Linköping University | Department of Computer and Information Science
Master's thesis, 30 ECTS | Computer Science and Engineering
2025 | LIU-IDA/LITH-EX-A--25/114--SE

Improving SIEM Rules through Transformer-Based Rule Evasion Detection and Attribution

Erik Nordström
Hannes Widéen

Supervisor : Valency Colaco
Examiner : Simin Nadjm-Tehrani

External supervisor : Joakim Kahlström

LINKÖPING UNIVERSITY

Linköpings universitet
SE–581 83 Linköping
+46 13 28 10 00 , www.liu.se

Upphovsrätt

Copyright

Abstract

Every day, security analysts investigate potential breaches in organizations' digital infrastructure. Security Information and Event Management (SIEM) systems facilitate this by collecting and analysing security data in real time. Using analytics and automation, SIEMs help analysts detect threats and filter irrelevant information. At its core, SIEM systems rely on detection rules to analyse data. When a rule identifies a suspicious event, an alert is generated, drawing the attention of an analyst. These alerts form the basis of investigations, meaning that if an adversary evades a rule, a breach may go unnoticed.

To address this, we propose leveraging a transformer model to tokenize and embed executed PowerShell commands and PowerShell SIEM detection rules, enabling computation of cosine similarity between them. This allows for identification of evasions and determining which rules were evaded. Additionally, we introduce a regressor-based metamodel that selects the most suitable Large Language Model (LLM) from a pool and provides recommendations to improve rule coverage.

Using this approach, we found that at least 24% of the analysed rules in the 2025 SIGMA dataset were evadable. Our system detected 71% of hand-crafted evasions with a False Positive Rate (FPR) below 1% and correctly attributed 98% of the detected evasions to their corresponding detection rules, advancing the state-of-the-art system AMIDES. By selecting the most suitable LLM, the metamodel recommendations successfully updated 70% of the rules to detect previously unknown evasions. Additionally, security analysts, when presented with recommendations from our LLM-based metamodelling process reported that 31% of the recommendations could replace original rules with minor adjustments, while 47% offered key insights for manual refinement.

Acknowledgments

First and foremost, we would like to express our gratitude to Sectra Communications for the opportunity to conduct our thesis and for providing the necessary resources and support. It has been a true privilege to explore our topic of interest in an industry often defined by secrecy and legislative restrictions. A substantial part of our memorable experience is owed to our external supervisor, Joakim Kahlström, and colleagues across the Critical Infrastructure department. Their guidance, encouragement and shared laughter made writing our thesis both inspiring and much more enjoyable.

We also extend our gratitude to our supervisor, Valency Colaco, and our examiner, Simin Nadjm-Tehrani. Their continuous and constructive feedback throughout the thesis greatly helped us refine our work and achieve results far beyond our initial expectations.

Finally, this thesis would not have been possible without the support of close friends and relatives. During moments when motivation wavered, you reminded us of the purpose behind our efforts and what we were ultimately striving for. Though no names are mentioned, none are forgotten.

Contents

Abstract iii

Acknowledgments iv

Contents v

List of Figures vii

List of Tables viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Aim . . . 2
- 1.3 Research Questions . . . 3
- 1.4 Approach . . . 3
- 1.5 Contributions . . . 4
- 1.6 Limitations . . . 4

2 Background 5
- 2.1 Security Information and Event Management . . . 5
- 2.2 Machine Learning and Natural Language Processing Techniques . . . 7

3 Related Works 11
- 3.1 Security Monitoring and Detection Approaches . . . 11
- 3.2 Machine Learning Techniques . . . 14

4 Identifying SIEM Rule Evasions 16
- 4.1 Creating a Dataset . . . 17
- 4.2 Rule Pre-Processing . . . 20
- 4.3 Fine-Tuning the Transformer . . . 20
- 4.4 Creating a SIEM Evasion Detection Pipeline . . . 21
- 4.5 Evaluating the SIEM Evasion Detection Pipeline . . . 22

5 Improving Rule Coverage 23
- 5.1 Constructing a Meta-Modelling Regressor . . . 24
- 5.2 Evaluating Updated Rule Coverage . . . 26

6 Experimental Evaluation 27
- 6.1 Expert Insights . . . 27
- 6.2 Experimental Setup . . . 28
- 6.3 Detecting Rule Evasions . . . 30
- 6.4 Evaluation of Rule Evasion Detection and Attribution . . . 30
- 6.5 Results of the Rule Evasion Detection and Attribution . . . 31
- 6.6 Results of the Rule Coverage Improvements . . . 33

7 Discussion **35**
7.1 Results 35
7.2 Method 36
7.3 Limitations 38
7.4 Practical Implications 39

8 Conclusion and Future Works **41**

Bibliography **43**

Appendix **48**

List of Figures

1.1 Proposed approach. . . . 3

2.1 SIGMA detection component example . . . 6
2.2 Cosine similarity between two PowerShell commands. . . . 7
2.3 Multi-output random forest regressor inference. . . . 8
2.4 Confusion matrix. . . . 9

4.1 Methodology for identifying rule evasions. . . . 16
4.2 Generating and validating benign commands. . . . 17
4.3 Identifying evadable rules. . . . 19
4.4 Creating and validating evasions. . . . 20
4.5 Pre-processing rules. . . . 21
4.6 The SIEM evasion detection pipeline. . . . 22
4.7 Methodology for evaluating the SIEM evasion detection pipeline. . . . 22

5.1 Improving evaded SIEM rules. . . . 23
5.2 Prompt sent to LLMs. . . . 24
5.3 Scoring output from LLMs. . . . 25

6.1 Topology of the simulated enterprise network. . . . 29
6.2 Execution flow of the system. . . . 30
6.3 Rule analysis and classification. . . . 31
6.4 Receiver operating characteristic curve after fine-tuning. . . . 32
6.5 Detection outcomes with updated rules. . . . 34
6.6 Classification of updated rules. . . . 34

List of Tables

2.1 Metric intervals and value interpretations. 10

4.1 Mutation strategies for benign commands. 18
4.2 Used evasion techniques. 19
4.3 SIEM rule, matching command, and evading command. 19
4.4 Hyperparameters used for fine-tuning the transformer. 21

5.1 Average rule coverage improvement per suitability criteria. 25
5.2 Average suitability score per LLM. 25

6.1 Performance metrics for evasion detection. 31
6.2 Rule attribution accuracy at different similarity thresholds. 32
6.3 Detection performance comparison between our system and AMIDES. 33
6.4 Rule attribution accuracy comparison between our system and AMIDES. 33
6.5 Statistics on number of times each LLM was chosen to update a rule. 33

1 Introduction

As legislative frameworks, such as the European Union's NIS2 Directive[1] become adopted in Sweden, organizations are required to strengthen their resilience against cyberattacks. One way to contribute to this goal is through the implementation of Security Information and Event Management (SIEM) systems. To improve the effectiveness of SIEM, many vendors utilize Artificial Intelligence (AI) and Machine Learning (ML)[2,3]. However, the growing reliance on AI and ML in SIEM systems warrants closer examination, particularly in terms of the opportunities and constraints they present in real-world security operations. This chapter elaborates on the initial motivation while also outlining the research aim, research questions, methodological approach, key contributions, and limitations.

1.1 Motivation

In today's digital landscape, the rising number of cybersecurity threats to enterprise networks places sensitive information and critical operations at risk, with the potential to harm human lives [1]. When the conflict between Russia and Ukraine escalated in early 2022, multiple Ukrainian organisations fell victim to the WhisperGate malware [2]. WhisperGate displayed ransomware notes and encrypted files, underscoring the growing threats to organizations' digital infrastructure and highlights the urgent need for stronger measures to prevent data breaches and cyberattacks.

To mitigate the risk of similar or other attacks, many organisations adopt a defence-in-depth strategy, employing multiple layers of security controls to protect against a wide range of threats across the network [3], such as firewalls and access control systems. This layered approach offers redundancy so that if one security control fails, others remain in place to provide protection. In addition to security controls, enterprise networks also rely on threat detection systems to identify adversaries who have successfully breached their defences [4, 5]. If a breach occurs, these systems play a critical role in quickly identifying the intrusion, allowing organisations to assess its scope and respond accordingly.

[1] https://eur-lex.europa.eu/eli/dir/2022/2555
[2] https://www.elastic.co/elasticsearch/machine-learning
[3] https://www.splunk.com/en_us/blog/platform/what-is-splunk-doing-with-ai.html

One approach to implementing such passive detection capabilities is through SIEM systems [6]. They help organisations analyse network activity, detect potential security incidents and increase visibility into their digital infrastructure. SIEM systems primarily use rules based on misuse detection, designed to flag events that violate rules set in accordance with established security policies. This would raise an alert when, for example, a rule targeting unauthorised access attempts is triggered. However, while effective at identifying known attacks, many SIEM rules are still susceptible to evasive techniques that an adversary could exploit [7].

Security experts typically create SIEM rules using repositories like SIGMA[4], which is open source and continuously reviewed by professionals. Organizations may adapt these rules to their own environments and needs. When alerts are generated, they are reviewed and validated by security analysts, typically within a Security Operations Centre (SOC) [8]. However, since these rules are publicly accessible, adversaries can view and evade them by making slight modifications to commands while preserving their original malicious intent, thus avoiding detection. In addition, adversaries who employ Living-Off-the-Land (LOL) techniques leverage legitimate applications and native system tools for malicious purposes. PowerShell, which is a command-line shell and scripting language, is commonly used to abuse Windows binaries in LOL attacks through versatile commands [5,6]. Its flexibility reduces the need for adversaries to install additional tools and binaries on a compromised system. As a result, distinguishing benign from malicious activity becomes a significant challenge when deciding whether to raise an alert.

Various approaches, including statistical techniques, shallow machine learning models, and other deep learning methods, have been explored to detect LOL attacks in the form of PowerShell commands [9, 10, 11]. Integrating a transformer-based model with a SIEM tool could help address this challenge by analysing commands and detecting subtle evasions through their semantic meaning [12]. However, evaluating such models require representative data containing both benign activity and SIEM rule evasions, which is rarely available due to privacy restrictions. Furthermore, if the system could suggest improvements to evaded rules, it could strengthen defences and help prevent future cyberattacks.

1.2 Aim

This work aims to investigate how SIEM rule evasions can be detected when PowerShell commands are deliberately modified such that the resultant execution logs evade SIEM detection rules, while maintaining their underlying malicious intent. To enable a more extensive evaluation of evasion detection in this context, we also aim to improve data availability for representative problems within SOCs. The work further seeks to identify the specific SIEM rules that were evaded, and to recommend coverage improvements to the rules by leveraging machine learning to analyse the similarity between executed commands and existing detection rules. Coverage in the context of this thesis refers to the extent to which a detection rule successfully identifies the malicious behaviour it targets. Coverage improvement thus involves enhancing a rule so that it more comprehensively detects the targeted malicious behaviour.

[4]https://github.com/SigmaHQ/sigma
[5]https://www.cisecurity.org/insights/blog/living-off-the-land-the-power-behind-powershell
[6]https://redcanary.com/threat-detection-report/techniques/powershell/

1.3 Research Questions

The research questions that guide this thesis are formulated as follows:

RQ 1. How can machine learning be utilized to identify SIEM rule evasions?

In particular, in order to evaluate the success of an evasion detector:

- How can a representative dataset of benign activity and SIEM rule evasions be constructed?

RQ 2. How can machine learning improve detection coverage of evadable SIEM rules?

1.4 Approach

Figure 1.1 outlines the approach. To address the research questions, the approach was structured into a preliminary study followed by three main phases: Data and Rule Analysis, Development Phase and Evaluation Phase.

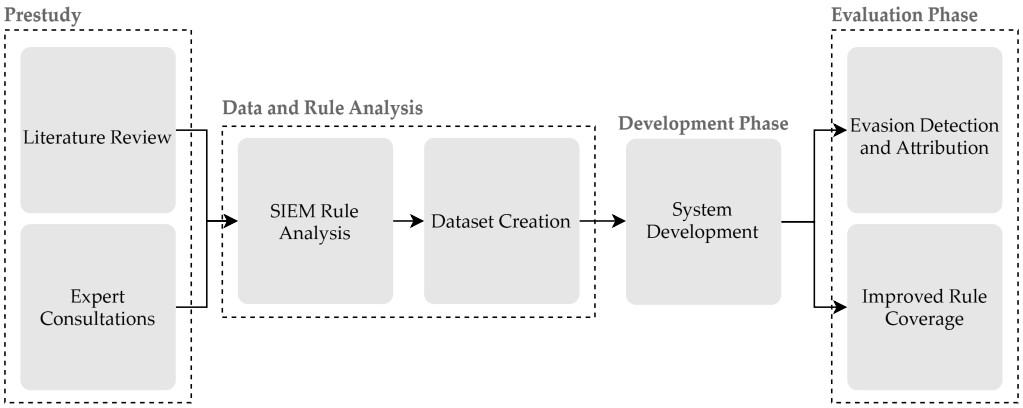


Figure 1.1: Proposed approach.

The prestudy involved a literature review aimed at exploring relevant research about the field and the state-of-the-art, along with consultations with security analysts from the company Sectra Communications to understand the challenges they face and the metrics they consider most useful for threat detection and incident analysis.

The Data and Rule Analysis phase was divided into two sub-phases, Rule Analysis and Dataset Creation. In the Rule analysis phase, PowerShell rules from the SIGMA rule set, released 2025-02-03 [7], were examined to determine which ones were evadable, based on set criteria. These criteria, which included evasion techniques such as substitution, alternative method and insertion, along with others are listed in Table 4.2. The dataset creation phase included leveraging evadable rules to craft evasions that were syntactically correct. A rule can be evaded in a number of ways, and multiple variations were added to the dataset. The benign subset consisted of standard commands that a system administrator is likely to execute on a daily basis.

[7] https://github.com/SigmaHQ/sigma/releases/tag/r2025-02-03

In the System Development phase, a simulated enterprise network consisting of virtual machines were configured. The network supported command logging via the ELK stack (Elasticsearch, Logstash, and Kibana), with Elastic Security serving as its SIEM tool. A transformer-based proof-of-concept system was also developed, aimed at identifying rule evasions using semantic similarity. In addition, it generates rule improvement suggestions that increase rule coverage through the use of LLMs.

During the evaluation phase, the proof-of-concept system was assessed based on its ability to detect rule evasions using various ML metrics. The results were then compared against the state-of-the-art system AMIDES [7]. Finally, the proposed rule improvements were evaluated by security analysts, who assessed whether the suggestions enhance detection capabilities and reduce evasion risks. The proposed rule improvements were also integrated into the SIEM tool to be evaluated on whether they successfully detected the previously identified evasions. Our criteria for success was met when a majority of the improvements were considered at minimum helpful, indicating practical value in strengthening the SIEM detection rules.

1.5 Contributions

The main contributions of this thesis are summarized as follows:

- A method to detect SIEM rule evasions based on transformer-guided semantic similarity comparisons.
- A rule-attribution method that, for each incoming SIEM evasion attempt, accurately identifies the predefined SIEM rules best aligned with that evasive behaviour.
- A method based on LLM meta-modelling to suggest rule improvements that increase coverage in the context of evasion attacks.
- A new set of SIEM rule evasions within the SIGMA repository, along with corresponding updated detection rules.

1.6 Limitations

Due to ethical concerns our set of evasions and updated rules will not be made freely available to the public. However, they can be shared with other researchers upon personal request.

2 Background

The reliability of rule-based SIEM detection in addition to Natural Language Processing (NLP) and ML techniques constitute the core of this research. Thus a brief overview is presented below to introduce the key concepts and terminology used in this thesis.

2.1 Security Information and Event Management

A SIEM system combines two functions, Security Information Management (SIM) and Security Event Management (SEM). SIM is responsible for collecting, storing, and analysing historical log data for investigations. SEM, on the other hand, focuses on real-time monitoring and event correlation to detect and respond to active security threats. Together, these functions make SIEM a unified platform for managing and analysing security events. This enables organizations to monitor their entire IT infrastructure [6], identify suspicious patterns or known threat signatures, and generate timely alerts [13]. Beyond detection, SIEM systems offer additional benefits. These include enhanced visibility into an organizations security posture, compliance support through detailed logging and reporting, and valuable insights for forensic analysis[1]. Some platforms also include automated response capabilities, such as blocking malicious IP addresses or isolating compromised devices, allowing security teams to act swiftly[2]. By providing centralized visibility and actionable intelligence through context-rich alerts, SIEM helps organizations strengthen their defences, meet compliance requirements, and reduce the impact of security incidents.

[1] https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-92.pdf
[2] https://www.splunk.com/en_us/blog/learn/siem-security-information-event-management.html

2.1.1 SIGMA

SIGMA[3] is an open-source initiative behind the SIEM Detection Format, a generic YAML rule format designed to be translatable to most widely used SIEM systems[4]. The project includes over 3000 detection rules, covering diverse log sources and attack categories, with severity levels ranging from low to critical. SIGMA rules can be automatically converted into formats supported by popular SIEM platforms, including Splunk Security[5], Microsoft Sentinel[6], and Elastic Security[7].

The SIGMA Detection Format consists of multiple components, the most important of which is the detection component [8]. It specifies the malicious behaviour that the rule is designed to identify. Each SIGMA rule's detection component consists of one or more selections, which are blocks containing search terms or conditions that the rule is looking for. These selection blocks can be combined using logical operators such as *and/or* to define more complex conditions. An example of a detection component in a SIGMA rule can be seen in Figure 2.1. In this case, the analysed log must contain all of *'powershell'*, *'nslookup'*, and *'[1]'*, as well as at least one of *'-q=txt http'* or *'-querytype=txt http'*, to generate an alert.

```
detection:
    selection:
        Data|contains|all:
            - 'powershell'
            - 'nslookup'
            - '[1]'
        Data|contains:
            - '-q=txt␣http'
            - '-querytype=txt␣http'
    condition: selection
```

Figure 2.1: SIGMA detection component example

2.1.2 ELK Stack

The ELK Stack is comprised of Elasticsearch, Logstash, and Kibana[9]. Each of these can be viewed as a tool on which the SIEM tool Elastic Security relies upon. Together they facilitate collecting, processing, indexing, searching, and visualizing large volumes of security related data. Agents, daemon processes running on hosts, collect data in the form of log files and send them to Logstash. Its main responsibility is to provide a processing pipeline that deduplicates, aggregates and processes data received from agents for ingestion into Elasticsearch. Once the data is transformed, it is indexed and stored in Elasticsearch, which serves as a search engine. Finally, Kibana provides the end user with a visualization interface to explore and analyse the data indexed in Elasticsearch.

[3]https://github.com/SigmaHQ/sigma
[4]https://sigconverter.io/
[5]https://www.splunk.com/en_us/products/enterprise-security.html
[6]https://learn.microsoft.com/sv-se/azure/sentinel/overview
[7]https://www.elastic.co/cloud/cloud-trial-overview/security
[8]https://sigmahq.io/docs/basics/rules.html
[9]https://www.elastic.co/docs/get-started/the-stack

2.2 Machine Learning and Natural Language Processing Techniques

The tools developed as a result of this thesis rely on both Ensemble Learning (EL) and Deep Learning (DL). Specifically, shallow EL is employed in the form of a Random Forest (RF) regressor, while DL is utilized through transformer architectures. In the following, we provide descriptions of these techniques and the metrics used to evaluate their performance.

2.2.1 Transformer

Under the hood, a transformer is essentially a deep neural network based on self-attention. It is designed to handle sequential data such as natural language sentences, where the order and context are important [14]. During training, input is converted into tokens that are embedded into vectors. Through backpropagation, the mapping between tokens and embedding vectors is learned.

During inference, the input tokens are mapped to their learned embedding vectors, which are enriched with contextual meaning as they pass through the transformer's layers. The objective is usually to find items similar to a specific query using a metric, such as cosine similarity. Additionally, tasks like embedding-based search can be performed. In this case the transformer generates one embedded vector for each input. In order to quantify the similarity between the vectors, their cosine similarity can be calculated. When negative values are disregarded, the resulting numerical value lies within the interval [0, 1]. A conceptual example of this is illustrated in Figure 2.2, although in practice, many more dimensions are used. In the figure, $x_n^{(0)}$ denotes embedded input tokens before passing through any transformer layers, whereas $x_n^{(L)}$ represents the output after all layers (L) have been passed.

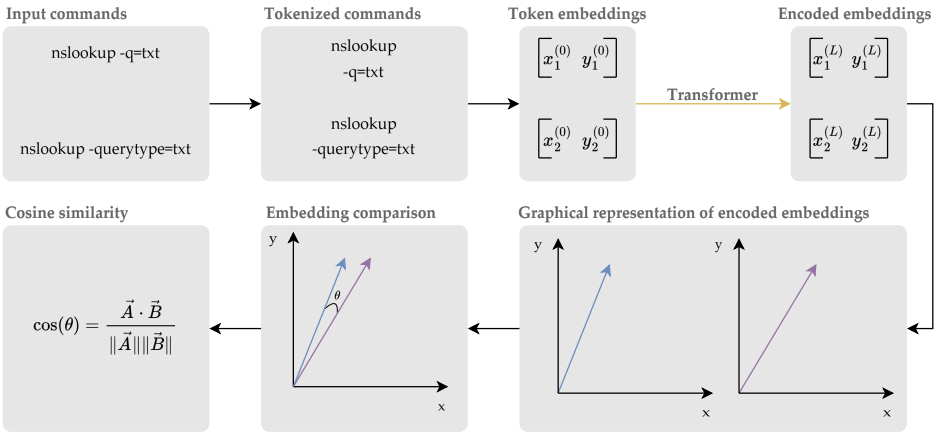


Figure 2.2: Cosine similarity between two PowerShell commands.

2.2.2 Random Forest Regressor

The main idea behind a RF regressor is to train an ensemble of decision trees that collaboratively predict numerical values by averaging their individual outputs [15]. A decision tree is a single predictive model that generates outputs by recursively splitting the input data based on feature values.

During training, bootstrapping is applied to generate random subsets of the original data with replacement, ensuring each tree is trained on a slightly different version of the dataset. Additionally, at each decision node, only a random subset of features is considered instead of the entire set.

This randomness helps decorrelate the trees, allowing them to grow deep while maintaining diversity. At each branching node, it selects a feature and threshold to split the data into two segments, recursively partitioning the data into increasingly smaller regions. Each leaf node stores a numerical value typically calculated as the average of all the target values of training samples that fell into that leaf during training. Once the trees are trained, inference is performed by running a new input through every tree and averaging their outputs to obtain the final prediction. An inference example of a multi-output random forest regressor can we viewed in Figure 2.3.

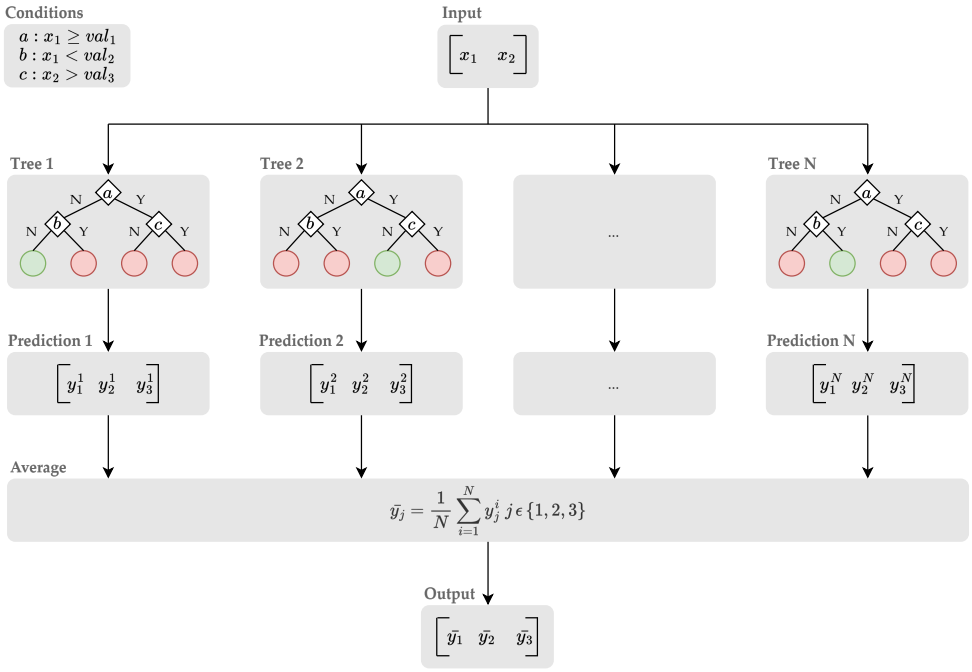


Figure 2.3: Multi-output random forest regressor inference.

2.2.3 Metrics

Evaluating ML models does not only require knowledge of which metrics to consider, but also understanding their significance in the context of the given task or application. In the area of supervised classification, the confusion matrix (see Figure 2.4), provides the foundation on which most metrics are calculated from.

The matrix encapsulates the relationship between a classifier's predictions and the ground truth labels. When a classifier correctly classifies a sample from the positive class, it is considered a True Positive (TP). Conversely, if a sample from the positive class is incorrectly classified as negative, it is considered a False Negative (FN).

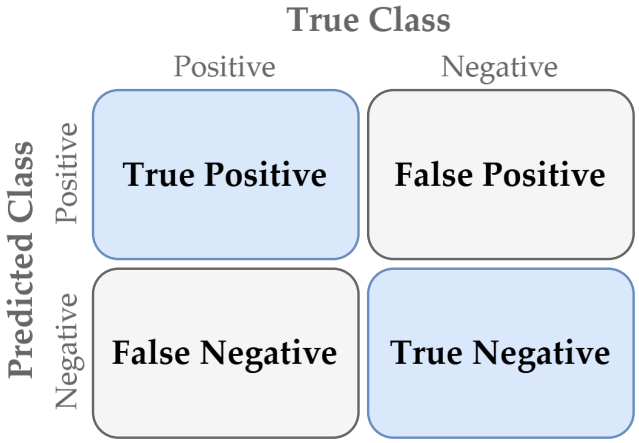


Figure 2.4: Confusion matrix.

From the confusion matrix, the True Positive Rate (TPR), False Positive Rate (FPR) and Accuracy (A) be computed according to Equations 2.1, 2.2 and 2.3.

$$TPR = \frac{TP}{TP + FN} \tag{2.1}$$

$$FPR = \frac{FP}{FP + TN} \tag{2.2}$$

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

For a binary classifier, the Receiver Operating Characteristic (ROC) curve can be plotted by varying the discrimination threshold [16]. The x-axis of the ROC curve represents the FPR, while the y-axis represents the TPR, both computed at each threshold. The Area Under the Curve (AUROC) is then estimated by applying the composite trapezoidal rule[10], as illustrated in Equation 2.4.

$$\int_b^a f(x)\,dx \approx \frac{1}{2}\sum_{i=0}^{n-1}(x_{i+1} - x_i)\,(f(x_i) + f(x_{i+1})) \tag{2.4}$$

Precision (P) measures how many of the positive labels predicted by the model are truly positive. Recall (R), on the other hand, measures the proportion of actual positive cases that the model identifies. In other words, out of all the truly positive cases, what fraction did the model actually catch?

Precision and recall can be combined into a single metric called F1-score, which provides a balanced evaluation when both False Positives (FP) and FNs matter [17]. Specifically, the F1-score is the harmonic mean of precision and recall. The formulas for precision, recall, and F1-score are provided in Equations 2.5, 2.6 and 2.7.

$$P = \frac{TP}{TP + FP} \tag{2.5}$$

$$R = \frac{TP}{TP + FN} \tag{2.6}$$

$$F1 = \frac{2 \times P \times R}{P + R} \tag{2.7}$$

[10]https://www.kth.se/social/files/56a4f0e2f27654626ce0020f/Ant-Integration.pdf

The observant reader may notice that the F1-score does not take True Negatives (TN) into account, which can lead to misleading results when the classes are highly imbalanced. In contrast, the Matthews Correlation Coefficient (MCC) considers all four components of the confusion matrix and provides a balanced assessment at a specific discrimination threshold, whereas the Area Under the Receiver Operating Characteristic (AUROC) summarizes performance across all thresholds [18]. The formula for MCC is provided in Equation 2.8, while both the intervals and value interpretations for selected metrics are summarized in Table 2.1.

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \tag{2.8}$$

Table 2.1: Metric intervals and value interpretations.

Metric	**Interval**	**Completely Incorrect**	**Random Guessing**	**Perfect Classifier**
Accuracy	[0, 1]	0	0.5	1
F1-score	[0, 1]	0	0.5	1
AUROC	[0, 1]	0	0.5	1
MCC	[-1, 1]	-1	0	1

3 Related Works

To the best of our knowledge at the time of writing this thesis, no prior research has focused on improving the coverage of SIEM rules in the context of preventing evasions. However, recent work by Uetz et al. [7] explores how SIEM rules from the publicly available SIGMA rule set can be evaded. Of the 292 SIGMA rules analysed, 110 were found to be fully evadable, 19 partially evadable, and 51 were considered potentially evadable. To address this issue, the authors propose a proof-of-concept system called Adaptive Misuse Detection System (AMIDES[1]). AMIDES is a tool for detecting when SIEM rules are being evaded. AMIDES successfully detected 70% of their crafted SIGMA-rule evasions (358 out of 512) at zero FPR. Our work improves upon this by not only detecting rule evasions but also providing improvement suggestions for SIEM rules to increase their coverage.

Augmenting this line of research, Huang et al. [12] introduce a method for generating a dataset consisting of semantically similar pairs of Windows and PowerShell commands. Using this dataset, they train three transformer-based models, CMDCaliper-small, CMDCaliper-base and CMDCaliper-large, built on the transformers created by Li et al. [19], capable of identifying semantic similarities between commands. Their results show that these models are able to capture semantic relationships beyond simple lexical similarities for PowerShell and Windows commands. In our work, we leverage one of the CMDCaliper transformer models as a foundation and further fine-tune it for our specific task, which focuses on detecting SIEM rule evasions and enhancing rule coverage. To understand the broader context of how SIEM rules are created and applied, the following section focuses on existing detection rules and systems.

3.1 Security Monitoring and Detection Approaches

Bhatt et al. [6] provide a good overview of why and how SIEM systems are used by SOCs and Teuwen et al. [20] presents a study on the trade-off between specificity and coverage. As noted by Teuwen et al., SOCs can observe a large number of alerts over a small time-period. Therefore it is important that detection rules are specific enough to only generate meaningful alerts while still providing sufficient coverage, thus balancing the amount of false positives with false negatives.

[1] https://github.com/fkie-cad/amides

The authors analysed network traffic data and created six security principles aimed at reducing the workload for analysts; four of these are particularly relevant to our thesis. The first focuses on determining whether a malicious action has been successful; that is, whether an adversary was able to turn their attack attempt into an actual compromise. In practice, detection rules should be designed to trigger only on successful malicious actions. However, for critical rules, it can be advantageous to also trigger on unsuccessful attack attempts. The second principle is alert throttling, which limits noisy rules to generating only a single alert within a specified time window. This helps reduce alert fatigue, a situation in which analysts are overwhelmed by the sheer number of alerts [21]. The third emphasizes the need for adding exceptions to rules, encouraging the tuning of rules that may be triggered by benign activity under certain conditions to reduce false positives. If a rule is triggered by a particular benign software, an exception for that software alone can be added to reduce false alerts. In the final fourth principle, the authors recommend defining a rule broadly such that it captures a wide range of variations in known malicious behavior.

Holm and Hyllienmark [22] conduct an empirical study on antimalware evasion techniques, evaluating their effectiveness against two antivirus tools. In total, 29,504 tests were carried out using 11 evasion techniques, generated by 16 different antimalware evasion tools. Out of all tests, 54% evaded antimalware scans, and 5% even led to an interactive shell. Their results further showed that packers, AES/RC4 encryption, and common data formats helped bypass scans, while custom encryption and code injection were the most effective for achieving shell access. Interestingly, a number of evasion techniques increased the likelihood of being detected by the antimalware tools.

3.1.1 Misuse Detection

Misuse-based detection approaches focus on detecting malicious activity by leveraging prior knowledge of known threats, such as labeled commands, scripts, or behaviours. These methods flag activities that match known attack patterns. This is closely related to detection strategies commonly used in SIEM systems, which often rely on known attack patterns to identify potential threats. None of the works listed below employed a transformer-based approach for detecting malicious activity, distinguishing our work from theirs.

Ryan Stamp [10] developed a novel algorithm to detect LOL attacks. The training and test data consisted of benign commands from a partner organization and malicious Living-Off-The-Land Binaries (LOLBin) commands gathered from the LOLBAS[2] project. The validation dataset consisted of 83 malicious LOLBin commands collected from The DFIR Report[3], with a total of 16 LOLBins selected for the study. With the usage of NLP techniques and supervised learning, the model outperforms Windows Defender in the validation dataset. The model detected all 83 malicious LOLBin commands, while Windows Defender detected only 8.

Ongun et al. [9] focus on detecting and labelling LOL attacks using an ensemble classifier. The classifier is trained on command-line representations generated using vectorized embeddings of the commands. Additionally, the authors incorporate a token scoring method to enhance detection performance. They propose an active learning framework called Living-Off-The-Land detection with Active Learning (LOLAL), specifically designed for identifying LOL attacks. Their approach achieves a high F1-score of 96%, demonstrating the effectiveness of the proposed framework.

[2]https://lolbas-project.github.io/
[3]https://thedfirreport.com/

Hendler et al. [11] use Deep Neural Networks (DNN) to detect malicious PowerShell commands. Their dataset consists of a large collection of distinct PowerShell commands, with the benign portion collected from Microsoft's enterprise network and the malicious portion created by Microsoft Security experts. After pre-processing, both DL-based detectors and traditional NLP-based detectors were trained, delivering promising results. The best performing model, which combines DL and traditional NLP, achieved 89% TPR, with an FPR of 0.1% on the test set.

Li et al. [23] create a de-obfuscation and semantic-aware attack detection system capable of detecting malicious scripts that uses several obfuscation techniques to evade detection. Using a collection of obfuscated benign and malicious PowerShell scripts, and when applying their de-obfuscation method in combination with Windows Defender and Virus Total, the detection rates increase from 0.3% and 2.65%, to 75% and 90%, respectively. Further, using their de-obfuscation method together with their attack detection system, they achieve 92.3% TPR with a 0% FPR, outperforming both Windows Defender and Virus Total together with the de-obfuscation method.

In [24], Fang et al. develop a method for detecting malicious PowerShell scripts using a hybrid feature extraction approach. It combines manually engineered features with automatically learned ones. From a dataset of benign and malicious PowerShell scripts, they construct an Abstract Syntax Tree (AST) and extract features that convey malicious behaviour, Indicators of Compromise (IOCs) based on special variables, shell code in scripts, etc. They test the performance using multiple classifiers, such as Support Vector Machine (SVM) and RF [15]. Their approach achieves 97.76% accuracy on a complex, obfuscated dataset and 98.93% on the original scripts, demonstrating its utility in both scenarios.

Alsaheel et al. [25] in their study create a framework called ATLAS that creates an end-to-end attack story constructed from different audit logs. They attempt to correlate individual attack instances to detect Advanced Persistent Threats (APTs). ATLAS uses a combination of causality analysis, NLP, and ML techniques to identify attack sequences. Their evaluation on real-world APT scenarios demonstrated that ATLAS achieves high accuracy, with over 91% precision, 97% recall, and a 94% F1-score in reconstructing attack steps, providing security analysts with an effective tool to uncover complex attacks.

3.1.2 Anomaly Detection

In contrast to misuse-based detection, anomaly detection identifies behaviour that deviates from a learned baseline. Since anomaly detection is related to misuse detection, techniques from both areas provide insights into identifying deviations and malicious behaviours, which are relevant for evasions. However, since these approaches learn a baseline, they are susceptible to adversaries who mimic normal behaviour by exploiting legitimate applications and native system tools for malicious purposes, a limitation that our approach tries to overcome. Another common problem with an anomaly-detection-based Intrusion Detection Systems (IDS) is the high number of false positives [8], as all behaviour that deviates from the established baseline is flagged as anomalous. While anomaly detection is intended to identify novel attacks, evaluating how well these systems perform on previously unseen attacks remain challenging in practice [26].

Vinay and Mangal [27] in their paper create the SCADE framework used to detect malicious activities in high-performance systems, such as data centers. By taking the executed commands as input, SCADE tries to determine the presence of any malicious activity. SCADE is trained on unlabelled normal executed command data. Their result shows that SCADE is capable of identifying anomalous command-line commands and unveiling hidden risks; for example, it flagged two processes created for testing over a year ago. SCADE also improves operational efficiency by reducing the manual effort needed to analyse command-line activities.

NoDoze, developed by Hassan et al. [28], is a tool intended to complement existing threat detection systems by using historical and contextual information to create causal dependency graphs. Each edge in the graph is assigned an anomaly score based on how frequently related events have occurred. The scores propagate along neighbouring edges to produce an aggregated score used for triaging alerts. The sub-graph with the highest anomaly score is extracted and deemed to be an adversaries true path. Experimental evaluations on 1 billion events containing 364 threats alerts showed a significant reduction of false alerts by 84%.

3.1.3 Incident Triaging

By combining SIEM data with attack goals, Sadlek et al. [29] utilized a graph generator to create attack paths. An attack path describes a chain of events representing a possible cyber threat scenario. Using SIEM alerts, evidence paths were created in the same manner. A matching algorithm then compared the evidence paths to the attack paths, more specifically the maximum ratio of the count of alerts found in some attack path to the length of the evidence path was computed. Ultimately, if an evidence and attack path matched, a sequence of attacks could be labelled with a severity level, description, ATT&CK tactic and asset. The AUROC score was evaluated to be 0.95, this approach thus seems promising in regards to improving security incident management.

3.2 Machine Learning Techniques

ML has become an essential tool in cybersecurity for detecting patterns and anomalies in large volumes of data. In the area of cybersecurity, finding high-quality datasets is difficult due to security and privacy concerns. Muthalagu et al. [30] highlight the risk of poisoning attacks, when an adversary can affect the input data on which the models are trained. There are also many common pitfalls when utilizing ML in cybersecurity, as presented by Arp et al. [31], such as an imbalanced dataset, which can lead to an overestimation of performance. As Bender et al. [32] argue in the context of LLMs, deploying powerful models without attention to data context, provenance and downstream effects (i.e., consequences of how outputs are used), can pose security risks. This is especially important in cybersecurity, where such risks can directly impact organizations through compromised systems and/or flawed decision-making.

The following subsection outlines research leveraging semantic similarity, built upon transformer architectures like Bidirectional Encoder Representations from Transformers (BERT) [33]. In this thesis, we adopt a transformer-based approach to detect SIEM rule evasions by identifying executed PowerShell commands that are semantically similar to SIEM rules.

3.2.1 Semantic Similarity

Devlin et al. [33] introduce BERT, a transformer-based large language model, which has been explored in several studies [34, 35, 36]. In [19], Li et al. create GTE, a general-purpose text embedding model trained with contrastive learning that leverages BERT to generate text embeddings suitable for a wide range of NLP tasks, such as zero-shot classification, unsupervised text retrieval, and code search. The authors develop three variants: GTE-base, GTE-small, and GTE-large. Notably, GTE-base outperforms OpenAI's black box embedding API and surpasses embedding models that are 10 times larger on the Massive Text Embedding Benchmark (MTEB)[4].

[4]https://huggingface.co/blog/mteb

Schneider et al. [37] utilize semantic analysis to group similar findings from reports generated by various security tools. This approach helps to consolidate duplicate vulnerability reports, allowing security analysts to recognize instances of the same vulnerability across multiple reports, reducing redundant analysis efforts.

Lastly, Yang et al. [38] presented a Successful Web Injection Detection Engine (SWIDE), a semantic-aware engine that can identify successful web injection attacks. SWIDE can identify web injection attacks even with obfuscations and uses semantic analysis to recover the intention of the attack.

3.2.2 Evasion Attacks

Evasion attacks aim to bypass detection without altering underlying behaviour. The following studies represent related work on evasion attacks and mitigation strategies, relevant to detecting SIEM rule evasions.

Mezawa et al. [39] demonstrate that evasion attacks can significantly degrade the performance of ML-based malicious PowerShell detection systems. They attempt to evade detection by modifying malicious scripts to incorporate features from benign samples, thereby preserving script functionality while circumventing detection. The authors tested their evasion scripts against various NLP techniques (Doc2Vec [40], Bag-of-Words (BoW) [41], and Latent Semantic Indexing (LSI) [42]) combined with ML classifiers such as RF [15] and XGBoost [43]. Their results show that the model using LSI was the most resilient to evasion attacks, exhibiting only a slight degradation in performance, while the model using Doc2Vec proved to be the most susceptible. Our method focus on detecting subtle evasion attempts that reduce similarity, whereas theirs targets malicious scripts padded with benign features.

Verma and Swami [44], as well as Song et al. [45], drew inspiration from coding theory to design a DNN with enhanced resistance to adversarial attacks, including evasion attacks. Their DNNs employ Error-Correcting Output Codes (ECOS) to prevent misclassification of inputs that an adversary has permuted without altering the semantics. Each class is assigned a unique binary codeword, with every bit representing a smaller binary classification task. The final classification is determined by comparing the predicted binary outputs to the closest codeword using Hamming Distance (HD). The authors evaluated their DNNs on image datasets and found significant improvements in classification accuracy during both black-box and white-box testing. More specifically, the best model achieved up to 0.858 benign accuracy on CIFAR10[5]. The accuracy under adversarial attacks ranged from 0.536 (Projected Gradient Descent) to 0.710 (Carlini–Wagner), and was 0.736 under noisy inputs (Gaussian).

[5]https://www.cs.toronto.edu/~kriz/cifar.html

4 Identifying SIEM Rule Evasions

To understand how SIEM rules can be evaded, it is essential to comprehend how they are applied in real-world threat detection and incident response. Therefore, a literature review was conducted to explore existing research, identify current gaps, determine relevant metrics, and review ML models capable of detecting SIEM rule evasions. In parallel, expert insights were obtained by inquiring into the challenges, best practices, and considerations that security analysts at Sectra Communications face when configuring, deploying, and fine-tuning rules in their daily operations. A summary of these insights is provided in section 6.1, and the full transcripts, notes taken during the consultations and later refined, along with the prompts can be found in the appendix. The methodology for identifying SIEM rule evasions is outlined in figure 4.1. The first step was to identify evadable SIEM rules and construct a dataset consisting of benign commands and evasive commands targeting those rules.

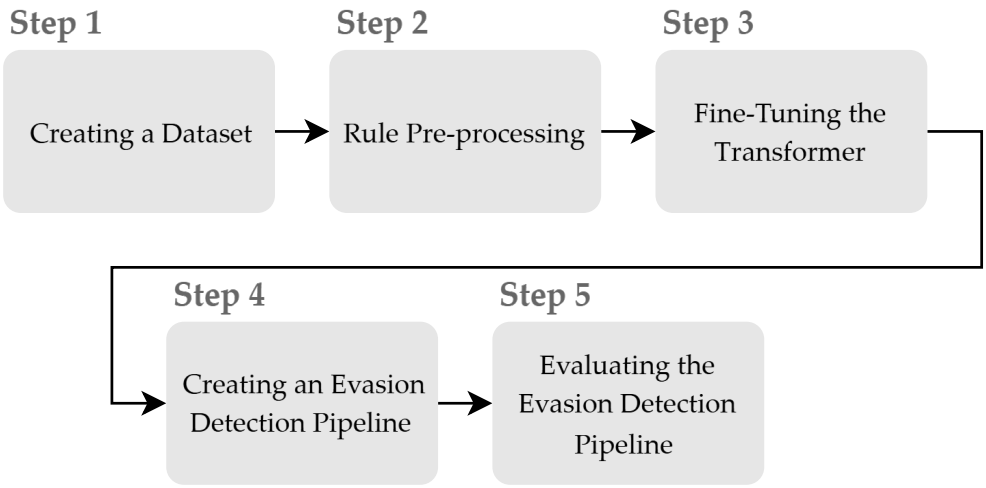


Figure 4.1: Methodology for identifying rule evasions.

4.1 Creating a Dataset

In order to measure our systems ability to detect rule evasions and improve coverage of rules, a dataset was needed in order to test and evaluate the system. In the cybersecurity domain, datasets are typically kept private since they include sensitive information, thus one was created from scratch. In accordance with Uetz et al. [7], the dataset was divided into 4500 benign and 500 evasion commands. The dataset had to be created from scratch, as the evasions presented by Uetz et al. [7] date back to 2021 and their benign data is not publicly available, which removed the possibility of directly reusing their dataset.

4.1.1 Benign Commands

In this work, benign commands are defined as standard PowerShell commands executed by system administrators without malicious intent. PowerShell has evolved through multiple releases, significantly expanding the capabilities of its command-line interface and scripting language. These expansions have resulted in an extensive array of available commands, creating a very large command set. Therefore, the method illustrated in Figure 4.2 was adopted to systematically generate valid and relevant benign commands.

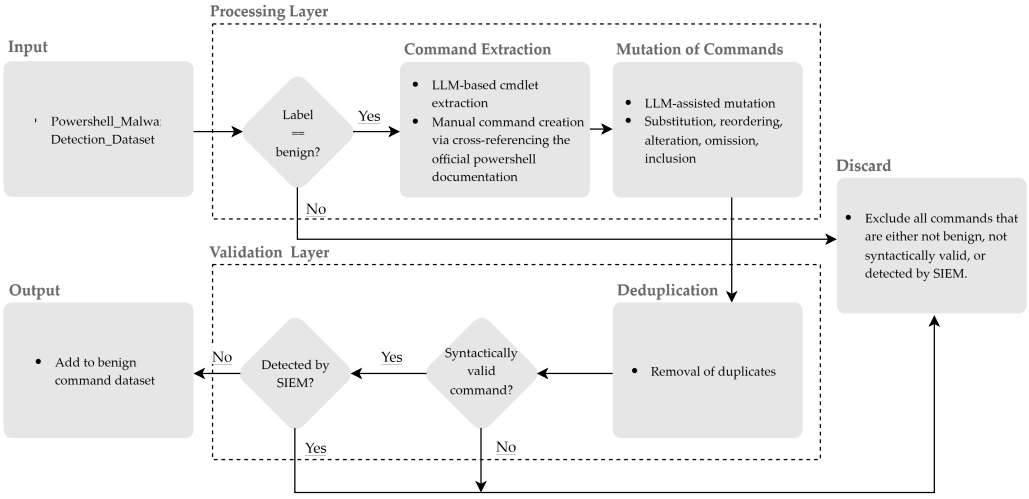


Figure 4.2: Generating and validating benign commands.

Due to the immense variety of unique PowerShell commands, inspiration was drawn from a publicly available PowerShell malware detection dataset[1] published on Hugging Face. This dataset originally contained approximately 38,000 script entries labelled as either malicious or benign. Malicious entries were quickly discarded, resulting in a subset of roughly 21,000 benign scripts. The cmdlets from these benign scripts were extracted using Mistral 7b[2], due to unrestricted free API access. In PowerShell, a cmdlet is a small, specialized command used to perform a specific task[3]. By cross-referencing 180 randomly chosen extracted cmdlets with the official PowerShell documentation, an initial selection of benign commands was made. Due to time constraints, these base commands were subsequently mutated with the assistance of the GPT-4o LLM[4] to create the remainder of the benign portion of the dataset, resulting in a total of 4500 benign commands. Mutation strategies were adopted in line with

[1]https://huggingface.co/datasets/rr4433/Powershell_Malware_Detection_Dataset
[2]https://huggingface.co/mistralai/Mistral-7B-v0.1
[3]https://learn.microsoft.com/en-us/powershell/scripting/lang-spec/chapter-13
[4]https://openai.com/index/hello-gpt-4o

the work done by Uetz et al. [7], and are detailed in Table 4.1. Finally, the commands underwent deduplication and were executed in a virtual machine environment to ensure syntactic validity. Given the virtual machines configuration and environment, runtime errors were considered acceptable. Only a few commands produced runtime errors, caused by missing dependencies or unreachable resources.

Table 4.1: Mutation strategies for benign commands.

Mutation	**Description**
Substitution	Replacing a cmdlet with an alias.
Reordering	Reordering the parameters of a cmdlet.
Alteration	Modifying the value of a parameter.
Inclusion	Adding a new parameter.
Omission	Excluding an existing parameter.

4.1.2 Evasions

In addition to benign commands, we also included a set of corrresponding evasion samples in our dataset. This was achieved by analysing rules commonly used by organizations and crafting evasions for those deemed vulnerable using the strategies outlined in table 4.2. The first step in crafting evasions of this kind was to perform a rule analysis in which open-source SIEM rules were analysed.

Rule Analysis

All rules under the sub-category Windows/PowerShell from the SIGMA ruleset were exhaustively enumerated by manually reviewing them, looking for potential applicable evasion techniques. Windows/PowerShell was chosen so that a comparison with state-of-the-art AMIDES[7] could be performed, since they also used this sub-category. If a rule had related or derived counterparts in other categories, those were also included in the analysis. Related rules were included because their detection logic can differ, meaning that an evasion that bypasses one may not bypass the other. For example, the SIGMA rule *Nslookup PowerShell Download Cradle*[5] had a related rule, *Nslookup PowerShell Download Cradle - ProcessCreation*[6] in the Windows/Process_creation sub-category. In total, 215 rules were analysed. The analysis involved cross-referencing the rules with official Microsoft documentation[7] along with the application of the evasion strategies as detailed in Table 4.2, to determine which techniques the rules were susceptible to (see Figure 4.3). A rule was considered fully evadable if all of its logical branches (i.e., the distinct conditions or paths within the rule's logic) could be evaded.

A rule was considered partially evadable if only some of its branches could be evaded, meaning the rule may still detect certain activities but can be bypassed in specific cases. Rules for which no working evasion techniques were found were not included in the dataset of evadable rules.

The rules that were classified as evadable were then transformed from SIGMA-format to Event Query Language (EQL) and added to Elastic Security to be triggered on logs from the virtual machines. Furthermore, explicit rules were already in place to detect obfuscation techniques such as Base64 encoding, string concatenation, and similar methods. Therefore, these types of evasion techniques were not considered during the analysis.

[5]https://github.com/SigmaHQ/sigma/blob/master/rules/windows/powershell/powershell_classic/posh_pc_abuse_nslookup_with_dns_records.yml

[6]https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_nslookup_poweshell_download.yml

[7]https://learn.microsoft.com/

Table 4.2: Used evasion techniques.

Evasion Technique	Description
Substitution	Replacing a command or its parameters with an alias.
Reordering	Changing order of parameters for a command.
Alternative Method	Using alternative command or parameter.
Insertion	Inserting redundant characters, new parameter or new argument.
Omission	Removing redundant characters or parameters.

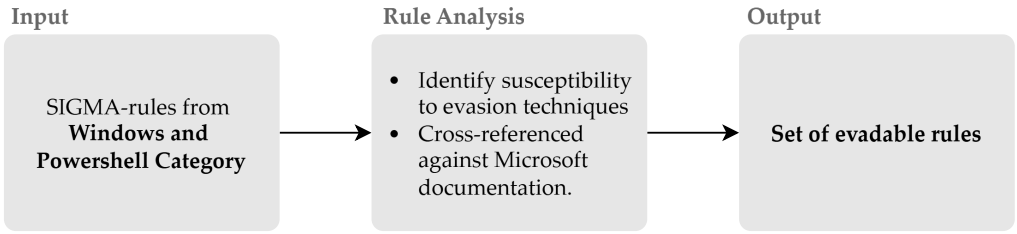


Figure 4.3: Identifying evadable rules.

Manual Evasion Crafting

After identifying which rules were evadable, evasion versions of the rules were manually crafted by using the evasion techniques they were susceptible to. We first selected a base command that triggered a SIEM alert, and then used it as the starting point for mutations. Then, by applying applicable evasion techniques to the base command, evasion versions were generated that would bypass detection. Table 4.3 shows an example of a triggering, and an evading command for a rule. To ensure that the evasions produced the same outcome as the original detected command and did not trigger an alert, they were executed on virtual machines with the SIEM setup unaltered and checked for detection while verifying that the result of the command execution remained unchanged. Figure 4.4 illustrates the process of creating and validating evasion commands. After all evasions were validated to ensure they performed identical actions as the original command without triggering any alerts by the SIEM system, they were added to the dataset.

Table 4.3: SIEM rule, matching command, and evading command.

Category	Details
SIEM Rule	`any where( process.command_line:"*nslookup*" and process.command_line:"*[1]*" and process.command_line: ("*-q=txt http*", "*-querytype=txt http*"))`
Affected Search Term	`process.command_line: ("*-q=txt http*", "*-querytype=txt http*")`
Used Technique	Substitution
Matching Command	`(nslookup -q=txt https://some-site.com)[1]`
Evading Command	`(nslookup -type=txt https://some-site.com )[1]`

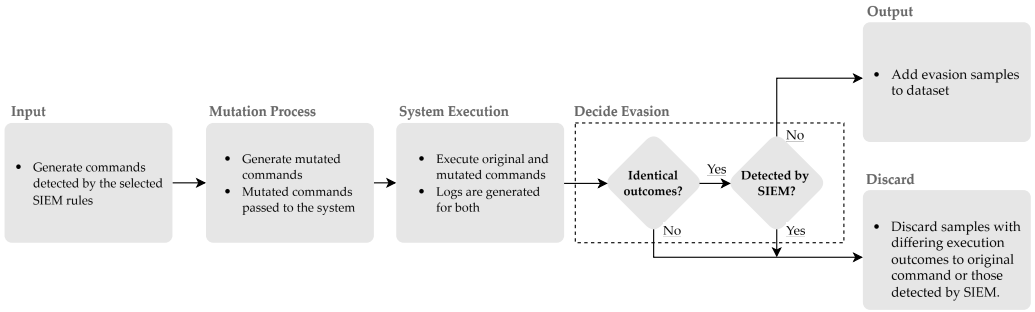


Figure 4.4: Creating and validating evasions.

4.2 Rule Pre-Processing

To detect rule evasions by performing semantic analysis, the SIEM rules had to be pre-processed to extract the features they were designed to detect. Given that SIEM rules can have multiple branches matching different command patterns, each extracted branch was individually compared against the executed command to determine its semantic similarity score. As seen in the example in figure 4.5, the SIEM Rule *Nslookup PowerShell Download Cradle* has two distinct branches, one with the search term *"*-q=txt http**"*, and the other one with the search term *"*-querytype=txt http**"*. Search terms in a SIEM rule are the specific keywords, pattern or field values the rule looks for in log data. We hypothesize that if one branch shows a high similarity score with a command, it would suggest that this branch may have been evaded. Because SIEM-rule evasions often involve syntactic changes rather than semantic ones, the underlying intent remains the same. A transformer, which maps commands and rule branches (extracted keywords from SIEM rules) into a high-dimensional space organized by meaning, is expected to place a command close to the branches representing its semantic content, regardless of superficial token differences.

The rule pre-processing included two key steps, pruning and feature extraction. The pruning step involved removing non-essential components of the rules that were tailored to a specific setup, also known as rule-cleaning. For example, some rules may target a specific machine, including its hostname in the rule. Such environment-specific elements were excluded to focus the analysis on the core features that the rules were meant to detect. Next, in the feature extraction step, each branch search terms were extracted. These were later used for performing the semantic analysis. Figure 4.5 shows the extraction process. After rule pre-processing, the rules were further tokenized and embedded to be used by the transformer.

4.3 Fine-Tuning the Transformer

The literature review ultimately led to the selection of the transformer-based CmdCaliper-large model available on HuggingFace [8]. CmdCaliper-large was considered suitable since it was trained on semantically similar pairs of Windows and PowerShell commands. It was also the best-performing transformer among the three CmdCaliper models: CmdCaliper-base, CmdCaliper-small, and CmdCaliper-large [12]. To adapt the transformer for the specific task of comparing SIEM rules with commands, it was fine-tuned using a dataset consisting of various aliases, small rule examples, and corresponding evading commands. In total, it consisted of 125 entries[9]. The training parameters are listed in Table 4.4.

[8] https://huggingface.co/CyCraftAI/CmdCaliper-large/tree/main

[9] This dataset is distinct from the previously mentioned one. It was used exclusively for training the transformer, not for testing.

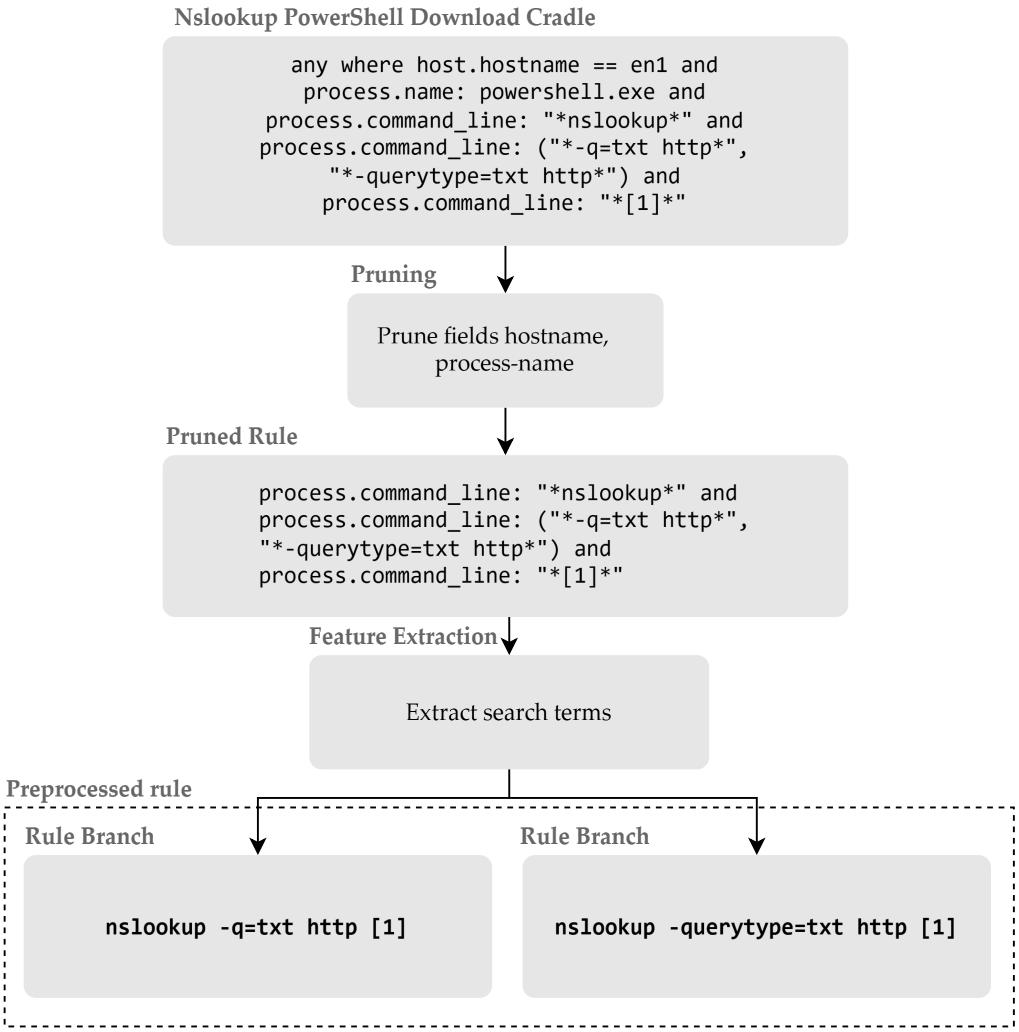


Figure 4.5: Pre-processing rules.

Table 4.4: Hyperparameters used for fine-tuning the transformer.

Parameter	Value
Loss Function	CosineSimilarityLoss
Epochs	5
Batch Size	16

4.4 Creating a SIEM Evasion Detection Pipeline

To evaluate our fine-tuned transformer, we developed an evasion detection pipeline integrated with the ELK stack. The pipeline consisted of multiple components, further detailed in subchapter 6.2. Vitrual Machines (VM) running Elastic Agents collected logs and sent them to Logstash, which were then stored on warm and hot indexes. These logs were analysed by querying SIEM rules to determine if any matched. Logs that did not match any rules were further sent to the transformer to evaluate whether they were close to triggering a rule by having a high cosine similarity with the pre-processed SIEM rules. Figure 4.6 shows the pipeline of identifying rule evasions.

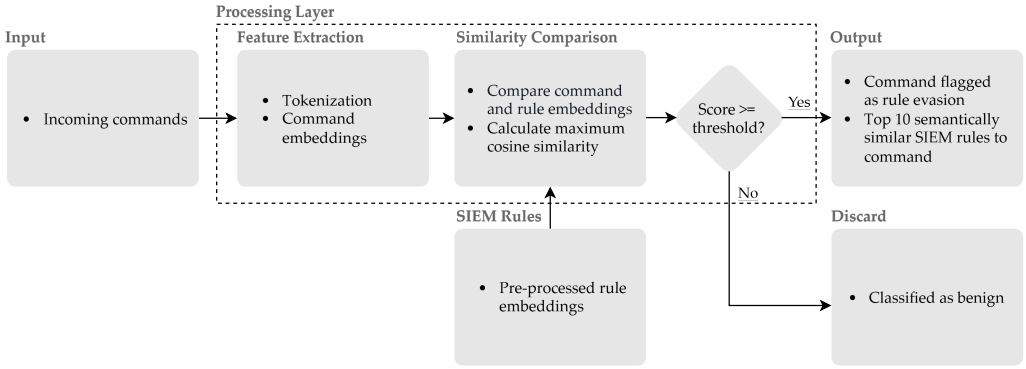


Figure 4.6: The SIEM evasion detection pipeline.

4.5 Evaluating the SIEM Evasion Detection Pipeline

To evaluate the SIEM evasion detection performance, we executed our dataset of rule evasions and benign commands on a virtual machine. The resulting execution logs were collected by Elastic Agents, sent through the SIEM system, and subsequently analysed by the SIEM evasion detection pipeline. The output of the analysis was a list of similarities between the embeddings generated by an executed command and branches of the pre-processed rules. If the maximum semantic similarity score surpassed a predefined threshold, it was flagged as a potential rule evasion. The threshold value was selected from the ROC curve, which was plotted to evaluate model performance. It was chosen from a region where both MCC and AUROC scores were consistently high, providing a good balance between TPR and FPR. Because the corresponding branch and rule were known, it was possible to pinpoint the exact detection logic that may have been circumvented.

After identifying that a potential rule evasion had occurred, the 10 rules with the highest similarity scores were compiled into a list. By examining this list for a specific incoming command identified as an evasion, a security analyst could further investigate which rule was bypassed and assess whether the flagged event constituted an evasion attempt. Figure 4.7 shows how the created pipeline was evaluated.

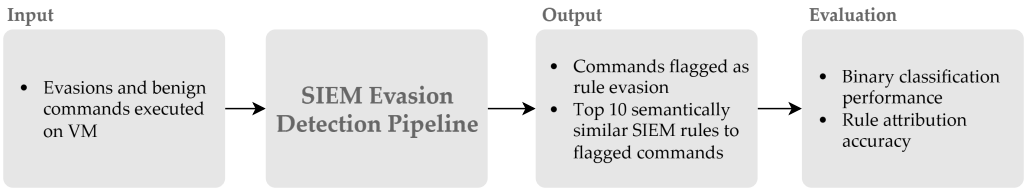


Figure 4.7: Methodology for evaluating the SIEM evasion detection pipeline.

5 Improving Rule Coverage

Rule coverage improvement, in the context of this thesis, refers to the process of updating a SIEM rule to more comprehensively detect malicious behaviour and identify evasions that were previously undetected by the original rule. To the best of our knowledge at the time of writing, this thesis presents the first approach to improving SIEM rule coverage using LLMs.

The first step of our method involved providing commands identified as rule evasions, together with their corresponding original rules, as input to a RF regressor. As depicted in Figure 5.1, the regressor then acted as a metamodel which selected the most suitable LLM from a model pool to process a crafted evasion and its associated evaded rule. Finally, the LLM with the highest suitability score was prompted with instructions on how to provide improvement recommendations for the evaded rule. From the previously constructed dataset of rule evasions, 70 percent was used to train the RF regressor and the remaining 30 percent was reserved to evaluate the updated rules.

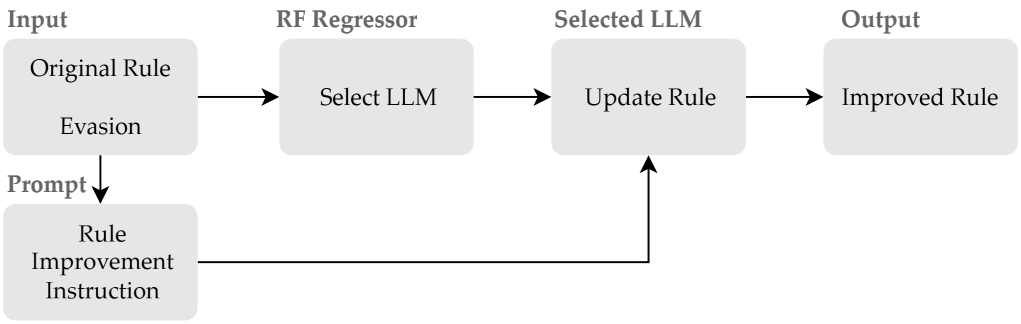


Figure 5.1: Improving evaded SIEM rules.

5.1 Constructing a Meta-Modelling Regressor

As mentioned in the chapter overview, the primary objective of the RF regressor was to act as a metamodel for the LLM selection process. It was developed using scikit-learn[1] and predicts the most suitable model from a pool to provide recommendations that enhance the coverage of a detected evadable SIEM rule. The models included in the pool were required to either have an API accessible at no cost or have API access provided by Sectra Communications. Additional factors, such as the number of parameters, popularity, and intended use cases, were also taken into consideration. Since a SIEM rule is a logical statement similar to code, *Codestral-2501* was selected. *GPT-4o* was included due to its widespread adoption and status as one of the most commonly used models. *Gemini-2.0-Flash* was chosen because it is freely accessible to anyone with a Google account. To evaluate the LLMs, the RF regressor was configured for multi-output regression, producing a separate score for each LLM. Since the RF model cannot directly process textual input, both the SIEM rule and the PowerShell command were converted into numerical vector representations using *all-MiniLM-L6-v2*[2], the default model of the Python sentence-transformers package. Its relatively small embeddings support semantically meaningful representations suitable for tree-based models, allowing the regressor to handle and learn from the textual information. Output from the regressor was three continuous scores between 0 and 1, each representing the predicted suitability of each LLM for updating an evaded SIEM rule. The LLM with the highest suitability score was selected to receive an API call, issuing the prompt depicted in Figure 5.2 to recommend improvements aimed at enhancing the coverage of the detected evaded rule.

> The provided command has evaded the current EQL rule. Return only a revised EQL rule that, in addition, successfully detects the command. Attempt to strike a balance between general coverage and necessary specificity, although the highest priority is that the new rule detects the command. No additional context or explanation should be included, return only the updated rule.
>
> Command:
> <*evasion*>
>
> EQL rule:
> <*rule*>

Figure 5.2: Prompt sent to LLMs.

5.1.1 Scoring Matrix

To assess the suitability of an LLM in terms of improving rule coverage, we developed a scoring matrix based on three key criteria: effectiveness, retention, and specificity. *Effectiveness* measures whether the LLM-updated rule successfully detects the evasion. *Retention* assesses how much of the original rule structure remains intact, thereby avoiding any alteration of the rule's behaviour beyond its intended purpose. *Specificity* evaluates whether the updated rule is overly narrow or excessively broad and considers the extent to which it may generate false positives. While effectiveness and specificity were scored binary as either 0 or 1, retention was graded on a scale of 0, 0.5, or 1.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
[2] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

These scores were combined using a weighted formula that we defined to calculate a suitability score (τ), providing a quantitative measure of how suitable the LLMs were for recommending improvements to a specific rule–evasion pair. The formula weights effectiveness most heavily because if the updated rule fails to detect the evasion, the improvement is fundamentally unsuccessful. Retention is weighted above specificity because the consulted security analysts emphasized that preventing false negatives, by preserving the original rule's coverage, is more important than the risk of additional false positives. They also noted that any additional false positives introduced by the update can be tuned away more easily, meaning that a low specificity can be managed. A detailed breakdown of the scoring criteria in addition to the weight assignment is provided in Figure 5.3.

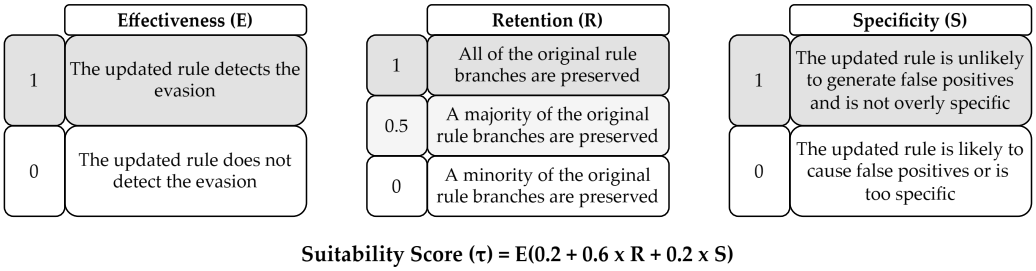


Figure 5.3: Scoring output from LLMs.

5.1.2 Score Distribution

As a prerequisite to training the RF regressor, every rule-evasion pair in the training set had to be included in the prompts issued to all of the LLMs. The responses were then evaluated using our scoring matrix as shown in Figure 5.3. Table 5.1 presents the average scores for our previously defined metrics *effectiveness*, *retention*, and *specificity* for each LLM. Additionally, the weighted suitability scores for each LLM are shown in Table 5.2. It is important to note that these tables do not represent experimental outcomes. Instead, they are by-products generated during the manual evaluation of LLM outputs used to train the regressor model.

Table 5.1: Average rule coverage improvement per suitability criteria.

LLM	Effectiveness	Retention	Specificity
Gemini-2.0-Flash	0.89	0.35	0.37
GPT-4o	0.98	0.50	0.65
Codestral-2501	0.80	0.18	0.15

Table 5.2: Average suitability score per LLM.

Gemini-2.0-Flash	GPT-4o	Codestral-2501	Maximum
0.43	0.62	0.24	0.67

5.2 Evaluating Updated Rule Coverage

The usefulness of a detection rule depends not only on its ability to detect evasions, but also on how much of the original rule is retained. It is also influenced by the degree to which the updated rule captures intended commands while minimizing false positives. Thus two methods were used to determine whether an updated rule has improved coverage. First, rule effectiveness was validated by observing the responses from the SIEM system. Second, a qualitative analysis was performed through expert reviews.

5.2.1 Rule Effectiveness Validation

To validate that the updated rules effectively detected their respective evasions, all updated rules were first integrated into the SIEM system. Each evasive command was then executed on a virtual machine, and the system logs were monitored to confirm whether an alert was generated or not. An evasion was considered successfully detected if an alert was raised, demonstrating that the coverage of the corresponding rule had been successfully improved. It is important to note that this validation did not assess whether the updated rules preserved the detection capabilities of the original rules. Some updates removed rule branches or made them overly specific, which meant that fully evaluating retention would have required a far more comprehensive dataset of benign commands and evasions. To address these limitations, we conducted expert reviews to evaluate the quality and applicability of the updated rules.

5.2.2 Expert Qualitative Reviews

Upon receiving an updated rule from the most suitable LLM, a triplet was constructed, comprising the original rule, the evasion, and the updated rule. Four security analysts at Sectra Communications independently evaluated the same set of 150 triplets. Using their experienced judgment, they qualitatively assessed each updated rule and categorized it as *effective*, *helpful*, or *flawed*. A rule was deemed effective if it could be deployed with minor to no modifications. Helpful rules were not effective but still provided clear, actionable insights for improving the evadable rule. Ultimately, an updated rule was deemed flawed if it failed to offer any value in terms of improving coverage against evasion.

6 Experimental Evaluation

This chapter presents the evaluation of our system, when tasked with identifying SIEM rule evasions and improving rule coverage. Before detailing the experimental setup and its results, we first present the key insights from the structured consultations conducted with three security analysts at Sectra Communications.

6.1 Expert Insights

The experts largely agreed that the most significant challenge in implementing new detection rules is to verify their effectiveness through testing. However, they cannot be tested in a production environment since they often require malicious events to trigger an alert. Thus, reliable data, a structured testing environment, and proper tools would help ensure effective detection of malicious events and tuning of rules. Furthermore, they placed significant emphasis on tuning, as excessive amount of alerts overwhelm analysts and cause alert fatigue, while too few alerts tend to reduce trust in the system due to elevated concerns regarding if the system is even working as intended.

When asked about common evasion techniques used by attackers, they mentioned the use of process injection, using scripts to assemble commands, rearranging arguments, mixing uppercase and lowercase characters, using aliases, and employing base64 encoding. To write rules that are robust to these types of evasion techniques, they mention that focus should be placed on indicators that are difficult to manipulate, as described by the Pyramid of Pain [46]. At the bottom of the pyramid are indicators that are easy to change or obfuscate, such as IP addresses or hash values. Higher up the pyramid are indicators that require more effort to modify, such as the tools used and Tactics, Techniques, and Procedures (TTPs). By focusing on indicators that are difficult for attackers to change, rules become more robust against evasion. Additionally, detecting malicious events based solely on user input analysis can be ineffective in isolation. Collecting logs closer to the hardware can help reveal an attacker's true intentions, as all commands will eventually resolve to actions such as process creation. Thus the key to defence is understanding the semantics of what is actually being executed, not how the input was formulated.

FP alerts, which are alerts triggered by benign events, were deemed important when measuring the effectiveness of SIEM rules. If a rule frequently generates alerts from benign events, it is considered ineffective. However, a rule can only fulfil its purpose if it accurately identifies attacks from malicious events, commonly referred to as TPs. In real-world scenarios, alert classification is rarely black and white. For critical rules, as in highly consequential, an acceptable number of FPs may sometimes be expected or even desirable. For example, it may be important to highlight changes to an administrator group, regardless of whether the change was made by an authorized user or an attacker. Additionally, rules targeting malicious behaviour are more difficult to measure because they require a real-world attack or an elaborate attack simulation to trigger an alert. Similarly, rules that contain many wildcard characters are likely to produce noise but may still be necessary. As an example they are required in situations were one wants to identity abnormal user activity on multiple hosts in a specific IP range.

Regarding challenges in designing and fine-tuning detection rules, the experts agreed that both FPs and FNs are difficult to manage. FNs are rarely observed and have the potential to be extremely dangerous, as they have the potential to cause damage to an organisation without raising a single alert. On the other hand, too many FPs can lead to alert fatigue. However, it is generally better to accept some FPs since rules that generate such alerts can be tuned, whereas undetected threats due to evadable SIEM rules may leave analysts in the dark.

All three analysts agreed that the open-source nature of the SIGMA repository enhances overall security. While potential attackers can analyse the rules in hopes of identifying vulnerabilities, sharing information publicly allows defenders and security researchers to reduce the collective effort required to detect such vulnerabilities. For example, SIGMA provides a strong foundation for writing new rules, although some manual work may be necessary since log schemas across SIEM systems are not standardized. Furthermore, many new attack patterns reuse known evasion techniques for which effective detection rules already exists.

6.2 Experimental Setup

As our system integrates with both a simulated enterprise network and a SIEM, this section details how the system environment was configured. In regards to model training, default parameters were largely retained, with random seeds fixed to ensure deterministic behaviour and reproducibility.

6.2.1 Network Topology

Figure 6.1 depicts the topology of a simulated enterprise network. It is comprised of a Cluster Zone consisting of a collection of nodes running VMs, and an Internal Zone that contains a log cluster and an intranet. The Cluster Zone represents an organization that is being monitored, thus log collecting agents are placed on the VMs. The logs are sent to the Internal Zone, to be analysed by the SIEM system Elastic Security. Notably agents only collect host data, such as Windows event logs, from the VMs.

Incoming traffic to and from a particular zone is routed through the central switch and passes through a firewall. Traffic entering the log cluster passes through a set of proxies with a diode in-between them, thus devices in the Internal Zone cannot send packets to the Cluster Zone. Logged data is used by the SIEM system Elastic Security. It is first received by the Logstash node and then stored on a Hot index, followed by a Warm index, and is accessible from the intranet via Kibana's graphical user interface. The Hot index is actively being written to while the Warm index is generally only being queried.

Traffic between two VMs on the same Virtual Local Area Network (VLAN) is invisible to the central switch, and the High Availability virtual switch manages communication between nodes. This communication is represented as dashed lines.

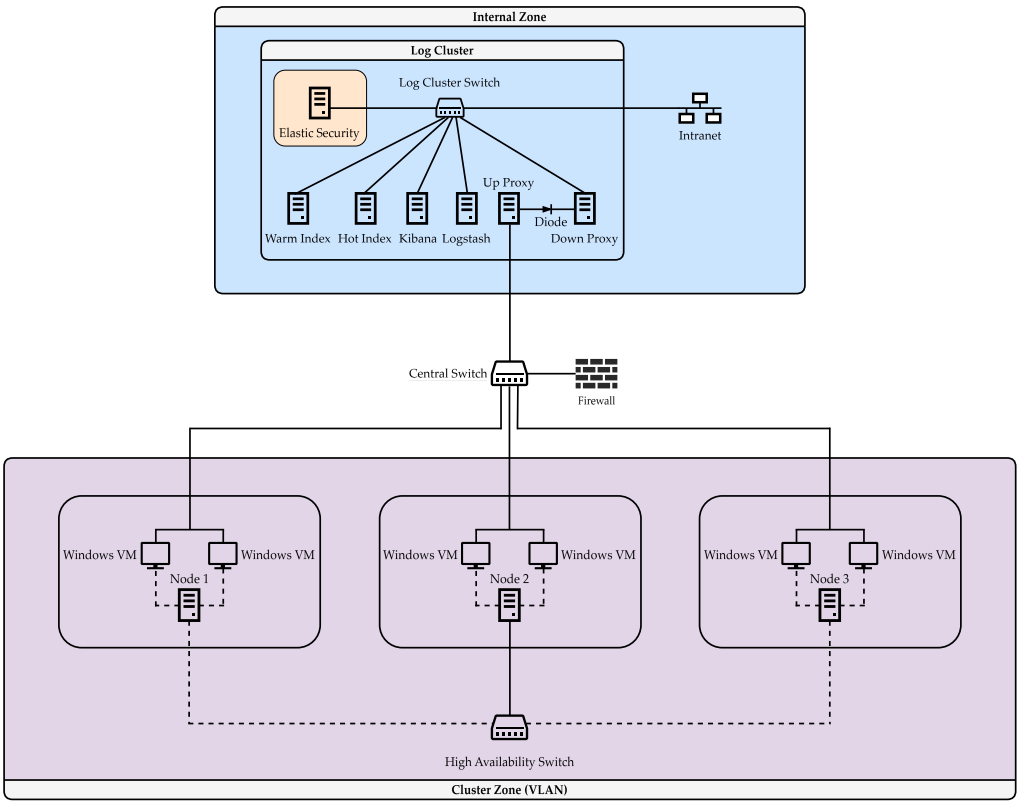


Figure 6.1: Topology of the simulated enterprise network.

6.2.2 Virtual Machine Environment

All VMs were deployed using the Proxmox Virtual Environment hypervisor[1], and run Microsoft Windows 10 Pro with PowerShell version 5.1. The commands were executed from the Command Prompt using the `-C` flag, ensuring that each command has a unique Parent Process Identifier (PPID) to link it with a potential SIEM alert. Additionally, the VMs run an Elastic Agent that collects and ingests system, Windows, and PowerShell logs into the SIEM via Elasticsearch in the ELK stack.

6.2.3 Execution Flow

The execution spans three zones, as illustrated in Figure 6.2. On the VMs, execution logs are generated and an agent collects system, Windows, and PowerShell logs on behalf of the SIEM. The SIEM then evaluates whether any of the collected logs match an existing rule. If so, an alert is raised.

By transferring both the execution logs and the raised alerts to our proposed system, it becomes possible to detect any SIEM rule evasion attempts. Detected events are not processed further. However, the undetected events are tokenized and then embedded as vectors in high-dimensional space. If the cosine similarity between the command vector and any rule vector exceeds a defined threshold, the command is postulated as a likely evasion attempt. The rule with the highest similarity together with the respectively flagged evasion is then passed to our RF regressor, which selects the most suitable LLM for improving the rule. Lastly an API call is made to the selected LLM, prompting it to improve the coverage of the evadable rule.

[1] https://www.proxmox.com/en/products/proxmox-virtual-environment/overview

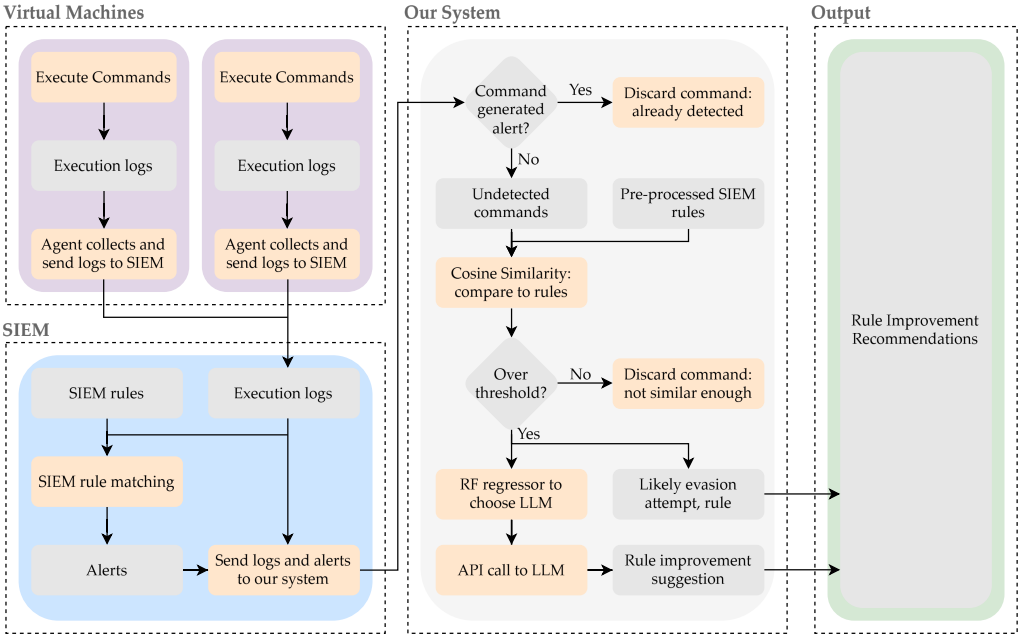


Figure 6.2: Execution flow of the system.

6.3 Detecting Rule Evasions

The analysis of the 215 Windows/PowerShell rules, presented in figure 6.3, revealed that 45 rules were fully evadable, and 6 rules were partially evadable. Additionally, 7 rules were identified as potentially evadable; however, due to significant engineering overhead required along with resource constraints, we were unable to find evasions in these cases. As a result, 24% of all analysed rules were found to be fully or partially evadable, with concrete counterexamples. The most common evasion technique was substitution, where, for instance, a cmdlet flag had an alias that yielded the same result. Many commands and flags in Windows/PowerShell have abbreviations that could also be used as aliases. For example, the command *Set-Alias* can be abbreviated as *sal*. Another evasion technique to which some of the rules were susceptible of was reordering, where changing the order of parameters would allow an attacker to bypass detection.

6.4 Evaluation of Rule Evasion Detection and Attribution

The system's ability to detect evasions was evaluated through semantic similarity analysis using the fine-tuned transformer on the dataset comprised of 4500 benign commands and 500 evasive ones. Each command was compared to all branches of the evadable rules, and the maximum cosine similarity score for each command was recorded. Based on these scores, the ROC curve was plotted, AUROC approximated, and an appropriate threshold for the cosine similarity score was selected. Once the threshold was established, performance metrics including TPR, FPR, Accuracy, F1-score and MCC-score were computed.

The effectiveness of the system's rule attribution ability was evaluated by processing all 500 evasions through the system, recording each evasion's match to all rules and then determining at which position, the truly evaded rule would be in. If a command evaded 2 rules and the rules would be ranked as most likely and second most likely, both would be considered as number one, as excluding one of them would result in being ranked as one.

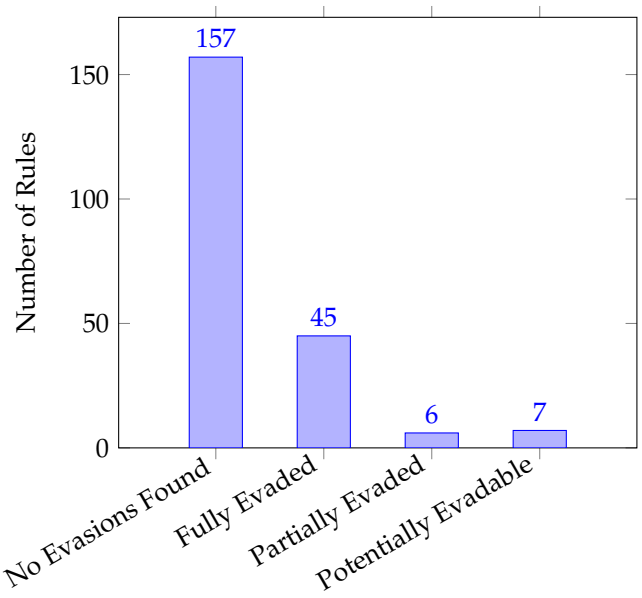


Figure 6.3: Rule analysis and classification.

6.5 Results of the Rule Evasion Detection and Attribution

Pertaining to the task of detecting SIEM rule evasion attempts, our system was able to detect 71% of all evasions with a low FPR of approximately 0.9%. This performance was achieved using a threshold value of 0.77, where any command with a higher cosine similarity score was classified as malicious. For a more comprehensive assessment, Table 6.1 summarizes security-focused metrics including TPR, FPR, and others. While a higher TPR could be achieved by lowering the threshold, this would also increase FPR. Conversely, raising the threshold reduces FPR but lowers TPR. The chosen value balances a high TPR with a low FPR, which is important to avoid overwhelming security analysts with unnecessary alerts.

Table 6.1: Performance metrics for evasion detection.

Metric	**Score**
TPR	0.71
FPR	0.0093
Accuracy	0.96
F1-score	0.79
MCC	0.78

To further evaluate the performance of our system across different threshold values, a ROC curve was plotted (see Figure 6.4). The ROC curve illustrates the trade-off between the TPR and the FPR at various threshold settings. The AUROC score, at approximately 0.93, indicated a good overall discriminative ability of our system in the context of detecting evasion attempts.

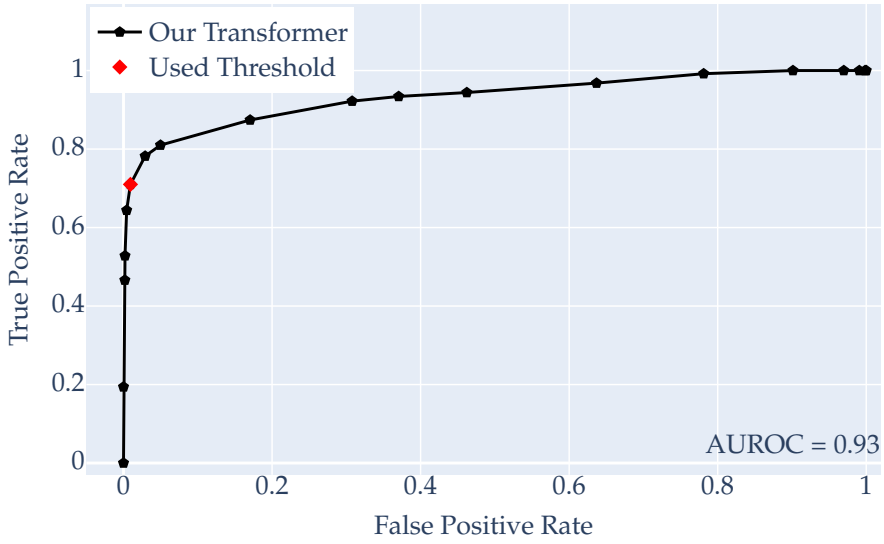


Figure 6.4: The system's ROC curve after fine-tuning, with an AUROC score of 0.93.

6.5.1 Results of the Rule Attribution

By examining the most semantically similar rules for the detected evasion cases, the fine-tuned transformer successfully identified the evaded rule as the top match in 98% of these cases (see Table 6.2). Additionally, the system identified the evaded rule within the top 5 most similar rules in 99% of all evasion cases like this. These results demonstrate that our system can effectively determine which rule was most likely evaded, and in 98% of all cases, pinpoint the exact rule that was bypassed.

Table 6.2: Rule attribution accuracy at different similarity thresholds.

Rule Attribution (Top n-rule match)	**Accuracy (%)**
Top 10	99.2
Top 5	99.2
Top 1	98.0

6.5.2 Comparison with the State-of-the-Art

AMIDES [7] was able to detect 70% of all their crafted Windows and PowerShell evasions without raising any FPs. On the same task that AMIDES was evaluated on, our system was able to detect 71% of evasions, with a low FPR of 0.9%. This demonstrates that our system offers similar (and slightly better) detection performance compared to AMIDES while maintaining a low false positive rate. Moreover, it outperforms AMIDES on the rule attribution task in both the Top-1 and Top-10 scenarios, as shown in table 6.4, demonstrating the effectiveness of our approach. In this table, the Top-1 and Top-10 scores represent the percentage of identified evasions for which the correct underlying rule is ranked first or appears within the top ten predicted rules, respectively. These percentages therefore reflect how often the system successfully identifies the evaded rule among the evasions it detects.

Table 6.3: Detection performance comparison between our system and AMIDES.

System	TPR	FPR
Our System	71%	0.9%
AMIDES	70%	0%

Table 6.4: Rule attribution accuracy comparison between our system and AMIDES.

System	Top-1	Top-10
Our System	98.0%	99.2%
AMIDES	63%	95%

6.6 Results of the Rule Coverage Improvements

To determine the effectiveness of the improved rules, each rule was evaluated based on its ability to detect its corresponding evasion. Security analysts also reviewed and graded the improved rules to provide more detailed insights as to their coverage and integration. Based on our custom dataset used to train the RF regressor, its output revealed a substantial difference in selected LLM for generating updated rules. As shown in Table 6.5, OpenAi's GPT-4o was selected significantly more often than both Google's Gemini-2.0-Flash and Mistral AI's Codestral-2501. Notably, Codestral-2501 was never chosen to update a rule, indicating that its generated rule updates were consistently considered less effective or irrelevant compared to those of the other models.

Table 6.5: Statistics on number of times each LLM was chosen to update a rule.

Gemini-2.0-Flash	GPT-4o	Codestral-2501
24	126	0

6.6.1 Detection Results using LLM-based Updated SIEM Rules

After integrating the updated rules into the SIEM system and executing the evasive commands on a virtual machine, a total of 105 alerts were generated. This means that 70% of the 150 suggested rule updates successfully detected their corresponding evasive commands, as shown in Figure 6.5.

6.6.2 Expert Validation Results for the Rule Improvement Recommendations

When presented with the recommendations from our system, on average, the security analysts found that 46 out of 150 rule improvement recommendations were effective, meaning they could replace the original rules with only minor adjustments. Another 71 recommendations were considered helpful, while not suitable for direct integration, they provided valuable input that could assist in manually fine-tuning the original rules. The remaining 33 recommendations were deemed flawed, offering little to no assistance in the rule improvement process. A breakdown of the expert responses is shown in Figure 6.6.

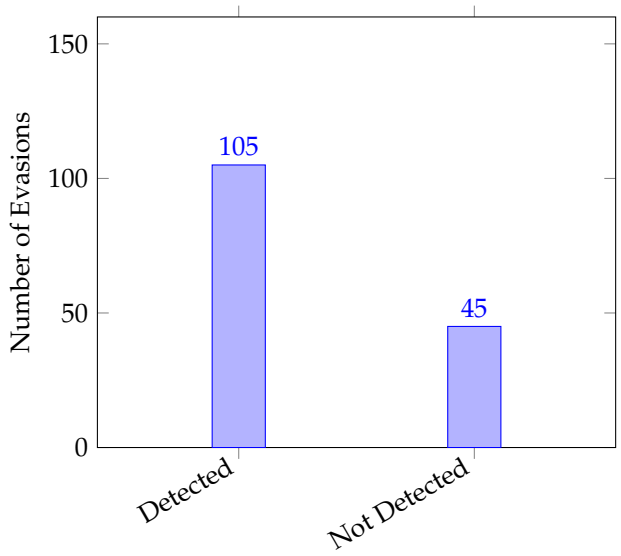


Figure 6.5: Detection outcomes with updated rules.

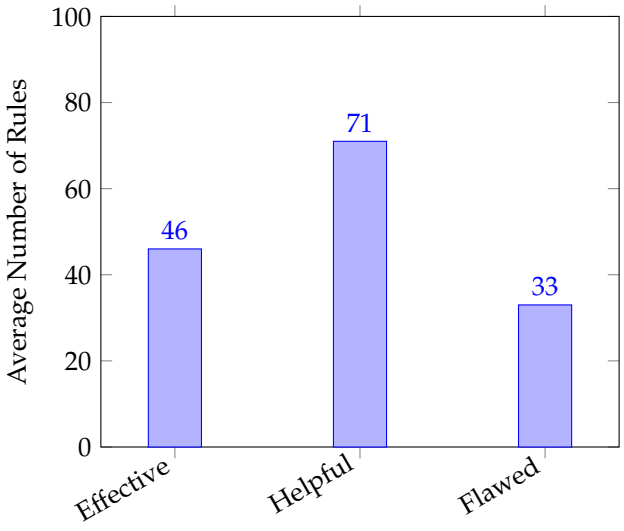


Figure 6.6: Classification of updated rules.

7 Discussion

This chapter serves to provide some deeper insights into the findings of the study, the applied methodology, and its limitations, highlighting their implications for advancing evasion detection in SIEM systems.

7.1 Results

This section interprets key metrics, and assesses detection effectiveness and rule attribution in the context of our experiments. We also discuss the performance comparison with the State-of-the-Art in addition to improvements in rule coverage.

7.1.1 Detecting and Attributing Rule Evasions

Using our transformer-based approach, 71% of all evasion attempts were successfully detected, with an FPR of merely 0.9%. Although an FPR of 0.9% might seem low, in an enterprise environment where thousands of events are generated every hour, this could still translate to a considerable number of false alerts. For instance, out of 10000 true positive alerts the approach will correctly have raised 7100, while for every 10000 false positive ones it will incorrectly raise 100. Regarding model bias, it appears to be minimal, as the F1 and MCC scores are nearly identical. Thus, the class imbalance does not seem to have a significant impact on model performance.

Compared to AMIDES, the method used in this thesis demonstrated significantly better performance in rule attribution. Our method was able to identify exactly which rule was evaded almost every time. However, it is important to note that AMIDES was tested in a production environment using real world data, making its evaluation more realistic. AMIDES was also evaluated for its suitability in real-world applications, an aspect not addressed in this study. Realistically, deploying our system as it is would likely create additional workload for analysts. Consequently, its potential for detecting evasions might be overlooked. However, the system could still be valuable for rule validation. When a new vulnerability is published, it can identify the rule best suited for amendment and suggest an update for it.

7.1.2 Improved Rule Coverage

As there is no prior research on leveraging ML to improve detection rule coverage, no established benchmarks exist for meaningful comparison. In this context, our findings offer valuable insights into the potential of LLMs in this domain. Evaluation results show that the LLM pool demonstrated a reasonable capability to generate improved rules that successfully detected their corresponding evasions, with a 70% success rate. However, challenges were evident in maintaining the original rule structure and avoiding rule modifications that were either overly narrow or excessively broad. Among the LLMs assessed, GPT-4o proved to be the most effective, being selected by the regressor in 84% of cases. Additionally, expert validation indicated that approximately one-third of the rule improvements generated by LLMs could be deployed with minimal modification and are likely to detect evasions effectively.

Nevertheless, the use of LLMs in this context is subject to fundamental limitations, they are non-factual, and should not be blindly trusted [47]. LLMs do not possess an understanding of the language they process, nor the meaning of individual tokens. They are trained solely on next-token prediction tasks, statistically estimating the most likely context based on preceding tokens. This limitation embodies the stochastic parrot metaphor, which highlights a models tendency to generate responses that sound plausible but are factually incorrect [32]. This issue is further exacerbated due to biases in the training data and token encoding. Since the output of the LLMs cannot be inherently trusted, due to lack of intent, contextual understanding, and awareness of real-world consequences, we adapted our methodology. While the rule improvements may be able to detect evasions, there is no guarantee that they are always coherent or meaningful. Thus the process was augmented with expert validation. Finally it should be mentioned that even though an updated rule may detect the manually crafted evasions, it could still allow new evasions to bypass detection, generate excessive FPs, or be overly specific.

7.2 Method

The method employed in this study comprises multiple stages, each examined in detail in this section to assess their influence on the results.

7.2.1 Dataset Creation

While the dataset formulation was originally posed as a research question, it is discussed within the methodological framework, as its construction was necessary for performing SIEM rule evasion detection, attribution, and rule coverage improvement. This subsection therefore details the creation of the dataset used to fine-tune the transformer-based model and examines the limitations of this approach, along with potential improvements to enhance dataset quality and experimental outcomes.

Evasions

The SIEM rules were manually reviewed to identify evasion techniques to which they were susceptible. Due to the manual nature of this process, it is possible that some rules may be vulnerable to additional evasion techniques that were not enumerated in Table 4.2. Likewise, rules classified as non-evadable might, in fact, be susceptible to evasion methods of the aforementioned nature. Therefore, the actual percentage of evadable rules within the analysed set may be higher than reported.

The SIEM rules, originally written in SIGMA detection format and identified as vulnerable to evasion, were converted to EQL using the SIGMA Rule Converter tool, supplemented by manual adjustments to ensure valid conversions. The main limitation of the SIGMA Rule Converter is that it often requires manual intervention to ensure that the translated rules function correctly within the target SIEM system, in our case, EQL in Elasticsearch. As a result, our dataset cannot be used directly to detect evasions in another target SIEM system due to the manual step in our translation processes. A more reliable method without any manual conversion refinement would have been desired.

The 500 evasions were manually developed by identifying methods capable of bypassing Windows and PowerShell SIEM detection rules. Each evasion was verified to ensure that it replicated the functionality of the original (detected) command while successfully avoiding detection by the SIEM system. However, because the detection rules were translated manually into the SIEM system rather than using native or vendor-provided implementations, some crafted evasions may not have produced the desired effect. In the context of this thesis, however, our crafted evasions were considered as valid when used in conjunction with our translated rules.

Benign Commands

Benign PowerShell commands were created by first extracting cmdlets from the publicly available dataset of scripts introduced in Section 4.1 using the Mistral 7b LLM. Although the model exhibited suboptimal performance, it was utilized locally to reduce computational costs, as the scripts would have otherwise generated a substantial number of tokens. Following this, the cmdlets were cross-referenced with official PowerShell documentation to form an initial base set of 180 valid commands. Using GPT-4o, the base set of commands were then mutated to produce a total of 4500 benign entries, following predefined mutation strategies. The resulting commands were deduplicated and validated through execution in a VM environment. However, due to biases present in LLMs, it is undeniable that some degree of bias subsequently transferred into our dataset. It is therefore important to recognize that the dataset does not accurately represent genuine benign activity, as the commands were synthetically generated rather than derived from real-world operations. While incorporating actual benign data would have introduced its own challenges in terms of privacy, it could have improved the realistic nature of the dataset.

7.2.2 Identifying Rule Evasions

The detection of rule evasions focused primarily on the command line field found in logs collected by the Elastic Agent. Specifically, the classification relied solely on the executed command, alongside a corresponding SIEM rule. However, since intrusions into organizations typically involve multiple stages, additional context incorporated into the evasion detection process could likely improve overall effectiveness. Additionally, the SIEM rules were preprocessed into branches. Since a complete rule often contained significantly more characters than an individual command, the preprocessing reduced the discrepancy in token counts, thereby increasing the similarity score between them. This helped determine an appropriate discrimination threshold for classification by producing more accurate similarity scores with a clearer separation between similar and dissimilar pairs of rules and commands. However, it comes at the cost of reducing contextual information since no command was compared again a complete rule. Lastly, classification was based on the maximum similarity between an evasion and all rule branches. In theory, this approach reduces FNs, as even a single strong match can trigger detection. Alternatively, using the average similarity might reduce FPs since multiple branches would require a high similarity compared to the command.

Moreover, the choice of transformer had a significant impact on evasion detection. CMDCaliper-large was pre-trained to generate embeddings from tokenized Windows and PowerShell commands. When we fine-tuned the model, the choice of hyper-parameters clearly influenced the results. This became especially evident when experimenting with different model sizes, measured by the frobenius norm of the weight matrices. Besides the choice of transformer, the method used to compare embeddings likely influenced the classification.

7.2.3 Improving Rule Coverage

In our approach, LLMs were used for improving the coverage of the rules in terms of detecting evasion attempts. However, the selection pool of LLMs was limited, as evident from the fact that GPT-4o was chosen in more than four out of five cases. It is also worth noting that the manual grading of an LLMs ability to improve a rule may have introduced bias into our regressor-based model that determines LLM suitability for the rule-evasion pair. Nevertheless, this does not prevent general conclusions from being drawn about the applicability of LLMs to improve SIEM rules.

Regarding the model used to select the suitable LLM for rule improvement, several options were considered. One approach would have been to use a classifier instead of a regressor, training it solely on which LLM was preferred for each rule–evasion pair. Additionally, the RF model could have been replaced with other algorithms such as a SVM or eXtreme Gradient Boosting. However it is important to note that a regressor is what allowed us to compute a suitability score, as it outputs a number instead of a classification label.

7.3 Limitations

While using a transformer to detect stealthy alterations in executed commands to evade SIEM detection proved effective, it is still possible for attackers to escape detection by the transformer itself. Since the approach relies on comparing features that SIEM rules look for with executed commands, an attacker could introduce irrelevant or 'junk' tokens to reduce the cosine similarity between the two. This highlights a limitation of our method, as it may be susceptible to evasion attacks, which makes the use of AI in cybersecurity a double-edged sword. This could potentially be mitigated by pruning redundant parts of the commands prior to analysis, but doing so would require a reliable method for distinguishing between meaningful and non-meaningful tokens, which introduces additional complexity and the risk of discarding important context. Another way to mitigate this risk is by leveraging ECOS. However, this approach would introduce significant computational overhead.

Additionally, the current approach does not handle the use of negations within the branching process of the SIEM rule analysis. Since the transformer is solely used to generate embeddings for rules and commands, the presence of a negation has a small impact on the resulting cosine similarity. This is because the embeddings capture the overall semantic context of a rule, making the effect of negations less pronounced than in the rule itself. In practice, this can lead to FPs when a negation significantly alters the intended meaning. One potential solution is to apply more rigorous pre-processing, such as explicitly identifying and handling negations. While it is also possible to train the transformer to recognize negation terms, this would likely require substantial additional effort due to expensive transformer training and fine-tuning.

Another consideration is the dimensionality of the embeddings. To compute the dot product and, by extension, the cosine similarity, embeddings must have the same dimensions. When the number of tokens differs between a rule and a command, corrective measures must be taken. If the embeddings are thought of as vectors, this issue is typically addressed by padding the shorter vector with zeros. Alternatively, both vectors can be projected into a lower-dimensional space. However, each of these methods distorts the embeddings to some degree, potentially affecting the accuracy of the cosine similarity.

Moreover, the transformer model was specifically trained on PowerShell and Windows commands and fine-tuned using aliases and similar command patterns. As a result, its effectiveness in other environments such as Linux systems, cloud platforms, or web applications has not been tested. At the minimum, adapting the model to these domains would require retraining or additional fine-tuning on relevant data to ensure accurate and effective evasion detection. Because this approach depends on aligning features from executed commands with those defined in existing SIEM rules, novel attacks that do not match any current SIEM rule are likely to go undetected. Therefore, this method is inherently limited in its ability to detect previously unseen attack techniques.

Another aspect that was initially beyond the scope of this thesis, but is important to address, is the impact of time constraints and resource usage. In real-world scenarios, a SOC investigates thousands of alerts each day, resulting in a significantly higher volume of logs and commands ingested by the SIEM system. This imposes strict performance and scalability requirements on any additional system designed to detect rule evasions and suggest improvements. If such a system cannot keep up with the incoming traffic, it quickly becomes ineffective. This challenge is further illustrated by the use of wildcards in all tested SIEM rules. From an analysts perspective, this can be beneficial, as it increases the likelihood of detecting malicious activity. However, wildcard usage leads to highly resource-intensive database queries. That also significantly extends the time required to complete each query.

7.4 Practical Implications

The findings of this thesis have meaningful implications for the future of cybersecurity operations, particularly in the domain of threat detection. As attacker techniques evolve to bypass traditional detection methods, incorporating ML and LLMs into SIEM workflows offers a way to enhance robustness.

7.4.1 Enhancing Threat Detection and Response Workflows

By integrating ML models such as transformers into the workflow of a SIEM system, variations in known malicious activities can be identified which traditional rule-based systems may fail to detect. Failure to identify such activity may result in extended periods during which a threat remains undetected in the system, potentially leading to data breaches or operational disruptions. Identifying which SIEM rule was likely evaded can provide valuable insights into the attacker's intent and help analysts quickly understand and improve the corresponding detection logic. Furthermore, LLM-generated recommendations for rule improvements can assist in identifying potential coverage gaps and suggest modifications to existing detection rules. This could potentially ease the process by reducing the manual effort needed to identify gaps and craft new rules.

7.4.2 The Necessity of Human Oversight

Although detecting evasions and attributing them to specific SIEM rules using a transformer proved effective, relying solely on a LLM to automatically update SIEM rules is not advisable. As such, the involvement of a security analyst remains essential to review, validate, and implement rule modifications. While LLMs can help in generating recommendations for rule improvements, the final responsibility ultimately lies with the security analyst.

7.4.3 Challenges and Considerations for Real-World Deployment

While the integration of transformer-based models and LLM-generated rule improvements offers clear benefits, practical deployment in operational environments presents significant challenges. The computational cost of running LLMs at scale may be substantial, requiring careful consideration of infrastructure capabilities and resource constraints. Additionally, trained models must be stored securely to prevent unauthorized access, tampering, or theft (in terms of model theft). Additional mitigation techniques to defend these (transformer-based) models against evasion attacks must also be developed.

8 Conclusion and Future Works

Although ML is becoming increasingly prevalent in the cybersecurity domain, it has not yet matured to the point of being entirely trustworthy. Thus this work explores a transformer-based approach for detecting SIEM rule evasions, while keeping human analysts in the loop. By leveraging NLP to compute the semantic similarity between pre-processed SIEM rules and incoming evasion attempts, our method achieves a comparable detection rate and delivers a significant improvement to the process of attributing a detected evasion to its corresponding SIEM rule compared to the State-of-the-Art AMIDES [7]. Furthermore, our method was also able to generate improved rules based on a given evasion and the corresponding SIEM rule. The following provides a summarized overview of this thesis in terms of the answers to our postulated research questions and contributions.

RQ 1. How can machine learning be utilized to identify SIEM rule evasions?
By using a transformer-based model to perform semantic similarity analysis, this thesis shows that this approach can effectively identify SIEM rule evasions by capturing subtle variations in executed commands that traditional rule-based systems may miss. By fine-tuning a pre-trained transformer, we detected 71% of all evasion attempts while maintaining a low FPR of only 0.9%, demonstrating the effectiveness of this approach. Additionally, for the detected evasions, we were able to identify the corresponding SIEM rule with an accuracy of 98%. This process was used to improve the coverage of the SIEM rules to account for potential evasion attempts.

- **How can a representative dataset of benign activity and SIEM rule evasions be constructed?**
  Through analysis of open-source SIEM rules using a structured set of evasion techniques, we identified and produced 500 SIEM rule evasions. At the same time, we extracted benign cmdlets from a publicly available dataset containing both benign and malicious scripts. By applying LLM-assisted mutation techniques, we expanded these cmdlets into 4500 benign commands. Combining the evasive and benign PowerShell commands resulted in a representative dataset.

RQ 2. How can machine learning improve detection coverage of evadable SIEM rules?
Using an RF Regressor metamodelled to select the most suitable LLM to generate recommendations for rule improvements, our approach lead to the improved rules being able to detect their corresponding evasions with an accuracy of 70%. However, since some of the rule improvement recommendations failed to preserve the structure of the original rule branches, we therefore incorporated human experts (security analysts) into the process. Since trusting LLMs to update rules on their own without any oversight could be detrimental to the cybersecurity outcomes and be counterproductive to the goals of this thesis. The security analysts found, on average, 46 of the improved rule recommendations to be effective, 71 to be helpful, and 33 to be flawed.

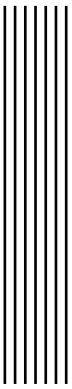
Most of the future research directions identified in this thesis arise directly from the limitations or practical implications discussed in the previous chapter. These limitations point to several areas where the current approach can be expanded or improved upon to enhance its applicability and effectiveness in a wide-range of scenarios.

The use of a pre-trained transformer model, CMDCaliper-large, with only minimal fine-tuning demonstrated promising results in comparing SIEM rules with executed commands. However, only fine-tuning the model limited its full potential. Future work could aim to improve performance by expanding the training dataset with more complex examples and by exploring alternative fine-tuning methods. Additionally, the current transformer architecture was not specifically designed for rule-command comparison, suggesting an opportunity to develop more specialized models that can offer better mapping between detection rules and command behaviour. Furthermore, the model currently cannot handle all logical operators found in SIEM rules, such as negations, which restricts its ability to fully process complex rule logic.

Another promising enhancement involves moving beyond evaluating individual commands in isolation and instead, extending the detection method to analyse sequences of commands within a defined time window. This could capture patterns related to complex multi-step attacks that may not be possible by simply analysing single commands.

Regarding rule improvements, this thesis utilized a pool of only three LLMs to produce recommendations for expending rule coverage. Increasing the number and diversity of LLM models, experimenting with a wider variety of prompts, and enhancing the scoring matrix to include additional features such as rule complexity could lead to better rule improvement outcomes. Similar to transformers, fine-tuning the LLMs for the specific task of recommending rule improvements could also increase the overall process quality. However, future work should consider whether leveraging multiple LLMs is always necessary or whether a single, well-suited model may be preferable, especially given practical constraints such as computational cost and energy footprint. Considering that one LLM was selected in the majority of cases in our study, additional models may not provide substantial added value.

Finally, future studies should examine practical performance metrics such as model inference time, power consumption, and hardware requirements to fully assess the model's viability for deployment in real-world security operations. This assessment should specifically focus on investigating the performance trade-offs of integrating transformers and LLMs within SIEM systems in production environments, with special attention paid to added latencies and their impacts on real-time threat detection.



Bibliography

[1] Cisco. *2024 Global Networking Trends Report*. Accessed: 2025-01-30. URL: https://www.cisco.com/c/dam/en/us/solutions/enterprise-networks/2024-global-networking-trends.pdf.

[2] Joint Cybersecurity Advisory. *Russian Military Cyber Actors Target U.S. and Global Critical Infrastructure*. Accessed: 2025-02-13. URL: https://media.defense.gov/2024/Sep/05/2003537870/-1/-1/0/CSA-Russian-Military-Cyber-Target-US-Global-CI.PDF.

[3] Lance Cleghorn. "Network defense methodology: A comparison of defense in depth and defense in breadth". In: *Journal of Information Security* 4.3 (2013), pp. 144–149. URL: https://doi.org/10.4236/jis.2013.43017.

[4] Thuy Nguyen. *Effective Threat Detection: What Is Threat Detection and Response (TDR)?* Published: January 09, 2023, Accessed: January 30, 2025. URL: https://www.crowdstrike.com/en-us/cybersecurity-101/threat-intelligence/threat-detection/.

[5] Nikos Virvilis and Dimitris Gritzalis. "The big four-what we did wrong in advanced persistent threat detection?" In: *2013 international conference on availability, reliability and security*. IEEE. 2013, pp. 248–254. URL: https://doi.org/10.1109/ares.2013.32.

[6] Sandeep Bhatt, Pratyusa K Manadhata, and Loai Zomlot. "The operational role of security information and event management systems". In: *IEEE security & Privacy* 12.5 (2014), pp. 35–41. URL: https://doi.org/10.1109/MSP.2014.103.

[7] Rafael Uetz, Marco Herzog, Louis Hackländer, Simon Schwarz, and Martin Henze. "You Cannot Escape Me: Detecting Evasions of SIEM Rules in Enterprise Networks". In: *33rd USENIX Security Symposium (USENIX Security 24)*. Philadelphia, PA: USENIX Association, Aug. 2024, pp. 5179–5196. ISBN: 978-1-939133-44-1. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/uetz.

[8] Bushra A. Alahmadi, Louise Axon, and Ivan Martinovic. "99% False Positives: A Qualitative Study of SOC Analysts' Perspectives on Security Alarms". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 2783–2800. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/alahmadi.

[9] Talha Ongun, Jack W Stokes, Jonathan Bar Or, Ke Tian, Farid Tajaddodianfar, Joshua Neil, Christian Seifert, Alina Oprea, and John C Platt. "Living-off-the-land command detection using active learning". In: *24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '21)*. 2021, pp. 442–455. URL: https://doi.org/10.1145/3471621.3471858.

[10] Ryan Stamp. *Living-off-the-Land Abuse Detection Using Natural Language Processing and Supervised Learning*. 2022. arXiv: 2208.12836 [cs.CR]. URL: https://doi.org/10.48550/arXiv.2208.12836.

[11] Danny Hendler, Shay Kels, and Amir Rubin. "Detecting malicious powershell commands using deep neural networks". In: *Proceedings of the 2018 on Asia conference on computer and communications security*. 2018, pp. 187–197. URL: https://doi.org/10.1145/3196494.3196511.

[12] Sian-Yao Huang, Cheng-Lin Yang, Che-Yu Lin, and Chun-Ying Huang. "CmdCaliper: A Semantic-Aware Command-Line Embedding Model and Dataset for Security Research". In: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. by Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 20188–20206. DOI: 10.18653/v1/2024.emnlp-main.1126. URL: https://aclanthology.org/2024.emnlp-main.1126/.

[13] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. "Beehive: large-scale log analysis for detecting suspicious activity in enterprise networks". In: *Proceedings of the 29th Annual Computer Security Applications Conference*. ACSAC '13. New Orleans, Louisiana, USA: Association for Computing Machinery, 2013, pp. 199–208. ISBN: 9781450320153. URL: https://doi.org/10.1145/2523649.2523670.

[14] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems*. Ed. by Isabelle Guyon, Ulrike Von Luxburg, Samy Bengio, Hanna Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[15] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[16] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves". In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML '06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 233–240. ISBN: 1595933832. URL: https://doi.org/10.1145/1143844.1143874.

[17] Peter Christen, David J. Hand, and Nishadi Kirielle. "A Review of the F-Measure: Its History, Properties, Criticism, and Alternatives". In: *ACM Comput. Surv.* 56.3 (Oct. 2023). ISSN: 0360-0300. URL: https://doi.org/10.1145/3606367.

[18] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (2020), p. 6. ISSN: 1471-2164. URL: https://doi.org/10.1186/s12864-019-6413-7.

[19] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. *Towards General Text Embeddings with Multi-stage Contrastive Learning*. 2023. arXiv: 2308.03281 [cs.CL]. URL: https://arxiv.org/abs/2308.03281.

[20] Koen T. W. Teuwen, Tom Mulders, Emmanuele Zambon, and Luca Allodi. "Ruling the Unruly: Designing Effective, Low-Noise Network Intrusion Detection Rules for Security Operations Centers". In: *Proceedings of the ACM Asia Conference on Computer and Communications Security (ASIA CCS '25)*. Hanoi, Vietnam: ACM, Aug. 2025, pp. 1–14. DOI: 10.1145/3708821.3710823.

[21] Tom Krantz and Alexandra Jonker. *What is alert fatigue?* Accessed: 2025-08-18. May 2025. URL: https://www.ibm.com/think/topics/alert-fatigue.

[22] Hannes Holm and Erik Hyllienmark. "Hide my payload: An empirical study of antimalware evasion tools". In: *2023 IEEE International Conference on Big Data (BigData)*. IEEE, 2023, pp. 2989–2998. URL: https://doi.org/10.1109/BigData59044.2023.10386838.

[23] Zhenyuan Li, Qi Alfred Chen, Chunlin Xiong, Yan Chen, Tiantian Zhu, and Hai Yang. "Effective and light-weight deobfuscation and semantic-aware attack detection for powershell scripts". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2019, pp. 1831–1847. URL: https://doi.org/10.1145/3319535.3363187.

[24] Yong Fang, Xiangyu Zhou, and Cheng Huang. "Effective method for detecting malicious PowerShell scripts based on hybrid features". In: *Neurocomputing* 448 (2021), pp. 30–39. URL: https://doi.org/10.1016/j.neucom.2021.03.117.

[25] Abdulellah Alsaheel, Yuhong Nan, Shiqing Ma, Le Yu, Gregory Walkup, Z. Berkay Celik, Xiangyu Zhang, and Dongyan Xu. "ATLAS: A Sequence-based Learning Approach for Attack Investigation". In: *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3005–3022. ISBN: 978-1-939133-24-3. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/alsaheel.

[26] Dominik Kus, Eric Wagner, Jan Pennekamp, Konrad Wolsing, Ina Berenice Fink, Markus Dahlmanns, Klaus Wehrle, and Martin Henze. "A false sense of security? Revisiting the state of machine learning-based industrial intrusion detection". In: *Proceedings of the 8th ACM on Cyber-Physical System Security Workshop (CPSS)*. 2022, pp. 73–84. URL: https://doi.org/10.1145/3494107.3522773.

[27] Vaishali Vinay and Anjali Mangal. *SCADE: Scalable Framework for Anomaly Detection in High-Performance System*. 2024. arXiv: 2412.04259 [cs.CR]. URL: https://arxiv.org/abs/2412.04259.

[28] Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. "NODOZE: Combatting threat alert fatigue with automated provenance triage". In: *Network and Distributed Systems Security (NDSS) Symposium*. Internet Society, 2019. URL: https://doi.org/10.14722/ndss.2019.23349.

[29] Lukáš Sadlek, Muhammad Mudassar Yamin, Pavel Čeleda, and Basel Katt. "Severity-based triage of cybersecurity incidents using kill chain attack graphs". In: *Journal of Information Security and Applications* 89 (2025), p. 103956. ISSN: 2214-2126. URL: https://doi.org/10.1016/j.jisa.2024.103956.

[30] Raja Muthalagu, Jasmita Malik, and Pranav M Pawar. "Detection and prevention of evasion attacks on machine learning models". In: *Expert Systems with Applications* 266 (2025), p. 126044. URL: https://doi.org/10.1016/j.eswa.2024.126044.

[31] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. "Dos and Don'ts of Machine Learning in Computer Security". In: *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 3971–3988. ISBN: 978-1-939133-31-1. URL: https://www.usenix.org/conference/usenixsecurity22/presentation/arp.

[32] Emily M Bender, Timnit Gebru, Angelina McMillan-Major, and Shmargaret Shmitchell. "On the dangers of stochastic parrots: Can language models be too big?" In: *Proceedings of the 2021 ACM conference on fairness, accountability, and transparency*. 2021, pp. 610–623.

[33] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: https://aclanthology.org/N19-1423/.

[34] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[35] Nils Reimers and Iryna Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. Ed. by Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan. Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3982–3992. DOI: 10.18653/v1/D19-1410. URL: https://aclanthology.org/D19-1410/.

[36] Ilias Chalkidis, Manos Fergadiotis, Prodromos Malakasiotis, Nikolaos Aletras, and Ion Androutsopoulos. "LEGAL-BERT: The Muppets straight out of Law School". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. Online: Association for Computational Linguistics, Nov. 2020, pp. 2898–2904. DOI: 10.18653/v1/2020.findings-emnlp.261. URL: https://aclanthology.org/2020.findings-emnlp.261/.

[37] Phillip Schneider, Markus Voggenreiter, Abdullah Gulraiz, and Florian Matthes. "Semantic Similarity-Based Clustering of Findings From Security Testing Tools". In: *Proceedings of the 5th International Conference on Natural Language and Speech Processing (ICNLSP 2022)*. Ed. by Mourad Abbas and Abed Alhakim Freihat. Trento, Italy: Association for Computational Linguistics, Dec. 2022, pp. 134–143. URL: https://aclanthology.org/2022.icnlsp-1.16/.

[38] Ronghai Yang, Xianbo Wang, Kaixuan Luo, Xin Lei, Ke Li, Jiayuan Xin, and Wing Cheong Lau. "SWIDE: A Semantic-aware Detection Engine for Successful Web Injection Attacks". In: *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2024, pp. 540–554. URL: https://doi.org/10.1145/3658644.3670304.

[39] Yuki Mezawa and Mamoru Mimura. "Evaluating the Possibility of Evasion Attacks to Machine Learning-Based Models for Malicious PowerShell Detection". In: *Information Security Practice and Experience*. Ed. by Chunhua Su, Dimitris Gritzalis, and Vincenzo Piuri. Cham: Springer International Publishing, 2022, pp. 252–267. ISBN: 978-3-031-21280-2. URL: https://doi.org/10.1007/978-3-031-21280-2_14.

[40] Quoc Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *Proceedings of the 31st International Conference on Machine Learning*. Ed. by Eric P. Xing and Tony Jebara. Vol. 32. Proceedings of Machine Learning Research 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1188–1196. URL: https://proceedings.mlr.press/v32/le14.html.

[41] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. "Understanding bag-of-words model: A statistical framework". In: *International Journal of Machine Learning and Cybernetics* 1 (Dec. 2010), pp. 43–52. DOI: 10.1007/s13042-010-0001-0.

[42] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. "Indexing by latent semantic analysis". In: *Journal of the American society for information science* 41.6 (1990), pp. 391–407.

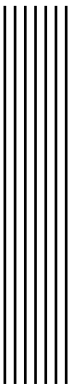
[43] Tianqi Chen and Carlos Guestrin. "Xgboost: A scalable tree boosting system". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016, pp. 785–794.

[44] Gunjan Verma and Ananthram Swami. "Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks". In: *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*. Vol. 32. 2019, pp. 8646–8656. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/cd61a580392a70389e27b0bc2b439f49-Paper.pdf.

[45] Yang Song, Qiyu Kang, and Wee Peng Tay. "Error-Correcting Output Codes with Ensemble Diversity for Robust Learning in Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.11 (May 2021), pp. 9722–9729. URL: https://doi.org/10.1609/aaai.v35i11.17169.

[46] David Bianco. *The Pyramid of Pain*. Accessed: 2025-09-27. URL: https://detect-respond.blogspot.com/2013/03/the-pyramid-of-pain.html.

[47] Andrei Kucharavy. "Fundamental Limitations of Generative LLMs". In: *Large Language Models in Cybersecurity: Threats, Exposure and Mitigation*. Ed. by Andrei Kucharavy, Octave Plancherel, Valentin Mulder, Alain Mermoud, and Vincent Lenders. Cham: Springer Nature Switzerland, 2024, pp. 55–64. ISBN: 978-3-031-54827-7. DOI: 10.1007/978-3-031-54827-7_5. URL: https://doi.org/10.1007/978-3-031-54827-7_5.



Appendix

Questions Posed to Security Analysts

1. *What is the most significant problem you face when implementing new detection rules for deployment?*
2. *What are common evasion techniques that need to be taken into consideration when writing rules?*
3. *What metrics would be most suitable for measuring the effectiveness of SIEM rules?*
4. *Which is the bigger challenge in detection rule design: false positives causing alert fatigue or false negatives?*
5. *Would using the SIGMA rule repository and its open source nature improve security, or could it pose risks?*

Insights from the First Expert

1. The most significant problem is knowing that they work. Rules need to be tested, but in a production environment one is unable to create the event to trigger it since it often involves malicious behaviour.

2. An attacker will often try to obfuscate themselves by creating a process that injects itself into another, hiding in files or utilizing scripts instead of executing atomic commands. In reality it is next to impossible to cover all of the aforementioned, and similar, techniques by analysing user input. One way to mitigate this is to collect logs closer to the hardware. If the input is allowed to be resolved by the kernel and translated to Assembly or machine code it becomes significantly harder for an attacker to hide their true intentions.

3. False positives is important, a rule should not trigger on events that are benign, common, or normal. This is very important to avoid exhausting analysts. However, if a rule is highly critical, generating false positives might be acceptable. Furthermore, wildcards are dangerous but sometimes necessary to specify certain things. If they need to be used the rules can be assigned to different sets for a more efficient classification.

4. In general, it's mostly false positives, you rarely encounter false negatives. If a false negative were to be discovered, it would be detrimental, but can be seen as a way to improve detection. If it is within the scope of what a security operations centre is intended to detect a rule can be written to address the false negative.

5. It increases security and can help with writing rules. However, it requires some manual work since the schema for different sources contains different fields. Additionally, you do not know if they work until you've tested them. When implementing a new rule, it is often beneficial to check if Sigma already has a similar one and use it as a reference rather than reinventing the wheel. Finally the repository tends to provide good sources.

Insights from the Second Expert

1. One of the most significant challenges in formulating rules is determining the most effective approach. Before implementing a rule, it is beneficial to thoroughly test the conditions it will apply to. This requires generating reliable data, setting up a well-structured testing environment, and using the right tools. Additionally, it's crucial to minimize false positives, as they can significantly prolong the tuning process and reduce overall efficiency.

2. Two common evasion techniques is obfuscation and argument rearrangement. In order to make the rules difficult to evade they should be able to handle variations in word order, extra white-space and unexpected characters. Furthermore, detection should be based on artifacts that an attacker cannot easily manipulate, as defined by the Pyramid of Pain concept. As an example hash values of files are trivial to modify, but behaviours such as techniques are inherently difficult to alter.

3. Measuring the number of false positives generated by a rule and comparing it to the true positives is important, but it doesn't provide the complete picture. Rules could produce benign true positives, where the alert is triggered correctly but the cause is legitimate activity that coincidentally matches the detection criteria. To prevent this, rules should be designed to distinguish between malicious and authorized behaviour.

4. False negatives are a greater concern since they are closely tied to proper testing and understanding evasion techniques. If a rule fails to detect actual malicious behaviour, there is no way to mitigate the threat. It is thus better to lean toward the risk of false positives because rules can be fine-tuned by adding exceptions. One has to keep in mind that a rule should not be able to generate excessive false alerts though, as this will put unnecessary strain on analysts.

5. Both perspectives have merit. If detection rules were not shared, everyone would have to conduct their own research, making it much harder to develop effective detection. If the rules are well-written, the fact that they are open-source is not a major concern. However, attackers could analyse the rules and potentially identify gaps to exploit. That being said, SIGMA rules are effective at detecting many known techniques which modern attacks tend to utilize, giving the defender an advantage.

Insights from the Third Expert

1. The rules either generates far too many alerts or none at all. If there are too many alerts it takes a long time to investigate them, leaving no time for actual analysis. Alerts can thus spill over from one day to the next, leading to alert fatigue or aggressive prioritization. If no alerts are generated, you may start questioning whether the rule is functioning correctly and capable of detecting relevant activity. A rule that produces no alerts is particularly problematic because false negatives are difficult to assess without any data to analyse.

2. Common evasion techniques include mixing upper- and lowercase letters, using aliases, and leveraging Base64 encoding. For example, files can be encrypted, and commands in scripts may be fragmented, obscuring their full context and disrupting normal analysis. To counter this, it is essential to rely on high-quality logs, particularly those that capture process creation. Logging closer to the hardware is preferable, as user input alone isn't always reliable. Ultimately, what matters most is understanding what is actually being executed.

3. False positives are important to determine if the rule triggers on the intended events. True benign positives, and rules targeting highly malicious behaviour are also of interest. However, the latter is challenging since they almost never trigger, but when they do, it is a critical situation.

4. Both are equally challenging, but false negatives are more dangerous. It is easy to write a rule that either triggers too much or nothing at all. The difficult part is identifying malicious activity without triggering too many alerts. Too many false positives can lead to alert fatigue, but false negatives are insidious and can create a false sense of security.

5. Overall, it enhances security, as sharing information makes it more difficult for attackers to succeed. If widely adopted, threat actors would be forced to develop more advanced attacks. However, there is a risk that attackers could leverage this knowledge to understand an organization's detection methods. That said, many modern attacks follow the same patterns as older ones, making them easier to detect.