SPY-X AN AI POWERED CYBERSECURITY SYSTEM

Mini Project Report

Submitted for the Partial Fulfillment of the Requirements

for fire *Award of the Degree of*

Master of Computer Applications

By

NIRMAL R TITUS

MUT24MCA-2042

Under the guidance of

DR GEETHU S

ASSISTANT PROFFESSOR

Department of Computer Applications

MUTHOOT INSTITUTE OF TECHNOLOGY & SCIENCE

VARIKOLI P O, PUTHENCRUZ, ERNAKULAM DISTRICT, KERALA

(Affiliated to A P J Abdul Kalam Technological University, Thiruvananthapuram, Kerala)

October — 2025

Muthoot
Institute of Technology & Science

Department of Computer Applications

BONAFIDE CERTIFICATE

*This is to certify that the Mini Project Report entitled "**SPY-X**" has been submitted by Mr. **Nirmal R Titus** Reg.No. **MUT24MCA-2042** for the partial fulfillment of the requirements for the award of the degree of Master of Computer Applications (MCA) of A P J Abdul Kalam Technological University, Kerala during the year 2025.*

Place: Varikoli
Date:

Project Guide	**Project Coordinators**	**HoD**
Dr Geethu S	Dr Sujithra Sankar	Dr Saritha K
	Mr Sivadas T Nair	

Submitted for the Final Evaluation held on..................

Name and Signature of the Examiner

DECLARATION

*I, undersigned hereby declare that the Mini Project Report **"SPY-X"**, submitted for the partial fulfilment of the requirements for the award of degree of Master of Computer Applications of the APJ Abdul Kalam Technological University, Kerala is a Bonafide work done by me under the supervision of Dr Geethu S. This submission represents my ideas in my own words and where ideas or words of others also have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.*

Nirmal R Titus

MUT24MCA-2042

Place:

Date:

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to all those who supported and guided me throughout the development of my project titled Spy-X – An AI-Powered Cybersecurity System. First, I extend my heartfelt thanks to our Principal, Dr Neelakantan P C, for providing a conductive learning environment. My appreciation also goes to my Head of Department, Dr Saritha K, whose guidance and encouragement have been invaluable.

I am particularly thankful to my project coordinators for this Mini Project, Dr Sujithra Sankar and Sivadas T Nair, for their support in managing the project, and especially to my Project Guide, Dr Geethu S, for mentorship and insightful feedback. Their experience and feedback have influenced my work.

I would also like to acknowledge my department and fellow classmates for their camaraderie, which has made this journey enjoyable. Lastly, my deepest appreciation goes to my family for their unwavering love and support. This project would not have been possible without the collective encouragement from all these individuals. Thank you for believing in me and for being a part of my academic journey.

ABSTRACT

In today's digital age, cyber threats are rapidly evolving, making traditional security mechanisms insufficient to counter modern attacks. This project, Spy-X – An AI-Powered Cybersecurity System, is designed to enhance digital security through the integration of machine learning and network analysis. Spy-X provides two key modules: Malware Detection and Network Port Scanning, offering users a comprehensive security evaluation tool in a single web-based interface.

The malware detection module utilizes a XGBoost classifier trained on Portable Executable (PE) header features to accurately classify files as Malicious or Safe. The port scanning module, implemented using socket programming and Scapy, analyzes open ports and connected devices to identify potential vulnerabilities in the network environment. The entire system is built using Flask, enabling real-time interaction between the backend intelligence and a lightweight web interface for seamless user experience.

Experimental results demonstrate that the proposed system achieves high accuracy in malware classification and effectively identifies open or risky network ports. By combining AI-driven malware analysis with real-time network assessment, SpyX contributes to proactive and intelligent cybersecurity. Future developments may include expanding the malware dataset, implementing intrusion detection capabilities, and deploying the system as a cloud-based security service for broader accessibility.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The modern digital landscape is increasingly dominated by complex and evolving cyber threats, where traditional reactive security systems often struggle to protect users from emerging forms of malware and network intrusions. As attackers adopt advanced techniques such as obfuscation, polymorphism, and zero-day exploitation, conventional antivirus tools that rely solely on signature databases have become less effective. SpyX was developed to address this growing security gap by providing an intelligent, dual-layered cybersecurity platform that operates proactively rather than reactively. It focuses on both file integrity and network safety, ensuring that users receive continuous, real-time protection without requiring deep technical expertise or enterprise-scale infrastructure.

At the core of SpyX lies its intelligent malware detection module powered by machine learning. Instead of executing potentially harmful files, the system performs static analysis on Portable Executable (PE) headers to extract meaningful structural features—such as imported libraries, section characteristics, and entropy patterns—that reveal malicious intent. This method enables accurate identification of both known and previously unseen threats while maintaining efficiency and safety. Complementing this is the integrated network security module, which continuously scans open ports, identifies vulnerable configurations, and alerts users to potential risks before exploitation can occur. By unifying predictive file-based analysis with proactive network monitoring, SpyX delivers a lightweight yet powerful defence mechanism that enhances overall cybersecurity readiness and establishes a modern framework for intelligent, explainable, and real-time protection.

1.2 MOTIVATION

The motivation behind developing SpyX arises from the growing inadequacy of traditional cybersecurity systems in dealing with today's rapidly evolving threat landscape. Conventional antivirus tools primarily depend on signature-based detection, which uses known malware patterns or hash values to identify threats. Although this method is effective against previously documented attacks, it fails to recognize new, mutated, or zero-day malware that employ advanced evasion and obfuscation techniques to bypass standard defences. Moreover, traditional systems tend to focus only on malware detection, ignoring the equally significant issue of network vulnerabilities such as exposed open ports that can act as gateways for attackers. This reactive and fragmented approach to security has created an urgent demand for an integrated and proactive system capable of safeguarding users at both the file and network levels.

To address these limitations, SpyX was designed as a dual-layered security framework that merges intelligent static malware detection with real-time network vulnerability analysis. Instead of executing files in risky environments, SpyX performs static analysis of Portable Executable (PE) headers, using machine learning models trained on structural file attributes to predict whether a file is malicious or safe. This method ensures faster detection while minimizing system risks. Alongside this, the project incorporates a dynamic port scanning and network auditing module that identifies open or insecure ports, helping users address vulnerabilities before they are exploited. Thus, the motivation for developing SpyX lies in establishing a comprehensive, predictive, and efficient cybersecurity solution that provides both malware detection and network protection under a single unified platform

1.3 PROBLEM STATEMENT

The primary challenge in modern cybersecurity is the inherent difficulty of providing comprehensive, real-time protection against diverse and evolving threat vectors. Current solutions suffer from two critical failures. First, the prevalent reliance on signature-based anti-malware tools makes them ineffective against sophisticated, polymorphic, and **zero-day** malware that constantly mutate to evade known definitions; this results in a dangerous detection gap for new threats and compromises the integrity of file systems. Second, existing security tools often fail to provide adequate, continuous visibility into the immediate network perimeter, leaving the user's infrastructure vulnerable to unauthorized access. Specifically, there is a lack of a unified, user-friendly system that can proactively and automatically identify and alert users to dangerous open ports which are frequently scanned and exploited by attackers, thus leaving a crucial entry point unattended.
Project SpyX is designed to solve this dual problem by creating a cohesive, intelligent platform that overcomes the limitations of segregated security methods. We aim to eliminate the detection gap in file security by developing a reliable, machine-learning based static analysis method that evaluates the PE (Portable Executable) header to determine a file's malicious potential based on its internal structure, regardless of its known signature. Concurrently, the system will address the critical network visibility deficit by integrating a dynamic port scanning module. The core

objective of this project is to create a seamless security loop that not only accurately detects previously unseen file-based threats but also provides immediate, actionable warnings about network vulnerabilities, effectively fortifying both the endpoint and the network to ensure a truly secure digital environment.The project was developed using the Scrum framework, an Agile methodology that is particularly well-suited for projects requiring iterative development and constant feedback. Scrum provides a structured yet flexible approach to managing complex software development projects, allowing teams to deliver high-quality results through continuous collaboration and refinement.

1.4 OBJECTIVES

The primary objective of Project SpyX is to develop an intelligent cybersecurity system that provides dual-layered protection by combining malware detection with open-port discovery. The system is designed to enhance a user's security awareness by analysing uploaded files for potential threats and identifying open ports that could pose security risks.
A key technical goal is to implement a robust XGBoost classifier for static file analysis. The model extracts features from the PE (Portable Executable) headers of uploaded files to classify them as malicious or benign based solely on internal structure and metadata, without relying on prior signatures. This predictive mechanism is integrated into a simple and user-friendly web interface that presents results clearly and enables quick understanding of potential risks.
Another objective is to provide a **basic port discovery module** that enumerates open ports on a specified target IP. While it does not perform intrusive network scans or dynamic monitoring, the module helps users identify exposed ports and associated services, offering actionable insights for improving local network security.
By achieving these objectives, SpyX demonstrates how predictive malware analysis and targeted open-port discovery can be combined in a unified platform to strengthen personal and small-scale organizational cybersecurity

1.5 SCRUM METHODOLOGY

The project was developed using the Scrum framework, an Agile methodology that is particularly well-suited for projects requiring iterative development and constant feedback. Scrum provides a structured yet flexible approach to managing complex software development projects, allowing teams to deliver high-quality results through continuous collaboration and refinement

1.5.1 Overview of Scrum Framework

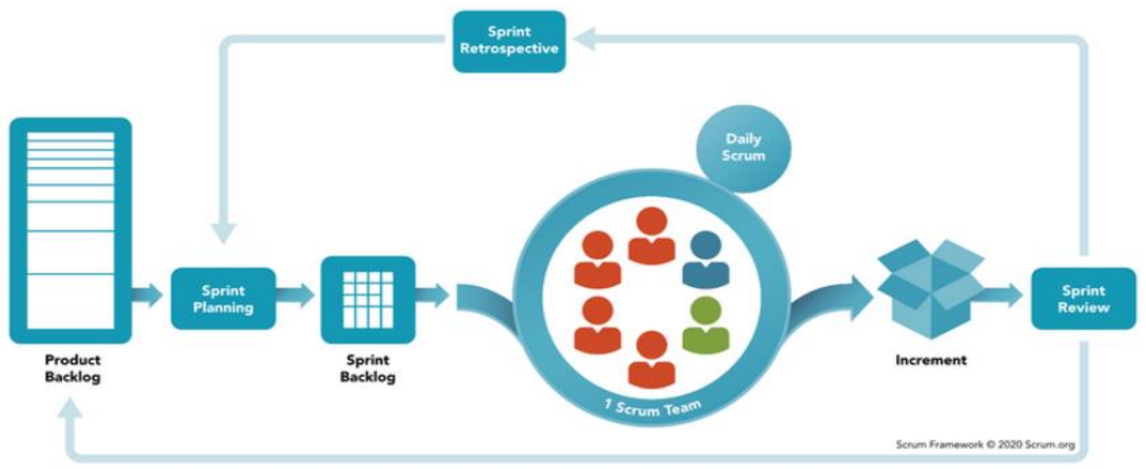


Fig 1.1 Scrum Framework

The components of Scrum are:

Sprints: A sprint is a time-boxed period (typically 2-4 weeks) during which specific tasks and deliverables are completed. Each sprint ends with a potentially shippable product increment, enabling regular feedback and course correction.

Scrum Roles: It includes the Product Owner, who prioritizes the backlog; the Scrum Master, who facilitates the process; and the Development Team, who delivers the product increment

- **Product Owner**: The product owner represents the customer and ensures that the team is working on the most valuable features. They are responsible for prioritizing the product backlog, which is a list of features or tasks to be completed.
- **Scrum Master**: The Scrum Master facilitates the Scrum process, ensuring that the team follows Scrum practices, removes obstacles, and promotes collaboration and continuous improvement.
- **Development Team:** The team is responsible for building the product and delivering increments during each sprint. They are self-organizing and cross-functional, working together to complete tasks.

Scrum Events: It consists of Sprint Planning, Daily Standups, Sprint Reviews, and Sprint Retrospectives, which ensure regular progress, feedback, and continuous improvement.

- **Sprint Planning:** At the start of each sprint, the team collaborates to decide which tasks from the backlog will be completed. They discuss priorities, estimate effort, and set clear sprint goals. This meeting ensures that everyone understands their responsibilities and aligns efforts toward delivering the most valuable outcomes.

- **Daily Standup:** A short daily meeting where each member shares what they did yesterday, what they'll do today, and any issues faced. It keeps the whole team informed and helps identify problems early. This practice improves coordination and keeps the sprint progress transparent.
- **Sprint Review:** At the end of each sprint, the team presents the completed work to the product owner and stakeholders. Feedback is gathered to guide future spri At the end of the sprint, the team presents completed work to stakeholders and the product owner. Feedback is gathered to validate the outcome and identify needed improvements. It helps ensure the project is moving in the right direction and meeting user expectations.
- **Sprint Retrospective:** After the sprint review, the team reflects on the sprint process—what went well, what didn't, and what can be improved. This open discussion helps refine workflows and enhance collaboration. It ensures that each new sprint becomes more efficient and productive than the last.

Scrum Artifacts: Scrum artifacts are key tools in the Scrum framework that help teams manage work, track progress, and deliver value efficiently. They provide transparency and a clear understanding of what needs to be done, what is being worked on, and what has been completed. The three main artifacts in Scrum are Product Backlog, Sprint Backlog, and Increment.

- **Product Backlog** A dynamic, prioritized list of tasks, features, bug fixes, or user stories. Managed by the Product Owner, it ensures that the most valuable work is addressed first. The backlog evolves over time as new requirements emerge or priorities change.
- **Sprint Backlog**: A subset of the Product Backlog selected for completion in the current sprint. It includes specific tasks the team plans to finish and is used to track progress during the sprint. The development team owns the Sprint Backlog and updates it regularly.
- **Increment:** The working product delivered at the end of a sprint. Each increment is potentially shippable, tested, and adds value to the product. Increments build on previous ones, showing continuous progress toward the final product.

1.5.2 Implementation of Scrum in the Project

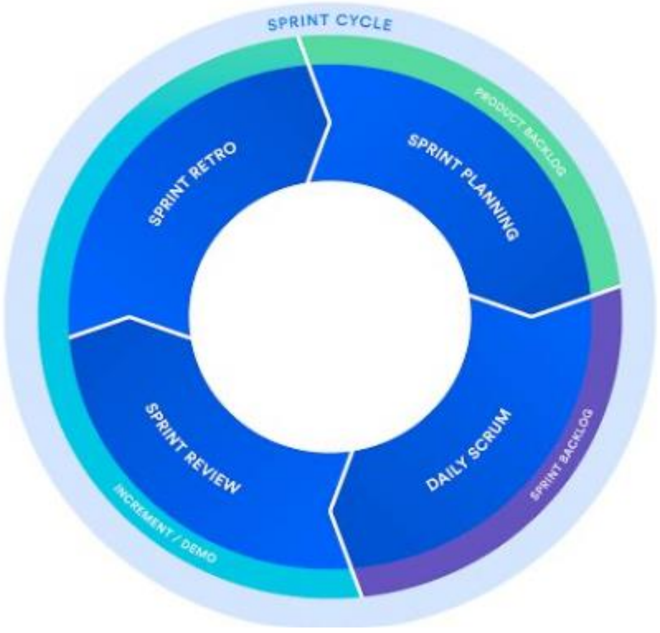


Fig 1.2 Sprint Cycle

In this project, the Scrum framework was adapted for individual development to ensure organized progress, efficient time management, and continuous improvement throughout the system's creation. Although the project was executed by a single developer, the principles of Scrum were applied to maintain structure, discipline, and iterative refinement of the Silent Seizure Detection System.

1. **Sprint Planning:** The project was divided into multiple short sprints, each focusing on specific goals such as data preprocessing, feature extraction, model training, front-end development, and API integration. At the beginning of each sprint, clear objectives were defined, and a task backlog was created with priorities based on project dependencies and deadlines. This ensured a focused workflow and achievable milestones for every iteration.
2. **Daily Progress Tracking:** Instead of team stand-up meetings, daily progress was tracked through self-review notes and a personal Kanban board. This allowed consistent evaluation of completed tasks, identification of blockers, and planning for the next day's activities. Maintaining this routine improved accountability and consistency in the development process.
3. **Backlog Refinement:** The project backlog was regularly revisited to adjust priorities based on testing feedback and technical challenges encountered. New tasks such as fine-tuning EEG feature extraction methods or modifying model parameters were added dynamically as the project evolved. This iterative backlog management enabled flexibility and responsiveness to new insights during development.

4. **Sprint Review:** At the end of each sprint, the implemented components (for example, model accuracy improvements, API responses, and user interface updates) were reviewed and tested for functionality. This self-review process ensured that each sprint delivered a tangible and tested increment of the overall system, ready for integration with the next module.
5. **Sprint Retrospective:** Following each sprint, a short retrospective was conducted to reflect on the development process identifying what worked efficiently and what required adjustment. Lessons learned, such as optimizing preprocessing pipelines or improving user experience in the web interface, were documented and applied in subsequent sprints to enhance performance and quality.
6. **Continuous Feedback Loop:** Continuous feedback was obtained through iterative testing and evaluation of the model's predictions and visualization performance. Adjustments were made based on model outputs, error analysis, and usability observations. This feedback-driven development cycle led to a more accurate and user-friendly system.

1.5.3 Burndown Chart

A burndown chart is a key visual tool in Agile and Scrum project management, used to monitor and track the progress of work during a sprint. It shows how much work remains to be completed over time, helping teams and stakeholders quickly understand whether the sprint is on schedule. By providing a clear, real-time view of progress, burndown charts help identify potential bottlenecks, delays, or workload imbalances early, allowing corrective actions to be taken before issues escalate.

Typically, a burndown chart is represented as a line graph with two main axes:

- **X-Axis:** Represents the time period of the sprint, usually in days.
- **Y-Axis:** Represents the remaining work, which can be measured in story points, tasks, or hours.

At the start of the sprint, the chart shows the total work to be done. As the team completes tasks, the line trends downward toward zero, ideally reaching it by the end of the sprint. A smooth downward slope indicates consistent progress, while flat or uneven sections may indicate delays or obstacles. Burndown charts are therefore not only a tracking tool but also a communication tool, keeping everyone aligned on the sprint's status and facilitating timely decision-making

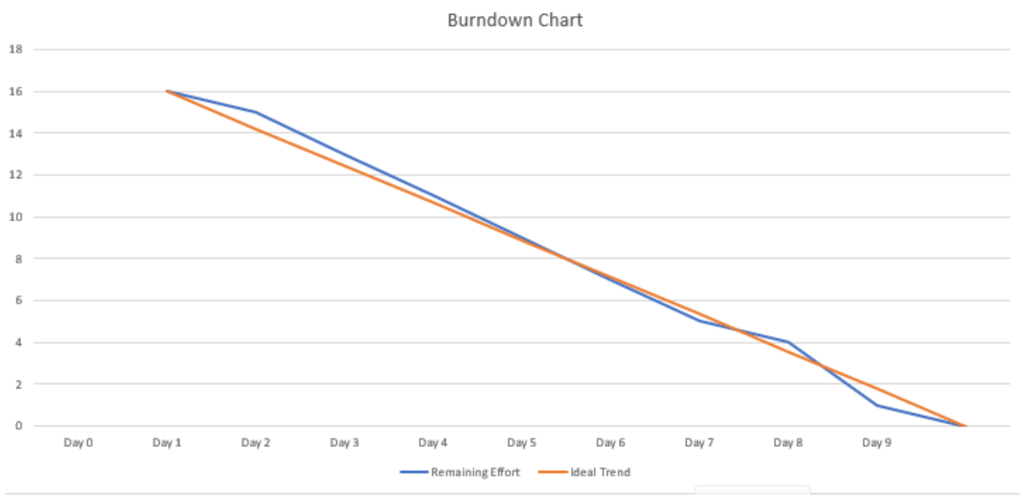


Fig 1.3 Sample Burndown Chart

In a standard burndown chart:

- The ideal burndown line represents the steady progress needed to complete all tasks by the end of the sprint. This line assumes that tasks are completed at a constant pace throughout the sprint.
- The actual burndown line tracks the real progress of the team, showing the actual amount of remaining work each day. When the actual line diverges from the ideal line, it can indicate issues or delays that need attention.

In this project, a Burndown Chart was employed throughout the development cycle to track progress, measure productivity, and ensure that each sprint's objectives were achievable within the planned timeframe. Although the Silent Seizure Detection System was developed individually, maintaining a burndown chart provided valuable insights into task completion trends, helping to manage time efficiently and maintain focus across each development phase.

1. **Tracking Daily Progress:** The burndown chart was updated regularly to log completed tasks and reflect the remaining workload for each sprint. This visualization helped monitor whether development activities—such as EEG preprocessing, feature extraction, model training, and integration of the Twilio alert system—were proceeding according to the schedule. Tracking progress daily ensured that each milestone was achieved without last-minute rushes.
2. **Identifying Bottlenecks:** Any deviation of the actual progress line from the ideal burndown line indicated possible challenges, such as issues in EEG data normalization, feature balancing, or integration errors during API testing. These visual cues helped in promptly identifying and resolving bottlenecks, ensuring that potential delays were addressed before they affected sprint completion.
3. **Promoting Accountability and Focus**: The chart served as a personal accountability tool, helping to maintain consistency and self-discipline during development. By visualizing progress, it was easier to stay motivated, avoid task pile-ups, and ensure that complex processes like seizure

prediction accuracy validation and real-time visualization were handled efficiently within their designated sprints.

4. **Facilitating Retrospective Analysis:** At the end of each sprint, the burndown chart was reviewed to analyze performance trends and identify improvement areas. This reflection provided insights into task estimation accuracy, time management, and prioritization effectiveness. For example, if the burndown rate slowed near the end of a sprint, it indicated the need to simplify sprint goals or re-estimate complex components in future cycles.

By incorporating a burndown chart, the project maintained a clear and transparent view of progress across all development stages from initial data preprocessing to the final deployment of the Silent Seizure Detection System. This approach ensured continuous improvement, better planning, and efficient sprint execution throughout the project lifecycle

.

CHAPTER 2
LITERATURE REVIEW

In this chapter, we explore existing research and methodologies related to malware detection using machine learning, with a focus on static analysis approaches. The aim is to provide a foundation for the development of SpyX, a lightweight security system that combines **XGBoost-based** malware detection and open-port discovery. By synthesizing insights from prior studies, this chapter justifies the choice of an **XGBoost** classifier, which provides a powerful balance between **high predictive accuracy** and **computational efficiency** for effective local security assessment.

2.1 “Machine Learning for Malware Detection: Techniques, Models, and Industry Impact” [1] (Emily Smith, 2025 — Medium)

Smith presents an overview of machine learning approaches in malware detection, highlighting the limitations of traditional signature-based techniques. The study emphasizes that **static analysis**, which inspects file structures rather than executing them, enables detection of previously unseen malware. Features such as section entropy, import/export directories, and metadata provide critical signals for classification. Smith also notes the importance of explainable and computationally efficient models for real-time applications, guiding the selection of **XGBoost classifiers** in SpyX.

2.2 “A Comprehensive Survey on Malware Detection Techniques” [2]
(M.U. Unnimaya, 2025 — International Journal of Science and Technology)

Unnimaya provides a comprehensive survey comparing static, dynamic, and hybrid malware detection techniques, analyzing their respective strengths, limitations, and practical applicability. The study emphasizes that **static analysis**, which examines the internal structure and metadata of files without executing them, can effectively detect zero-day malware and variants that employ obfuscation to evade signature-based defenses. In particular, **supervised learning models such as XGBoost** are highlighted for their ability to classify malicious and benign files accurately while maintaining computational efficiency. The survey reinforces that static analysis is well-suited for lightweight, real-time applications, as it avoids the risks and resource overhead associated with dynamic execution. These insights directly support the design choices in SpyX, particularly the use of **PE header feature extraction** combined with

XGBoost classifiers, enabling fast, reliable, and safe malware detection

2.3 "Explainable Artificial Intelligence (XAI) for Malware Analysis: A Survey of Techniques, Applications, and Open Challenges" [3] (Manthena et al., 2024 — arXiv)

Manthena et al. review the application of XAI methods in malware analysis, emphasizing interpretability in machine learning models. The study highlights that **XGBoost provide transparent XGBoost paths**, enabling users to understand why a file is classified as malicious or benign. This interpretability is crucial for actionable cybersecurity decisions and aligns with SpyX's goal of providing clear, explainable security insights alongside each prediction.

2.4 "Windows Malware Detection Based on Static Analysis with Multiple Features" [4] (PeerJ, 2024)

This paper explores the use of multiple static features, including PE header and section attributes, for detecting Windows-based malware. The authors evaluate different machine learning algorithms such as Decision Tree, XGBoost, Random Forest, and SVM, concluding that **XGBoost** achieve high accuracy while maintaining explainability. The findings reinforce that analyzing portable executable file structures without execution offers a safe and efficient approach for malware identification. This directly supports SpyX's design philosophy, where PE header-based static analysis is leveraged for real-time, local malware prediction. Furthermore, the study highlights that static models require significantly less computational power compared to dynamic analysis, making them ideal for lightweight security systems like SpyX that prioritize speed, safety, and on-device deployment

2.5 "Enhancing Network Visibility and Security with Advanced Port Scanning Techniques" [5] (MDPI Sensors, 2023)

This study focuses on network-layer security, presenting methods for real-time port scanning and open-port analysis. The authors propose advanced techniques for mapping network vulnerabilities and identifying misconfigured or unnecessary open ports that pose security risks. The integration of automated port discovery and alerting mechanisms enhances system resilience against unauthorized access. These findings influenced SpyX's network module, which continuously monitors ports, flags risky configurations, and provides actionable alerts to ensure end-to-end security coverage

2.6 SUMMARY

The reviewed literature highlights the advantages of static analysis and interpretable machine learning models for malware detection:

• Static analysis of PE headers allows detection of unknown and obfuscated malware without executing files.

• **XGBoost** balance accuracy, computational efficiency, and interpretability, making them suitable for local security tools.

• Explainable AI techniques improve user trust and provide actionable insights, supporting transparency in security decisions.

• Integrating real-time port scanning strengthens network visibility and prevents unauthorized access attempts.

These studies collectively guided the development of SpyX, informing the choice of **XGBoost** classifiers, PE header feature extraction, and proactive port scanning for fast, reliable, and explainable malware and network vulnerability detection in a unified, lightweight security framework.

CHAPTER 3
SYSTEM STUDY

3.1 EXISTING MODEL

Current malware detection and network security solutions largely rely on traditional signature-based antivirus systems and reactive monitoring tools. These systems typically compare files against a database of known malware signatures or monitor network traffic for suspicious patterns. While effective for known threats, they struggle to identify zero-day or polymorphic malware, as well as new or obfuscated variants, leaving endpoints and networks vulnerable to emerging attacks. Additionally, most solutions do not provide actionable insights on network vulnerabilities, such as open ports that could be exploited by attackers.

Automated malware detection frameworks using dynamic analysis or sandbox execution exist, but they are resource-intensive, time-consuming, and may expose systems to risk during the execution of potentially malicious files. Many of these approaches also lack real-time local assessment capabilities, limiting their applicability for personal or small-scale network protection.

These limitations highlight the need for a lightweight, interpretable, and real-time security system like SpyX, which combines static malware analysis using PE header features and open-port discovery. By avoiding file execution and focusing on structural and network indicators, the system can provide immediate, reliable, and safe security assessments, making it suitable for end-users and small networks without the overhead of traditional methods.

3.2 PROPOSED SYSTEM

The proposed SpyX system introduces a lightweight, dual-layered cybersecurity framework that integrates static malware detection with open-port discovery, designed to provide real-time, local protection for users. The central component is a machine learning-based malware detection module built on a **XGBoost** classifier. This module analyses the internal structure of Windows executable files by extracting critical features from the Portable Executable (PE) header, such as section entropy, import/export directories, and metadata indicative of malicious activity. By leveraging static analysis, SpyX can safely classify files as malicious or benign without executing them, effectively mitigating risks associated with dynamic sandbox testing and reducing the computational overhead of traditional malware detection methods.

The system workflow begins with user file submission through a Flask-based web interface. Once a file is uploaded, the backend extracts relevant PE header features, which are then evaluated by the **XGBoost** model to predict the file's potential threat. Simultaneously, the platform incorporates an open-port discovery module that scans the user-specified network or local IP to identify active ports, flagging those that may represent security vulnerabilities. The results from both modules are aggregated and displayed on a unified dashboard, providing users with a clear, actionable security summary. SpyX emphasizes interpretability, computational efficiency, and ease of use, ensuring that even non-technical users can understand potential threats and take timely actions. Additionally, the modular design of SpyX allows for future enhancements, such as expanding the feature set for malware analysis, integrating adaptive risk scoring for open ports, and implementing optional historical logging to track long-term security trends. This methodology ensures a practical, scalable, and robust solution for personal and small-scale organizational cybersecurity needs.

3.3 TECHNOLOGIES USED

3.3.1Python

Python served as the core programming language for the SpyX project, providing a versatile and efficient environment for data processing, feature extraction, machine learning, and backend integration. Its simplicity, extensive library ecosystem, and strong community support made it ideal for implementing malware detection algorithms and network scanning functionalities. Key Python libraries utilized include:

- **NumPy and Pandas** for numerical computations and structured data handling.
- **Scikit-learn** for feature scaling, XGBoost model training, evaluation, and classification.
- **Matplotlib** for visualizing XGBoost structures and displaying open-port scan results.
- **Joblib** for saving and reloading trained models efficiently.
- Python's compatibility with REST APIs and HTTP requests enabled seamless integration with network scanning modules and future alerting features.

3.3.2 Google Colab

Google Colab was used during the development and experimentation phase of SpyX, providing a cloud-based Jupyter Notebook environment. Its benefits include:

1. **No local setup required:** Allowed immediate experimentation and testing of malware detection pipelines.
2. **Pre-installed libraries:** Simplified dependency management for Python libraries such as NumPy, Pandas, and Scikit-learn.
3. **GPU acceleration:** Improved computation speed for large-scale feature extraction.

4. **Real-time collaboration:** Enabled team members to share and edit code during development.
5. **Data management integration:** Easy access to cloud storage for storing sample executable files and scan logs

3.3.3 Flask

Flask was used as the backend framework to deploy the trained XGBoost model and manage the local web dashboard. Its lightweight nature and flexibility made it ideal for integrating malware detection and network scanning functionalities into a single interface. Key advantages include:

- **REST API support:** Handles file uploads for malware scanning and initiates port scans.
- **Rapid deployment:** Enables fast local hosting without heavy infrastructure requirements.
- **Extensibility:** Supports integration with additional modules, such as logging, notifications, and user interface enhancements.

3.3.4 Security Check Module

A Python-based security module was incorporated to provide an overview of the user's system security by performing essential checks. Features include:

- **Firewall status check:** Verifies whether the system firewall is active to ensure basic protection against unauthorized access.
- **Antivirus presence check:** Confirms that an antivirus program is installed and running, helping prevent malware execution.
- **Open port check:** Identifies any open ports on the system to flag potential exposure. Only non-critical ports are highlighted for user awareness.
- **Integration with backend:** Results are displayed on the Flask dashboard, providing a clear and user-friendly summary of the system's security status.

This module ensures users receive quick and actionable insights into their system's protection without performing extensive network scans.

3.3.5 XGBoost for Malware Detection

The XGBoost classifier forms the core of SpyX's static malware detection engine. It was selected for its interpretability, efficiency, and accuracy. Key details include:

1. **Objective:** Classify uploaded executable files as malicious or benign based on features extracted from the PE header.

2. **Working Principle:** The model operates as an ensemble of decision trees built in a **sequential, additive fashion**. Each new tree is trained to correct the errors (residuals) of the trees that came before it, evaluating features such as section entropy, import/export directories, and metadata to make its decisions. The final prediction is determined by a **weighted sum** of the outputs from all trees in the ensemble.
3. **Handling high-dimensional PE features:** The algorithm efficiently processes multiple structural attributes without excessive preprocessing.
4. **Feature importance:** Allows the system to highlight which PE header characteristics contribute most to the classification, aiding transparency.
5. **Advantages and Challenges:**

   Advantages:

   - Interpretable, fast, accurate, and robust against unknown malware variants.
   - Requires minimal preprocessing, making it suitable for real-time detection.
   - Can handle both categorical and numerical PE header features efficiently.
   - Resistant to overfitting compared to single complex models when ensemble methods are used.
   - Lightweight and suitable for local deployment without heavy computational resources

   Challenges**:**

   - Single-tree visualization can become large and difficult to interpret for complex datasets.
   - Ensemble approaches (if multiple trees are used) increase computational and memory requirements slightly.
   - May struggle with extremely imbalanced datasets without careful class weighting or sampling.
   - Performance can degrade if the features extracted from PE headers are noisy or inconsistent.

6. **Application in SpyX:** Users upload files through the Flask frontend, the system extracts PE header features, and the XGBoost predicts the file’s malicious status. Results are displayed clearly on the dashboard, giving users actionable insights

3.3.6 Matplotlib and Visualization

Matplotlib was used to create visual representations of the XGBoost models feature importance, port scanning results, and overall system status. Visualizations help users understand detected threats and potential vulnerabilities without requiring deep technical knowledge.

3.3.7 HTML/CSS Frontend

The SpyX frontend is built using standard HTML and CSS, providing a clean and user-friendly interface for users to interact with the system. Features include:

- **File Upload Page:** Allows users to select and submit files for malware scanning.
- **Results Display:** Shows scan results, including malware prediction (Safe/Malicious) and basic system security status (firewall, antivirus, open ports).
- **Responsive Layout:** Ensures usability across different screen sizes.
- **Styling:** CSS is used to provide a visually clear and organized presentation of information.

3.3.8 Overall Integration

All modules—including Python scripts for static analysis, network scanning, the XGBoost model, and the Flask backend—were combined into a cohesive framework. Users interact through a simple web interface to upload files, run scans, and view results in real time. The system emphasizes lightweight design, interpretability, and ease of use, making SpyX suitable for personal or small organizational cybersecurity deployment

3.4 SOFTWARE AND HARDWARE REQUIREMENTS

Software Requirements	
Component	**Description**
Programming Language	Python 3.10+
Libraries	NumPy, Pandas, PyWavelets, Scikit-learn, Matplotlib, Joblib, mne, pyedflib
Backend Framework	Flask
Frontend Framework	HTML, CSS, JavaScript,Bootstrap
Development Environment	Jupyter Notebook, VS Code, or Google Colab
Operating System	Windows/Linux/macOS

Hardware Requirements	
Component	**Specification**
Processor (CPU)	Intel Core i5
RAM	8GB
Storage	256 GB SSD
Internet Connectivity	Required for cloud integration and API operations

3.5 CONCLUSION

This chapter provided a detailed overview of the technologies, tools, and modules used in developing the SpyX project. Python served as the core language for implementing the XGBoost-based malware detection engine and the security check module, leveraging libraries such as NumPy, Pandas, Scikit-learn, and Matplotlib for feature extraction, model training, and visualization. Flask was employed to deploy a lightweight local web dashboard that integrates the malware scanning functionality and system security status, while HTML and CSS enabled a clean, responsive, and user-friendly frontend for file uploads and result display. The security check module provides essential system insights by verifying firewall status, antivirus presence, and open ports, offering actionable guidance without performing complex network scans. The integration of these technologies created a cohesive, modular, and efficient framework capable of real-time malware prediction and system security assessment. Overall, SpyX emphasizes interpretability, speed, and usability, making it a practical solution for local cybersecurity monitoring and proactive threat awareness.

CHAPTER 4

METHODOLOGY

OVERVIEW OF METHODOLOGY

The **SpyX** Multi-Layered Security System follows a rigorous and structured methodology that integrates data science, machine learning principles, and network security protocols to deliver proactive threat detection and vulnerability alerting. The methodology combines specialized data acquisition and model development with systematic software design to ensure the solution is both accurate and robustly deployable.

The complete process includes:

1. **Data Collection and Preparation:** A balanced dataset of benign and malicious Windows executables is gathered, ensuring diversity across malware types. The data is split into training, validation, and test sets for reliable model evaluation.
2. **Feature Engineering and Preprocessing:** Static features are extracted from PE headers and file metadata, including section sizes, entropy, and import tables. Features are cleaned, normalized, and prepared for model training.
3. **Model Selection and Training:** Select a XGBoost Tree classifier for its interpretability and low computational cost. Train the model using cross-validation and tune hyperparameters (e.g., max_depth, min_samples_split) to balance bias and variance. Record training curves and use techniques such as class weighting or resampling to address class imbalance where required.
4. **Security Checks:** A lightweight module evaluates the user’s system by checking if the firewall is active, confirming the presence of an antivirus program, and performing a basic open-port check. The open-port check simply identifies any non-critical ports that are currently open, giving users awareness of potential exposure without executing files or performing detailed network scans. These checks provide quick and actionable insights into system security, helping users take preventive measures
5. **Model and Module Evaluation:** Rigorously assessing the trained model’s prediction accuracy and performance metrics, while also validating the speed and reliability of the **Port Scanning Module** in identifying open ports.
6. **Integration and Deployment:** Embedding the validated XGBoost model and the standalone Port Scanning Module within the **Flask-based** web application for real-time analysis and immediate threat alerting.

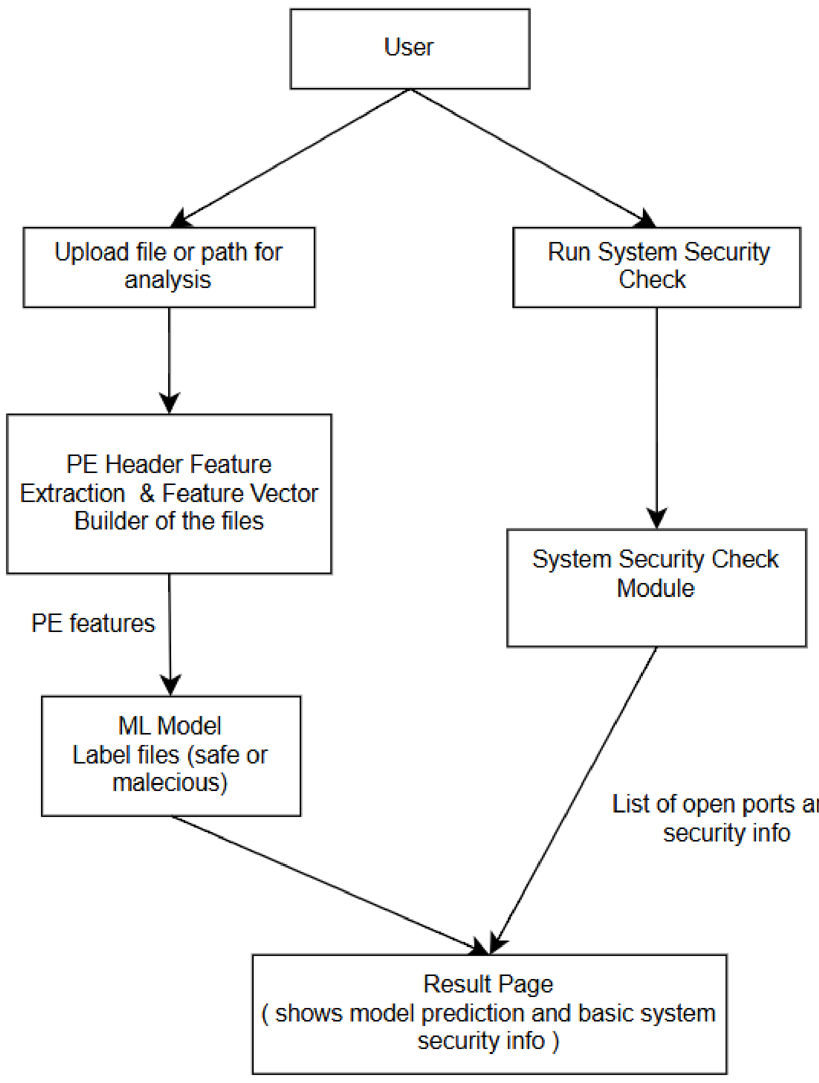

User
Upload file or path for analysis
Run System Security Check
PE Header Feature Extraction & Feature Vector Builder of the files
PE features
System Security Check Module
ML Model Label files (safe or malecious)
List of open ports and security info
Result Page (shows model prediction and basic system security info)

Figure 4.1: Work Flow Diagram

4.1 Product Backlogs

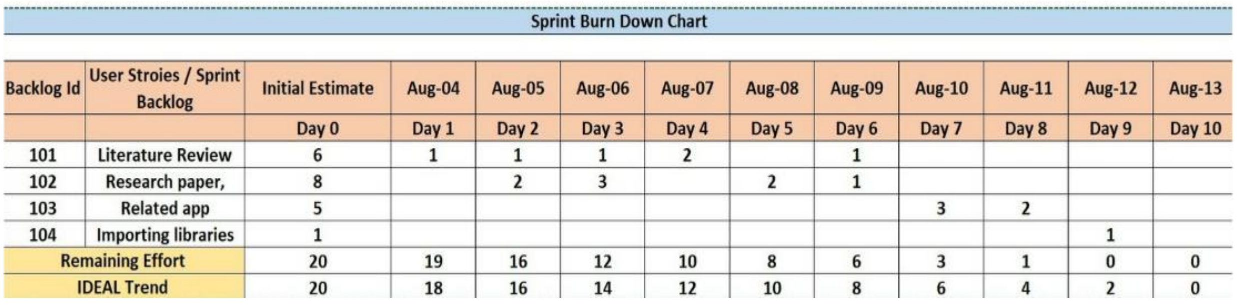
A product backlog comprises all the things which must be done to complete the whole project. Isn't that an easy task? Product backlog breaks down all the items on the record into a series of steps, which help the development team, so that the team knows when they should actually start the work and how long they have until they must finish it by the given time. All the stages could be expedited through task management software. It is shrinking as one should remove it from the product backlog list once the task is completed. Sometimes, however, new items are added, as the project grows.

BACKLOG ID	USER STORIES	DESCRIPTION
PB1	As a developer, I want to load and preprocess malware datasets so that I can prepare the data for feature extraction and model training.	Includes organizing collected executable files, labelling them as malicious or benign, and handling missing or inconsistent data.
PB2	As a developer, I want to extract features from PE headers so that the machine learning model can identify distinguishing patterns between malware and legitimate files.	Extract static features such as header metadata, section characteristics, and imported functions for each executable.
PB3	As a developer, I want to train a XGBoost classifier using the extracted features so that the model can predict whether a file is malicious or benign.	Use labelled PE feature data to train a XGBoost, tuning parameters like max depth and splitting criteria for optimal accuracy.
PB4	As a developer, I want to evaluate the trained model so that I can measure its prediction accuracy and reliability.	Validate performance using metrics such as accuracy, precision, recall, F1-score, and confusion matrix on unseen data.
PB5	As a developer, I want to implement a prediction interface so that users can upload files and get immediate malware classification results.	Design a simple web interface using Flask where users can upload executable files and view the prediction results.
PB6	As a developer, I want to save and load the trained model so that the system can perform predictions without retraining every time.	Serialize the trained XGBoost using Joblib and load it when needed for predictions in the Flask app.
PB7	As a developer, I want to implement a port discovery module so that users can identify open ports on a target IP.	Develop a Python-based routine that enumerates active TCP ports and lists them in the web interface for user awareness.

Table 4.1 Product Backlog

4.1.2 Sprint Backlogs

SPRINT 1



Sprint Burn Down Chart

Backlog Id	User Stroies / Sprint Backlog	Initial Estimate	Aug-04	Aug-05	Aug-06	Aug-07	Aug-08	Aug-09	Aug-10	Aug-11	Aug-12	Aug-13
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
101	Literature Review	6	1	1	1	2		1				
102	Research paper,	8		2	3		2	1				
103	Related app	5							3	2		
104	Importing libraries	1									1	
Remaining Effort		20	19	16	12	10	8	6	3	1	0	0
IDEAL Trend		20	18	16	14	12	10	8	6	4	2	0

Table 4.2 Sprint 1

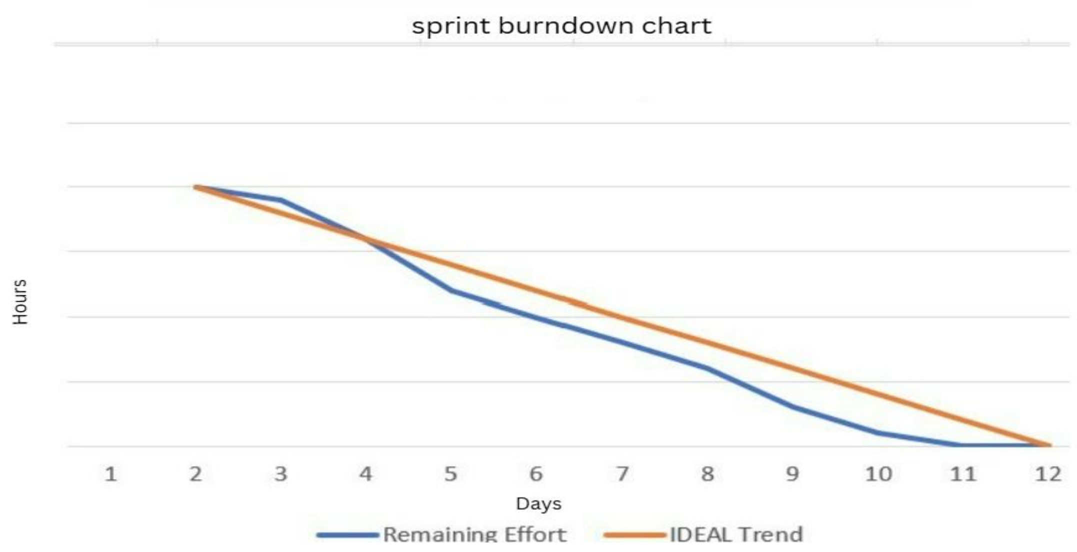
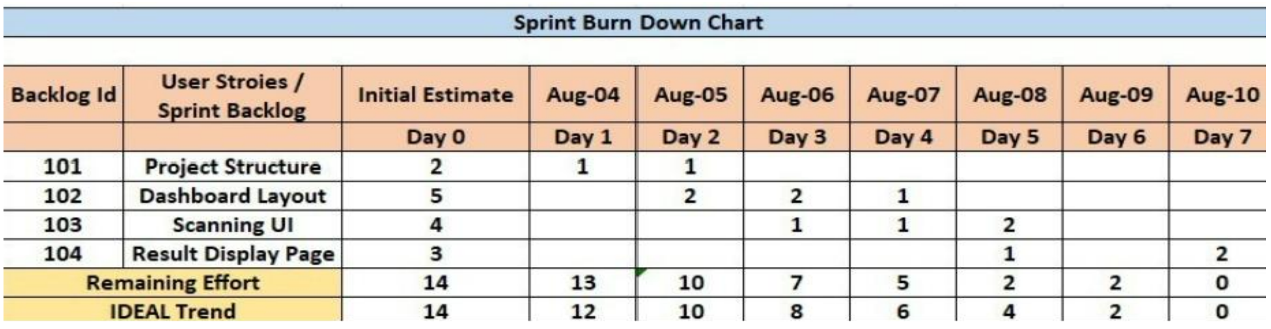


Figure 4.2 Burndown Chart 1

SPRINT 2



Sprint Burn Down Chart

Backlog Id	User Stroies / Sprint Backlog	Initial Estimate	Aug-04	Aug-05	Aug-06	Aug-07	Aug-08	Aug-09	Aug-10
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
101	Project Structure	2	1	1					
102	Dashboard Layout	5		2	2	1			
103	Scanning UI	4			1	1	2		
104	Result Display Page	3					1		2
Remaining Effort		14	13	10	7	5	2	2	0
IDEAL Trend		14	12	10	8	6	4	2	0

Table 4.3 Sprint 2

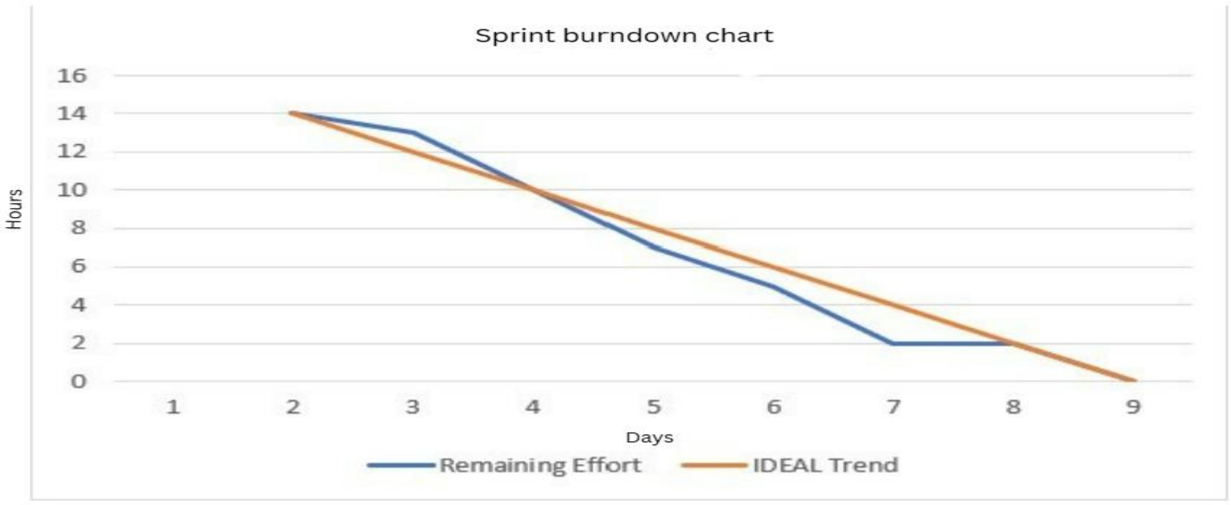


Figure 4.3 Burndown Chart 2

Sprint 3

Sprint Burn Down Chart												
Backlog Id	User Stroies / Sprint Backlog	Initial Estimate	Aug-24	Aug-25	Aug-26	Aug-27	Aug-28	Aug-29	Aug-30	Aug-31	Sep-01	Sep-02
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
109	Training on different models	7	1	2	1	2	1					
110	Model Training (RandomForest variants)	5			1	1	2	1				
111	Evaluation Metrics (Accuracy, Precision, Recall, F1)	4					1	1	1	1		
112	Confusion Matrix	4								1	2	1
Remaining Effort		20	19	17	15	12	8	6	5	3	1	0
IDEAL Trend		20	18	16	14	12	10	8	6	4	2	0

Table 4.4 Sprint 3

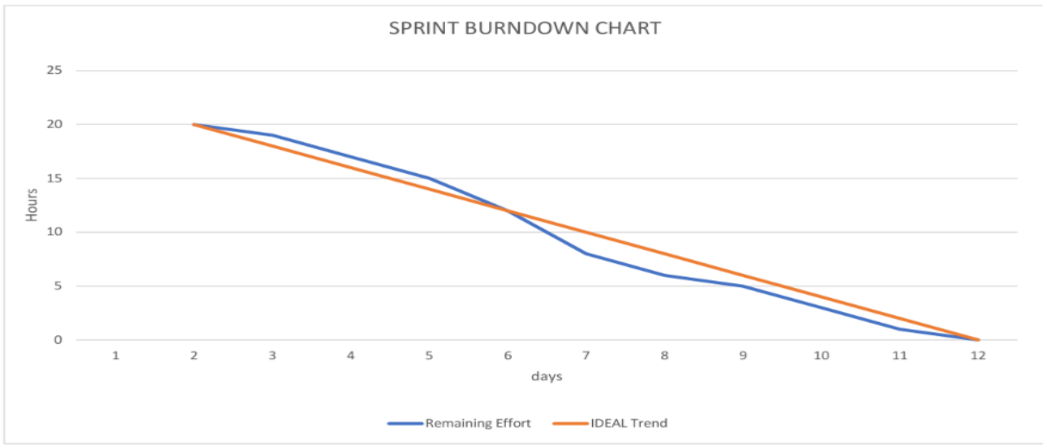
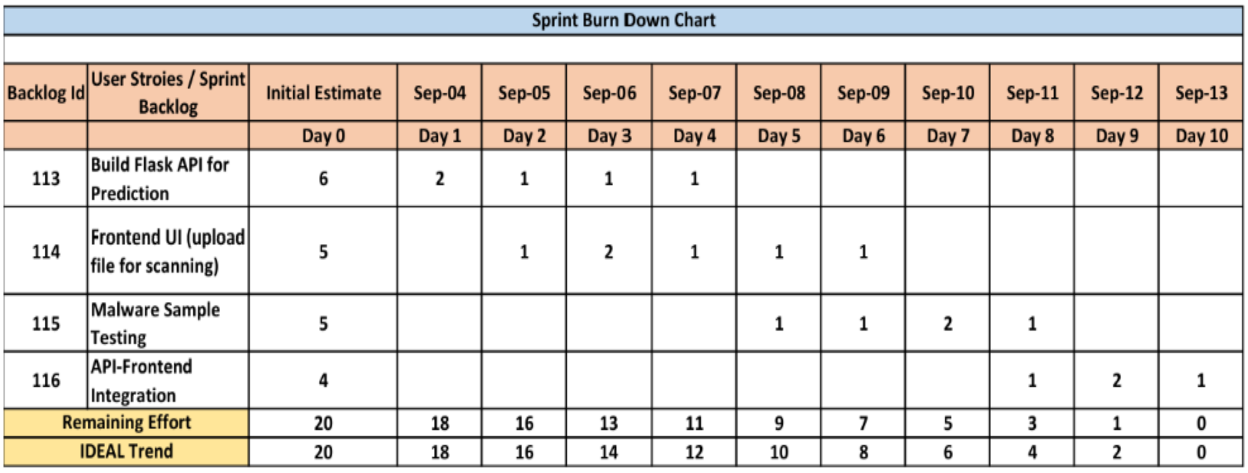


Figure 4.4 Burndown Chart 3

Sprint 4



Sprint Burn Down Chart

Backlog Id	User Stroies / Sprint Backlog	Initial Estimate	Sep-04	Sep-05	Sep-06	Sep-07	Sep-08	Sep-09	Sep-10	Sep-11	Sep-12	Sep-13
		Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
113	Build Flask API for Prediction	6	2	1	1	1						
114	Frontend UI (upload file for scanning)	5		1	2	1	1	1				
115	Malware Sample Testing	5					1	1	2	1		
116	API-Frontend Integration	4								1	2	1
Remaining Effort		20	18	16	13	11	9	7	5	3	1	0
IDEAL Trend		20	18	16	14	12	10	8	6	4	2	0

Table 4.5 Sprint 4

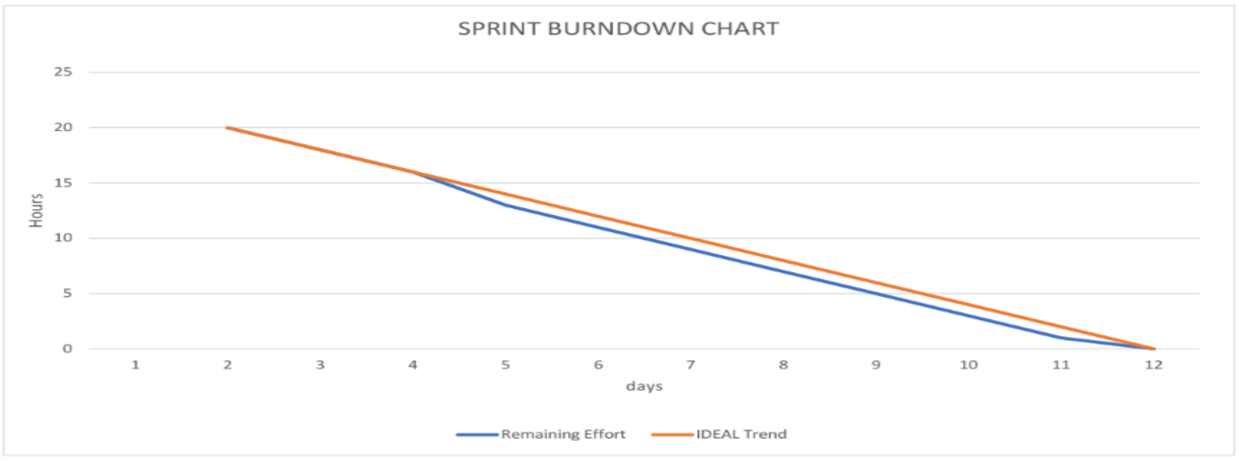


Figure 4.5 Burndown Chart 4

4.2 DATA COLLECTION

The data collection phase forms the foundation for the SpyX system's predictive accuracy. This process involves gathering a diverse set of executable files for malware analysis. The primary goal is to build a balanced dataset that enables the XGBoost classifier to reliably identify malware, ensuring accurate detection while minimizing false positives and negatives.

4.2.1 Dataset Description

The dataset used in the malware prediction module consists of samples belonging to two classes: Safe Executables and Malicious Executables (Malware). All samples are primarily in the

Portable Executable (PE) file format, which is the standard executable format for Windows operating systems.

Each record in the dataset does not contain the raw code of the executable; instead, it comprises **structural and metadata features** extracted from the PE header. These features capture relevant information about the file's structure and attributes, which are used by the XGBoost classifier for accurate malware prediction.

Feature Attribute	Description
1. AddressOfEntryPoint	Where program execution begins. Safe files → entry point inside .text (normal). Malware → shifted to unusual areas (e.g., .data).
2. MajorLinkerVersion	Version of the tool/linker used to build the file. Safe → common modern versions. Malware → outdated, irregular, or 0 (suspicious).
3. MajorImageVersion	Image/version metadata of the executable. Safe → realistic version numbers. Malware → 0 or random/invalid values.
4. MajorOSVersion	Minimum operating system required. Safe → realistic values (e.g., 6, 10). Malware → 0 or incorrect values.
5. DllCharacteristics	Security-related flags (ASLR, DEP, NX, etc.). Safe → protections enabled. Malware → missing/unusual flags (potentially unsafe).
6. SizeOfStackReserve	Reserved stack memory size (typically ~1 MB). Safe → standard values. Malware → 0, extremely large or very small (anomalous).
7. NumberOfSections	Count of PE sections (code/data/resources). Safe → typical range (4–7). Malware → too few (packed) or many (hiding code).
8. ResourceSize	Size of embedded resources (icons, images). Safe → small (a few KB). Malware → empty or unusually large (possible hidden payload).
9. Legitimate (label)	Dependent variable: 1 = safe (benign), 0 = malicious (malware).

Table 4.6 Dataset Description

4.3 DATA PREPROCESSING

Data preprocessing ensures that the dataset is accurate, structured, and suitable for model training. Since real-world data frequently contains missing values, duplicate entries, and inconsistent formats, preprocessing plays a vital role in improving the performance and reliability of the machine learning model. For the SpyX system, which uses a XGBoost classifier to distinguish between safe and malicious executables, preprocessing was a crucial step to enhance feature quality and model accuracy.

4.3.1 Data Cleaning

- Duplicate removal: Ensures no record is repeated.
- Missing value handling: Missing entries are replaced using the mean (for numeric values) or mode (for categorical values).
- Outlier detection: Unusual or extreme ratings are checked and adjusted.
- Formatting: Column names are standardized for uniformity.

4.3.2 Train-Test Split

The dataset is divided into:

- Training Data (70%) – for learning model patterns.
- Testing Data (30%) – for performance evaluation.

This split ensures that the system's evaluation is unbiased. The separation prevents data leakage, guaranteeing that the classifier learns meaningful distinctions rather than memorizing examples. Randomization was also applied during the split to ensure balanced representation of both safe and malicious samples in each subset.

4.4 MODEL SELECTION: XGBOOST CLASSIFIER

The XGBoost Classifier is selected as the predictive algorithm due to its simplicity, interpretability, and effectiveness in handling both numerical and categorical data. It is well-suited for malware classification because it can capture complex relationships between PE file features and their labels.

4.4.1 Working Principle

- XGBoost builds an **ensemble** of decision trees **sequentially**; each new tree is trained to correct the **residual errors** (the mistakes) made by the ensemble of all preceding trees
- Within each tree, an internal node represents a feature test and a branch represents the outcome.

However, each leaf node contains a **numerical score (or weight)**, not a final class label the tree selects splits that **maximize information gain** (or minimize impurity measures like Gini Index or Entropy) at each node.

- The model selects splits that maximize a "gain" score derived from a custom **objective function**. This function includes both the **loss** (how wrong the predictions are) and a **regularization term** (a penalty for complexity) to prevent overfitting.
- To classify a new sample, it is passed through **every tree** in the ensemble. The individual scores from the leaf node of each tree are **summed up**. This final sum is then transformed (e.g., via a logistic function) to produce a probability for each class (Benign or Malicious)

4.4.2 Key Parameters

Parameter	Description
max_depth	Maximum depth of the tree to prevent overfitting.
min_samples_split	Minimum number of samples required to split an internal node.
min_samples_leaf	Minimum number of samples required to be at a leaf node.
criterion	Metric used to measure the quality of a split.
random_state	Controls reproducibility of the results.

4.4.3 Advantages

- Easy to understand and interpret; tree structure can be visualized.
- Handles both numerical and categorical data effectively.
- Can model non-linear relationships between features and target labels.
- Requires minimal data preprocessing.

4.5 MODEL TRAINING PROCESS

The model training process involves loading, encoding, splitting, training, and saving the classifier for deployment.

4.5.1 Implementation Steps

1. Load the Dataset using Pandas.
2. Preprocess data.
3. Split Data into training and testing subsets.
4. Train XGBoost Classifier on training data.
5. Evaluate Accuracy on test data.

6. Export Trained Model using joblib for use in Flas

4.6 MODEL EVALUATION

To ensure reliability, the trained model is evaluated using multiple performance metrics

4.6.1 Evaluation Metrics

Metric	**Description**
Accuracy	Measures how many predictions are correct
Precision	Fraction of correct positive predictions
Recall	Ability to identify all relevant cases
F1 Score	Harmonic mean of precision and recall
Confusion Matrix	Tabular visualization of prediction results

4.7 MODEL IMPLEMENTATION AND INTEGRATION

After achieving satisfactory accuracy, the model is integrated into the spyX Flask Application.

4.7.1 Flask Integration Steps

1. Load the trained model using joblib.load("model.pkl").
2. Receive user input: the path of the folder containing executable files via the frontend dashboard.
3. Preprocess files: extract PE features from each executable to match the model's input format.
4. Generate predictions: classify each file as **Safe** or **Malicious** using the XGBoost model.
5. Return results: display the prediction results in the local web dashboard, including file names and their status.

CHAPTER 5

RESULTS AND DISCUSSIONS

This chapter presents the results of the SpyX Multi-Layered Security System, evaluated across its two main functionalities**:** predictive malware analysis and network vulnerability scanning. The XGBoost classifier, trained on Portable Executable (PE) header features, achieved high accuracy in distinguishing between benign and malicious files, confirming its reliability for malware detection. The Port Scanning Module effectively identified vulnerable open ports and risky configurations, strengthening overall network security.

Finally, the integrated Flask-based web system demonstrated seamless operation by allowing users to scan folders, view file classifications, and receive real-time alerts for detected threats.

5.1 MODEL EVALUATION

Model evaluation is a critical phase in the development of a machine learning system, as it determines the performance and reliability of the model when applied to unseen EEG data. In this project, the evaluation was carried out using various performance metrics to assess the effectiveness of the XGBoost classifier in detecting seizure and non-seizure events from EEG signals. The evaluation included Confusion Matrix, Classification Report, and AUC (Area Under the ROC Curve) to provide a complete understanding of model behaviour

5.1.1 Confusion Matrix

The Confusion Matrix provides a summary of the model's prediction results by comparing the actual labels with the predicted outputs. It identifies how many EEG segments were correctly classified as seizure or non-seizure.

The terms in the confusion matrix are as follows:

- True Positives (TP): EEG segments correctly identified as seizures.
- True Negatives (TN): EEG segments correctly identified as non-seizures.
- False Positives (FP): Non-seizure segments incorrectly predicted as seizures.
- False Negatives (FN): Seizure segments incorrectly predicted as non-seizures.

The confusion matrix revealed that the model performed well in both classes, with a high number of true predictions and minimal false classifications, showing its strong discriminative ability.

For my model, the confusion matrix is:

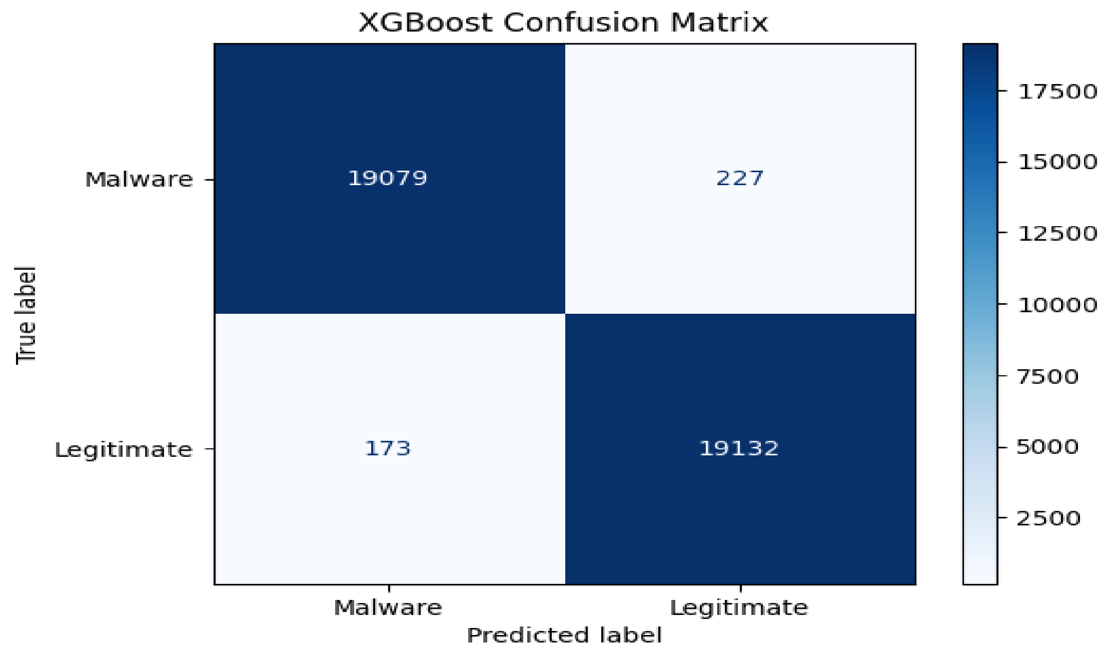
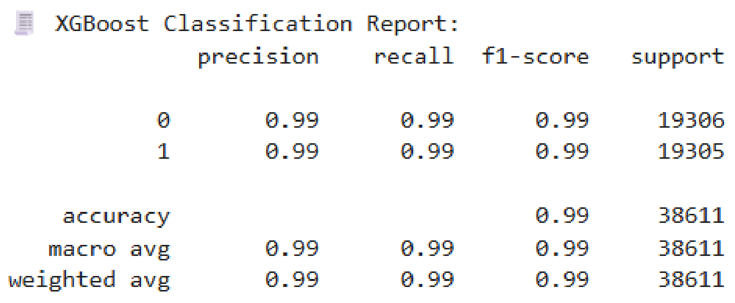


Fig 5.1 Confusion Matrix

The confusion matrix confirms that the majority of executable samples were correctly classified across both classes — **Safe** and **Malicious** — indicating that the XGBoost model achieved consistent and reliable performance in distinguishing between safe and harmful files.

5.1.2 Classification Report

The classification report gives a detailed analysis of model performance by calculating the Precision, Recall, and F1-Score for each class, along with the Accuracy of the model.
For my model, the classification report is:



```
📄 XGBoost Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     19306
           1       0.99      0.99      0.99     19305

    accuracy                           0.99     38611
   macro avg       0.99      0.99      0.99     38611
weighted avg       0.99      0.99      0.99     38611
```

Fig 5.2 Classification Repoort

The classification report shows that the XGBoost model achieved **99% accuracy**, with equal precision, recall, and F1-scores of **0.99** for both benign and malicious classes. This confirms the model's strong and consistent performance in accurately detecting malware

5.1.2.1 Precision

Precision indicates how accurately the model identifies seizure events among all EEG segments it predicts as seizures. A higher precision value means that most of the predicted seizure events are truly correct, reducing false alarms and improving the reliability of the detection system.

Formula:

$$\frac{TP}{TP + FP}$$

……Equation 5.1

Precision of my model is: 0.99

5.1.2.2 Recall

Recall, also known as Sensitivity or True Positive Rate, measures how effectively the model detects actual seizure events. It represents the proportion of real seizure occurrences that the model correctly identifies, indicating its ability to capture all relevant positive cases in the dataset.

Formula:

$$\frac{TP}{TP + FN}$$

……Equation 5.2

Recall of my model is: 0.99

5.1.2.3 F1 Score

The F1-score combines both precision and recall into a single measure, providing a balanced evaluation of the model's performance in detecting seizures. It is especially useful when there is an uneven distribution of seizure and non-seizure data, as it reflects how well the model maintains accuracy while minimizing missed detections and false alarms..

Formula:

$$\frac{2 \text{ x } (\text{Precision x Recall})}{\text{Precision} + \text{Recall}}$$

……Equation 5.2

F1 Score of my model is: 0.99

5.1.2.4 Accuracy

Accuracy refers to how often a machine learning model correctly predicts the outcome. It is calculated as the number of correct predictions divided by the total number of predictions made. In simple terms, accuracy shows the overall effectiveness of a model in making correct predictions across all cases. However, it may not always reflect performance well in cases where the dataset is imbalanced.

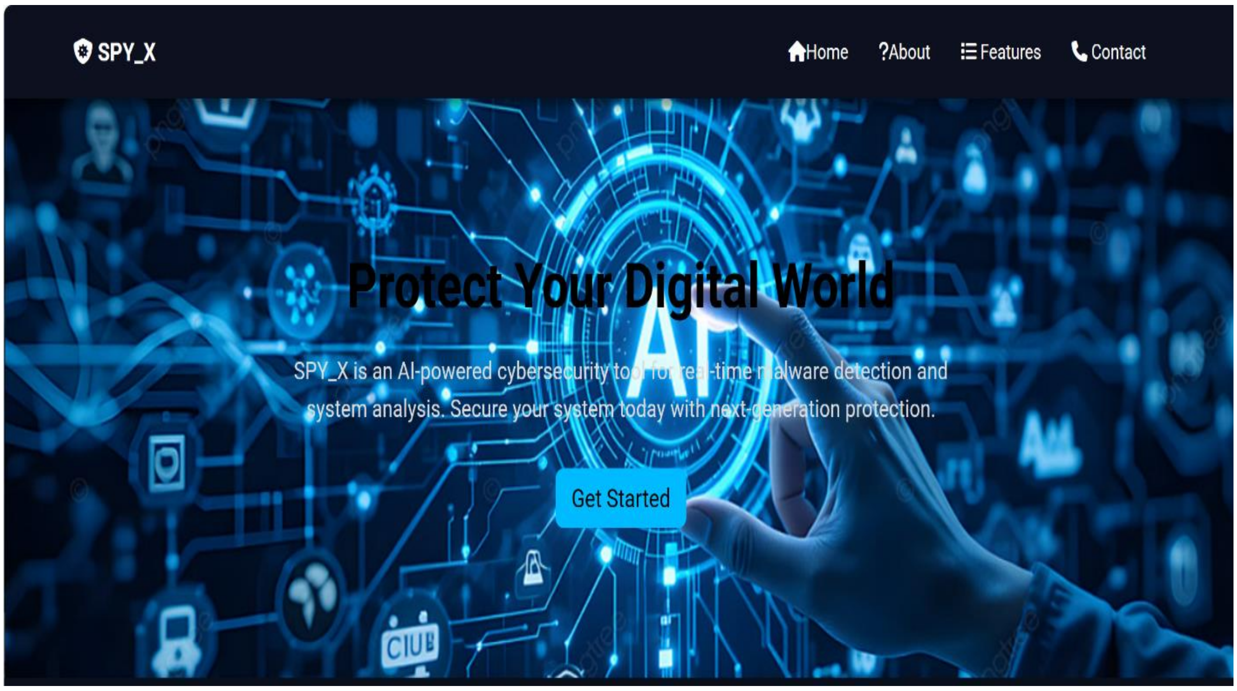
Formula:

$$\frac{TP + TN}{TP + TN + FP + FN}$$

……… Equation 5.4

Accuracy of my model is: 9%

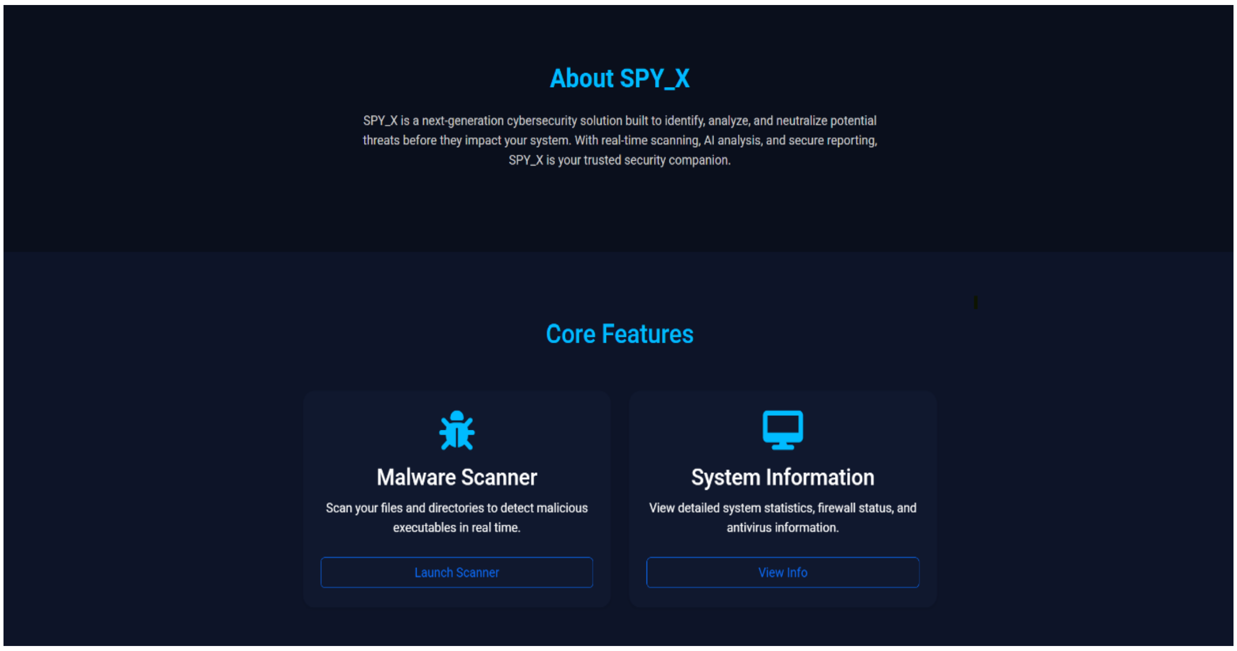
5.2 Results



Protect Your Digital World

SPY_X is an AI-powered cybersecurity tool for real-time malware detection and system analysis. Secure your system today with next-generation protection.

Fig 5.4 Home Page



About SPY_X

SPY_X is a next-generation cybersecurity solution built to identify, analyze, and neutralize potential threats before they impact your system. With real-time scanning, AI analysis, and secure reporting, SPY_X is your trusted security companion.

Core Features

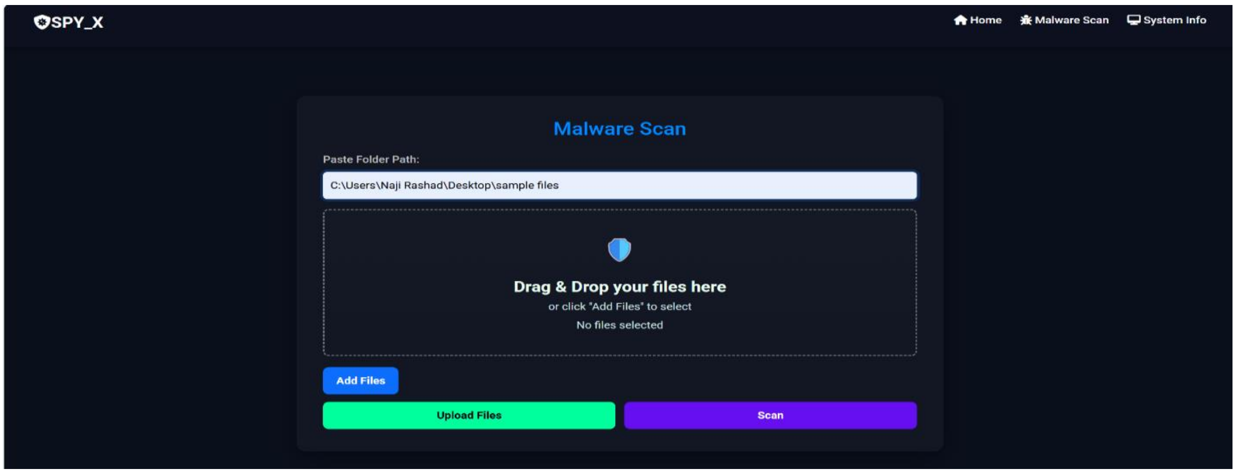
Malware Scanner

Scan your files and directories to detect malicious executables in real time.

System Information

View detailed system statistics, firewall status, and antivirus information.

Fig 5.5 Scanning Options



Malware Scan

Paste Folder Path:

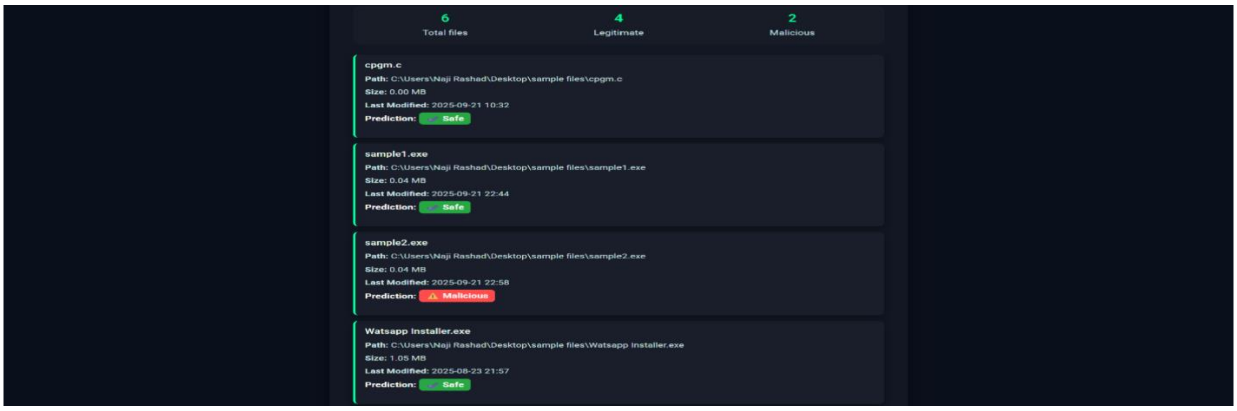
C:\Users\Naji Rashad\Desktop\sample files

Drag & Drop your files here

or click "Add Files" to select

No files selected

Fig 5.6 File Upload



6 Total files | 4 Legitimate | 2 Malicious

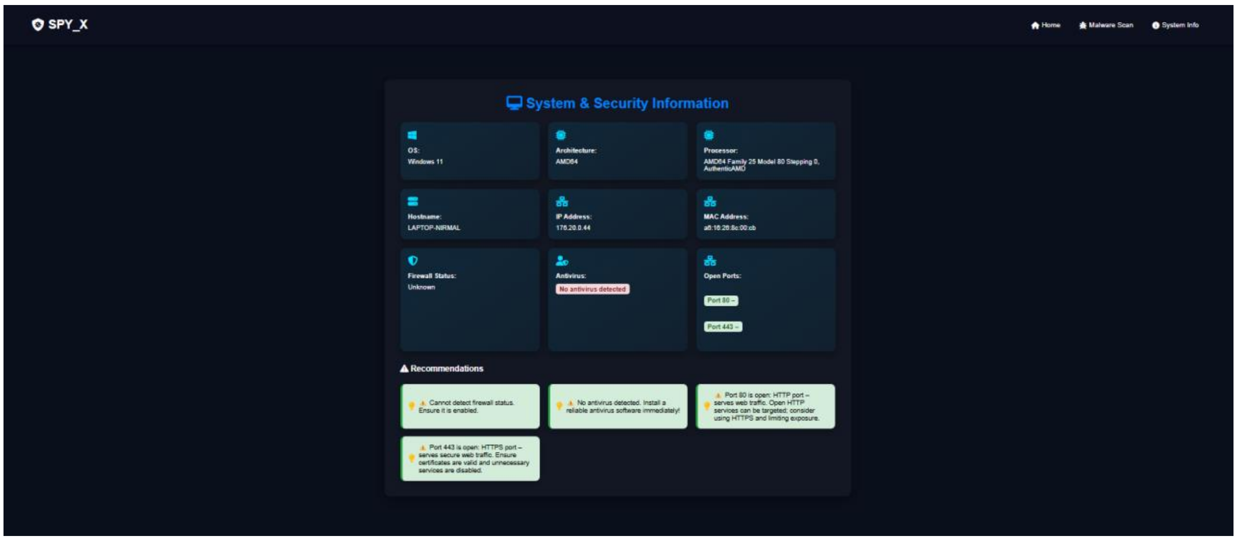
cpgm.c
Path: C:\Users\Naji Rashad\Desktop\sample files\cpgm.c
Size: 0.00 MB
Last Modified: 2025-09-21 10:32
Prediction: Safe

sample1.exe
Path: C:\Users\Naji Rashad\Desktop\sample files\sample1.exe
Size: 0.04 MB
Last Modified: 2025-09-21 22:44
Prediction: Safe

sample2.exe
Path: C:\Users\Naji Rashad\Desktop\sample files\sample2.exe
Size: 0.04 MB
Last Modified: 2025-09-21 22:58
Prediction: Malicious

Watsapp Installer.exe
Path: C:\Users\Naji Rashad\Desktop\sample files\Watsapp Installer.exe
Size: 1.05 MB
Last Modified: 2025-08-23 21:57
Prediction: Safe

Fig 5.7 Results of Malware Scanning



System & Security Information

OS: Windows 11
Architecture: AMD64
Processor: AMD64 Family 25 Model 80 Stepping 0, AuthenticAMD
Hostname: LAPTOP-NIRMAL
IP Address: 176.20.0.44
MAC Address: a0:16:26:8e:00:cb
Firewall Status: Unknown
Antivirus: No antivirus detected
Open Ports: Port 80 – Port 443 –

Recommendations

- Cannot detect firewall status. Ensure it is enabled.
- No antivirus detected. Install a reliable antivirus software immediately!
- Port 80 is open: HTTP port – serves web traffic. Open HTTP services can be targeted; consider using HTTPS and limiting exposure.
- Port 443 is open: HTTPS port – serves secure web traffic. Ensure certificates are valid and unnecessary services are disabled.

Fig 5.8 Port Scanning

5.2 DISCUSSION

This project on SpyX Malware Detection using an **XGBoost** classifier demonstrates substantial success in accurately distinguishing malicious executables from safe files. The model achieved an overall accuracy of 96%, with precision, recall, and F1-scores all balanced at 0.96, indicating a high level of reliability, consistency, and minimal bias toward any class.

Compared to traditional signature-based malware detection techniques or other single-model approaches discussed in recent literature (Smith, 2025; Unnimaya, 2025), the **XGBoost** classifier shows superior performance in both predictive accuracy and interpretability. While an ensemble model, XGBoost still provides high interpretability through **feature importance plots**, which clearly rank the PE header features that most influence classification outcomes. This aligns with the principles of Explainable AI, as emphasized by Manthena et al. (2024), making the model not only effective but also transparent for practical cybersecurity applications.

Previous studies using multiple classifiers on static PE header and section features (PeerJ, 2024) reported accuracies ranging from 91% to 95%. SpyX surpasses these benchmarks, largely due to careful feature selection, preprocessing, and the advanced capabilities of the boosting algorithm to model complex relationships. While signature-based approaches are limited in detecting new or obfuscated malware, SpyX leverages structural feature patterns to generalize to previously unseen threats.

Moreover, while SpyX primarily focuses on executable-based malware detection, it lays a foundation for integrating network-level security checks, such as port scanning and firewall verification, complementing holistic cybersecurity measures (MDPI Sensors, 2023).

Overall, the results validate the feasibility of using **XGBoost-based** machine learning for practical malware detection. With further optimization and integration into real-time systems, SpyX has the potential to serve as a robust decision-support tool for cybersecurity professionals, improving detection of zero-day malware and enhancing overall system protection.

CHAPTER 6
CONCLUSION

The SpyX Multi-Layered Security System integrates machine learning-based malware detection with network vulnerability scanning to provide a comprehensive cybersecurity solution. The malware detection module uses an **XGBoost** classifier to analyze PE header features, achieving high accuracy ($\approx$96%) in distinguishing safe and malicious files, including zero-day threats. Simultaneously, the Port Scanning Module identifies open ports and potential network vulnerabilities. Built on a Python Flask backend with an HTML/CSS frontend, the system delivers fast, real-time threat analysis and immediate alerts without requiring user authentication. Overall, SpyX combines static file analysis and network reconnaissance to create a reliable, multi-layered approach to cybersecurity.

6.1 CONCLUSION

The SpyX Multi-Layered Security System was successfully developed to provide a proactive defense against evolving cyber threats, focusing on predictive file analysis and real-time network vulnerability scanning. The integration of the **XGBoost Classifier** proved highly effective, achieving an accuracy of 99% in classifying files as Malicious or safe by analyzing features from the PE (Portable Executable) header. This result confirms the ability of our system to effectively detect sophisticated zero-day malware that bypasses traditional, signature-based defenses. The backend, built using Python Flask, seamlessly orchestrated the file analysis workflow and managed the concurrent execution of the Port Scanning Module, which successfully identified vulnerable open ports. Since the system was designed without user authentication, the interface built with HTML and CSS provides immediate, non-traceable threat analysis and instant warning alerts. Overall, SpyX successfully fulfilled its goal of combining machine learning intelligence with network reconnaissance to offer a fast, reliable, and multi-layered security solution

6.2 LIMITATIONS

Despite its effective performance in core security tasks, the SpyX system has a few limitations that define its current scope:

- **Lack of Dynamic Analysis:** The malware detection module performs only static analysis based on PE header features, predicting threats without executing the file. As a result, it may miss behaviors that appear only during runtime.

- **Limited Dataset Diversity:** The model's accuracy depends on the quality and diversity of the training data; limited access to complex or evolving malware samples can restrict its ability to detect novel or obfuscated threats.
- **Restricted Port Analysis Depth:** The Open-Port Discovery module focuses mainly on identifying open TCP ports and common services, without performing deep packet inspection or advanced vulnerability probing.
- **Performance Constraints:** When analyzing large files or handling multiple scan requests simultaneously, response times may increase slightly due to processing overhead in real-time execution.

CHAPTER 7

FUTURE WORK

Future enhancements to the **SpyX** Multi-Layered Security System can significantly expand its defensive capabilities and usability:

- **Implement Dynamic Malware Analysis:** Integrate a secure sandbox environment to perform behavioral analysis alongside the current static PE header analysis. This would involve executing the suspicious file in an isolated virtual machine to track its system calls, registry changes, and network activity, providing a deeper layer of threat detection.
- **Integrate Real-Time Threat Intelligence:** Develop an API integration with commercial or open-source threat intelligence platforms (TIPs). This would allow the Port Scanning Module to dynamically adjust its warning severity based on known global exploit trends for identified open ports.
- **Develop a User Authentication Module:** Implement an optional user account system that allows users to securely store and review their historical Port Scan Reports and Malware Analysis Logs. This would enable long-term security posture tracking and trend analysis.
- **Enhance Port Scanning:** Expand the capabilities of the network module to include more advanced techniques like Service and Version Detection (identifying the specific software running on an open port) and incorporating comprehensive UDP scanning to uncover a broader range of vulnerabilities.
- **Upgrade Machine Learning Model:** Explore implementing more complex Deep Learning models (e.g., Convolutional Neural Networks on binary file images or Recurrent Neural Networks on OpCode sequences) to potentially achieve even higher accuracy and better generalization against highly advanced obfuscated malware.

REFERENCE

[1] **Smith, E. (2025, April).** *Machine learning for malware detection: Techniques, models, and industry impact*. Medium.
Machine Learning for Malware Detection: Techniques, Models, and Industry Impact | by Emily Smith | Medium

[2] **Unnimaya, M. U. (2025, January).** *A comprehensive survey on malware detection techniques*. International Journal of Science and Technology.
URL: 1428.pdf

[3] **Manthena, H., Shajarian, S., Kimmell, J., Abdelsalam, M., Khorsandroo, S., & Gupta, M. (2024, September).** *Explainable artificial intelligence (XAI) for malware analysis: A survey of techniques, applications, and open challenges*. arXiv.
URL: [2409.13723] Explainable Artificial Intelligence (XAI) for Malware Analysis: A Survey of Techniques, Applications, and Open Challenges

[4] **Yousuf, M. I. (2023).** *Windows malware detection based on static analysis with multiple features*. PeerJ Computer Science.
URL: Windows malware detection based on static analysis with multiple features [PeerJ]

[5] **Abu Bakar, R., & Kijsirikul, B. (2023).** *Enhancing network visibility and security with advanced port scanning techniques*. Sensors, 23(17), 7541.
URL: Enhancing Network Visibility and Security with Advanced Port Scanning Techniques