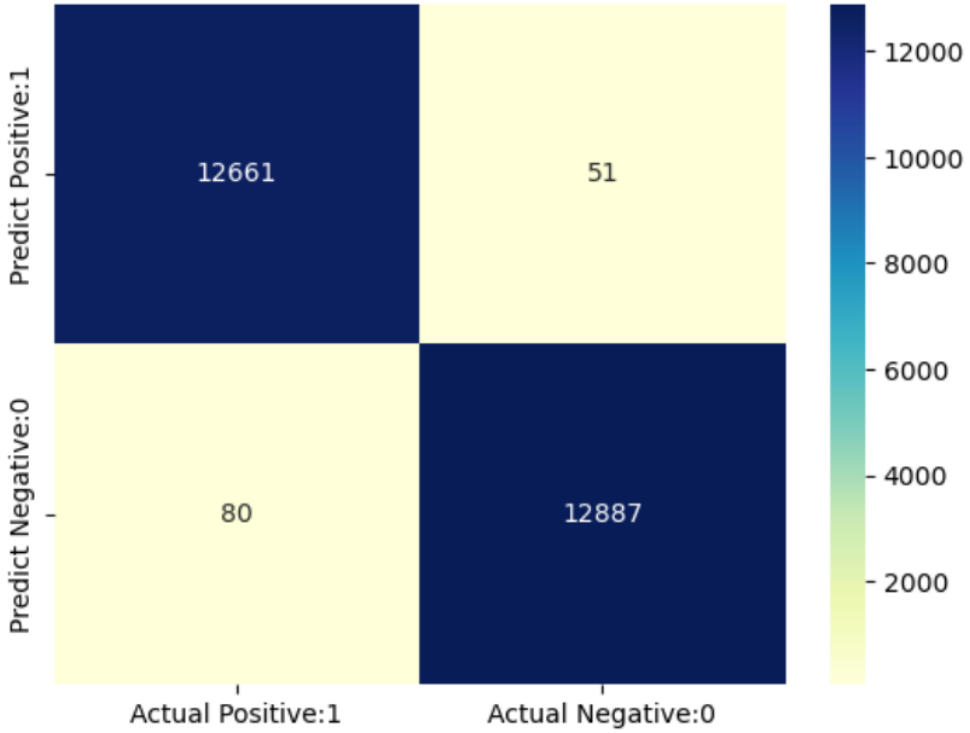


Figure 5.15: Accuracy - SVM - Balanced Dataset

	precision	recall	f1-score	support
0	0.99	1.00	0.99	12712
1	1.00	0.99	0.99	12967
accuracy			0.99	25679
macro avg	0.99	0.99	0.99	25679
weighted avg	0.99	0.99	0.99	25679

5.3.2 Experiment using Random Forest - Balanced Dateset

For balanced dataset, upgrading technique was applied to unbalanced dataset, To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.9: Confusion Matrix of RF - Balanced Dataset

Type	Count
True Positive (TP)	12661
True Negative (TN)	12909
False Positive (FP)	51
False Negative (FN)	58

Discussion on RF - Balanced Dataset Results

The confusion matrix shows the following;

- that 12661 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12909 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 51 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 58 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.16: Confusion Matrix - RF - Balanced Dataset

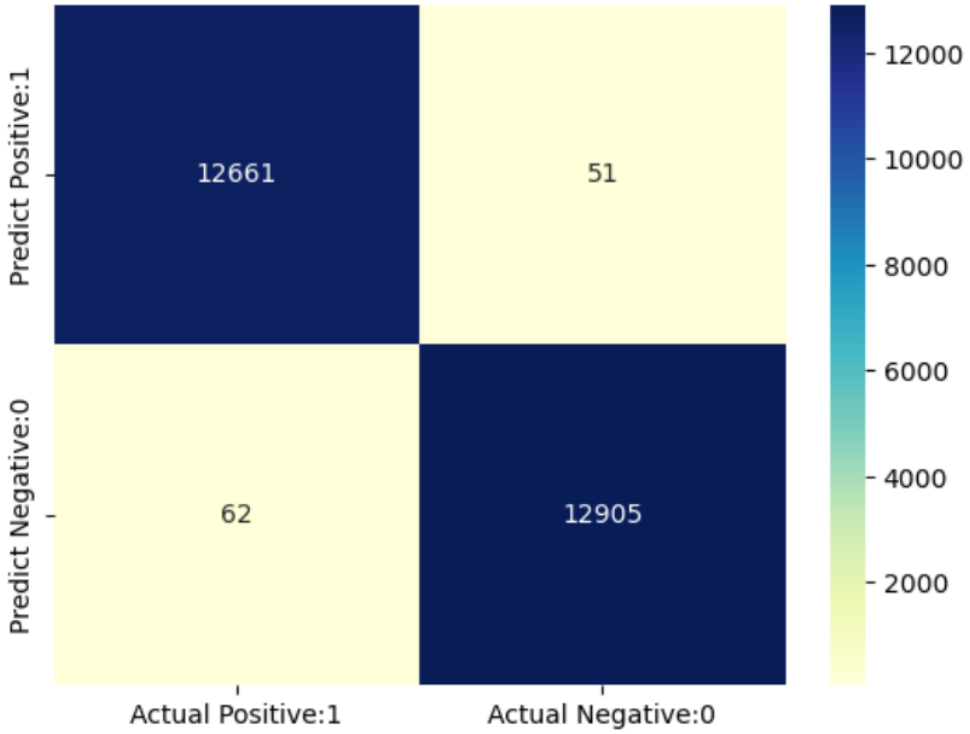


Figure 5.17: Accuracy - RF - Balanced Dataset

	precision	recall	f1-score	support
0	1.00	1.00	1.00	12712
1	1.00	1.00	1.00	12967
accuracy			1.00	25679
macro avg	1.00	1.00	1.00	25679
weighted avg	1.00	1.00	1.00	25679

5.3.3 Experiment using K-Nearest Neighbor - Balanced Dateset

For balanced dataset, upgrading technique was applied to unbalanced dataset, To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.10: Confusion Matrix KNN Balanced Dataset

Type	Count
True Positive (TP)	12661
True Negative (TN)	12836
False Positive (FP)	51
False Negative (FN)	131

Discussion on KNN - Balanced Dataset Results

The confusion matrix shows the following;

- that 12661 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12836 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 51 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 131 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.18: Confusion Matrix - KNN - Balanced Dataset

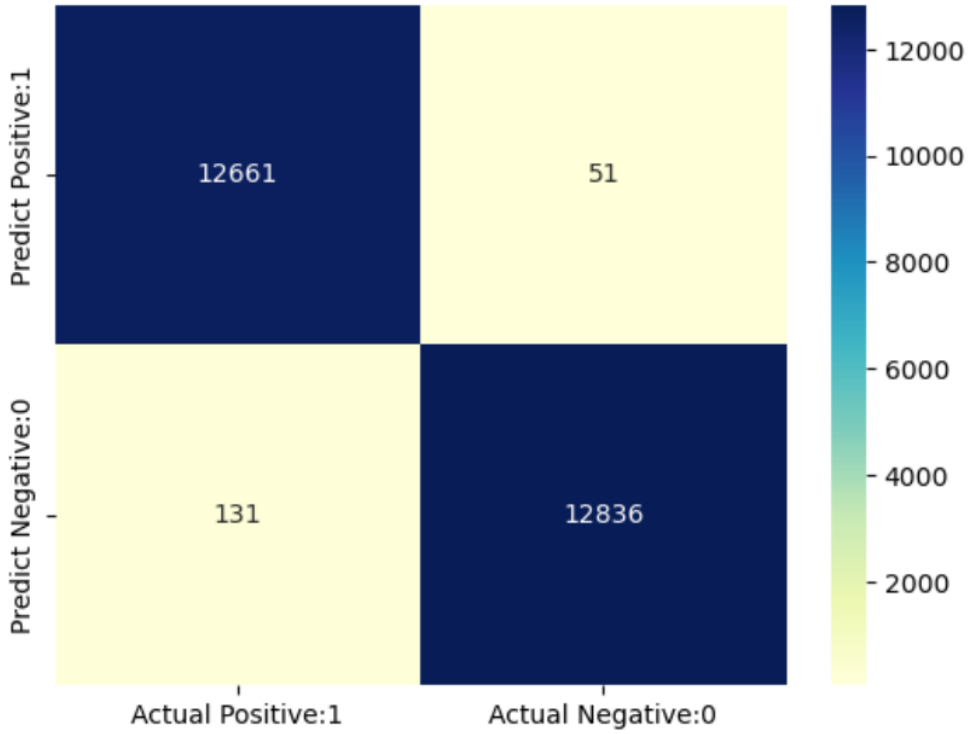


Figure 5.19: Accuracy - KNN - Balanced Dataset

	precision	recall	f1-score	support
0	0.99	1.00	0.99	12712
1	1.00	0.99	0.99	12967
accuracy			0.99	25679
macro avg	0.99	0.99	0.99	25679
weighted avg	0.99	0.99	0.99	25679

5.3.4 Experiment using Decision Tree - Balanced Dateset

For balanced dataset, upgrading technique was applied to unbalanced dataset, To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.11: Confusion Matrix of DT - Balanced Dataset

Type	Count
True Positive (TP)	6741
True Negative (TN)	12811
False Positive (FP)	5971
False Negative (FN)	156

Discussion on DT - Balanced Dataset Results

The confusion matrix shows the following;

- that 6741 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 12811 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 5971 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 156 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.20: Confusion Matrix - DT - Balanced Dataset

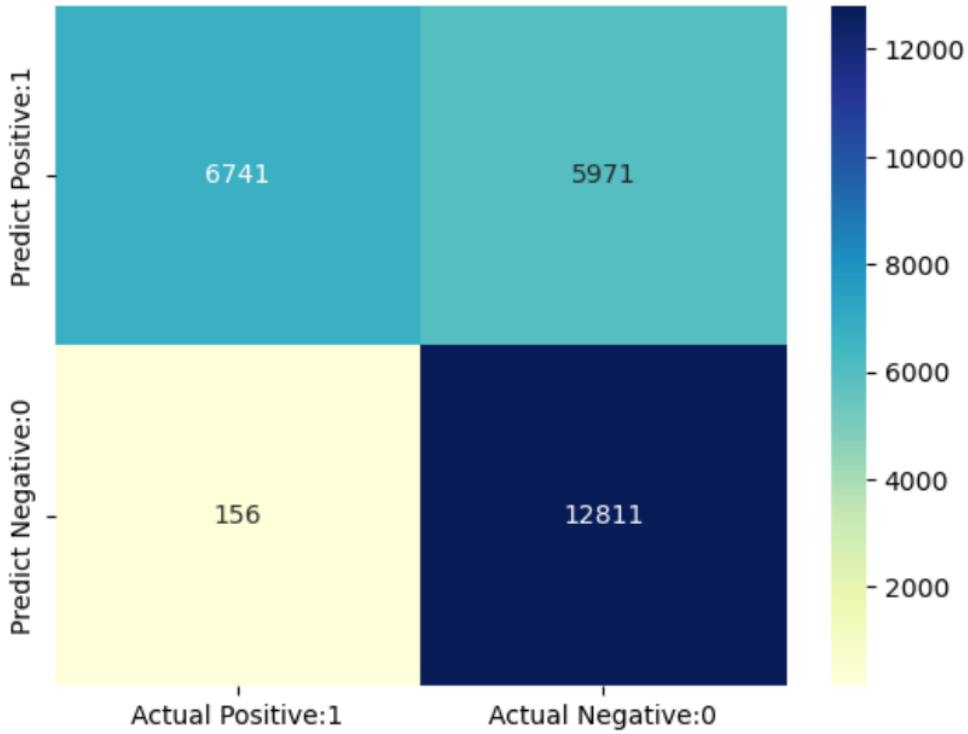


Figure 5.21: Accuracy - DT - Balanced Dataset

	precision	recall	f1-score	support
0	0.98	0.53	0.69	12712
1	0.68	0.99	0.81	12967
accuracy			0.76	25679
macro avg	0.83	0.76	0.75	25679
weighted avg	0.83	0.76	0.75	25679

5.3.5 Experiment using Naive Bayes - Balanced Dateset

For balanced dataset, upgrading technique was applied to unbalanced dataset, To find out the Accuracy, Precision, Recall and F1 Score, the dataset was trained and test with the ratio 70% : 30%. The following results were obtained;

Table 5.12: Confusion Matrix NB Balanced Dataset

Type	Count
True Positive (TP)	8492
True Negative (TN)	10344
False Positive (FP)	4220
False Negative (FN)	2623

Discussion on NB - Balanced Dataset Results

The confusion matrix shows the following;

- that 8492 times, when there is malware in the dataset and it is predicted as malware (i.e. True Positive).
- that 10344 times, when there is no malware in the dataset and it is predicted as not malware (i.e. True Negative).
- that 4220 times, when there is no malware sample in the dataset and it is predicted as a malware (i.e. False Positive).
- that 2623 times, when there is malware in the dataset and it is predicted as not malware (i.e. False Negative).

Figure 5.22: Confusion Matrix - NB - Balanced Dataset

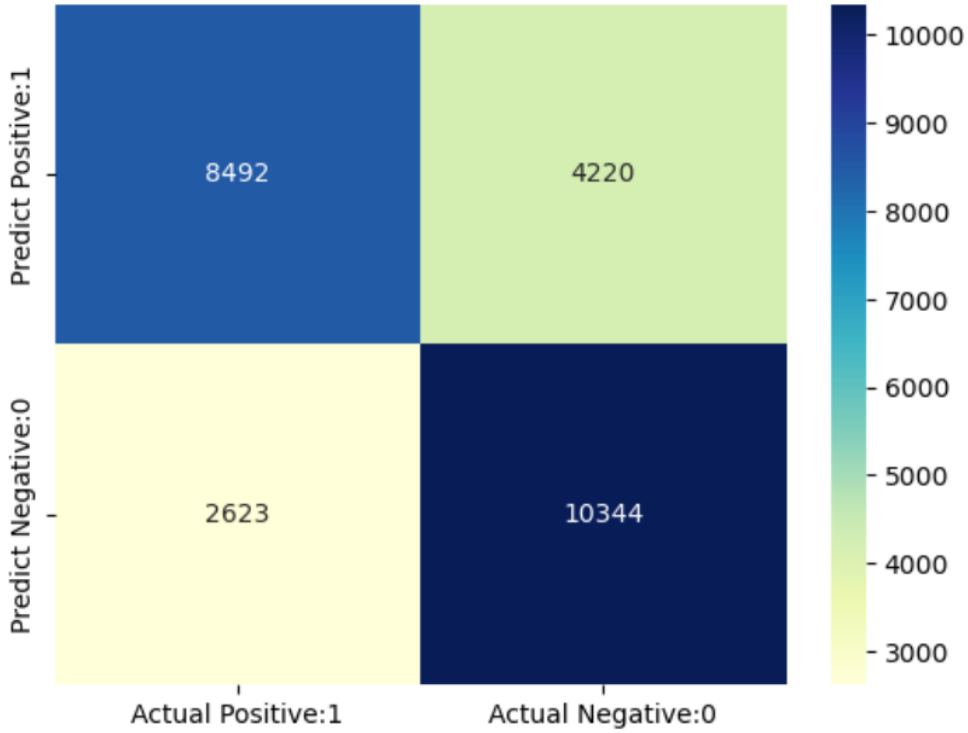
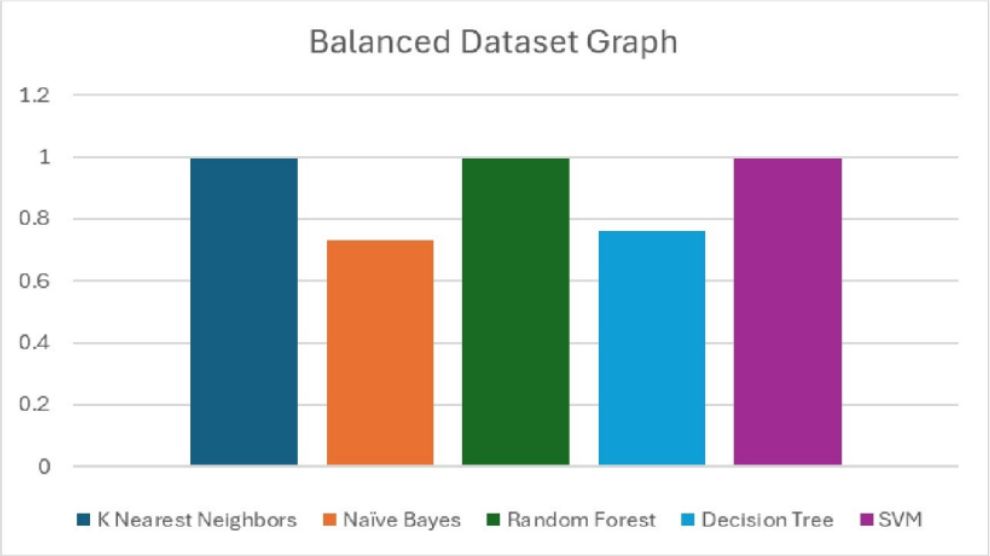


Figure 5.23: Accuracy - NB - Balanced Dataset

	precision	recall	f1-score	support
0	0.76	0.67	0.71	12712
1	0.71	0.80	0.75	12967
accuracy			0.73	25679
macro avg	0.74	0.73	0.73	25679
weighted avg	0.74	0.73	0.73	25679

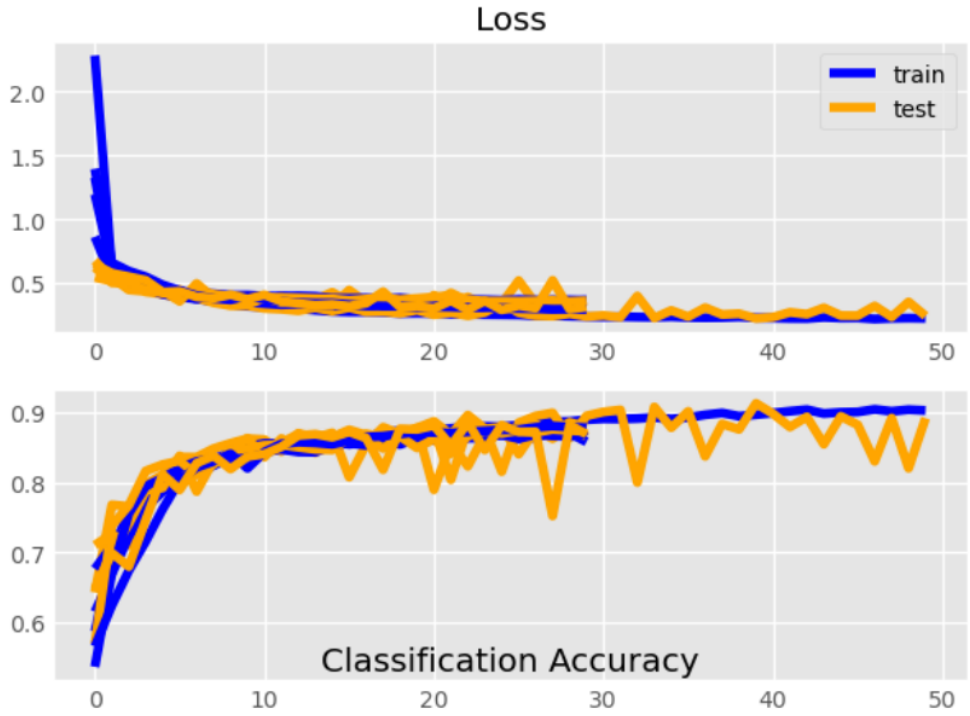
5.4 Accuracy Comparison of The Balanced Dataset Using Different Models

Figure 5.24: Accuracy Comparison - Balanced Dataset



5.5 Loss & Accuracy of The balanced Dataset Using ANN Deep Learning Model

Figure 5.25: Loss & Accuracy Graph of ANN - Balanced Dataset



The above graph shows the loss and accuracy of the Artificial Neural Network (ANN) - Deep Learning Model with two hidden layers (Balanced Dataset).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

We have worked on Binary classification in the context of malware detection involving categorizing files or data into two distinct classes: ”malicious” and ”benign.” This process is critical in cybersecurity for identifying and mitigating threats posed by malware. We have used five different Machine Learning Models (SVM, KNN, RF, NB, DT). Here's a structured approach to performing binary classification for malware detection:

- Data Collection
- Feature Extraction
- Data Preprocessing
- Model Selection
- Model Training & Model Evaluation

We have also discussed the Artificial Neural Network with two hidden layers for deep analysis of unique API call sequences and produced the loss and accuracy results.

6.2 Future Work

Following the success of the binary classification of malwares, our next proposed future work will be on Multiclass malware detection. Multiclass Malware Detection is a dynamic and ongoing challenge requiring robust data collection, sophisticated feature extraction, and powerful machine learning models.

MS Thesis

ORIGINALITY REPORT

8%	4%	5%	4%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	link.springer.com Internet Source	1%
2	Akusok, Anton. "Extreme learning machines: Novel extensions and application to big data.", The University of Iowa Publication	1%
3	midra.uni-miskolc.hu Internet Source	1%
4	"Proceedings of the 2nd International Conference on Big Data, IoT and Machine Learning", Springer Science and Business Media LLC, 2024 Publication	1%
5	Submitted to Purdue University Student Paper	<1%
6	www.mdpi.com Internet Source	<1%
7	Submitted to Coventry University Student Paper	<1%

8	Submitted to Higher Education Commission Pakistan Student Paper	<1%
9	Submitted to Hult International Business School, Inc. Student Paper	<1%
10	repository.ju.edu.et Internet Source	<1%
11	academic-accelerator.com Internet Source	<1%
12	Eslam Amer, Ivan Zelinka. "A dynamic Windows malware detection and prediction method based on contextual understanding of API call sequence", Computers & Security, 2020 Publication	<1%
13	Lei Cui, Jiancong Cui, Yuede Ji, Zhiyu Hao, Lun Li, Zhenquan Ding. "API2Vec: Learning Representations of API Sequences for Malware Detection", Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, 2023 Publication	<1%
14	dspace-jcboseust.refread.com Internet Source	<1%
15	www.researchgate.net Internet Source	<1%

16	Submitted to Florida Institute of Technology Student Paper	<1 %
17	Submitted to Ibri College of Technology Student Paper	<1 %
18	Submitted to Colorado Technical University Student Paper	<1 %
19	Submitted to The Robert Gordon University Student Paper	<1 %
20	Submitted to University of East London Student Paper	<1 %
21	Submitted to University of Tennessee Chattanooga Student Paper	<1 %
22	Submitted to Wilmington University Student Paper	<1 %
23	researchrepository.wvu.edu Internet Source	<1 %
24	www.ibm.com Internet Source	<1 %
25	Submitted to University of Abertay Dundee Student Paper	<1 %
26	Submitted to University of Zululand Student Paper	<1 %

27	Jianchao Lu, Xi Zheng, Lihong Tang, Tianyi Zhang, Quan Z. Sheng, Chen Wang, Jiong Jin, Shui Yu, Wanlei Zhou. "Can Steering Wheel Detect Your Driving Fatigue?", IEEE Transactions on Vehicular Technology, 2021 Publication	<1 %
28	mdpi-res.com Internet Source	<1 %
29	Serena McDonnell, Omar Nada, Muhammad Rizwan Abid, Ehsan Amjadian. "CyberBERT: A Deep Dynamic-State Session-Based Recommender System for Cyber Threat Recognition", 2021 IEEE Aerospace Conference (50100), 2021 Publication	<1 %
30	Stavros Vakalis, Jeffrey A. Nanzer. "Microwave Imaging Using Noise Signals", IEEE Transactions on Microwave Theory and Techniques, 2018 Publication	<1 %
31	core.ac.uk Internet Source	<1 %
32	ntnuopen.ntnu.no Internet Source	<1 %
33	www.fastercapital.com Internet Source	<1 %

34	Valente, Maria Elisa Maciel. "Network Anomaly Detection using Adversarial Deep Learning", Universidade do Minho (Portugal), 2024 Publication	<1 %
35	www.ripublication.com Internet Source	<1 %
36	www.sciencegate.app Internet Source	<1 %
37	www.wrike.com Internet Source	<1 %
38	Ferhat Ozgur Catak, Ahmet Faruk Yazı, Ogerta Elezaj, Javed Ahmed. "Deep learning based Sequential model for malware analysis using Windows exe API Calls", PeerJ Computer Science, 2020 Publication	<1 %

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		