INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Classificação Automática de Alertas de Cibersegurança num SOC

TOMÁS LADEIRO DOMINGUES
Junho de 2025

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Automatic Classification of Security Alerts in SOC

Tomás Ladeiro Domingues

A dissertation submitted in partial fulfillment of the requirements for the degree of Master of Science, Specialisation Area of Cybersecurity And Systems Administration

Advisor: Jorge Pinto Leite
Supervisor: Gonçalo Amaro

Porto, June 27, 2025

Statement of Integrity

I hereby declare having conducted this academic work with integrity.

I have not plagiarised or applied any form of undue use of information or falsification of results along the process leading to its elaboration.

Therefore the work presented in this document is original and authored by me, having not previously been used for any other end.

I further declare that I have fully acknowledged the Code of Ethical Conduct of P.PORTO.

ISEP, Porto, June 27, 2025

Dedicatory

To my family and friends.

Abstract

Threats to Cybersecurity have grown ever more sophisticated over the years, making Security Operations Center (SOC) more important than ever.

This dissertation explores the application of Machine Learning (ML) to automate the triage of security alerts, addressing alert fatigue and high false positive rates. The solution integrates a Random Forest (RF) model, trained on historical SOC data, with a Reinforcement Learning (RL) feedback loop that dynamically adapts to analyst input over time. A comprehensive review of Security Information and Event Management (SIEM) systems, ticketing tools, and ML frameworks was conducted to support the development of this system.

The research involved real-world deployment within a production SOC environment, using live data from ArtResilia's infrastructure. The proposed solution demonstrated significant improvements across key metrics, increasing classification accuracy for both alert priority and taxonomy after iterative refinement.

Moreover, the adaptive RL feedback loop appeared to enable continuous improvement while maintaining model stability. The findings suggest that integrating ML and RL into SOC workflows may help reduce false positives, improve response times, and alleviate analyst workload, potentially contributing to enhanced overall resilience.

Keywords: Cybersecurity, SIEM, QRadar, Machine Learning, Automation, Ticket Triage

Resumo

A crescente complexidade e frequência das ameaças cibernéticas têm destacado a importância dos Centros de Operações de Segurança (SOC) na defesa de organizações contra incidentes de segurança.

Os problemas abordados neste trabalho emergem da necessidade crescente de eficiência nos processos iniciais de triagem. As equipas SOC lidam diariamente com milhares de alertas, sendo uma parte considerável destes irrelevantes ou falsos positivos. Esta sobrecarga compromete não apenas o desempenho operacional dos analistas, mas também o tempo de deteção (MTTD) e o tempo de resposta (MTTR) da organização, aumentando o risco de incidentes não detetados ou tardiamente priorizados.

Para enfrentar estes desafios, foi concebida uma solução baseada na integração de modelos de ML com o ecossistema já existente de ferramentas SIEM e de gestão de incidentes. A metodologia desenvolvida contemplou uma extensa revisão bibliográfica, analisando os principais sistemas SIEM, plataformas de gestão de tickets, bem como frameworks de ML, avaliando as respetivas vantagens, limitações e adequação a ambientes SOC.

O modelo desenvolvido neste trabalho adota uma arquitetura híbrida e modular, composta por duas camadas principais. Na primeira camada, um modelo Random Forest (RF) foi treinado com um conjunto de dados históricos disponibilizado pela ArtResilia, empresa onde decorreu a investigação e a implementação prática da solução. Este dataset continha cerca de 100 mil alertas registados entre fevereiro de 2022 e janeiro de 2025, após uma formatação, normalização e balanceamento de classes, de forma a garantir a representatividade adequada das categorias de prioridade e de taxonomia de alertas.

A segunda camada integra um modelo de Aprendizagem por Reforço (Reinforcement Learning - RL). Este modelo é responsável por adaptar as previsões iniciais do Random Forest, incorporando feedback contínuo dos analistas do SOC relativamente à precisão das classificações automáticas realizadas. A implementação de RL permite que o sistema evolua progressivamente à medida que novos casos e padrões de ameaças emergem, combatendo assim uma das limitações habituais dos modelos exclusivamente supervisionados.

A integração prática da solução foi realizada diretamente no ambiente tecnológico da ArtResilia, com comunicação entre a plataforma IBM QRadar SOAR, o SIEM, e o módulo de ML desenvolvido, através de interfaces API específicas para previsões e recolha de feedback. Este design permitiu assegurar um fluxo contínuo de dados em tempo real, respeitando os processos operacionais existentes e garantindo uma adoção progressiva da solução pelos analistas do SOC.

A fase de validação incluiu testes locais (offline), bem como testes em ambiente real (produção). Nos testes locais, o modelo Random Forest evoluiu significativamente ao longo de múltiplas versões de treino, destacando-se o progresso obtido entre a versão inicial (V1) e a versão final (V11), onde se verificou um aumento da precisão geral de classificação, tanto na atribuição de prioridade (P1, P2, P3) como de taxonomia (fraude, intrusões, conteúdos

abusivos, entre outros). Estas melhorias refletiram-se também nos valores de recall e F1-score, demonstrando uma maior capacidade do modelo em reconhecer corretamente eventos relevantes, particularmente nas categorias minoritárias que inicialmente apresentavam maior dificuldade de identificação.

No ambiente de produção, a componente de RL passou a desempenhar um papel determinante, ajustando continuamente os seus parâmetros com base no feedback manual recolhido pelos analistas sobre as previsões emitidas. Esta capacidade de aprendizagem incremental, aliada à robustez inicial do modelo Random Forest, permitiu à solução lidar de forma eficaz com a chegada de novos padrões de ataque não contemplados no conjunto de treino inicial, reforçando a sua aplicabilidade prática em contextos dinâmicos e em constante mutação.

Do ponto de vista tecnológico, a implementação recorreu a um conjunto de ferramentas modernas e eficientes, como o Scikit-learn para o treino do modelo Random Forest, o Stable Baselines3 para o modelo RL, o FastAPI para a exposição dos serviços API, o Celery para o processamento assíncrono das tarefas de treino e o Redis para a gestão de comunicação entre os módulos. Esta escolha tecnológica permitiu garantir escalabilidade, modularidade e facilidade de manutenção da solução proposta.

A arquitetura modular da solução, bem como a sua integração transparente com o ambiente SOC, demonstrou ser eficaz em reduzir a carga de trabalho manual dos analistas, melhorar os tempos de resposta e deteção, e mitigar o risco associado à perda de alertas críticos devido a erros humanos ou à sobrecarga de trabalho.

Em conclusão, o trabalho desenvolvido demonstra que a aplicação combinada de modelos supervisionados e de reforço permite obter ganhos significativos de eficiência operacional num SOC, contribuindo para uma melhor priorização de alertas, redução de falsos positivos, otimização da carga de trabalho dos analistas e, em última instância, para o fortalecimento da resiliência cibernética das organizações.

Palavras-Chave: Cibersegurança, SIEM, QRadar, Aprendizagem Automática, Automação, Triagem de Alertas

Acknowledgement

I would like to express my deepest gratitude to my advisor, Professor Jorge Pinto Leite at ISEP, for his invaluable guidance, support, and dedication throughout this project. His insights and expertise were crucial in shaping this work and guiding me through the challenges of research and development.

I also extend my heartfelt thanks to my supervisor, Gonçalo Amaro, at ArtResilia, for accepting my idea and providing the opportunity to implement it within the organization. His continuous support, advice, and willingness to share his knowledge were essential in bringing this project to fruition.

Their combined mentorship and encouragement were instrumental in the completion of this dissertation, and I am profoundly grateful for their time and efforts.

Contents

List of Figures **xv**

List of Tables **xvii**

List of Listings **xix**

List of Acronyms **xxi**

1 Introduction **1**
1.1 Context . . . 1
1.2 Problem . . . 2
1.3 Objectives . . . 2
1.4 Research Contextualization . . . 3
1.5 Research Questions . . . 4
1.6 Dissertation Structure . . . 6
1.7 Ethical Considerations . . . 6

2 State of the Art **7**
2.1 Theoretical Introduction . . . 7
2.1.1 Security Operations Center . . . 7
Definition and Characteristics of a SOC . . . 8
Key Responsibilities of a SOC . . . 8
Tiers of Operation . . . 8
Triage Specialist . . . 9
2.1.2 Security Information and Event Management . . . 10
Architectural Components . . . 11
2.1.3 Machine Learning . . . 12
2.1.4 Machine Learning Models . . . 14
Decision Trees . . . 14
Ensemble classifiers . . . 15
Random Forests . . . 17
Naive Bayes . . . 19
2.2 Automatic Classification of Security Alerts in SOC . . . 20
2.3 Technologies . . . 23
2.3.1 SIEM Tools . . . 23
QRadar . . . 23
Splunk . . . 23
LogRhythm . . . 24
2.3.2 Ticketing Tools . . . 24
IBM QRadar SOAR . . . 24
ServiceNow . . . 24

JIRA . . . 25
2.3.3 Machine Learning Frameworks . . . 25
Scikit-learn . . . 25
TensorFlow . . . 25
PyTorch . . . 26
2.3.4 Comparative Analysis . . . 26
SIEM Tools . . . 26
Ticketing Tools . . . 27
Machine Learning Frameworks . . . 27
2.3.5 Conclusion . . . 28

3 Method and Implementation **29**
3.1 Method . . . 29
3.1.1 Technological Overview . . . 29
3.1.2 Problem-Solving Approaches . . . 30
Random Forest with Reinforcement Learning Feedback Loop . . . 31
End-to-End Deep Learning Classifier with Feature Fusion . . . 32
Rule-Augmented Decision Tree with Feedback Aggregation . . . 33
Architecture . . . 35
3.1.3 Comparative Analysis . . . 36
3.2 Proof of Concept . . . 37
3.2.1 Data Processing . . . 37
3.2.2 Data Normalization . . . 40
3.2.3 Dataset Division . . . 41
3.2.4 Machine Learning Model Development . . . 42
Random Forest Model Implementation . . . 42
Reinforcement Learning Model Implementation . . . 44
3.2.5 Hyperparameter Tuning . . . 46

4 Model Evaluation and Results Analysis **49**
4.1 Local Testing Environment . . . 49
4.2 Live Testing Environment . . . 53
4.2.1 Deployment Architecture and Integration Workflow . . . 54
4.2.2 SOAR Integration via Playbook . . . 55
4.2.3 Runtime Challenges and Solutions . . . 55
4.2.4 Results and Observations . . . 56
Accuracy and Learning Curves Over Time . . . 56
Training Feedback via Celery and Flower Monitoring . . . 58
Evaluation of Overfitting and Underfitting Patterns . . . 58
4.2.5 Final Analysis . . . 59

5 Conclusion and Future Work **61**
5.1 Future Work . . . 61
5.2 Conclusion . . . 62

Bibliography **63**

Appendix A Raw Outputs: Version 1 vs. Version 11 **67**

Appendix B Daily Prediction Accuracy During Live Testing **71**

List of Figures

2.1 SOC Analyst Tier Responsibilities, from (Kokulu et al. 2019). 9
2.2 SIEM Architecture 12
2.3 A basic structure of a Decision Tree, based on (Chauhan 2022) 14
2.4 Diagram of Parallel Ensemble Learning from (Mienye and Sun 2022) 16
2.5 Diagram of Sequential Ensemble Learning. from (Mienye and Sun 2022) . 17
2.6 Workflow of Random Forests: Bootstrapping, Decision Trees, and Aggregated Predictions from (Yang et al. 2019) 18
2.7 Schematic Diagram of the Naive Bayes Classification Process from (Sneha and Gangil 2019) 19

3.1 General Pipeline for all Three Solutions 30
3.2 Part C of the General Pipeline for the Random Forest with Reinforcement Learning Feedback Loop Solution 32
3.3 Architecture of the Random Forest with Reinforcement Learning Feedback Loop Solution 35
3.4 Original Dataset Before Preprocessing 37
3.5 Original Dataset After Preprocessing 40
3.6 Redacted Description Text 40

4.1 Confusion Matrix for Priority (Version 1) 51
4.2 Confusion Matrix for Priority (Version 11) 51
4.3 Confusion Matrix for Taxonomy (Version 1) 52
4.4 Confusion Matrix for Taxonomy (Version 11) 53
4.5 Correct Predictions per Day (Priority + Taxonomy) 56
4.6 Accuracy per Day (Both Labels Must Match) 57
4.7 Flower Monitoring Dashboard showing Successful and Failed Training Tasks 58

List of Tables

1.1 PICOCS Framework for Research Questions 5

2.1 Summary of the Existing Related Works based on Ali, Shah, and ElAffendi (2024). 21
2.2 Comparative Analysis of SIEM Tools. 26
2.3 Comparative Analysis of Ticketing Tools. 27
2.4 Comparative Analysis of Machine Learning Frameworks. 27

3.1 Comparison of Proposed Solutions 36

4.1 Model Performance Metrics Comparison: Version 1 vs. Version 11 50

A.1 Version 1 Priority Classification Metrics 67
A.2 Version 1 Taxonomy Classification Metrics 68
A.3 Version 11 Priority Classification Metrics 68
A.4 Version 11 Taxonomy Classification Metrics 68

B.1 Daily Prediction Statistics During Live Testing 71
B.2 Live Priority Classification Metrics 72
B.3 Live Taxonomy Classification Metrics 72

List of Listings

1 Python Code Snippet for the First Part of Preprocessing the Dataset. . . . 38
2 Python Code Snippet for the Second Part of Preprocessing the Dataset. . 38
3 Python Code Snippet for the Third Part of Preprocessing the Dataset. . . 39
4 Python Code Snippet for the Fourth Part of Preprocessing the Dataset. . . 39
5 Splitting Dataset. 42
6 Stratified Sampling. 42
7 Vectorizing Text Data. 43
8 Encoding Target Labels. 43
9 Splitting the Dataset. 43
10 Training the Random Forest Model. 44
11 RL Training Task with Celery. 45
12 Random Forest Model Initialization. 46
13 Custom Garbage Collection Strategy 56

List of Acronyms

24/7	twenty-four seven.
AI	Artificial Intelligence.
APT	Advanced Persistent Threats.
CND	Computer Network Defense.
CSIRT	Computer Security Incident Response Team.
DNN	Deep Neural Network.
DTC	Decision Tree Classifier.
DTM	Decision Tree Model.
EDR	Endpoint Detection and Response.
IDS	Intrusion and Detection System.
IPS	Intrusion and Prevension System.
ML	Machine Learning.
RF	Random Forest.
RL	Reinforcement Learning.
SIEM	Security Information and Event Management.
SOC	Security Operations Center.

Chapter 1

Introduction

This chapter provides context for automating alert triage and its significance in enhancing security operations in a SOC, particularly for managing high volumes of security alerts. It begins with defining the problem and detailing the challenges associated with manual alert analysis and the critical requirements for automation. First, the objectives of this thesis are presented; second, this project's expected outcomes, including efficiency and accuracy improvements, are outlined, along with the selected approach for integrating an existing ML model to automate alert classification. Finally, an overview of the report's structure is provided.

1.1 Context

In today's world, cybersecurity has become essential for organizations of all sizes. (Vielberth et al. 2020) The rapid advancement of technology, particularly artificial intelligence, has fostered innovation and growth and equipped cybercriminals with increasingly sophisticated tools and techniques. Cyber threats have surged over recent years, and this trend is expected to continue, with global cybersecurity spending projected to rise from $6 trillion to $30 trillion by 2030. (Sovilj et al. 2020)

As these threats become more complex and frequent, Security Operations Center (SOC) have emerged as the primary line of defense. SOCs are crucial in monitoring, detecting, and responding to cyber threats, ensuring the protection of organizations' core digital assets—data, applications, and infrastructure. By combining human expertise with advanced technology, SOCs proactively defend systems, respond to incidents, and work to minimize the impact of potential breaches, safeguarding essential operations. (Jalalvand et al. 2024)

However, managing the overwhelming number of alerts generated daily within SOCs presents a significant challenge. Analysts must review and respond to thousands of tickets, a process that can quickly lead to burnout, as stated by Tines (2023), especially when the majority of alerts are false positives. A study found that many organizations receive over 10,000 alerts daily, with more than 50% being false positives. (CriticalStart 2019)

One of the stages in the SOC workflow with a higher alert volume is the initial triage process. At this stage, Tier 1 analysts are responsible for gathering raw data, assessing alarms, and determining the urgency and nature of each alert. For every ticket, they must evaluate whether the alert is valid or a false positive, assign priority based on the alert's severity, and identify any potential high-risk incidents. (Vielberth et al. 2020) This process demands significant time and effort, placing a heavy mental load on analysts as they deal with large volumes of information.

Integrating automation in the triage process, particularly for determining alert criticality and classification, could substantially reduce analyst fatigue. Machine Learning (ML) techniques are up-and-coming for processing large data volumes. They facilitate the classification and prioritization of alerts, enabling analysts to focus on high-priority threats. (Jalalvand et al. 2024)

This work explores how SOCs can use automation and ML to streamline alert triage, improve SOC efficiency, and more effectively protect organizations from an ever-evolving cyber threat landscape.

1.2 Problem

The main objective of this work is to develop an independent program that automates the triage of security alerts within a Security Operations Center (SOC) environment. This automation aims to enhance the overall efficiency and accuracy of the triage process, reduce the number of false positives, and alleviate the workload during the initial stages of the SOC workflow.

Improving the speed of analyzing initial alerts is crucial in a Security Operations Center (SOC) workflow. Faster analysis enables SOC analysts to respond to incidents quickly, which helps minimize the time that threats remain undetected. Without automation, this initial alert analysis depends heavily on manual inspection. Analysts face the challenge of sifting through a large volume of alerts to identify potential threats, which is time-consuming and susceptible to human error.

This reactive, manual method increases response times and raises the risk of missing critical threats. Analysts must manually classify alerts, assess their severity, and determine appropriate response actions while managing a continuous influx of new alerts. This approach can strain resources and often leads to alert fatigue, making it difficult for analysts to prioritize and respond effectively.

Key metrics such as Mean Time to Detect (MTTD) and Mean Time to Respond (MTTR) can significantly improve by reducing the time required for initial analysis. These metrics are essential for maintaining strong cybersecurity defenses. Automation in alert triage allows for faster and more accurate threat identification, enabling analysts to focus on high-priority threats. This not only increases overall productivity but also reduces fatigue among analysts, while boosting the effectiveness of cybersecurity.

This efficiency results in a more manageable workload, decreased burnout, and greater job satisfaction for employees. For the organization, streamlined response times enhance the security posture, lower the risk of breaches, and ultimately protect critical assets more effectively.

1.3 Objectives

The objectives for this project are as follows:

1. Literature search and review
    - A research into the latest advancements in security analysis and a thorough analysis of the tools, methodologies, and approaches researchers use in SOC automation and machine learning applications.

2. Study of existing Machine Learning (ML) models
    - An analysis of different models for alert triage in SOC environments and related fields, aiming to identify and improve the adopted approach.
3. Collection of security alert datasets
    - A compilation of accessible datasets on security alerts organized to supply training data for the selected ML model.
4. Integration with SIEM and other data sources
    - Connect the SIEM and relevant data sources for real-time data ingestion and analysis.
5. Developing a custom dashboard for analysts to submit feedback on the machine learning model.
    - While the ML model will be trained with existing datasets before deployment, it is still beneficial to provide a method for analysts to continue training the model over time.
6. Testing and evaluation of the selected ML model
    - The selected ML model will be tested with live data, followed by an analysis to assess its effectiveness in categorizing alerts and minimizing false positives.

1.4 Research Contextualization

This research will be conducted at ArtResilia, a cybersecurity firm dedicated to improving organizational cyber resilience through proactive threat anticipation, strong security measures, and efficient recovery. As they emphasize:

> "Everyone will be attacked someday, what matters is how fast they recover!"

This reflects their focus on minimizing the impact of cyber threats and enabling organizations to maintain operational continuity (ArtResilia n.d.).

ArtResilia offers a broad range of cybersecurity services divided into three main areas:

- **Defensive Security:** Design, implementation, and management of security solutions. Their state-of-the-art Security Operations Center (SOC), known as Helix, provides end-to-end security operations, including threat anticipation, detection, protection, and incident recovery.
- **Offensive Security:** Comprehensive testing and validation of organizational assets by simulating real-world threat actor techniques, including penetration testing, threat intelligence, and deception-based strategies.
- **Advisory Services:** Consulting services focused on governance, risk management, compliance (GRC), and security architecture to ensure organizations meet regulatory and operational security requirements.

ArtResilia was approached for a research project exploring strategies to enhance cybersecurity resilience through automation and machine learning. The project aligns with ArtResilia's mission to strengthen incident detection and response capabilities against emerging cyber threats.

After reviewing the proposal, ArtResilia accepted the project and committed to providing support and resources for its implementation. This collaboration underscores ArtResilia's dedication to fostering cybersecurity innovation and ensuring real-world applications in professional security operations.

1.5 Research Questions

For this research, three key research questions (RQs) were devised to address the central issue of improving the efficiency and accuracy of security alert triage within Security Operations Centers (SOCs). Each question offers a targeted perspective for examining specific facets of the problem, contributing to the overarching objective of minimizing both false positives and false negatives in security alerts. Below are the RQs, along with a brief discussion of their significance.

1. **How can ML techniques be applied to the triage of security alerts to reduce false positives and negatives?**
   This question explores how machine learning (ML) can tackle the challenge of excessive alerts in SOCs, characterized by high false positive and negative rates. The goal is to identify effective ML techniques and assess their impact on improving alert accuracy.
2. **What are the main challenges in implementing an AI bot integrated with SIEM systems, and how can they be mitigated?**
   Implementing AI in security systems presents a range of challenges, from broader issues like data integration, system compatibility, and performance maintenance under varying conditions, to narrower challenges such as trusting the AI to serve as an advisor to the manual work of the SOC analyst. It's also crucial to ensure that the percentage of incorrect classifications does not jeopardize trust in the solution. This research question aims to identify these challenges and propose strategies for successful AI integration.
3. **What ML frameworks and methodologies exist, and which are most suitable for developing and training AI models for ticket triage in SOC environments?**
   Understanding ML frameworks and methodologies is essential for choosing the right tools for AI model development in SOC ticket triage. This question examines the suitability of different frameworks based on their capabilities and alignment with SOC needs.

To answer these research questions systematically and comprehensively, a structured approach was adopted. This approach involved defining keywords, utilizing relevant digital libraries, and employing robust frameworks for the literature review.

The following keywords and their combinations were used to identify relevant papers:

- **Keywords:** Machine learning, AI, security alerts, SOCs, SIEM systems
- **Keywords:** False positives, false negatives, alert triage, alert fatigue
- **Keywords:** AI integration, ticket prioritization, frameworks, methodologies

The primary digital libraries used were:

- **ACM Digital Library**: For accessing peer-reviewed articles on machine learning and security applications.

- **ResearchGate**: For exploring academic discussions, preprints, and supplementary materials.

Search strings using these keywords were used to search the digital libraries effectively.

- **Search String 1:** ("Machine learning" AND "security alerts") AND ("false positives" OR "false negatives")
- **Search String 2:** ("AI integration" AND "SIEM systems") OR ("ticket prioritization")
- **Search String 3:** ("Frameworks" AND "SOC environments") AND ("alert triage")

In addition to the search strings, a new constraint was implemented to limit the findings to the most recent five years, ensuring the collection of up-to-date research and information.

To structure my literature review, two complementary frameworks were used: **PICOCS** and **snowballing**.

- **PICOCS Framework**: This framework helped define the Population, Intervention, Comparison, Outcome, Context, and Study type for each RQ as can be seen on Figure 1.1. Using PICOCS ensured a focused and systematic approach to identifying papers aligned with this research objectives.

Table 1.1: PICOCS Framework for Research Questions

Element	Definition	RQ1	RQ2	RQ3
P	Population	SOC analysts dealing with alerts	SOC analysts and SIEM administrators	SOC analysts developing AI models
I	Intervention	Machine learning techniques	AI bots integrated with SIEM systems	ML frameworks and methodologies
C	Comparison	Manual triage methods	Existing rule-based systems	Alternative ML frameworks
O	Outcome	Reduction in false positives/negatives	Mitigation of integration challenges	Identification of suitable frameworks
C	Context	SOC environments	SOC environments using SIEM systems	SOC environments implementing AI models
S	Study Type	Empirical studies, experiments	Case studies, technical reports	Framework evaluations, experiments

- **Snowballing Method**: Starting with an initial set of highly relevant papers, I used snowballing to identify additional studies by exploring their references (backward snowballing) and citations (forward snowballing). This iterative process expanded the breadth of my review.

Combining PICOCS and Snowballing The combination of PICOCS and snowballing provided a balance between structure and adaptability. PICOCS ensured a methodical and targeted search, while snowballing allowed for iterative exploration beyond initial search results, capturing emerging trends and overlooked studies.

By following this approach, it was possible to systematically answered the RQs and developed insights that directly contribute to improving the triage of security alerts in SOC environments.

1.6 Dissertation Structure

This dissertation is organized into five chapters, each addressing a key component of the research process and contributing to the overall objective of developing an automated security alert triage system for SOC.

- **Chapter 1 - Introduction**: Presents the context, motivation, and objectives of the research. It introduces the problem statement, research questions, and the organizational environment where the study was conducted.
- **Chapter 2 - State of the Art**: Provides a comprehensive literature review covering the foundations of Security Operations Centers, Security Information and Event Management (SIEM) systems, Machine Learning techniques, and existing approaches to automated alert classification. It also includes a comparative analysis of relevant technologies.
- **Chapter 3 - Method and Implementation**: Describes the methodological approach adopted, including the technological choices, the proposed solutions, and the architectural design. It details the implementation steps, dataset preparation, model development, and system integration.
- **Chapter 4 - Model Evaluation and Results Analysis**: Presents the experimental evaluation of the developed system, discussing both local and live testing environments, performance metrics, and a detailed analysis of the obtained results.
- **Chapter 5 - Conclusion and Future Work**: Summarizes the main findings of the research, reflects on its contributions and limitations, and proposes potential directions for future work to enhance and extend the developed solution.

1.7 Ethical Considerations

During the writing of this dissertation, AI-powered features of Grammarly were used to support the revision and improvement of the text. This included grammar correction, clarity suggestions, and occasional rephrasing to enhance readability and coherence.

No content generated by AI contributed to the research results, technical implementation, or core arguments of this work. The use of AI was strictly limited to language polishing and did not influence the academic or scientific content in any way.

Chapter 2

State of the Art

This chapter is divided into three parts. Part I starts on the theoretical side by introducing Machine Learning, Security Operation Centers, Security Information and Event Managers, and response mechanisms such as Intrusion and Prevension System (IPS), Intrusion and Detection System (IDS), and Endpoint Detection and Response (EDR). It also explains how popular ML classification models work and how to evaluate them. Understanding these topics is the groundwork for material considered later in the chapter.

The second part focuses on the automatic classification of security alert tickets. It begins with an overview of proposed solutions and then reviews several publicly available studies. The chapter concludes with a detailed evaluation of existing systems that are similar to the one presented in this thesis.

The third and final part discusses existing technologies that SOC may utilize. These technologies are designed to support the workflow outlined in the first section about SOC. At least three different technologies will be presented and discussed for each aspect of the SOC workflow. The section will conclude with a summary of the key differences among the technologies and, if relevant, evaluate which one may be superior.

2.1 Theoretical Introduction

Before diving into the technologies, tools, and methodologies used for automating the classification of security alert tickets, it is essential to understand some fundamental concepts. Concepts like, SOCs, Security Information and Event Management (SIEM) systems, Artificial Intelligence (AI), particularly ML, and advanced threat detection and response mechanisms. Such a foundation is indispensable for comprehending these systems' functions, construction work, advantages and limitations, and the actual performance comparison among different methods. The chapter introduces SOC and how it functions, then moves on to the foundations of ML and provides an overview of models frequently used for classification problems. It provides an overview of methods for assessing the performance of classification models, placing this in context for their use within the project.

2.1.1 Security Operations Center

The complexity and frequency of cyber threats are increasing (Arianna 2024), which has led to the emergence of SOCs as a critical component of modern IT enterprises. SOCs are the primary defenders in incident response planning, vulnerability management, and regulatory compliance. In today's interconnected world, integrating security operations to reduce

defensive barriers allows organizations to optimize resources, enhance security posture, and safeguard critical assets.

A SOC is a unit (Rutledge 2024) that provides tailored and centralized Computer Network Defense (CND) (Zimmerman 2014). It defends computer networks against the growing world of cyber threats. The main objective of a SOC is to ensure continuous monitoring and incident response for enterprise systems (Zimmerman 2014). This primarily focuses on preventing unauthorized access, cyber-attacks, and data breaches.

This twenty-four seven (24/7) facility leverages advanced technologies and skilled information security professionals to monitor the network continuously (Zimmerman 2014). With sophisticated tools to detect anomalies, a SOC can address threats before they escalate.

Definition and Characteristics of a SOC

Zimmerman (2014) defined a SOC as:

> "A team primarily composed of security analysts organized to detect, analyze, respond to, report on, and prevent cybersecurity incidents."

This definition integrates elements from various sources, including the historical definition of Computer Security Incident Response Team (CSIRT) as detailed in references (Shirey 2007) and (Brownlee and Guttman 1998).

For an organization to qualify as a SOC, according to Zimmerman (2014), it must:

1. Establish a system for constituents to report cybersecurity incidents.
2. Provide comprehensive support for managing and resolving incidents effectively.
3. Convey incident-related information to internal and external stakeholders.

Key Responsibilities of a SOC

A SOC has several critical missions, as outlined by various sources (Muniz, McIntyre, and AlFardan 2015; Zimmerman 2014):

- Preventing cybersecurity incidents by implementing proactive measures such as vulnerability scanning and threat analysis.
- Monitor, detect, and analyze potential security intrusions.
- Handle confirmed incidents and coordinate resources for effective countermeasures.
- Providing stakeholders with situational awareness regarding cybersecurity incidents, trends, and threats.

Tiers of Operation

Analysts within a SOC operate in tiers (Vielberth et al. 2020). Tier 1 analysts monitor and conduct initial investigations, escalating complex cases to Tier 2 analysts, who perform in-depth analyses and take further actions like blocking activities or deactivating accounts. Generally, higher-tier analysts handle more complex incidents, which require more time to resolve.

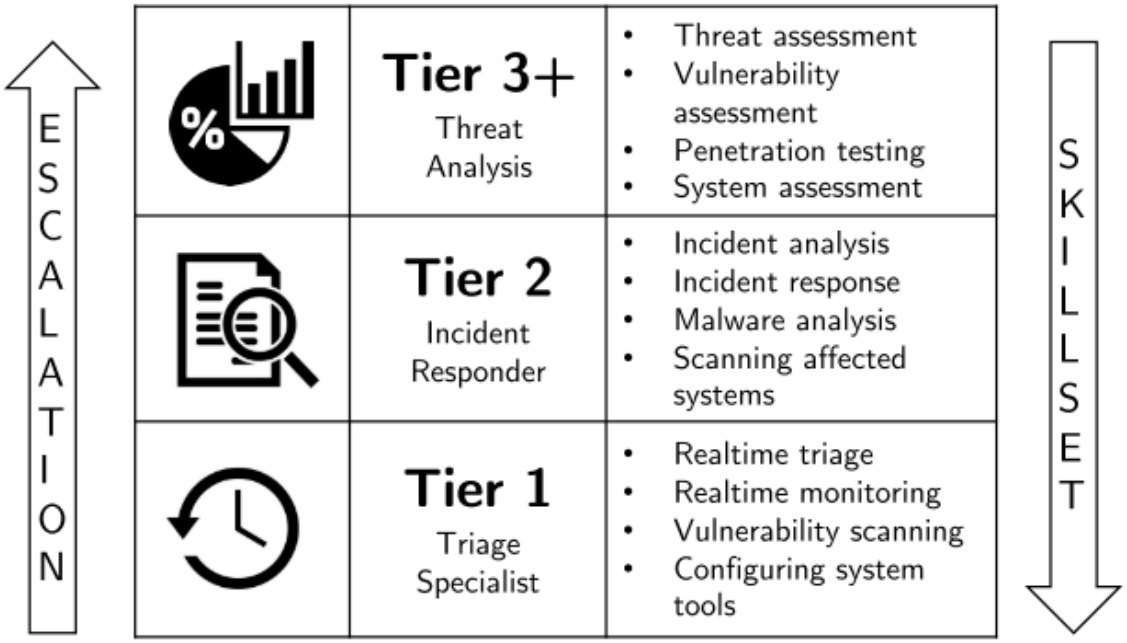

Figure 2.1: SOC Analyst Tier Responsibilities, from (Kokulu et al. 2019).

Extra levels may exist that handle responsibilities like threat hunting, vulnerability assessments, and penetration testing. These levels are collectively termed Tier 3+. Figure 2.1 was taken from a paper on a qualitative study on a SOC (Kokulu et al. 2019) and illustrates a visual representation of these tiers and their associated tasks. Not all SOCs follow a hierarchical model. In some collaborative frameworks, team members may possess comparable skill sets, enabling them to manage incidents independently (Kokulu et al. 2019).

Triage Specialist

Since this project focuses on automating the triage process, it is crucial to gain a detailed understanding of the role of a triage specialist, their functions, and the advantages of automated triage in comparison.

Tier 1 analysts, also known as triage specialists, play a critical role in the initial stages of a SOC workflow (Vielberth et al. 2020).

As stated by Vielberth et al. (2020), a Tier 1 analyst's primary responsibilities include:

1. Collecting and analyzing raw data.
2. Reviewing alarms and alerts generated by monitoring systems.
3. Determining the validity and criticality of each alert.

They must improve alerts with additional contextual information and decide whether an alert represents a real threat or a false positive (Hámornik and Krasznay 2018; Sundaramurthy et al. 2014). This process demands meticulous attention to detail, as the triage specialist must assess individual alerts, notify potential high-risk events, and prioritize them according to their severity (Tao 2018).

The repetitive nature of triage work, coupled with the need to escalate unresolved issues to Tier 2 analysts, can result in mental fatigue and burnout (Iamnitchi et al. 2017; Tines 2023).

This exhaustion affects individual performance and can compromise the overall efficiency of the SOC, as delayed or missed alerts may result in critical threats going undetected (CriticalStart 2019).

In a 2018 study (Crowley and Pescatore 2018), 53% of respondents in a security survey identified inadequate automation as the most common shortcoming.

This study demonstrates that effectively structured and implemented automation can help mitigate some or many of a SOC's weaknesses, particularly in the repetitive aspects of the SOC's workflow, such as the triage process.

2.1.2 Security Information and Event Management

The increasing complexity of cybersecurity threats has compelled organizations to implement advanced technologies to protect their digital assets. SIEM systems have emerged as vital tools in this context (Shaw 2022).

SIEM systems gather and centralize security-related data to detect threats and respond to incidents effectively. These systems connect logs from various sources to support security analytics, enabling real-time monitoring and retrospective analysis of past events (Shaw 2022). They integrate with cyber threat intelligence platforms, providing human analysts with advanced visual tools for seamless information sharing between organizations. Additionally, they retain event data over extended periods, ensuring robust log management capabilities.

The key features of a SIEM system, as gathered from various published sources (Ali, Shah, and ElAffendi 2024; Harper et al. 2010; Sheeraz et al. 2023), are:

- **Log Collection:** SIEM gathers log data from various network devices such as servers, firewalls, and switches. Data can be collected using two methods:
    1. **Agent-based collection:** An intermediary agent collects and forwards logs.
    2. **Agent-less collection:** Servers retrieve logs directly from the source devices.
- **Log Aggregation:** Collected logs are analyzed and structured for meaningful insights. Aggregation methods include:
    1. **Push method:** Devices actively send logs to the SIEM.
    2. **Pull method:** SIEM retrieves logs as needed.
- **Parsing and Normalization:** Parsing converts raw logs into structured data, while normalization standardizes logs from diverse sources to eliminate redundancy.
- **Threat Analysis and Detection:** By correlating log data with known threat indicators, SIEM systems identify malicious activities. Statistical methods and predefined rules enhance their ability to detect sophisticated threats.
- **Response Automation:** SIEM systems issue real-time alerts and notifications, enabling rapid responses to potential incidents.
- **Reporting and Visualization:** Advanced reporting tools provide security analysts with actionable insights, enabling detailed investigations and trend analysis.

Architectural Components

The architectural components of a security information and Event Management (SIEM) system consist of several essential elements that enable effective security monitoring, incident detection, and response (Sheeraz et al. 2023).

Data sources provide the raw material for analysis and threat detection (Ali, Shah, and ElAffendi 2024). These include a wide variety of log-generating devices and applications:

- **Network devices**: Firewalls, routers, and switches.
- **Endpoint devices**: Workstations, servers, and mobile devices.
- **Applications**: Web servers, databases, and cloud platforms.

A variety of data sources is crucial for effective monitoring and threat detection.

Data collection is a vital step with two main approaches to consider:

- **Agent-based collection**: Agent-based collection uses proxy agents on endpoint devices for better control and flexibility in log collection, but it is costly and complex to manage.
- **Agent-less collection**: Agent-less collection allows devices to send data directly to the SIEM, simplifying deployment but may reduce efficiency in high-volume data environments.

The **SIEM processing engine** is one of the critical components (Sheeraz et al. 2023) responsible for:

- **Parsing**: The conversion of raw log data into a structured format used for analysis.
- **Normalization**: Ensure the log formats are standardized to facilitate easier comparisons.
- **Correlation**: Finding relationships between events to strengthen security posture or incident response.

Storage and rationalization are vital for ensuring scalability and compliance. SIEM systems must store logs for future analysis. This means there has to be enough storage and a logical way of organizing this data for scalability or compliance requirements like GDPR and HIPAA (Sheeraz et al. 2023). Effective storage solutions should scale dynamically to handle large datasets without compromising performance or compliance standards.

Visualization and reporting tools play a key role in making data accessible and actionable. Critical functions that significantly benefit from good data visualization and reporting tools are incident investigation, trend analysis, and compliance audits (Sheeraz et al. 2023). These features not only improve user experience but also aid in making data actionable and accessible. Organizations can use these tools to conduct incident investigations, identify patterns occurring over time, and monitor compliance (Sheeraz et al. 2023). This comprehensive approach enables the development of scalable systems that can effectively handle increasing data demands.

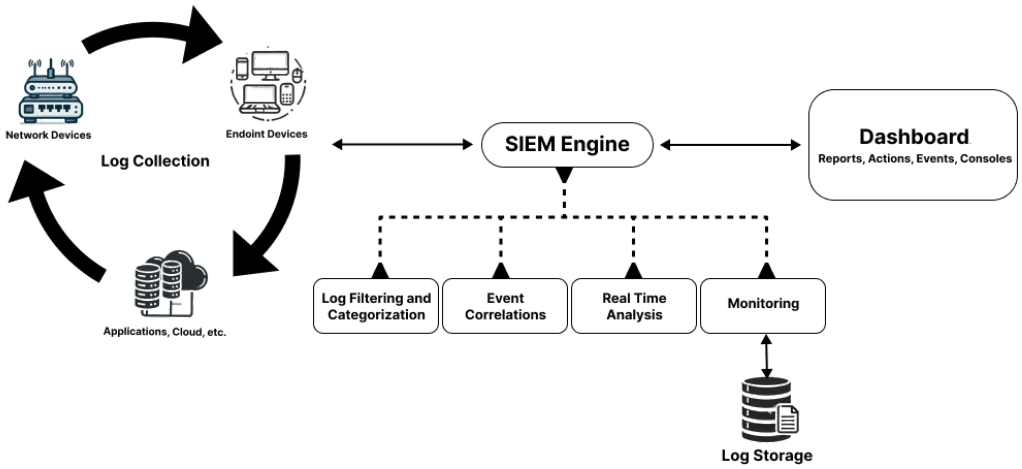

Figure 2.2: SIEM Architecture

Figure 2.2 illustrates the architecture of a SIEM system, highlighting its critical components and their interactions. Data flows from network and endpoint devices, as well as applications, to the log collection module. The SIEM processing engine then parses, normalizes, and correlates this data for real-time security alert analysis, event monitoring, and threat detection.

The diagram also emphasizes the importance of visualization and reporting tools for turning data into actionable insights.

The graphic illustrates how these elements integrate into a SIEM.

2.1.3 Machine Learning

ML is a core part of AI but also overlaps with data mining, statistics, probability, and mathematics (Mohri, Rostamizadeh, and Talwalkar 2012). Unlike traditional rule-based systems that rely on predefined logic, ML uses induction—it learns patterns from past data and forms assumptions that can be generalized to new cases (Ali, Shah, and ElAffendi 2024). This method relies on datasets, which are groups of examples the ML algorithm analyzes to find patterns (Mohri, Rostamizadeh, and Talwalkar 2012; Suthaharan 2016). The goal is to use these learned patterns to predict or describe new data.

There are three primary types of techniques employed in machine learning, (Mohri, Rostamizadeh, and Talwalkar 2012):

1. **Reinforcement Learning**: This is a subset of machine learning in which an agent learns by interacting with the environment (Moradi, Acker, and Denil 2023). It observes the environment, selects an action, and receives a reward if the action is beneficial or a penalty if it is detrimental. Over time, it refines its approach to achieve maximum rewards.

2. **Supervised Learning**: In this type of machine learning, models are trained on labeled data, meaning that each example includes input features and the corresponding expected output (or label). The model learns to map the inputs to the outputs, enabling it to predict the output for new, unseen data.

3. **Unsupervised Learning**: Unsupervised learning analyzes unlabeled data to reveal patterns without predefined labels. Unlike supervised learning, it allows algorithms to explore data independently. It is often used for clustering similar data points or modeling probability distributions. This approach is valuable for understanding the inherent organization in data without prior knowledge.

Supervised learning can be further categorized into several types, (Mohri, Rostamizadeh, and Talwalkar 2012):

- **Regression**: Regression algorithms predict numerical values within a continuous range by analyzing input data to identify patterns. They can forecast future values, such as calculating the next number in a sequence based on previous numbers and trends. Techniques like linear regression, polynomial regression, and others enable these algorithms to draw conclusions and make predictions effectively.

- **Similarity**: Similarity algorithms analyze and compare two distinct instances to measure their resemblance. They are vital in recommender systems for suggesting products based on user preferences and in visual identity tracking and verification by comparing images or features. Their versatility makes them essential in data analysis, security, and personalized user experiences.

- **Classification**: Classification algorithms classify the input data into predefined groups. Classification tasks can be binary when there are only two possible categories (such as a yes-no decision) or multiclass when the number of categories exceeds two, such as recognizing handwritten letters in the alphabet.

However, ML has its limitations. Since datasets are finite, no algorithm can predict every scenario, which highlights an essential aspect of inductive reasoning: it can suggest likely outcomes but cannot ensure certainty (Mohri, Rostamizadeh, and Talwalkar 2012; Suthaharan 2016).

In ML, achieving an optimal model involves balancing between two critical concepts: **bias** and **variance**. Bias refers to the error introduced when a model is excessively simplistic, which can lead to underfitting. Underfitting occurs when the model fails to capture the underlying patterns of the data, resulting in poor performance on both training and unseen datasets (ElSahly and Abdelfatah 2023). On the other hand, variance arises when a model becomes overly complex and sensitive to the fluctuations in the training data, culminating in overfitting. Overfitting means the model performs exceptionally well on the training dataset but poorly on new, unseen data because it has memorized the noise instead of learning the actual signal (ElSahly and Abdelfatah 2023).

The primary objective in constructing a machine learning model is to identify the appropriate level of complexity with the right balance, ensuring the model generalizes well and performs effectively on new data (Suthaharan 2016).

Despite these challenges, ML continues to evolve, with increasingly sophisticated algorithms enabling breakthroughs in many fields (Mohri, Rostamizadeh, and Talwalkar 2012).

2.1.4 Machine Learning Models

The processes and computations of training a machine learning model are structured within a framework. This chapter will briefly overview the most frequently used algorithms relevant to this project's goals.

Decision Trees

Decision Trees (DTs) are a fundamental supervised machine learning algorithm for classification and regression tasks (Huang 2024). They work by splitting data into subsets based on the value of input features, forming a tree-like structure composed of nodes and branches.

The process begins at the root node, representing the entire dataset, and iteratively divides the data into homogeneous subsets using decision nodes until a terminal leaf node is reached, representing the output prediction or class (Chauhan 2022), as seen in Figure 2.3.

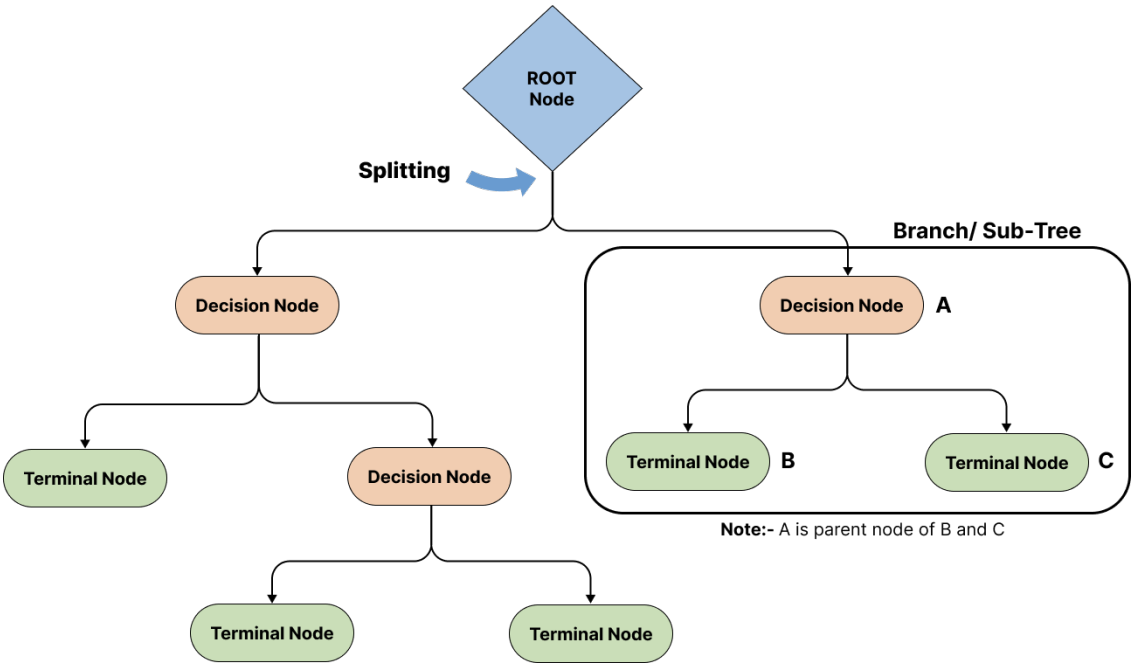

Figure 2.3: A basic structure of a Decision Tree, based on (Chauhan 2022)

The algorithm operates by evaluating all potential splits and selecting the one that optimizes a specific criterion, such as information gain or Gini impurity. Information gain measures the reduction in entropy after a split, while Gini impurity quantifies the likelihood of misclassification at a node.

The entropy of S is defined as:

$$H(S) = \sum_{i=1}^{n} -P(s_i) \times \log_b P(s_i) \tag{2.1}$$

where S is a set of values $s_1, s_2, \ldots, s_n$, $P(s_i)$ is the probability of observing a certain value, and b is the logarithm base, most commonly 2, e, or 10.

Using entropy, the **information gain (IG)** for a node t and a candidate split x is calculated as:

$$IG(t, x) = H(t) - H(x, t) \tag{2.2}$$

In contrast, the **Gini index**, another commonly used metric for evaluating splits, is defined as:

$$\text{Gini} = 1 - \sum_{i=1}^{n} P(s_i). \tag{2.3}$$

At each decision node, the algorithm tests a single feature and branches according to its value, guiding data instances down the tree until they reach a leaf node (Chauhan 2022). During training, the algorithm continues splitting until a stopping criterion is met, such as achieving a maximum tree depth, minimum node size, or no further improvement in the splitting metric (Chauhan 2022).

Pruning techniques prevent overfitting, which occurs when a tree becomes too complex and overly specific to the training data. These involve removing unnecessary branches or nodes to simplify the tree while maintaining predictive accuracy. Pruning can be preemptive (stopping tree growth early) or post hoc (removing branches after the tree is fully grown). This ensures that the decision tree generalizes well to unseen data (Huang 2024).

Decision trees are non-parametric, meaning they do not assume any specific distribution for the data, and they can capture both linear and non-linear relationships (Huang 2024). However, they are sensitive to slight variations in the dataset (Huang 2024), which may cause significant changes in the tree structure. Despite this, they remain popular for their simplicity, interpretability, and ability to handle numerical and categorical data (Chauhan 2022).

Ensemble classifiers

Ensemble learning is a powerful machine learning technique that combines the predictions of multiple base models to achieve superior performance compared to individual learners (Joseph 2022).

The fundamental idea is to address the limitations of single models, such as high variance, high bias, or low accuracy, by leveraging the diversity and complementary strengths of multiple models (Dasarathy and Sheela 1979; Hansen and Salamon 1990; Mienye and Sun 2022).

The concept of ensemble learning has evolved significantly since its inception. Early work by Dasarathy and Sheela (1979) introduced the partitioning of feature spaces using multiple classifiers. Later, Hansen and Salamon (1990) demonstrated that ensembles of artificial neural networks achieved superior predictive performance compared to single networks. Schapire (1990) groundbreaking work laid the foundation for boosting, one of the primary techniques in ensemble learning.

Ensemble learning methods are evaluated on two main principles: **accuracy** and **diversity** of the base learners (Hansen and Salamon 1990; Mienye and Sun 2022).

- **Accuracy**: Each base learner should perform better than random guessing on the given task (Li, Yu, and Zhou 2012).

- **Diversity**: The errors made by individual learners should be uncorrelated, which can be achieved through techniques like subsampling, feature randomization, or algorithmic diversity (Breiman 1996; Schapire 1990).

A model is accurate if it generalizes well on unseen instances, while diversity ensures that the errors of individual base models are not correlated. Achieving this balance is crucial for effective ensembles.

Ensemble Classifiers offer several advantages over single models, making them a popular choice in diverse applications of machine learning:

- **Improved Generalization**: Ensembles reduce overfitting and improve generalization by aggregating the predictions of multiple learners (Hansen and Salamon 1990).
- **Bias-Variance Trade-off**: Techniques like bagging reduce variance while boosting minimizes bias, addressing the critical limitations of individual learners (Li, Yu, and Zhou 2012; Mienye and Sun 2022).
- **Adaptability**: Ensembles can be designed to work with homogeneous (same algorithm) or heterogeneous (different algorithms) base learners, making them versatile across domains (Mienye and Sun 2022).

Ensemble learning methods are broadly categorized into:

- **Parallel Ensembles**: Base learners are trained independently on different subsets of data or features. Techniques like bagging and Random Forests fall into this category (Breiman 1996). The illustration in Figure 2.4 demonstrates how parallel ensembles operate.
- **Sequential Ensembles**: Models are trained iteratively, with each learner focusing on correcting the errors of its predecessor. Boosting is the most prominent example of this approach (Schapire 1990). The illustration in Figure 2.5 demonstrates how sequential ensembles operate.
- **Stacked Ensembles**: Predictions from multiple base models are combined using a meta-model trained on the outputs of the base learners (Mienye and Sun 2022).

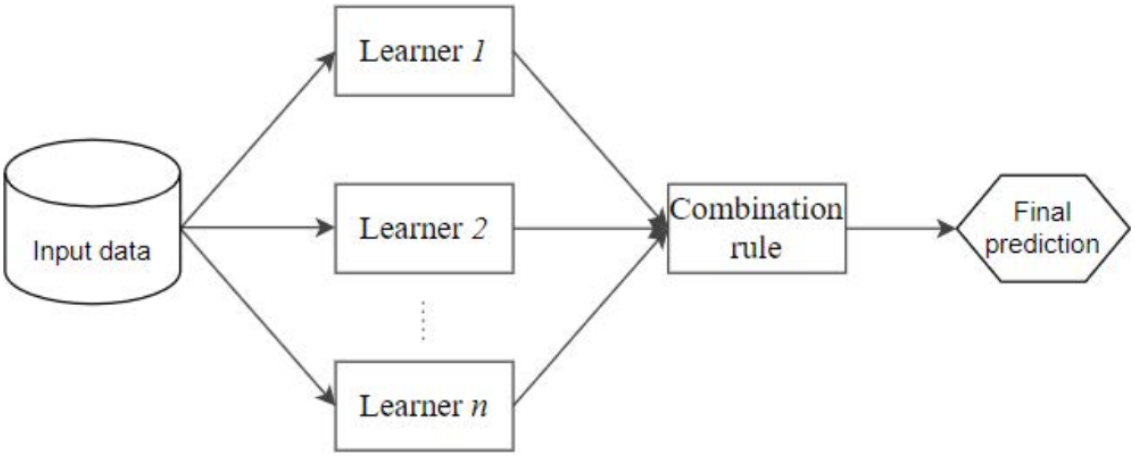

Figure 2.4: Diagram of Parallel Ensemble Learning from (Mienye and Sun 2022)

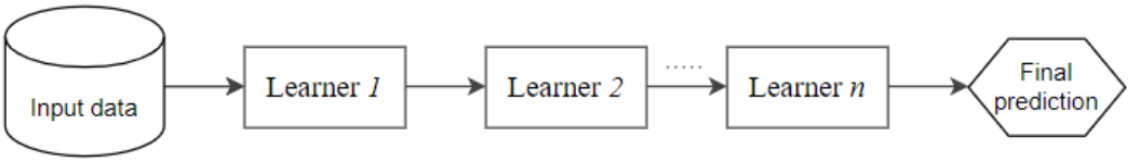

Figure 2.5: Diagram of Sequential Ensemble Learning. from (Mienye and Sun 2022)

Ensemble methods can then be categorized into three main approaches based on how they train and combine base models: **Bagging**, **Boosting**, and **Stacking**. Each method has a distinct mechanism for generating diversity and aggregating predictions, addressing the specific limitations of single models.

Bagging Bagging, which stands for bootstrap aggregating, is a parallel ensemble technique where multiple base models are independently trained on random subsets of the training data, referred to as bootstrapped samples. The final predictions are made by aggregating the outputs of these models, often using majority voting or averaging. Random Forests are a well-known implementation of bagging, recognized for their ability to reduce variance without increasing bias (Mienye and Sun 2022).

Boosting Boosting is a sequential ensemble method that aims to reduce bias by iteratively training base models. Each subsequent model focuses on correcting the errors made by the previous one, assigning higher weights to misclassified samples. Algorithms such as AdaBoost and Gradient Boosting exemplify this approach, often achieving superior performance on complex datasets, but this can lead to increased susceptibility to overfitting (Mienye and Sun 2022).

Stacking Stacking integrates predictions from multiple base models using a meta-model that learns the optimal way to combine their outputs. Unlike bagging and boosting, stacking allows for heterogeneous base learners, providing greater flexibility and diversity. The meta-model is typically trained on the predictions made by the base models, enabling it to make more accurate decisions (Mienye and Sun 2022).

Random Forests

Random Forests represent a significant advancement in machine learning, particularly in the domain of ensemble classifiers (Ali, Shah, and ElAffendi 2024).

As a bagging-based ensemble method, Random Forests combine multiple decision trees to improve classification accuracy and robustness. This approach leverages the strengths of individual trees while mitigating their limitations, such as overfitting, by aggregating their predictions (Ali, Shah, and ElAffendi 2024).

The Random Forest algorithm begins by generating multiple decision trees, as can be seen on Figure 2.6, each trained on a random subset of the training data through bootstrapping[1].

[1]Bootstrapping is a resampling technique where random samples are drawn with replacement from the dataset.

Additionally, features are randomly sampled at each split to ensure diversity among the trees, a process that reduces the correlation between individual tree predictions (Farooq and Otaibi 2018). Once trained, the predictions of all trees are aggregated, typically through majority voting for classification tasks or averaging for regression tasks, to produce the final output (Nila, Apostol, and Patriciu 2020).

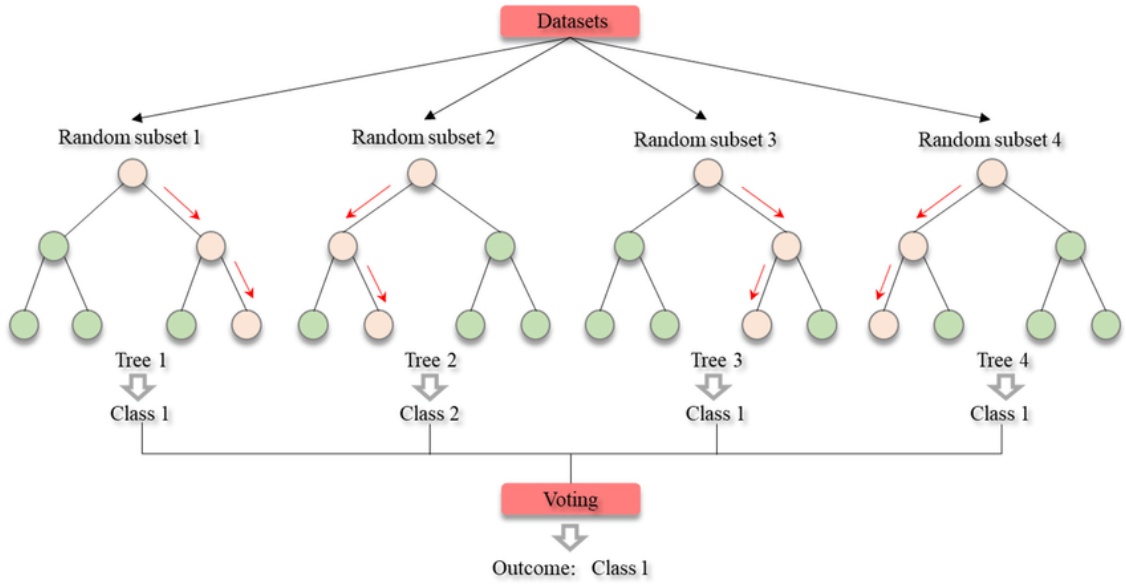

Figure 2.6: Workflow of Random Forests: Bootstrapping, Decision Trees, and Aggregated Predictions from (Yang et al. 2019)

Random Forests are highly valued for their versatility and robustness. They perform exceptionally well with large and high-dimensional datasets, effectively handling numerical and categorical data. Furthermore, their inherent ability to measure feature importance makes them interpretable, a critical requirement in security domains where trust and explanation are essential (Ali, Shah, and ElAffendi 2024).

Random Forests also excel at reducing overfitting, a common issue with individual decision trees. By averaging the predictions of multiple trees, the model achieves better generalization, even when faced with noisy data (Sopan et al. 2019).

Random Forests have been widely adopted in cybersecurity applications, particularly for detecting anomalies, classifying alerts, and predicting the likelihood of malicious behavior. For example, they are effectively used in SIEM systems to process large volumes of log data, identify patterns, and reduce false positives (Ali, Shah, and ElAffendi 2024; Farooq and Otaibi 2018).

Studies have shown that Random Forest-based models can achieve high accuracy in detecting advanced persistent threats (APTs) and other cyberattacks by leveraging their ability to handle complex feature interactions and high-dimensional data (Ali, Shah, and ElAffendi 2024). For instance, combining Random Forests with additional modules for feature selection and preprocessing has improved detection rates and reduced manual intervention in incident response workflows (Nila, Apostol, and Patriciu 2020).

Naive Bayes

The Naive Bayes algorithm is a probabilistic classifier based on Bayes' theorem, which assumes conditional independence among features given the class (Chandra, Challa, and Pasupuleti 2016). This simplicity makes it computationally efficient and easy to implement.

Naive Bayes is particularly effective in scenarios where the assumption of feature independence is approximately attained, such as spam detection and document classification. The model's reliance on this assumption allows it to scale effectively to high-dimensional datasets while remaining computationally efficient (Chandra, Challa, and Pasupuleti 2016).

Naive Bayes calculates the posterior probability of each class given the observed data using the formula:

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{2.4}$$

where:

- $P(C|X)$ is the posterior probability of class C given the feature vector X,
- $P(X|C)$ is the likelihood of observing X given class C,
- $P(C)$ is the prior probability of class C,
- $P(X)$ is the probability of the feature vector X.

In practice, the Naive Bayes model, Figure 2.7, estimates the prior probability $P(C)$ for each class and the conditional probability $P(X_i|C)$ for each feature X_i given the class. These probabilities are typically derived from the frequency of occurrences in the training data. For continuous features, it is common to assume a Gaussian distribution and compute the likelihood accordingly. The model assigns a new instance to the class with the highest posterior probability $P(C|X)$ (Chandra, Challa, and Pasupuleti 2016; Nila, Apostol, and Patriciu 2020).

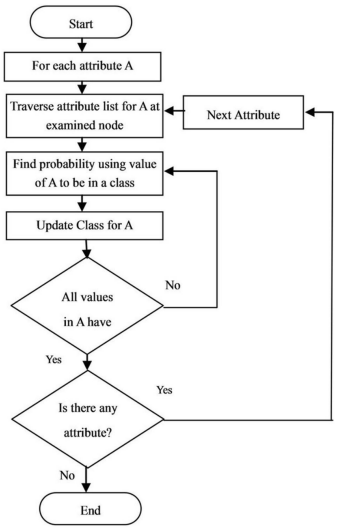

Figure 2.7: Schematic Diagram of the Naive Bayes Classification Process from (Sneha and Gangil 2019)

This decision rule is computationally efficient, making Naive Bayes suitable for real-time prediction tasks (Chandra, Challa, and Pasupuleti 2016; Nila, Apostol, and Patriciu 2020).

However, the strong independence assumption may only hold in some scenarios, potentially leading to suboptimal performance when features are highly correlated. Despite this limitation, Naive Bayes remains popular due to its simplicity and effectiveness in many practical applications (Chandra, Challa, and Pasupuleti 2016).

2.2 Automatic Classification of Security Alerts in SOC

The growing complexity of cyber threats, especially Advanced Persistent Threats (APT), has led to significant advancements in automated detection and mitigation mechanisms. Many of these mechanisms utilize ML techniques. This section examines key works, focusing on their methodologies, limitations, and relevance to the problem of alert triage automation in SOC.

Recent research has explored hybrid approaches to anomaly detection. Saini et al. (2023) proposed a hybrid ensemble model combining Random Forest and XGBoost classifiers, achieving a 99.91% accuracy on the CIC-IDS2017[2] dataset with a False Positive Rate (FPR) as low as 0.12%. Despite its high performance, the model relies on static datasets, which limits its adaptability to evolving attack patterns. Similarly, Ghafir et al. (2018) developed Machine Learning for Advanced Persistent Threats (MLAPT), a multi-phase system incorporating correlation frameworks for APT detection, achieving an accuracy of 84.8%. While this approach improved early-stage prediction, its accuracy was limited compared to ensemble methods and lacked scalability for dynamic environments.

Ali, Shah, and ElAffendi (2024) extended APT datasets to include a diverse range of alerts. They integrated Random Forest and XGBoost models into a SIEM environment, demonstrating the ability to reduce FPR and achieve a near-perfect 99.6% accuracy. However, the computational demands of such systems may hinder practical deployment in resource-constrained environments. On the contrary, Brogi and Tong (2016) utilized Information Flow Tracking (IFT) for APT detection, concentrating on the correlation between the various stages of an attack. While this method excels at tracing complex multi-stage attacks, it requires further automation to support large-scale SOC operations effectively.

Incorporating reinforcement learning, Sethi et al. (2020) introduced a Deep Q-Network (DQN)-based context-adaptive intrusion detection system, which improved FPR and demonstrated robustness against several attacks. Nevertheless, the distributed nature of their approach introduces practical deployment challenges, especially in SOC environments where centralized oversight is critical. Similarly, Nila, Apostol, and Patriciu (2020) explored ML-based triage systems to automate alert classification and reduce false positives. Their work effectively alleviated analyst fatigue by prioritizing actionable alerts, yet the scope of implementation remains limited to standalone systems.

Chandra, Challa, and Pasupuleti (2016) conducted a comprehensive study on email-based APT entry points, explicitly exploring the effectiveness of Bayesian spam filters[3]. These filters demonstrated a strong capability to identify and differentiate between spear-phishing attempts and general spam emails, which is crucial for enhancing cybersecurity measures.

[2] Labeled dataset of network traffic, including normal behavior and various attacks, used to evaluate intrusion detection models.

[3] Bayesian spam filters classify emails as spam or legitimate using probabilities based on Bayes theorem.

However, the authors noted that while their solution provided significant benefits in detecting these specific types of email threats, it needed to address the broader range of APT life cycles. This limitation restricts its effectiveness in SOC environments with diverse attack vectors, requiring a more holistic approach to threat detection and response strategies to encompass the full spectrum of APT strategies and tactics.

Table 2.1 presents a detailed summary of the relevant studies explored, highlighting their respective strengths, weaknesses, and potential areas for enhancement.

Table 2.1: Summary of the Existing Related Works based on Ali, Shah, and ElAffendi (2024).

References	APT Life Cycle Coverage	AI Models Used	Lowest FPR	Highest Accuracy	Limitations
Ghafir et al. (2018)	Complete	Decision Tree, SVM, KNN, Ensemble Classifier	4.5%	84.8%	The blacklist-based detection modules require continuous update. Moreover, the accuracy is lower.
Brogi and Tong (2016)	Complete	No ML model used	High	Not calculated	Information Flow Tracking is used to detect APTs; however, the FPR is high.
Giura and Wang (2012)	Complete	No ML model used	27.88%	Not calculated	Sufficient knowledge is required to set up the mechanism.
Sopan et al. (2019)	No	Random Forest	Not calculated	98.5%	The post-alert decision is not automated; it involves the intervention of security experts.
Farooq and Otaibi (2018)	Partial	SVM, Random Forest	Not calculated	Not calculated	The process anomaly detection is presented formally using One-Class SVM. However, the paper lacks ML-based experimental results.
Sethi et al. (2020)	No	Deep Q-network	0.35%	96.12%	The proposed model enables the detection of APTs.
Nila, Apostol, and Patriciu (2020)	No	ZeroR, OneR, NaiveBayes, SVM, J48, RandomForest	Not calculated	95.5%	The authors did not discuss the FPR of the proposed model.
Chandra, Challa, and Pasupuleti (2016)	Partial	NaiveBayes	Not calculated	87%	The authors did not discuss the FPR of the proposed model.
Saini et al. (2023)	Partial	Random Forest, XGBoost	0.12%	99.91%	This study used existing datasets, posing a challenge because these datasets might not contain the latest attack scenarios.
Ali, Shah, and ElAffendi (2024)	Complete	Random Forest, XGBoost	Not calculated%	99.6%	High resource demand for training; dataset generated in controlled environments, limiting real-world adaptability.

This study introduces a new approach to automating alert triage in SOCs by combining ML models with existing SIEM systems. In contrast with the previous studies, this research will use live data as its training data and will be deployed in a real-world environment. Therefore, analysts can focus on high-risk issues by analyzing the alerts based on the severity of the cyber threat shown by the ML model.

While the related works reviewed in this study showcase significant advancements in SOC automation and alert triage, none closely reflects this research's objectives. However, beyond academic research, commercially available solutions like Check Point's SOC automation tools demonstrate functionalities that align with aspects of this study. These solutions offer valuable insights into the application of AI and automation in SOC environments, providing a practical point of comparison for the approach proposed in this research.

Check Point's approach to SOC automation stands out for its focus on streamlining SOC operations using artificial intelligence (AI) and automation tools (CheckPoint n.d.).

SOC automation, as described by Check Point, employs AI to take over repetitive and manual tasks in the SOC, such as alert triage, incident response, and threat detection. Their solution includes tools like:

- **Generative AI:** Used to improve usability and streamline workflows, enabling analysts to query and interact with data using natural language.
- **Playbooks:** Predefined and customizable workflows for incident response that enable rapid and consistent remediation.
- **Threat Detection and Analytics:** Leveraging advanced analytics, behavioral insights, and proprietary threat intelligence from Check Point Research and ThreatCloud AI to identify threats and reduce false positives.
- **Malware Analysis and Phishing Detection:** Using sandboxes and natural language processing (NLP) to analyze suspicious files and emails.
- **Infinity Extended Prevention and Response (XDR/XPR):** A platform that integrates automated playbooks with real-time threat intelligence and analytics to correlate events across the security estate, detect sophisticated attacks, and prevent lateral movement.

The research in this study aligns closely with Check Point's SOC automation approach. Both aim to automate alert triage and streamline SOC operations using AI to enhance efficiency. This study also shares key goals, such as reducing false positives, prioritizing alerts, and providing analysts with tools to focus on high-risk threats. Like Check Point, this research emphasizes the use of playbooks for consistent responses and aims to alleviate the burden of repetitive tasks on analysts.

While the Check Point solution is comprehensive and includes a full suite of tools, this research focuses on a customized solution tailored to the specific needs of the organization conducting this study differing in:

1. **Integration with Existing Infrastructure:** This study develops a solution that seamlessly integrates with the organization's current SIEM system, avoiding the need for a complete overhaul of existing systems, as would be required to adopt Check Point's platform.

2. **Live Data and Real-World Deployment:** Unlike Check Point's reliance on proprietary threat intelligence and prebuilt systems, this research involves using live data collected within the organization's environment, ensuring the model is trained and deployed in a real-world context.

2.3 Technologies

The following section presents some of the important technologies available in the three categories discussed: SIEM Tools, Ticketing Tools, and Machine Learning Frameworks. The company utilizes QRadar as its SIEM solution and IBM QRadar SOAR and JIRA for ticketing and incident management. The subsequent subsections provide an overview of the tools in each category, followed by a comparative analysis highlighting their strengths and trade-offs.

The following section presents some of the important technologies available in the three categories discussed: SIEM Tools, Ticketing Tools, and Machine Learning Frameworks. Each category serves a unique purpose in the automatic classification of security alerts in a SOC. The tools used for this project were the same as those the company utilizes, including QRadar as its SIEM solution, IBM QRadar SOAR, and JIRA for ticketing and incident management. The subsequent subsections provide an overview of existing tools in each category, followed by a comparative analysis highlighting their strengths and trade-offs.

2.3.1 SIEM Tools

In this subsection, the three leading SIEM tools, QRadar, Splunk, and LogRhythm (exabeam 2024), were chosen for their advanced threat detection, log aggregation, and real-time security analytics. Based on industry adoption, integration capabilities, and performance in SOC environments, each platform offers unique features for efficient incident detection and response.

QRadar

QRadar is a SIEM solution initially developed by IBM and later acquired by Palo Alto Networks (Alto 2024). It excels in advanced threat detection, log aggregation, and automated incident response by collecting and correlating data from various sources like firewalls and network devices. Its features, including anomaly detection and event prioritization, help analysts focus on critical threats, reducing false positives.

A key advantage of QRadar is its integration with other IBM and Palo Alto tools, facilitating seamless workflows. Its machine learning support enables organizations to detect emerging threats, while its modular architecture ensures scalability for businesses of all sizes. Recent updates enhance QRadar's capabilities for modern hybrid and cloud-native infrastructures.

Splunk

Splunk is a powerful and versatile SIEM platform known for its real-time monitoring and analytics. It enables organizations to detect and respond to cybersecurity threats quickly. It excels at processing large volumes of machine-generated data, making it essential for security teams in complex IT environments.

With a broad ecosystem of applications, Splunk supports various use cases such as threat hunting, compliance management, and anomaly detection. Splunk Enterprise Security (ES)'s add-on offers tailored solutions for SOC environments, including prebuilt dashboards and customizable alerts.

Splunk's user-friendly interface and robust reporting tools cater to analysts of all skill levels, while its integration with machine learning enhances proactive threat management. Splunk's flexibility and capabilities make it a preferred choice for organizations seeking a reliable SIEM solution.

LogRhythm

LogRhythm is a robust SIEM platform aimed at enhancing threat detection and incident response. Its user-friendly design and integration capabilities help streamline SOC workflows. The platform utilizes advanced analytics, behavioral anomaly detection, and machine learning for effective threat identification.

LogRhythm also excels in compliance management with preconfigured templates for regulations like GDPR, HIPAA, and PCI-DSS. Its Threat Lifecycle Management (TLM) framework facilitates efficient threat detection and remediation with structured workflows.

A key highlight is its seamless integration with various third-party tools, along with a centralized dashboard that provides actionable insights for SOC teams. Additionally, LogRhythm offers flexible deployment options, making it suitable for diverse infrastructure needs.

2.3.2 Ticketing Tools

This subsection focuses on three ticketing tools: IBM QRadar SOAR, ServiceNow, and JIRA. These tools were selected for their capabilities in automating workflows, improving incident management, and integrating with SIEM systems. Practical ticketing tools are vital for efficiently managing security incidents, reducing response times, and promoting collaboration within SOC teams.

IBM QRadar SOAR

IBM QRadar SOAR enhances the QRadar SIEM platform by streamlining incident management and security operations. It automates workflows, minimizing manual tasks, and features playbook automation that allows analysts to execute consistent responses tailored to organizational policies. The comprehensive case management system centralizes tracking of incidents, evidence, and communications, fostering real-time collaboration. Additionally, it integrates with various third-party tools like EDR systems and cloud services, enabling SOC teams to focus on high-priority threats and improving overall efficiency in incident resolution (IBM n.d.).

ServiceNow

ServiceNow is a versatile platform originally designed for IT service management (ITSM) that has expanded to include strong incident management for cybersecurity. It is widely adopted across industries and integrates effectively with SIEM tools and other security systems, making it valuable for SOC teams.

The Security Incident Response (SIR) module offers a centralized approach for managing security incidents, featuring workflow automation for incident triage, escalation, and remediation. Its dashboard capabilities enable real-time monitoring and reporting, providing security managers with visibility into ongoing incidents.

A key strength of ServiceNow is its seamless integration with vulnerability scanners, endpoint protection platforms, and threat intelligence feeds, enhancing its ability to prioritize alerts and correlate events. Its scalable architecture makes it suitable for organizations of all sizes, from small businesses to global enterprises.

JIRA

JIRA is widely recognized as a project management tool and has been adapted to manage security incidents in SOC environments. Its user-friendly interface allows organizations to enhance incident management with minimal training requirements. JIRA's ticketing system helps analysts create, assign, and update tickets for security alerts, ensuring systematic documentation and resolution.

JIRA integrates effectively with SIEM tools and offers extensive customization options for workflows and notifications, further enhancing its functionality. Although it lacks advanced features like playbook-driven automation found in IBM QRadar SOAR, JIRA remains a practical and cost-effective solution for incident management, especially for organizations already familiar with its ecosystem.

2.3.3 Machine Learning Frameworks

This section discusses three popular machine learning frameworks: Scikit-learn, TensorFlow, and PyTorch (GeeksforGeeks 2024). These frameworks effectively implement scalable machine learning models in research and production.

Scikit-learn

Scikit-learn is a Python library known for its simplicity in implementing classical machine learning algorithms like Random Forest, Naive Bayes, and Support Vector Machines (SVM). It excels in classification, regression, and clustering tasks and offers comprehensive preprocessing tools for effective data cleaning and transformation. The library features strong cross-validation capabilities for evaluating model performance and integrates well with other Python libraries such as NumPy, pandas, and matplotlib. Scikit-learn is ideal for quick prototyping and development, making it a popular choice for both beginners and professionals, though it may not be as effective for deep learning or very large datasets.

TensorFlow

TensorFlow is an open-source framework developed by Google, designed for scalable machine learning and deep learning applications. It efficiently handles large-scale computations and supports various techniques, including traditional algorithms and advanced architectures like CNNs and RNNs. The Keras API within TensorFlow simplifies model building for users with limited experience.

One of its key strengths is deploying models across platforms like mobile, edge computing, and cloud environments. TensorBoard, a visualization tool, helps users monitor training

and performance, enabling effective algorithm fine-tuning. Despite a steeper learning curve compared to some alternatives, TensorFlow's extensive capabilities make it ideal for complex applications, including cybersecurity tasks like anomaly detection and threat prediction.

PyTorch

PyTorch is a flexible machine learning library that has gained popularity among researchers and developers due to its dynamic computation graph, allowing real-time changes during model execution. It excels in deep learning tasks like natural language processing, computer vision, and reinforcement learning, with the torch.nn module simplifying neural network creation and the autograd module providing efficient gradient computation. PyTorch's intuitive design makes it easy for new users and integrates well with libraries like NumPy and ONNX.

2.3.4 Comparative Analysis

A comparative overview of all the discussed technologies will be presented here. This analysis highlights the strengths and limitations of the SIEM tools, ticketing tools, and machine learning frameworks identified in previous subsections.

SIEM Tools

Table 2.2: Comparative Analysis of SIEM Tools.

Feature	QRadar	Splunk	LogRhythm
Integration	Deep IBM ecosystem integration	Extensive third-party support	Moderate third-party support
Threat Detection	Advanced anomaly detection	Real-time search and analytics	Strong ML-based detection
Compliance	Strong compliance capabilities	Flexible for custom compliance	Focused on compliance workflows
Machine Learning	Limited native ML	Third-party ML integrations	Native ML capabilities
Ease of Use	Moderate	User-friendly UI	Moderate

As shown in Table 2.2, QRadar stands out due to its deep integration with IBM solutions, making it an excellent choice for organizations already utilizing IBM's ecosystem. In contrast, Splunk offers powerful real-time analytics and extensive integration with third-party tools, which allows it to be adaptable for various use cases. LogRhythm focuses on compliance and features native machine learning capabilities, although it may not provide the same level of seamless integration within IBM environments as QRadar does.

Ticketing Tools

Table 2.3: Comparative Analysis of Ticketing Tools.

Feature	QRadar SOAR	ServiceNow	JIRA
Integration	Seamless with QRadar SIEM	Broad integrations across IT	Extensive integrations with development tools
Automation	Playbook-driven workflows	Customizable workflow automation	Moderate workflow automation
Incident Management	Case management and playbooks	Advanced IT service management	Simplified incident tracking
Ease of Use	Moderate	Customizable but complex	Highly user-friendly

Table 2.3 outlines the strengths and weaknesses of various ticketing tools. IBM QRadar SOAR closely integrates with the QRadar SIEM platform, providing strong automation through playbooks and advanced incident management features. ServiceNow is notable for its flexibility and capability to cater to multiple IT departments. In contrast, JIRA offers a more user-friendly and cost-effective solution, with moderate automation functionalities and extensive integration capabilities.

Machine Learning Frameworks

Table 2.4: Comparative Analysis of Machine Learning Frameworks.

Feature	Scikit-learn	TensorFlow	PyTorch
Algorithms Supported	Classical ML (Random Forest)	Deep Learning and Classical ML	Deep Learning and Research ML
Scalability	Moderate	High	High
Ease of Use	Beginner-friendly	Moderate complexity	High flexibility for research
Deployment	Less optimized for deployment	Production-ready	Suitable for research & testing
Flexibility	Limited to classical models	High for both models and tuning	High for experimentation

As outlined in Table 2.4, Scikit-learn is a user-friendly and effective library for implementing classical machine learning algorithms such as Random Forest and Naive Bayes. TensorFlow excels in scalability and is well-suited for production environments, making it ideal for deep learning applications. On the other hand, PyTorch is known for its flexibility and dynamic computation graphs, making it the preferred choice for research-focused machine learning and experimentation.

2.3.5 Conclusion

Based on the comparative analysis, QRadar, IBM QRadar SOAR, and Scikit-learn emerge as the most suitable technologies for this project due to their strong integration, automation capabilities, and alignment with the project goals. Importantly, these tools also align with the technologies available within the company where this project will be implemented. Therefore, had it not been the case and regardless of any prior conclusions, QRadar, IBM QRadar SOAR, and Scikit-learn will be used, ensuring both feasibility and alignment with organizational resources.

Chapter 3

Method and Implementation

This chapter details the methodology employed to tackle the problem, providing an overview of the technologies utilized and the implementation process.

It consists of two sections: the first describes the method's design and proposed solutions, while the second covers the proof of concept, including implementation and optimization in a live environment.

3.1 Method

This section describes the methodological foundation of the project. It begins by presenting the technologies used and continues with an outline of three distinct solution strategies to the problem.

Each approach is described through its pipeline and architectural design, offering a comparative view of possible solutions.

3.1.1 Technological Overview

The solution will be developed and implemented using the Python programming language, version 3.10.12. The following libraries are expected to be used throughout the development of the project:

- **scikit-learn** (version 1.6.1): This library will be used for building machine learning models, including the Random Forest model. It provides necessary functionalities for model training, testing, and performance evaluation.
- **pandas** (version 2.2.3): Pandas will be used for data manipulation and analysis, particularly for handling, processing, and cleaning the alert datasets. It provides efficient data structures for handling large amounts of structured data.
- **fastapi** (version 0.115.11): FastAPI will be used to create the necessary APIs for data flow between the components. It will expose endpoints for the system, allowing real-time predictions and feedback from analysts to be communicated between the machine learning models and the IBM SOAR interface.
- **matplotlib** (version 3.10.1): Matplotlib will be used to generate plots and graphs, particularly for data exploration, visualizing model performance, and evaluating results. It will assist in identifying patterns in the data and understanding how well the model is classifying the security alerts.

- **SentenceTransformer** (likely needed): SentenceTransformer will be used for text vectorization, particularly for converting textual data such as alert descriptions and analyst comments into numerical embeddings. These embeddings will be used as inputs to machine learning models for better understanding and classification of textual data.

3.1.2 Problem-Solving Approaches

This subsection analyzes three distinct approaches to tackling the identified problem. It outlines the specific methodologies employed for each approach, detailing the pipeline process involved and the architectural framework that supports its implementation.

Pipeline The pipeline shown in Figure 3.1 represents the general flow of data through various stages of the system. This flow starts with the ingestion of raw security alert data, continues through preprocessing, and is ultimately processed by a machine learning model to generate predictions. These predictions are then presented on the IBM SOAR dashboard, where analysts can review them and provide feedback, which is used to improve the model.

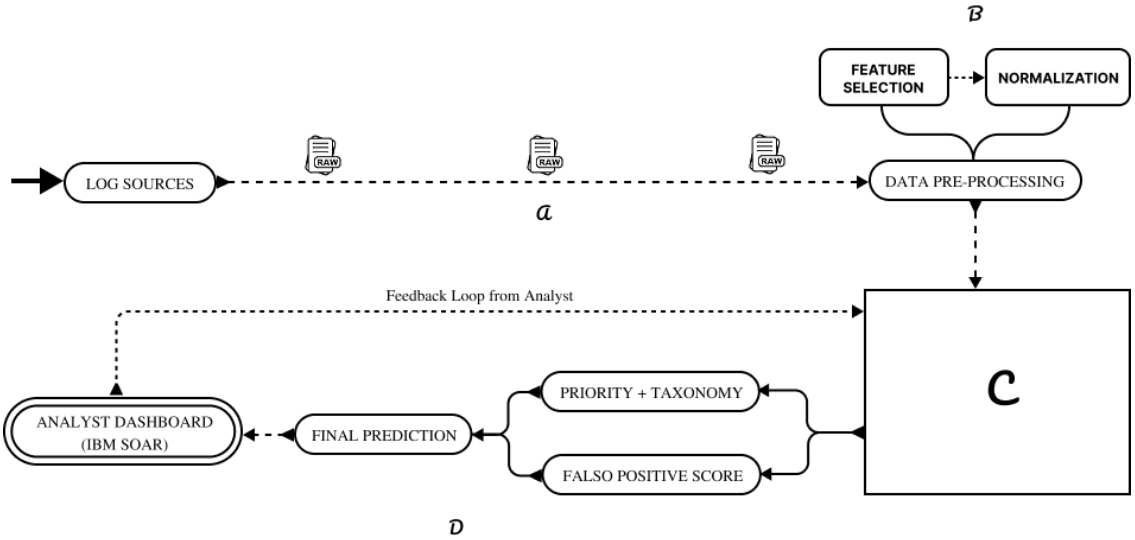

Figure 3.1: General Pipeline for all Three Solutions

This pipeline remains largely the same across all solutions, with the exception of section C, where the machine learning model varies depending on the solution. While the overall structure of the pipeline is consistent, the type of model used in section C determines the specific prediction process and outputs.

To interpret the figure, a reader should understand the following stages in the pipeline:

- **Section A:** Raw security alerts are collected from various sources, such as SIEM systems, IDS, and firewalls. The data is then standardized to ensure consistency.
- **Section B:** The collected data undergoes preprocessing, which includes cleaning, normalization, and feature extraction.
- **Section C:** This section involves the machine learning model. The specific model used here varies depending on the solution, and it generates predictions based on the preprocessed data.

- **Section D:** Finally, the predictions are displayed on the IBM SOAR dashboard. Analysts can provide feedback on the predictions, and this feedback is used to improve the model over time.

The diagram helps visualize how the raw data moves through the system and how the predictions are continuously refined with feedback from the analysts.

Random Forest with Reinforcement Learning Feedback Loop

The first solution proposed in this study is a two-layered machine learning system that integrates a Random Forest model with a Reinforcement Learning feedback loop.

This solution addresses the problem using a pre-trained RF model on historical data combined with an RL model that refines predictions based on analyst feedback. This integration of RF with RL contributes to the model's adaptability, making it highly responsive to new threats.

Design The RF model serves as the decision-making core, trained on a historical dataset of security alerts to classify incoming alerts by taxonomy and priority—Objective 3.

While the RF model provides strong initial predictions, it struggles to adapt to new attack vectors not represented in its training data. To address this, the RL model enhances the predictions made by the RF model and evaluates alerts as false positives or true positives, using feedback from security analysts.

The implementation complexity of this solution is considered moderate, as it does not involve highly intricate algorithms or architectures. However, the necessity to implement and integrate two distinct models—a RF for initial predictions and a RL model for feedback-driven refinement—adds an additional layer of complexity.

Key features of this solution include:

- The RL model adjusts its parameters through a continuous feedback loop, improving classification and generating confidence scores.
- This dynamic learning process ensures the system is able to handle both `cold start` issues effectively (thanks to the pre-trained RF model) and false positives in real-time.
- The system's adaptability to new threats is high via the RL model and helps manage false positives by learning patterns of false positives or true positives, refining its algorithms, and potentially reducing analysts' workloads.
- Designed to efficiently manage and analyze real-time alerts generated from a diverse range of log sources—Objective 4.

The interpretability of this solution is considered moderate due to the significant role played by AI in its decision-making processes. The RF model, being a supervised learning algorithm, provides a level of transparency as its outputs can be traced back to the quality and features of the historical dataset used for training.

However, integrating RL complicates interpretability. RL's iterative feedback and rewards lead to less transparent reasoning, making analyzing and communicating insights challenging.

Integrating this solution with IBM SOAR is relatively easy and straightforward. By relying on the API endpoints provided by this solution, IBM SOAR can seamlessly communicate

with the solution platform. This is vital for enhancing the overall effectiveness of the solution—Objective 5.

IBM SOAR provides customization of dashboards presented to analysts, enabling:

- Creation of custom fields where analysts can view the solution's outputs and mark them as correct or incorrect.
- Feedback from analysts, which is crucial for the RL model to learn from mistakes and improve over time.

This feedback loop supports continuous refinement and enhances accuracy for future predictions. Overall, this strategy leverages the strengths of supervised and reinforcement learning to create a robust solution for security alert triage. It enhances operational efficiency and threat response capabilities by:

- Minimizing false positives through continuous improvement.
- Allowing integration with IBM SOAR to evaluate and test the solution's effectiveness in categorizing alerts and minimizing false positives—Objective 6.

Pipeline The illustration in Figure 3.2 corresponds to section `C` of the general pipeline in Figure 3.1.

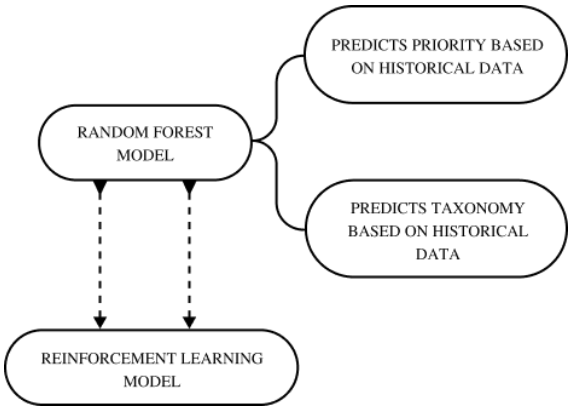

Figure 3.2: Part C of the General Pipeline for the Random Forest with Reinforcement Learning Feedback Loop Solution

This figure provides a detailed representation of the machine learning model's specific implementation for this solution.

It elaborates on the processes and components that occur within section `C`, as outlined in the general pipeline, showcasing the unique aspects of this solution's approach to data processing and prediction generation.

End-to-End Deep Learning Classifier with Feature Fusion

The second solution in this study employs a deep neural network (DNN) model trained end-to-end on labeled security alert data, based on historical alerts dataset—Objective 3.

Unlike solution one, which uses a pre-trained RF model, this approach processes raw SIEM logs, analyst comments, and metadata through the DNN's attention layers to automatically extract features and classify data, eliminating the need for manual feature engineering.

While solution one adapts to new threats with a reinforcement learning feedback loop, solution two relies solely on the DNN model, necessitating training on a comprehensive dataset before deployment. The DNN excels in complex feature extraction environments but requires periodic retraining based on analyst feedback.

Design In this solution, the DNN model processes raw security alert data directly, leveraging attention mechanisms to identify and prioritize critical features like alert descriptions, origins, and metadata.

The attention layers enhance the model's ability to focus on relevant patterns, improving classification accuracy and adaptability.

By automating feature extraction, this approach streamlines the pipeline, reducing dependency on domain-specific preprocessing techniques.

Key aspects of this solution are:

- The DNN model takes in raw security alert data, which is vectorized, and outputs predictions on alert taxonomy, priority, false positive status, and confidence score.
- The DNN model's adaptability is moderate and depends on the training data's quality and diversity. Once trained, it performs well on known data but is less responsive to new attack patterns unless retrained with fresh data.
- Solution two faces challenges with cold start problems, as the DNN model requires a substantial amount of labeled training data before it can begin making accurate predictions. Unlike solution one, which benefits from the pre-trained RF model, this solution requires extensive training before it can function effectively.
- Engineered to effectively process and evaluate real-time alerts originating from a wide variety of log sources—Objective 4.

The implementation complexity of solution two is considered high due to the extensive computational resources required to train the DNN model. Training a deep neural network, especially one that handles raw alert data with attention mechanisms, demands significant time and resources, making it more complex to implement.

The interpretability of the solution is low, primarily because the DNN model operates as a "black box". While it offers powerful predictions, the model's inner workings are difficult to explain, particularly when attention layers are involved, making it harder to interpret why certain predictions were made.

This solution integrates with IBM SOAR for real-time feedback and performance evaluation—Objective 5, allowing analysts to review predictions and provide feedback, which is subsequently used for periodic retraining to assess and validate the solution's capability in classifying alerts and reducing the occurrence of false positives—Objective 6.

Rule-Augmented Decision Tree with Feedback Aggregation

The third solution proposed in this study integrates static expert rules with a lightweight Decision Tree Classifier (DTC). This hybrid system uses a rule engine to pre-filter known benign or critical alert patterns before passing any unknown or ambiguous cases to the Decision Tree Model (DTM).

The rules provide initial filtering for quick decisions on clear alerts. For complex cases, the DTC predicts based on available data. Feedback is reviewed in batches, enabling periodic updates to the rule base and retraining of the decision tree.

Design This solution combines the benefits of domain-specific rules with machine learning. It starts with a rule engine that quickly processes alerts based on known patterns, tagging them as benign or critical and minimizing the need for further analysis. This approach is efficient in environments with well-defined, static patterns. This solution will be designed to ingest data from a variety of log sources, including SIEM systems and other security tools—Objective 4.

The rule engine will preprocess and filter alerts from these sources, leveraging predefined rules to handle known patterns efficiently, while less adaptive, predefined rules to filter known benign patterns, ensuring a baseline reduction in false positives.

For alerts that don't fit predefined rules, the DTC classifies them using features from raw data, allowing the system to manage both known and unknown patterns while balancing interpretability and accuracy.

The DTC works with a clean, segmented dataset to differentiate between alerts that match known patterns and those requiring further analysis—Objective 3.

Key features of this solution include:

- The rule engine filters known benign or critical patterns before passing uncertain alerts to the DTC. This approach reduces the overall processing time and helps prioritize more ambiguous cases.
- Feedback is logged and applied asynchronously, meaning that the system doesn't update in real-time. Instead, the rule base is updated, and the decision tree is retrained on a periodic basis (e.g., weekly). This ensures that the system evolves over time but does not immediately adjust to new threats in real-time.
- The system's ability to handle new and evolving threats is low, as it depends on manually updating the rule base to address new threats.
- *Cold start* handling is excellent because the rules-based filtering system can operate immediately without requiring training, and the DTC is lightweight enough to be quickly deployed after the initial setup.

The implementation complexity of this solution is low, as it uses well-understood decision tree models and a simple rule engine. This makes it relatively easy to implement when compared to the other solutions. However, the trade-off is that the DTC may not be as powerful as deep learning-based models or even RL models when dealing with complex, high-dimensional data.

The interpretability of this solution is considered high. Both the rule engine and the decision tree are inherently interpretable, allowing analysts to understand the reasoning behind the classifications. This makes it particularly useful in environments where auditability and transparency are important. However, the model's inability to adapt in real-time can make it less suitable for rapidly evolving attack patterns.

Integrating this solution with IBM SOAR is straightforward, allowing for seamless communication between the alert system and the dashboard used by analysts. Alerts classified by

the rule engine and decision tree are sent to the IBM SOAR dashboard, where analysts can review predictions and provide feedback—Objective 5. The feedback is stored and reviewed periodically to refine the rules and retrain the decision tree.

Architecture

The diagram in Figure 3.3 illustrates the architecture for all solutions.

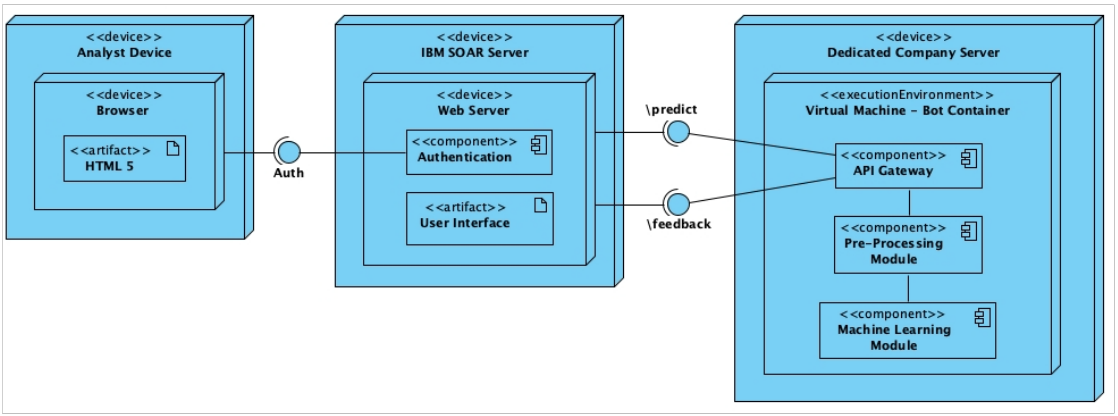

Figure 3.3: Architecture of the Random Forest with Reinforcement Learning Feedback Loop Solution

It's organized into three major sections:

1. **Analyst Device**: This device acts as the point of interaction where security analysts review and manage alerts. Represents the user interface. The analyst logs into the system through the **Authentication** component, enabling access to the dashboard.
2. **IBM SOAR Server**: The IBM SOAR Server is the central system for managing security alerts. The server sends requests to the **Bot Container** through the `\predict` endpoint, triggering the alert classification process. Once the alert is processed, the `\feedback` endpoint is used to receive the analyst's input for further training of the RL model.
3. **Dedicated Company Server**: The **Dedicated Company Server** hosts the **Bot Container**, which is deployed on a **Virtual Machine (VM)**. This container is responsible for the core functionality of the solution, consisting of three main components:
    - **API Gateway**: This component serves as the entry point for receiving requests from the IBM SOAR Server and handling communication between the components inside the bot container. It processes the incoming alert data and forwards it to the appropriate modules.
    - **Pre-Processing Module**: This module cleans and prepares the incoming alert data, extracting features and transforming them into a suitable format for the model.
    - **Machine Learning Module**: The **Machine Learning Module** applies the Random Forest (RF) model to classify the alert. These predictions are then further refined through the Reinforcement Learning (RL) model.

This architecture is designed to be modular and scalable, allowing for easy integration with existing systems and the ability to adapt to evolving security threats.

3.1.3 Comparative Analysis

Table 3.1 presents a comparison of the three proposed approaches based on key evaluation criteria. The goal is to assess their suitability in the context of a real-world SOC environment, taking into consideration implementation complexity, adaptability, performance, interpretability, and integration potential.

Table 3.1: Comparison of Proposed Solutions

Criteria	RF + RL	Deep Neural Network (DNN)	Rules + Decision Tree
Architecture Type	RF + RL (Two-layer)	DNN	Rule-based + Decision Tree
Implementation Complexity	Moderate	High	Low
Adaptability to New Threats	High (via RL)	Moderate (via retraining)	Low (manual updates)
Learning from Feedback	Online (via RL)	Periodic retraining	Batch/manual integration
Cold Start Handling	Excellent (RF pre-trained)	Poor	Excellent (rules pre-set)
Interpretability	Moderate	Low	High
Scalability	High	High	Moderate
Integration with SIEM (IBM SOAR)	Easy	Easy	Easy
False Positive Reduction	Adaptive (confidence scoring)	Model confidence only	Rigid (rule-defined)
Performance in Evolving Scenarios	High	Moderate	Low

Solution 1 offers the best combination of **moderate implementation complexity** and **high adaptability**. Its use of a pre-trained RF model combined with a **Reinforcement Learning (RL)** feedback loop allows the system to continuously learn and adapt in real-time. This continuous learning process ensures the solution remains effective in dynamic and rapidly evolving environments, handling new threats efficiently. The **moderate complexity** comes from integrating the two models, but this is outweighed by its high scalability and ability to adjust based on new data.

Solution 2, while **highly scalable**, has **high implementation complexity** due to the need for substantial computational resources to train the DNN. Although it offers excellent feature extraction capabilities, its **moderate adaptability** limits its response to new threats. The DNN requires retraining with new data, making it slower and less responsive compared to Solution 1's real-time learning.

In contrast, Solution 3 is **easy to implement** and offers **high interpretability** due to its rule-based system and decision trees. However, its **low adaptability** makes it unsuitable for handling evolving threats, as it depends on manual updates and cannot learn in real time. While the simplicity of this approach is beneficial in static environments, it fails to scale effectively in more complex, dynamic security landscapes.

In terms of performance, Solution 1 excels in **performance in evolving scenarios**, making it the most suitable for fast-changing environments. Solution 2 performs **moderately** in this regard, with slower adjustments compared to Solution 1. Solution 3, being static, performs **poorly** in evolving scenarios and requires manual intervention to remain effective.

Considering the trade-offs, **Solution 1** stands out as the optimal choice, balancing scalability, adaptability, and complexity. Its ability to evolve with emerging threats makes it the most reliable solution for real-time alert classification in dynamic cybersecurity environments.

3.2 Proof of Concept

This section presents the practical implementation of the proposed solution, including the setup of the test environment, preparation of the dataset, and execution of the ML models. It aims to demonstrate the feasibility and performance of the selected approach under realistic conditions.

3.2.1 Data Processing

This section outlines the process of refining the raw data, corresponding to phase B in the solution pipeline.

As previously mentioned, Solution One was the best choice among the three alternatives.

To implement this solution effectively, ArtResilia provided a dataset of security alerts generated by its SOC.

This dataset, covering the period from February 9, 2022, to January 31, 2025, encompasses various alerts categorized by attributes, such as priority and taxonomy. The primary goal of the data processing phase was to prepare this raw data for training ML models to predict the priority and taxonomy of alerts based on the textual descriptions associated with each alert.

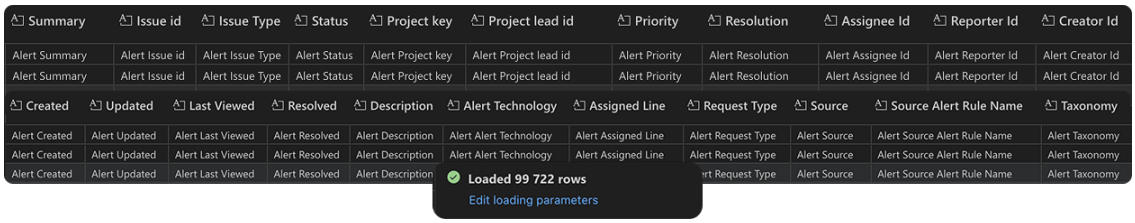

Figure 3.4: Original Dataset Before Preprocessing

Figure 3.4 illustrates the original dataset before preprocessing. The dataset[1] contains 99,722 rows, each representing a security alert.

To prepare the data for training ML models, several key preprocessing steps were performed:

First Step Rows missing data in "Priority" or "Taxonomy" were removed, retaining only the essential columns: "Description", "Priority", and "Taxonomy". These columns provide the text for training and the target labels.

[1]For privacy reasons, the actual data has been censored or altered to ensure confidentiality.

Listing 1 shows the Python code snippet used for this initial preprocessing step. In line 1, the dataset is filtered to exclude alerts where the 'Source' field is marked as 'Other' and to retain only those rows where the 'Source Alert Rule Name' field is not empty. In line 2, the list of columns relevant for the model training is defined, selecting only 'Description', 'Priority', and 'Taxonomy'. Line 3 applies this selection to the dataset, keeping only the specified columns. Finally, in line 4, any remaining rows that still contain missing values in these essential columns are removed to ensure the completeness of the dataset used for training.

```
1 df = df[(df['Source'] != 'Other') & (df['Source Alert Rule Name'].notna())]
2 required_columns = ["Description", "Priority", "Taxonomy"]
3 df = df[required_columns]
4 df = df.dropna(subset=required_columns)
```

Listing 1: Python Code Snippet for the First Part of Preprocessing the Dataset.

Second Step Alerts with a Priority of "P4" or a Taxonomy labeled as "Other" were also excluded as these were deemed less relevant for the model's prediction task. To address class imbalance in the Taxonomy column, random oversampling was performed. This involved ensuring that smaller categories had enough examples for the model by duplicating instances from underrepresented taxonomies until they matched the size of the largest category.

Listing 2 provides the Python code snippet used for the second preprocessing step, which includes filtering and oversampling to address class imbalance in the dataset. In line 1, rows where the priority is "P4" or taxonomy is "Other" are excluded to retain only relevant data for classification. In line 2, the distribution of samples across taxonomies is calculated. Line 9 performs the oversampling by randomly duplicating instances of underrepresented taxonomies to balance the dataset. Finally, in line 12, all oversampled subsets are concatenated into a single balanced dataset.

```
1  df = df[(df["Priority"] != "P4") & (df["Taxonomy"].str.lower() != "other")]
2  taxonomy_counts = df["Taxonomy"].value_counts()
3  max_len = taxonomy_counts.max()
4  oversampled_dfs = []
5  for tax, count in taxonomy_counts.items():
6      sub_df = df[df["Taxonomy"] == tax]
7      if count < max_len:
8          diff = max_len - count
9          extra_rows = sub_df.sample(n=diff, replace=True, random_state=42)
10         sub_df = pd.concat([sub_df, extra_rows], ignore_index=True)
11     oversampled_dfs.append(sub_df)
12 df = pd.concat(oversampled_dfs, ignore_index=True)
```

Listing 2: Python Code Snippet for the Second Part of Preprocessing the Dataset.

By applying the preprocessing code above from steps one and two, the dataset was reduced from its original 99,722 rows, representing security alerts, to approximately 81,732 rows.

However, after the oversampling process, the dataset was expanded to around 173,637 rows, ensuring a balanced representation of all taxonomies.

Third Step The "Description" column, which contains raw text data, was also cleaned. Unnecessary formatting was removed as well as special characters, and irrelevant sections (such as URLs and IBM SOAR links). It also handled tasks like stripping extra whitespace and removing HTML tags, ensuring the text was in a consistent and usable format for the machine learning pipeline.

Listing 3 shows the Python code snippet used for this third preprocessing step. When this line is executed, the clean_text function is applied to each row of the Description column, systematically cleaning and normalizing the text, while progress_apply provides a visual progress bar during the operation to monitor execution.

```
1 df["Description"] = df["Description"].progress_apply(clean_text)
```

Listing 3: Python Code Snippet for the Third Part of Preprocessing the Dataset.

The code snippet presented above applies the `clean_text` function uniformly to all entries within the "Description" column of the dataset, ensuring consistent preprocessing of textual data for subsequent machine learning tasks.

The rules defined for normalizing the description text will be detailed in the next section, where the `clean_text` function is explained.

Fourth Step Finally, the "Priority" and "Taxonomy" labels were stripped of any leading or trailing spaces to ensure consistency in the dataset.

Listing 4 shows the Python code snippet used for this final preprocessing step. In lines 1 and 2, both the Priority and Taxonomy columns are first explicitly converted to string to prevent type inconsistencies, and then any leading or trailing whitespace were removed to ensure label consistency across the dataset.

```
1 df["Priority"] = df["Priority"].astype(str).str.strip()
2 df["Taxonomy"] = df["Taxonomy"].astype(str).str.strip()
```

Listing 4: Python Code Snippet for the Fourth Part of Preprocessing the Dataset.

These preprocessing steps ensure that the dataset is clean, balanced, and properly formatted, making it ready for use in the ML models. The combination of filtering, oversampling, text cleaning, and label processing prepares the data for training while addressing potential issues such as class imbalance and noisy raw data.

Figure 3.5 illustrates the dataset[1] after preprocessing, showcasing the difference between the original and processed versions.

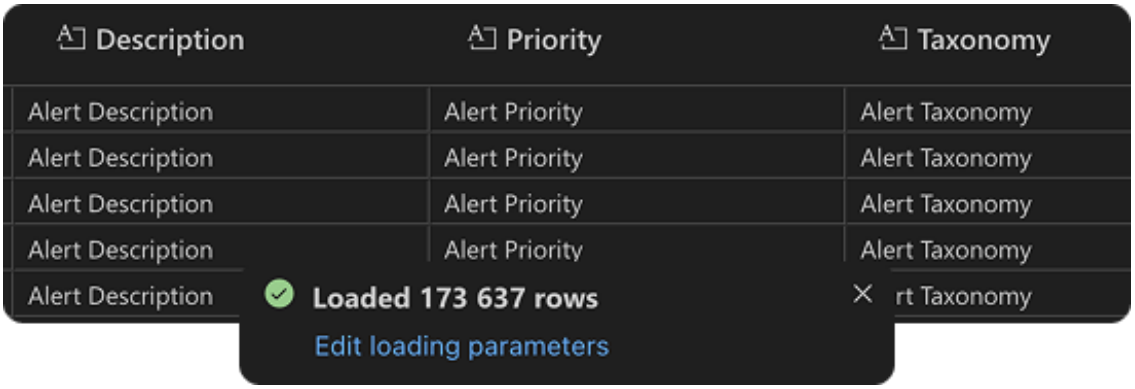

Figure 3.5: Original Dataset After Preprocessing

3.2.2 Data Normalization

Data normalization typically refers to the process of transforming data into a common scale to improve the performance and accuracy of machine learning models.

In this project, normalization was applied specifically to the "Description" column, which contains the raw text data.

This is crucial, as natural language data can contain noise, irrelevant characters, and inconsistencies that can negatively impact model training and performance.

Unlike the Priority and Taxonomy columns, which are already in categorical formats suitable for ML models, the Description column required more extensive processing to make it usable for training.

As stated in the previous section, the normalization process was performed using a function called `clean_text`, which was designed to normalize the raw text in the Description column. This function specifically targets common issues in textual data, such as extraneous formatting, unwanted characters, and irrelevant content.

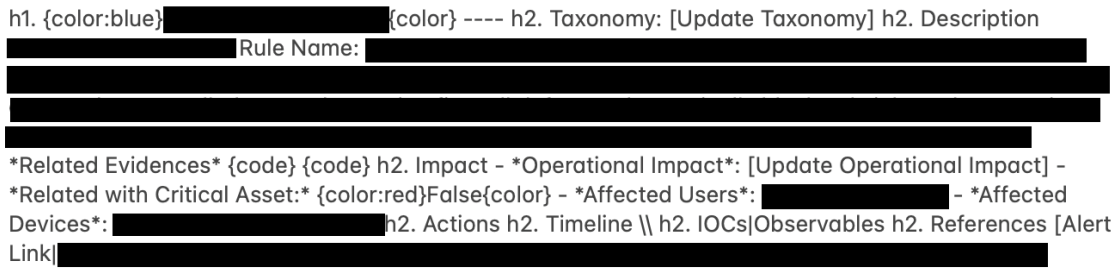

Figure 3.6: Redacted Description Text

Figure 3.6 presents a redacted[2] visualization that exemplifies the typical format of a description text. The formatting of this description may differ based on the source of the alert, for example, some information may not appear at all or the description text formatting is different, highlighting the diverse origins and structures of the data. To address these variations, the `clean_text` function was created to standardize the presentation of the text data, ensuring it remains consistent regardless of its source.

[2]For privacy reasons, the actual data has been censored or altered to ensure confidentiality.

The `clean_text` function performs several key operations:

- **Extract Relevant Content:** Identifies and retains the "Description" section, removing irrelevant sections like "Taxonomy" or "References" if present.
- **Remove Unwanted Characters:** Cleans special symbols, color tags, code markers, HTML tags, and URLs to retain only meaningful content.
- **Text Cleaning:** Strips extra spaces, removes irrelevant phrases (e.g., "action:" or "related evidences:"), and ensures consistent formatting.
- **Handle Rule Name:** Captures and formats rule names with associated narratives as `<Rule Name> - <Narrative>` for clarity.
- **Final Output:** Returns a cleaned, normalized description.

By applying a normalizing function to the Description column helps achieve consistency and reduce noise in the training text data. The Priority and Taxonomy columns did not require the same normalization process, as they are categorical and do not need extensive text processing.

3.2.3 Dataset Division

Dataset Division refers to how the dataset is split into subsets used for training, validation, and testing.

This division is crucial because it allows the model to be trained on one subset of the data while ensuring that the model's performance is evaluated on unseen data.

This prevents overfitting, where the model performs well on training data but fails to generalize to new, unseen data.

Similarly, proper dataset division also avoids underfitting, where the model might not learn enough from the training data to perform well.

In ML, the dataset is typically divided into three parts:

- **Training Set:** Used to train the model.
- **Validation Set:** Used to tune the model's hyperparameters and prevent overfitting during training.
- **Test Set:** Used to evaluate the final model's performance after training is complete.

In this project, the dataset was divided as follows:

- **Training and Test Split:** The data was split into 70% for training and 30% for testing. This ratio strikes a balance between providing enough data for the model to learn from, while still maintaining a sufficient amount of data to test its generalization ability. Listing 5 shows the Python code used to perform the dataset split. The train_test_split function divides the feature set X and the labels Y into training and testing subsets, while preserving the class distribution across both sets through stratification with the stratify parameter. This approach was chosen to maximize the model's ability to generalize to unseen data while preventing overfitting.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
↪  random_state=42, stratify=stratify_labels)
```

Listing 5: Splitting Dataset.

- **Stratified Sampling:** The split was performed using stratified sampling, ensuring that the proportion of each class in Priority and Taxonomy was maintained across the training and test sets. Listing 6 illustrates how stratified sampling was implemented. In the first line, the list stratify_labels is created by converting each row of Y into a tuple, combining both Priority and Taxonomy labels. This ensures that the stratification process maintains the joint distribution of both target variables during the subsequent data split performed by train_test_split. This helps avoid bias in the dataset and ensures that both the training and test sets have representative distributions of all classes.

```
stratify_labels = [tuple(row) for row in Y]
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
↪  random_state=42, stratify=stratify_labels)
```

Listing 6: Stratified Sampling.

This stratified split is particularly important for imbalanced datasets, as it ensures that each class is proportionally represented in both the training and testing subsets, thereby improving the model's ability to generalize and minimizing bias toward the majority class.

- **Test Set**: The 30% test set was kept completely separate from the training process, ensuring that the model's performance could be assessed on data it had never seen before. This provides an unbiased evaluation of the model's accuracy and generalization capabilities.

By properly dividing the dataset, this approach ensures that the model is trained on a representative subset of data, can be fine-tuned using the validation set, and is evaluated fairly on the test set. This helps in achieving a balance between training the model effectively and ensuring generalization without overfitting or underfitting.

3.2.4 Machine Learning Model Development

This section describes the implementation of the RF model and the RL feedback loop, explaining how these models were integrated to complement each other and enhance prediction accuracy in the system.

Random Forest Model Implementation

The **RF model** is implemented using scikit-learn's **RandomForestClassifier**. It is employed as the core decision-making component in the system, trained to predict two target variables: **priority** and **taxonomy** of security alerts. To handle these **multi-output targets**, **MultiOutputClassifier** is used, which allows the model to make predictions on both target variables simultaneously.

The first step in implementing the **RF model** is to transform the raw textual data in the **Description** column into numerical features. This is achieved using **SentenceBertVectorizer**, a tool that leverages the **Sentence-BERT model** to generate embeddings from the alert descriptions. These embeddings capture the **semantic meaning** of the descriptions, which is crucial for the model to make accurate predictions.

```
1 vectorizer = SentenceBertVectorizer(model_name="paraphrase-MiniLM-L6-v2")
2 X = vectorizer.fit_transform(df["Description"])
```

Listing 7: Vectorizing Text Data.

Listing 7 presents the code used for the text vectorization step. In the first line, the `SentenceBertVectorizer` is initialized with the pre-trained model `"paraphrase-MiniLM-L6-v2"`. In the second line, the `fit_transform` function converts all entries from the *Description* column into numerical embeddings that serve as feature vectors for model training.

Once the text data is vectorized, the target labels (Priority and Taxonomy) are encoded using LabelEncoder. This step converts the categorical labels into numeric form, making them suitable for input into the RF model.

```
1 # Encode targets for Priority and Taxonomy
2 le_priority = LabelEncoder()
3 le_taxonomy = LabelEncoder()
4 y_priority = le_priority.fit_transform(df["Priority"])
5 y_taxonomy = le_taxonomy.fit_transform(df["Taxonomy"])
6
7 # Combine the targets into a 2D array
8 Y = np.column_stack((y_priority, y_taxonomy))
```

Listing 8: Encoding Target Labels.

Listing 8 shows the encoding process. In lines 1 and 2, two `LabelEncoder` instances are initialized, one for each target. Lines 3 and 4 transform the textual labels into integer representations. Finally, in line 8, both encoded targets are combined into a two-dimensional array `Y`, allowing the model to predict both outputs simultaneously.

Next, the dataset is split into training and testing sets using train_test_split from scikit-learn. This ensures that the model is trained on one subset of data and evaluated on another, providing an unbiased assessment of its performance.

```
1 # Split the data into training and testing sets
2 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3,
  ↪  random_state=42, stratify=Y)
```

Listing 9: Splitting the Dataset.

Listing 9 shows the dataset splitting process. The `train_test_split` function allocates 70% of the data for training and 30% for testing. The `stratify` parameter ensures that the class distribution for both **Priority** and **Taxonomy** remains consistent across both sets, preserving representativeness and preventing class imbalance during training.

After splitting the data, the RandomForestClassifier is initialized and wrapped with MultiOutputClassifier to handle the multi-output prediction task. The model is trained on the training data and evaluated on the test set.

```
1 # Train a MultiOutputClassifier using RandomForest
2 base_rf = RandomForestClassifier(n_estimators=500, random_state=42,
  ↪  max_depth=40, min_samples_split=10)
3 multi_rf = MultiOutputClassifier(base_rf)
4 multi_rf.fit(X_train, Y_train)
```

Listing 10: Training the Random Forest Model.

Listing 10 details the model initialization and training. In line 1, the `RandomForestClassifier` is configured with specific hyperparameters optimized during prior experimentation. In line 2, it is wrapped with `MultiOutputClassifier` to handle multiple outputs, and in line 3, the model is trained on the training dataset.

Reinforcement Learning Model Implementation

While the RF model is capable of making initial predictions, it requires further refinement to adapt to new or unseen threats. This is where the RL model comes in. The RL model is designed to improve predictions by incorporating feedback from security analysts.

The RL model is built using the Proximal Policy Optimization (PPO) algorithm from Stable Baselines3, a popular library for RL. The model's role is to adjust the predictions made by the RF model based on feedback received from analysts.

At the core of this RL model is a reward function that is based on whether the model's predictions are correct, focusing on both false positive (FP) and true positive (TP) predictions for taxonomy and priority. The reward function is designed to penalize incorrect predictions and reward correct ones, with particular weight given to more critical mistakes.

False Positive (FP) Evaluation The third element of the action vector represents the **RL model's FP prediction**.

A positive reward is given if the model's FP prediction matches the ground truth, while a penalty is applied if the prediction is incorrect.

$$R_{\text{FP}} = \begin{cases} +w_{\text{tp}} \cdot \text{confidence} & \text{if FP prediction is correct} \\ -w_{\text{fn}} & \text{if predicted FP but true TP} \\ -w_{\text{fp}} & \text{if predicted TP but true FP} \end{cases}$$

Priority and Taxonomy Classification The **priority** and **taxonomy** predictions are compared to the true values. A penalty is applied if either the priority or taxonomy prediction is incorrect.

$$R_{\text{error}} = \alpha \cdot \text{priority\_error\_indicator} + \beta \cdot \text{taxonomy\_error\_indicator}$$

Final Reward The final reward is the sum of the FP evaluation and the error penalties for priority and taxonomy:

$$R = R_{\text{FP}} - R_{\text{error}}$$

Where:

- w_{tp} is the weight for true positives,
- w_{fn} and w_{fp} are penalties for false negatives and false positives, respectively,
- α and β are the penalty weights for priority and taxonomy errors,
- **confidence** is the model's confidence in its FP prediction,
- **priority_error_indicator** and **taxonomy_error_indicator** are binary values indicating whether the respective predictions are correct.

This reward function allows the RL model to adjust its predictions based on feedback, ensuring continuous improvement in the model's performance.

The RL agent uses the above reward function to adjust its policy (i.e., how it makes predictions) based on the rewards it receives after each prediction. The PPO algorithm ensures that policy updates are stable and efficient, making it well-suited for real-time systems.

To manage computational load during training, the RL model is trained asynchronously using Celery, a distributed task queue. Celery ensures that training tasks do not block other system operations, enabling the RL model to be trained in parallel with real-time alert classification.

Here's an example of how the RL training task is defined using Celery:

```
1 @celery_app.task
2 def train_rl_agent_task(feedback, model_version):
3     dummy_env = RLDummyEnv(observation_dim=387)
4     rl_agent = PPO.load(src.config.RL_AGENT_PATH, env=dummy_env)
5
6     model = load_model(model_version)
7     result_info = update_rl_agent_with_feedback(feedback, model, rl_agent)
8     return result_info
```

Listing 11: RL Training Task with Celery.

Listing 11 illustrates how the RL training task is defined asynchronously using Celery. In line 1, the function is registered as a Celery task. In line 2, a dummy reinforcement learning environment is instantiated to match the model's observation space. In line 3, the pre-existing RL agent is loaded from storage. In line 5, the corresponding Random Forest model version is loaded to provide context for the RL update. Finally, in line 6, the function update_rl_agent_with_feedback applies the feedback to refine the RL model, and the updated training information is returned.

This asynchronous processing model allows the RL agent to continuously improve based on real-time feedback while ensuring that the system remains responsive to new alert data. Although, for this to happen the RL model need to be trained in a separate process to ensure efficient and uninterrupted training. However, this separation means that the most recently

trained version of the model is not immediately available to the main process responsible for making predictions.

To address this, the training process saves the model to persistent storage each time training is completed.

The main process, whenever it needs to use the RL model, checks if the model has been updated since the last time it was loaded. If a newer version is detected, the main process reloads the model, ensuring that it always operates with the most up-to-date version of the RL model. This approach maintains consistency between training and inference while allowing both processes to function independently.

3.2.5 Hyperparameter Tuning

In the process of model development, hyperparameter tuning was performed for both the RF and RL models to optimize their performance.

The primary goal of hyperparameter tuning is to find the best configuration of hyperparameters that allows the models to generalize well to unseen data, reducing the risk of overfitting while maintaining high predictive accuracy.

For the RF model, tests were run using a `GridSearchCV` approach to explore a range of hyperparameters. However, after extensive testing, the most effective hyperparameters were identified and are now used in the model's initialization.

- **Number of estimators (`n_estimators`)**: Set to 500.
- **Maximum depth (`max_depth`)**: Set to 40.
- **Minimum samples required to split an internal node (`min_samples_split`)**: Set to 10.
- **Minimum samples required at each leaf node (`min_samples_leaf`)**: Set to 1.
- **Class weight (`class_weight`)**: Set to `'balanced'`.

The following code snippet reflects the initialization of the RF model with the chosen hyperparameters:

```
1 base_rf = RandomForestClassifier(n_estimators=500, random_state=42,
2                                  max_depth=40, min_samples_split=10,
3                                  min_samples_leaf=1, class_weight='balanced')
4 multi_rf = MultiOutputClassifier(base_rf)
```

Listing 12: Random Forest Model Initialization.

Listing 12 shows the initialization of the Random Forest model with the selected hyperparameters. In line 1, the `RandomForestClassifier` is instantiated with 500 estimators, a maximum tree depth of 40, a minimum of 10 samples required to split an internal node, and balanced class weights to handle class imbalance. In line 2, the classifier is wrapped with `MultiOutputClassifier` to enable simultaneous prediction of both **Priority** and **Taxonomy** labels.

For the RL model, a similar approach was used, but the model's hyperparameters, such as learning rate, batch size, and the number of epochs, were fine-tuned to ensure that

the model could learn efficiently from feedback while maintaining stability in training. The PPO algorithm was used, which is known for its stability and effectiveness in reinforcement learning tasks. The key parameters for the RL model include:

- **Learning rate**: Set to 3e-4.
- **Batch size**: Set to 64.
- **Number of epochs**: Set to 4.
- **Gamma**: Set to 0.99.
- **GAE lambda**: Set to 0.9.
- **Entropy coefficient**: Set to 0.01.

The RL model's training process is designed to adjust its predictions based on the feedback it receives, improving accuracy over time by refining its policy.

Both models underwent comprehensive testing to determine the optimal set of hyperparameters. Ultimately, the configurations used in the models' initialization were found to provide the best balance between performance, accuracy, and generalization.

The finalized hyperparameters are now used in the model pipelines, ensuring efficient training and reliable predictions.

Chapter 4

Model Evaluation and Results Analysis

This chapter provides an in depth analysis of the performance and results obtained from testing the models in two distinct environments: the local testing environment and the live testing environment.

4.1 Local Testing Environment

In the local testing environment, the models were evaluated using a dataset that was split into training and testing subsets

Due to hardware limitations[1], the RL model was not tested on this environment, therefore, the results presented in this section are only of the RF model's performance and its evolution from Version 1 to Version 11.

The RF model was trained and tested on the dataset, and its performance was measured using metrics such as accuracy, precision, recall, and F1-score.

- **Accuracy:** The proportion of correctly classified alerts out of the total alerts.
- **Precision:** The ratio of true positive predictions to the total predicted positives, indicating the model's ability to avoid false positives.
- **Recall:** The ratio of true positive predictions to the total actual positives, reflecting the model's ability to capture all relevant alerts.
- **F1-score:** The harmonic mean of precision and recall, providing a balanced measure of the model's performance.

The values for all the metrics are based on the final results from training. Since, after training the model, that same model is tested against a test data subset, it is possible to get values for the model's performance and accuracy.

Table 4.1 shows the values for the metrics of Version 1 and Version 11, showing the evolution of the model from the first to the last version.

[1] Training a single alert takes 10 to 15 minutes, making it impractical to train on 100k alerts in a reasonable timeframe.

Table 4.1: Model Performance Metrics Comparison: Version 1 vs. Version 11

	Priority		Taxonomy	
	Version 1	**Version 11**	**Version 1**	**Version 11**
Accuracy	82%	90%	82%	90%
Precision	82%	89%	81%	90%
Recall	80%	89%	65%	90%
F1-Score	82%	89%	69%	90%

From the table, it is evident that Version 11 demonstrates notable improvements over Version 1 in all metrics. Specifically:

- **Accuracy:** Both priority and taxonomy classifications improved from 82% in Version 1 to 90% in Version 11, indicating a higher proportion of correctly classified alerts overall.
- **Precision:** The precision for priority classification increased from 82% to 89%, and for taxonomy classification, it rose from 81% to 90%. Making it clear that Version 11 is better at minimizing false positives.
- **Recall:** A significant improvement is observed in recall, particularly for priority classification, which increased from 65% in Version 1 to 90% in Version 11. This indicates that the model is now more effective at identifying true positives, especially for minority or low frequency categories.
- **F1-Score:** The F1-Score, which balances precision and recall, also improved from 69% to 90% for priority classification, reflecting the overall enhancement in the model's robustness.

These improvements highlight the evolution of the model's ability to handle complex classification tasks, particularly in addressing challenges associated with low frequency categories. The enhanced recall for minority classes is particularly noteworthy, as it demonstrates the model's capacity to reduce bias and improve fairness in predictions.

Besides accuracy, recall is a fundamental factor, as being able to correctly identify an alert for what it truly is—whether in terms of priority or taxonomy—is critical. Confusing a P1 (high priority) with a P3 (low priority) can have serious consequences, potentially leading to delayed responses or misallocated resources.

The confusion matrix is a fundamental tool in evaluating the performance of classification models, providing a detailed breakdown of the model's predictions across all categories. It is a tabular representation that contrasts the actual labels with the predicted labels, allowing for the identification of true positives, false positives, true negatives, and false negatives for each class.

This granular insight is crucial for understanding the model's strengths and weaknesses, particularly in distinguishing between similar or imbalanced classes. The confusion matrices presented in this section are derived from the same output[2] as the previously discussed

[2]All outputs relevant to this section can be found in Appendix A.

performance metrics, generated during the execution of the command that initiates the training of the RF model.

Therefor, analyzing the confusion matrices is essential to understand the model's performance in detail. The confusion matrices provide a visual representation of the model's performance, allowing us to see how well the model classifies each category and where it tends to make mistakes.

Figures 4.1 and 4.2 illustrate the evolution of the model's performance in priority classification from Version 1 to Version 11.

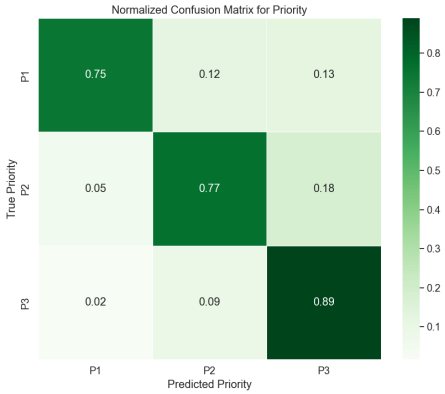

Figure 4.1: Confusion Matrix for Priority (Version 1)

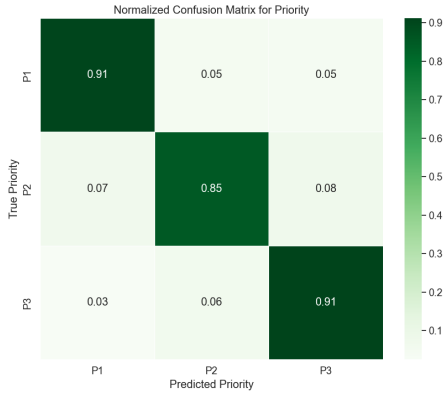

Figure 4.2: Confusion Matrix for Priority (Version 11)

In Version 1, the confusion matrix (Figure 4.1) reveals notable misclassifications, particularly between P1 and P2 categories. The recall values for P1 and P2 were 0.75 and 0.77, respectively, while P3 achieved a higher recall of 0.89, indicating relatively better performance for lower priority classes.

In contrast, Version 11 (Figure 4.2) demonstrates significant improvements, achieving recalls of 0.91 for P1, 0.85 for P2, and 0.91 for P3. These enhancements reflect a substantial reduction in misclassifications, particularly between P1 and P2, showcasing the model's improved ability to differentiate between priority levels.

The key advancements from Version 1 to Version 11 are evident in the increased recall for P2, which rose from 0.77 to 0.85, and for P3, which improved from 0.89 to 0.91. These results underscore the model's enhanced accuracy and robustness in handling priority based predictions, particularly for categories that were previously challenging to classify accurately.

The same analysis can be applied to taxonomy classification, where the confusion matrices provide insights into the model's performance across different categories.

In Version 1 (Figure 4.3), the model's performance was somewhat lacking, especially with low frequency categories. For example:

- **"Abusive Content"** had a recall of only **0.19**, meaning that a substantial portion of this class was misclassified.
- **"Fraud"** performed well with **0.95** recall, demonstrating that the model successfully identifies most instances of fraud.

- **"Vulnerable"** had **0.36** recall, indicating the model struggled to identify alerts related to vulnerabilities.
- **"Intrusions"** also performed poorly with **0.54** recall.
- **"Information Gathering"** was significantly more challenging with **0.47** recall, suggesting that the model could not detect a substantial portion of critical security threats in this category.

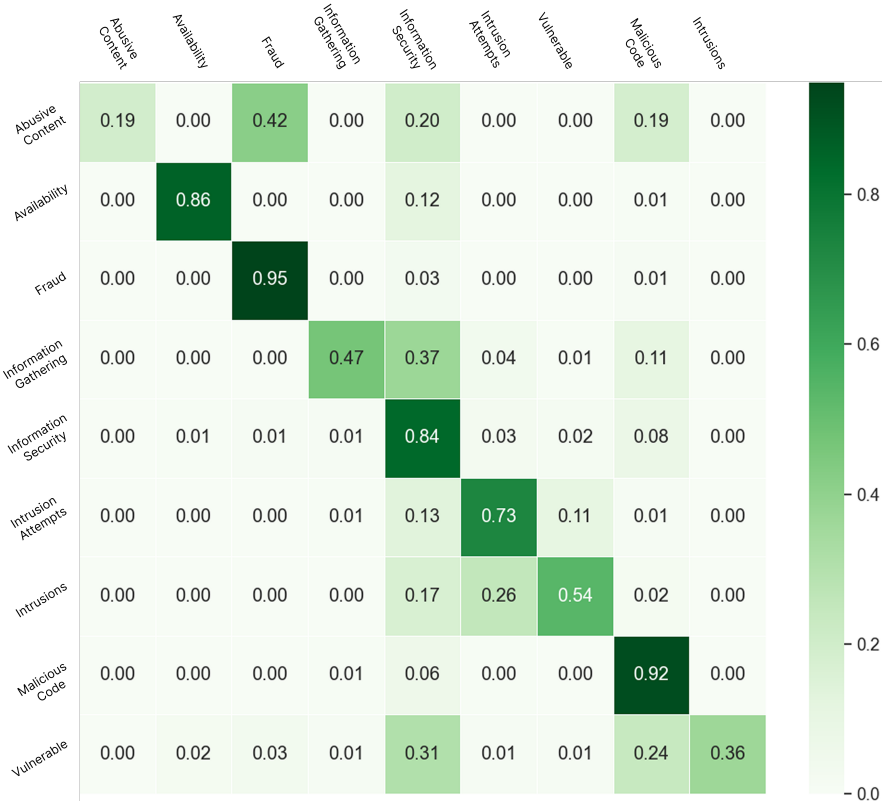

Figure 4.3: Confusion Matrix for Taxonomy (Version 1)

Whereas, in Version 11 (Figure 4.4), the model's performance improved significantly across all categories.

In Version 11, several key improvements were made:

- **"Abusive Content"** showed a dramatic improvement in recall to **0.46**, almost a threefold increase. The model is now better at identifying such content, which is crucial in detecting harmful communications or activity.
- **"Fraud"** maintained a high recall of **0.94**, demonstrating that the model continues to perform well in this critical category.
- **"Vulnerable"** showed a notable improvement in recall to **0.99**, addressing the initial weakness seen in Version 1. This is essential for detecting vulnerabilities and minimizing risks.
- **"Intrusions"** also improved significantly to **0.89**, indicating that the model is now better at identifying intrusion attempts, which are often targeted threats.

- **"Information Gathering"** improved from **0.47** in Version 1 to **0.94** in Version 11, reflecting a better capability to identify security related alerts, which was a major challenge in the previous version.

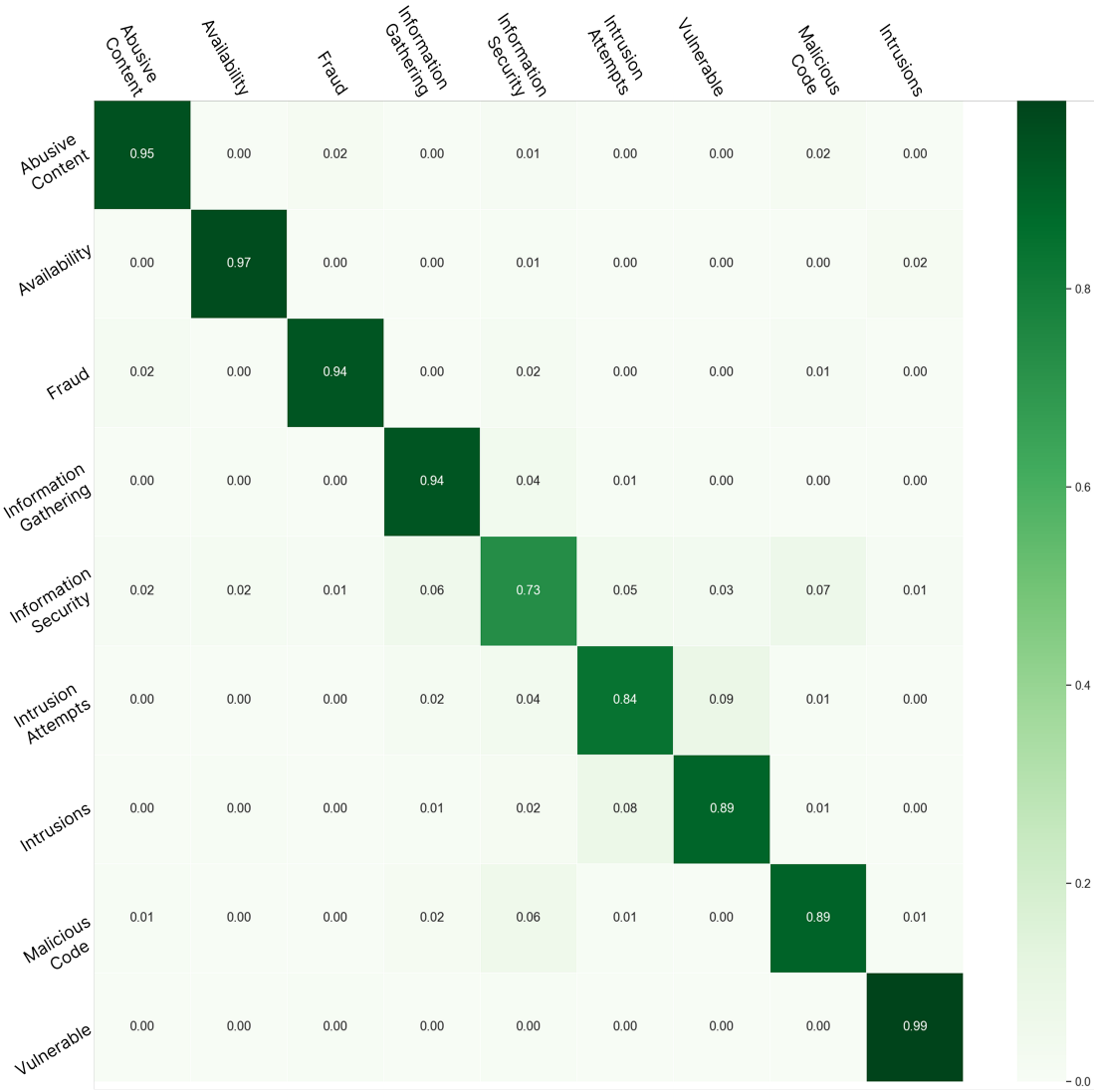

Figure 4.4: Confusion Matrix for Taxonomy (Version 11)

The improvements in recall across all categories in Version 11 indicate that the model is now more effective at identifying relevant alerts, particularly in low frequency categories. This is crucial for ensuring that the model can accurately classify and prioritize alerts, especially in cases where certain categories may have been previously overlooked or misclassified.

4.2 Live Testing Environment

The live testing environment aims to simulate real world deployment, where the models are evaluated based on their actual performance in the company's infrastructure. In this environment, both the RF and RL models are expected to be used. However, unlike the local environment where testing was done solely on the RF model, the live environment will

also involve continuous training of the RL model. As feedback is gathered, the RL model is expected to improve its performance over time.

For the live testing, the application will not be evaluated separately on the RF and RL models but will instead use both models as part of an integrated system. During this phase, feedback from real time predictions will be used to adjust and fine tune the RL model's decisions. The expectation is that with time, the RL model will show improvements in its ability to predict priority and taxonomy, surpassing the performance of the initial RF model version.

This section describes the deployment and validation of the proposed system in a real world setting, within the infrastructure of ArtResilia. The goal was to evaluate the performance and operational viability of the integrated solution under realistic conditions. This included the full deployment of the API, containerized services, integration with IBM QRadar SOAR via playbooks, and the subsequent observation and resolution of runtime constraints.

4.2.1 Deployment Architecture and Integration Workflow

Following the successful training of the RF model and implementation of the full API according to previously agreed specifications, a Swagger documentation interface was developed to facilitate integration with the security orchestration team. This interface described the available endpoints and their expected input/output structure.

The integration and deployment process involved several key steps:

- **Swagger Delivery:** The Swagger API documentation was sent to the responsible party at ArtResilia. This enabled the creation of a custom playbook within IBM QRadar SOAR to handle ticket classification and feedback submission.
- **VM Provisioning:** In parallel, a new virtual machine (VM) was prepared to host the deployed solution. Unlike the local development environment with 16GB of RAM, the testing VM had only 4GB of RAM. This difference significantly impacted the resource management strategy, particularly during RL model training.
- **Dockerized Deployment:** To ensure portability and simplify environment replication, Docker was used to containerize the application. This approach offers several advantages over native installation, including consistent runtime environments, isolation of dependencies, simplified updates, and ease of monitoring and scaling individual services. The deployed architecture consisted of four containers:
    1. **API and Bot Container:** Handles real time prediction requests and exposes the REST endpoints.
    2. **Celery Worker Container:** Executes background tasks such as RL model training based on analyst feedback.
    3. **Flower Monitoring Container:** Provides a web interface for monitoring Celery tasks and Redis queue status.
    4. **Redis Container:** Serves as the message broker between the API and Celery worker.
- **Shared Volume and Model Caching:** A shared Docker volume was mounted between the API and Celery containers to allow direct access to cached model files. This

ensured consistency when loading and updating model versions during feedback based training.

- **Embedding Model Caching:** Due to strict firewall configurations enforced via iptables, the VM had no internet access outside the ArtResilia VPN. Therefore, pre-trained models such as Sentence-BERT were bundled directly into the Docker image to prevent the need for online downloads during runtime.
- **Access Control via NGINX:** An NGINX reverse proxy was configured to provide external access to the API and the Flower dashboard. Basic HTTP authentication was enforced to restrict access to authorized users only.

4.2.2 SOAR Integration via Playbook

The integration with IBM QRadar SOAR was implemented through an asynchronous playbook, designed to avoid disruption of SOC workflows in case the system became unavailable. The playbook behavior can be summarized as follows:

- When a new ticket is created, a GET request is sent to the API to retrieve the predicted taxonomy and priority.
- These predictions are displayed in the SOAR web interface in dedicated fields labeled "Predicted Taxonomy" and "Predicted Priority".
- Upon ticket closure, the actual analyst defined values for taxonomy and priority are submitted to the API via a POST request to trigger the feedback learning mechanism.
- The playbook was configured to execute asynchronously. This ensures that even if the prediction system is temporarily offline, the SOAR platform continues to operate normally, skipping the classification step without interruption.

4.2.3 Runtime Challenges and Solutions

Upon initial deployment, the system began to operate successfully, processing classification requests for incoming tickets. However, problems emerged when the RL model attempted to train based on the feedback data.

Memory Problem The Celery container crashed due to memory exhaustion, consuming over 4GB of RAM plus all available swap space.

To address this problem:

- A new VM with 8 GB of RAM was provisioned, but the same crashes reoccurred during RL training.
- Since further increasing the VM's memory was not a sustainable solution, a custom memory management strategy was developed using Python's built in gc (garbage collector) module.

Solution - Custom Garbage Collection Strategy To reduce memory usage, a targeted cleanup process was applied at the end of each Celery training task. This was done by explicitly deleting large intermediate variables and invoking Python's garbage collector.

```
1 del {variable_name}
2 gc.collect()
```

Listing 13: Custom Garbage Collection Strategy

The code in Listing 13 performs explicit memory management in Python. Line 1 explicitly deletes a variable from memory, removing its reference and making it eligible for garbage collection. Line 2 then manually triggers Python's garbage collector to immediately reclaim memory from objects that are no longer referenced.

This strategy ensured that each training session released its memory footprint before the next one started. After implementing this fix, memory usage was stabilized and kept under 6GB, preventing further crashes and allowing uninterrupted operation.

4.2.4 Results and Observations

With the system stabilized, the final step was to let it run uninterrupted for a continuous period to gather performance metrics. During this period, the system operated in parallel with the standard SOC workflow, producing live predictions and processing analyst feedback asynchronously.

Accuracy and Learning Curves Over Time

Over the course of 15 days of uninterrupted live testing, a total of 519 alerts were processed. The model made predictions for each alert, and all of these predictions received analyst feedback. Based on this feedback, training tasks were dispatched to the RL model, enabling incremental learning.

In this evaluation, a prediction was considered "correct" only when both the priority and taxonomy classifications matched the analyst submitted values. The following figures used the data explained in appendix B to visualize the model's performance over time.

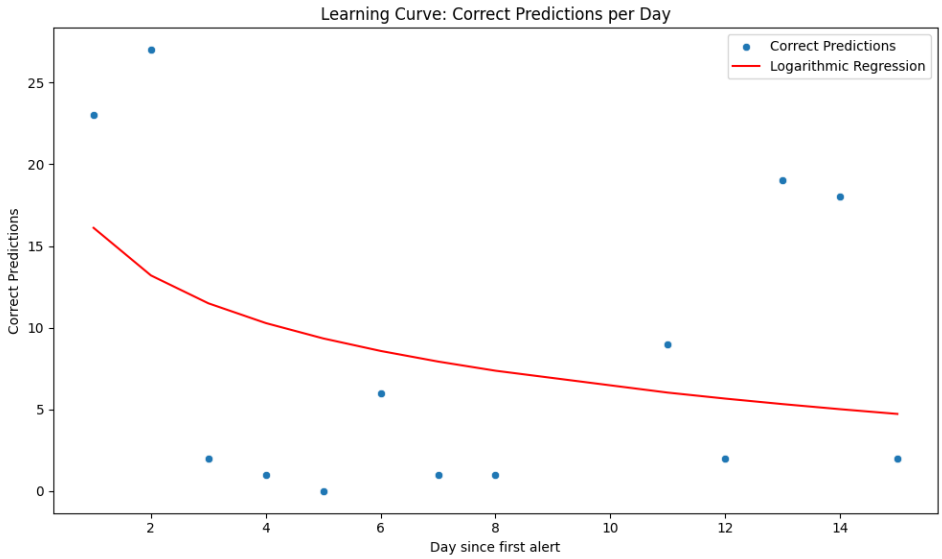

Figure 4.5: Correct Predictions per Day (Priority + Taxonomy)

Figure 4.5 shows the number of correct predictions made each day. While the model had strong performances on Day 0, Day 1, and Day 10, it also exhibited weaker results on Days 2 through 7. A logarithmic regression line was fitted to capture the general trend across the observation window. Rather than the expected upward curve that would suggest a growing influence of the RL model refining the baseline RF predictions, the trend displays a slight decline.

This result differs with the intended behavior of the system, where RL was expected to gradually assume control from the RF model, leveraging historical feedback to increase accuracy and confidence. Instead, the data suggests that within the limited 15 day window, the RL component had not yet reached sufficient knowledge to impact performance. Still, this graph offers valuable insight into how the hybrid RF+RL model performed in early deployment, and establishes a baseline for what an improving curve might look.

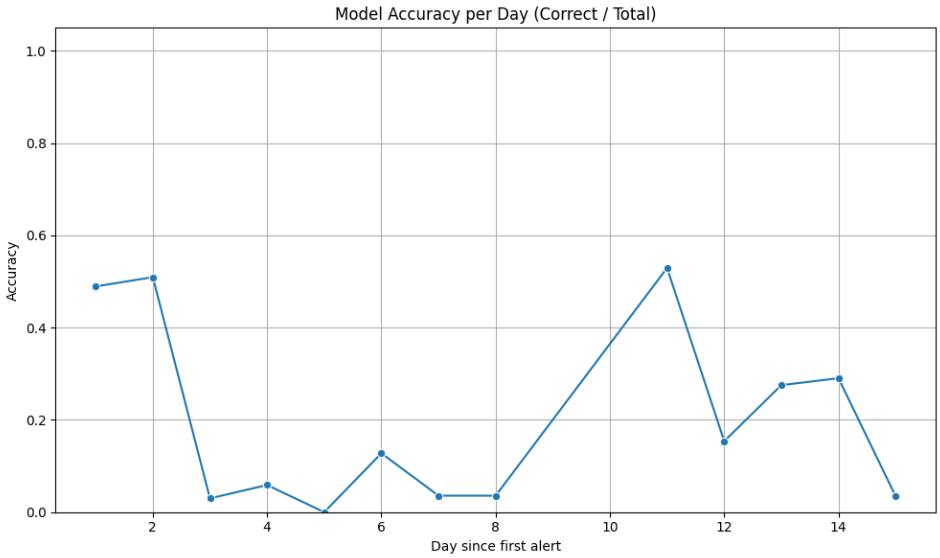

Figure 4.6: Accuracy per Day (Both Labels Must Match)

Figure 4.6 presents the model's daily accuracy, defined as the proportion of alerts for which both the predicted priority and taxonomy matched the analyst-submitted values. While this graph shows similar information to Figure 4.5, it provides a complementary perspective.

Whereas the previous figure illustrated the absolute number of correct predictions, this one accounts for the daily volume of processed alerts. For example, a day with only ten alerts might have a higher accuracy despite few total correct predictions, while a high volume day with modest accuracy may still result in many useful classifications. Together, these graphs offer a more complete picture of the system's behavior.

By observing both, one can distinguish between days where the system performed well due to higher precision versus days where volume may have tilted the perception of success. Notably, the decline seen in the accuracy curve aligns with the drop in absolute correct predictions, reinforcing the notion that performance was inconsistent during the early testing phase.

This pattern diverges from the ideal learning curve expected in RL environments. Ideally, as the RL model accumulates feedback, its corrections should incrementally improve classification outcomes producing an upward trend in both accuracy and correct predictions over time.

The likely cause is the relatively short evaluation window. In just 15 days of operation, the RL model lacked sufficient data to learn with confidence from past feedback. As a result, predictions remained largely influenced by the static RF model, which was trained offline and does not adapt over time.

Despite this, both images, not only document the initial performance of the RF+RL hybrid system, but also provide a benchmark for future testing. With extended operation and richer feedback history, it is expected that a more optimistic trajectory marked by steadily increasing accuracy will begin to appear.

Training Feedback via Celery and Flower Monitoring

The RL model was retrained based on feedback submitted by analysts through the SOAR playbook integration. Each task corresponded to a single training iteration using the feedback values of taxonomy and priority.

Figure 4.7: Flower Monitoring Dashboard showing Successful and Failed Training Tasks

As shown in Figure 4.7, a total of 212 training tasks were executed, of which 133 completed successfully and 79 failed. The failed tasks were due to unexpected formatting, inconsistencies in the taxonomy labels, such as trailing whitespaces, casing differences, or unexpected variants not previously seen by the model during training or in the training data set.

It is also important to note that although 519 tickets were processed, only 212 training tasks were recorded. This discrepancy is due to the fact that the Flower monitoring dashboard is only displaying tasks executed in the last 4 days of the test period, which corresponds to the deployment of the latest stable release of the system. As such, earlier training attempts are not visible in the current view but were processed during prior system iterations.

Evaluation of Overfitting and Underfitting Patterns

A key objective during evaluation was to determine whether the model exhibited signs of overfitting or underfitting, especially when exposed to live, real-world data that diverged from the training distribution.

All these results are detailed in Appendix B, Tables B.2 and B.3.

In the local test environment, both priority and taxonomy models achieved high levels of performance. Version 11 reached 90% accuracy on both tasks, with balanced precision and recall across most classes, even those with lower support. This indicated strong generalization within the boundaries of a dataset.

However, the transition to the live environment exposed several weaknesses that suggest underfitting in certain areas and potential overfitting in others. During live testing, the overall accuracy dropped to 47% for priority and 41% for taxonomy, with a pronounced decline in performance on underrepresented classes.

For priority classification, P1 tickets showed the weakest performance, with a recall of only 13%. This contrasts sharply with its recall of 91% in offline testing. Given that P1 tickets were rare both in the training set and in production, this drop indicates underfitting: the model was not sufficiently exposed to the characteristics of these alerts to generalize effectively.

In taxonomy classification, classes such as "Vulnerable" and "Information Gathering" received zero recall, suggesting the model could not recognize or generalize from the limited examples it had seen during training. This outcome suggests the oversampling techniques applied during local training were not sufficient to address the imbalance when faced with more diverse, noisy live data. These classes appear to have been memorized during training but failed to be correctly identified when presented in a different context, behavior typically associated with overfitting.

Conversely, taxonomy classes like "Malicious Code" and "Fraud", which had high representation in the training set and continued presence in the live data, maintained relatively stronger performance. This supports the idea that the model retained generalization capabilities for dominant classes but struggled to scale that performance to edge cases.

Overall, these results point to a mixed behavior: overfitting in low-support classes that failed to generalize outside of the training data distribution, and underfitting in critical classes like P1, which did not receive enough exposure during training to support accurate predictions in production. This highlights the need for more robust class balancing strategies, better augmentation of rare categories, and longer RL adaptation windows to mitigate the limitations of the static RF model and improve long-term generalization.

4.2.5 Final Analysis

The deployment and evaluation of the hybrid RF+RL model revealed both encouraging strengths and critical areas for improvement.

On the positive side, the system demonstrated the feasibility of deploying a containerized, real-time classification pipeline integrated with existing SOC workflows. The initial Random Forest model achieved high accuracy and recall in controlled environments, and the end-to-end feedback loop with IBM QRadar SOAR functioned as intended, enabling continuous learning via analyst input.

However, the transition to live data exposed several limitations. The most notable issues were a sharp drop in performance on minority classes, inconsistent feedback labeling, and insufficient time for the RL model to adapt. These limitations affected both precision and generalization, especially for rare or noisy categories.

To address these challenges, several strategies are proposed:

- Improve class balancing through more sophisticated data augmentation or synthetic generation of minority class samples.
- Normalize and sanitize feedback labels before training to reduce inconsistencies.
- Extend the live evaluation period to allow the RL model to accumulate sufficient training data and stabilize learning.
- Monitor per-class performance continuously to trigger targeted retraining when specific categories degrade.

The system's architecture and workflow proved operationally viable, but model generalization in production remains an open challenge requiring further refinement and sustained adaptation.

Chapter 5

Conclusion and Future Work

This chapter presents the main conclusions drawn from the research and proposes several directions for future work that could further enhance and extend the developed solution. The discussion reflects on the accomplishments of the current system while identifying areas for potential improvement and continued evolution.

5.1 Future Work

While the current solution shows what can be done in automating the security alert triage process, there remain several areas for future improvement and refinement.

First, one of the main limitations observed in the live environment was the system's reduced ability to generalize to underrepresented or noisy alert categories. Future iterations should therefore focus on enhancing dataset diversity and volume. In particular, having larger and more balanced datasets, either through long-term data accumulation or synthetic data generation, could significantly improve the mode's ability to generalize in production settings.

Secondly, the RL component requires a longer feedback window to become fully effective. Extending the live testing period and accumulating more high-quality, consistent feedback will allow the RL model to adapt its policy more accurately. Additionally, implementing stronger input validation and preprocessing for feedback, particularly to handle inconsistencies in label formatting—would improve the success rate of training tasks and reduce noise during learning.

Model explainability also remains an open area. While the system functions well as a black-box predictor, integrating interpretable ML tools such as SHAP or LIME could help analysts understand the rationale behind predictions. This would be especially important for justifying decisions in high priority cases and for refining model behavior in edge scenarios.

From an engineering perspective, future work could also include fine-grained monitoring of per-class performance in real time, enabling automated retraining triggers when class-specific accuracy drops below thresholds. Moreover, improvements in resource handling—such as memory-efficient training procedures and container orchestration—would further strengthen the system's robustness under constrained environments.

Finally, exploring ensemble strategies that dynamically balance the contribution of the RF and RL components based on confidence levels or historical performance could lead to better hybrid predictions. This would help the system adaptively prioritize models depending on alert complexity and feedback richness.

5.2 Conclusion

This work presented the design, implementation, and evaluation of an intelligent system for automating security alert triage, integrating supervised learning via a RF classifier with a RL agent capable of evolving based on analyst feedback. The system was successfully deployed within a live SOC environment, where it processed real alerts, adapted to feedback, and maintained integration with existing tools such as IBM QRadar SOAR.

In controlled local testing, the RF model demonstrated excellent classification accuracy and recall across both priority and taxonomy labels. In the live environment, the hybrid system showed initial promise, successfully processing hundreds of alerts while adapting to real-world constraints such as inconsistent data and limited resources. Although the RL model had not yet reached its full potential within the short evaluation window.

The successful deployment of a containerized architecture, the integration with SOAR platforms, and the implementation of asynchronous training workflows via Celery and Flower confirm the operational feasibility and practical value of augmenting SOC operations with intelligent, adaptive models.

While challenges remain—particularly in generalizing to minority classes and scaling learning over time, the progress demonstrated in this work offers a strong foundation for future development. With continued refinement, broader datasets, and longer deployment times, this approach has the potential to significantly reduce analyst workload, increase triage accuracy, and enhance the responsiveness of SOC operations to emerging threats.

This thesis concludes with the conviction that combining classical machine learning with reinforcement learning in real-time SOC workflows is feasible and promising towards a smarter, more autonomous cybersecurity system.

Bibliography

Ali, Gauhar, Sajid Shah, and Mohammed ElAffendi (Sept. 2024). *Enhancing Cybersecurity Incident Response: AI-Driven Optimization for Strengthened Advance Persistence Threat Detection*. doi: 10.20944/preprints202409.1725.v1. url: https://www.preprints.org/manuscript/202409.1725/v1.

Alto, Palo (2024). *Palo Alto Networks® Closes Acquisition of IBM's QRadar SaaS Assets - Palo Alto Networks*. url: https://www.paloaltonetworks.com/company/press/2024/palo-alto-networks--closes-acquisition-of-ibm-s-qradar-saas-assets.

Arianna, By (2024). *Cybersecurity Management Services: Why ITSM Services Essential?* url: https://forlicoupon.it/2024/07/17/cybersecurity-management-services-why-itsm-services-essential/.

ArtResilia (n.d.). *ArtResilia*. url: https://www.artresilia.com/.

Breiman, Leo (1996). "Bagging predictors". In: *Machine Learning* 24 (2), pp. 123–140. issn: 08856125. doi: 10.1007/BF00058655/METRICS. url: https://link.springer.com/article/10.1007/BF00058655.

Brogi, Guillaume and Valérie Viet Triem Tong (Dec. 2016). "TerminAPTor: Highlighting advanced persistent threats through information flow tracking". In: *2016 8th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2016*. doi: 10.1109/NTMS.2016.7792480.

Brownlee, N. and E. Guttman (June 1998). "Request for Comments 2350 Expectations for Computer Security Incident Response," in: url: http://www.ietf.org/rfc/rfc2350.txt.

Chandra, J. Vijaya, Narasimham Challa, and Sai Kiran Pasupuleti (Aug. 2016). "A practical approach to E-mail spam filters to protect data from advanced persistent threat". In: doi: 10.1109/ICCPCT.2016.7530239.

Chauhan, Nagesh (Feb. 2022). *Decision Tree Algorithm, Explained - KDnuggets*. url: https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html.

CheckPoint (n.d.). *What is SOC Automation? - Check Point Software*. url: https://www.checkpoint.com/cyber-hub/threat-prevention/what-is-soc/what-is-soc-automation/.

CriticalStart (2019). *THE IMPACT OF SECURITY ALERT OVERLOAD*. url: https://www.criticalstart.com/wp-content/uploads/2021/02/CS_Report-The-Impact-of-Security-Alert-Overload.pdf.

Crowley, Christopher and John Pescatore (2018). *The Definition of SOC-cess? A SANS Survey*.

Dasarathy, Belur V. and Belur V. Sheela (1979). "A composite classifier system design: Concepts and methodology". In: *Proceedings of the IEEE* 67 (5), pp. 708–713. issn: 0018-9219. doi: 10.1109/PROC.1979.11321. url: https://www.academia.edu/30910041/A_composite_classifier_system_design_concepts_and_methodology.

ElSahly, Osama and Akmal Abdelfatah (Aug. 2023). "An Incident Detection Model Using Random Forest Classifier". In: *Smart Cities* 6 (4), pp. 1786–1813. issn: 26246511. doi: 10.3390/SMARTCITIES6040083.

exabeam (2024). *Best SIEM Solutions: Top 10 SIEM systems and How to Choose | Exabeam*. url: https://www.exabeam.com/explainers/siem-tools/siem-solutions/.

Farooq, Hafiz M. and Naif M. Otaibi (Dec. 2018). "Optimal machine learning algorithms for cyber threat detection". In: *Proceedings - 2018 UKSim-AMSS 20th International Conference on Modelling and Simulation, UKSim 2018*, pp. 32–37. doi: 10.1109/UKSIM.2018.00018.

GeeksforGeeks (2024). *Top 10 Machine Learning Frameworks in 2025 - GeeksforGeeks*. url: https://www.geeksforgeeks.org/machine-learning-frameworks/.

Ghafir, Ibrahim et al. (July 2018). "Detection of advanced persistent threat using machine-learning correlation analysis". In: *Future Generation Computer Systems* 89, pp. 349–359. issn: 0167739X. doi: 10.1016/J.FUTURE.2018.06.055. url: https://bradscholars.brad.ac.uk/handle/10454/17614.

Giura, Paul and Wei Wang (2012). "A context-based detection framework for advanced persistent threats". In: *Proceedings of the 2012 ASE International Conference on Cyber Security, CyberSecurity 2012*, pp. 69–74. doi: 10.1109/CYBERSECURITY.2012.16.

Hámornik, Balázs Péter and Csaba Krasznay (2018). "A team-level perspective of human factors in cyber security: Security operations centers". In: *Advances in Intelligent Systems and Computing* 593, pp. 224–236. issn: 21945357. doi: 10.1007/978-3-319-60585-2_21. url: https://www.researchgate.net/publication/318177610_A_Team-Level_Perspective_of_Human_Factors_in_Cyber_Security_Security_Operations_Centers.

Hansen, Lars Kai and Peter Salamon (1990). "Neural Network Ensembles". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12 (10), pp. 993–1001. issn: 01628828. doi: 10.1109/34.58871. url: https://www.researchgate.net/publication/3191841_Neural_Network_Ensembles.

Harper, Allen et al. (Dec. 2010). *Security Information And Event Management (SIEM) Implementation*. Ed. by MCGRAW-HILL EDUCATION - EUROPE, p. 464. isbn: 9780071701099.

Huang, Xiang (July 2024). "Predictive Models: Regression, Decision Trees, and Clustering". In: *Applied and Computational Engineering* 79 (1), pp. 124–133. issn: 2755-273X. doi: 10.54254/2755-2721/79/20241551. url: https://www.ewadirect.com/proceedings/ace/article/view/13989.

Iamnitchi, Adriana et al. (2017). "An Anthropological Study of Security Operations Centers to Improve Operational Efficiency". In: url: https://digitalcommons.usf.edu/etd.

IBM (n.d.). *Integrations - IBM QRadar SOAR*. url: https://www.ibm.com/products/qradar-soar/integrations?utm_source=chatgpt.com.

Jalalvand, Fatemeh et al. (Nov. 2024). "Alert Prioritisation in Security Operations Centres: A Systematic Survey on Criteria and Methods". In: *ACM Computing Surveys* 57 (2), pp. 1–36. issn: 0360-0300. doi: 10.1145/3695462. url: https://dl.acm.org/doi/10.1145/3695462.

Joseph (Aug. 2022). *TensorFlow Ensemble Learning: What You Need to Know - reason.town*. url: https://reason.town/tensorflow-ensemble-learning/.

Kokulu, Faris Bugra et al. (Nov. 2019). "Matched and mismatched SOCs: A qualitative study on security operations center issues". In: *Proceedings of the ACM Conference on Computer and Communications Security*. Association for Computing Machinery, pp. 1955–1970. isbn: 9781450367479. doi: 10.1145/3319535.3354239.

Li, Nan, Yang Yu, and Zhi Hua Zhou (2012). "Diversity regularized ensemble pruning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 7523 LNAI (PART 1), pp. 330–345. issn:

03029743. doi: 10.1007/978-3-642-33460-3_27. url: https://www.researchgate.net/publication/267404624_Diversity_Regularized_Ensemble_Pruning.

Mienye, Ibomoiye Domor and Yanxia Sun (2022). "A Survey of Ensemble Learning: Concepts, Algorithms, Applications, and Prospects". In: *IEEE Access* 10, pp. 99129–99149. issn: 21693536. doi: 10.1109/ACCESS.2022.3207287.

Mohri, Mehryar, Afshin Rostamizadeh, and Ameet Talwalkar (2012). *Foundations of Machine Learning*. isbn: 978-0-262-01825-8. url: https://www.hlevkin.com/hlevkin/45MachineDeepLearning/ML/Foundations_of_Machine_Learning.pdf.

Moradi, Mehrdad, Bert Van Acker, and Joachim Denil (Feb. 2023). "Failure Identification Using Model-Implemented Fault Injection with Domain Knowledge-Guided Reinforcement Learning". In: *Sensors 2023, Vol. 23, Page 2166* 23 (4), p. 2166. issn: 1424-8220. doi: 10.3390/S23042166. url: https://www.mdpi.com/1424-8220/23/4/2166/htm%20https://www.mdpi.com/1424-8220/23/4/2166.

Muniz, Joseph, Gary McIntyre, and Nadhem AlFardan (2015). *Security Operations Center: Building, Operating, and Maintaining your SOC*. url: https://books.google.pt/books?hl=pt-PT&lr=&id=riraCgAAQBAJ&oi=fnd&pg=PT28&ots=SBNh-G3lCL&sig=6w_ipaEbsLDSFja8OQABKEBW7GE&redir_esc=y#v=onepage&q&f=false.

Nila, Constantin, Ioana Apostol, and Victor Patriciu (June 2020). "Machine learning approach to quick incident response". In: *2020 13th International Conference on Communications, COMM 2020 - Proceedings*, pp. 291–296. doi: 10.1109/COMM48946.2020.9141989.

Rutledge, Samantha (2024). *How SOC as a Service can help Sarah in Operations*. url: https://fractionalciso.com/how-soc-as-a-service-can-help-sarah-in-operations/.

Saini, Neeraj et al. (Dec. 2023). "A hybrid ensemble machine learning model for detecting APT attacks based on network behavior anomaly detection". In: *Concurrency and Computation: Practice and Experience* 35 (28), e7865. issn: 1532-0626. doi: 10.1002/CPE.7865. url: https://researcher.manipal.edu/en/publications/a-hybrid-ensemble-machine-learning-model-for-detecting-apt-attack.

Schapire, Robert E (1990). *The Strength of Weak Learnability*.

Sethi, Kamalakanta et al. (Dec. 2020). "A context-aware robust intrusion detection system: a reinforcement learning-based approach". In: *International Journal of Information Security* 19 (6), pp. 657–678. issn: 16155270. doi: 10.1007/S10207-019-00482-7.

Shaw, Megan (Aug. 2022). *Importance of SIEM in Supporting Digital Transformation Initiatives | DNIF*. url: https://www.dnif.it/en/blog/importance-of-siem-in-supporting-digital-transformation-initiatives.

Sheeraz, Muhammad et al. (Apr. 2023). *Effective Security Monitoring Using Efficient SIEM Architecture*. doi: 10.22967/HCIS.2023.13.023.

Shirey, R (2007). *Network Working Group*. url: http://tools.ietf.org/html/rfc4949.

Sneha, N. and Tarun Gangil (Dec. 2019). "Analysis of diabetes mellitus for early prediction using optimal features selection". In: *Journal of Big Data* 6 (1). issn: 21961115. doi: 10.1186/S40537-019-0175-6.

Sopan, Awalin et al. (May 2019). "Building a Machine Learning Model for the SOC, by the Input from the SOC, and Analyzing it for the SOC". In: *2018 IEEE Symposium on Visualization for Cyber Security, VizSec 2018*. doi: 10.1109/VIZSEC.2018.8709231.

Sovilj, Dušan et al. (2020). "A comparative evaluation of unsupervised deep architectures for intrusion detection in sequential data streams". In: *Expert Systems with Applications* 159, p. 113577. issn: 0957-4174. doi: https://doi.org/10.1016/j.eswa.2020.113577. url: https://www.sciencedirect.com/science/article/pii/S0957417420304012.

Sundaramurthy, Sathya Chandran et al. (Nov. 2014). "A tale of three security operation centers". In: *Proceedings of the ACM Conference on Computer and Communications Security* 2014-November (November), pp. 43–50. issn: 15437221. doi: 10.1145/2663887.2663904. url: https://www.researchgate.net/publication/265786529_A_Tale_of_Three_Security_Operation_Centers.

Suthaharan, Shan (2016). *Machine Learning Models and Algorithms for Big Data Classification*. Vol. 36. Springer US. isbn: 978-1-4899-7640-6. doi: 10.1007/978-1-4899-7641-3. url: https://link.springer.com/10.1007/978-1-4899-7641-3.

Tao, Lin (May 2018). "A DATA TRIAGE RETRIEVAL SYSTEM FOR CYBER SECURITY OPERATIONS CENTER". In.

Tines (2023). *Voice of the SOC 2023 Report*. url: https://www.tines.com/reports/voice-of-the-soc-2023/.

Vielberth, Manfred et al. (2020). "Security Operations Center: A Systematic Study and Open Challenges". In: *IEEE Access* 8, pp. 227756–227779. issn: 21693536. doi: 10.1109/ACCESS.2020.3045514.

Yang, Jianxin et al. (Nov. 2019). "Delineation of urban growth boundaries using a patch-based cellular automata model under multiple spatial and socio-economic scenarios". In: *Sustainability (Switzerland)* 11 (21). issn: 20711050. doi: 10.3390/SU11216159.

Zimmerman, Carson (2014). *Ten Strategies of a World-Class Cybersecurity Operations Center*. isbn: 9780692243107. url: www.mitre.org.

Appendix A

Raw Outputs: Version 1 vs. Version 11

This appendix presents the raw outputs of the models for Version 1 and Version 11, which were used for the evaluation and analysis in Chapter 4, Section 4.1. These outputs include the confusion matrices and classification metrics for both priority and taxonomy classifications.

Version 1 Performance Metrics

Priority Classification

Table A.1: Version 1 Priority Classification Metrics

Class	Precision	Recall	F1-Score	Support
P1	0.81	0.75	0.78	4162
P2	0.81	0.77	0.79	9374
P3	0.83	0.89	0.86	12462

- **Overall Accuracy:** 82%
- **Macro Average:** Precision 0.82, Recall 0.80, F1-Score 0.81
- **Weighted Average:** Precision 0.82, Recall 0.82, F1-Score 0.82

Taxonomy Classification

- **Overall Accuracy:** 82%
- **Macro Average:** Precision 0.81, Recall 0.65, F1-Score 0.69
- **Weighted Average:** Precision 0.82, Recall 0.82, F1-Score 0.82

Table A.2: Version 1 Taxonomy Classification Metrics

Class	Precision	Recall	F1-Score	Support
0	0.82	0.19	0.31	394
1	0.94	0.86	0.90	1117
2	0.95	0.95	0.95	5937
3	0.78	0.47	0.59	1066
4	0.72	0.84	0.78	6051
5	0.78	0.73	0.75	3467
6	0.66	0.54	0.59	1890
7	0.86	0.92	0.89	5988
8	0.74	0.36	0.49	88

Version 11 Performance Metrics

Priority Classification

Table A.3: Version 11 Priority Classification Metrics

Class	Precision	Recall	F1-Score	Support
P1	0.82	0.91	0.86	9511
P2	0.89	0.85	0.87	18294
P3	0.92	0.91	0.92	24287

- **Overall Accuracy:** 90%
- **Macro Average:** Precision 0.89, Recall 0.89, F1-Score 0.88
- **Weighted Average:** Precision 0.89, Recall 0.89, F1-Score 0.89

Taxonomy Classification

Table A.4: Version 11 Taxonomy Classification Metrics

Class	Precision	Recall	F1-Score	Support
0	0.95	0.95	0.95	5788
1	0.96	0.97	0.97	5788
2	0.96	0.94	0.95	5788
3	0.90	0.94	0.92	5788
4	0.78	0.73	0.76	5787
5	0.86	0.84	0.85	5788
6	0.87	0.89	0.88	5788
7	0.89	0.89	0.89	5788
8	0.95	0.99	0.97	5789

- **Overall Accuracy:** 90%

- **Macro Average:** Precision 0.90, Recall 0.90, F1-Score 0.90
- **Weighted Average:** Precision 0.90, Recall 0.90, F1-Score 0.90

Appendix B

Daily Prediction Accuracy During Live Testing

This appendix presents the daily statistics on model prediction accuracy during the 15-day live testing period described in Chapter 4.

The values were derived from a CSV file exported directly from the SOAR platform, containing feedback data for each alert processed by the system. The data includes the number of predictions where both the priority and taxonomy labels matched the values later submitted by the analysts (considered as "correct" predictions), as well as the total number of predictions made per day.

Table B.1: Daily Prediction Statistics During Live Testing

Day	Correct Predictions	Accuracy
0	23 / 47	48.9%
1	27 / 53	50.9%
2	2 / 67	3.0%
3	1 / 17	5.9%
4	0 / 9	0.0%
5	6 / 47	12.8%
6	1 / 28	3.6%
7	1 / 28	3.6%
10	9 / 17	52.9%
11	2 / 13	15.4%
12	19 / 69	27.5%
13	18 / 62	29.0%
14	2 / 58	3.4%

- **Overall Accuracy:** 47%
- **Macro Average:** Precision 0.41, Recall 0.36, F1-Score 0.36
- **Weighted Average:** Precision 0.45, Recall 0.47, F1-Score 0.44
- **Overall Accuracy:** 41%
- **Macro Average:** Precision 0.39, Recall 0.25, F1-Score 0.26

Table B.2: Live Priority Classification Metrics

Class	Precision	Recall	F1-Score	Support
P1	0.30	0.13	0.18	47
P2	0.41	0.28	0.34	209
P3	0.50	0.68	0.58	259

Table B.3: Live Taxonomy Classification Metrics

Class	Precision	Recall	F1-Score	Support
Availability	0.90	0.20	0.33	45
Fraud	0.82	0.44	0.57	126
Information Gathering	0.00	0.00	0.00	13
Information Security	0.25	0.71	0.37	94
Intrusion Attempts	0.26	0.29	0.27	62
Intrusions	0.38	0.15	0.21	40
Malicious Code	0.95	0.44	0.60	132
Other	0.00	0.00	0.00	2
Vulnerable	0.00	0.00	0.00	1

- **Weighted Average:** Precision 0.63, Recall 0.41, F1-Score 0.44