University of Zurich[UZH]

MASTER THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Anomaly Detection with Windows Event Logs: A comparative study between traditional and ML based approaches

Layla Husselman
Zürich, Switzerland
Student ID: 17-733-130

Supervisor: Thomas Grübl, Dr. Alberto Huertas Celdran
Date of Submission: October 1, 2024

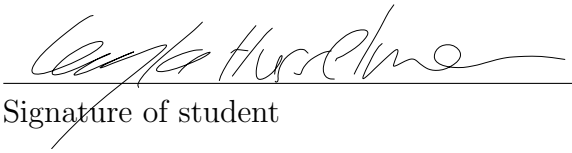
University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Master Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Declaration of Independence

I hereby declare that I have composed this work independently and without the use of any aids other than those declared (including generative AI such as ChatGPT). I am aware that I take full responsibility for the scientific character of the submitted text myself, even if AI aids were used and declared (after written confirmation by the supervising professor). All passages taken verbatim or in sense from published or unpublished writings are identified as such. The work has not yet been submitted in the same or similar form or in excerpts as part of another examination.

Zürich, 

Signature of student

Zusammenfassung

Die zunehmende Zahl von Cyber-Bedrohungen erhöht den Bedarf an besseren Erkennungssystemen. Die Erkennung von Anomalien auf der Grundlage von Protokolldaten kann dazu beitragen, zuverlässigere und sicherere Computersysteme zu erschaffen. Log-Analyse Tools, wie Splunk, können dabei helfen, die zugrunde liegenden Daten zu analysieren und unterstützen somit die Entwicklung von Techniken, um Cyber-Angriffe abzuwehren. Ziel ist es, einen traditionellen, regelbasierten Ansatz mit mehreren Machine Learning (ML) Modellen zu vergleichen und ihre Vor- und Nachteile anhand der betrachteten Angriffe, basierend auf Windows Event Log (WEL)s, zu evaluieren. Die verwendeten ML-Klassifikatoren sind Random Forest, Multilayer Perceptron (MLP), Support Vector Machine (SVM), Logistic Regression, Naive Bayes und One-Class Support Vector Machine (OCSVM). Diese Klassifikatoren wurden verwendet, um ihre Leistung im Vergleich zum traditionellen, regelbasierten Anomalieerkennungsansatz anhand von vier Szenarien zu bewerten: Brute Force, Bypass User Account Control, Remote Services und Disable Windows Event Logging Attacken. Zusammengenommen zeigten die Ergebnisse, dass sich der Random Forest-Klassifikator als das effektivste ML-Modell erwies und den traditionellen Ansatz oft übertraf. Wenn jedoch nur eine geringe Menge an Angriffsdaten verfügbar war, kann der traditionelle Ansatz die ML-Ansätze übertreffen. Zukünftige Arbeiten sollten mehr Angriffe, Taktiken und Techniken beinhalten, um einen umfassenderen Einblick der Unterschiede zwischen den traditionellen und modernen Ansätzen zu ermöglichen. Darüber hinaus, da Angriffsdaten limitiert sind, könnte die Generierung neuer Angriffsproben ausserdem die Robustheit und Verallgemeinerung der Erkennung verbessern. Dies würde es erlauben, bessere Warnsysteme (Intrusion Detection Systems (IDSs)) in der Zukunft zu erschaffen.

Abstract

The increasing number of cyber threats raises the need for better detection systems. Anomaly detection based on log data can help to create more reliable and secure computer systems. Log analysis tools, such as Splunk, can help to analyze the underlying data and therefore support the creation of techniques to counteract cyber attacks. This thesis aims to compare a traditional rule-based approach to several Machine Learning (ML) models and evaluate their strengths and weaknesses in detecting attacks based on Windows Event Log (WEL)s. The used ML classifiers are Random Forest, Multilayer Perceptron (MLP), Support Vector Machine (SVM), Logistic Regression, Naive Bayes, and One-Class Support Vector Machine (OCSVM). These classifiers were used to evaluate their performance compared to the traditional rule-based anomaly detection approach, considering four different attack scenarios: Brute Force, Bypass User Account Control, Remote Services, and Disable Windows Event Logging. Overall, the results showed that the Random Forest classifier proved to be the most effective ML model, often outperforming the traditional However, if only a small amount of attack data is available, the traditional approach can outperform ML ones. Future work should include more attacks, tactics, and techniques to allow for a more all-encompassing view of the differences between the traditional and modern approaches. Moreover, since attack data is limited, generating new attack samples could improve the robustness and generalization of detection. This would then allow to create better Intrusion Detection System (IDS) in the future.

Acknowledgments

Special thanks to my supervisor, Thomas Grübel, who greatly supported me during this thesis with valuable feedback and guidance. I also thank Dr. Alberto Huertas Celdran, Prof. Dr. Burkhard Stiller, and the CSG for giving me the opportunity to write my master's thesis on such an interesting subject in their department.

I would also like to thank my family and friends for constantly supporting me during this time.

Contents

Declaration of Independence i

Zusammenfassung iii

Abstract v

Acknowledgments vii

1 Introduction 1

1.1 Motivation 1

1.2 Description of Work 3

1.3 Thesis Outline 3

2 Background 5

2.1 Anomalies and Anomaly Detection 5

2.2 Windows Event Logs 10

2.3 Attacks 13

2.3.1 Brute Force Attacks 14

2.3.2 Bypass User Account Control 15

2.3.3 Remote Services 16

2.3.4 Disable Windows Event Logging 16

2.4 Splunk 16

2.5 Modern Anomaly Detection Approaches 18

2.5.1 Artificial Intelligence 19

2.5.2 Machine Learning 20

2.5.3 Deep Learning 26

3 Related Work **29**

3.1 Anomaly detection 29

3.1.1 Traditional Approaches 29

3.1.2 Machine Learning Approaches 31

3.2 Windows Event Logs 33

3.3 Limitations 36

3.4 Conclusion 37

4 Methodology and Design **39**

4.1 Data Pre-Processing and Workflow 40

4.2 Model performance evaluation 43

4.3 Feature Selection 43

4.3.1 Brute Force 44

4.3.2 Bypass User Account Control 44

4.3.3 Remote Services 44

4.3.4 Disable Windows Event Logging 45

5 Implementation **47**

5.1 Preprocessing 47

5.2 Implementation 48

5.2.1 Traditional Approach 48

5.2.2 Machine Learning Approach 57

6 Results and Evaluation **75**

6.1 Results . . . 75

6.1.1 Brute Force Attack . . . 75

6.1.2 Bypass User Account Control . . . 85

6.1.3 Remote Services . . . 91

6.1.4 Disable Windows Event Logging . . . 97

6.1.5 Revision of the One-Class Support Vector Machine classifier . . . 104

6.2 Evaluation . . . 105

6.2.1 Bypass User Account Control . . . 106

6.2.2 Remote Services . . . 107

6.2.3 Disable Windows Event Logging . . . 107

6.2.4 Summary . . . 107

7 Discussion **109**

8 Conclusion, Limitations, and Future Work **115**

8.1 Conclusion . . . 115

8.2 Limitations . . . 116

8.3 Future Work . . . 117

Abbreviations **124**

List of Figures **124**

List of Tables **128**

Listings **130**

A Code and Files **135**

A.1 Python Code . . . 135

A.1.1 Splunk Code . . . 140

A.2 Files . . . 146

B Figures **149**

Chapter 1

Introduction

1.1 Motivation

The number of cyber threats has increased in recent years, especially during the COVID-19 pandemic, due to the increasing dependence on the Internet by individuals and corporations. This raises the need for better detection systems and mechanisms to make a rapid incident response possible. It is more likely that there exist vulnerabilities, especially for systems that are more complex and larger. Every day, thousands of systems are attacked, causing reputational damage and financial losses for targeted organizations [42, 31, 29, 56].
To counteract this trend, anomaly detection based on log data has become a more prevalent research field. Anomaly detection is also known as outlier or novelty detection [26, 25]. Anomalies are data points or patterns in the data that differ from normal behaviour. Anomaly detection is the task of finding data points or patterns that are not within the expected norms within the data, given previous observations [69, 60, 71].

Anomaly detection can help to create more reliable and secure computer systems. Discrete event logs, which include chronological event information, allow for time-series analysis and event aggregation and are available in almost all computer and information systems. This makes them an excellent data source for anomaly detection [71, 34].

The main purpose of system logs is to record system states and other relevant events to help debug system failures and perform root cause analysis [49]. However, this is not the only purpose. Log data are also useful, as they can be used to track the history of an intruder in everyday work and provide evidence for further investigation. Such log data also help with investigating the cause after a break-in, as they can help with internal investigations and system-correlated problems. Unfortunately, attacks are developed alongside the security that is trying to protect against these threats. One particular logging system, which contains information about user and system activities in Windows environments, is the Windows Event Log (WEL). WELs are accessible via the Windows Event Viewer, installed by default on Windows systems. WELs include a lot of information about the system and are produced automatically while the system is running, except if they are turned off. However, the amount of WELs produced and their complexity are difficulties

that need to be addressed since there is no time to look through all of them manually and react accordingly. Therefore, it is important to reduce the number of logs that need to be revisited by administrators and only sort out events that may help to find vulnerabilities [34]. The WELs stay locally within the host system and the centralization of the logging process is an important task to make these WELs even more meaningful [56]. Since the Windows Event Viewer only supports basic functions and cannot be used to monitor entire networks, Splunk[1] is better suited to analyze larger amounts of data logs. It can evaluate the WELs in a centralized manner through forwarders and indexers [17, 42].

Splunk, a readily available commodity software, allows for different approaches in anomaly detection [10]. It has its own language, called Search Processing Language (SPL). Splunk helps reduce complexity through centralization [47]. Moreover, Splunk allows the visualization of data on dashboards [47].

Computers can learn without explicit programming with the help of Machine Learning (ML). A system is intended to learn from past or present events and make use of this knowledge to make future predictions or decisions. The primary objective of ML is the creation of a system capable of identifying relevant patterns in data in an automated and correct manner. Deep Learning (DL), which is a subset of ML, brings this to an even more advanced point, as it allows the detection of new or quite complex anomalies. It uses networks capable of unsupervised learning from unstructured data [60].

DL in anomaly detection is a fairly new advancement in this field and has shown success in tackling complexities for which shallow methods could not find solutions [26, 60].
In contrast to ML, DL is designed to work with high-dimensional and multivariate data, which eases the integration of information from multiple sources and eliminates associated challenges with the individual modeling of anomalies for each variable separately and then aggregating the results afterwards [19].
Also, DL approaches are well-adapted to modeling the interactions between multiple variables concerning a given task jointly. DL models even require minimal tuning to achieve good results. Another advantage is their performance, which can potentially scale with appropriate training data. This makes such models suitable for data-rich problems, such as anomaly detection, by offering the possibility of modeling complex and nonlinear relationships within the data [19].
Knowing all this leads to the following research questions:

RQ 1.1: *For which attack scenarios are traditional anomaly detection approaches, conducted via Splunk, well suited?*

RQ 1.2: *Are modern ML anomaly detection approaches well suited to detect certain attack scenarios?*

RQ 2: *In what aspects do ML based techniques surpass traditional ones?*

RQ 3: *Which anomalies can be detected by*

a) Splunk and

[1] https://www.splunk.com/

b) ML approaches

through the analysis of WELs?

1.2 Description of Work

This thesis conducts a comparative study between the "traditional" anomaly detection techniques by using the Splunk Enterprise Security Information and Event Management (SIEM) and the "modern" ML approach on WEL datasets. The used ML classifiers are Random Forest, Multilayer Perceptron (MLP), Support Vector Machine (SVM), Logistic Regression, Naive Bayes, and One-Class Support Vector Machine (OCSVM). To achieve this comparison, this thesis starts with an in-depth analysis of the WEL environment, identifying attack types that are possible to be detected effectively based on the information available in the logs. Based on this, a taxonomy is produced and the prioritization of attacks is done according to [12]. After understanding the WELs, a review is conducted of the existing techniques in ML and for Splunk-based anomaly detection.
In the next step, the scope of the attacks is defined and the theoretical basis is prepared to define the methodology and the design, which are the preparation for the implementation phase.
The following practical part demands the collection of an adequate amount of logs and pre-processes these. The logs should include benign and malign data. Subsequently, conventional anomaly detection techniques in Splunk are designed and queries are implemented to detect the anomalies selected in the previous phase and differentiate them from benign behaviour. For the implementation of ML based anomaly detection techniques, a similar approach is followed and also trained on WEL datasets. Best practices, requirements, and constraints are incorporated within this phase. This is the base for the implementation part of this thesis.
In the last part of this thesis, the effectiveness of the proposed Splunk queries and ML based approach is tested and builds the basis for the comparative evaluation. The evaluation is made according to their accuracy in predicting events correctly, their overall functionality, and their classification Runtime Performance.

1.3 Thesis Outline

This thesis will continue with Chapter 2 Background, where background information on the important topics of this thesis is provided by discussing them in more detail. These topics are Windows Event Logs, the considered attacks Brute Force, Bypass User Account Control, Remote Services, and Disable Windows Event Logging, Splunk, Artificial Intelligence, Machine Learning, and Deep Learning.
The next chapter, 3 Related Work, is further divided into work related to Anomaly detection and Windows Event Logs and followed with found limitations of the analyzed studies and finishes with a conclusion. Anomaly detection gives an insight into the work

correlated to anomaly detection of the traditional and ML approaches. Windows Event Logs dives into what WELs are capable of and which attacks can be recognized through WELs. In the whole chapter, the main focus lies on methods using WELs.
Based on the information found so far, chapter 4 Methodology and Design defines the attacks that are taken into consideration and presents the chosen ones. In addition, it also describes the methodology used and further defines the steps taken to reach the final study design.
The following chapter, 5 Implementation, is about the implementation and the queries written to define and find anomalies.
In 6.1 Results, the findings are presented. The found queries are evaluated and interpreted in 6.2 Evaluation based on the quality of the classification models and the classification Runtime Performance.
This thesis then continues with Chapter 7 Discussion, followed by 8.1 Conclusion wrapping up the thesis, 8.2 Limitations and 8.3 Future Work, giving inputs for the future.

Chapter 2

Background

This chapter aims to provide background information for the most important topics and concepts correlated to this thesis. To understand how the modern and traditional approaches for anomaly detection work, it is fundamental to know what anomalies exactly are. Therefore, this chapter first provides a description of anomalies. It is about what types exist and how they can be categorized and concludes with an explanation of anomaly detection. The next subchapter, 2.2, describes WELs. It explains what logs are and what the Windows Event Viewer is. After giving these definitions, some important attacks are explained and an insight into the methods attackers use is given. Furthermore, this chapter explains in more detail what Splunk is and what it is capable of, but also its use cases. In the last part of this chapter, the modern approaches, represented through Artificial Intelligence (AI), ML, and DL, are explained as well as their relationship and some important methods and techniques are presented.

2.1 Anomalies and Anomaly Detection

For understanding anomaly detection, it is important to understand what anomalies are. They are known under many names as discordant observations, aberrations, exceptions, surprises, outliers, peculiarities, novelties, deviants, or contaminants, depending on the application domain. However, novelties are not necessarily anomalies. Generally, outliers and anomalies are the best-known and most-used terms. Anomalies describe patterns in data that do not meet a certain norm, meaning that they are unusual in some way and do not represent normal behaviour. They are assumed to be different from normal behaviour and rare. They can occur in many forms, which makes it difficult to identify them [69, 58, 57].
Anomalies can be categorized in many ways. In 2021, R. Foorthuis [58] summarized many earlier studies and their anomaly classifications and gave an improved classification for data-centric anomalies in Figure 2.1. This figure shows three different types of data, two cardinalities of relationships, and two anomaly levels, giving a total of nine different anomaly types. These nine anomaly types can then be grouped into three categories, represented as the rows [58].

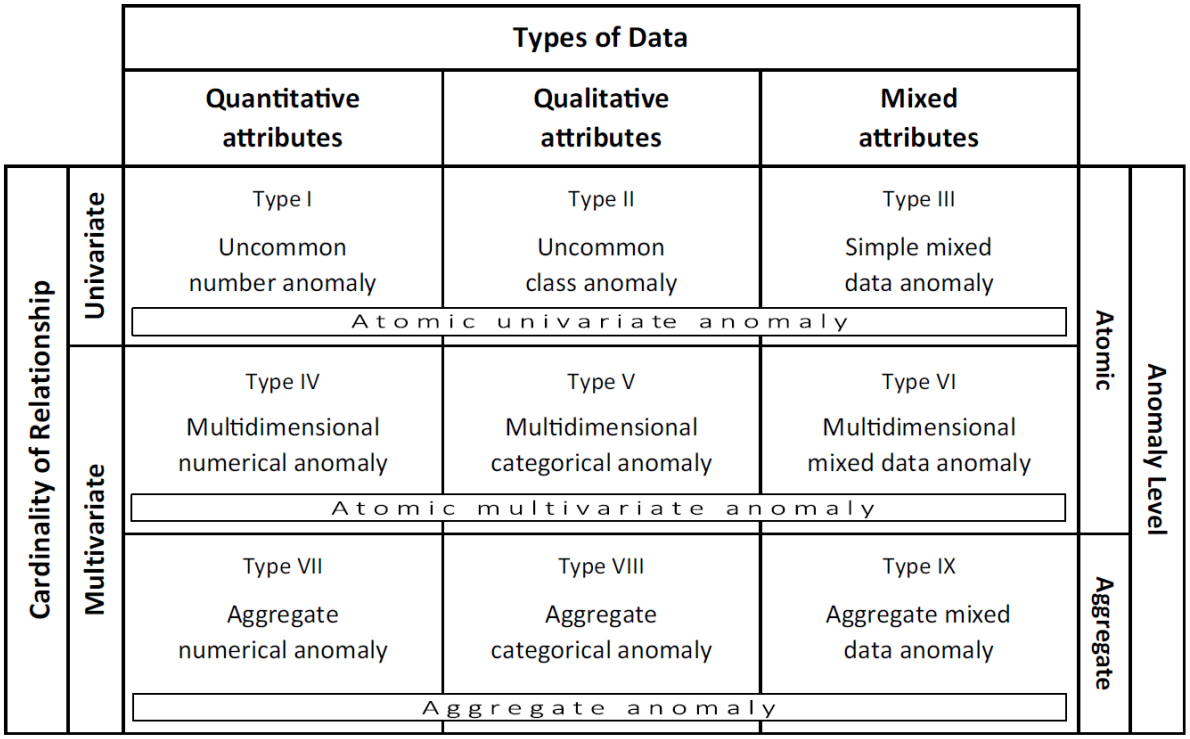


Figure 2.1: Anomaly types reproduced from [58]

In the top of Figure 2.1, the types of data are divided into three categories, as defined by [58]:

Quantitative Attributes: In this category, anomaly attributes only have numerical values, thus allowing arithmetic operations. These attributes can be both discrete or continuous. An example of a quantitative attribute is age [58, 72].

Qualitative Attributes: The attributes in this category are categorical and therefore belong to distinct classes. Such data indicates the presence of a property, however, it does not indicate an amount or degree. An example of this attribute is gender, which could be represented as numbers (0, 1) or as text (male, female). However, for both cases, the representation is nominal, and therefore no arithmetic operations are allowed for qualitative data. Moreover, quantitative attributes always have discrete values [58, 72].

Mixed Attributes: In this case, the variables capturing anomalies are quantitative and qualitative. The data describing the anomaly includes at least one qualitative and one quantitative attribute. As an example, country of birth and body length can be given, where the country of birth is a qualitative, and body length is a quantitative attribute [58].

On the left side of Figure 2.1, it is shown the cardinality of relationships between attributes. The cardinality of the relationship refers to whether it is sufficient to look at just one attribute to detect and define the anomaly types. Sometimes, several attributes

have to be taken into consideration simultaneously to define and detect the anomaly types. It is distinguished between univariate and multivariate anomalies, as defined by [58]:

Univariate: Such anomalies do not have any relationship to other variables describing the anomaly, showing independence between the anomalies' attributes [58, 39].

Multivariate: This describes anomalies, for which more than one attribute has to be taken into consideration because of a relationship between them, which makes them anomalous. Attributes need to be analyzed together in order to take their relationship into account [58, 24, 45, 39].

As a result of the massive data collection nowadays, it is possible for a dataset to contain many attributes, which makes it feasible that an occurrence can be anomalous in one subspace and normal in others. However, it is also possible that an occurrence could be of a certain type of anomaly in one subspace but another type in another subspace [58]. Lastly, on the right side of Figure 2.1, the anomaly level is divided into two subcategories, as mentioned by [58]:

Atomic: Classifies anomalies that are individual data points or low-level cases [58, 57].

Aggregate: Includes groups or collective structures of anomalies [58, 71, 42].

R. Foorthuis [58] mentions two more dimensions, namely the data structure and the data distribution, which are more correlated to the entire data set. They help to distinguish different subtypes within the nine different anomaly types. Finally, Figure 2.1 divides the anomaly types into the three rows:

Atomic univariate anomalies: These three types (types I-III) describe single cases with an anomalous value for one or more individual attributes. Such anomalies are relatively simple to describe and detect since these observations are unusual. Relationships are not relevant for such types of anomalies.

Atomic multivariate anomalies: Such types (types IV-VI) describe single deviant cases, where the relationship between two or more attributes makes the occurrence anomalous. The attribute values are not anomalous on their own, but together they are.

Aggregate anomalies: The last three types (VII-IX) define several anomalous cases that deviate as a collective. Individually, they would not be anomalous. The relationship between the attributes, but also between the anomalous cases, is the most important characteristic to define a derived group and position an occurrence within the set with dependent data.

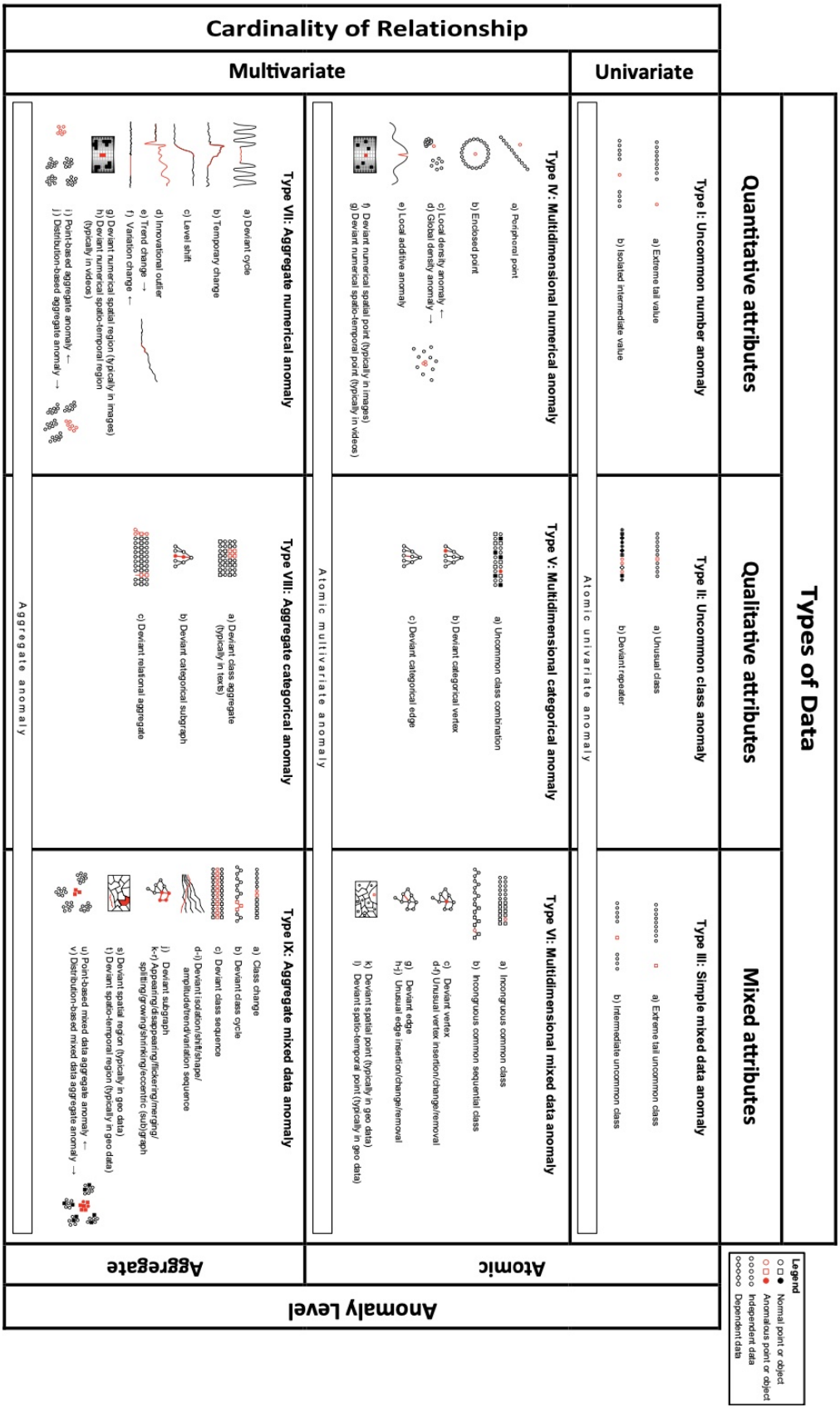


Figure 2.2: Anomaly Subtypes reproduced from [58]

Figure 2.2 shows the 63 different subtypes of the defined anomaly types I-IX.

During this study, only the following anomaly subtypes were found:

- Type II: a (Unusual Class) and b (Deviant Repeater)
- Type IV: c (Local Density Anomaly)
- Type V: a (Uncommon Class Combination)

The Unusual Class subtype describes a rare or unique categorical value for one or more individual qualitative variables and the Deviant Repeater describes values that occur not frequently in normal data, while in anomalous data, they are frequent [58].
Of the Type IV anomalies, only the subtype Local Density Anomaly was found during the work. This subtype describes an anomaly that is only isolated in the context of its neighborhood [58].
The last subtype found during the scope of this thesis was the Uncommon Class Combination. It describes the fact that a given combination of features is anomalous even though it may be normal to find the features not in combination with each other [58].

The other subtypes were not found during the analysis of the data used, which does not mean that they do not exist in the context of anomaly detection. Since they were not found, they will not be explained, but R. Foorthuis defines them. For more details about anomaly subtypes not mentioned here, see his work [58].

After introducing what anomalies are, in the following anomaly detection will be described in more detail. Anomaly detection is a diverse and large field and is used in many application domains like insurance or health care, fraud detection for credit cards, military surveillance for enemy activities, intrusion detection for cyber-security, or fault detection in safety-critical systems but there are many other applications for anomaly detection [69, 10, 25, 26, 57].
In system logs, anomaly detection aims to uncover abnormal system behaviours promptly. This is important for incident management of large-scale systems. Timely detection of anomalies allows system developers or operators to point out issues promptly and quickly resolve them. This can reduce system downtime [66]. In general, anomaly detection refers to the task of finding patterns within data that do not conform to an expected behaviour [69]. This basically refers to finding anomalies. Finding anomalies is crucial for helping to build a trustworthy and secure system [49] but it faces several challenges, as mentioned by [69], [65], [57], and [25]:

- The difficulty of defining a normal region, including every possible normal behaviour and excluding only anomalies. This is challenging since the boundary between anomalous and normal is often not precise, especially if normal and anomalous observations are close together [69, 57]. Even more, a reason for this difficulty arises because anomalies are very diverse [65, 25], as can be seen in Figure 2.2, denoting many different subtypes of anomalies [58]. Oftentimes, normal behaviour evolves and the normal behaviour defined at a certain point in time may not be

sufficiently representative in the future [69, 57]. Moreover, the unknownness of anomalies, meaning it is not known in advance how they behave, their underlying data structures, and their distributions, define challenges [25].

- If anomalies occur as a result of malicious actions, adversaries may adapt to make anomalous observations appear normal. This makes it more difficult to define normal behaviour [69]. Therefore, it is crucial to detect anomalies in time. This timely identification of anomalies helps mitigate threats and maintain the integrity of systems. If such an anomaly is not detected as soon as possible, this can lead to increased exposure to security threats, maybe even data breaches [65].
- Different application domains have different anomaly definitions. This makes it difficult to apply anomaly detection techniques developed in one domain to other domains [69].
- Somehow similar to the last point is the fact of the heterogeneity of anomaly classes because anomalies are irregular. This raises the problem that different abnormal characteristics differ depending on the anomaly class [25].
- The availability of labeled data for the training and validation of an anomaly detection model is also a major issue [69, 65]. (Therefore, mostly unsupervised approaches are used, which are explained in section 2.5.2 [69])
- The rarity and imbalanced amount of normal and anomalous data is another problem to face. This arises through the fact that anomalies occur quite sparsely compared to normal data [25, 65].
- Lastly, data often contains noise, which is sometimes difficult to distinguish from anomalies. Since anomalies sometimes are quite similar to normal data, noise within the data makes it even more difficult to distinguish them [69].

Moreover, since nowadays computer systems and applications are getting more complex, they incorporate more bugs and vulnerabilities an adversary may be able to exploit and launch attacks. As a consequence, anomaly detection has become a more challenging task, and many traditional anomaly detection techniques are no longer effective [49]. The advantages of traditional approaches are discussed in section 2.4, where Splunk represents the traditional, rule-based approach.
In recent years, DL approaches could successfully tackle complexities in many applications [26]. This is discussed in more detail in section 2.5.3.
All the approaches taken into consideration are log-based ones. But to understand these approaches, the next section discusses WELs to give a better understanding of the underlying data log-based approaches used.

2.2 Windows Event Logs

As already mentioned in the Motivation, the main purpose of logs, a specific file type, is to record relevant events. Log files store recorded events in chronologically ordered lists.

WELs are generated on computers running Windows as the Operating System (OS). But there exist other event logs, for example, the syslog, which is based on Linux OS [49, 6]. Depending on the system creating the log, different types of events are tracked and recorded [6]. They can help to determine the cause of an error and the context in which the error occurred. Since, in this thesis, only WELs are taken into consideration, there will be no further explanation for other logging systems. WELs are records of software and hardware events containing information about actions that took place on installed applications, the system itself, or the computer. This includes system-based actions, which are taken by processes executing on the computer, but also user-based actions. It can help administrators diagnose system problems but also predict future issues [46, 36, 30].

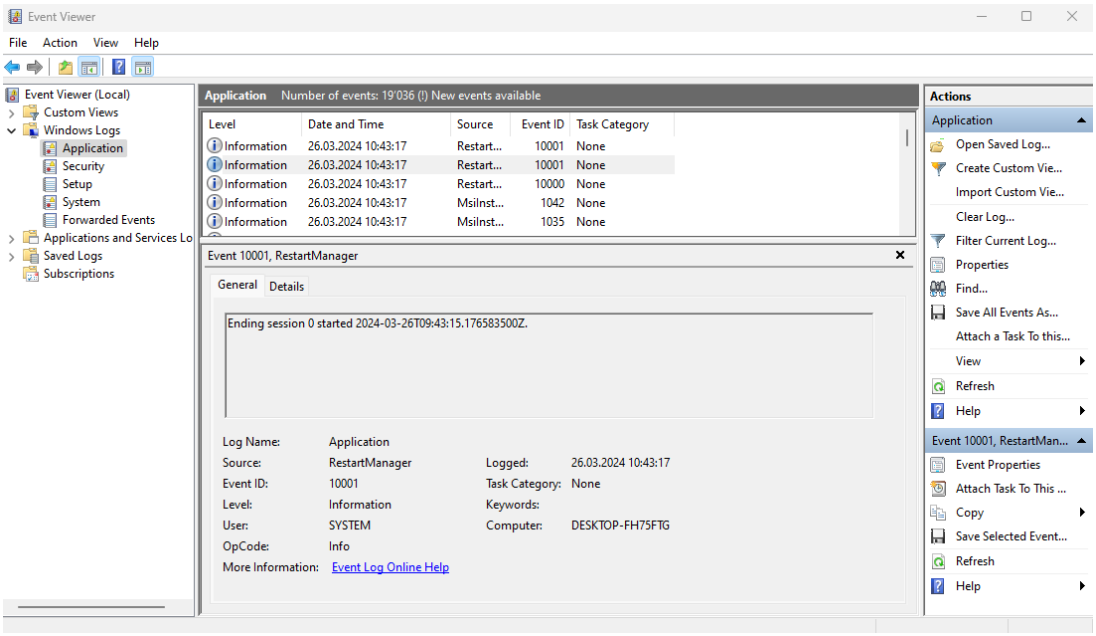


Figure 2.3: Example log in the Windows Event Viewer

Figure 2.3 shows an example log in the Windows Event Viewer. This tool is provided by Windows for managing and accessing the event logs associated with remote and local Windows machines. The different sorts of events logged are explained below in more detail. Clicking on any of these drop-down options will open the corresponding log files [46]. Following, the areas mentioned above are explained in more detail, as defined by [36]:

Application events are connected to events related to installed software on the local computer.

Security events store information that is based on the audit policies of the Windows system.

Setup events keep track of occurrences involving the "Active Directory" on domain controllers, but they also include enterprise-focused events that are related to the control of domains.

System events relate to Windows-specific system incidents.

Forwarded Events are events that originate from different machines within the same network.

By clicking on one of the options on the left, mentioned above, all events are displayed in the middle part of the Event Viewer. The upper part of it shows all the events that took place on a list. They can be sorted according to the headers ("Level", "Date and Time", "Source", "Event ID", "Task Category", "Log" and "Computer"). The bottom panel of the middle part shows more details for the selected event log from the list above [46].
All WELs are organized in a standardized way to make it easier to understand. As shown in Figure 2.3, a WEL usually consists of the information shown in Figure 2.3 in the middle panel within the bottom part. Following, the most important elements are explained in more detail as defined by [46]:

The **Log Name** shows the name of the log. The name is one of the options from the drop-down on the left side of the Event Viewer. Application event logs include events related to installed or being installed services and applications. Security event logs include audit records and system event logs include events logged by components at the system level.

Source is the name of the component that logged the event. Mostly this is the name of the process or the application that wrote the event log.

Event ID serves as an identifier for the logged events and it should connect to a message that describes the cause of the problem.

Level can take the values verbose, information, warning, error, and critical and indicates the severity of the event logged.

User shows the user who was logged in to the particular Windows machine when the event was logged.

Logged refers to the date and time the event was logged.

Task Category can be used as information to assist with debugging issues.

Computer shows the machine name that logged the event.

There are three more fields: "OpCode" (can provide detail on the source application, component triggering the message or service), "Keywords", and "More Information" (which is a hyperlink that leads to the Microsoft-support page on the internet) [7].
Logs in the Event Viewer can be filtered and sorted such that an administrator can find relevant information faster [46]. However, the Event Viewer in Windows allows only for basic filtering of events and does not provide options to analyze for correlations or other helpful tools that would help an administrator to find a problem efficiently and quickly [34].
The last panel of the Windows Event Viewer on the right side offers several options as to filter for specific logs. Saving event logs may also be useful, which allows the saving of

event logs for the outside use of the component [46].
It is important to know that event logs are the data source of many anomaly detection approaches, which will be further explained in the corresponding sections.

2.3 Attacks

As of today, technology helps people in their everyday life in many different fields to make things easier or faster or to help people with certain tasks. However, with this increasing use of technology, the attack surface has become larger as well such that now a wide variety of cyber attacks exist, which target individuals but also entire organizations or agencies [41, 32]. It is also a key challenge to distinguish malicious users from actual customers [69]. Threat actors typically use techniques for evading the security within computer systems and target misconfigurations and vulnerabilities to gain privileged access to resources that are restricted and therefore cause damage to a company. This is also known as technical debt or tech debt and describes the cost incurred when organizations do not fix vulnerabilities and problems that may affect them in the future. If such tech debt accumulates, this causes existing problems to get even worse over time [42, 9]. A cyber attack is defined as an intended malicious attempt to gain access to computer systems, computer networks, or to a computer to cause damage. Any group or individual can launch a cyber attack from wherever they are by using one or more attack strategies. Such attackers are cyber criminals and are often referred to as threat actors, hackers, adversaries, or bad actors. These can be individuals or groups and they try to find weaknesses or vulnerabilities in the target computer system that they can exploit for gain according to their computer skills. However, such threat actors try to use advanced techniques to hide the malicious behaviour, making it even more difficult to detect such actions [5, 57].
Cyber attacks are designed to cause damage and can have different objectives. Many such attacks try to spread misinformation, access sensitive information, steal information, cause disruption, take revenge, create financial losses or gains and are also used for espionage to name a few goals of attackers. Developing adequate security measures requires a deep understanding of such attacks [41, 5, 32].
Adversaries use a wide variety of attack techniques, but the choice of technique they use is largely dependent on whether they want to attack an untargeted or a targeted entity. Attackers try to get access to as many devices as possible in an untargeted attack. They try to achieve this by looking for vulnerabilities in software code without being detected or blocked. Adversaries try to get access to a specific target in a targeted attack. The techniques they use vary depending on the objective of the attack. Cyber attacks often take place in stages, starting with the scanning for weaknesses, then adversaries initiate the initial compromise and lastly execute the full attack [5]. [42] proposes four rules to capture the phase of a cyber-attack. The first rule is the “Successful logins”, where the attacker gains access. The second rule flags communication attempts with the command and control server, which is part of the Exfiltration phase. They call this phase the “Malware IPS”. The third rule is called “Security log was cleared”, which looks for the Windows event 1102. This notifies the ending phase of a cyber-attack. The fourth rule is for reporting and is based on the Windows event 4625 that is logged for all failed logon

attempts [42].

To prevent cyber attacks, it is essential to reduce the risk by preventing attempted attacks from entering the IT system of the organization, detecting intrusions, and disrupting attacks that are already in motion as early as possible. However, this is a very challenging task. There are several best practices to reduce the risk, according to Gillis and Pratt [5] they include:

- The implementation of perimeter defenses, as firewalls. They help to block attack attempts, but they also block access to known malicious domains.
- The adoption of zero-trust frameworks, through which organizations need to verify every access attempt to its network or system for both internal users and from other systems.
- The use of software against malware, like anti-virus software. This adds a layer of protection.
- Monitoring and detection programs to alert and identify suspicious activities.
- The use of patch management to tackle known software vulnerabilities adversaries could exploit through updates.
- The setting of appropriate security configurations, user access controls, and password policies.
- The creation of incident response plans as a guide after a breach.
- The initiation of security teams, which actively look for the presence of adversaries and suspicious activities.
- The training and education of users about attack scenarios and how they can take part in the protection of the organization.

In the following subsections, the attacks taken into consideration for the thesis are explained in more detail. It is important to know that this list is not complete and there exist many more attacks.

2.3.1 Brute Force Attacks

Brute Force attacks are part of the Credential Access techniques in [12]. This method uses a trial-and-error procedure to guess login information and encryption keys such that an adversary may gain unauthorized access to a system [28, 37]. The Brute Force method needs exhaustive effort rather than using intellectual strategies. This means an adversary tries many combinations of legal characters and can be effective if common passwords are used. A Brute Force attack can take just a few seconds to days, depending on the password strength. If an attacker can successfully get access to a system through this attack, it is possible to install malware, conduct data breaches, or shut down web applications

[28]. There exist subtypes of Brute Force attacks like the dictionary attack, where an adversary tests combinations of common words or phrases, or the Reverse Brute Force Attack, where the attacker knows the password instead of the account and needs to find the account matching the password [37].

According to MITRE ATT&CK [12], the technique of Brute Force attacks is further divided into four sub-techniques, which are Password Guessing, Password Cracking, Password Spraying, and Credential Stuffing.
In Password Guessing, the attacker has no knowledge of credentials within the system and therefore tries to get access to accounts by guessing the correct password for a given account [12].
Password Spraying is similar to the previous sub-technique, however, it tries one password or just a small list of passwords for many accounts within the system. This avoids account lockouts that normally occur if Brute Force attacks are used on a single account [12].
In Password Cracking, password hashes are tried to be obtained to get the plaintext password of an account. Lastly, Credential Stuffing is used by adversaries if a breach occurred that dumped credentials of unrelated accounts. Attackers then may use this information to gain access to a target account through credential overlap [12].

2.3.2 Bypass User Account Control

This is a technique of Abuse Elevation Control Mechanism, according to MITRE ATT&CK [12] it is listed under Privilege Escalation. Currently, there exist many methods to bypass the User Account Control (UAC). For this technique, attackers may bypass UAC mechanisms and get process privileges on a system. The windows UAC grants a program permission to elevate its privileges to perform a task with administrator-level permissions. This may be possible by prompting the user for their confirmation. The consequences for the users range from denying the operation under high enforcement to permitting the user to act if they are in a local administrator group and click through the prompt, this may even let them input an administrator password, which then completes the action. If a computer has the UAC protection level set to anything except the highest level, some Windows programs are able to elevate privileges or even execute some elevated Component Object Model objects in the absence of the need to prompt the user through UACs notification box. It is also possible for an adversary to inject malicious software into a trusted process. Like this, it is possible to gain elevated privileges without the user being prompted [12].
It is also possible to bypass the UAC through Lateral Movement techniques if the credentials for the account with the administrator privileges are known. This is possible because UAC is just a single system security mechanism and hence, the integrity or privilege of a running process on a system will not be known on other remote systems, therefore, the default is set to high integrity [12].

2.3.3 Remote Services

For this attack technique, which includes eight sub-techniques (Remote Desktop Protocol, SMB/Windows Admin Shares, Distributed Component Object Model, SSH, VNC, Windows Remote Management, Cloud Services, and Direct Cloud VM Connections), adversaries use valid accounts for logging into services that accept remote connections. The attacker then performs actions as the logged-on user. With an organization environment, workstations and servers can be organized into domains. These provide a centralized identity management that allows users to log in by using a set of credentials across the network. If now an attacker can obtain this set of domain credentials, they may log in to many different machines through the use of remote access protocols like the Secure Shell (SSH) or the Remote Desktop Protocol (RDP) [12].
Also, legitimate applications may use Remote Services to access remote hosts. Adversaries could abuse such applications to gain the possibility of remote code execution and therefore may perform Lateral Movement. Lateral Movement is a tactic where adversaries enter and control Remote Services on a given network. They often explore the network to find their target and then gain access to it. It is possible that adversaries install their own remote access tools to do the Lateral Movement. They may also use legitimate credentials, which may be stealthier [12].

2.3.4 Disable Windows Event Logging

Since it is important to have WELs to work with the method provided in this thesis, it should be noted if the logging of the Windows events is disabled. This Defense Evasion technique belongs to the Impair Defenses technique in [12]. If an adversary can disable the Windows event logging, they can operate in the system while at the same time leaving less evidence of an attack and therefore limit the amount of data that can be leveraged for audits and detections [12].
WELs record system and user activity. This data can then be used by analysts and security tools to generate detections. The service starts when the system powers on and the EventLog service maintains logs of events from system components and applications. Therefore, an adversary can target the system-wide logging or only the logging of a particular application [12].

2.4 Splunk

Splunk, a readily available commodity software, implements both signature- and anomaly-based detection methods based on log data and therefore helps to make such methods more accessible [10]. In general, it offers a big data platform to help its clients with different tasks like cybersecurity, network operations, and observability [11]. Splunk introduces itself in the following way: *"Our purpose is to build a safer and more resilient digital world. Every day, we live this purpose by helping security, IT and DevOps teams keep their organizations securely up and running. When organizations have resilient digital systems, they can adapt, innovate, and deliver for their customers."* [11]

Overall, Splunk helps in removing complexity through segmentation over different tools. This is achieved by combining the data preprocessing, the implementation of the classifier, and even the visualization of the data [47]. In Splunk, the visualization of data can be done by creating dashboards to get a better overview and summarize the findings [17].
It also offers some additional tools that can be helpful for anomaly detection for example the Machine Learning Toolkit (MLTK). Without the use of the MLTK, Splunk provides the option to implement powerful rule-based mechanisms. This implies that in order to flag anomalies, it is needed to define rules in advance. Splunk then detects the anomalies according to the predefined rule set. The rule set can detect known anomalies very well, however, it can be especially problematic for new attack patterns, which are not detectable by the rule set. Moreover, the defined rules should constantly be updated and improved to match unknown attacks as well as changing properties within the system, which is very time-consuming [25, 17].

Splunk can read in data if it is textual, but it can not read binary data like images and sound files. For some binary files, however, it is possible to convert them into textual information. These files can then be used by Splunk such that they can be applied for indexing and searching. During indexing, Splunk reads machine data from a variety of sources. Common input sources are files, the network, and scripted inputs [17].
Splunk has its own language, called Splunk SPL. This language can be used to extract features and clean or remove outliers [11].
Splunk offers a wide variety of software, Application Programming Interface (API), and apps, which makes it useable in many areas. One of Splunk's software products is Splunk Enterprise SIEM, a tool to manage security events. It enables its users to search, analyze, and visualize log and event data, providing insights and allowing the user to react in real-time data. This data is gathered from components of the IT infrastructure or business of the user. After defining the data source, the data stream is indexed and parsed into a series of individual events, which can be viewed and searched [11, 68, 42].
The Splunk Enterprise environment can be extended depending on the needs of the user by apps. These apps are collections of configurations, views, knowledge objects, and dashboards that run on the Splunk platform [68].
In the following, the most important features of Splunk Enterprise are listed:

Indexing: With Splunk Enterprise, it is possible to process and store data that represents a business and its infrastructure, enabling the collection of data from different devices, applications, and more. "Once the data is collected, the index segments, stores, compresses the data, and maintains the supporting metadata to accelerate searching." [68] Indexing combines the data gathered from diverse locations into centralized indexes. This erases the need to log in to many different machines to access the data [17].

Searching: This is the primary way to navigate through data in Splunk Enterprise. It is possible to save a search as a report and to use it for dashboard panels. Through searches, it is possible to retrieve events from an index, calculate metrics, predict future trends, identify patterns in the data, and search for specific conditions, where also a rolling time window can be used [17, 68].

Dashboards: Containing panels of modules as search boxes, charts, fields, and more, dashboards are used to display results of completed and real-time searches. Dashboard panels are normally connected to pivots or searches [17].

Besides these three features, there exist others like alerting, which allows the notification if search results for real-time and historical searches meet pre-configured conditions, pivots, reports, and the data model [68].
The structure of Splunk Enterprise SIEM has three major components, which are the search head, the indexer, and the forwarder. Forwarders collect and forward the information to an indexer or other forwarders. An indexer can get data from several forwarders, as depicted in Figure 2.4. The indexer is used for processing and storing logs in indexes. Raw data goes into the indexer and gets parsed. Finally, it is stored in a directory with a timestamp, where raw data and indexes are saved. The search head enables querying for events by using the SPL after the data is indexed. It can also be used to create different charts, reports, and dashboards [42].

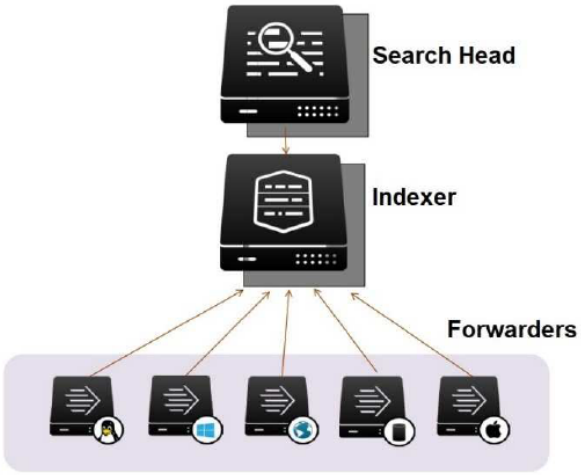


Figure 2.4: Splunk structure reproduced from [42]

2.5 Modern Anomaly Detection Approaches

As noted in the Introduction, the modern approaches describe those including AI, ML and DL. These techniques arose especially in the last years, making it possible to detect different patterns of attacks, traditional approaches could not detect as the detection of zero-day attacks [20]. Figure 2.5 shows that DL is a subset of ML, which are both part of AI. These approaches are explained in more detail in the corresponding subchapters of this chapter. The modern approaches arose also as a reaction to the Internet of Things (IoT), which is a major data generation source for intelligent processing and analysis [64].

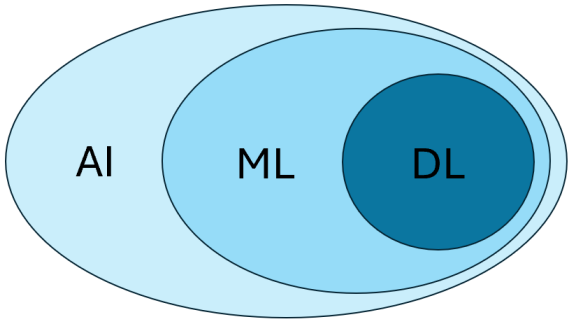


Figure 2.5: Relationship between AI, ML and DL as defined by [64]

AI, ML, and DL are sometimes used interchangeably, but there are differences. AI, which was coined in the 1950s, refers to the attempt to imitate human intelligence and their behaviour through machines and it is an always-changing set of capabilities [52, 55]. ML allows software applications to improve their performance by predicting outcomes more accurately, and these algorithms do not need to be programmed to do this explicitly. Such algorithms use historical and current data as input for algorithms and predict new output values after a training phase [52, 18, 55].
DL is based on our understanding of neural networks and imitates the way humans gain knowledge. This makes collecting, examining, and interpreting enormous amounts of data easier and faster. It also empowers software through the use of neural networks to understand complex patterns [18, 55].

2.5.1 Artificial Intelligence

According to Ertel and Black [23], the journey of AI already started in the 20th century and so far, many advancements contributed until we now have our current state of the art with generative AI, data mining, better processing power to name a few [23, 64, 52].

It is important to be aware of AI's potential drawbacks. One of which is the ethical aspect. This can be somewhat problematic since AI systems use past data for training and therefore learn past patterns, which in turn will act accordingly [52]. This can be dangerous in the way that our world keeps evolving and our view of certain topics changes. An example could be the working of women in the past has now changed quite significantly, since nowadays many women are working compared to the past. The reinforcement of the past through AI can be caused by the fact that such tools are only as smart as the input data they are fed with. Since a human selects the data that will be used to train an AI program, there exists a potential for bias and we need to be aware of this fact. Besides bias, AI faces several other challenges, like the problem of explainability, justifiability, legal issues, privacy concerns, and more [52].
However, AI is now more popular and better known because of its use for high volume data but also in advanced analytics, which nowadays got easier through the improved storage and computational capacity [64]. Moreover, it helps by automating tasks that are normally done by humans, sometimes AI can even better solve certain tasks than humans,

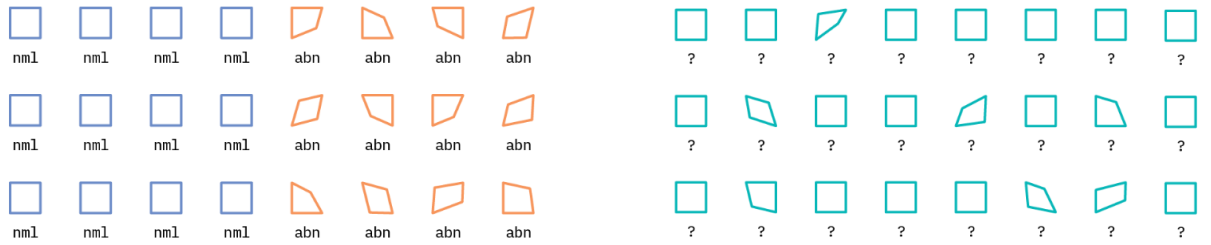
especially if tasks are repetitive and detail-oriented, it can complete the task quickly and with relatively few mistakes [52].

AI is the attempt to simulate intellectual tasks typically performed by humans and their behaviour. It therefore refers to machines that mimic human intelligence and human cognitive functions like learning or problem-solving. Moreover, it uses automation and predictions to be more efficient and to solve complex tasks humans already finished. AI is a field including ML and DL but also comprises other methods not requiring learning ever again [23, 64, 52, 55]. Early AI systems were computer programs that were rule-based, and therefore, they lacked the flexibility to learn and evolve. But today, AI systems use large amounts of training data, analyze it for patterns and correlations, and use these to make predictions for future states. At today's state, it is possible to create images, music, text, and other media through generative AI, which was fed a huge amount of examples to create realistic outputs [52, 18].
AI can be divided into weak and strong. Weak AI or narrow AI is designed and trained to fulfill specific tasks. Strong AI or artificial general intelligence can simulate the behaviour of the human brain. This means that when confronted with an unfamiliar task, strong AI can use fuzzy logic [52]. This stays in contrast to the boolean logic. Fuzzy logic uses the boundaries zero and one which are both included, but there are degrees in between [70]. Strong AI applies knowledge from domains to different ones to find solutions autonomously [52].

2.5.2 Machine Learning

ML, a subset of AI, is used for optimization when set up correctly and can help to make predictions that reduce errors that otherwise arise from merely guessing. ML is seen as the science of getting a computer to act without explicitly programming it. This tries to mimic human intelligence and stays in contrast to the rule-based AI systems that use if-then conditions. ML uses a different technique. It lets the machines learn on their own by going through vast amounts of data and analysing patterns within. Often, ML algorithms use big data and statistic formulas [18, 52, 16].
Overall, [55] gave a good definition of ML, it defines ML as a *"branch of AI trained on statistical models and algorithms, which enable[s] it to make predictions and decisions. Using training and historical data, machine learning algorithms can improve and adapt over time, enriching its capabilities."*
Moreover, the ML process consists of the steps of receiving data and analysing it, finding patterns in the data, and, according to that, making predictions and giving answers [55]. For doing so, it is key to do accurate feature selection and engineering. Since the features are used to find patterns within data, they need to be selected carefully, otherwise, it is possible to have reduced accuracy and efficiency of the anomaly detection model. Only relevant features should be selected and maybe feature engineering is needed, which may involve the combination or transformation of features within the given data to create a better representation for the model [65, 57].
Classic machine learning, excluding DL, depends on human interventions to allow computer systems to recognize patterns, learn, provide accurate results, and perform specific tasks. Within ML, there exists the subcategories DL and reinforcement learning, and at

its intersection is the so-called deep reinforcement learning. DL will be explained in more detail in the next section (2.5.3). In reinforcement learning, a computer can learn by interacting with the surroundings and getting feedback for its actions, which can be either rewards or penalties this is a kind of trial-and-error approach. Reinforcement learning can also be seen as a type of algorithm, where the two other types that exist besides are supervised and unsupervised learning [52, 16, 18, 55, 19].
More important for anomaly detection and this thesis, however, are supervised and unsupervised learning techniques, where the latter is even more crucial. IAnomalous samples in anomaly detection mostly occur only in a small percentage of the dataset. In labeled data, if available at all, there are considerably more normal data samples than abnormal ones. The availability of labeled data highly influences the choice of whether to use supervised or unsupervised approaches [19].



(a) Supervised ML with labeled training data, nml stands for normal and abn means abnormal

(b) Unsupervised ML with unlabeled training data

Figure 2.6: Supervised and unsupervised ML training inputs reproduced from [19]

Figure 2.6 shows the difference in training sets for supervised and unsupervised learning. In supervised learning, shown in 2.6(a), the machines create a function that maps inputs to outputs on the fundament of input-output pairs. A data scientist then supervises the process by either confirming the machine's accurate response or correcting wrong responses. Labeled examples for all possible types of anomalies that may occur are assumed to be in the data to correctly classify them. However, this is normally not the case, since there are many different types of anomalies and they can take different forms and sometimes even novel anomalies arise during testing time. The inputs consist of several features and the output is a categorical variable in this case, which will also be used for this thesis. If sufficient normal and anomalous data is available, anomaly detection can be reframed to a classification task through which machines can learn to predict accurately whether a given input is an anomaly or not. This makes supervised algorithms efficient for specific tasks. In anomaly detection, there may arise a problem with a strongly imbalanced data set of normal and anomalous data. Even worse, many diverse classes of anomalies, each being highly underrepresented, can be a problem too. For these reasons, approaches that are more effective at identifying unseen anomalies and generalize better are preferable [19, 55, 51].
Because labeled anomalous data is rare, unsupervised techniques are regularly used in the anomaly detection field. Unsupervised learning does not have labeled input-output examples, instead, unlabeled data is given to a machine, as shown in Figure 2.6(b). The machine then tries to find patterns within the input features without human intervention.

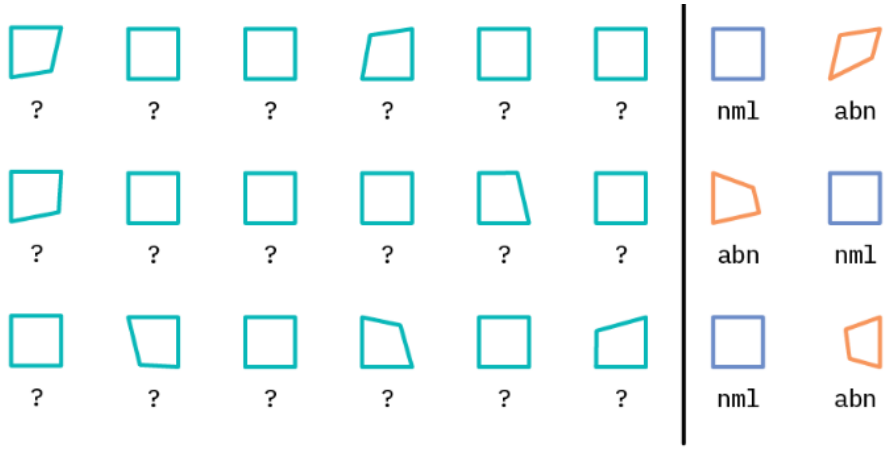


Figure 2.7: Semi-supervised learning, nml stands for normal and abn means abnormal, reproduced from [19]

The nature of anomalies is often highly specific, therefore, many anomalies from a completely unsupervised manner might correspond to noise, therefore not being of interest for the current task [19, 55].
Yet, there exists an approach combining supervised and unsupervised learning. It is called semi-supervised learning and uses an extensive amount of unlabeled data for training, which is depicted on the left in Figure 2.7, and a small amount of labeled data for testing as depicted on the right side of the figure. This fits anomaly detection well since there exists a huge amount of normal data and just very few anomalous data to learn from. The semi-supervised learning approach is well suited if several normal classes and anomalous classes exist and maybe new kinds arise over time [19].

As mentioned before, the huge amount of normal and the small number of anomalous data is a problem, it is known as the class imbalance problem. This can lead to a model, which maybe is not robust through the skewed data. This leads to a poor performance in classifying anomalous examples. However, normal data most likely will be classified quite accurately. Furthermore, there may arise the problem of False Positive (FP) data, where a normal example is misclassified as anomalous, and False Negative (FN), where an anomalous example is wrongly classified as normal. These errors are important to consider but moreover, it is also crucial to consider the precision and recall to determine how accurate a certain model performs. The formulas of precision and recall are provided in the Methodology and Design chapter (Equation 4.1 shows the precision formula and Equation 4.2 shows the recall formula). Optimizing for precision can be appropriate for reducing human workload or when the cost of failure is small. An optimization for recall can be useful if there are high costs for FN [19]. Other often used metrics for evaluation of the performance of a ML algorithm are the accuracy and the F1-score [20].
Accuracy evaluates the accuracy of the predicted samples [20]. The F1-score, also called the F1-measure, is used to measure the accuracy of a model and is defined as the combination of the precision and the recall values. The higher the F1-score, the better the anomaly detection of the model [20, 71].

Different techniques for the anomaly detection task exist, as shown in Figure 2.8 below. However, for the modern approach that will be used for this thesis, a classification method

with different ML classifiers, including the mentioned OCSVM, will be used. Therefore, the other methods will not be discussed in more detail.

Anomaly Detection Method	Assumptions	Anomaly Scoring	Notable Examples
Clustering	Normal data points belong to a cluster (or lie close to its centroid) in the data while anomalies do not belong to any clusters.	Distance from nearest cluster centroid	Self-organizing maps (SOMs), *k*-means clustering, expectation maximization (EM)
Nearest Neighbour	Normal data instances occur in dense neighborhoods while anomalous data are far from their nearest neighbors	Distance from _k_th nearest neighbour	*k*-nearest neighbors (KNN)
Classification	• A classifier can be learned which distinguishes between normal and anomalous with the given feature space • Labeled data exists for normal and abnormal classes	A measure of classifier estimate (likelihood) that a data point belongs to the normal class	One-class support vector machines (OCSVMs)
Statistical	Given an assumed stochastic model, normal data instances fall in high-probability regions of the model while abnormal data points lie in low-probability regions	Probability that a data point lies in a high-probability region in the assumed distribution	Regression models (ARMA, ARIMA)
Deep learning	Given an assumed stochastic model, normal data instances fall in high-probability regions of the model while abnormal data points lie in low-probability regions	Probability that a data point lies in a high-probability region in the assumed distribution	autoencoders, sequence-to-sequence models, generative adversarial networks (GANs), variational autoencoders (VAEs)

Figure 2.8: Anomaly detection methods, reproduced from [19]

ML and DL based classification approaches used on selected features allow for enhanced performance compared to traditional method-based and rule-based approaches. Because of their classification and prediction proficiency, they have been widely used since 2016 [20, 19].

The Random Forest classification uses multiple decision trees with different random subsets of features and the data to get to the conclusion of classifying given data. Each decision tree provides its prediction. The Random Forest then takes the most popular result for its class prediction. In Figure 2.9 below, the path a tree chooses is shown by the green points, and the class each tree decides on is visible below the trees. Since four out of the five trees predicted “Cat”, the Random Forest returns “Cat” as its class prediction [3].

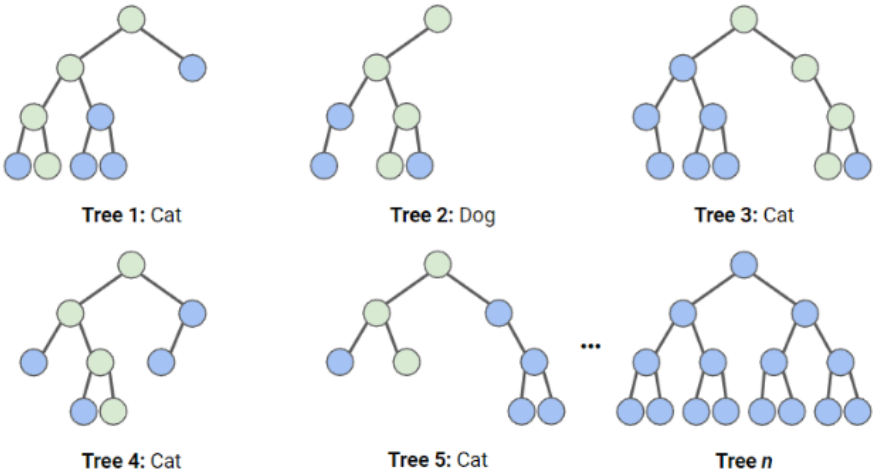


Figure 2.9: Decision trees, reproduced from [3]

SVMs try to find a hyperplane in a space with n dimensions, where n is the number of features. This hyperplane should then classify the data points distinctly. Figure 2.10 shows on the left-hand side how two classes could be separated by different possible hyperplanes. The right-hand side shows what the SVM aims for.

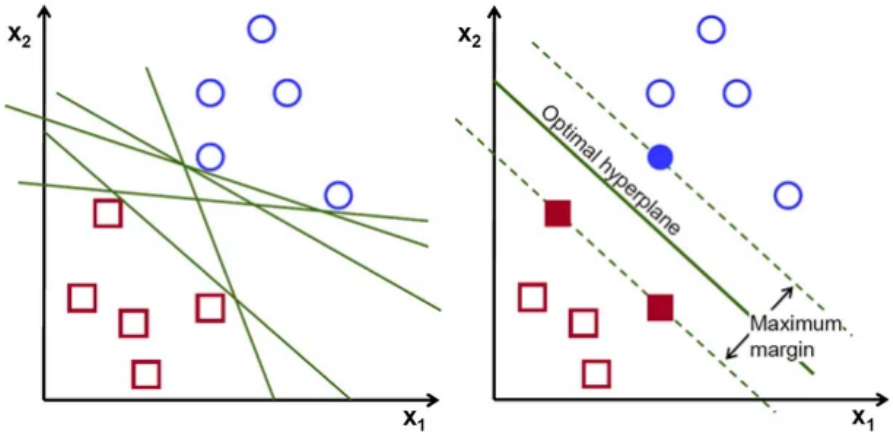


Figure 2.10: Possible hyperplanes, reproduced from [61]

Data points closer to the hyperplane are called support vectors. They influence the orientation and position of the hyperplane. Using support vectors allows to maximize the margin of this classifier [61].

An artificial neural network is a kind of ML model that was inspired by the function and structure of the human brain. MLP is a kind of artificial neural network. It consists of multiple layers of neurons. In MLPs, information only flows in one direction, namely from input to output. Therefore, they are part of feed-forward neural networks. The neurons in this classifier are fully connected and use a nonlinear activation function. It is often used to distinguish not linearly separable data [63].

Logistic Regression was also chosen as a classifier since it is well-suited for classification tasks. It is a discriminative and probabilistic classifier and uses a supervised ML approach. The Logistic Regression uses the sigmoid function to decide on the output class 1, meaning it is a member of the class or 0 if it is not. The sigmoid function, which is also called the logistic function, is shown in Figure 2.11 below. It takes a real value and then maps this value to the range (0,1) [14].

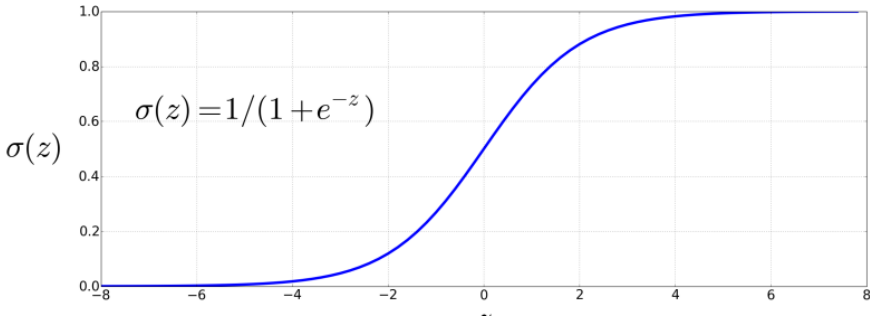
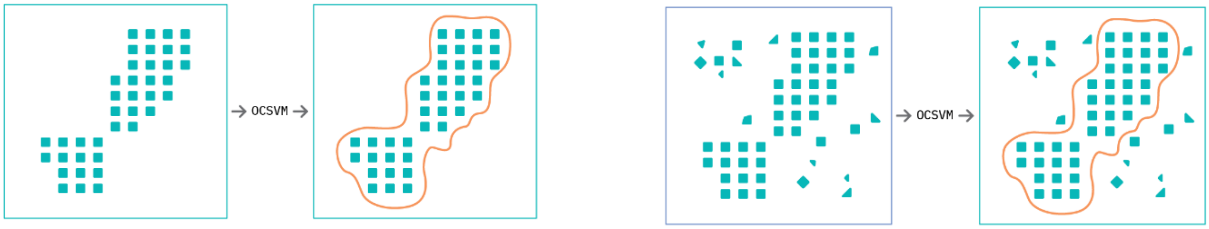


Figure 2.11: Sigmoid function, reproduced from [14]

If the probability returned by this function is smaller than 0.5, it will be counted towards the class, otherwise, it is not counted towards this class [14].

Naive Bayes is a generative and probabilistic classifier that is fast also for large datasets. It is based on the Bayes Theorem [2] [14] and is called naive because it assumes the features to be independent, which in reality is rarely the case. It calculates the probability that event A takes place, given event B happens [2].

In the last part of this section, the used ML classifiers are introduced. For this study, the OCSVM will also be used to classify the data. Since normally, datasets with anomalies mostly have normal data points and just a few anomaly data available, the OCSVM is trained with normal data points and everything not corresponding to the learned normal is an anomaly. OCSVMs are used for novelty detection and the detection of unfamiliar samples. They learn a decision function that specifies regions where the probability density of data is high. Figure 2.12(a) shows how such a boundary could be set around the training data. To apply this OCSVM for anomaly detection, the model is trained to use only or mainly normal data, without or with just a few anomalous samples. In most implementations of OCSVMs, the model gives back an estimate of the similarity of data points compared to the samples seen during the training phase. Such models are valuable when anomalies are expected to occur rarely and differ significantly from normal data points. If they do not differ significantly from normal data, then it is very hard to create a boundary not including anomalies as well [19, 65].



(a) OCSVM classifier decision boundary

(b) OCSVM model with decision boundary, datapoints out of this boundary are marked as anomalies during testing

Figure 2.12: OCSVM during training and testing, both figures are reproduced from [19]

OCSVMs do not require large data amounts and are fast to train, also, they have a fast inference time. However, they have limited capacity to capture complex relationships within the data and they require a carefully tuned parameter selection, moreover, they do not model a probability distribution, which makes it harder to compute the estimates of confidence [19].

2.5.3 Deep Learning

DL is an evolution of ML, and a subset of it that differs in the way it solves problems. They also differ in how they process and learn from the data. Moreover, basic ML models still need some human intervention. If an algorithm gives an inaccurate prediction, an engineer needs to make adjustments. In DL, on the other hand, the need for a domain expert is eliminated because DL understands the features incrementally [55, 16, 18, 4].
[55] again gives a good definition. It defines DL as follows: *"[DL is] a subfield of machine learning that structures algorithms in layers to create an "artificial neural network" that can autonomously learn and make intelligent decisions."*
With DL models, algorithms can determine whether a prediction is precise enough on its own. This is done via their neural network, requiring minimal to no human help. This means, DL models can learn through their own methods of computing. There exist more key differences between ML and DL. ML already consists of thousands of data points and performs usually well with relatively small datasets, but DL uses even more data points, in the range of millions, and requires accordingly huge amounts of data to understand but also to perform better than the traditional ML algorithms. In contrast to ML algorithms, DL ones solve problems based on neural network layers. Also, ML algorithms need relatively little training time, while DL algorithms need much longer, which can range from a few hours to many weeks. However, in testing, this is the other way around. Here, DL algorithms take much less time. But DL needs costly high-end machines and high-performing Graphics Processing Units GPU [55, 4]. Nonetheless, data scientists sometimes choose ML over DL because of the superior interpretability, or the possibility of understanding the solutions easier. They are also preferred if the amount of data is small [4].
On the contrary, DL is preferable if there are large data amounts, a lack of domain understanding, or complex problems [4].

DL programs have several layers of interconnected nodes. Each layer builds upon the last for refining and optimizing predictions. Nonlinear transformations are performed to the input and create a statistical model based on what it learned as output. This is done iteratively until the output has gained an acceptable accuracy level. The word *"deep"* refers to the amount of layers of the neural network through which data must pass. A neural network with input and output layers and more than one layer in between can be considered as a DL algorithm [4, 16]. The following figure, Figure 2.13, represents a deep neural network with an input, an output layer, and three layers in between. Moreover, it pictures a feed-forward network, since the arrows only flow in one direction, namely from the input to the output layers. However, there also exist other training variants as back-propagation, from output to input and more [16].

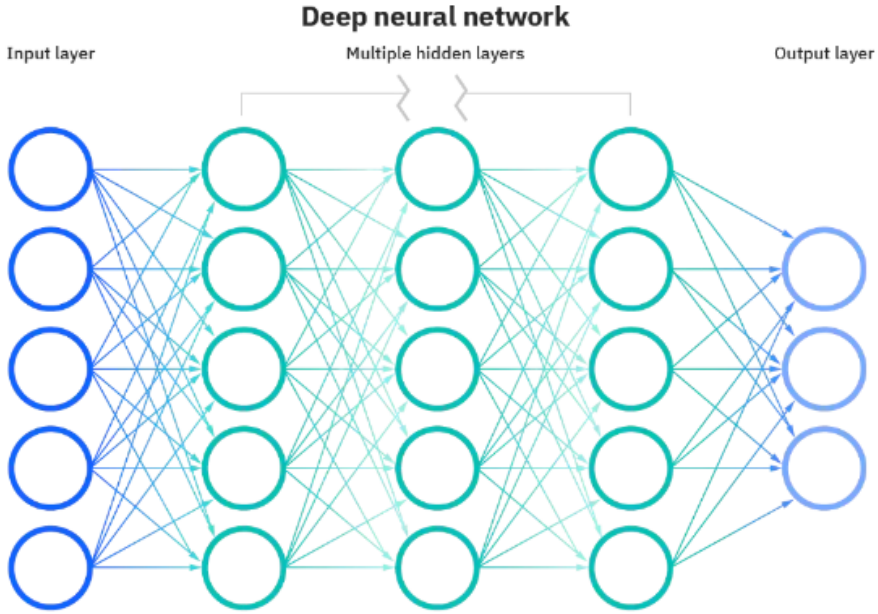


Figure 2.13: Deep neural feed-forward network with input and output layer and three layers in between, reproduced from [16]

As mentioned above in the quote, DL works with artificial neural networks. This is a layered algorithm structure. To start, unlabeled data is given as an input to a DL model. This input is then sent through the neural network's hidden layers. Mathematical operations are used to recognize patterns and develop an output [55].

DL approaches, if applied to anomaly detection, offer many advantages [19]. DL for anomaly detection, in short, Deep Anomaly Detection (DAD), aims to learn anomaly scores or feature representations via neural networks. Anomaly scores describe the level of outlierness for a given data point. If this score surpasses a given threshold, commonly known as the decision score, it is marked as an anomaly. Decision scores are more informative than binary labels [25, 26, 57]. Popular DL techniques do not apply to anomaly detection due to unique characteristics of anomalies, like heterogeneity, rarity, or prohibitively high cost to collect large-scale anomaly data [26] and more mentioned in

section 2.1. It is not always beneficial to use DL methods, since they are not always the best approach. According to Federchuk [48], DL is mostly used for complex tasks with unstructured data. Unstructured or qualitative data is easy and inexpensive to store. Moreover, it can be used in different formats because it does not have a specified purpose. Even though ML models can perform certain more hard tasks like face recognition, DL models are more accurate. However, structured or quantitative data has a predefined format that limits its use cases and its flexibility [48]. DL also faces other challenges, as noted in [4]:

- **Hardware requirements:** DL needs better hardware like multicore high-performing GPUs, improved Random-Access Memory (RAM), Hard Disk Drive (HDD) and more. This is needed to ensure improved efficiency and also to allow for decreased time consumption.
- **Learning rate:** If this rate is too high, the model converges too fast, leading to a suboptimal solution and if this rate is too low, the process can get stuck.
- **Biases:** If the data that is given to a model contains biases, the model will reproduce this for its predictions.
- **Learning through observations:** This means that DL models only know observations that occurred in the data they were trained on. Depending on the dataset, this can lead to unrepresentative or not generalizable models.
- **Data amount:** DL requires large amounts of data because of the need for more parameters for more accurate and powerful models. This is not always easy to get.
- **Lack of multitasking:** If a DL model is trained, it becomes very inflexible and, therefore, can not handle multitasking. They only can give effective and accurate solutions for a specific problem. For similar problems, the model would need re-training.
- **Lack of reasoning:** Reasoning is not possible with DL methods, even with large data amounts.

According to these challenges, it may sometimes be the better choice not to use DL models if one of these challenges could lead to problems.

Chapter 3

Related Work

For this thesis, anomaly detection is an important topic, therefore related work about traditional and Machine Learning anomaly detection approaches identifying outliers is especially useful and therefore discussed in this chapter. Firstly, anomaly detection in general is discussed. Afterwards, the traditional approaches, especially the rule-based ones are mentioned, and after this, the ML approaches used in the related work, used in this thesis, are described. Moreover, WEL related sources are analyzed and discussed to give an overview of how to use them. Finally, limitations mentioned in or arising through related work are pointed out and discussed.

Table 3.1 on the following page shows the most important related works and gives an overview of their topics.

3.1 Anomaly detection

Anomaly detection has been a topic of research for many decades. The research does not focus on specific aspects of it, on the contrary, it is a very diverse field. Not only the fields it can be applied to is very diverse but also the techniques that are used vary. Therefore, there exist several specific approaches to quite narrow use cases as in [1] or mentioned by [69] but sometimes slight changes made to proposed models make it possible to use them for different use cases or tools as mentioned in [35, 43, 47].
Innovations, development, and research will focus on the accuracy of anomaly detection and precise identification but also on reducing FP [65].

3.1.1 Traditional Approaches

The traditional approaches are techniques that do not use modern ML or DL methods. Instead, they use statistics or rule-based approaches. Because of the need to define in advance which anomalies should be detected through rule-based approaches, this makes it impossible to defend against new attacks [31].

Author & Reference	Year	AD[a]	Splunk	Logs[b]	TA[c]	ML	DL	Attacks
Al-amri et al. [60]	2021							
Kitsuchart and Wisuwat [54]	2016							
Chandola et al. [69]	2009							
Su et al. [31]	2016							
Steverson et al. [35]	2021			WEL				
Chalapathy and Chawla [57]	2019							
Pang et al. [25]	2021							
Du et al. [49]	2017							
Thi Quynh et al. [51]	2020		Dataset used[d]					
Fujimoto et al. [43]	2018			WEL				
Cersosimo and Lara [47]	2022							
Dwyer and Truta [34]	2013			WEL				
Hristov et al. [42]	2021							
Wang et al. [71]	2022							
Foorthuis [58]	2021							
Chen and Gurganus [10]	2017							
This thesis	2024							

Table 3.1: Authors of the main papers used for the thesis

[a]AD stands for Anomaly Detection
[b]if not explicitly marked with WEL, then logs in general are used for the study
[c]Traditional Approach
[d]They used a dataset of Splunk

However, if it is defined what to look for, rule-based approaches will detect all anomalies defined and a warning system can notify administrators during an attack to take quick actions. This is a tedious process since they need to be updated constantly. Nevertheless, [31] could identify Distributed Denial of Service (DDoS) attacks with the use of Splunk, a commercial log analysis tool that includes automated log parsers and allows to analyze and visualize the data [20, 38, 31]. They analyzed traffic characteristics during the attacks on firewalls and through visualizations, they could quickly define whether an attack occurred or not. Moreover, they could get a deeper understanding of different attack types and detect security problems before an attack took place by the use of a warning system. Another study, [10], also used Splunk and its alerting system for their research and used a statistical approach as a foundation for anomaly detection. They were able to detect ransomware attacks through behaviour-based anomaly detection methods.
The authors of the study [43] compared several previous studies that undertook anomaly detection methods on golden ticket and privilege escalation approaches and proposed a new method that uses the existing models of the earlier studies to detect attacks against the active directory by the use of predefined rules [43].
[66] and [20] used a similar approach. Both studies conducted a literature review. The former did it on six state-of-the-art anomaly detection approaches and compared their accuracy and efficacy. To conclude, they released an open-source toolkit of these methods to allow for easy reuse. [20] did its literature review on many more studies compared to [66]. However, the study of [20] focused more on ML and DL techniques and stated that traditional methods are less effective than such modern approaches.
The study [34] on the other hand, used a statistical approach. They found out that it is best to generate alerts across all included servers and users, and alerts should not be generated for specific servers or users.

3.1.2 Machine Learning Approaches

Three studies focused on the aspects of ML in a general context and not in relation to anomaly detection. The first study, [64] gives a good overview of ML and DL, moreover, it points out, among other things, that for traditional programming, data, and rules are inputted and processed by algorithms and answers are outputted. On the other hand, in ML, data and answers are inputted to algorithms, and rules are generated as output [64].
The second paper, [54], had a look at the difference between DL and what they called shallow learning techniques on a small face shape dataset. Their definition of shallow approaches is as follows: *"Shallow model learning is a type of machine learning algorithms [that] can generate good generalized predictive model[s] with only a few layers of composition. It requires samples with well-studied discriminative features extracted by experts. It can perform well even though only a limited number of samples is available"* [54, p.2]. As shallow approaches, they used an artificial neural network algorithm and a SVM, and compared it to the DL approaches Convolutional Neural Network and compared them by using a one-way Analysis of Variance (ANOVA). They could conclude that on their small dataset, the shallow approaches performed better than the DL approach for most cases [54].
The last of these studies is [47]. They focused on DNS and used Splunk to clean and combine their dataset, visualize, and analyze their data. They showed that prediction can

be improved by creating early visualizations and having flexibility in the feature selection [47].

The following papers all include modern approaches in combination with anomaly detection. Several authors of papers gave an overview of slightly different themes in the anomaly detection field.
The authors of [60] lied their focus on ML and DL techniques that could detect anomalies in IoT data and provide a review of this topic. They give an overview of anomaly types, detection learning models, window models, the nature of the data, developed techniques, and evaluation metrics used for measuring the performance of a given technique.
[69] presents an overview of the general research on the field of anomaly detection, including several ML and DL techniques. The authors of this study and those of [60] mention the difficulty that arises with the large amount of data being processed.
The author of [65] provides in his publication an overview of unsupervised and supervised ML techniques in the context of cloud data warehousing. Similarly, but a bit more general, [13] also did a literature review on ML and DL techniques. This study had a focus on existing failure detection and prevention techniques and analyzed these. The study [20] also conducted a literature review, and they concluded that ML and DL based classification approaches on selected features allow for enhanced performance compared to traditional rule-based or method-based approaches. [57] also presents an overview, quite similar to [69]. However, they do this only for DL based anomaly detection methods and they compare their effectiveness and mention some challenges that occur while working with DL in anomaly detection and point out existing solutions. [25] is a very similar study that reviewed 12 different modeling perspectives on DL techniques for anomaly detection. They also note down challenges and how some methods discussed address these anomaly detection challenges. Other researchers created new models. The paper [35] is one of these. They constructed a ML anomaly detection algorithm that used a self-supervised training. With this, they could show that it is possible to use ML algorithms with WELs to detect cyber-attacks.
DeepAnT was proposed by [50], it is an unsupervised DL based anomaly detection approach for time series data, and can also be applied to non-streaming cases. Another The authors of [49] came up with another DL based anomaly detection approach, they called it DeepLog. It is a deep neural network model that uses Long Short-Term Memory (LSTM) to model logs of a system as a natural language sequence.
MADDC (Multi-scale Anomaly Detection, Diagnosis, and Correction) was invented throughout the research of [71]. It is a multi-scale approach that can detect and diagnose anomalies in event logs. It combines the LSTM based variational autoencoder and a newly designed local anomaly detector. DabLog, proposed by the authors of [40] is also a LSTM based DL approach. It stands for "Deep Autoencoder-Based anomaly detection method for discrete event Logs" [40]. DabLog determines whether a given sequence is abnormal or not by analyzing and reconstructing the given sequence [40].
[43] was a bit more specific on the topic and compared four different unsupervised ML algorithms. They examined their performance on the same dataset provided by Splunk and finally created a novel approach that combined two out of these four approaches to get a very high detection rate on the tested dataset [43]. Another study that first reviewed existing approaches and then created a new technique is [66]. They analyzed six state-of-the-art log-based approaches for anomaly detection according to their accuracy

and efficiency and then created an open-source toolkit of these reviewed methods for easy reuse.

Thus far, several studies investigated very different areas in the broad field of anomaly detection, giving a good foundation for this topic. However, many of these recent studies are very specific or focus mainly on modern approaches, and up to now, it is not clear whether, for certain aspects and scenarios, a traditional approach may be superior to modern ones.
Nevertheless, according to several studies, modern approaches give the opportunity to detect newly established attacks, which is clearly not possible with a traditional rule-based approach. Although modern approaches enable this, rule-based approaches will always recognize attacks that follow the pattern they were programmed to look for, whereas this is maybe not always the case for ML or DL methods. Moreover, DL approaches are very specific to a certain situation, and therefore, they are not applicable to a different setting. If a certain DL approach should be used in an alternative setting, it would need to be retrained for the different use cases. For example, they would need retraining for every log source structure. This stays in contrast to generalized anomaly detection.

Moreover, the wide field of anomaly detection is bigger than just the part related to cybersecurity, which this thesis focuses on. Anomaly detection is used in many other fields, like medicine or risk management [25]. Therefore, it would be valuable to have anomaly detection that could be generalizable to several fields. However, to the best of the authors' knowledge, no study had looked at such a general aspect as to use one anomaly detection approach in several fields.

3.2 Windows Event Logs

Several studies have used different anomaly detection approaches with logs as their data source [35, 57, 49, 66, 34, 42, 20, 71, 40, 13, 10, 29, 51]. Logs contain important information like computer status, which makes them a valuable resource for detecting anomalies [29], but they also give a lot of information about different fields and are logged in huge numbers since many events are recorded [35, 49, 13]. This makes them an excellent data source for ML and DL algorithms that need a lot of data for proper training but do not need labeled data points [13].
A literature review done by [13] found that primarily it is needed to identify the fault leading to a failure and finding the cause of it. Moreover, it is important to know the system state. This information can be extracted from system logs. This study uncovered that many researchers have looked at systems from different areas, however, existing models are system specific [13, 43]. A problem with current models is that, although they are able to detect failures, they do not provide additional information, such as cause or location. Lastly, it was discovered that a research gap includes the lack of early warning for failures, failure rates are still not good enough, and that there exists no fully automated system [13].
A study that used WELs is [34]. They adapted intrusion detection techniques, used in Intrusion Detection System (IDS), for WEL datasets. They tried to automate the anomaly finding process within the log datasets by the use of standard deviation. In addition, they

uncovered that it is better to create alerts across all servers and users instead of generating them for specific servers and users. However, they used standard deviation and only generated alerts that occurred more often. They do not mention whether an alert would be generated for new events. If an attack would generate a single new event, it may not be detected through this approach [34].
There also exists previous work that employed logs to Splunk like [1]. They mention that the combination of threat model and threat intelligence to a single system by using Splunk helps to provide end-to-end security, in this case, to cloud systems. They also noted down the attacks that their threat model could detect. The categorization was done according to the MITRE ATT&CK framework [12], standing for "adversarial tactics, techniques, and common knowledge" [1]. Like this, it is possible to detect live threats and potential threats. Moreover, they mention that it would be possible to use this combined system as a SIEM tool [1].
[35] uses DL and Natural Language Processing (NLP) on WELs to detect attacks on enterprise networks. Their model was able to detect which devices were compromised, as well as the attack timing with very good precision and recall rates.
Previous research made a systematic literature review, which concluded that ML and DL based classification approaches enabled enhanced performance compared to traditional rule- and method-based ones [20].
The paper [66] compares six different state-of-the-art anomaly detection methods based on logs. They had a closer look at three supervised and three unsupervised methods. This study provides a review and evaluation of these methods by comparing their accuracy and efficiency on two production log datasets and gives an open source toolkit that should ease the reuse of the given anomaly detection methods [66].
Other papers also invented log-based anomaly detection methods as [49], [40], and [29]. According to [49] and [29], many tools have been designed for log mining but in recent years, DL based methods have become a more popular study field.
However, even though all these advances, [56] and [29] point out some issues when logs are used as a data source. [56] proposes a centralized architecture if WELs are used as a source. This centralized storage makes it more difficult for an adversary to destroy or alter log files. As mentioned in section 2.4, Splunk's indexing allows for a centralized data analysis, therefore eliminating this drawback of WELs. [29] mentions two more challenges, namely the semantic meaning and that logs are unstable in the real world. The former point informs about the semantic information that should be taken into consideration when working with logs, whereas unstable logs in the real world point out that a lot of unseen logs exist in the information systems. This happens because developers may change, delete, or insert words to update logging statements. The authors of [33] noted that it is difficult to detect missing words which can happen if logs are updated. Changes, deletions, or insertions then generate these unseen log events. The research of [29] then proposes a new semantic information embedding technique to overcome the first problem, and a combined anomaly detection approach of two methods to overcome the second problem. This makes it possible for the approach to learn semantics and the quantitative features from system logs. However, they mention that a drawback of their DL approach is that it takes longer to detect anomalies [29].
According to Ananthapadmanabhan and Krishnashree [1] and their categorization of threats introduced in their threat model, it is possible to detect the attack behaviour of the following categories and subcategories (The subcategories are techniques according

to MITRE ATT&CK [12], which may consist of one or more sub-techniques):

Initial Access: Drive-by Compromise, Exploit Public-Facing Application, Phishing, Trusted Relationship, and Valid Accounts

Execution: User Execution

Persistence: Account Manipulation, Create Account, Implant Internal Image, and Valid Accounts

Privilege Escalation: Valid Accounts

Defense Evasion: Impair Defense, Modify Cloud Compute Infrastructure, Unused/ Unsupported Cloud Regions, Use Alternate Authentication Material, and Valid Accounts

Credential Access: Brute Force, Forge Web Credentials, Steal Application Access Token, Steal Web Session Cookie, and Unsecured Credentials

Discovery: Account Discovery, Cloud Infrastructure Discovery, Cloud Service Dashboard, Cloud Service Discovery, Network Service Scanning, Permission Groups Discovery, Software Discovery, System Information Discovery, System Location Discovery, and System Network Connections Discovery

Lateral Movement: Internal Spearphishing and Alternate Authentication Material

Collection: Data Staged, Data from Cloud Storage Object, and Data from Information Repositories

Exfiltration: Transfer Data to Cloud Account

Impact: Data Destruction, Data Encrypted for Impact, Defacement, Endpoint Denial of Service, Network Denial of Service, and Resource Hijacking

Moreover, several studies examined which attacks are detectable by analyzing log files. [10] was able to show that a ransomware attack is detectable through logs ingested into Splunk. Another study also implies that it is possible to detect attacks with logs through statistics, like Brute Force attacks [34] and the authors of [51] found that Domain Name System (DNS)-attacks can be detected, and they could identify infected hosts. Lastly, [49] mentions that, in fact, all attacks that leave a trace in the system logs caused by the logging activities of the system could be detected.

Especially the statement made by [49] seems to be crucial for anomaly detection with WELs. If every attack leaves a trace within the logs of a system through its logging activities, we can find evidence if something looks anomalous. However, this becomes challenging if we suppose that an adversary is already in the system and can now alter the logging behaviour or delete log files. This makes it even more important to recognize attacks early on such that actions like stopping the logging of a system or altering the logs do not happen at first. This is a very challenging step and at the current state of the

research, it is not possible to detect all attacks, therefore, it may not be possible to fully prevent the changing or deleting of log files. Nevertheless, if it is possible to prevent some of these actions from happening, this can already be seen as an improvement.
Besides, ML and DL approaches seem to be promising in preventing attacks from happening through Intrusion Prevention Systems (IPSs). With advancements in this direction, it may become possible to predict attacks before they are finished, and therefore, this makes it possible to protect in a more advanced way. Instead of only reacting to attacks that happened, it becomes possible to act while they take place.

3.3 Limitations

The related work talks about different traditional, rule-based, and statistical approaches, but also ML and DL ones. However, to the best of the author's knowledge, no study looked at whether rule-based approaches perform better in certain scenarios compared to ML or DL techniques.
The work [69] talks about traditional and modern approaches, their advantages and disadvantages, but does not compare the approaches directly with each other. In this context, [66] mentions that it becomes infeasible because the log size increases drastically to use traditional approaches since a high amount of manual log inspection is needed.
A paper closely related to this thesis is [20]. It discusses traditional and modern approaches and mentions that they could ascertain *"that machine learning and deep learning-based classification approaches employed on selected features enable enhanced performance than traditional rule-based and method-based approaches"* [20]. They mention that modern approaches have higher accuracy, too. Nevertheless, this study does not explain why this is the case and whether this is valid in all contexts.

In the following, the limitations explained by the researchers will be discussed.
Several problems were noticed by different studies like [25], which notes that approaches for anomaly detection are either generic and not applicable to anomaly detection since they were designed for other scenarios and therefore, heterogeneous anomalies can not be detected by such traditional approaches. This is related to the problem pointed out by [20] and [13] that point out the need of generalized systems because current methods in use are mostly system specific. Therefore, [58] proposes to establish a more wide field, which is called anomaly analysis and detection. The study also underlines the importance of understanding and explaining why a given anomaly is not normal. Moreover, [20] mentions that a problem with different log types can arise since different types of logs have quality issues and are therefore not available in a certain standard format [20].
[20] mentions even more limitations when it comes to anomaly detection: The logs are considered with the assumption that they are complete and accurate. This assumption is made in several studies [13]. Inaccuracy or incompleteness can lead to the situation where logs have missing information when for deciding whether it belongs to an anomaly or not. Another limitation in anomaly detection occurs while preprocessing the data. Preprocessing can lead to missing data because important data was filtered out before the logs were analyzed [13, 20].
Two studies, [13] and [20], found that current methods detect or identify failure, however,

they do not provide information about the cause of failure, the location or path of it, and neither the components involved.
Another limitation of recent models is that they can not detect anomalies that co-occur [20]. This may be solved if multiple logs could be analyzed together, however, this is limited by memory and computing costs, especially during the training phase [35].
[60] points out a challenge that was mentioned by other studies as well, namely the fact that anomalies are heterogeneous. This was also mentioned by [25] and [57] for example. Through the diversity of anomalies, anomaly detection can become a difficult task because it is not possible to cover all anomalies in a training or testing dataset and therefore, the focus mainly lies on the recognition of significant anomalies [66, 13, 20].
Another assumption made by several studies is stated by [69]. It is mostly assumed that in anomaly detection, the amount of anomalies is very low [50, 69]. Consequently, [69] brought up the point that worms can not be detected since the amount of abnormal traffic in the system is bigger than the normal one.
Lastly, [69] mentions something interesting; the classification and clustering based approaches and statistical techniques they looked at in their study had expensive training times but the testing was usually fast. This is often acceptable because the training can be done in an offline fashion while the testing should be done in real-time.

These limitations show the potential that lies within the research field. Especially the fact that no study explicitly looked at the advantages and disadvantages of modern approaches and compared them with traditional methods seems to be peculiar. It may be possible to use a traditional method as a basis and then enhance it with modern approaches. Since both approaches can be used with WELs, this study will provide a first insight into this comparison by the use of WELs as a data source and use it with a rule-based approach and a ML approach.

3.4 Conclusion

Generally speaking, the field of anomaly detection is very wide and varied. This makes the detection of different anomalies through a single system a challenging task. If we only take a look at anomaly detection in the field of cybersecurity, it is still a very diverse research field. This can be seen within the diversity of attacks according to MITRE ATT&CK [12] but also the fact that attacks and therefore anomalies always change, underlines this problem. There has already been a lot of work done on this topic and in recent years, modern approaches gained more attention because they are superior to traditional methods in some aspects. For example, they need less human intervention and allow in certain scenarios even to predict future events. Although predictive approaches are very promising, we currently can not use them in daily life. Nevertheless, this is an important topic and can help in the future if the approaches are matured.
Log files can help with anomaly detection if they are not corrupted or deleted either during preprocessing or in the case of an attack through the adversary. This assumption is essential for all studies working with event logs because if they are deleted, anomaly detection is no longer possible and if they are changed, this may corrupt the outcome.

Previous research established many alternative approaches that are superior in some way to the ones currently in use. Therefore, many studies looked at modern approaches, but it seems that the related work did not discuss explicitly whether modern approaches are better compared to traditional ones. This raises the question of whether they are superior in all areas of anomaly detection. This needs to be researched because traditional rule-based approaches always will detect defined attacks except they change in behaviour and may be even superior to modern approaches in certain scenarios.
The combination of both traditional and modern approaches raises the need to understand the differences between modern and traditional anomaly detection approaches to determine their advantages. Knowing the advantages, maybe a combined approach could be superior to only a traditional or a modern one.

Chapter 4

Methodology and Design

Through this study, a comparison between Traditional and ML models for different attack patterns is conducted. This is done through Splunk with the help of the SPL for the traditional approach, and through Python scripts for the ML approach. Supervised ML was used since labeled data was used to train and test those algorithms. This comparison allows to answer all the RQs and to give an insight into the performance of the attacks studied.
This chapter first explains the work through to get to the practical part of this thesis, by mentioning which attack tactics were considered, and which techniques and sub-techniques finally were examined. Moreover, this chapter highlights the steps taken to get the data into Splunk to analyse it, as well as how this was done. In Section 4.2, the evaluation values are presented and the formulas are listed. Finally, it is concluded with the Feature Selection where it is described why the features were taken and what they are.

For the distinction of attacks,[12] is used, [1] summarizes [12] as follows: The adversary tactics in the framework are mainly categorized into the following 14 types:

1) Reconnaissance: initial phase where related information about the victim is collected.

2) Resource Development: in this phase, the required resources that will support the operation are listed.

3) Initial Access: initial attempt to get access to the network through different tactics like Phishing.

4) Execution: malicious code or data is executed in the acquired network.

5) Persistence: in this phase, the adversary tries to keep hold of access to the network. This is done by changing the configuration.

6) Privilege Escalation: the attacker attempts to acquire a higher-level permission.

7) Defense Evasion: now necessary actions are taken to prevent being detected.

8) Credential Access: in this phase, it is tried to steal the credentials of the victim, like usernames and passwords.

9) Discovery: exploring what the adversary can take control of on the acquired platform.

10) Lateral Movement: moving through the platform by using the obtained credentials.

11) Collection: accessing stored data on the platform.

12) Command and Control: the attacker attempts to communicate with the victim's acquired platform to control them.

13) Ex-filtration: transferring data to an external repository.

14) Impact: attacker tries to destroy and/or manipulate the platform.

4.1 Data Pre-Processing and Workflow

Before starting with the practical part of this study, four attack types were chosen to be considered for the beginning. These three attacks belong to three different tactics of [12]. All tactics investigated during this thesis are listed in Table 4.1.

MITRE ATT&CK Category	Technique	Dataset
Privilege Escalation	Abuse Elevation Control Mechanism	[62]
Defense Evasion	Abuse Elevation Control Mechanism, Impair Defenses	[62] [44], [62]
Credential Access	Brute Force	Self generated, [44], [62]
Lateral Movement	Remote Services	[44], [62]

Table 4.1: Considered attack tactics with inspected techniques

Since almost all the tactics were an option, it was checked which sub-tactics had enough data logs already created or which attacks were not that difficult to execute. It was decided to start with the Brute Force technique. This technique is further divided into Password Guessing, Password Cracking, Password Spraying, and Credential Stuffing, which are explained in section 2.3.1. Of these four, Password Guessing and Password Spraying were taken into consideration, since they are somehow similar. Moreover, the sub-technique Bypass User Account Control was taken into account, since [62] and [44] already provided a lot of such log data. The sub-technique Disable Windows Event Logging was also considered because it is crucial to detect the case if the logging of WELs is disabled. Otherwise, without the WELs, the technique used in this thesis would not be useable. However, this was then not chosen to be part of the research, since it would be very difficult to train a ML model on such single and rare events. Therefore, another attack was looked at, where [62] and [44] already provided several sample data to use. Moreover,

this attack seemed more promising since it is more likely to be possible to train a ML model on this data.

To start the practical part of this study, data was collected first. The data logs were taken from the two GitHub repositories [62] and [44]. This data was already anonymous. In addition, some data logs of Brute Force password guessing and password spraying attacks were generated by the author through a batch file and two text files, one containing passwords and the other usernames. Some of the most common usernames and passwords were taken according to [15].

Figure 4.1 shows the process of getting the data until the analysis of the data, which is further discussed in Section 6.1. To preprocess the data to upload and use it in Splunk, the data was opened in the Event Viewer and saved as Extensible Markup Language (XML) data. This XML data was then run through a python script (A.1), that was created for automation to run the preprocessing. This python script reads the XML data of an event log with several events and puts all the single events on one line, with line breaks only in between the events (An example of the processing of XML data is shown on the List of Tables. A.42 shows a file before the preprocessing and A.43 shows the file after it was changed through the python script). The single events of each source were then combined into one file. Like this, it was then possible to upload the single files, that included several events, to Splunk.
In parallel, to gather normal data logs, all events of two machines of the author were saved.

Before starting with the attacks, the normal data was uploaded to Splunk, returning a total of 127'322 events. For the normal data and each attack technique, a separate index was created in Splunk for better distinction of the log files. Moreover, fields were extracted from the data in Splunk, allowing to search for EventIDs and other major information sources.

For the traditional approach, the Splunk query was created by using the SPL. Through different SPL queries, valuable insights into the attack patterns could be gathered. Later on, to visualize the results, graphs were created in the dashboard of Splunk. Lastly, the values for Precision, Recall, Accuracy, and the F1-Score were calculated with the help of the SPL.

The approach of the ML technique was a bit different. Firstly, the Splunk MLTK was tried for training the ML model. For the data, the query in Listing 4.1 was used, which should return all Brute Force and normal events. However, since the MLTK needed to install the Python for Scientific Computing add-on, which did not work as expected, a different approach was chosen. Therefore, own code was written in Python to train and test a ML model, then the model was run and values for Precision, Recall, Accuracy, and the F1-Score were returned.

```
1    index="normal_events" OR index="brute_force" sourcetype!="All Brute
   Force Attacks"
2    | rex field=_raw "(?i)<EventID( Qualifiers=\"\d+\")?>\s*(?<EventID>\
   d+)\s*</EventID>"
3    | rex field=_raw "<TimeCreated SystemTime=\"(?<SystemTime>[^\"]+)\"
   />"
```

Listing 4.1: Query returning Brute Force and normal events

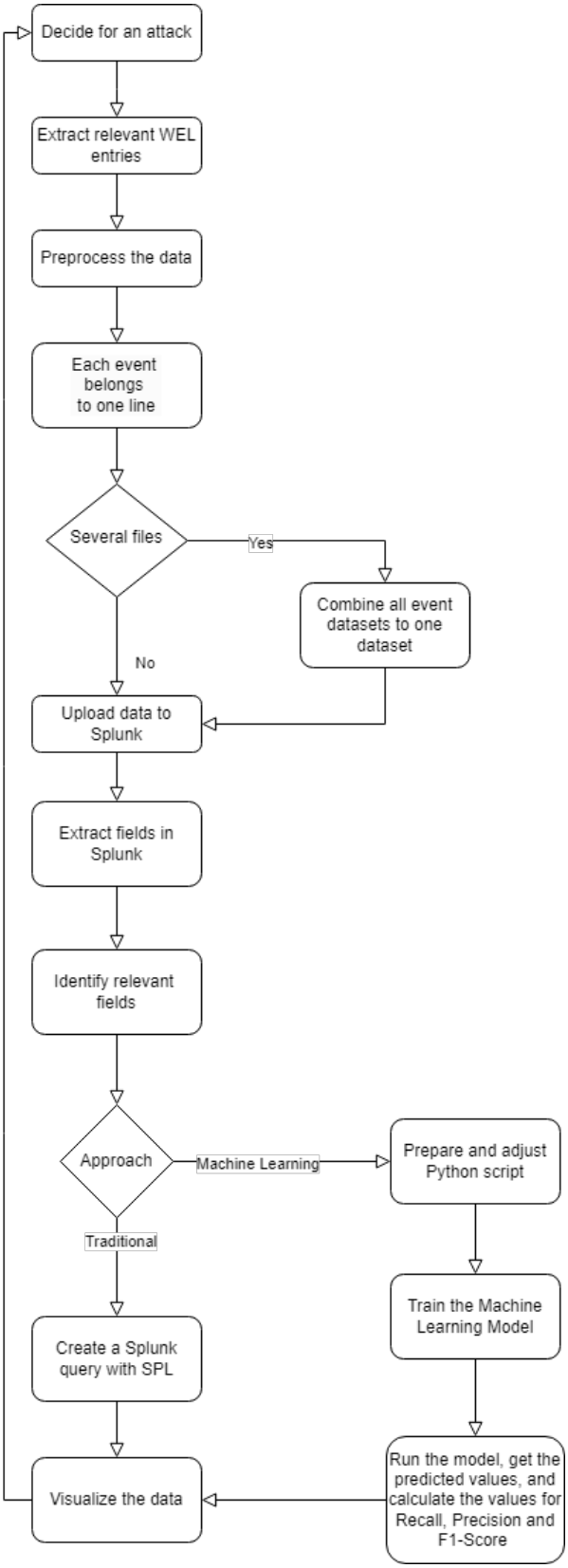


Figure 4.1: Process flow of data preparation and analyzation.

4.2 Model performance evaluation

To measure the performance of the different models, the Precision- and the Recall-rate were taken into consideration, as well as the F1-score and the accuracy. [60] shows that Accuracy, Precision, and Recall are often used in papers for evaluation. Sometimes also the True Positive (TP), True Negative (TN), FP, and FN and the F1-Score are used. This is emphasized through many other papers as well like [33], [71], [25], [50], and [35], just to mention a few using these evaluation criteria as well.
Precision is the number of TP divided by the TP plus the FP as defined by [19] and [20]:

$$\frac{TP}{TP + FP} \tag{4.1}$$

Recall is defined by the number of TP divided by the amount of TP plus the FN [19]:

$$\frac{TP}{TP + FN} \tag{4.2}$$

Accuracy evaluates the accuracy of the predicted samples and is defined as follows [20]:

$$\frac{TP + TN}{TP + FP + TN + FN} \tag{4.3}$$

The F1-score is used to measure the accuracy of a model and is defined as the combination of the precision and the recall values. A higher F1-score shows a better anomaly detection of the model. The formula looks as follows [20, 71]:

$$\frac{2 * (Precision * Recall)}{Recall + Precision} \tag{4.4}$$

In detail, this would result in the following equation:

$$\frac{2 * (\frac{TP}{TP+FP} * \frac{TP}{TP+FN})}{\frac{TP}{(TP+FN} + \frac{TP}{TP+FP}} \tag{4.5}$$

4.3 Feature Selection

In the following paragraphs, the features selected for the ML approach are explained. The features used to define rules for the traditional approach are explained in the Implementation chapter.

4.3.1 Brute Force

For this attack, different feature combinations were used. Finally, the features "EventID", "CountUsernamesLast1s", "CountLast1s", "CountSameUsernameLast1s", and "CountDifferentUsernamesLast1s" were used.
The "EventID" was used because the ID includes details about what happened in the event. This includes valuable information about wrong or correct login attempts, for example, which was especially useful for this kind of attack.
The "CountUsernamesLast1s" includes the amount of events, including a username, which is not necessarily unique. "CountSameUsernamesLast1s" does something similar, it counts the amount of a username given in a certain event and adds up the events including the same username. This field excludes the given event and counts until, but excluding, one second later. "CountDifferentUsernamesLast1s" does the same as the previous feature, except it counts the number of events including a different, unique username. These fields were helpful for the Brute Force attack, since in this type of attack, many usernames are tried together with different passwords.
Lastly, the "CountLast1s" field is included. This field returned the amount of events happening within one second, excluding exactly one second, after a given event happened. This is helpful since during a Brute Force attack, many events happen within a short time frame.

4.3.2 Bypass User Account Control

The Bypass User Account Control used six features. As already done in the previous attack, the "EventID" was used as a field for the reasons mentioned above.
Next, the "Image" field was taken since this only showed up in attack events, including code to execute programs like the Command Prompt.
For this attack, the command lines were important, therefore, the features "CommandLineLength", "ParentCommandLineLength", "CommandLineEntropy", and "ParentCommandLineEntropy" were used. "CommandLineLength" and "ParentCommandLineLength" define the length of the command lines, whereas the "CommandLineEntropy" and the "ParentCommandLineEntropy" include the entropy value of the command lines. The Bypass User Account Control attack uses code to bypass the UAC, therefore these command line features were used and returned promising results.

4.3.3 Remote Services

As for the previous attacks, this attack used the "EventID" as a feature, too. In addition, the "Offsec" keyword was used since this only showed up in attack events. The third feature used was the "Channel", which is the channel the event was logged on. Lastly, the ML approach used the "ProcessName" feature. "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" is an example value that was used in this attack scenario and includes the path of the executable filename as well as the name and the ".exe".

4.3.4 Disable Windows Event Logging

This attack used four features. Again, the feature “EventID” was used as for the other attacks.
Besides this feature, the “Channel” was also used since the normal events only had three values and the attack events sometimes included different values for this field. This field tells on which channel a given event was logged. The normal channels were “Application”, “Security”, and “System”.
For the Disable Windows Event Logging attack, also the “CommandLineLength” and the “CommandLineEntropy” were used as described in the Bypass User Account Control attack.

Chapter 5

Implementation

Before starting with the implementation of the traditional approach and the Python scripts, the data needed to be preprocessed. This is explained in Section 5.1. Section 5 is divided into two main parts, Section 5.2.1 discussing the traditional part of all the attacks taken into consideration, and Section 5.2.2, which is about the ML approaches of the respective attacks. The attacks taken into consideration are the Brute Force, the Bypass User Account Control, the Remote Services, and the Disable Windows Event Logging attacks.

5.1 Preprocessing

Starting with the preprocessing of the data, first, the data was saved as XML files which were then further processed with the file "XML Parser.py". This file consists of a function that saves events to a file, the "parse_xml", and the "process_xml_files" functions are displayed in the Listings 5.1 and 5.2 below.

```
def parse_xml(xml_file, output_file):
    # Parse XML file
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Get all events' elements
    events = root.findall('.//{http://schemas.microsoft.com/win/2004/08/
    events/event}Event')
    filename = os.path.basename(xml_file)
    output_file = os.path.join(output_folder, filename)

    # Save all events to a single file
    save_events_to_file(events, output_file)
```

Listing 5.1: parse_xml function

```
def process_xml_files(input_folder, output_folder):
    # Iterate over each XML file in the input folder
    for filename in os.listdir(input_folder):
```

```
        if filename.endswith(".xml"):
            xml_file = os.path.join(input_folder, filename)
            output_file = os.path.join(output_folder, filename)
            parse_xml(xml_file, output_file)
```

Listing 5.2: process_xml_files function

Next, since there were many XML files, to shorten the upload for Splunk, the file "Concatenator.py" was created. It consists of only one function that combines all XML files into one. This function is shown in Listing 5.3.

```
def concatenate_files(input_folder, output_file):
    with open(output_file, 'w') as outfile:
        for filename in os.listdir(input_folder):
            file_path = os.path.join(input_folder, filename)
            if os.path.isfile(file_path):
                with open(file_path, 'r') as infile:
                    outfile.write(infile.read() + "\n\n")
```

Listing 5.3: concatenate_files function

The next two sections differentiate between the traditional and the ML approaches.

5.2 Implementation

5.2.1 Traditional Approach

This section will show the implementation of the Splunk queries that were used to get the results for the traditional approaches. First, Splunk indices had to be created, which store the data in a repository-like structure. The indices "normal_events", "brute_force", "buac", "remote_services", and "dwel" were created for the normal events, the Brute Force events, the Bypass User Account Control events, the Remote Services, and the Disable Windows Event Logging.

Brute Force

First, the query for the Brute Force attack is explained: If only the data needed is uploaded once, there is no need to specify the sourcetype as shown in Listing 5.4 on the second line. This was only done since in the "brute_force" index, there was a sourcetype called "All Brute Force Attacks", used for analyzing the data, which included all the attack events and there were more sourcetypes containing only part of the attack events. Therefore, to avoid redundant events, this sourcetype was excluded.
The query visible in Listing 5.4 shows the filtering for potentially malicious event IDs in line three, lines four to eleven show the creation of the ten-second bins, and the sorting, which is done in line eleven. The last line, line twelve, shows the plotting of the cart with the sum of the count of the events into the bins. Line ten lets the plot only show those bins that have one or more events, therefore excluding time bins where no events

took place. Moreover, it distinguishes the indices by color. However, there are no normal events occurring that have one of the potentially malicious events, as can be seen in Figure B.5, otherwise there would be another bin displayed with a different colour.

```
    index="brute_force" OR index="normal_events"
    sourcetype!="All Brute Force Attacks" AND
    sourcetype!="BF-3-days-dataset"
    (EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772 OR
   EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040 OR
   EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625 OR
   EventID=24003 OR EventID=24299 OR EventID=529 OR EventID=530 OR
   EventID=531 OR EventID=532 OR EventID=533 OR EventID=534 OR EventID
   =535 OR EventID=536 OR EventID=537 OR EventID=539 OR EventID=681 OR
   EventID=18456)
    | eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
    | eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
    | eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") /
   10) * 10
    | eval bin_time = strftime(bin_time, "%Y-%m-%d %H:%M:%S")
    | eventstats count as event_count by bin_time
    | stats count as event_count by bin_time index
    | where event_count >1
    | sort bin_time
    | chart sum(event_count) over bin_time by index
```

Listing 5.4: Full SPL query searching for Brute Force events

To have a comparable result with the ML methods, for both approaches the same dataset was taken for evaluating them. In the case of the Brute Force attack, this was the three-day dataset. To start, the query presented in Listing 5.5 allowed to check for the three days, day by day, whether there were malicious events in the normal data, which was not the case as shown in Figures 6.6, 6.7, and 6.8, through the fact that all these bins show the time when the attacks took place. These plots have one-minute bins. These figures were created using the following code:

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-04T00:00:00.000000Z", "%Y-%m-%dT
   %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-05T00:00:00.000000
   Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time", date as "Date"
```

Listing 5.5: SPL query that creates plots for a single day, here 04.06.2024

To get the plots of the other days, only line ten needs to be adjusted to the correct time to get the other days. The plot in Figure B.7 is the combination of the three figures mentioned above. This figure was created through the following code:

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-04T00:00:00.000000Z", "%Y-%m-%dT
   %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-07T00:00:00.000000
   Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time", date as "Date"
```

Listing 5.6: SPL query that creates the plot combining the three days shown in Figures 6.6, 6.7, and 6.8

Now it was possible to start with the evaluation. The sourcetype "All Brute Force Attacks", was deleted because it was not needed anymore. Two different evaluations were made: one including all data except the three-day dataset and the other exclusively including this dataset. For both dataset evaluations the TP, TN, FP, and FN values needed to be calculated to be able to compute the Precision, Recall, Accuracy, and F1-Score. Starting with the dataset excluding the three-day dataset, the SPL query is shown in Listing 5.7. The counts of the TPs, TNs, FPs, and FNs are shown in Table 6.1 and the evaluation results returned by the query are shown in Table 6.2.

```
index="brute_force" OR index="normal_events"
sourcetype!="BF-3-days-dataset"

| eval label=case(
    EventID IN (675, 676, 4771, 4772, 4822, 4823, 4824, 5040, 5041,
   5042, 6279, 4625, 24003, 24299, 529, 530, 531, 532, 533, 534, 535,
   536, 537, 539, 681, 18456), "attack",
    true(), "normal"
)

| eval is_true_positive=if(
    label=="attack" AND index="brute_force", 1, 0)
| eval is_false_positive=if(
    label=="attack" AND index=="normal_events", 1, 0)
| eval is_true_negative=if(
    label=="normal" AND index="normal_events", 1, 0)
| eval is_false_negative=if(
    label=="normal" AND index=="brute_force", 1, 0)
```

```
17
18 | stats sum(is_true_positive) as TP
19         sum(is_false_positive) as FP
20         sum(is_false_negative) as FN
21         sum(is_true_negative) as TN
22
23 | eval Precision=if(TP + FP > 0, TP / (TP + FP), "N/A")
24 | eval Recall=if(TP + FN > 0, TP / (TP + FN), "N/A")
25 | eval F1=if(isnum(Precision) AND isnum(Recall) AND (Precision + Recall)
       > 0, 2 * (Precision * Recall) / (Precision + Recall), "N/A")
26 | eval Accuracy=if((TP + FP + FN + TN) > 0, (TP + TN) / (TP + FP + FN +
     TN), "N/A")
27 | table TP, FP, FN, TN, Precision, Recall, Accuracy, F1
```

Listing 5.7: Evaluation of all data except the three-day dataset

The second evaluation included the three-day dataset. There, in a first step, the FNs were calculated:

```
1 index="brute_force" sourcetype="BF-3-days-dataset"
2 (EventID=4625 OR EventID=4648)
3 | table _time EventID EventRecordID SystemTime
4 | eval parsed_SystemTime = strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%QZ")
5 | sort 0 parsed_SystemTime
6 | streamstats current=f window=1 last(EventID) as previous_EventID last(
     parsed_SystemTime) as previous_SystemTime
7 | eval time_diff = parsed_SystemTime - previous_SystemTime
8 | where EventID=4625 AND previous_EventID=4648 AND time_diff <= 0.1
9 | stats count as count_4625_within_0_1_seconds_after_4648
```

Listing 5.8: Calculation of the FNs

After this was determined, the TPs, TNs, and FPs could be calculated. Additionally, the Precision, Recall, Accuracy, and F1-Score were computed. All of this is shown below in Listing 5.9. On lines eleven to 14, the evaluation of the "is_false_positive" is done. This returns zero since if the label is "attack", the part after the "AND" is always true. This was written for clarification since this would define the FPs, the same applies to the "is_true_negative", since the FN count was determined in the previous listing, this evaluation is not needed in this case, but it is written for clarification. Since the "BF-3-days-dataset" sourcetype belongs to the "brute_force" index and includes normal and attack data, it was not possible to distinguish between indices. If this would be possible, it could be done according to Listing 5.7.

```
1 index="brute_force" sourcetype="BF-3-days-dataset"
2 | eval label=case(
3     EventID IN (
4     675, 676, 4771, 4772, 4822, 4823, 4824, 5040, 5041, 5042, 6279,
5     4625, 24003, 24299, 529, 530, 531, 532, 533, 534, 535, 536, 537,
6     539, 681, 18456), "attack",
7     true(), "normal"
8 )
9
10 | eval is_true_positive=if(label=="attack", 1, 0)
11 | eval is_false_positive=if(label=="attack" AND !(EventID IN (
12     675, 676, 4771, 4772, 4822, 4823, 4824, 5040, 5041, 5042, 6279,
```

```
13      4625, 24003, 24299, 529, 530, 531, 532, 533, 534, 535, 536, 537,
14      539, 681, 18456)), 1, 0)
15  | eval is_true_negative=if(label=="normal" AND !(EventID IN (
16      675, 676, 4771, 4772, 4822, 4823, 4824, 5040, 5041, 5042, 6279,
17      4625, 24003, 24299, 529, 530, 531, 532, 533, 534, 535, 536, 537,
18      539, 681, 18456)), 1, 0)
19  | stats sum(is_true_positive) as TP sum(is_false_positive) as FP
20      sum(is_false_negative) as FN sum(is_true_negative) as TN
21  | eval FN = 16608
22  | eval TN = TN-FN
23  | eval FP = 1
24  | eval TP = TP-1
25  | eval Precision=if(TP+FP > 0, TP/(TP+FP), "N/A")
26  | eval Recall=if(TP+FN > 0, TP/(TP+FN), "N/A")
27  | eval Accuracy=if(TP+TN+FP+FN > 0, (TP+TN)/(TP+TN+FP+FN), "N/A")
28  | eval F1=if(isnum(Precision) AND isnum(Recall) AND
29      (Precision+Recall) > 0,2*(Precision*Recall)/(Precision+Recall),
30      "N/A")
31  | table TP, FP, FN, TN, Precision, Recall, Accuracy, F1
```

Listing 5.9: SPL query for calculating the TP, TN, and FP and FN values and calculating the Precision, Recall, Accuracy, and F1-Score

In Listing 5.9 above, on line ten, the number for the FNs was retrieved from the result of Listing 5.8. One was subtracted through the fact that one 4625 event was caused by an unsuccessful login through the user that did not belong to the attack, which was added as the only FP.

The results of the calculations in Listing 5.9 are shown in Tables 6.4 and 6.17.

Bypass User Account Control

In this section, the query for the Abuse Elevation Control Mechanism, more specifically for the subtype Bypass User Account Control, is explained in more detail.
Listing 5.10 shows the SPL that was used to filter out the attack events and shows to which index how many events belong.
Line one of the listing shows the two indices that were used to get the data, namely the "buac" that includes all the Bypass User Account Control events and the "normal_events". Lines two to ten then set "suspicious" to "true" if at least one condition mentioned in the if-statement is true, otherwise, it is set to "false". The last line finally plots the table (Table 5.1) and distinguishes by index and whether "suspicious" is set to "true" or "false".

```
1  index="buac" OR index="normal_events"
2  | eval suspicious=if(
3      match(Description, ".exe") OR isnotnull(CommandLine) OR
4      EventID=4688 OR Product="-" OR Product="?" OR Product="UACMe" OR
5      isnotnull(Image) OR match(_raw, "Masquerading") OR
6      match(_raw, "Masquerade") OR match(_raw, "UAC") OR
7      match(_raw, "uac") OR match(_raw, "Bypass") OR
8      match(_raw, "whoami") OR Description="?" OR
9      match(_raw, "CallTrace"),
10     "true", "false")
```

```
11 | stats count by index, suspicious
```

Listing 5.10: SPL query to show the TPs, TNs, and FPs, producing the table in Figure 5.1

Main Search Results

index	suspicious	count
buac	false	7
buac	true	702
normal_events	false	127321
normal_events	true	1

Figure 5.1: Output of Listing 5.10

Figure 5.1 shows the counts for the TP, TN, FP, and FN values. The TP count is visible in the count column of the second line since they are attack events and were marked as "suspicious". Line three shows the TNs, where the events were not marked as "suspicious" and belong to the normal events. The FPs are visible in the last row, where a normal event is marked as "suspicious". The First row reflects the FN, where the events were not detected by the query but belong to the attack events.
These counts are used in Listing 5.11 to calculate the Precision, Recall, Accuracy, and F1-Score values. The values are assigned in the lines eleven to 14. Lines one to ten stayed the same.

```
1     index="buac" OR index="normal_events"
2 | eval suspicious=if(
3     match(Description, ".exe") OR isnotnull(CommandLine) OR
4     EventID=4688 OR Product="-" OR Product="?" OR Product="UACMe" OR
5     isnotnull(Image) OR match(_raw, "Masquerading") OR
6     match(_raw, "Masquerade") OR match(_raw, "UAC") OR
7     match(_raw, "uac") OR match(_raw, "Bypass") OR
8     match(_raw, "whoami") OR Description="?" OR
9     match(_raw, "CallTrace"),
10    "true", "false")
11 | stats count(eval(index="buac" AND suspicious="true")) AS TP
12        count(eval(index="normal_events" AND suspicious="false")) AS TN
13        count(eval(index="buac" AND suspicious="false")) AS FN
14        count(eval(index="normal_events" AND suspicious="true")) AS FP
15 | eval Precision=if(TP + FP > 0, TP / (TP+FP), "N/A")
16 | eval Recall=if(TP + FN > 0, TP / (TP+FN), "N/A")
17 | eval Accuracy = if ( TP + TN + FP + FN > 0 , (TP+TN) / (TP+TN+FP+FN),
     " N / A " )
18 | eval F1=if(isnum(Precision) AND isnum(Recall) AND (Precision + Recall)
     > 0, 2 * (Precision * Recall) / (Precision + Recall), "N/A")
19 | table TP, FP, FN, TN, Precision, Recall, Accuracy, F1
```

Listing 5.11: SPL query returning the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score

The evaluation is done in lines 15 to 18 and the last line of the code above creates a table displaying the TP, TN, FP, FN, Precision, Recall, Accuracy, and F1-Score values.

The results of the calculations in Listing 5.11 are shown in Tables 6.8 and 6.17.

Remote Services

The SPL query used to filter for the attack events of the Remote Services is shown below in Listing 5.12.
The first line defines the indices of the events that were used for the filtering. The "normal_events" includes all the normal events and the "remote_services" consists of all the attack events. In the eval statement of the second line, "suspicious" is set to "true" if one or more of the conditions in the if statement is true, otherwise "suspicious" is set to "false". The third line plots the table shown in Figure 5.2. The last line counts the events by index and checks whether the value for "suspicious" is set to "true" or "false".

```
1  index="remote_services" OR index="normal_events"
2  | eval suspicious=if(
3      EventID=4688 OR EventID=5145 OR EventID=4103 OR
4      match(_raw, "(?i)anonym") OR match(_raw, "(?i)offsec") OR
5      (Channel!="Application" AND Channel!="Security" AND
6      Channel!="System") OR
7      (isnull(Workstation) AND match(_raw, "\"Workstation\"")) OR
8      match(UserData, "SubjectUserName") OR match(ProcessName, "-") OR
9      (match(_raw, "AcessList") AND match(_raw, "PrivilegeList")),
10     "true", "false")
11 | stats count by index, suspicious
```

Listing 5.12: SPL filter query of the Remote Services events

Main Search

index	suspicious	count
normal_events	false	127321
normal_events	true	1
remote_services	false	16
remote_services	true	586

Figure 5.2: Output of Listing 5.12

The above figure shows that there were 127'321 TNs, one FP, 16 FN, and 586 TPs. These values are then used for the evaluation shown in Listing 5.13.

```
1  index="remote_services" OR index="normal_events"
2  | eval suspicious=if(
3      EventID=4688 OR EventID=5145 OR EventID=4103 OR
```

```
4       match(_raw, "(?i)anonym") OR match(_raw, "(?i)offsec") OR
5       (Channel!="Application" AND Channel!="Security" AND Channel!="System
        ") OR
6       (isnull(Workstation) AND match(_raw, "\"Workstation\"")) OR
7       match(UserData, "SubjectUserName") OR match(ProcessName, "-") OR
8       (match(_raw, "AcessList") AND match(_raw, "PrivilegeList"),
9       "true", "false")
10  | stats
11      count(eval(index="remote_services" AND suspicious="true")) AS TP
12      count(eval(index="normal_events" AND suspicious="false")) AS TN
13      count(eval(index="remote_services" AND suspicious="false")) AS FN
14      count(eval(index="normal_events" AND suspicious="true")) AS FP
15  | eval Precision=if(TP + FP > 0, TP / (TP+FP), "N/A")
16  | eval Recall=if(TP + FN > 0, TP / (TP+FN), "N/A")
17  | eval Accuracy=if(TP+TN+FP+FN > 0, (TP+TN)/(TP+TN+FP+FN), "N/A")
18  | eval F1=if(isnum(Precision) AND isnum(Recall) AND (Precision + Recall)
        > 0, 2 * (Precision * Recall) / (Precision + Recall), "N/A")
19  | table TP, FP, FN, TN, Precision, Recall, Accuracy, F1
```

Listing 5.13: SPL query showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score

In the above listing, lines two to eight do the filtering and the value assignment of "suspicious" as already mentioned above. The values for the TPs, TNs, FPs, and FNs are assigned on lines nine to thirteen. These were then used on lines 14-17 to calculate the Precision, Recall, Accuracy, and F1-Score values. Lastly, line 18 plots a table that makes the results visible.

The results of these calculations done in Listing 5.13 are shown in Tables 6.11 and 6.17.

In all attack scenarios, it would have been sufficient to filter for whitelisted values of the Computer field. Therefore, the computer names of the author's machines "DESKTOP-FH75FTG" and "DESKTOP-EFV1E39" would have returned all events correctly, because there was always a different value in the attack events. However, this would not work since this field always includes the machine that logged the event. This makes this field a bad feature.

Disable Windows Event Logging

Listing 5.14 shows the SPL query of the Disable Windows Event Logging attack.
Line one starts by defining the indices that were used to filter out the events needed. These were the normal events, included in the index "normal_events", and the attack events of the "dwel" index. Lines two to six then evaluate whether "suspicious" should be set to true or false depending on the event. If at least one condition of the if-statement is true, "suspicious" is also set to "rue", otherwise, it is set to "false". The last line finally counts the events according to their index and whether "suspicious" is true or false. This then returns the table shown in Figure 5.3.

```
1   index=dwel OR index=normal_events
2   | eval suspicious=if(
3       match(_raw, "clear") OR EventID=4688 OR EventID=4908 OR
```

```
    EventID=4719 OR isnotnull(CommandLine) OR
    (Channel!=Security AND Channel!=Application AND Channel!=System) OR
    match(_raw, "hackerserver") OR
    match(_raw, "DomainPolicyChanged") OR
    match(_raw, "AuditPolicyChanges") OR
    match(_raw, "EnableScriptBlockLogging") OR
    "true", "false")
| stats count by index, suspicious
```

Listing 5.14: SPL query to filter the Disable Windows Event Logging events

Main Search

index	suspicious	count
dwel	true	114
normal_events	false	127321
normal_events	true	1

Figure 5.3: Output of Listing 5.14

Figure 5.3 shows that there were 127321 TNs, 114 TPs, and one FP. The evaluation is shown in Listing 5.15.

```
index="dwel" OR index="normal_events"
| eval suspicious=if(
    match(_raw, "clear") OR EventID=4688 OR EventID=4908 OR
    EventID=4719 OR isnotnull(CommandLine) OR
    (Channel!=Security AND Channel!=Application AND Channel!=System) OR
    match(_raw, "hackerserver") OR
    match(_raw, "DomainPolicyChanged") OR
    match(_raw, "AuditPolicyChanges") OR
    match(_raw, "EnableScriptBlockLogging") OR
    "true", "false")
| stats count(eval((index="dwel") AND suspicious="true")) AS TP
        count(eval(index="normal_events" AND suspicious="false")) AS TN
        count(eval((index="dwel") AND suspicious="false")) AS FN
        count(eval(index="normal_events" AND suspicious="true")) AS FP
| eval Precision=if(TP + FP > 0, TP / (TP+FP), "N/A")
| eval Recall=if(TP + FN > 0, TP / (TP+FN), "N/A")
| eval Accuracy=if(TP+TN+FP+FN > 0, (TP+TN)/(TP+TN+FP+FN), "N/A")
| eval F1=if(isnum(Precision) AND isnum(Recall) AND (Precision + Recall)
    > 0, 2 * (Precision * Recall) / (Precision + Recall), "N/A")
| table TP, FP, FN, TN, Precision, Recall, Accuracy, F1
```

Listing 5.15: SPL query for showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score

In the listing above, lines one to twelve evaluate the "suspicious" value and check whether the events fulfill at least one condition. In the next lines, 13-16 define the TPs, TNs, FPs, and FNs. The main evaluation was done in lines 17-20, where the Precision, the Recall, the Accuracy, and the F1-Score were calculated. Lastly, line 21 creates the table visualizing the results of the evaluation. These results are shown in Tables 6.14 and 6.17.

5.2.2 Machine Learning Approach

To get all the versions for the libraries used in the Python scripts, a separate Python file was created. Table 5.1 shows all the libraries used within the Python scripts.

library	Version
os	3.9.13
re	2.2.1
pandas	1.4.4
chardet	4.0.0
datetime	3.9.13[1]
matplotlib	3.5.2
sklearn	1.0.2
Counter	3.9.13[1]
math	3.9.13[1]

Table 5.1: Libraries used for the ML approaches

Brute Force

At the top of the main File, the imports were made as shown in the Listing for the corresponding classifiers. In the main file, three functions were created, shown below in Listings 5.16, 5.17, and 5.18.

```
1 def detect_encoding(file_path):
2     with open(file_path, 'rb') as f:
3         raw_data = f.read()
4 
5     result = chardet.detect(raw_data)
6 
7     return result['encoding']
```

Listing 5.16: detect_encoding function

```
1 def parse_events(xml_file):
2     events = []
3 
4     event_id_pattern = re.compile(r'<EventID[^>]*>(\d+)</EventID>')
5     time_created_pattern = re.compile(r'SystemTime="(\d{4}-\d{2}-\d{2}T\
6                                 d{2}:\d{2}:\d{2}\.\d+Z)"\s*/?>')
7     username_pattern = re.compile(r'<Data Name="TargetUserName">(.*?)<')
```

[1]Part of the standard library of Python, same version as the Python version

```
    encoding = detect_encoding(xml_file)

    with open(xml_file, 'r', encoding=encoding) as f:
        for line in f:
            line = line.strip()

            if not line:  # Skip empty lines
                continue

            # Extract EventID
            event_id_match = event_id_pattern.search(line)
            event_id = event_id_match.group(1) if
                        event_id_match else
                        None

            # Extract TimeCreated SystemTime
            time_created_match = time_created_pattern.search(line)
            timestamp = time_created_match.group(1) if
                        time_created_match else
                        None

            # Extract Username
            username_match = username_pattern.search(line)
            username = username_match.group(1) if
                        username_match else
                        None

            events.append((event_id, timestamp, username))
```

Listing 5.17: parse_events function

```
def load_events_from_files(filepaths):
    all_events = []

    for filepath in filepaths:
        events = parse_events(filepath)

    return all_events
```

Listing 5.18: load_events_from_files function

First, in the file, the normal events were added to a list named “filepaths_normal” and to this list, the function “load_text_from_files” was applied to get the normal events that were converted into a data frame with the columns “EventID”, “Timestamp”, and “Username”. To these normal events, another column named “Label” was added that had label 0.
Next, the attack events were prepared the same way as the normal data; a list was created with the filenames of the attack, and then the events were first put into a list and then into a data frame with the same column names. To the attack events, the column “Label” had the value 1.
After having the two data frames of the normal and attack data, these were concatenated with the command in Listing 5.19.

```
df_features_concatenated = pd.concat([df_events_normal,
                                      df_events_attack], axis=0)
```

Listing 5.19: Combining the normal and attack data frames

After they were combined, all the events were sorted according to their timestamp. To get the count within the second of the event, meaning that at the time the event occurs until one second later (exclusive exactly one second), the code shown in Listing 5.20 was used.

```
df_features_concatenated['CountLast1s'] = df_features_concatenated
['Timestamp'].apply(lambda x: df_features_concatenated[(
    df_features_concatenated['Timestamp'] >= x) &
    df_features_concatenated['Timestamp'] < x +
       pd.Timedelta(seconds=1))].shape[0])
```

Listing 5.20: Counting the events within one second

A similar approach was used for the count of usernames within one second ("CountUsernamesLast1s"), different usernames within one second ("CountDifferentUsernamesLast1s"), and for the same username within one second ("CountSameUsernameLast1s"). The last one, "CountSameUsernameLast1s" did not count the username of the current event, but all the other same usernames of events within one second. "CountDifferentUsernamesLast1s" was used because of the Password Guessing attack and "CountSameUsernameLast1s" was used for the Password Spraying attack. The code for this part of the script is shown in Listing 5.21.

```
# Count usernames within 1 second (not unique)
df_features_concatenated['CountUsernamesLast1s'] =
    df_features_concatenated.apply(
     lambda row:
     df_features_concatenated[(df_features_concatenated['Timestamp'] >=
         row['Timestamp']) &
     (df_features_concatenated['Timestamp'] <
         row['Timestamp'] + pd.Timedelta(seconds=1)) &
     (df_features_concatenated['Username'].notna()) &
     (df_features_concatenated['Username'] != '')].shape[0], axis=1)

# Count same usernames within 1 second
df_features_concatenated['CountSameUsernameLast1s'] =
    df_features_concatenated.apply(
     lambda row:
     df_features_concatenated[(df_features_concatenated['Timestamp'] >=
         row['Timestamp']) &
     (df_features_concatenated['Timestamp'] <
         row['Timestamp'] + pd.Timedelta(seconds=1)) &
     (df_features_concatenated['Username'] ==
         row['Username'])].shape[0], axis=1)

# Count different usernames within 1 second
df_features_concatenated['CountDifferentUsernamesLast1s'] =
    df_features_concatenated.apply(
     lambda row:
     df_features_concatenated[(df_features_concatenated['Timestamp'] >=
         row['Timestamp']) &
     (df_features_concatenated['Timestamp'] <
         row['Timestamp'] + pd.Timedelta(seconds=1)) &
     (df_features_concatenated['Username'] !=
```

```
29          row['Username'])]['Username'].nunique(), axis=1)
```

Listing 5.21: Counting usernames within one second (same username, different username, and all usernames)

Next, the "Timestamp" and "Username" columns were removed, since the model should not learn about the time or the usernames, only about the amount of events taking place and the number of usernames.
In the following step, the "X" and "y" were defined for splitting the data, which will be used for training the model. "X" is defined as the entire table without the column "Label" since the model should not know which events belong to the attack and which are normal events. The "y" includes only the "Label" column that is used to check whether the model has been predicted correctly. The data splitting for training the model was done by the following line of code (Listing 5.22):

```
X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size=0.2, random_state=42)
```

Listing 5.22: Splitting the data for training the model

Only the OCSVM did not need to be split into testing and training data. All the normal events were used for training in this model.
Before and after the training, the time was taken to get the Runtime Performance. The end time was subtracted from the start time to get the time that had passed until the training finished.
The training itself was then done depending on the algorithm used. First, the Random Forest algorithm was used, because according to [27], this classifier seems to be the most promising algorithm when it comes to Brute Force attacks. The training and fitting of this model are shown in Listing 5.23 on lines four and five.

```
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Capture the start time
start_time = datetime.utcnow()

rf.fit(X_train, y_train)

# Capture the end time
end_time = datetime.utcnow()

# Calculate the training duration
training_duration = end_time - start_time
```

Listing 5.23: Training and fitting the model for a Random Forest algorithm and calculating the Runtime Performance

The training of an MLP was done with the following code:

```
mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=300,
                    random_state=42)
mlp.fit(X_train, y_train)
```

Listing 5.24: Training and fitting of the model for a MLP algorithm

The training of a SVM was done with the following code:

```
svm = SVC(random_state=42)
svm.fit(X_train, y_train)
```

Listing 5.25: Training and fitting of the model for a SVM algorithm

The training of a Logistic Regression was done with the following code:

```
log_reg = LogisticRegression(max_iter=1000, random_state=42)
log_reg.fit(X_train, y_train)
```

Listing 5.26: Training and fitting of the model for a Logistic Regression algorithm

The training of a Naive Bayes was done with the following code:

```
nb = MultinomialNB()
nb.fit(X_train, y_train)
```

Listing 5.27: Training and fitting of the model for a Naive Bayes algorithm

Lastly, the training of the OCSVM was done as follows:

```
ocsvm = OneClassSVM()
ocsvm.fit(X_train)
```

Listing 5.28: Training and fitting of the model for an OCSVM algorithm

The "random_state" allows to reproduce the results. Moreover, the OCSVM was trained only on normal data and no attack events, whereas the others were trained on both attack and normal events. For the OCSVM, the approach is therefore a bit different from the others. This model was run on the attack data from the GitHub repositories as well, but for comparison, only the performance of the three-day dataset was taken into consideration.
Now that the model is fitted, the new data (the data of the three-day dataset) is loaded and the same preprocessing is applied. After the events were sorted according to the "Timestamp" column, the counts were generated similarly to Listing 5.20 and 5.21.
To check that all events received a new label during the next phase in which the attack events were distinguished from the normal events for the three-day dataset to a list of the length of the number of events, 5 was assigned. To define which events were normal and which were not, the same pattern as in the traditional approach was applied, all events with ID 4625 were classified as attacks. Moreover, if an event with ID 4648 occurred right before an event with ID 4625 and this happened within 0.1 seconds, the event with ID 4648 was also classified as an attack. All other events were labeled as normal events with label 0. Listing 5.29 shows the corresponding code part.

```
for i in range(len(new_df) - 1):
    current_event = new_df.iloc[i]
    next_event = new_df.iloc[i + 1]
    time_diff = next_event['Timestamp'] - current_event['Timestamp']

    if pd.notnull(current_event['EventID']):
        if (current_event['EventID'] == 4648) and
            (next_event['EventID'] == 4625) and
            (time_diff <= pd.Timedelta(seconds=0.1)):
```

```
            labels[i] = 1 # Attack event

        else:
            labels[i] = 0  # Normal event

        if current_event['EventID'] == 4625:
            labels[i] = 1  # Labels all 4625 events as attack,
                           # Also if not in combination with 4648
```

Listing 5.29: Labeling of the three-day dataset, differentiating between normal and attack events

This newly created list was then added to the data frame of the three-day dataset. To do the predictions, a new variable called "X_new" was created that consisted of the two columns "EventID", "CountUsernamesLast1s", "CountLast1s", "CountSameUsernameLast1s", and "CountDifferentUsernamesLast1s". To get the time the prediction took, again the time was taken before and after the prediction, which returned the total prediction time if the start time was subtracted by the end time. This resulting time needed to be divided by the number of events to get the average prediction time. The prediction itself is shown in Listing 5.30.

```
# Random Forest
new_df['PredictedLabel'] = rf.predict(X_new)

# MLP
new_df['PredictedLabel'] = mlp.predict(X_new)

# SVM
new_df['PredictedLabel'] = svm.predict(X_new)

# Logistic Regression
new_df['PredictedLabel'] = log_reg.predict(X_new)

# Naive Bayes
new_df['PredictedLabel'] = nb.predict(X_new)

# OCSVM
new_df['PredictedLabel'] = ocsvm.predict(X_new)

# OCSVM outputs -1 for outliers (anomalies) and 1 for inliers (normal
    data). Change these values to 0 and 1
new_df['PredictedLabel'] = new_df['PredictedLabel'].apply(lambda x: 0 if
                           x == 1 else 1)
```

Listing 5.30: Code for predicting the outcomes of the models

To evaluate the performance of the model, the two variables "y_true", including the true labels, and "y_pred", containing the labels predicted by the models, were defined through which the TP, TN, FP, and FN could be determined. This can be seen in Listing A.6. By using these four values, the Precision, Recall, Accuracy, and F1-Score values could be determined.

Two plots were generated to visualize the data. The first plot showed the actual labels, and the predicted ones but has also marks for all the misclassified and correctly classified

events predicted by the used model. The code to produce this plot is shown in Listing A.7.
Listing (A.8) displays code for a similar plot, but without the misclassified and correctly classified points. Instead, the TP, TN, FP, and FN are shown in the plot.

For both Listings (A.7 and A.8), the titles were adjusted to include the classifier name in the string of the title.

Bypass User Account control

The functions “detect_encoding” and “load_events_from_files” remained identical to the Brute Force attack use case (shown in Tables 5.16 and 5.18.

For the Bypass User Account Control attack, the function “parse_events” required some modifications. This adjusted function is shown in Listing 5.31.

```
def parse_events(xml_file):
    events = []

    patterns = {
        'event_id': re.compile(r'<EventID[^>]*>(\d+)</EventID>'),
        'system_time': re.compile(r'SystemTime=
            "(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d+Z)"\s*/?>'),
        'image': re.compile(r'Name="Image">(.*?)</Data>'),
        'parent_command_line': re.compile(r'<Data Name=
            "ParentCommandLine">(.*?)</Data>'),
        'command_line': re.compile(r'<Data Name=
            "CommandLine">(.*?)</Data>')
    }

    encoding = detect_encoding(xml_file)

    with open(xml_file, 'r', encoding=encoding) as f:
        for line in f:
            line = line.strip()
            if not line:  # Skip empty lines
                continue

            event_data = {key: None for key in patterns}

            for key, pattern in patterns.items():
                match = pattern.search(line)
                if key == 'image':
                    event_data[key] = 1 if match else 0
                else:
                    event_data[key] = match.group(1) if match else None

            events.append((
                event_data['event_id'], event_data['system_time'],
                event_data['image'],
                event_data['parent_command_line'],
                event_data['command_line']
            ))

```

```
39     return events
```

Listing 5.31: parse_events function for Bypass User Account Control attack

In addition, there was one more function defined. The "compute_entropy", was used to compute the entropy of the fields "ParentCommandLine" and "CommandLine". The listing below shows the "compute_entropy" function:

```
1 def compute_entropy(s):
2     if not s:  # Handles empty strings
3         return 0
4 
5     counter, len_s = Counter(s), float(len(s))
6 
7     return -sum(count/len_s * math.log(count/len_s, 2) for
8             count in counter.values())
```

Listing 5.32: compute_entropy function for Bypass User Account Control attack

In the main part of the file, after defining these functions, the normal events were added to a list called "filepaths_normal", to this list the function "load_events_from_files" was applied to get the normal events that were converted into a data frame with the columns "EventID", "SystemTime", "Image", "ParentCommandLine", and "CommandLine". Next, the "EventID" was converted to an integer, and the "SystemTime" was converted to a datetime-format, additionally, to all these events, zero was assigned to the "Label" column to distinguish between normal and attack events later on. The same was made with the attack files, except "Label" was set to one there.
After that, the two data frames were concatenated and sorted by "SystemTime", the concatenation was done in the same manner as in Listing 5.19. The sorting was done as shown in the listing below:

```
1 df_features_concatenated = df_features_concatenated.sort_values(
2                                     by='SystemTime')
```

Listing 5.33: Sorting according to the "SystemTime" column

Following, four new columns were created as follows:

"CommandLineLength": shows the length of the "CommandLine" field, and if empty, returns zero

"ParentCommandLineLength": returns the length of the "ParentCommandLine" field and, if empty, returns zero

"CommandLineEntropy": returns the entropy of the "CommandLine" field, and if empty, returns zero

"ParentCommandLineEntropy": returns the entropy of the "ParentCommandLine" field, and if empty returns zero

The code for creating these fields looks as follows:

```
df_features_concatenated['CommandLineLength'] =
    df_features_concatenated['CommandLine'].apply(
        lambda x: len(str(x)) if x else 0)
df_features_concatenated['ParentCommandLineLength'] =
    df_features_concatenated['ParentCommandLine'].apply(
        lambda x: len(str(x)) if x else 0)
df_features_concatenated['CommandLineEntropy'] =
    df_features_concatenated['CommandLine'].apply(
        lambda x:    compute_entropy(str(x)) if x else 0)
df_features_concatenated['ParentCommandLineEntropy'] =
    df_features_concatenated['ParentCommandLine'].apply(
        lambda x: compute_entropy(str(x)) if x else 0)
```

Listing 5.34: Code for creating four new columns

In the next step, the columns "CommandLine" and "ParentCommandLine" could be dropped since the models should not learn about these two fields.
This concatenated data frame was then saved and subsequently the value assignment for "X" and "y" was done. For "X", the whole data frame was taken except the "Label" column, and for "y", only this "Label" column was used. The splitting was done exactly as in Listing 5.22. From the "X_test", a copy of the "SystemTime" column was taken that was used later for plotting the graphs. After this copy was taken, the "SystemTime" column was dropped from "X_train", "X_test", "y_train", and "y_test". For the OCSVM model, there was an additional line:

```
df_features_concatenated_for_prediction = df_features_concatenated.drop(
    columns=['SystemTime', 'Label'])
```

Listing 5.35: New data frame for the OCSVM model, needed for the prediction

The time was taken before and after training the models, which was done in the same manner as for the Brute Force attack (shown in Listing 5.23). Also, for the other trainings, they looked the same as shown in Listings 5.24-5.27. Only for the OCSVM model, it was a bit different because now, the normal data had to be split into training and testing data. The training and fitting for this model looked as shown in Listing 5.36 below:

```
ocsvm = OneClassSVM()
ocsvm.fit(X_train, y_train))
```

Listing 5.36: Training and fitting of the OCSVM model

For the prediction, again, the time was taken before and after training had taken place and the total time as well as the average prediction time were calculated. The code for all the models looked the same as shown in Listing 5.30 except that for the models inside the predict brackets "X_test" from the train-test-split is used instead of "X_new". For the OCSVM, the value of "df_features_concatenated_for_prediction" is used in the bracket (created in Listing 5.35) instead of "X_new". Another difference was that instead of creating a new column, a new variable was created ("y_pred") that included the predicted values.
For the classification report and the confusion matrix, the code in Listing 5.37 was used. "y_train" are the true labels of the train-test-split and "y_pred" are the predicted labels.

```
# Generate classification report and accuracy
classification_report_str = classification_report(y_test, y_pred_test)
accuracy = accuracy_score(y_test, y_pred_test)

# Combine confusion matrix and accuracy with classification report
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test).ravel()
full_report = (
    f"True Negatives (TN): {tn}\n"
    f"False Positives (FP): {fp}\n"
    f"False Negatives (FN): {fn}\n"
    f"True Positives (TP): {tp}\n"
    f"Accuracy: {accuracy}\n\n"
    f"{classification_report_str}"
)
```

Listing 5.37: Code for generating the classification report and confusion matrix

This report was saved for later revision. The predictions were then added to the test set and the "SystemTime" was added again for creating the plots, which is shown in Listing 5.38. The OCSVM used a slightly different plotting preparation because of the difference in training and testing data, which is shown in Listing 5.39.

```
X_test['PredictedLabel'] = y_pred_test
X_test['ActualLabel'] = y_test.values
X_test['SystemTime'] = system_time_test  # Add back SystemTime for
    plotting
```

Listing 5.38: Adding columns to prepare the plotting

```
plot_data=
    df_features_concatenated_for_prediction_sorted_time_with_labels
plot_data['PredictedLabel'] = y_pred_test
plot_data['ActualLabel'] = y_true.values
```

Listing 5.39: Adding columns to prepare the plotting for the OCSVM

The code for plotting the graphs is shown in the Appendix.

The resulting plots of the correct and misclassified labels of Listing A.10 and for all the other models are shown in Figures B.17 and B.18. The plots for the TPs, TNs, FPs, and FNs of Listing A.11 and all the other classifiers are shown in Figures B.19 and B.20.

Remote Services

For the Python scripts, the same libraries as for the Brute Force attack were used, therefore they are all listed in Table 5.1. Also, the same imports were used that are visible in Listings A.3 and A.5. However, there was one additional import that had to be done to encode the string values of the "Channel" and the "ProcessName" columns to an integer (shown in Listing A.4).

Moreover, the two functions used in the Brute Force attack stayed the same, namely the "detect_encoding" and "load_events_from_files". These are shown in Listings 5.16 and

5.18. The “parse_events” function is different and more similar to the same function of the Bypass User Account Control attack. For the Remote Services, this function looked like presented in Listing 5.40 with some smaller differences compared to the Bypass User Account Control script.

```
def parse_events(xml_file):
 events = []

    patterns = {
        'event_id': re.compile(r'<EventID[^>]*>(\d+)</EventID>'),
        'system_time': re.compile(r'SystemTime=
            "(\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d+Z)"\s*/?>',,
        'offsec_keyword': re.compile(r'(?i)\boffsec\b'),
        'channel': re.compile(r'<Channel>([^<]+)</Channel>'),
        'process_name': re.compile(
                        r'<Data Name="ProcessName">([^<]*)</Data>')
    }

    encoding = detect_encoding(xml_file)

    with open(xml_file, 'r', encoding=encoding) as f:
        for line in f:
            line = line.strip()
            if not line:  # Skip empty lines
                continue

            event_data = {key: None for key in patterns}

            for key, pattern in patterns.items():
                match = pattern.search(line)
                if key == 'offsec_keyword':
                    event_data[key] = 1 if match else 0
                else:
                    event_data[key] = match.group(1) if match else None

            events.append((
                event_data['event_id'], event_data['system_time'],
                event_data['offsec_keyword'], event_data['channel'],
                event_data['process_name']


            ))

    return events
```

Listing 5.40: parse_events function for the Remote Services attack

Starting with the main part of the script, the normal events were loaded and subsequently put into a data frame with the four columns “EventID”, “SystemTime”, “Computer”, and “Offsec”. The “EventID” column was converted to integers, the “SystemTime” to a date-time format and additionally, a column “Label” was created that had the value zero for all the normal events.
The same was done with the attack events, except that the “Label” column had the value one. In a next step, these two data frames containing the normal and attack data were concatenated into one data frame and sorted according to “SystemTime” which is visible

in Listing 5.33.
After the two data frames were concatenated, an integer value was assigned to the "Channel" and the "ProcessName" columns, because the values of these columns were a string, they had to be converted to integers. Two approaches were taken into consideration, shown in Listing 5.41.

```
# Variant 1: One-hot encoding of categorical features
df_features_concatenated = pd.get_dummies(df_features_concatenated,
    columns=['Channel', 'ProcessName'], dummy_na=True, drop_first=True)


# Variant 2: String to integer transformation:
# Create separate encoders for 'Channel' and 'ProcessName'
channel_encoder = LabelEncoder()
processname_encoder = LabelEncoder()

# Fit on the combined dataset first to ensure consistent labels
# Channel Encoding
df_features_concatenated['Channel_encoded'] = channel_encoder.
    fit_transform(df_features_concatenated['Channel'])

# Now transform for normal and attack data based on the fitted encoder
df_events_normal['Channel_encoded'] = channel_encoder.transform(
    df_events_normal['Channel'])
df_events_attack['Channel_encoded'] = channel_encoder.transform(
    df_events_attack['Channel'])

# ProcessName Encoding
df_features_concatenated['ProcessName_encoded'] = processname_encoder.
    fit_transform(df_features_concatenated['ProcessName'])

# Now transform for normal and attack data based on the fitted encoder
df_events_normal['ProcessName_encoded'] = processname_encoder.transform(
    df_events_normal['ProcessName'])
df_events_attack['ProcessName_encoded'] = processname_encoder.transform(
    df_events_attack['ProcessName'])
```

Listing 5.41: One-hot encoding and string to integer transformation

For the OCSVM, these two variants were a bit more difficult. It is shown how it was done in the listing below:

```
# Variant 1: One-Hot Encoding
# Apply One-Hot Encoding to the 'Channel' column for normal events
df_events_normal = pd.get_dummies(df_events_normal, columns=['Channel'],
    dummy_na=True, drop_first=True)

# Apply One-Hot Encoding to the 'Channel' column for attack events
df_events_attack = pd.get_dummies(df_events_attack, columns=['Channel'],
    dummy_na=True, drop_first=True)

# Apply One-Hot Encoding to the 'ProcessName' column for normal events
df_events_normal = pd.get_dummies(df_events_normal, columns=['
    ProcessName'], dummy_na=True, drop_first=True)

# Apply One-Hot Encoding to the 'ProcessName' column for attack events
```

```
df_events_attack = pd.get_dummies(df_events_attack, columns=['ProcessName'], dummy_na=True, drop_first=True)

# Get all unique column names across both data frames
all_columns = set(df_events_normal.columns).union(set(df_events_attack.columns))

# Fill missing columns with 0 in the normal dataset
for col in all_columns:
    if col not in df_events_normal.columns:
        df_events_normal[col] = 0

# Fill missing columns with 0 in the attack dataset
for col in all_columns:
    if col not in df_events_attack.columns:
        df_events_attack[col] = 0

# Ensure the same order of columns
df_events_normal = df_events_normal[sorted(df_events_normal.columns)]
df_events_attack = df_events_attack[sorted(df_events_attack.columns)]

# Concatenate the data frames
df_features_concatenated = pd.concat([df_events_normal, df_events_attack], axis=0)


# Variant 2: String to integer transformation
# Combine normal and attack data temporarily for encoding
combined_df = pd.concat([df_events_normal, df_events_attack], axis=0)

# Initialize the label encoders
channel_encoder = LabelEncoder()
processname_encoder = LabelEncoder()

# Fit the LabelEncoders on the combined data
combined_df['Channel_encoded'] = channel_encoder.fit_transform(combined_df['Channel'])
combined_df['ProcessName_encoded'] = processname_encoder.fit_transform(combined_df['ProcessName'])

# Reset the index for all data frames to avoid duplicate index conflicts
df_events_normal = df_events_normal.reset_index(drop=True)
df_events_attack = df_events_attack.reset_index(drop=True)
combined_df = combined_df.reset_index(drop=True)

# Split the encoded data back into normal and attack datasets
df_events_normal['Channel_encoded'] = combined_df.loc[combined_df.index.isin(df_events_normal.index), 'Channel_encoded']
df_events_normal['ProcessName_encoded'] = combined_df.loc[combined_df.index.isin(df_events_normal.index), 'ProcessName_encoded']

df_events_attack['Channel_encoded'] = combined_df.loc[combined_df.index.isin(df_events_attack.index), 'Channel_encoded']
df_events_attack['ProcessName_encoded'] = combined_df.loc[combined_df.index.isin(df_events_attack.index), 'ProcessName_encoded']
```

Listing 5.42: One-hot encoding for the OCSVM classifier

If the one-hot encoding was not used when defining the "X" variable, the "Channel" and the "ProcessName" columns needed to be dropped in addition. Since the second variant returned better results for the best model, the Random Forest, it was decided to only use this approach for this attack scenario. Nevertheless, for the SVM and the OCSVM classifiers, both variants returned equally well results and the remaining three classifiers returned worse results with the second variant. Moreover, while using different features, the one-hot encoding sometimes returned better results. Therefore, for different datasets or features, it may be advantageous to test both approaches for which one suits better.
Now, everything was prepared for training the model, and two variables "X" and "y" were defined. "X" included the whole data frame without the "Label" column and "y" exclusively consisted of this one "Label" column. Moreover, the "train_test_split" function was applied to these two variables as shown in Listing 5.22.
As already done in the Bypass User Account Control script, a copy of the "SystemTime" column of "X_test" was taken to create the graphs later on. Then the "SystemTime" column was dropped from all the four variables "X_train", "X_test", "y_train", and "y_test". Since the training data now was ready, the model could be trained and fitted. This was done in the same manner as shown in Listings 5.24-5.27, 5.36 for the OCSVM model, and 5.23 in lines four and five for the Random Forest classifier. The time was again taken before and after the training and fitting.
For the prediction, the time was also taken before and after prediction to get the total amount of time it took to predict the data, but also the average time a prediction took was calculated. The code for prediction looked the same as shown in Listing 5.30 with the exception that inside the brackets of the "predict" function "X_new" is exchanged with "X_test". For the OCSVM model "df_features_concatenated_for_prediction_wo_timeLabel" was used inside the brackets instead of "X_new", which did not include the label and the time columns. Another difference was that instead of creating a new column, a new variable was created ("y_pred") that included the predicted values.
The classification report and the confusion matrix were done as already presented in Listing 5.37, but for the OCSVM instead of "y_test", "y_true" was used. The predictions were added as shown in Listing 5.38, and the plotting was done as written in Listings A.10 and A.11. A slightly different approach was used for the OCSVM since it was trained on only the normal events. This is shown in Listing 5.39.
The resulting plots of the correctly and wrongly classified labels are shown in Figures B.21 and B.22 and the plots for the TPs, TNs, FPs, and FNs are visible in Figures B.23 and B.24.

Disable Windows Event Logging

For the Python script of this attack, again, the same libraries were used as for the Brute Force, Bypass User Account Control, and Remote Services attacks, as shown in Table 5.1. The imports for these scripts stayed the same as in Listings A.3 and A.5 additionally, the imports of the previous attacks were also used.
The functions "detect_encoding", "load_events_from_files", and "compute_entropy" (Listings 5.16, 5.18, and 5.32) used for the previous attacks stayed the same, but the function "parse_events" was slightly different because other features were used. The adjusted "parse_events" function is shown in the listing below:

```
def parse_events(xml_file):
    events = []

    patterns = {
        'event_id': re.compile(r'<EventID[^>]*>(\d+)</EventID>'),
        'system_time': re.compile(r'SystemTime="(\d{4}-\d{2}-\d{2}T\d
    {2}:\d{2}:\d{2}\.\d+Z)"\s*/?>'),
        'commandline': re.compile(r'<CommandLine>([^<])<'),
        'channel': re.compile(r'<Channel>([^<])</Channel>')

    }

    encoding = detect_encoding(xml_file)

    with open(xml_file, 'r', encoding=encoding) as f:
        for line in f:
            line = line.strip()
            if not line:  # Skip empty lines
                continue

            event_data = {key: None for key in patterns}

            for key, pattern in patterns.items():
                match = pattern.search(line)
                event_data[key] = match.group(1) if match else
    event_data[key]

            events.append((
                event_data['event_id'], event_data['system_time'],
                event_data['commandline'], event_data['channel']

            ))

    return events
```

Listing 5.43: parse_events function for the Disable Windows Event Logging attack

After defining the functions, the scripts for the Disable Windows Event Logging start by loading the normal events with the "load_events_from_files" function after which they were put into a data frame with the columns "EventID", "SystemTime", "CommandLine", and "Channel". As already done for the Remote Services, the "EventID" column was converted to integers, the "SystemTime" to a datetime format and additionally, a column "Label" was created that had the value zero for all the normal events.
The same was done with the attack events, except that the "Label" column has the value one. In a next step, these two data frames containing the normal and attack data were concatenated into one data frame and sorted according to "SystemTime" which is visible in Listing 5.33, similarly to the Remote Services use case.
After concatenating the two data frames, the "CommandLineLength" and the "CommandLineEntropy" were calculated as already written in Listing 5.34. Then, an integer value needed to be assigned to the columns "CommandLine" and "Channel" since they included string values. Listing 5.44 below shows the code for two different approaches to achieve this transformation. For the OCSVM in the first transformation variant, also the "df_events_normal" and "df_events_attack" were run separately with the "LabelEncoder".

In the second variant, the OCSVM also had a slightly different approach, similar to the one presented in Listing 5.42.

```
## Variant 1
# Initialize the label encoder
label_encoder_channel = LabelEncoder()
label_encoder_command = LabelEncoder()

# Fitting and transforming of the 'Channel' column
df_features_concatenated['Channel_encoded'] = label_encoder_channel.
    fit_transform(df_features_concatenated['Channel'])

# Fitting and transforming of the 'CommandLine' column
df_features_concatenated['CommandLine_encoded'] = label_encoder_command.
    fit_transform(df_features_concatenated['CommandLine'])

# Transforming of the 'Channel' column
df_features_concatenated['Channel_encoded'] = label_encoder_channel.
    transform(df_features_concatenated['Channel'])
df_features_concatenated['CommandLine_encoded'] = label_encoder_command.
    transform(df_features_concatenated['CommandLine'])

## Variant 2
# One-hot encode categorical features
df_features_concatenated = pd.get_dummies(df_features_concatenated,
    columns=[
     'CommandLine', 'Channel'
], dummy_na=True, drop_first=True)
```

Listing 5.44: Transformating the "Channel" and "CommandLine" columns

For the former approach, the columns "Channel" and "CommandLine" needed to be dropped additionally when defining "X". Since the two approaches in the above listing returned equal results, it was decided to use the one-hot encoding.
In the next step, the model was trained and two variables "X" and "y" were defined, where the former included the entire data frame except the "Label" column and the latter only consisted of this "Label" column. Then, the "train_test_split" function was applied to these two variables as shown in Listing 5.22.
As already done in the previous scripts of the Bypass User Account Control and the Remote Services attacks, a copy of the "SystemTime" column of "X_test" was taken to plot the graphs at the end of the script. After creating this copy, the "SystemTime" column was dropped from all variables used for training ("X_train", "X_test", "y_train", and "y_test") and the model was trained and fitted as shown in Listings 5.24-5.27, 5.36 for the OCSVM model, and 5.23 in lines four and five for the Random Forest classifier. For this attack, too, the time was taken before and after the training and fitting. The time was also measured for the prediction, where the average prediction time was also calculated. For the prediction, the code of Listing 5.30 was reused with the same exception as in the previous attack, namely that instead of "X_new", "X_test" was used. For the OCSVM classifier, the "X_new" was exchanged by "df_features_concatenated_for_prediction_wo_timeLabel". Another difference was that instead of creating a new column, a new variable was created ("y_pred_test") that included the predicted values of the model.
The classification report and the confusion matrix were created as already presented in Listing 5.37. "y_pred_test" instead of "y_true" was used for the OCSVM, which consisted

of the “Label” column of the “df_features_concatenated_wo_timeLabel” data frame. The predictions were added as shown in Listing 5.38, again for the OCSVM the approach was slightly different, which is shown in Listing 5.39, and the plotting was done as written in Listings A.10 and A.11.
The OCSVM used a slightly different approach since it was trained on only the normal events. This is shown in Listing 5.39.

Chapter 6

Results and Evaluation

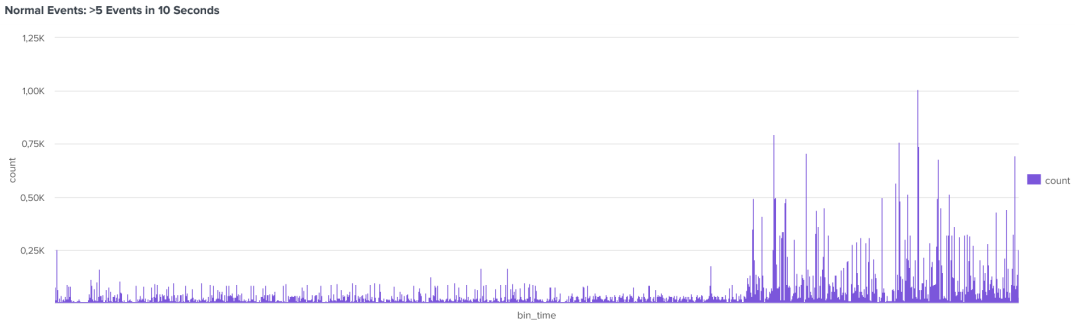
In this chapter, the findings are presented. For each attack, first the results of the traditional and subsequently, those of the ML approach are presented. The first attack examined was the Brute Force attack, followed by the Bypass User Account Control, the Remote Services, and finally, the Disable Windows Event Logging attacks.
After presenting the findings in Section 6.1 the results are then evaluated in the second part of this chapter (Section 6.2). Key findings are summarized in tables in the evaluation part and then evaluated. The evaluation is again divided into the attacks considered and concludes with a section about all the attacks taken into consideration (Section 6.2.4).

6.1 Results

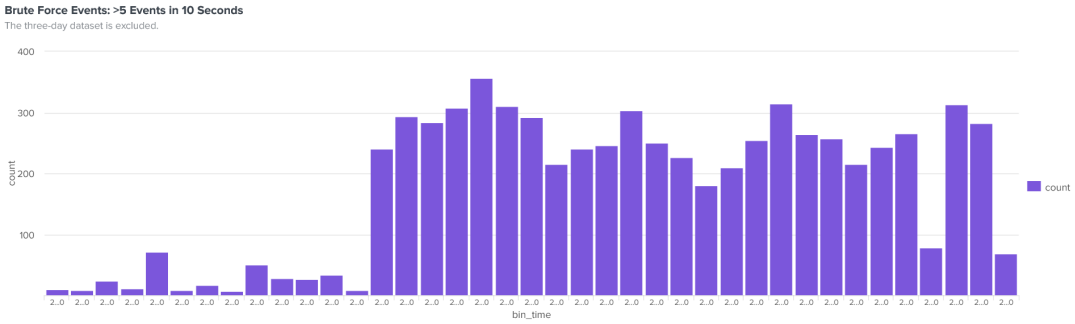
6.1.1 Brute Force Attack

Traditional Approach

The first attack to investigate was the Brute Force attack. This technique belongs to the Credential Access tactic and includes the two sub-techniques Password Guessing and Password Spraying. Then the own Brute Force attacks were launched, and the log files were saved. The attack data was preprocessed and uploaded to the "brute_force" index in Splunk and the fields were extracted. This gave a total of 6'843 events, of which most (6'517) belonged to the own Password Spraying and Password Guessing attack.
Afterward, to visualize the data better and get a deeper understanding of how Brute Force attacks may work, a dashboard was created in Splunk. This dashboard gives insights into the differences between normal and anomalous behaviour through graphs and tables.
Figure 6.1 visualizes the difference between all normal and Brute Force events, where more than five events occurred within ten seconds. The code creating the figure is shown in Listings A.14 for normal events and A.15 for attack events. It looks as if there are many more normal events that fulfill this condition, however, there are also many more normal events (127'322) compared to Brute Force events (6'843).



(a) Normal Events



(b) Attack Events

Figure 6.1: Difference between normal and Brute Force events in the data, if more than five events occurred in ten seconds

After looking at the frequency of the occurring events, the EventIDs of the normal data were analyzed. As can be seen in Figure 6.2, by far the most common EventID is 5379, telling that the credentials were read by the Credential Manager [67]. The code to create this figure is shown in Listing A.16.

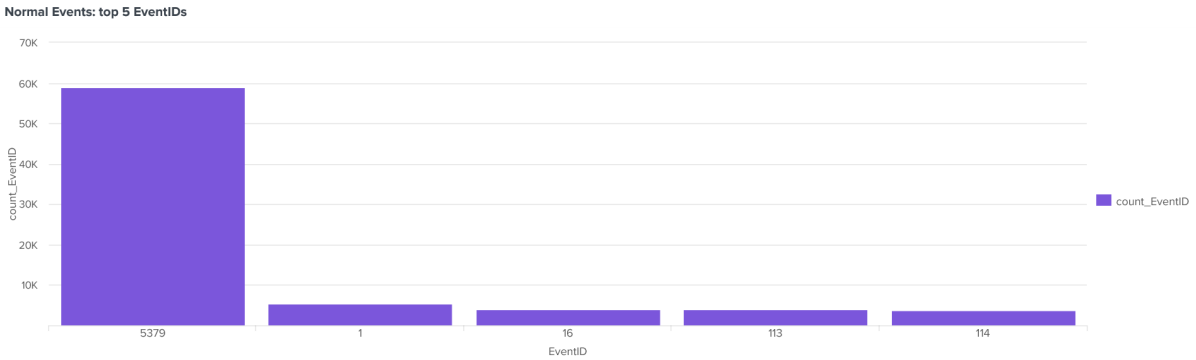


Figure 6.2: Top five EventIDs of normal events

After that, the Brute Force events were analyzed. Since the own Password Guessing and Password Spraying attacks had mostly the EventIDs 4625 (“An account failed to log on” [67]) and 4648 (“A logon was attempted using explicit credentials” [67]), they were looked at separately and then the rest of the attack files were investigated. The code

for creating Figure 6.3 is visible in Listing A.17, this figure shows all EventIDs from the Brute Force attacks except the own attacks, clearly the most frequent EventID is 33205, but all Structured Query Language (SQL) audit events are logged under this EventID.

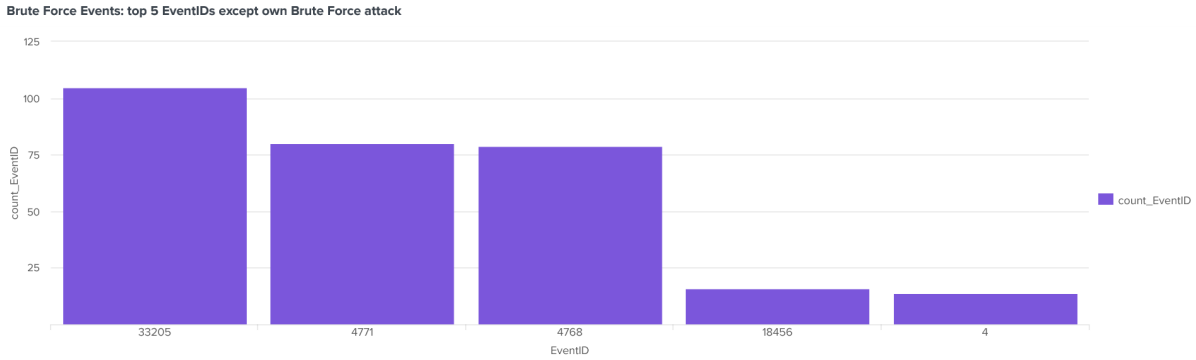


Figure 6.3: Brute Force Events EventIDs except for the own Brute Force attacks

Figure 6.4 below shows all the EventIDs of the own attacks. It has almost only the two EventIDs 4625 and 4648 and visualizes that these two most common EventIDs appear nearly the same number of times. The code for setting up the graph is shown in Listing A.18.

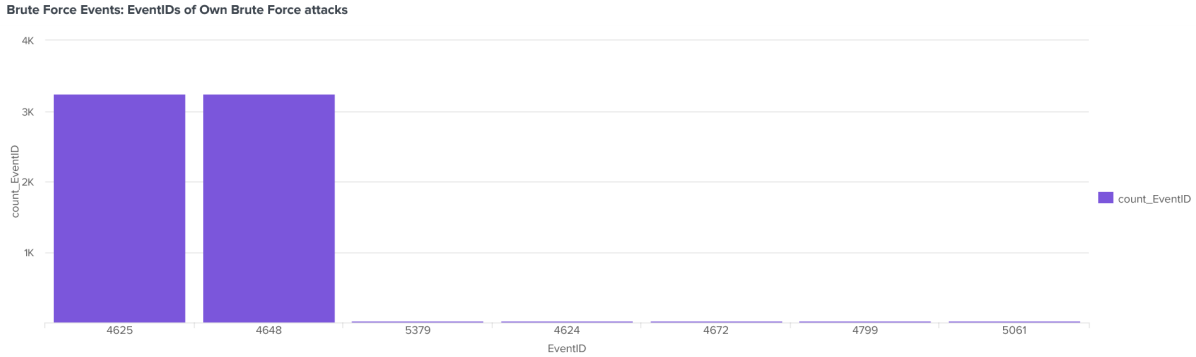


Figure 6.4: Brute Force attacks EventIDs of the own attacks if ten events took place within one second

Figures B.1 and B.2 combine both approaches used above and the corresponding code can be found in Listings A.12 and A.13, respectively. The former figure shows the top ten EventIDs of the normal data if, within one second, more than ten events took place. The most frequent EventID is 5379, telling that the credentials were read by the Credential Manager [67]. Figure B.2 shows the EventIDs of attack data if within one second more than ten events took place, but it does not include the own Password Guessing and Password Spraying attacks.

The figure below displays all the own attack events if within one second more than ten events took place. It shows a very similar picture as already seen in Figure 6.4. The code of this figure is shown in Listing A.19.

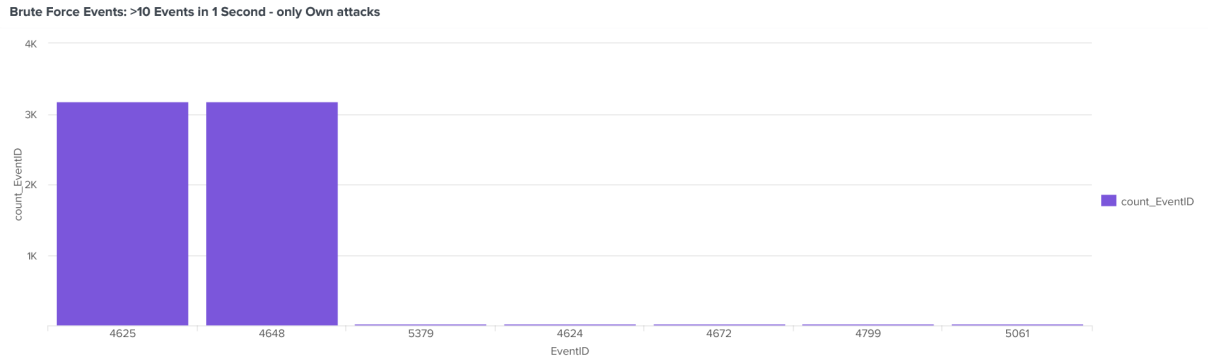


Figure 6.5: Shows all EventIDs of own Brute Force attack if within one second more than ten events took place

In the next step, it was checked which events included words containing the keyword "fail". Figure B.3 shows the counts by index and words found. The code can be found in Listing A.20. This returned 3'855 events, of which 3'376 belonged to Brute Force attack events. This was then repeated for words including "fail" and "log". The result is shown in Figure B.4 and the code is presented in Listing A.21. It is visible that by far the most events, including one or both words, correspond to Brute Force events.

After understanding the foundation of the Brute Force attacks, the next step was to create a SPL query that filters out most events that belonged to the Brute Force index and as few as possible normal events. To start, EventIDs corresponding to failed authentication and failed login were filtered out according to [67] and put into the query, leading to the first part, visible on line three in Listing 5.4. The code mentioned in Listing 5.4 concatenates two indices, including the Brute Force events and the events of the normal system behaviour. It excludes the sourcetype "All Brute Force Attacks" since this includes all events and if it would be included, it would just double all the events. The code on line three in Listing 5.4 returned 3'376 events, most of them are Brute Force events, and only 27 include normal events. It was noticed that never more than one of the filtered EventIDs from normal events occurred within a ten-second window, whereas Brute Force events always occurred more than one time in the same time window. Therefore, the query was refined to create a graph that only includes Brute Force events. Listing 5.4 shows the final code that creates the chart in Figure B.5. Since in this plot, only attack events showed up, this assured that the EventIDs chosen will not filter out normal events. The same EventIDs were applied to the Password Guessing and Spraying attack data, and there too, only attack data showed up. This is shown in Figure 6.5. Finally, the event count pattern was investigated and indeed there was a different pattern while the attacks were launched according to the amount of events taking place.

Now the traditional approach was evaluated, including the normal and attack data except for the three-day dataset. The results are shown in Table 6.2 and 6.1.

Before finishing with the traditional approach, during three-day, more Brute Force attacks were conducted to look for similar patterns. Password Spraying and Password Guessing were executed on both machines of the author to gather more data. Figures 6.6, 6.7, and 6.8 below shows the data of the three days where the own Brute Force attacks were

launched. On the first day, the attack was executed at about 13:20 according to the author's time, but in Splunk, for all the times, two hours were subtracted. On the second day, the attack was conducted shortly after 20:30, and on the last day, the attacks were executed separately at about 9:15, 10:20, 12:45, and 14:05.

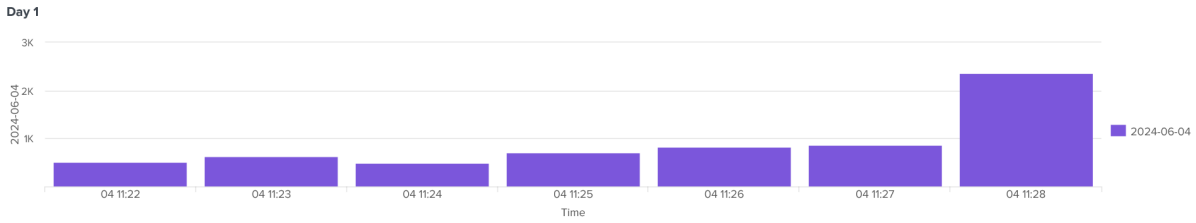


Figure 6.6: First day filtered for EventIDs noted in 5.4 and split into one minute buckets

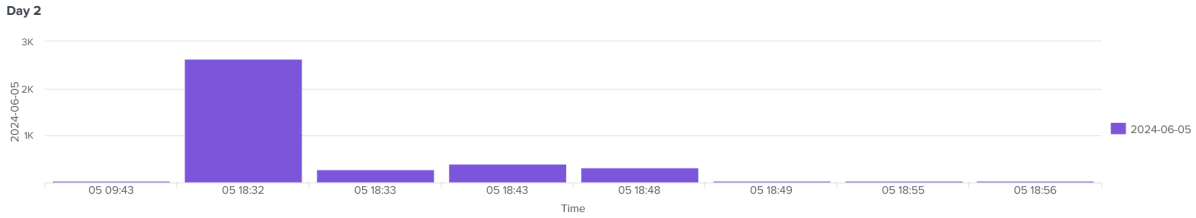


Figure 6.7: Second day filtered for EventIDs noted in 5.4 and split into one minute buckets

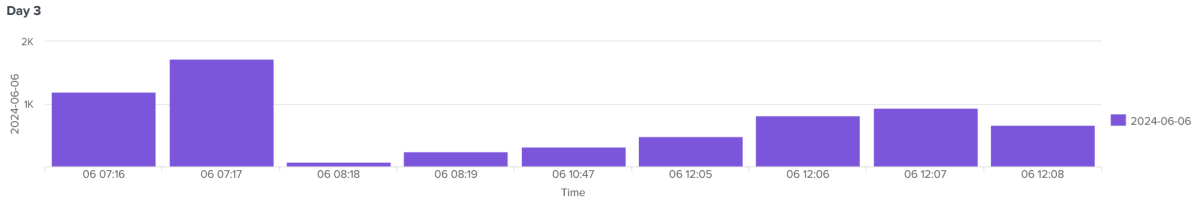


Figure 6.8: Third day filtered for EventIDs noted in 5.4 split into one minute buckets

The figures above show that except for the Brute Force attacks, no eventual malicious EventIDs were showing up on both the machines except for the wrong manual login on the second day at 9:43 as shown in Figure 6.7. This is the only event with a filtered "EventID'" not belonging to the attacks that can be detected (If during an attack, a user would log in unsuccessfully, this would not be recognized, on the contrary, it would count towards the attack). Figures 6.6, 6.7, and 6.8 visualize that these malicious events occurred within just a few minutes. The corresponding code can be found in Listings A.24, A.25, and A.26. Even though sometimes also many events happen within one second, none of these events are potentially dangerous on their own.
Figure B.7 combines the Figures 6.6, 6.7, and 6.8. The code to create this figure is shown in Listing A.27. Each color belongs to one day and the event counts are shown per minute in which one or more detected events happened.
In total, this approach returned the values as presented in Table 6.3, the results of this table were used for calculating the values for the traditional approach in Table 6.4. Table 6.17 presents the resulting evaluation.

Machine Learning Approach

For this approach, a Python script was created, running different algorithms to find the one with the best results for the three-day dataset. A Random Forest, a SVM, a Naive Bayes, a Logistic Regression, and a MLP classifier were run within similar Python scripts to get the algorithm with the best results. Also, an OCSVM was executed with a slightly different Python script.
All the ML models, except the OCSVM, were trained on 80% of the attack and normal event data, which equals 112'563 events. The other 20% (28141 events) were used to test the models. The OCSVM classifier was trained only on the normal data. Finally, all models were tested on the same dataset as the traditional approach, namely the dataset that was collected during a three-day time period. To decide whether an event within this three-day dataset belonged to an attack or not, which is used for later evaluation, if an event with ID 4625 follows an event with ID 4648 within 0.1 seconds, both events were labeled as attack, these were the only attack events within this dataset. There was one exception where these two events were caused by a wrong logon attempt of the user.
In the beginning, the Random Forest algorithm was used with different amounts of features. At the start, only two features were used, the "EventID" and the amount of events within one second. This already returned promising results. In the last step, five features were used. The features were the "EventID", the count of events within one second, the number of usernames within one second, the number of different usernames within one second, and the number of same usernames within one second. These five features were used for all the ML algorithms. The results of the correct and misclassified labels by the classifiers can be seen in the subfigures of Figures B.8 and B.9 and the TP, TN, FP, and FN can be seen in the subfigures of Figures B.10 and B.11. Table 6.4 shows the results of the evaluation of the traditional approach and the different ML approaches and Table 6.3 presents the TP, TN, FP, and FN that were used for the evaluation.
Figures B.8 and B.9 show the actual labels in blue and the labels predicted by the ML algorithm in red. It also shows the correctly classified labels in pink and the misclassified labels in black. The labels are either 0, for normal events, or 1, for attack events, but for visualization, they were labeled as "Attack" and "No Attack" in the graphs. The correctly and wrongly classified labels have a slight offset to see the difference between the labels better. In the plot, it can be seen when the attacks occurred, that is where the blue dots have the label "Attack".
The plot in Figures B.10 and B.11 visualizes the data points of Table 6.3. Blue dots again represent the actual labels and in red, the predicted labels are displayed. The TPs are indicated with pink crosses, the TN with orange crosses, the FP with grey crosses, and the TN with black crosses. It can be observed that the algorithms perform quite well and mostly predict the correct labels.
Table 6.4 summarizes all the evaluation results of the ML approaches and allows for an easy comparison with the traditional approach of the three-day dataset. Table 6.3 on the other hand, gives an overview of all the TP, TN, FP, and FN values across the different ML methods but also shows the values for the traditional approach for comparison.
Since for training 80% of the normal and attack data was used, with the remaining data, an evaluation was made as for the traditional approach. The results of the evaluation are shown in Table 6.2 and the counts are presented in Table 6.1. This equals to the traditional approach using the data except the three-day dataset.

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	TP	2616	2544	2564	2518	2535	13213	
	TN	25519	25506	24945	25507	24783	13033	
	FP	1	14	575	13	737	12432	
	FN	5	77	57	103	86	169	
Traditional	TP							3349
	TN							127295
	FP							27
	FN							3494

Table 6.1: Brute Force except for three-day dataset TP, TN, FP, and FN values, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Metric***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	Precision	1	0.99	0.82	0.99	0.77	0.52	
	Recall	1	0.97	0.98	0.96	0.97	0.99	
	Accuracy	1	1	0.98	1	0.97	0.68	
	F1-Score	1	0.98	0.89	0.98	0.86	0.68	
Traditional	Precision							0.99
	Recall							0.49
	Accuracy							0.97
	F1-Score							0.66

Table 6.2: Brute Force except three-day dataset evaluation results of all approaches, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	TP	32496	32783	33157	32604	32679	0	
	TN	43049	43043	40634	42889	41923	30216	
	FP	189	195	2606	349	1315	13022	
	FN	724	437	63	616	541	33220	
Traditional	TP							16615
	TN							43234
	FP							1
	FN							16608

Table 6.3: three-day dataset Brute Force ML TP, TN, FP, and FN values, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Metric***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	Precision	0.99	0.99	0.93	0.99	0.96	0	
	Recall	0.98	0.99	1	0.98	0.98	0	
	Accuracy	0.99	0.99	0.97	0.99	0.98	0.4	
	F1-Score	0.99	0.99	0.96	0.99	0.97	0	
Traditional	Precision							1
	Recall							0.5
	Accuracy							0.78
	F1-Score							0.67

Table 6.4: three-day dataset Brute Force evaluation results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

The tables above show that the MLP is the best model for the Brute Force attack, the Random Forest and the Logistic Regression showed similar performance, closely followed by the Naive Bayes and the SVM classifiers. The OCSVM shows worse results. The traditional approach performs better than the OCSVM but worse than all the other ML classifiers.

Lastly, the training and testing times were measured to get the Runtime Performance. The results are shown in Table 6.5 for the data except for the three-day dataset, including the test time, and in Table 6.6 the Runtime Performance of the different ML models is shown if the three-day dataset was used.

Classifier	Training	Testing Total	Testing Average
Random Forest	6.84	0.29	0.000010
MLP	180.54	0.09	0.000003
SVM	148.53	65.92	0.002343
Logistic Regression	3.11	0.00	0.000000
Naive Bayes	0.03	0.25	0.000009
OCSVM	1458.72	359.34	0.009250

Table 6.5: Runtime Performance of the ML classifiers in seconds for training and testing of the Brute Force attack (not including the three-day dataset)

Classifier	Testing Total	Testing Average
Random Forest	1.57	0.000021
MLP	0.29	0.000004
SVM	122.83	0.001606
Logistic Regression	0.04	0.000000
Naive Bayes	0.01	0.000000
OCSVM	849.12	0.011106

Table 6.6: Runtime Performance of the ML classifiers in seconds for testing the Brute Force attack of the three-day dataset

Clearly, the OCSVM takes by far the longest and the Naive Bayes is the fastest variant for training the model. For testing, the OCSVM is again by far the slowest for both datasets and the Logistic Regression the fastest for the first case not including the three-day dataset but for the three-day dataset, the Naive Bayes is slightly quicker than the Logistic Regression. However, Table 6.5 allows for a better comparison with more similar amounts of data that needed to be predicted.

6.1.2 Bypass User Account Control

Traditional Approach

The Bypass User Account Control is a sub-technique that is part of the Abuse Elevation Control Mechanism technique. This technique is used for two tactics, the Privilege Escalation and the Defense Evasion [12].

Before the analysis of the attack, the events gathered from [62] were preprocessed and subsequently uploaded to Splunk into the "buac" index. After the upload was completed, fields were extracted. This index included 709 attack events. These events were analyzed in a separate dashboard created within the Splunk environment.
In the beginning, the EventIDs of the attack events were analyzed, which is visualized in Figure 6.9 below (the corresponding code is visible in Listing A.28):

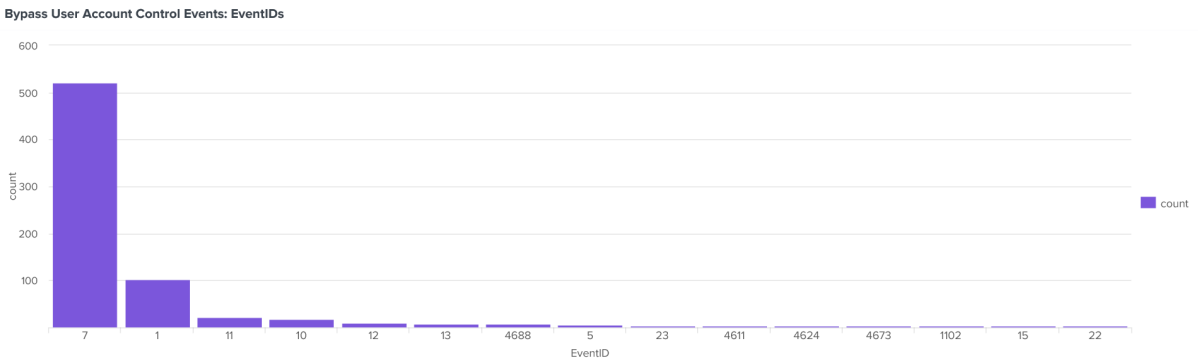


Figure 6.9: EventIDs of the Bypass User Account Control attack events

For comparison, the top five "EventID" values of the normal events are shown in Figure 6.2. The SubjectDomainName was a field investigated shortly, which is shown in Figure B.12, and the code is shown in Listing A.33. There were three values for the attack events, but the value with the highest count (ten) was also showing up frequently in the normal data. The other two attack values were also quite rare and only showed up once and three times.
It was noted that there were two different types of ProcessIDs: "ProcessID" and "ProcessId". Therefore, these two fields were explored whether they may provide value. Figures were created to show the different types of process IDs for the normal and attack data. Figures B.13 and B.14 show the values for "ProcessID" and Figures B.15 and B.16 show the values of the "ProcessId" field.
This showed that the "ProcessID" seemed less random compared to the "ProcessId" field. However, as noted in [22], each process is assigned a "ProcessID", which is a unique decimal number. Therefore, both types of ProcessIDs were not chosen as features.
Normal events never included the feature "ParentProcessID", whereas 102 attack events included this field. However, according to [53], a ParentProcessID is also a ProcessID that started a process. Therefore, this was also not taken as a feature of the ML approach. Similarly, the "CommandLine" field only occurred once in the normal events, whereas in the attack data, it showed up in 110 events. Therefore, this was also taken as a feature.
During the analysis of these attack events, also the fields "AuthentificationPackageName", "Product", "ThreadID", "Task", and "Description" however, they did not seem to be promising if used as features.
The last field being explored was the "Image" field. Normal events did not include any "Image" field, only four events included the word "Image" in their event. In contrast, many of the attack events, in total 676, included this field, therefore this field was taken as a feature, too.

Now that the Bypass User Account Control attack was understood, the SPL filtering for as many attacks and as normal events as possible could be created.
Listing 5.10 starts by defining the events taken into consideration by the two indices used on the first line. It then filters for one EventID (4688). If this EventID occurs, this means that a new process has been created [59]. Moreover, it was checked whether the "CommandLine" or the "Image" fields were empty since only one normal event included a "CommandLine" value and none included the "Image" field. Additionally, the "Product" field was checked whether it is equal to "-", "?" or "UACMe" since this also only showed up in the attack events. Similarly, the "Description" was examined whether it had the value "?" or if it included ".exe". Furthermore, the "_raw" was checked whether certain keywords were included in the event that only were within some of the attack events.

Table 6.8 shows the counts of the TP, TN, FP, and FN values as received by Listing 5.10. These values were used for the evaluation in Table 6.17.

ML Approach

For the Bypass User Account Control attack, the Python script used for the Brute Force attack was slightly adjusted to fit the new attack. In total, there were 128'031 events, of these 127'322 were normal, and 709 attack events.
The same classifiers were run as already done for the Brute Force attack with the best performing algorithm, the Random Forest, as can be seen in Tables 6.8 and 6.7. The models were trained on 80% of the attack and normal data and the remaining 20% were used to evaluate how well the classifiers performed. The only exception is the OCSVM, which was only trained on 80% of the normal data and was then evaluated on all attack events and the remaining 20% of the normal data. For this attack scenario, first, an approach with seven features was used, where the Random Forest already performed very well. The features used were "EventID", "SubjectDomainName", "ProcessId", "ProcessID", "ParentProcessID", "ParentCommandLine", and "CommandLine".
In the second step, two features were excluded, namely the "SubjectDomainName" and the "ProcessId". With only five features, the Random Forest had an improved performance.
After that, also the "ProcessID" and the "ParentProcessID" were excluded but for this, the Random Forest and the Logistic Regression classifier had slightly worse predictions but the Naive Bayes and the MLP had an increased prediction power. The SVM stayed the same.
In the final step, the feature "Image" was added, improving the performance of the Random Forest and the MLP also had increased values even though it now had more FPs. All other classifiers performed equally well.
The results of the correct and misclassified labels by the classifiers can be seen in the subfigures of Figures B.17 and B.18 and the TP, TN, FP, and FN can be seen in the subfigures of Figures B.19 and B.20. Table 6.8 shows the results of the evaluation of the traditional approach and the different ML approaches and Table 6.7 presents the TP, TN, FP, and FN values that were used for the evaluation.
As for the Brute Force attack figures, Figures B.17 and B.18 show the actual labels in blue, and the labels predicted by the ML algorithm in red. Moreover, they present the correctly classified labels in pink and the misclassified labels in black. The labels are

either 0, for normal events, or 1, for attack events, but for visualization, they were labeled as “Attack” and “No Attack” in the graphs. The correctly and wrongly classified labels have a slight offset to see the difference between the labels better. In the plots, it can be seen when the attack events took place on the timeline, that is where the blue dots have the label “Attack”. Since these attacks belong to a GitHub repository [62], they took place in 2019 and 2021, therefore, all these attack events are at the start of the timeline and the normal events start in 2023 until 2024.
The graphs in Figures B.19 and B.20 visualize the data points of Table 6.7. Blue dots again represent actual labels and the red ones are the predicted labels. The TPs are indicated as pink crosses, the TNs with orange crosses, the FPs with grey crosses, and the FNs with black crosses.

Since there were only 709 attack events, it has a higher impact if one event more is counted towards another value of the TP, TN, FP, and FN and one is deducted from another value compared to the previous Brute Force attack. For training 25’607 and for testing 102’424 events were used this equals an 80% training and 20% testing data division except for the OCSVM, where 80% of the normal data was used, and for training the remaining 20% of this data and the attack data was used. The OCSVM therefore had more attack events since it was trained only on the normal events and none of the attack events were used for training, this means that the whole set of attack events could be used for evaluation (in total there were 709 attack events).

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	TP	148	146	0	146	146	15	
	TN	25458	25459	25459	25459	25458	20791	
	FP	1	0	0	0	1	4674	
	FN	0	2	148	2	2	694	
Traditional	TP							697
	TN							127321
	FP							1
	FN							12

Table 6.7: Bypass User Account Control results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Metric***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	Precision	0.99	1	0	1	0.99	0	
	Recall	1	0.99	0	0.99	0.99	0.02	
	Accuracy	1	1	0.99	1	1	0.79	
	F1-Score	1	0.99	0	0.99	0.99	0.01	
Traditional	Precision							1
	Recall							0.98
	Accuracy							1
	F1-Score							0.99

Table 6.8: Bypass User Account Control evaluation results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

The presented tables above reveal that the Random Forest has the best performance, it only wrongly predicted one event. The next best models were the MLP and the Logistic Regression, both misclassified two events, closely followed by the Naive Bayes, which misclassified three events. The traditional approach is slightly worse than the Naive Bayes. Of the ML approaches, the SVM and the OCSVM were much worse.

Finally, for this attack, the training and testing times were also measured to determine the Runtime Performance. The results are shown in Table 6.9.

Classifier	Training	Testing Total	Testing Average
Random Forest	5.3	0.21	0.000008
MLP	20.73	0.04	0.000001
SVM	12.06	120.38	0.004701
Logistic Regression	3.19	0.01	0.000000
Naive Bayes	0.07	0.01	0.000000
OCSVM	1220.86	334.9	0.012795

Table 6.9: Runtime Performance of the ML classifiers in seconds for the Bypass User Account Control attack

This table shows that the OCSVM is the slowest and the Naive Bayes is the fastest for training the models. OCSVM takes the longest time for testing, and the Logistic Regression is the fastest in this case, although the Naive Bayes is also very fast. The MLP is also very fast during testing.

6.1.3 Remote Services

Traditional Approach

For this technique, which is part of the Lateral Movement tactic, the index "remote_services" was created where attack data of [62] and [44] were uploaded after preprocessing it. In total, this index included 602 attack events. After uploading the data, the fields were extracted, and the data was analyzed in a new dashboard in Splunk to get a better understanding of the attack patterns.
As already done for the Bypass User Account Control attack, first, the EventIDs were analyzed, which is shown in Figure 6.10 and the corresponding code in Listing A.34. The top five EventIDs of the normal events are shown in Figure 6.2. However, for a better comparison, it was checked whether the EventIDs found in this attack data were also present in the normal dataset. The resulting plot is shown in Figure 6.11 and the corresponding code in Listing A.35.

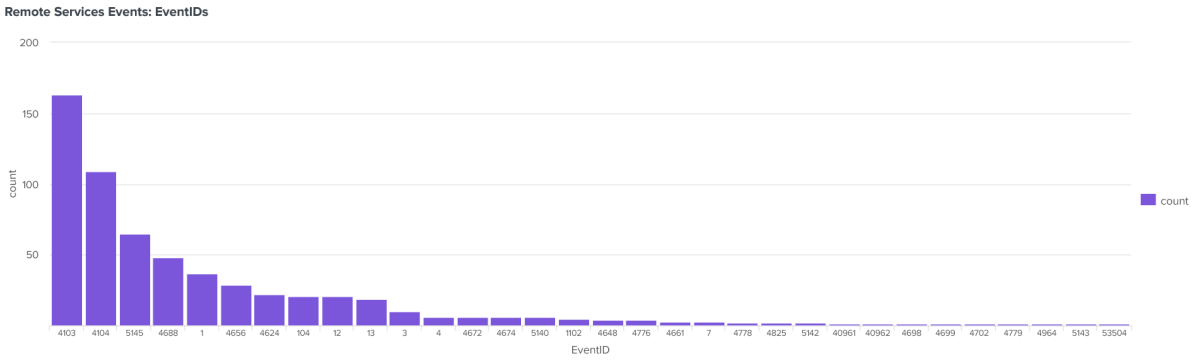


Figure 6.10: EventID values of the attack events

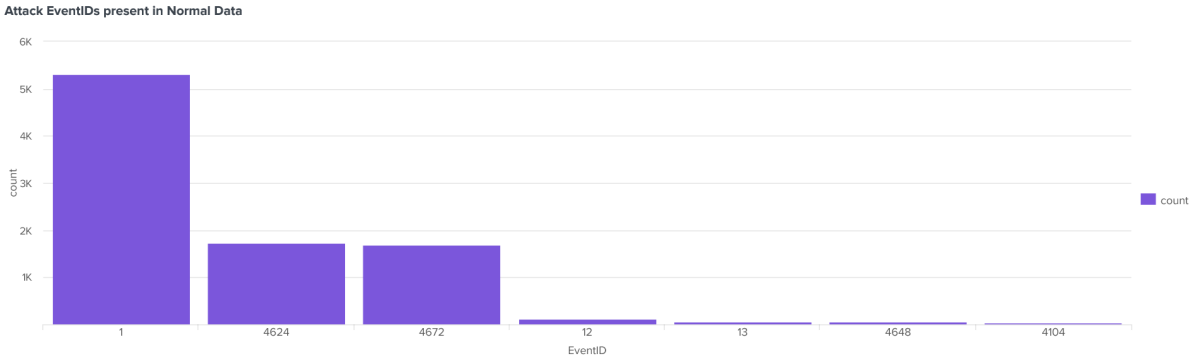


Figure 6.11: EventID values of attack events also present in the normal dataset

This shows that only seven EventIDs of the attack data were also present in the normal dataset. EventID 4103 is not within the normal dataset (this EventID occurs if the computer has corrupted modules, insufficient permissions, or execution policy restrictions [8]). Events with ID 5145 are also not present in the normal data. This ID indicates that a network share object has been checked whether a client can be granted the desired access. Moreover, the EventID 4688 with the fourth highest count is also not present in the normal events at all. This EventID implies that a new process was created [59]. However, there are more EventIDs not showing up in the normal event data, but of the top five, the three mentioned are not present in the normal data. The fifth ID, 1, is present in the normal events with over 5'000 occurrences and the second highest count of the attack data (EventID 4104) only shows up six times in the normal data. By only using these three EventIDs, it was possible to already filter out 276 of 602 attack events.
Besides, it was looked for some keywords only showing up in the attack data. Only two keywords could be defined, which are "anonym" and "offsec" that showed up on their own and indicated attack events. However, the former keyword only filtered out one more attack event. The latter was present in 383 events of the attack data. Other keywords like "remote" and "unknown" were checked, but both keywords returned mostly normal events.
As another feature, the "CommandLine" field was taken into consideration as well. Since we know from the Bypass User Account Control attack that the normal dataset only includes one event with a "CommandLine" field, this field seemed also to be of interest.

Filtering out this field returned 85 attack events and one normal event. However, 48 of these events had "EventID" 4688, and were already filtered out. The remaining 37 events had a "Channel" value different from those of the normal events, which only included the three "Channel" values "Application", "Security", and "System". These 37 attack events included the value "Microsoft-Windows-Sysmon/Operational". Therefore, the "Channel" field could also serve to filter out some attack events.
Next, the keyword "workstation" and the field "Workstation" were taken into consideration, however, there were few events including these. Filtering for events with the word "workstation" and additionally not including the field "Workstation" returned three more events. Besides, the field "WorkstationName" had the value "-" within both, normal and attack events, and besides, in the normal events, it only had the computer names of the author as other values, whereas the attack data included only three different values, each present in only one event ("0426W-Win10", "NULL", and "PC01").
Then, it was noted that for all the normal events, the names of the author's computers showed up, whereas for all attack events, these were other names. With this whitelisting approach, all events were classified correctly. It would be sufficient to only take the field "Computer" into consideration for the Remote Services attack. For previous attacks, this would also return all events correctly. However, the attack logs always would be recorded from the computer creating the logs. Therefore, this approach is not meaningful consequently, this feature was not considered. Other fields that would be suitable for a whitelisting approach are "Channel" and "Username".
It was also noted that if the "UserData" field included the value "SubjectUserName", the event was part of an attack. The same was observable for the field "ProcessName" and the value "-". Lastly, if the two words "AccessList" and "PrivilegeList" showed up in the same event, this indicated an attack event, too.
All these facts lead to the filtering query shown in Listing 5.12 and finally, the SPL query for the evaluation of the traditional approach could be designed, which is shown in Listing 5.13. The former listing starts by defining the indices used on the first line. It then filters for three EventIDs (4103, 4688, and 5145). Moreover, it was checked for two keywords, "anonym" and "offsec" and whether the field "Channel" was equal to a value used in the normal data. Additionally, it was filtered for events having no "Workstation" field and at the same time including the word "Workstation". The "UserData" field and the "ProcessName" fields were checked for these specific values and finally, the event was checked whether it included both words, "AccessList" and "PrivilegeList", at the same time. If one of these statements were true, suspicious was set to "true".

Table 6.10 presents the counts of the TP, TN, FP, and FN values as received by Listing 5.12. These values were used for the evaluation in Table 6.11.

ML Approach

The ML approach of the Remote Services attack used similar Python scripts as before, smaller changes were made to fit the new attack. In total, there were 127'924 events, of which 602 belonged to the attack data.
As already done for the Brute Force and Bypass User Account Control attacks, the same six classifiers were run in the Python scripts. The best performing model was, as for

the previous Bypass User Account Control attack, the Random Forest, which predicted everything correctly. The performance of the different classifiers can be seen in Tables 6.11 and 6.10. 80% of the attack and normal data was used to train the models and the remaining data was used for evaluating how well the model could predict. The OCSVM used a slightly different approach, as already mentioned in the previous attacks.
As features, "EventID", "Offsec", "Channel", and "ProcessName" were used. The "EventID" was already used in the previous attacks. The "Offsec" feature had either the value 0 if this keyword did not show up within the event or 1 if it was present. "Channel" includes the channel the event belonged to and "ProcessName" tells the name of the process.

The results of the correct and misclassified labels by the classifiers can be seen in the subfigures of Figures B.21 and B.22 and the TP, TN, FP, and FN can be seen in the subfigures of Figures B.23 and B.24. Table 6.10 presents the TP, TN, FP, and FN values that were used for the evaluation and Table 6.11 shows the results of the evaluation of the traditional approach and the different ML approaches.
As for the previous attacks, Figures B.21 and B.22 show the actual labels in blue and the labels predicted by the ML algorithm in red. Moreover, they present the correctly classified labels in pink and the misclassified labels in black. The labels are either 0, for normal events, or 1, for attack events, but for visualization, they were labeled as "Attack" and "No Attack" in the graphs. The correctly and wrongly classified labels have a slight offset to see the difference between the labels better. In the plots, it can be seen when the attack events took place on the timeline, that is where the blue dots have the label "Attack". Since these attacks belong to GitHub repositories [44] and [62], they took place from 2019 until 2021. The normal events start in 2023 until 2024.
The graphs in Figures B.23 and B.24 visualize the data points of Table 6.10. Blue dots again represent actual labels and the red ones the predicted labels. The TPs are indicated as pink crosses, the TNs with orange crosses, the FPs with grey crosses, and the FNs with black crosses.

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	TP	117	26	0	77	77	482	
	TN	25468	25468	25468	25468	25468	20791	
	FP	0	0	0	0	0	4674	
	FN	0	91	117	40	40	120	
Traditional	TP							586
	TN							127321
	FP							1
	FN							16

Table 6.10: Summary of Remote Services ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	Precision	1	1	0	1	1	0.09	
	Recall	1	0.22	0	0.66	0.66	0.8	
	Accuracy	1	1	1	1	1	0.82	
	F1-Score	1	0.36	0	0.79	79	0.17	
Traditional	Precision							1
	Recall							0.97
	Accuracy							1
	F1-Score							0.99

Table 6.11: Summary of Remote Services ML evaluation results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

The tables above revealed that the Random Forest classifier performed the best. It predicted everything correctly. The traditional approach also worked quite well. The Logistic Regression and the Naive Bayes predicted equally well and also the amount of true and wrong classified events was exactly the same. The SVM and OCSVM did a poor job predicting the events, also the MLP was not very good.

Finally, for this attack, the training and testing times were also measured to determine the Runtime Performance. The results are shown in Table 6.12.

Classifier	Training	Testing Total	Testing Average
Random Forest	4.24	0.39	0.000015
MLP	31.76	0.04	0.000002
SVM	16.06	6.82	0.000267
Logistic Regression	1.61	0.01	0.000000
Naive Bayes	0.04	0.03	0.000001
OCSVM	1524.98	314.24.86	0.012055

Table 6.12: Runtime Performance of the ML classifiers in seconds for the Remote Services attack

As for the two previous attacks, Naive Bayes is the fastest and OCSVM is the slowest for training time. When looking at the testing time, in this case, the Logistic Regression is the fastest, however, the MLP and the Naive Bayes are also quick for testing. The slowest classifier for testing is again the OCSVM.

6.1.4 Disable Windows Event Logging

Traditional Approach

Disable Windows Event Logging is part of the Impair Defense technique, belonging to the Defense Evasion tactic [12].
The data was first preprocessed and then uploaded to Splunk after creating a new index, "dwel", including 115 attack events. Fields were then extracted to analyze the data in a new dashboard in Splunk to get a better understanding of it.
In the beginning, the EventIDs were examined, shown in Figure 6.12 are the events that were present in the attack data (for code see Listing A.36) and Figure 6.13 presents the EventIDs occurring in both, the attack and the normal data (code shown in Listing A.37).

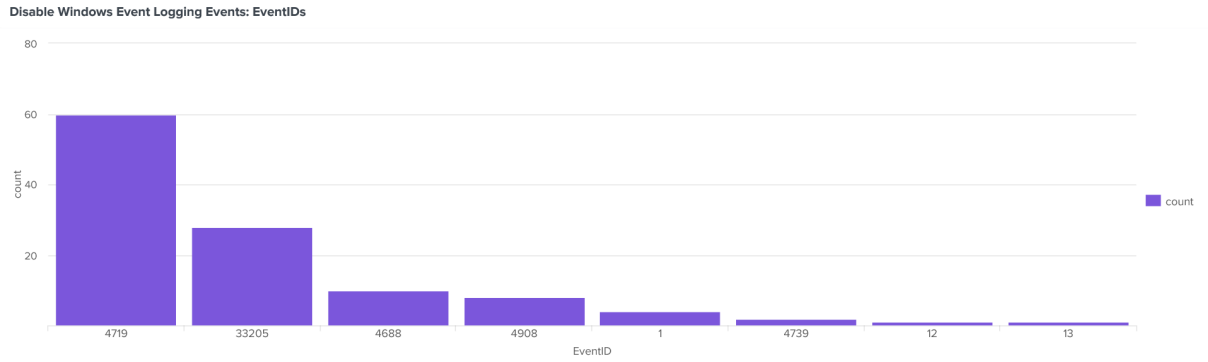


Figure 6.12: EventID values of the attack events

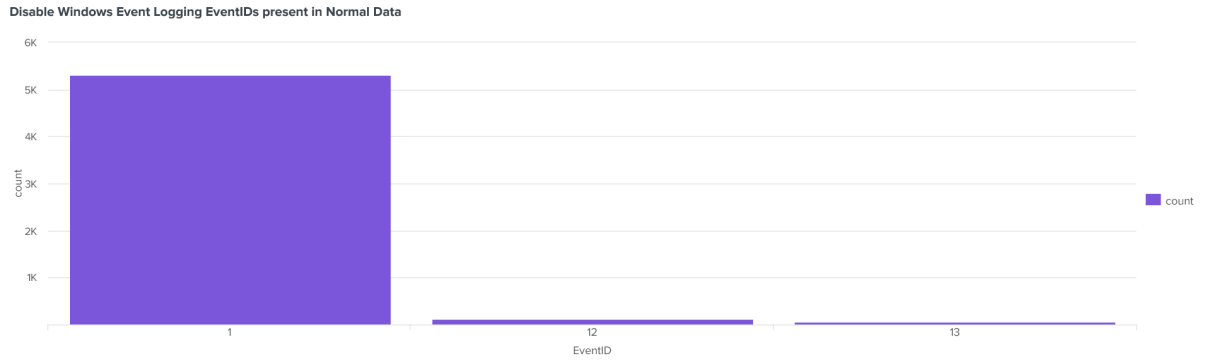


Figure 6.13: EventID values of the attack events also showing up in the normal data

These two figures above show that only the EventIDs 1, 12, and 13 were also present in the normal data. The EventIDs 4719, 4688, and 4908 showed up more than five times in the data, did not show up in the normal events, and may imply malicious behaviour. Therefore, they were used for the query shown in Listing 5.14 for filtering out malign events. EventID 4688 was already described in the Remote Services attack, events with ID 4908 are titled “Special Groups Logon table modified” [59], and the EventID 4719 implies that the system audit policy had been changed [59].
Moreover, it was noted that there were some suspicious-looking keywords within the attack data. For that reason, six keywords were filtered, as shown in the figure below. The code creating the figure is shown in Listing A.38. It can be observed that only the “Disable” keyword shows up in the normal events, all others only were present within the attack data.

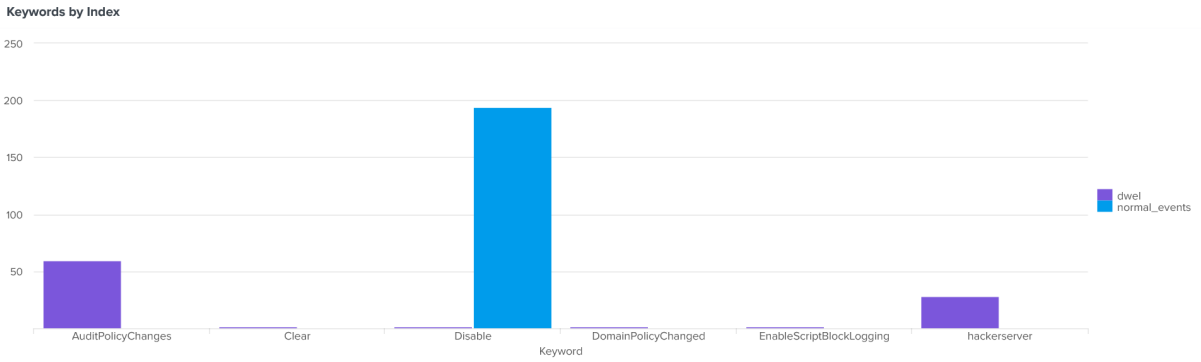


Figure 6.14: Keywords in attack and normal events

After defining some suspicious keywords, for this attack, which were "AuditPolicyChanges", "Clear", "DomainPolicyChanged", "EnableScriptBlockLogging", and "hackerserver", the "Channel" field was examined. Since for the normal data, only the three Channels "Application", "Security", and "System" were present (see Figure 6.15) and in the attack data, there was a different channel instead of "System", shown in Figure 6.16. For the code of the two Figures 6.15 and 6.16, see Listing A.39 and A.40, respectively.

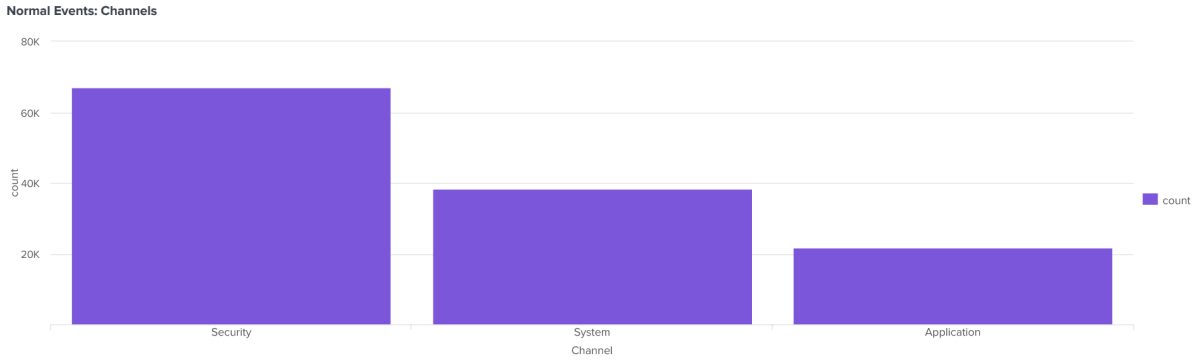


Figure 6.15: Channel values of normal data

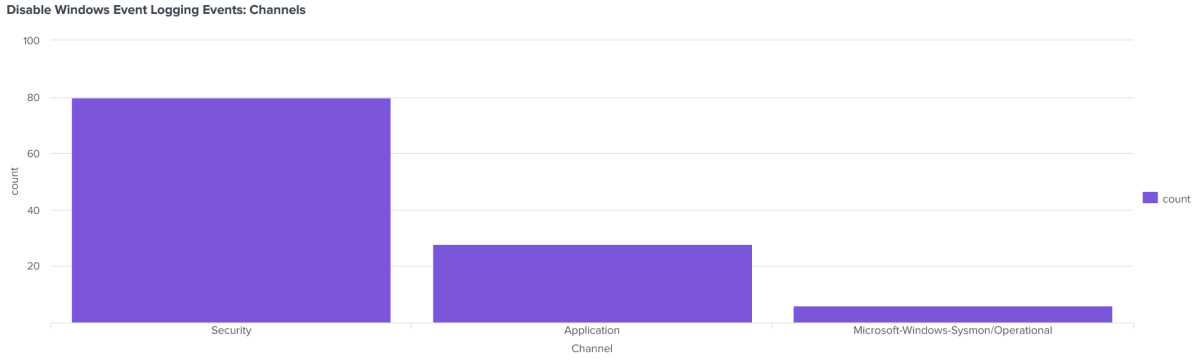


Figure 6.16: Channel values of attack data

Finally, the "CommandLine" field was inspected. As already known through the Bypass User Account Control attack, the normal events included only one event having this field.

However, there were 14 attack events, including this field. Figure 6.17 represents these findings and Listing A.41 provides the code for creating the plot in the figure.

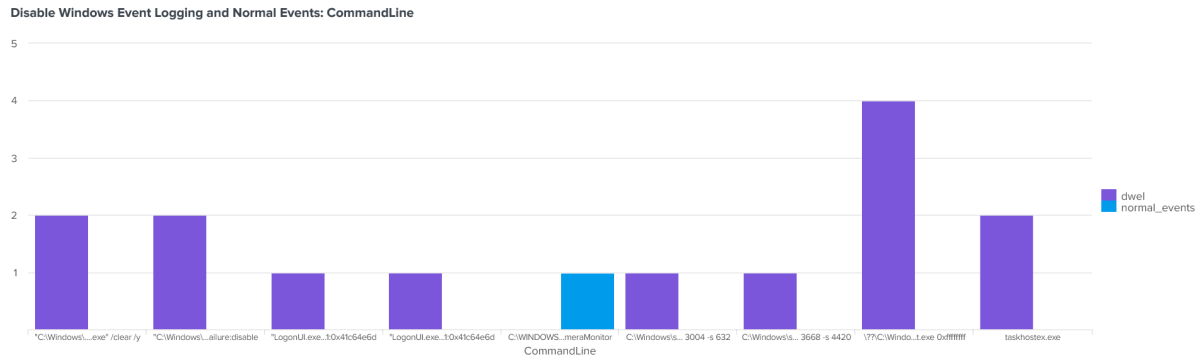


Figure 6.17: "CommandLine" values of attack and data

All these indices lead to the filtering query in Listing 5.14 and the same filtering approach was used in Listing 5.15 for the evaluation of the traditional approach. The filtering parts of these listings start by defining the two indices for the attack and normal event data. It then filters for three "EventID" values (4688, 4719, 4908) and checks whether the "CommandLine" field is not empty. Moreover, it looks for a value in the "Channel" field different from the ones found in the normal data ("Application", "Security", and "System"), and lastly, the filtering checks whether the keywords "hackerserver", "DomainPolicyChanged", "AuditPolicyChanges", "EnableScriptBlockLogging", and "clear" are found within an event.

Table 6.13 presents the counts of the TP, TN, FP, and FN values as received by Listing 5.14. These values were used for the evaluation in Table 6.14.

ML Approach

As for the previous attacks, the Disable Windows Event Logging uses a similar Python script for the ML approaches as before. In total, there were 127'436 events, of which 114 were attack events.
Six different classifiers were run in different Python scripts, namely the Random Forest, the MLP, the SVM, the Logistic Regression, the Naive Bayes, and the OCSVM. The best model was the Random Forest, only misclassifying one attack event, as already for the Bypass User Account Control and Remote Services attack scenarios. 80% of the attack and normal data were used to train the models except for the OCSVM. The remaining 20% of the data was used for the evaluation, the performance of the classifiers can be seen in Tables 6.14 and 6.13. The OCSVM used a slightly different approach, as already mentioned in the Bypass User Account Control attack.
The "EventID", the "CommandLine", and the "Channel" were used as features, where these three belong to fields within the events.

The results of the correct and misclassified labels by the classifiers can be seen in the subfigures of Figures B.25 and B.26 and the TPs, TNs, FPs, and FNs can be seen in the

subfigures of Figures B.27 and B.28. Like for the previous attacks, the figures show the actual labels in blue and the labels predicted by the ML algorithms in red. Moreover, Figures B.25 and B.26 present the correctly classified labels in pink and the misclassified labels in black. The labels are either 0, for normal events, or 1, for attack events, but for visualization, they were labeled as "Attack" and "No Attack" in the graphs. The correctly and wrongly classified labels have a slight offset to see the difference between the labels better. In the plots, it can be seen when the attack events took place on the timeline, that is where the blue dots have the label "Attack". Since these attacks belong to GitHub repositories [44] and [62], they took place from 2019 until 2021. The normal events start in 2023 until 2024. The graphs in Figures B.27 and B.28 visualize the data points of Table 6.13. Blue dots again represent actual labels and the red ones the predicted labels. The TPs are indicated as pink crosses, the TNs with orange crosses, the FPs with grey crosses, and the FNs with black crosses.

Table 6.14 shows the results of the evaluation of the traditional approach and the different ML approaches and Table 6.13 presents the TP, TN, FP, and FN values that were used for the evaluation.

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	TP	14	0	5	0	0	80	
	TN	25473	25473	25473	25473	25473	9751	
	FP	0	0	0	0	0	15714	
	FN	1	15	10	15	15	34	
Traditional	TP							114
	TN							127321
	FP							1
	FN							0

Table 6.13: Summary of Disable Windows Event Logging ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

Approach	***Value***	***Classifier***						
		RF	***MLP***	***SVM***	***LR***	***NB***	***OCSVM***	***Trad***
ML	Precision	1	0	1	0	0	0.01	
	Recall	0.93	0	0.33	0	0	0.7	
	Accuracy	1	1	1	1	1	0.38	
	F1-Score	0.97	0	0.5	0	0	0.01	
Traditional	Precision							0.99
	Recall							1
	Accuracy							1
	F1-Score							1

Table 6.14: Summary of Disable Windows Event Logging ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional

The tables above show that the traditional approach performs the best. The best ML classifier is again the Random Forest, the performance of the other classifiers is not good.

Finally, the training and testing times were measured for this attack to determine the Runtime Performance. The results are shown in Table 6.15.

Classifier	Training	Testing Total	Testing Average
Random Forest	8.72	0.35	0.000014
MLP	77.89	0.05	0.000002
SVM	4.62	2.45	0.000096
Logistic Regression	12.69	0.02	0.000001
Naive Bayes	0.02	0.09	0.000004
OCSVM	1110	215.01	0.008406

Table 6.15: Runtime Performance of the ML classifiers in seconds for the Disable Windows Event Logging attack

As for the two previous attacks, Naive Bayes is the fastest and OCSVM is the slowest for training time. When looking at the testing time, the Logistic Regression is the fastest, however, the MLP and the Naive Bayes are also quick for testing.

6.1.5 Revision of the One-Class Support Vector Machine classifier

Since this classifier performed poorly for all the attacks with the default settings, some more variants were examined. The different variants are shown in Table 6.16, the results are also presented in this table. The three-day dataset was used for evaluating the Brute Force attack. The default values of the OneClassSVM function are as follows according to [21]: kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, nu=0.5, shrinking=True, cache_size=200, verbose=False, max_iter=-1.

Attack	Kernel	Gamma	Nu	Precision	Recall	Accuracy	F1-Score
Brute Force	rbf	scale	0.1	0	0	0.56	0
	rbf	auto	0.1	0.64	1	0.75	0.78
	linear	-	0.1	0	0	0.55	0
Bypass User	rbf	scale	0.1	0.03	0.86	0.11	0.05
Account Control	rbf	auto	0	0	0.31	0	0
	linear	-	0.1	0	0.03	0.11	0
Remote Services	rbf	scale	0.1	0.03	1	0.15	0.05
	rbf	auto	0.1	0	0	0.74	0
	linear	-	0.1	0.02	0.84	0.13	0.04
Disable Windows	rbf	scale	0.1	0	0.75	0.11	0.01
Event Logging	rbf	auto	0.1	0	0.04	0.26	0
	linear	-	0.1	0	0.95	0.12	0.01

Table 6.16: OCSVM variants and their results

The table above reveals that, even with different parameter values, this classifier performs poorly compared to the other ML models.

6.2 Evaluation

Brute Force Attack

For the evaluation of the Brute Force events, the three-day period was taken for the model evaluation. First, the traditional approach is evaluated. The FNs were calculated in the query 5.8. Figure 6.4 presents all the EventIDs that showed up during the attacks that took place. It was visualized that the only "EventID" values that count toward the attacks were 4625 and 4648. The rest of the EventIDs were normal events that were logged while the attack took place. Therefore, it is obvious that all the 4625 events are TPs, and all 4648 events are FNs. There was one unsuccessful login during the attacks, therefore one event does not count towards the TPs. Instead, it belongs to the FPs (event with ID 4625). The wrong login caused by the user had no co-occurring 4648 event.
The FNs included all events that were not recognized by the query but would have belonged to the attacks (these are the 4648 Events). To define which 4648 events also should belong to the attack, a simple pattern was used. It was observed that always a 4648 event appears before a 4625 event. And this happens within 0.02 seconds. Therefore, the time constraint was added to avoid having a 4648 event not actually belonging to the 4625 event. Moreover, the 4625 event should be the next event and no other event should be in between these two events, such that the 4648 event counts towards the attack events. Finally, the amount of FNs had to be deducted from the total amount of negatives to get the TNs.
All these calculations are shown in Listing 5.9, which also displays the main query for the calculation of the Precision, Recall, and F1-Score. The results of the query are shown in Table 6.17 in the Brute Force row.

The evaluation of all used ML approaches is shown in Table 6.4. All classifiers have values over 0.90 except the OCSVM. The best-performing algorithm is the MLP, having 0.99 for all the evaluation metrics. Close to the leader are the Random Forest and the Logistic Regression with three values of 0.99 and one of 0.98 for the Recall rate. The Naive Bayes classifier takes the fourth position with two values of 0.98, one of 0.97, and one of 0.96. The second worst performing one is the SVM with a Recall value of one, however, Precision shows a value of 0.93, and Accuracy and F1-Score are 0.97 and 0.96 respectively. The OCSVM performed very poorly compared to the other classifiers. It could not detect any attack events, leading the values of Precision, Recall, and F1-Score to be zero. The Accuracy also only reached 0.4.
For the Brute Force attack, the ML approach returns a better value for Recall, Accuracy, and F1-Score. The traditional approach only exceeds the ML model in Precision by less than 0.01. The reason for the high Precision and Accuracy values for the ML models except the OCSVM are caused by the relatively high number of TPs (for all approaches over 32'000, except for the OCSVM) and quite low FP values except for the SVM classifier. This raises the value of the Precision value. For the Accuracy, especially the high TN

(over 40'000 for all classifiers except the OCSVM) and relatively low FN values (all are below 800, except for the OCSVM model) lead to such high values.
The improved performance of most of the ML model is caused by the high amount of FNs in the traditional model. The accuracy in the ML approach is better since this approach has almost double the amount of TPs. For Recall, Accuracy, and F1-Score, most ML approaches were better because the traditional approach has many FNs and this value is used in the division part of the formulas (see Equations 4.2, 4.3, and 4.4) thus returning smaller values compared to the ML methods.

The TP, TN, FP, and FN are displayed in Table 6.3, this shows that the OCSVM classifier performs the worst with over 20'000 wrongly classified events. This algorithm has the highest amount of FPs but it has the fewest FNs (zero). However, this amount of wrongly classified labels is close to the traditional approach, with 16'609 misclassified events. ML approaches, except the OCSVM, have double the amount of TP values compared to the traditional approach. The reason the traditional approach has half the amount of TPs lies in the fact that it has a lot of FNs that are the events with ID 4648 that co-occur with events with ID 4625. The best performing classifier is the MLP with 632 wrongly classified labels. However, the MLP still has over 400 FNs, which may be problematic for a real-world scenario because this would mean that these events were not labeled as an attack, even though they belong to the attack. FPs are less severe if they are not too high because this means that it is labeled as an attack but they are normal events. These events would need to be checked whether they are actually an attack. Therefore, if the value is too high, the effort of looking at all the events is very high.
To conclude, the MLP classifier performs very well on the given dataset with few FPs and FNs and does not return a too high effort to check values that are not attack events (FPs), therefore showing a good balance. Given the Brute Force attack, it is not severe, if a few attack events are not detected (FNs), since there are many more and it would still be recognizable that an attack is ongoing.

6.2.1 Bypass User Account Control

Also, for the Bypass User Account Control attack, a query was created in Splunk to calculate the evaluation with the TP, TN, FP, and FN. This query is shown in Listing 5.11. On line three, the TP, TN, FP, and FN are defined and on lines seven to eleven, the calculation for the Precision, Recall, Accuracy, and F1-Score is made.
The only FP, a normal event that had a "CommandLine" value, was counted towards the attack, since the query filters for this keyword within the "CommandLine" field. So generally, the filtering through keywords or checking for specific values in certain fields worked very well because, in the end, only one FP and seven FNs were counted in the traditional approach, therefore, the evaluation returned very high values. The remaining twelve FNs did not show a common pattern.
Of the ML methods, the Random Forest, the MLP, the Logistic Regression, and the Naive Bayes were even better than the traditional approach. The Random Forest only misclassified one event as a FP. The resulting evaluation of all approaches is shown in Table 6.8. Overall, the Random Forest can be considered the best approach. The table also reveals that all classifiers except for the SVM and OCSVM had very good values.

Overall, for the Bypass User Account Control, the Random Forest, and therefore again a ML model was the best, with the traditional approach being on fifth place according to the count of misclassified events and if the sum of the evaluation results is taken into consideration, the traditional approach shares the fourth place with the Naive Bayes (both summing up to 3.97 out of 4).

6.2.2 Remote Services

The query of the Remote Services attack, shown in Listing 5.13, returns one FP and 16 FNs. The FP was again caused by the use of the "CommandLine" filtering. The remaining 16 events did not show a common pattern. There were no more rules created to avoid too specific filtering for this dataset. The only "Channel" value that was also used in attack data was "Security". The "Channel" field was important since it reduces the amount of FNs drastically.
The evaluation on Table 6.11 and the counts of the values returned by the query of the listing mentioned above (see Table 6.10) reveal that only the Random Forest could correctly classify every single event, therefore having all the evaluation values of Precision, Recall, Accuracy, and F1-Score set to one.

6.2.3 Disable Windows Event Logging

Listing 5.14 shows the query used for getting the TP, TN, FP, and FN values as well as the evaluation results. The query mainly filters out keywords, but it also looks for certain EventIDs and the "CommandLine" field, which returns the only FP, as already known through the Bypass User Account Control attack. Moreover, this query filters out every channel not showing up in the normal data. However, only six of the attack events had a value different from the normal channel values. Most events were filtered out by the EventIDs, with 4719 filtering out 60, 4688 filtering out ten, and 4908 filtering out eight events. All the keywords filter out 94 events, however, all events with ID 4719 have the keyword "AuditPolicyChanges" included, which on its own therefore filters out 60 events. The "hackerserver" keyword filters out 28 events, and the other keywords each filtered two events.
Table 6.13 shows that the traditional model and the Random Forest classifier only misclassified one value. The other models returned worse results. However, since the Random Forest had a smaller dataset, this had a much higher impact on the evaluation values shown in Table 6.14, resulting in worse values. In this attack, the traditional approach surpasses the ML ones if evaluation values are taken into consideration, but if the counts of the misclassified values were used, the traditional approach performs equally well as the Random Forest classifier.

6.2.4 Summary

Tables 6.17 and 6.18 show the differences between the traditional approach and the ML ones, where the former summarizes the evaluation values of the traditional approach and

the latter the one of the ML approaches, including the algorithm performing the best.

Attack	Precision	Recall	Accuracy	F1-Score
Brute Force	1	0.5	0.78	0.67
Bypass User Account Control	1	0.98	1	0.99
Remote Services	1	0.97	1	0.99
Disable Windows Event Logging	0.99	1	1	1

Table 6.17: Performance of the considered attack tactics for the traditional approach

Attack	ML Algorithm	Precision	Recall	Accuracy	F1-Score
Brute Force	MLP	0.99	0.99	0.99	0.99
Bypass User Account Control	Random Forest	1	1	1	1
Remote Services	Random Forest	1	1	1	1
Disable Windows Event Logging	Random Forest	1	0.93	1	0.97

Table 6.18: Performance of the considered attack tactics for the ML approach

The tables reveal that the traditional approach is better for the Disable Windows Event Logging attack. However, it is worse in predicting whether a given event is an attack or a normal one for the Brute Force, the Bypass User Account Control, and the Remote Services scenarios.
Moreover, Table 6.18 shows that the Random Forest is the best choice except for the Brute Force attack, where the MLP performs slightly better having a Recall value that is better by 0.01, which arises because the Random Forest misclassified 281 more events than the MLP classifier.
The reason the OCSVM classifier performed so poorly for the attack scenarios is likely due to the EventIDs also being present in the normal dataset that was used for training. Therefore, the classification boundary could not be set, excluding the EventIDs of the attack events.

Table 6.19 shows the Runtime Performance of the fastest ML algorithms per attack. For the Brute Force attack, the three-day dataset was taken. It can be seen that the Naive Bayes is always the fastest classifier for training and testing it is mostly the Logistic Regression. But often the Naive Bayes is also very fast, this is shown in Tables 6.5, 6.6, 6.9, 6.12, and 6.15. The mentioned classifiers are quick for training and testing compared to the others.

Attack	ML Algorithm Training	ML Algorithm Testing Average
Brute Force	Naive Bayes: 0.03	Naive Bayes: 0.000000
Bypass User Account Control	Naive Bayes: 0.07	Logistic Regression: 0.000000
Remote Services	Naive Bayes: 0.04	Logistic Regression: 0.000000
Disable Windows Event Logging	Naive Bayes: 0.02	Logistic Regression: 0.000001

Table 6.19: Runtime Performance in seconds of the fastest ML classifier per attack

Chapter 7

Discussion

This chapter starts with a general paragraph about the data and fields used. It is then continued with discussing the four attacks, Brute Force, Bypass User Account Control, Remote Services, and Disable Windows Event Logging, separately. Later, the research questions are answered and the conclusions that could be taken throughout the thesis are presented.

Before starting with the attacks, it is important to know that if different data is used, it is likely to include different values because they can change per machine. This different data may lead to non-identical ML models. However, in case the features change, it would be possible to adjust the values used for the models, leading to a model that looks like the one used here.

Tables 6.17 and 6.18 show that for the Brute Force attack, the traditional approach was surpassed by the ML one except for the Precision value, where the traditional approach reaches the value 1 and the ML approaches reach 0.99. The traditional approach is better for the Precision, because it only has one FP, whereas all the ML approaches have 189 FPs upwards. For all other values (Recall, Accuracy, and F1-Score) the ML models perform better, except for the OCSVM. The reason why the Random Forest may not be the best model as was the case in [27] could be that they used a different dataset. Moreover, they did not look at the MLP classifier in detail, they mostly looked at the Random Forest, Naive Bayes, and Logistic Regression classifiers. Nevertheless, the difference between the MLP and the Random Forest classifiers is quite small.
If traditional and ML approaches are compared, both return high TP and TN values and a very low FP value, especially since the traditional approach has only one FP value. However, the FN count is the highest for the traditional approach if the poorly performing OCSVM is excluded. The FNs in the traditional approach consist of all 4648 events that co-occurred with 4625. In general, the ML approaches are clearly the better solution, except for the OCSVM. However, in this scenario, if only a log-in event is looked at, which consists of two events (with IDs 4648 and 4625), and it would only be needed to detect one of the two events to detect the attack, the traditional approach would surpass the ML one. The traditional approach is better in this case since the traditional would always detect the 4625 event, whereas the ML approach has some of the 4625 events not detected as an attack.

The next attack was the Bypass User Account Control. The ML classifiers, except for the SVM and the OCSVM had an improved performance compared to the traditional approach. Even though the ML models, except for the OCSVM, used only 148 attack events and the traditional used all the 709, the traditional had one FP and twelve FNs, whereas for the ML classifiers performing better they had only one, two or three misclassified events. However, the Naive Bayes misclassified three events, resulting in an equally good performance as the traditional approach if the values of the Precision, Recall, Accuracy, and F1-Score are summed up.

The Remote Services attack was the third attack examined. The evaluation, shown in Tables 6.17 and 6.18, revealed that the best ML approach, the Random Forest classifier, was able to classify all the events correctly. The traditional approach also had very high evaluation values. It would have been possible for this attack, but also for the others, to correctly classify all or almost all events with the traditional approach. However, this likely would have caused over-finetuning and it would not be possible to be applicable to other datasets.

For the Disable Windows Event Logging attack, the traditional approach returned better results compared to the ML classifiers as can be seen in Tables 6.17 and 6.18. However, the best ML classifier, the Random Forest in this case, misclassified the same amount of events as the traditional approach. The reason for the worse evaluation results lies in the fact that the test set of the ML classifiers only included 15 attack events, therefore it has a much higher impact if one of these 15 events is misclassified compared to 114 attack events of the traditional approach. The traditional approach only had one FP, therefore, the Precision value is not one. On the other hand, the only misclassified value of the Random Forest classifier is one FN, therefore the values of Recall and F1-Score are not equal to one.

For choosing the right ML model, it depends on which values need to be high. Sometimes it is necessary to have no FNs but it would not be severe to have high FPs.
Another point that needs to be defined is whether the selected features change or not. The ML models are easier to maintain since they are given a set of training data and establish their own rules to classify events. The rules do not need to be updated manually, which is needed for the traditional approach.
Also, the Runtime Performance may be taken into consideration for choosing the right ML classifier. Sometimes it needs to be fast, which would be helpful for real-time data and sometimes it is better to have higher Precision, Recall, Accuracy, and F1-Score but it does not need to be fast.
Moreover, with each attack, the amount of attack events became smaller. While the Brute Force attack included 6'843 attack events, the last attack, Disable Windows Event Logging, only included 114 attack events. Through the Tables 6.18 and 6.17, it looks like the traditional approach performs better for smaller datasets because it allows to finetune the rules for certain scenarios and to cover all eventualities. However, this raises the possibility of creating rules that only fit the given dataset because they are too narrow and would not detect new attacks.

After examining all the facts and investigating the attacks, it is now possible to answer the three research questions that were established in Section 1.1. It is started with the

first research question, consisting of two subquestions:

> **RQ 1.1:** *For which attack scenarios are traditional anomaly detection approaches, conducted via Splunk, well suited?*
>
> **RQ 1.2:** *Are modern ML anomaly detection approaches well suited to detect certain attack scenarios?*

RQ 1.1: Traditional anomaly detection approaches are especially well suited if there exists a rule that can label an event as an attack or not without a doubt. An example of this is the whitelisting approach of several EventIDs in some attacks, where all specific IDs only were present in the attack data, setting the label of "suspicious" to "true".
The example of the Brute Force attack, where the traditional approach labeled only one of the two co-occurring malicious events, namely the event with ID 4625, as an attack would be good enough to detect this kind of attack, however for other kinds of attacks, this may not be sufficient. The case with the Brute Force attack returned many FNs.
On the other hand, it would also be possible to have FP events as in the example of the Disable Windows Event Logging attack scenario. In this scenario, it may even be plausible to have a small amount of FP events but at the same time, the rules filter out all attack events. Sometimes this is the more desirable outcome compared to missing some attack events that could alter something in the computer, in this scenario through the "CommandLine" feature.
Overall, the traditional approach is useful for scenarios where there is a clear boundary between normal and abnormal behaviour. If certain features are sometimes true for normal events but also sometimes for attack events, the traditional is likely not a good approach to use.

RQ 1.2: The modern ML approaches used in this thesis were the Random Forest, the MLP, the SVM, the Logistic Regression, the Naive Bayes, and the OCSVM classifiers. Generally, it can be said according to the results, that at least one classifier, mostly the Random Forest, performed very well for each attack scenario.
For the first attack, the Brute Force, especially the Random Forest and the MLP classifiers performed very well for both testing datasets, the one with and the one without the three-day dataset. All evaluation values were at least 0.97. In the Bypass User Account Control again the Random Forest classifier outperformed all others with very high evaluation results of 0.99 or higher. For the Remote Services attack, the evaluation values always reached 1 when using the Random Forest classifier. In the last attack, Disable Windows Event Logging, the Random Forest again surpassed all other ML classifiers with evaluation values of 0.93 or higher. However, sometimes also the runtime needs to be taken into consideration, where the Naive Bayes and the Logistic Regression are the fastest ones. Moreover, it may also depend on the amount of data the model is trained on. More data can lead to more accurate predictions.
Knowing all this, it can be said that the ML classifiers did an excellent job on the data used in this work. The Random Forest seems to be the best choice if a single ML classifier needs to be chosen. To answer the research question, it can be reasoned that the ML classifier Random Forest is well suited for all the used attack scenarios since it has high values for Precision, Recall, Accuracy, and F1-Score, which are never below 0.93.

RQ 2: *In what aspects do ML based techniques surpass traditional ones when analyzing WELs for anomalies?*

RQ 2: Assuming the set of features taken for training the ML models are chosen carefully, returning good results, it is more likely to detect new attacks compared to the traditional rule-based approach [31]. Moreover, the rules of the traditional model need to be manually updated constantly, and they may become quite complex. Whereas the ML model just needs to be fine-tuned every now and then if the dataset taken at the beginning is varied.
The Brute Force attack supports this fact because the ML approach surpasses the traditional one since the traditional approach only labels one of the two co-occurring as suspicious events though both would belong to the attack. In contrast, the Random Forest classifier mostly labeled both events as attacks. If the traditional approach wanted to label both attack events as such, either all events with both IDs would need to be labeled as attacks, resulting in more FPs, or a quite complex query needed to be established to include them. For the Brute Force attack, this was not necessary, since it is sufficient to find half of the attack events in the used data avoiding more FPs.

RQ 3: *Which anomalies can be detected by*

a) Splunk and

b) ML approaches

through the analysis of WELs?

RQ 3: In Section 2.1, it was determined that only anomaly types of Type II (Uncommon class anomaly), Type IV (Multidimensional numeric anomaly), and Type V (Multidimensional categorical anomaly) were found during the work of this thesis.
WELs include a lot of information, allowing to distinguish normal from abnormal events. However, it is important to have a carefully selected set of features used and well-defined rules for the traditional approach, and for the ML approach, it is important to have a good and representative data set.
a) Through the traditional approach the anomaly subtypes Unusual Class, Deviant Repeater, Local Density Anomaly, and Uncommon Class Combination are theoretically all possible to detect through WELs. However, during the work of this thesis, through the traditional approach only the anomaly subtypes Unusual Class, Deviant Repeater, and Uncommon Class Combination were detected.
An example of events that belonged to the Unusual Class can be found in the Remote Services attack for the "Channel" field. The Deviant Repeater subtype occurred in the "CommandLine" field of the Disable Windows Event Logging attack when it is not empty, the event most likely belongs to an attack event. An example of the Uncommon Class Combination is the field "Workstation" and the word "Workstation". In this case, if the field is empty but the word is included, the event belonged to an attack event. However, it would also be possible to detect the Local Density Anomaly by the rule-based approach of Splunk. In the Brute Force attack, it would be possible to set a time frame and therefore

detect attacks that show up often within the given time frame, this would need a more complex set of rules and was not used in this work.

b) ML models allow the detection of the same anomaly subtypes as are possible with the traditional approach by using WELs. However, with this modern approach, it is harder to reason about the outcome of the classification of events because it is not simple, and sometimes not even possible, to explain how the model classifies.
In short, the modern approaches allow to detect all the anomaly subtypes mentioned in **a)**, however, it is hard to tell why they were found and because of which anomaly subtype they were determined. However, since the same dataset was given to the ML classifiers, it can be said that these anomaly subtypes can indeed be detected by the modern approach if WELs are used.

Chapter 8

Conclusion, Limitations, and Future Work

This chapter is divided into three sections. Starting with the Conclusion, the thesis is summarized, and it talks about the conclusions that could be taken during the work of this thesis. Next, the limitations are mentioned in Section 8.2 and discussed. After this, the chapter gives a short outlook on what could be done in the future to get better insights into this topic or to improve the outcome of what has been done so far. This is done in the second part of this chapter, in 8.3.

8.1 Conclusion

This thesis looked at four different attacks and tried to detect them using Windows Event Logs. The attacks that were considered are Brute Force, Bypass User Account Control, Remote Services, and Disable Windows Event Logging. Moreover, the performance difference between six different ML classifiers (Random Forest, MLP, SVM, Logistic Regression, Naive Bayes, and OCSVM) and a traditional rule-based approach was examined. The traditional approach was conducted in Splunk.
After a short data exploration phase, rules for the traditional model were created and features were selected for the ML approaches. The ML models were trained, and the events were used to obtain the counts of the TPs, TNs, FPs, and FNs. These values allowed to compare the traditional and the different ML classifiers by using the Precision, Recall, Accuracy, and F1-score values.
Overall, it turned out that the Random Forest was the best ML classifier for the attacks, except for the Brute Force attack, where the MLP was slightly better. If the ML and the traditional approaches are compared, the former mostly outperformed the traditional approach. The only exception is the Disable Windows Event Logging, where the traditional performed better according to the evaluation values Precision, Recall, Accuracy, and F1-Score. However, the Random Forest and the traditional approach both misclassified one event, but since there were more events in the traditional approach, this had a smaller impact on the evaluation values.

Generally, the Random Forest was a good classifier for all four attacks and performed quite well. Nevertheless, some models are faster for training and testing, like the Naive Bayes and the Logistic regression, but both were outperformed by the Random Forest classifier.

8.2 Limitations

The following three of the 14 tactics of [12] were not taken into account:

1. The first tactic, Reconnaissance, was not taken into account, since in this phase the adversary gathers information that can later be used to support targeting [12] and therefore, this tactic is difficult to distinguish from the normal behaviour of users.

2. In the second tactic, Resource Development, the adversary is trying to gather resources for supporting later operations. This is also hard to detect since it is possible to use different machines that are not known by the defenders, like buying or renting additional infrastructure that can be used during the attack [12].

3. The last tactic that did not seem to make sense to look at in this study is the twelfth, Command and Control since in this phase the attacker tries to mimic normal traffic behaviour to avoid detection. The adversary tries to communicate with compromised systems within a victim network and control them if they are under their control [12]. It is difficult to detect such behaviour since the compromised systems are under the attacker's control, which makes it hard to detect anomalous behaviour of such systems.

All other techniques would be interesting and possible to look at. This includes the techniques and sub-techniques of the following tactics: Initial Access, Execution, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Exfiltration, and Impact. In this study, only a small amount of the many sub-techniques were examined. The study [1] excluded the same three tactics.
It is possible that the ML classifiers would perform even better if they had a bigger training dataset. However, these data needed to be generated manually, since the amount of public data is very restricted. Moreover, attacks can be launched in different ways. Therefore, it is important to generate a broad training set, including as many variants as possible, to have a representative dataset. It also depends on what normal values are, this can change depending on the use case. It is possible that the features taken are too specific for the datasets used and may not be reused for different datasets. This raises the need for a broader and more diverse set of data.
The OCSVM classifier performed the worst. For this classifier, it would be possible to fine-tune it. In this work, only the default values were taken to train the model. In this work, six different ML classifiers were examined. However, it may be possible that there is another classifier that performs better.
Lastly, there are more possibilities to fine-tune the different ML classifiers. This may increase the prediction power of the different models. Nevertheless, it is very time-consuming

to find the best possible combination of parameters for all the classifiers. The fine-tuning could not be done in more detail for time-constraint reasons.

8.3 Future Work

For future studies and examinations of this topic, it would be of interest to look at more tactics, techniques, and sub-techniques to see whether there are differences or similarities between them. However, it is possible that not all the sub-techniques are possible to detect through WELs. As mentioned in the Limitations section (8.2), it is likely not possible to detect the three tactics Reconnaissance, Resource Development, and Command and Control with WELs. However, all other tactics seem to be possible to detect with the help of WELs. As it seems to be faster and easier to maintain, a ML setting may be superior to the traditional approach, if enough data is available. Therefore, it may be the better choice. Since, for this study, the Random Forest classifier always predicted the best or took the second place, it is likely that also for different sub-techniques it would perform quite well. However, since the attack with the most data available, the Brute Force attack, was surpassed by the MLP, the Random Forest model needs to be checked whether it also surpasses other classifiers if more data is available. Then it would be feasible for a different classifier to perform better. Nonetheless, it is possible that different classifiers may perform the best if all are fine-tuned and better parameters are used. This would also be interesting in the future, to use different parameters, allowing to fine-tune the classifiers.
Another fact is that a bigger dataset probably would return more precise data and lower the possibility of having too precise features for a given dataset. However, the generation of such data that is representative is a difficult and time-consuming process. Moreover, testing such a dataset on a more varied testing set may also allow for different conclusions. Moreover, as written in [47], real-time data could be used to see how a given model would react in a real-time setting and how it would perform. If this would return plausible results, it would be exciting to use this simulating companies with many different machines and networks. This would then may allow one to use it in production if the results would be satisfying and allow for real-world protection.

Bibliography

[1] Ananthapadmanabhan A. and Krishnashree Achuthan. "Threat Modeling and Threat Intelligence System for Cloud using Splunk". In: *2022 10th International Symposium on Digital Forensics and Security (ISDFS)*. 2022, pp. 1–6. DOI: 10.1109/ISDFS55398.2022.9800787.

[2] Awan Abid Ali and Naviani Avinash. *Naive Bayes Classification Tutorial using Scikit-learn*. Online, accessed: 20.09.2024. Mar. 2023. URL: https://www.datacamp.com/tutorial/naive-bayes-scikit-learn.

[3] Shafi Adam. *Random Forest Classification with Scikit-Learn*. Online, accessed: 20.09.2024. Feb. 2023. URL: https://www.datacamp.com/tutorial/random-forests-classifier-python.

[4] Gillis Alexander S., Burns Ed, and Brush Kate. *deep learning*. Online, accessed: 06.04.2024. July 2023. URL: https://www.techtarget.com/searchenterpriseai/definition/deep-learning-deep-neural-network.

[5] Gillis Alexander S. and Pratt K. *cyber attack*. Online, accessed: 28.03.2024. Dec. 2023. URL: https://www.techtarget.com/searchsecurity/definition/cyber-attack.

[6] Berlin Amanda. *What Are Event Logs and Why Do They Matter?* Online, accessed: 26.03.2024. Feb. 2023. URL: https://www.blumira.com/what-are-event-logs-and-why-do-they-matter/.

[7] Sharif Arfan. *WINDOWS LOGGING GUIDE: THE BASICS*. Online, accessed: 26.03.2024. Feb. 2023. URL: https://www.crowdstrike.com/guides/windows-logging/.

[8] Aurelie. *How to Fix Microsoft-Windows-PowerShell Event ID 4103?* Online, accessed: 27.08.2024. July 2023. URL: https://www.minitool.com/news/event-id-4103.html.

[9] Lutkevich Ben. *technical debt*. Online, accessed: 29.03.2024. Feb. 2023. URL: https://www.techtarget.com/whatis/definition/technical-debt.

[10] Chen C. and Gurganus J. "Statistical anomaly detection on metadata streams via commodity software to protect company infrastructure: a case study". In: *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)* (2017), pp. 252–257. DOI: 10.1109/ICDCSW.2017.53.

[11] Kidd Chrissy. *What Is Splunk & What Does It Do? A Splunk Intro*. www.splunk.com. Online, accessed: 28.03.2024. Mar. 2024. URL: https://www.splunk.com/en_us/blog/learn/what-splunk-does.html.

[12] T. M. Corporation. *MITRE ATT&CK*. attack.mitre.org. Online, accessed: 16.02.2024. 2023. URL: https://attack.mitre.org/.

[13] Arun D. et al. "Review and Analysis of Failure Detection and Prevention Review and Analysis of Failure Detection and Prevention Techniques in IT Infrastructure Monitoring Techniques in IT Infrastructure Monitoring". In: 2021. URL: https://api.semanticscholar.org/CorpusID:250528817.

[14] Jurafsky Daniel and Martin James H. *Speech and Language Processing*. Online, accessed: 20.09.2024. Aug. 2024. URL: https://web.stanford.edu/~jurafsky/slp3/5.pdf.

[15] danielmiessler. *SecLists*. 2024. URL: https://github.com/danielmiessler/SecLists/tree/master.

[16] IBM Data and AI Team. *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the difference?* Online, accessed: 01.04.2024. June 2023. URL: https://www.ibm.com/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks/.

[17] Carasso David. *Exploring Splunk*. CITO Research, 2012. ISBN: 9780982550670. URL: https://www.splunk.com/en_us/pdfs/exploring-splunk.pdf.

[18] Petersson David and Hashemi-Pour Cameron. *AI vs. machine learning vs. deep learning: Key differences*. Online, accessed: 02.04.2024. Nov. 2023. URL: https://www.techtarget.com/searchenterpriseai/tip/AI-vs-machine-learning-vs-deep-learning-Key-differences.

[19] *Deep Learning for Anomaly Detection*. Online, accessed: 04.04.2024. URL: https://ff12.fastforwardlabs.com/ff12-deep-learning-for-anomaly-detection.pdf.

[20] Bhanage Deepali Arun, Pawar Ambika Vishal, and Kotecha Ketan. "IT Infrastructure Anomaly Detection and Failure Handling: A Systematic Literature Review Focusing on Datasets, Log Preprocessing, Machine & Deep Learning Approaches and Automated Tool". In: *IEEE Access* 9 (2021), pp. 156392–156421. DOI: 10.1109/ACCESS.2021.3128283.

[21] scikit-learn developers. *OneClassSVM*. Online, accessed: 16.09.2024. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[22] DOMARS, alexbuckgit, and Kellylorenebaker. *Finding the process ID*. Online, accessed: 27.08.2024. Dec. 2023. URL: https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/finding-the-process-id.

[23] W. Ertel and N.T. Black. *Introduction to Artificial Intelligence*. Undergraduate Topics in Computer Science. Springer International Publishing, 2018. ISBN: 978-3319584874. URL: https://books.google.ch/books?id=geFHDwAAQBAJ.

[24] Hair Franklin Joseph et al. *Multivariate Data Analysis: A Global Perspective*. Pearson, Mar. 2010. ISBN: 0135153093.

[25] Pang G. et al. "Deep learning for anomaly detection: A review". In: *ACM computing surveys (CSUR)* (2021), pp. 1–38. DOI: https://doi.org/10.48550/arXiv.2007.02500.

[26] Pang Guansong, Cao Longbing, and Aggarwal Charu. "Deep Learning for Anomaly Detection: Challenges, Methods, and Opportunities". In: WSDM '21. Virtual Event, Israel: Association for Computing Machinery, 2021, pp. 1127–1130. ISBN: 978-1-4503-8297-7. DOI: 10.1145/3437963.3441659. URL: https://doi.org/10.1145/3437963.3441659.

[27] Amer Hamza and Rana Al-Janabi. "Detecting Brute Force Attacks Using Machine Learning". In: *BIO Web of Conferences* 97 (Apr. 2024), p. 00045. DOI: 10.1051/bioconf/20249700045.

[28] Katie Terrell Hanna. *brute-force attack*. Online, accessed: 01.05.2024. Sept. 2021. URL: https://www.techtarget.com/searchsecurity/definition/brute-force-cracking.

[29] Chen Hao, Xiao Ruizhi, and Jin Shuyuan. "Unsupervised Anomaly Detection Based on System Logs". In: *The 33rd International Conference on Software Engineering and Knowledge Engineering, SEKE 2021, KSIR Virtual Conference Center, USA, July J, 2021*. KSI Research Inc., 2021, pp. 92–97. DOI: 10.18293/SEKE2021-126. URL: https://doi.org/10.18293/SEKE2021-126.

[30] Swaminathan J. *What is Windows Event Log?* Online, accessed: 26.03.2024. May 2023. URL: https://www.eginnovations.com/blog/what-is-windows-event-log/.

[31] Su Te-Jen et al. "Attack detection of distributed denial of service based on Splunk". In: *2016 International Conference on Advanced Materials for Science and Engineering (ICAMSE)*. 2016, pp. 397–400. DOI: 10.1109/ICAMSE.2016.7840355.

[32] Biju Jibi Mariam, Gopal Neethu, and Prakash Anju J. "Cyber Attacks And Its Different Types". In: vol. 06. Internal Research Journal of Engineering and Technology (IRJET), 2019, pp. 4849–4852. URL: https://www.irjet.net/archives/V6/i3/IRJET-V6I31244.pdf.

[33] Wagner Joachim, Foster Jennifer, and Genabith Josef. "A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors". In: *Wagner, Joachim and Foster, Jennifer and van Genabith, Josef (2007) A comparative evaluation of deep and shallow approaches to the automatic detection of common grammatical errors. In: EMNLP-CoNLL 2007 - Joint Meeting of the Conference on Empirical Methods in Natural Language Processing and the Conference on Computational Natural Language Learning, 28-30 June 2007, Prague, Czech Republic.* (Jan. 2007).

[34] Dwyer John and Truta Traian Marius. "Finding anomalies in windows event logs using standard deviation". In: *9th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 2013, pp. 563–570. DOI: 10.4108/icst.collaboratecom.2013.254136.

[35] Steverson K. et al. "Cyber Intrusion Detection using Natural Language Processing on Windows Event Logs". In: *IEEE International Conference on Military Communication and Information Systems (ICMCIS)* (2021), pp. 1–7. DOI: 10.1109/ICMCIS52405.2021.9486307.

[36] Yasar Kinza and Gillis Alexander S. *Windows event log*. Online, accessed: 26.03.2024. Mar. 2023. URL: https://www.techtarget.com/searchwindowsserver/definition/Windows-event-log.

[37] Bart Lenaerts-Bergmans. *BRUTE FORCE ATTACKS*. Online, accessed: 01.05.2024. June 2022. URL: https://www.crowdstrike.com/cybersecurity-101/brute-force-attacks/.

[38] *Loggly*. Online, accessed: 12.04.2024. URL: https://www.loggly.com/product/.

[39] Torgo Luis. *Data Mining with R: Learning with Case Studies, Second Edition*. 2nd. Chapman & Hall/CRC. Boca Raton : CRC Press, 2017. ISBN: 9781482234893.

[40] Yuan Lun-Pin, Liu Peng, and Zhu Sencun. "Recompose Event Sequences vs. Predict Next Events: A Novel Anomaly Detection Approach for Discrete Event Logs". In: *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*. ASIA CCS '21. Virtual Event, Hong Kong: Association for Computing Machinery, 2021, pp. 336–348. ISBN: 9781450382878. DOI: 10.1145/3433210.3453098. URL: https://doi.org/10.1145/3433210.3453098.

[41] Uma M. and Padmavathi G. "A Survey on Various Cyber Attacks and their Classification". In: *Int. J. Netw. Secur.* 15 (Sept. 2013), pp. 390–396. URL: http://ijns.jalaxy.com.tw/contents/ijns-v15-n5/ijns-2013-v15-n5-p390-396.pdf.

[42] Hristov Marian et al. "Integration of Splunk Enterprise SIEM for DDoS Attack Detection in IoT". In: *2021 IEEE 20th International Symposium on Network Computing and Applications (NCA)*. 2021, pp. 1–5. DOI: 10.1109/NCA53618.2021.9685977.

[43] Fujimoto Mariko, Matsuda Wataru, and Mitsunaga Takuho. "Detecting Abuse of Domain Administrator Privilege Using Windows Event Log". In: *2018 IEEE Conference on Application, Information and Network Security (AINS)*. 2018, pp. 15–20. DOI: 10.1109/AINS.2018.8631459.

[44] mdecrevoisier. *EVTX-to-MITRE-Attack*. 2024. URL: https://github.com/mdecrevoisier/EVTX-to-MITRE-Attack.

[45] Shyu Mei-Ling et al. "A Novel Anomaly Detection Scheme Based on Principal Component Classifier". In: Jan. 2003.

[46] Baldani Michael. *The Ultimate Guide to Windows Event Logging*. Online, accessed: 26.03.2024. Jan. 2024. URL: https://www.sumologic.com/blog/windows-event-logging/.

[47] Cersosimo Michelle and Lara Adrian. "Detecting Malicious Domains using the Splunk Machine Learning Toolkit". In: *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*. 2022, pp. 1–6. DOI: 10.1109/NOMS54207.2022.9789899.

[48] Federchuk Mikayla. *The Difference Between AI ML and DL*. Online, accessed: 10.04.2024. Oct. 2022. URL: https://www.cengn.ca/information-centre/innovation/difference-between-ai-ml-and-dl/.

[49] Du Min et al. "DeepLog: Anomaly Detection and Diagnosis from System Logs through Deep Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 2017, pp. 1285–1298. ISBN: 9781450349468. DOI: 10.1145/3133956.3134015. URL: https://doi.org/10.1145/3133956.3134015.

[50] Munir Mohsin et al. "DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series". In: *IEEE Access* 7 (2019), pp. 1991–2005. DOI: 10.1109/ACCESS.2018.2886457.

[51] Thi Quynh Nguyen et al. "Detecting abnormal DNS traffic using unsupervised machine learning". In: *2020 4th Cyber Security in Networking Conference (CSNet)*. 2020, pp. 1–8. DOI: 10.1109/CSNet50428.2020.9265466.

[52] Laskowski Nicole and Tucci Linda. *artificial intelligence (AI)*. Online, accessed: 02.04.2024. Nov. 2023. URL: https://www.techtarget.com/searchenterpriseai/definition/AI-Artificial-Intelligence.

[53] Winnie Ondara. *How to Find Parent Process PPID in Linux*. Online, accessed: 27.08.2024. July 2023. URL: https://www.tecmint.com/find-parent-process-

ppid/#:~:text=Thus%2C%20the%20PPID%20is%20the,the%20child%20process%20is%206..

[54] Kitsuchart Pasupa and Wisuwat Sunhem. "A comparison between shallow and deep architecture classifiers on small dataset". In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. 2016, pp. 1–6. DOI: 10.1109/ICITEED.2016.7863293.

[55] Grieve Patrick. *Deep learning vs. machine learning*. Online, accessed: 01.04.2024. Nov. 2023. URL: https://www.zendesk.com/blog/machine-learning-and-deep-learning/.

[56] Sahoo Pradipta Kumar, Chottray R. K., and Pattnaiak S. "Research Issues on Windows Event Log". In: *International Journal of Computer Applications* 41 (2012), pp. 40–48. URL: https://api.semanticscholar.org/CorpusID:3262257.

[57] Chalapathy Raghavendra and Chawla Sanjay. "Deep Learning for Anomaly Detection: A Survey". In: *CoRR* abs/1901.03407 (Jan. 2019). arXiv: 1901.03407. URL: http://arxiv.org/abs/1901.03407.

[58] Foorthuis Ralph. "On the nature and types of anomalies: a review of deviations in data". In: *International Journal of Data Science and Analytics* 12 (Aug. 2021), pp. 1–35. DOI: 10.1007/s41060-021-00265-1.

[59] Smith Randy Franklin. *Ultimate IT Security*. www.ultimatewindowssecurity.com. Online, accessed: 18.03.2024. 2023. URL: https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/default.aspx.

[60] Al-amri Redhwan et al. "A Review of Machine Learning and Deep Learning Techniques for Anomaly Detection in IoT Data". In: *Applied Sciences* 11 (June 2021). DOI: 10.3390/app11125320. URL: https://doi.org/10.3390/app11125320.

[61] Gandhi Rohith. *Support Vector Machine - Introduction to Machine Learning Algorithms*. Online, accessed: 20.09.2024. June 2018. URL: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.

[62] sbousseaden. *EVTX-ATTACK-SAMPLES*. 2023. URL: https://github.com/sbousseaden/EVTX-ATTACK-SAMPLES.

[63] Jaiswal Sejal. *Multilayer Perceptrons in Machine Learning: A Comprehensive Guide*. Online, accessed: 20.09.2024. July 2024. URL: https://www.datacamp.com/tutorial/multilayer-perceptrons-in-machine-learning.

[64] Shende Shailendra W. "Artificial intelligence and machine learning for internet of things". In: *Journal of Physics: Conference Series* (2021). DOI: 10.1088/1742-6596/1913/1/012151. URL: https://dx.doi.org/10.1088/1742-6596/1913/1/012151.

[65] Nabil Shallik and Hubert Klaus. "Machine Learning for Anomaly Detection in Cloud Data Warehousing: Anomaly detection techniques to identify unusual activities". In: (Jan. 2024).

[66] He Shilin et al. "Experience Report: System Log Analysis for Anomaly Detection". In: *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 2016, pp. 207–218. DOI: 10.1109/ISSRE.2016.21.

[67] Randy Franklin Smith. *Ultimate IT Security*. Online, accessed: 23.05.2024. URL: https://github.com/danielmiessler/SecLists/tree/master.

[68] *Splunk Enterprise Overview*. Online, accessed: 28.03.2024. Feb. 2024. URL: https://docs.splunk.com/Documentation/Splunk/9.2.0/Overview/AboutSplunkEnterprise.

[69] Chandola Varun, Banerjee Arindam, and Kumar Vipin. "Anomaly detection: A survey". In: *ACM Comput. Surv.* 41.3 (July 2009). ISSN: 0360-0300. DOI: 10.1145/1541880.1541882. URL: https://doi.org/10.1145/1541880.1541882.

[70] Chai Wesley. *fuzzy logic*. Online, accessed: 03.04.2024. June 2021. URL: https://www.techtarget.com/searchenterpriseai/definition/fuzzy-logic.

[71] Wang X. et al. "MADDC: Multi-Scale Anomaly Detection, Diagnosis and Correction for Discrete Event Logs". In: *Proceedings of the 38th Annual Computer Security Applications Conference* (2022), pp. 769–784. DOI: https://doi.org/10.1145/3564625.3567972.

[72] Yang Ying, Webb Geoffrey Ian, and Wu Xindong. "Discretization Methods". English. In: *The Data Mining and Knowledge Discovery Handbook*. Ed. by Maimon Oded and Rokach Lior. 1st ed. Springer, 2005, pp. 101–116. ISBN: 9780387244358. DOI: 10.1007/978-0-387-09823-4_6.

List of Figures

2.1 Anomaly types reproduced from [58] 6

2.2 Anomaly Subtypes reproduced from [58] 8

2.3 Example log in the Windows Event Viewer 11

2.4 Splunk structure reproduced from [42] 18

2.5 Relationship between AI, ML and DL as defined by [64] 19

2.6 Supervised and unsupervised ML training inputs reproduced from [19] . . . 21

2.7 Semi-supervised learning, nml stands for normal and abn means abnormal, reproduced from [19] 22

2.8 Anomaly detection methods, reproduced from [19] 23

2.9 Decision trees, reproduced from [3] 24

2.10 Possible hyperplanes, reproduced from [61] 24

2.11 Sigmoid function, reproduced from [14] 25

2.12 OCSVM during training and testing, both figures are reproduced from [19] 26

2.13 Deep neural feed-forward network with input and output layer and three layers in between, reproduced from [16] 27

4.1 Process flow of data preparation and analyzation. 42

5.1 Output of Listing 5.10 53

5.2 Output of Listing 5.12 54

5.3 Output of Listing 5.14 56

6.1 Difference between normal and Brute Force events in the data, if more than five events occurred in ten seconds 76

6.2 Top five EventIDs of normal events 76

6.3 Brute Force Events EventIDs except for the own Brute Force attacks . . . 77

6.4 Brute Force attacks EventIDs of the own attacks if ten events took place within one second 77

6.5 Shows all EventIDs of own Brute Force attack if within one second more than ten events took place 78

6.6 First day filtered for EventIDs noted in 5.4 and split into one minute buckets 79

6.7 Second day filtered for EventIDs noted in 5.4 and split into one minute buckets 79

6.8 Third day filtered for EventIDs noted in 5.4 split into one minute buckets . 79

6.9 EventIDs of the Bypass User Account Control attack events 86

6.10 EventID values of the attack events 92

6.11 EventID values of attack events also present in the normal dataset 92

6.12 EventID values of the attack events 98

6.13 EventID values of the attack events also showing up in the normal data . . 98

6.14 Keywords in attack and normal events 99

6.15 Channel values of normal data 99

6.16 Channel values of attack data 99

6.17 “CommandLine” values of attack and data 100

B.1 Normal EventIDs if within one second more than ten events took place . . 150

B.2 Shows all EventIDs if within one second more than ten events took place excluding the own Password Guessing and Password Spraying attack events 150

B.3 Count of words including “fail”, distinguished by normal and Brute Force events. 150

B.4 Events with words including “fail” and “log”, distinguished by index 151

B.5 Filtered EventIDs including words like fail and login with more than one event in ten seconds. 151

B.6 EventIDs of all events excluding the three-day dataset where more than ten events happened within one second 151

B.7 All three days filtered for the EventIDs noted in 5.4 and split into one minute buckets, each day has a color 152

B.8 Results of the prediction of all Brute Force ML approaches with their correctly and misclassified labels (1/2) 153

B.9 Results of the prediction of all Brute Force ML approaches with their correctly and misclassified labels (2/2) 154

B.10 Results of the TP, TN, FP, and FN predictions of selected Brute Force ML approaches with their correctly and misclassified labels (1/2) 155

B.11 Results of the TP, TN, FP, and FN predictions of selected Brute Force ML approaches with their correctly and misclassified labels (2/2) 156

B.12 SubjectDomainNames of the Bypass User Account Control attack events and normal events, the color shows the indices 157

B.13 All ProcessID values of the Bypass User Account Control (corresponding code in Listing A.29) 157

B.14 Top ten ProcessID values of the normal events (the code for this plot is given in Listing A.30) 157

B.15 Top ten ProcessId values of the attack events (code in Listing A.31) 158

B.16 Top ten ProcessId values of the normal events (find the corresponding code in Listing A.32) 158

B.17 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Bypass User Account Control attack scenario (1/2) 159

B.18 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Bypass User Account Control attack scenario (2/2) 160

B.19 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Bypass User Account Control attack (1/2) 161

B.20 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Bypass User Account Control attack (2/2) 162

B.21 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Remote Services attack scenario (1/2) 163

B.22 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Remote Services attack scenario (2/2) 164

B.23 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Remote Services attack (1/2) 165

B.24 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Remote Services attack (2/2) 166

B.25 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Disable Windows Event Logging attack scenario (1/2) 167

B.26 Results of the prediction of all ML approaches with their correctly and misclassified labels for the Disable Windows Event Logging attack scenario (2/2) 168

B.27 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Disable Windows Event Loggings attack (1/2) 169

B.28 Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Disable Windows Event Logging attack (2/2) 170

List of Tables

3.1 Authors of the main papers used for the thesis 30

4.1 Considered attack tactics with inspected techniques 40

5.1 Libraries used for the ML approaches 57

6.1 Brute Force except for three-day dataset TP, TN, FP, and FN values, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 81

6.2 Brute Force except three-day dataset evaluation results of all approaches, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 82

6.3 three-day dataset Brute Force ML TP, TN, FP, and FN values, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 83

6.4 three-day dataset Brute Force evaluation results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 84

6.5 Runtime Performance of the ML classifiers in seconds for training and testing of the Brute Force attack (not including the three-day dataset) 85

6.6 Runtime Performance of the ML classifiers in seconds for testing the Brute Force attack of the three-day dataset 85

6.7 Bypass User Account Control results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional . . . 89

6.8 Bypass User Account Control evaluation results, RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 90

6.9 Runtime Performance of the ML classifiers in seconds for the Bypass User Account Control attack 91

6.10 Summary of Remote Services ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 95

6.11 Summary of Remote Services ML evaluation results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 96

6.12 Runtime Performance of the ML classifiers in seconds for the Remote Services attack 97

6.13 Summary of Disable Windows Event Logging ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 102

6.14 Summary of Disable Windows Event Logging ML results with Traditional results for comparison. RF stands for Random Forest, LR for Logistic Regression, NB for Naive Bayes, and Trad for Traditional 103

6.15 Runtime Performance of the ML classifiers in seconds for the Disable Windows Event Logging attack 104

6.16 OCSVM variants and their results 104

6.17 Performance of the considered attack tactics for the traditional approach . 108

6.18 Performance of the considered attack tactics for the ML approach 108

6.19 Runtime Performance in seconds of the fastest ML classifier per attack . . 108

Listings

4.1 Query returning Brute Force and normal events . . . 41
5.1 parse_xml function . . . 47
5.2 process_xml_files function . . . 47
5.3 concatenate_files function . . . 48
5.4 Full SPL query searching for Brute Force events . . . 49
5.5 SPL query that creates plots for a single day, here 04.06.2024 . . . 49
5.6 SPL query that creates the plot combining the three days shown in Figures 6.6, 6.7, and 6.8 . . . 50
5.7 Evaluation of all data except the three-day dataset . . . 50
5.8 Calculation of the FNs . . . 51
5.9 SPL query for calculating the TP, TN, and FP and FN values and calculating the Precision, Recall, Accuracy, and F1-Score . . . 51
5.10 SPL query to show the TPs, TNs, and FPs, producing the table in Figure 5.1 . . . 52
5.11 SPL query returning the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score . . . 53
5.12 SPL filter query of the Remote Services events . . . 54
5.13 SPL query showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score . . . 54
5.14 SPL query to filter the Disable Windows Event Logging events . . . 55
5.15 SPL query for showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score . . . 56
5.16 detect_encoding function . . . 57
5.17 parse_events function . . . 57
5.18 load_events_from_files function . . . 58
5.19 Combining the normal and attack data frames . . . 58
5.20 Counting the events within one second . . . 59
5.21 Counting usernames within one second (same username, different username, and all usernames) . . . 59
5.22 Splitting the data for training the model . . . 60
5.23 Training and fitting the model for a Random Forest algorithm and calculating the Runtime Performance . . . 60
5.24 Training and fitting of the model for a MLP algorithm . . . 60
5.25 Training and fitting of the model for a SVM algorithm . . . 61
5.26 Training and fitting of the model for a Logistic Regression algorithm . . . 61
5.27 Training and fitting of the model for a Naive Bayes algorithm . . . 61
5.28 Training and fitting of the model for an OCSVM algorithm . . . 61

5.29 Labeling of the three-day dataset, differentiating between normal and attack events . . . 61
5.30 Code for predicting the outcomes of the models . . . 62
5.31 parse_events function for Bypass User Account Control attack . . . 63
5.32 compute_entropy function for Bypass User Account Control attack . . . 64
5.33 Sorting according to the "SystemTime" column . . . 64
5.34 Code for creating four new columns . . . 65
5.35 New data frame for the OCSVM model, needed for the prediction . . . 65
5.36 Training and fitting of the OCSVM model . . . 65
5.37 Code for generating the classification report and confusion matrix . . . 66
5.38 Adding columns to prepare the plotting . . . 66
5.39 Adding columns to prepare the plotting for the OCSVM . . . 66
5.40 parse_events function for the Remote Services attack . . . 67
5.41 One-hot encoding and string to integer transformation . . . 68
5.42 One-hot encoding for the OCSVM classifier . . . 68
5.43 parse_events function for the Disable Windows Event Logging attack . . . 71
5.44 Transformating the "Channel" and "CommandLine" columns . . . 72
A.1 Python file for preprocessing XML files such that each event is on one line with linebreaks inbetween the different events. It gets a file like A.42 and outputs a file like A.43 . . . 135
A.2 SPL query for showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score . . . 136
A.3 Imports for Random Forest . . . 136
A.4 Additional import for the Remote Services Python script . . . 136
A.5 Imports for other classifiers, replace line seven of Listing A.3 . . . 136
A.6 Code returning the TP, TN, FP, and FN values . . . 137
A.7 Code for the plot that shows the correctly and misclassified labels predicted by the chosen model . . . 137
A.8 Code for the plot that shows correct labels and the TP, TN, FP, and FN points . . . 138
A.9 hex_to_int function for Bypass User Account Control attack . . . 138
A.10 Adding columns to prepare the plotting . . . 139
A.11 Adding columns to prepare the plotting . . . 139
A.12 Plots the graph shown in B.1 . . . 140
A.13 Plots the graph shown in B.2 . . . 140
A.14 Plots the graph shown in 6.1(a) . . . 141
A.15 Plots the graph shown in 6.1(b) . . . 141
A.16 Plots the graph shown in 6.2 . . . 141
A.17 Plots the graph shown in 6.3 . . . 141
A.18 Plots the graph shown in 6.4 . . . 141
A.19 Plots the graph shown in 6.5 . . . 141
A.20 Plots the graph in B.3 . . . 142
A.21 Plots the graph in B.4 . . . 142
A.22 Plots the graph shown in B.5 . . . 142
A.23 Plots the graph shown in B.6 . . . 142
A.24 Plots the graph shown in 6.6 . . . 142
A.25 Plots the graph shown in 6.7 . . . 143

A.26 Plots the graph shown in 6.8 143
A.27 Plots the graph shown in B.7 144
A.28 Plots the ”Bypass User Account Control Events: EventIDs” graph shown in 6.9 144
A.29 Plots the ”Bypass User Account Control Events: ProcessIDs” graph shown in B.13 144
A.30 Plots the ”Normal Events: ProcessIDs” graph shown in B.14 144
A.31 Plots the ”Bypass User Account Control Events: Top 10 ProcessIds” graph shown in B.15 144
A.32 Plots the ”Normal Events: Top 10 ProcessIds” graph shown in B.16 145
A.33 Plots the ”Subject Domain Names” graph shown in B.12 145
A.34 Plots the ”Remote Services Events: EventIDs” graph shown in 6.10 145
A.35 Plots the ”Attack EventIDs present in Normal Data” graph shown in 6.11 . 145
A.36 Plots the ”Disable Windows Event Logging Events: EventIDs” graph shown in 6.12 145
A.37 Plots the ”Disable Windows Event Logging EventIDs present in Normal Data” graph in 6.13 145
A.38 Plots the ”Keywords by Index” graph in 6.14 146
A.39 Plots the ”Normal Events: Channels” graph in 6.15 146
A.40 Plots the ”Disable Windows Event Logging Events: Channels” graph in 6.16 146
A.41 Plots the ”Disable Windows Event Logging and Normal Events: CommandLine” graph in 6.17 146
A.42 The same context as A.43 but not yet processed by the python script A.1 . 146
A.43 The same context as A.42 but processed by the python script A.1 147

Appendix A

Code and Files

A.1 Python Code

```
import os
import xml.etree.ElementTree as ET

def save_events_to_file(events, filepath):
    # Write events content to file
    with open(filepath, 'w') as f:
        for event in events:
            # Remove the namespace prefix from the XML content
            content = ET.tostring(event, encoding='unicode')
            content = content.replace('ns0:', '').replace('xmlns:ns0=',
   'xmlns=')
            f.write(content.strip().replace('\n', '') + "\n\n")

def parse_xml(xml_file, output_file):
    # Parse XML file
    tree = ET.parse(xml_file)
    root = tree.getroot()

    # Get all Eventselements
    events = root.findall('.//{http://schemas.microsoft.com/win/2004/08/
   events/event}Event')
    filename = os.path.basename(xml_file)
    output_file = os.path.join(output_folder, filename)

    # Save all events to a single file
    save_events_to_file(events, output_file)

def process_xml_files(input_folder, output_folder):
    # Iterate over each XML file in the input folder
    for filename in os.listdir(input_folder):
        if filename.endswith(".xml"):
            xml_file = os.path.join(input_folder, filename)
            output_file = os.path.join(output_folder, filename)
            parse_xml(xml_file, output_file)

```

```
input_folder = 'G:/Meine Ablage/MA/Attacken/Credential Access - Brute Force/EVTX-ATTACK-SAMPLES/xml dateien/'  # Change to the path of XML files
output_folder = 'G:/Meine Ablage/MA/Attacken/Credential Access - Brute Force/EVTX-ATTACK-SAMPLES/Data for Splunk/'  # Change to the output file path

process_xml_files(input_folder, output_folder)
```

Listing A.1: Python file for preprocessing XML files such that each event is on one line with linebreaks inbetween the different events. It gets a file like A.42 and outputs a file like A.43

```
import sys
import os
import pandas as pd
import re
import chardet
import matplotlib as plt
import sklearn as sklearn
import datetime
import math
from collections import Counter


print(sys.version)
print(f"os: {os.sys.version}")
print(f"pandas: {pd.__version__}")
print(f"re: {re.__version__ if hasattr(re, '__version__') else 'N/A'}")
print(f"chardet: {chardet.__version__}")
print(f"matplotlib: {plt.__version__}")
print(f"sklearn: {sklearn.__version__}")
```

Listing A.2: SPL query for showing the TP, TN, and FP and FN values as well as the Precision, Recall, Accuracy, and F1-Score

```
import os
import pandas as pd
import re
import chardet
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from datetime import datetime
```

Listing A.3: Imports for Random Forest

```
from sklearn.preprocessing import LabelEncoder
```

Listing A.4: Additional import for the Remote Services Python script

```
# MLP
from sklearn.neural_network import MLPClassifier

# SVM
```

```
from sklearn.svm import SVC

# Logistic Regression
from sklearn.linear_model import LogisticRegression

# Naive Bayes
from sklearn.naive_bayes import MultinomialNB

# OCSVM
from sklearn.metrics import OneClassSVM
```

Listing A.5: Imports for other classifiers, replace line seven of Listing A.3

```
y_true = new_df['Label']
y_pred = new_df['PredictedLabel']

tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()
```

Listing A.6: Code returning the TP, TN, FP, and FN values

```
plt.figure(figsize=(15, 7))

# Plot actual labels
plt.scatter(new_df['Timestamp'], new_df['Label'].astype(int), c='blue',
    label='Actual Labels')

# Plot predicted labels with slight offset for distinction
plt.scatter(new_df['Timestamp'], new_df['PredictedLabel'].astype(int) -
    0.03, c='red', marker='o', label='Predicted Labels')

# Correctly classified labels
correct_labels = new_df[new_df['Label'] == new_df['PredictedLabel']]

# Misclassified labels
misclassified_labels = new_df[new_df['Label'] != new_df['PredictedLabel'
    ]]

# Plot correctly classified labels with a small offset
plt.scatter(correct_labels['Timestamp'], correct_labels['PredictedLabel'
    ].astype(int) - 0.06, c='violet', marker='x', label='Correctly
    Classified Labels')

# Plot misclassified labels with a small offset
plt.scatter(misclassified_labels['Timestamp'], misclassified_labels['
    PredictedLabel'].astype(int) - 0.09, c='black', marker='x', label='
    Misclassified Labels')

plt.xlabel('Time')
plt.ylabel('Label')
plt.title('Events Over Time')

# Set y-ticks to display "Attack" and "no Attack" instead of 0 up to 1
plt.yticks([0, 1], ['no Attack', 'Attack'])

plt.legend(fontsize=14) # This increases the size of the legend
```

```
plt.show()
```

Listing A.7: Code for the plot that shows the correctly and misclassified labels predicted by the chosen model

```
plt.figure(figsize=(15, 7))

# Plot actual labels
plt.scatter(new_df['Timestamp'], new_df['Label'].astype(int), c='blue',
    label='Actual Labels')

# Plot predicted labels with slight offset for distinction
plt.scatter(new_df['Timestamp'], new_df['PredictedLabel'].astype(int) -
    0.02, c='red', marker='o', label='Predicted Labels')

# TP, TN, FP, and FN labels
TP = new_df[(new_df['Label'] == 1) & (new_df['PredictedLabel'] == 1)]
TN = new_df[(new_df['Label'] == 0) & (new_df['PredictedLabel'] == 0)]
FP = new_df[(new_df['Label'] == 0) & (new_df['PredictedLabel'] == 1)]
FN = new_df[(new_df['Label'] == 1) & (new_df['PredictedLabel'] == 0)]



# Plot labels with a slight offset
plt.scatter(TP['Timestamp'], TP['PredictedLabel'].astype(int) - 0.04, c=
    'violet', marker='x', label='TP')
plt.scatter(TN['Timestamp'], TN['PredictedLabel'].astype(int) - 0.04, c=
    'orange', marker='x', label='TN')
plt.scatter(FP['Timestamp'], FP['PredictedLabel'].astype(int) - 0.06, c=
    'grey', marker='x', label='FP')
plt.scatter(FN['Timestamp'], FN['PredictedLabel'].astype(int) - 0.06, c=
    'black', marker='x', label='FN')


plt.xlabel('Time')
plt.ylabel('Label')
plt.title('Events Over Time')
# Set y-ticks to display "Attack" and "no Attack" instead of 0 up to 1
plt.yticks([0, 1], ['no Attack', 'Attack'])
plt.legend(fontsize=14)
plt.show()
```

Listing A.8: Code for the plot that shows correct labels and the TP, TN, FP, and FN points

```
def hex_to_int(hex_str):
    try:
        return int(hex_str, 16)  # Convert hexadecimal string to integer
    except ValueError:
        return np.nan  # Handle invalid values
```

Listing A.9: hex_to_int function for Bypass User Account Control attack

Listing A.10 plots the graphs in Figures B.17 and B.18, visualizing the correctly and misclassified labels, and Listing A.11 plots the graphs of Figures B.19 and B.20, showing the TPs, TNs, FPs, and FNs. For the OCSVM approach, the plotting looked the same as the two listings below, except the ”X_test” is exchanged with ”plot_data”.

```
plt.figure(figsize=(15, 7))

# Plot actual labels
plt.scatter(X_test['SystemTime'], X_test['ActualLabel'].astype(int), c='blue', label='Actual Labels')

# Plot predicted labels with slight offset for distinction
plt.scatter(X_test['SystemTime'], X_test['PredictedLabel'].astype(int) - 0.03, c='red', marker='o', label='Predicted Labels')

# Correctly classified labels
correct_labels = X_test[X_test['ActualLabel'] == X_test['PredictedLabel']]
# Misclassified labels
misclassified_labels = X_test[X_test['ActualLabel'] != X_test['PredictedLabel']]

# Plot correctly classified labels with a slight offset
plt.scatter(correct_labels['SystemTime'], correct_labels['PredictedLabel'].astype(int) - 0.06, c='violet', marker='x', label='Correctly Classified Labels')

# Plot misclassified labels with a slight offset
plt.scatter(misclassified_labels['SystemTime'], misclassified_labels['PredictedLabel'].astype(int) - 0.09, c='black', marker='x', label='Misclassified Labels')

plt.xlabel('SystemTime')
plt.ylabel('Label')
plt.title('SVM - Events Over Time')

# Define x-axis labels to show only whole years
years = pd.date_range(start=X_test['SystemTime'].min(), end=X_test['SystemTime'].max(), freq='YS')
plt.xticks(years, years.year)
# Set y-ticks to display only "Attack" and "no Attack"
plt.yticks([0, 1], ['no Attack', 'Attack'])

plt.legend(fontsize=14)
plt.show()
```

Listing A.10: Adding columns to prepare the plotting

```
plt.figure(figsize=(15, 7))
# Plot actual labels
plt.scatter(X_test['SystemTime'], X_test['ActualLabel'].astype(int), c='blue', label='Actual Labels')

# Plot predicted labels with slight offset for distinction
plt.scatter(X_test['SystemTime'], X_test['PredictedLabel'].astype(int) - 0.02, c='red', marker='o', label='Predicted Labels')

# TP, TN, FP, and FN labels
TP = X_test[(X_test['ActualLabel'] == 1) & (X_test['PredictedLabel'] == 1)]
TN = X_test[(X_test['ActualLabel'] == 0) & (X_test['PredictedLabel'] == 0)]
```

```
FP = X_test[(X_test['ActualLabel'] == 0) & (X_test['PredictedLabel'] ==
    1)]
FN = X_test[(X_test['ActualLabel'] == 1) & (X_test['PredictedLabel'] ==
    0)]

# Plot labels with a slight offset
plt.scatter(TP['SystemTime'], TP['PredictedLabel'].astype(int) - 0.04, c
    ='violet', marker='x', label='TP')
plt.scatter(TN['SystemTime'], TN['PredictedLabel'].astype(int) - 0.04, c
    ='orange', marker='x', label='TN')
plt.scatter(FP['SystemTime'], FP['PredictedLabel'].astype(int) - 0.06, c
    ='grey', marker='x', label='FP')
plt.scatter(FN['SystemTime'], FN['PredictedLabel'].astype(int) - 0.06, c
    ='black', marker='x', label='FN')

plt.xlabel('SystemTime')
plt.ylabel('Label')
plt.title('SVM - Events Over Time')

# Set y-ticks to display only "Attack" and "no Attack"
plt.yticks([0, 1], ['no Attack', 'Attack'])

plt.legend(fontsize=14)
plt.show()
```

Listing A.11: Adding columns to prepare the plotting

A.1.1 Splunk Code

```
index="normal_events"
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 1) *
    1
| eventstats count as event_count by bin_time
| where event_count > 10
| stats count as count_EventID by EventID
| sort -count_EventID
| head 10
```

Listing A.12: Plots the graph shown in B.1

```
index="brute_force" NOT (source="PWSpraying-OWN-Laptop.xml" OR
    source="PasswordGuessing-OWN-Laptop.xml")
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 1) *
    1
| eventstats count as event_count by bin_time
| where event_count > 10
| stats count as count_EventID by EventID
| sort -count_EventID
```

Listing A.13: Plots the graph shown in B.2

```
index="normal_events"
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 10)
   * 10
| eval bin_time = strftime(bin_time, "%Y-%m-%d %H:%M:%S")
| eventstats count as event_count by bin_time
| where event_count > 5
| stats count by bin_time
| sort bin_time
```

Listing A.14: Plots the graph shown in 6.1(a)

```
index="brute_force"
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 10)
   * 10
| eval bin_time = strftime(bin_time, "%Y-%m-%d %H:%M:%S")
| eventstats count as event_count by bin_time
| where event_count > 5
| stats count by bin_time
| sort bin_time
```

Listing A.15: Plots the graph shown in 6.1(b)

```
index="normal_events"
| stats count(eval(EventID)) AS count_EventID by EventID
| sort -count_EventID
| head 5
```

Listing A.16: Plots the graph shown in 6.2

```
index="brute_force" NOT (source="PWSpraying-OWN-Laptop.xml" OR
    source="PasswordGuessing-OWN-Laptop.xml")
| stats count(eval(EventID)) AS count_EventID by EventID
| sort -count_EventID
| head 5
```

Listing A.17: Plots the graph shown in 6.3

```
index="brute_force" source="PWSpraying-OWN-Laptop.xml" OR
    source="PasswordGuessing-OWN-Laptop.xml"
| stats count(eval(EventID)) AS count_EventID by EventID
| sort -count_EventID
```

Listing A.18: Plots the graph shown in 6.4

```
index="brute_force" source="PWSpraying-OWN-Laptop.xml" OR source="
   PasswordGuessing-OWN-Laptop.xml"
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 1) *
    1
| eventstats count as event_count by bin_time
| where event_count > 10
| stats count as count_EventID by EventID
```

```
| sort -count_EventID
```

Listing A.19: Plots the graph shown in 6.5

```
index="brute_force" OR index="normal_events"
| rex field=_raw "(?i)(?<fail_word>\w*fail\w*)"
| chart count over fail_word by index
```

Listing A.20: Plots the graph in B.3

```
index="brute_force" OR index="normal_events" ("*fail*" AND "*log*")
| rex field=_raw "(?i)(?<keyword_log>\b\w*log\w*\b)"
| rex field=_raw "(?i)(?<keyword_fail>\b\w*fail\w*\b)"
| where isnotnull(keyword_log) AND isnotnull(keyword_fail)
| eval keyword_combined=keyword_log + "+" + keyword_fail
| stats count by keyword_combined, index
| xyseries keyword_combined, index, count
```

Listing A.21: Plots the graph in B.4

```
index="brute_force" OR index="normal_events"
( EventID =675 OR EventID =676 OR EventID =4771 OR EventID =4772 OR
EventID =4822 OR EventID =4823 OR EventID =4824 OR EventID =5040 OR
EventID =5041 OR EventID =5042 OR EventID =6279 OR EventID =4625 OR
EventID =24003 OR EventID =24299 OR EventID =529 OR EventID =530 OR
EventID =531 OR EventID =532 OR EventID =533 OR EventID =534 OR EventID
=535 OR EventID =536 OR EventID =537 OR EventID =539 OR EventID =681 OR
EventID =18456)

| eval SystemTime = strptime(SystemTime,"%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime = strftime(SystemTime,"%Y-%m-%d%H:%M:%S")
| eval bin_time = floor(strptime (SystemTime, "%Y-%m-%d%H:%M:%S")/10) *
   10
| eval bin_time = strftime(bin_time, "%Y-%m-%d%H:%M:%S")
| eventstats count as event_count by bin_time
| stats count as event_count by bin_time index
| where event_count >1
| sort bin_time
| chart sum(event_count) over bin_time by index
```

Listing A.22: Plots the graph shown in B.5

```
index="normal_events"
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval SystemTime=strftime(SystemTime, "%Y-%m-%d %H:%M:%S")
| eval bin_time = floor(strptime(SystemTime, "%Y-%m-%d %H:%M:%S") / 1) *
    1
| eventstats count as event_count by bin_time
| where event_count > 10
| stats count as count_EventID by EventID
| sort -count_EventID
| head 10
```

Listing A.23: Plots the graph shown in B.6

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
```

```
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-04T00:00:00.000000Z", "%Y-%m-%dT
   %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-05T00:00:00.000000
   Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time", date as "Date"
```

Listing A.24: Plots the graph shown in 6.6

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-05T00:00:00.000000Z", "%Y-%m-%dT
   %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-06T00:00:00.000000
   Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time", date as "Date"
```

Listing A.25: Plots the graph shown in 6.7

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-06T00:00:00.000000Z", "%Y-%m-%dT
   %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-07T00:00:00.000000
   Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
```

```
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time"
| rename date as "Count"
```

Listing A.26: Plots the graph shown in 6.8

```
index="brute_force" sourcetype="BF-3-days-dataset"
(EventID=675 OR EventID=676 OR EventID=4771 OR EventID=4772
OR EventID=4822 OR EventID=4823 OR EventID=4824 OR EventID=5040
OR EventID=5041 OR EventID=5042 OR EventID=6279 OR EventID=4625
OR EventID=24003 OR EventID=24299 OR EventID=529
OR EventID=530 OR EventID=531 OR EventID=532 OR EventID=533
OR EventID=534 OR EventID=535 OR EventID=536 OR EventID=537
OR EventID=539 OR EventID=681 OR EventID=18456)
| eval SystemTime=strptime(SystemTime, "%Y-%m-%dT%H:%M:%S.%6NZ")
| where SystemTime >= strptime("2024-06-04T00:00:00.000000Z", "%Y-%m-%dT
  %H:%M:%S.%6NZ") AND SystemTime < strptime("2024-06-07T00:00:00.000000
  Z", "%Y-%m-%dT%H:%M:%S.%6NZ")
| eval date=strftime(SystemTime, "%Y-%m-%d")
| eval time=strftime(SystemTime, "%d %H:%M")
| eval bin_time = floor(strptime(time, "%d %H:%M") / 60) * 60
| eval bin_time = strftime(bin_time, "%d %H:%M")
| stats count as event_count by bin_time date
| sort bin_time
| chart sum(event_count) over bin_time by date
| rename bin_time as "Time", date as "Date"
```

Listing A.27: Plots the graph shown in B.7

```
index="buac"
| stats count by EventID
| sort -count
```

Listing A.28: Plots the ”Bypass User Account Control Events: EventIDs” graph shown in 6.9

```
index=buac ProcessID=*
| stats count by ProcessID
| sort -count
```

Listing A.29: Plots the ”Bypass User Account Control Events: ProcessIDs” graph shown in B.13

```
index=normal_events ProcessID=*
| stats count by ProcessID
| sort -count
| head 10
```

Listing A.30: Plots the ”Normal Events: ProcessIDs” graph shown in B.14

```
index=buac ProcessId=*
| stats count by ProcessId
| sort -count
```

```
| head 10
```

Listing A.31: Plots the ”Bypass User Account Control Events: Top 10 ProcessIds” graph shown in B.15

```
index=normal_events ProcessId=*
| stats count by ProcessId
| sort -count
| head 10
```

Listing A.32: Plots the ”Normal Events: Top 10 ProcessIds” graph shown in B.16

```
index=buac OR index=normal_events SubjectDomainName=*
| stats count by SubjectDomainName, index
| sort -count
| xyseries SubjectDomainName index count
```

Listing A.33: Plots the ”Subject Domain Names” graph shown in B.12

```
index="remote_services" EventID=*
| stats count by EventID
| sort -count
```

Listing A.34: Plots the ”Remote Services Events: EventIDs” graph shown in 6.10

```
index="remote_services" EventID=*
| stats count by EventID
| join type=inner EventID [
    search index="normal_events" EventID=*
    | stats count by EventID
]
| table EventID, count
| sort -count
```

Listing A.35: Plots the ”Attack EventIDs present in Normal Data” graph shown in 6.11

```
index=dwel
| stats count by EventID
| sort -count
```

Listing A.36: Plots the ”Disable Windows Event Logging Events: EventIDs” graph shown in 6.12

```
index="dwel" EventID=*
| stats count by EventID
| join type=inner EventID [
    search index="normal_events" EventID=*
    | stats count by EventID
]
| table EventID, count
| sort -count
```

Listing A.37: Plots the ”Disable Windows Event Logging EventIDs present in Normal Data” graph in 6.13

```
index=dwel OR index=normal_events
| eval Keyword=if(match(_raw, "(?i)hackerserver"), "hackerserver",
    if(match(_raw, "(?i)DomainPolicyChanged"), "DomainPolicyChanged",
    if(match(_raw, "(?i)AuditPolicyChanges"), "AuditPolicyChanges",
    if(match(_raw, "(?i)EnableScriptBlockLogging"),
    "EnableScriptBlockLogging",
    if(match(_raw, "(?i)powershell"), "powershell",
    if(match(_raw, "(?i)clear"), "Clear",
    if(match(_raw, "(?i)disable"), "Disable",
     null()))))))))
| search Keyword!=null
| chart count by Keyword, index
| sort -count
```

Listing A.38: Plots the "Keywords by Index" graph in 6.14

```
index="normal_events"
| stats count by Channel
| sort -count
```

Listing A.39: Plots the "Normal Events: Channels" graph in 6.15

```
index="dwel"
| stats count by Channel
| sort -count
```

Listing A.40: Plots the "Disable Windows Event Logging Events: Channels" graph in 6.16

```
index="dwel" OR index="normal_events"
| chart count over CommandLine by index
| sort -count
```

Listing A.41: Plots the "Disable Windows Event Logging and Normal Events: CommandLine" graph in 6.17

A.2 Files

```
 <Events><Event xmlns='http://schemas.microsoft.com/win/2004/08/
events/event'><System><Provider Name='OpenSSH' Guid='{c4b57d35-0636-4
bc3-a262-370f249f9802}'/><EventID>4</EventID><Version>0</Version><
Level>4</Level><Task>0</Task><Opcode>0</Opcode><Keywords>0
x4000000000000000</Keywords><TimeCreated SystemTime='2021-05-20T12
:49:54.9544162Z'/><EventRecordID>21</EventRecordID><Correlation/><
Execution ProcessID='5848' ThreadID='4412'/><Channel>OpenSSH/
Operational</Channel><Computer>fs01.offsec.lan</Computer><Security
UserID='S-1-5-18'/></System><EventData><Data Name='process'>sshd</
Data><Data Name='payload'>Disconnecting invalid user hacker
10.23.23.9 port 60096: Too many authentication failures [preauth]</
Data></EventData></Event><Event xmlns='http://schemas.microsoft.com/
win/2004/08/events/event'><System><Provider Name='OpenSSH' Guid='{
c4b57d35-0636-4bc3-a262-370f249f9802}'/><EventID>4</EventID><Version
>0</Version><Level>4</Level><Task>0</Task><Opcode>0</Opcode><Keywords
>0x4000000000000000</Keywords><TimeCreated SystemTime='2021-05-20T12
:49:54.9457408Z'/><EventRecordID>20</EventRecordID><Correlation/><
```

```
Execution ProcessID='5848' ThreadID='4412'/><Channel>OpenSSH/
Operational</Channel><Computer>fs01.offsec.lan</Computer><Security
UserID='S-1-5-18'/></System><EventData><Data Name='process'>sshd</
Data><Data Name='payload'>Failed password for invalid user hacker
from 10.23.23.9 port 60096 ssh2</Data></EventData></Event>

 ...

 <Event xmlns='http://schemas.microsoft.com/win/2004/08/events/event
'><System><Provider Name='OpenSSH' Guid='{c4b57d35-0636-4bc3-a262-370
f249f9802}'/><EventID>4</EventID><Version>0</Version><Level>4</Level
><Task>0</Task><Opcode>0</Opcode><Keywords>0x4000000000000000</
Keywords><TimeCreated SystemTime='2021-05-20T12:49:51.6281728Z'/><
EventRecordID>15</EventRecordID><Correlation/><Execution ProcessID
='5848' ThreadID='4412'/><Channel>OpenSSH/Operational</Channel><
Computer>fs01.offsec.lan</Computer><Security UserID='S-1-5-18'/></
System><EventData><Data Name='process'>sshd</Data><Data Name='payload
'>Invalid user hacker from 10.23.23.9 port 60096</Data></EventData></
Event></Events>
```

Listing A.42: The same context as A.43 but not yet processed by the python script A.1

```
 <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event
"><System><Provider Name="OpenSSH" Guid="{c4b57d35-0636-4bc3-a262-370
f249f9802}" /><EventID>4</EventID><Version>0</Version><Level>4</Level
><Task>0</Task><Opcode>0</Opcode><Keywords>0x4000000000000000</
Keywords><TimeCreated SystemTime="2021-05-20T12:49:54.9544162Z" /><
EventRecordID>21</EventRecordID><Correlation /><Execution ProcessID
="5848" ThreadID="4412" /><Channel>OpenSSH/Operational</Channel><
Computer>fs01.offsec.lan</Computer><Security UserID="S-1-5-18" /></
System><EventData><Data Name="process">sshd</Data><Data Name="payload
">Disconnecting invalid user hacker 10.23.23.9 port 60096: Too many
authentication failures [preauth]</Data></EventData></Event>

 <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event
"><System><Provider Name="OpenSSH" Guid="{c4b57d35-0636-4bc3-a262-370
f249f9802}" /><EventID>4</EventID><Version>0</Version><Level>4</Level
><Task>0</Task><Opcode>0</Opcode><Keywords>0x4000000000000000</
Keywords><TimeCreated SystemTime="2021-05-20T12:49:54.9457408Z" /><
EventRecordID>20</EventRecordID><Correlation /><Execution ProcessID
="5848" ThreadID="4412" /><Channel>OpenSSH/Operational</Channel><
Computer>fs01.offsec.lan</Computer><Security UserID="S-1-5-18" /></
System><EventData><Data Name="process">sshd</Data><Data Name="payload
">Failed password for invalid user hacker from 10.23.23.9 port 60096
ssh2</Data></EventData></Event>

 ...

 <Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event
"><System><Provider Name="OpenSSH" Guid="{c4b57d35-0636-4bc3-a262-370
f249f9802}" /><EventID>4</EventID><Version>0</Version><Level>4</Level
><Task>0</Task><Opcode>0</Opcode><Keywords>0x4000000000000000</
Keywords><TimeCreated SystemTime="2021-05-20T12:49:51.6281728Z" /><
EventRecordID>15</EventRecordID><Correlation /><Execution ProcessID
="5848" ThreadID="4412" /><Channel>OpenSSH/Operational</Channel><
Computer>fs01.offsec.lan</Computer><Security UserID="S-1-5-18" /></
System><EventData><Data Name="process">sshd</Data><Data Name="payload
```

```
">Invalid user hacker from 10.23.23.9 port 60096</Data></EventData></
Event>
```

Listing A.43: The same context as A.42 but processed by the python script A.1

Appendix B

Figures

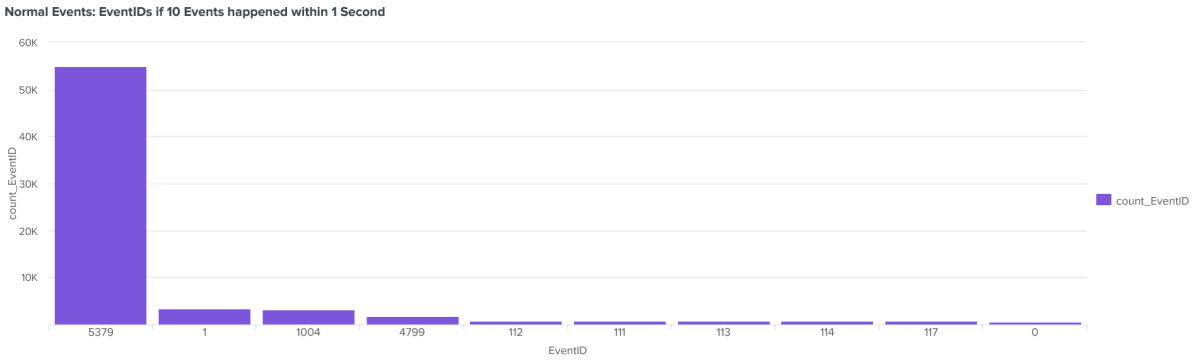


Figure B.1: Normal EventIDs if within one second more than ten events took place

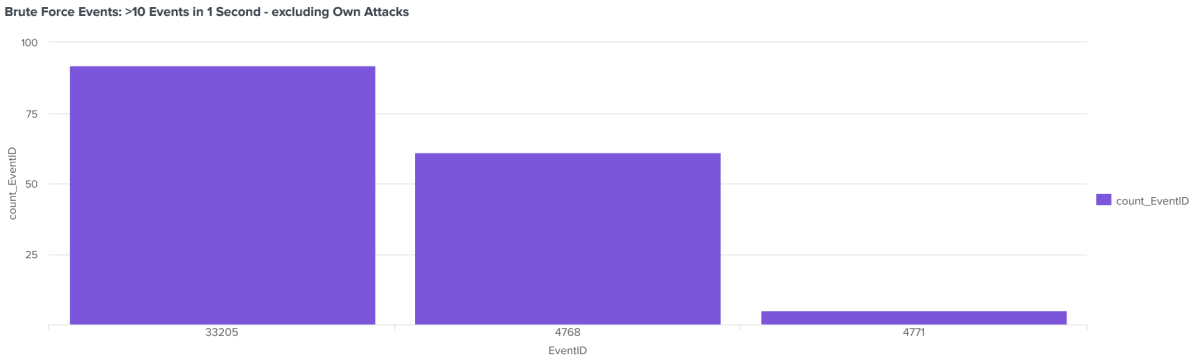


Figure B.2: Shows all EventIDs if within one second more than ten events took place excluding the own Password Guessing and Password Spraying attack events

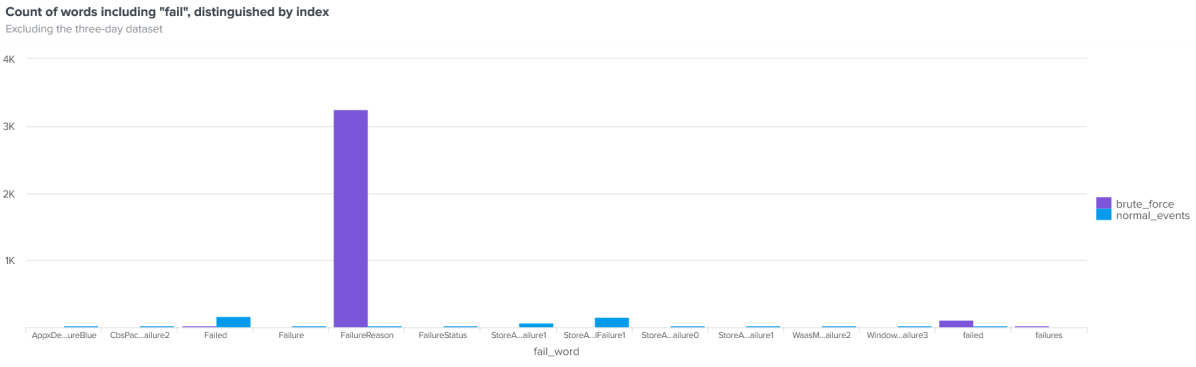


Figure B.3: Count of words including “fail”, distinguished by normal and Brute Force events.

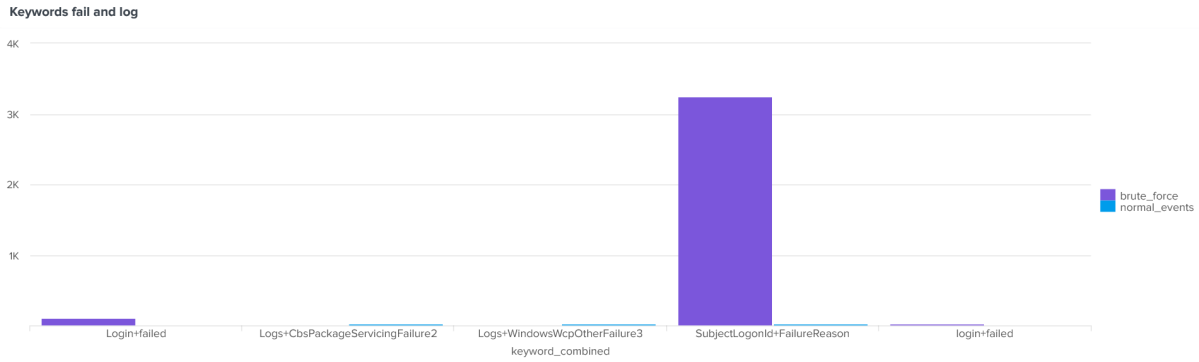


Figure B.4: Events with words including “fail” and “log”, distinguished by index

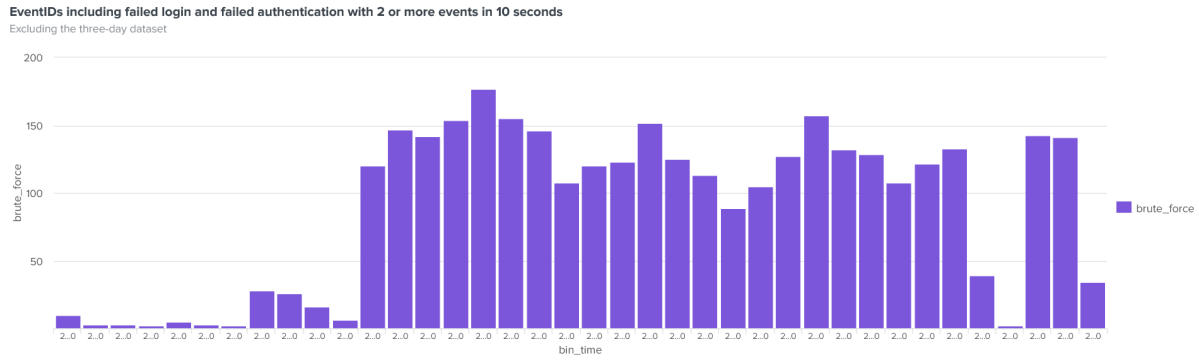


Figure B.5: Filtered EventIDs including words like fail and login with more than one event in ten seconds.

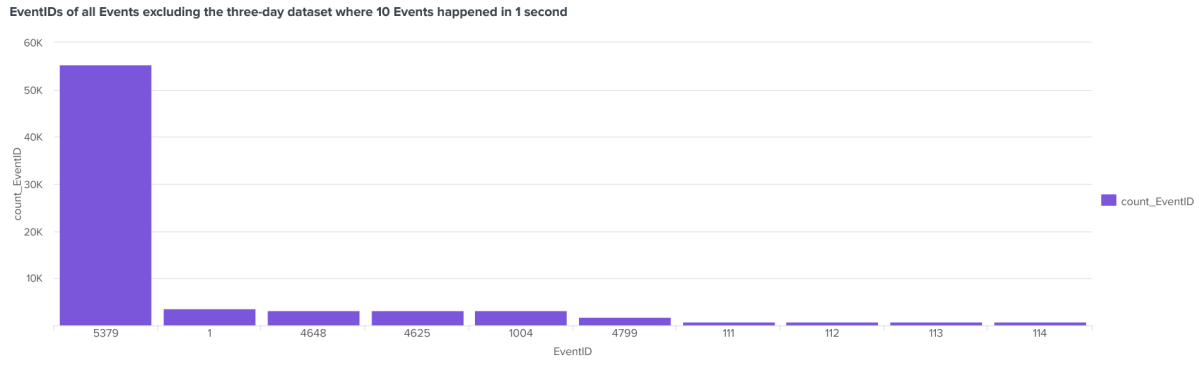


Figure B.6: EventIDs of all events excluding the three-day dataset where more than ten events happened within one second

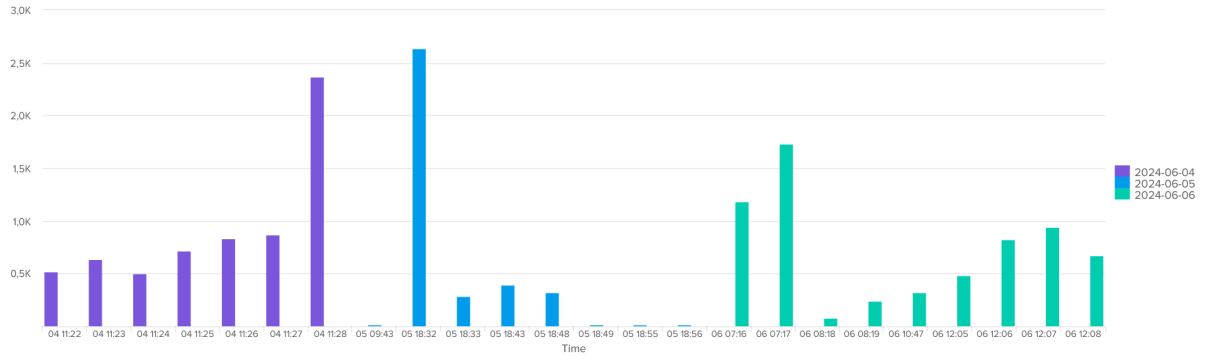
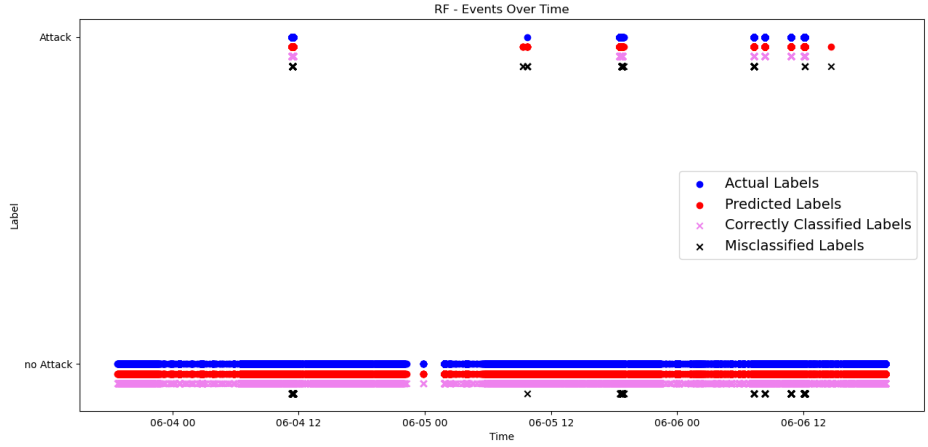
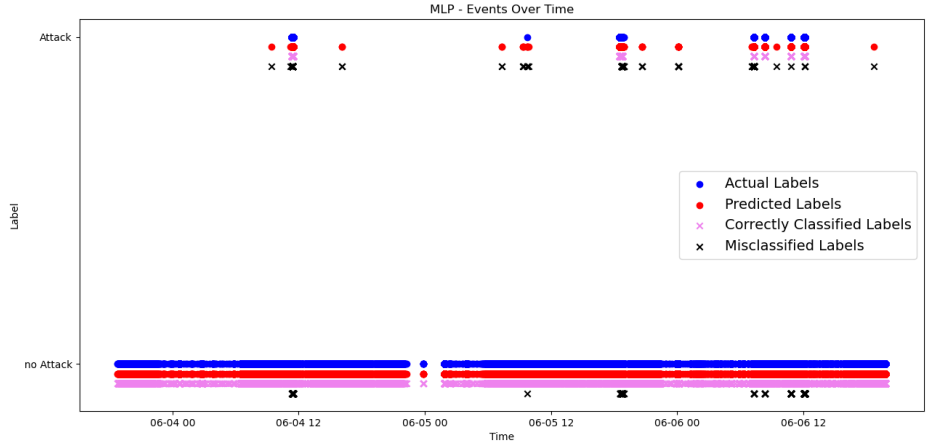


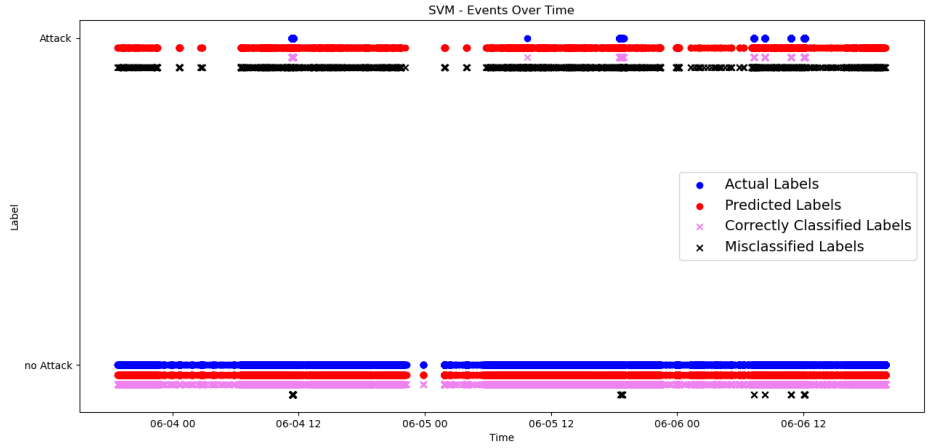
Figure B.7: All three days filtered for the EventIDs noted in 5.4 and split into one minute buckets, each day has a color



(a) Random Forest

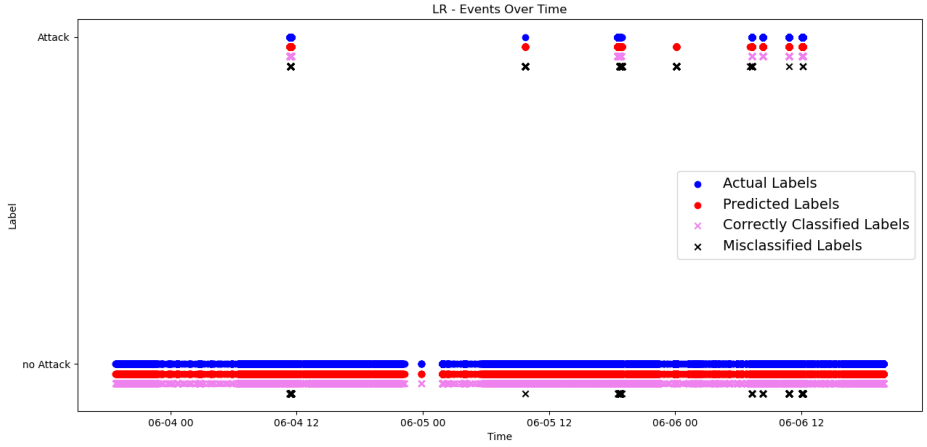


(b) MLP

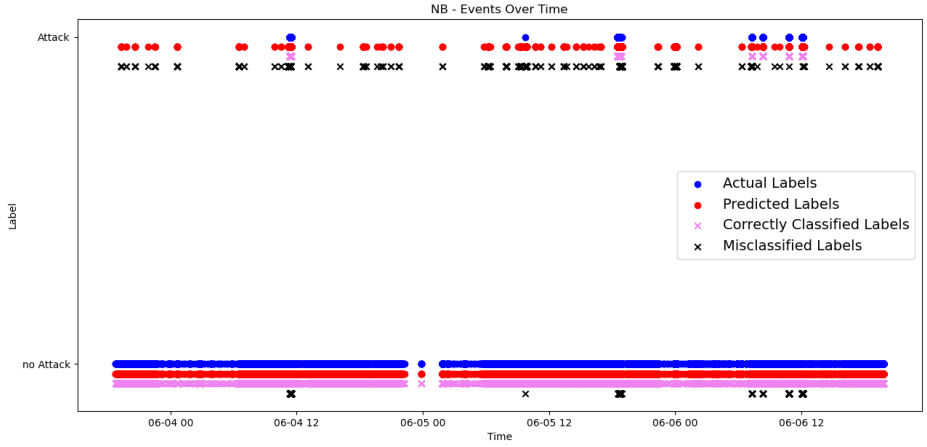


(c) SVM

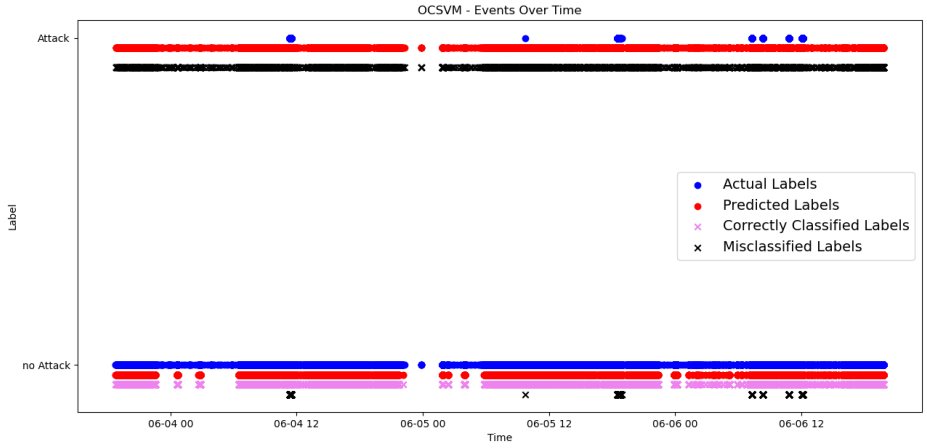
Figure B.8: Results of the prediction of all Brute Force ML approaches with their correctly and misclassified labels (1/2)



(d) Logistic Regression

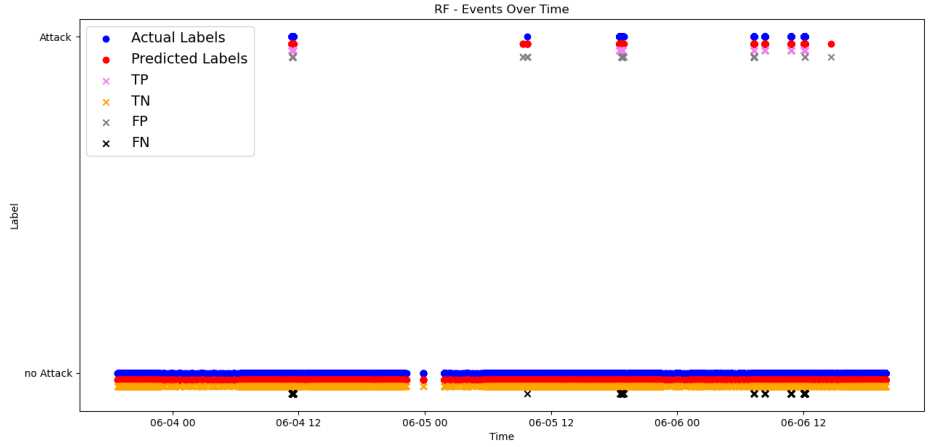


(e) Naive Bayes

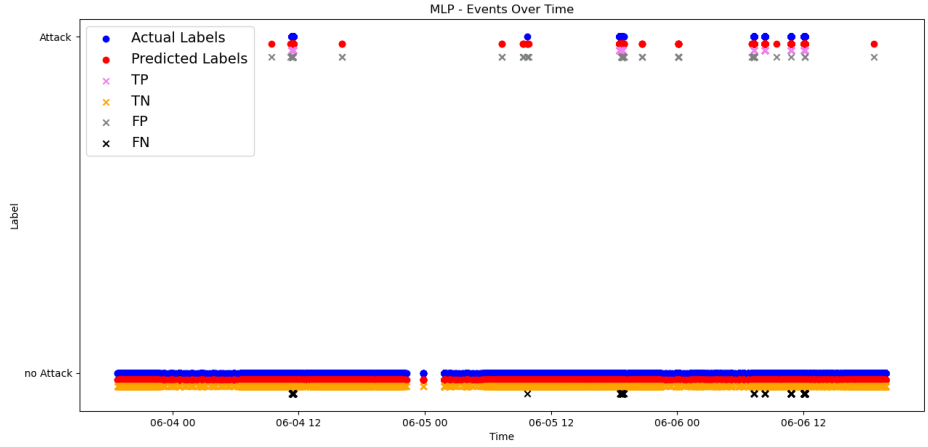


(f) OCSVM

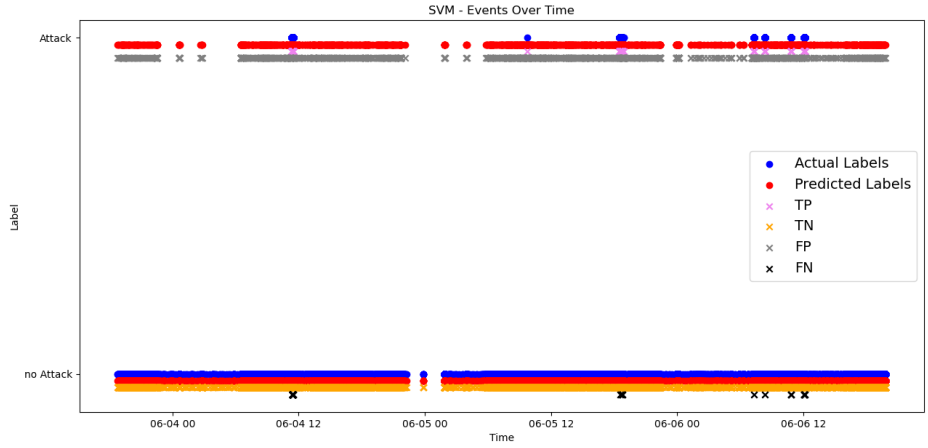
Figure B.9: Results of the prediction of all Brute Force ML approaches with their correctly and misclassified labels (2/2)



(a) Random Forest

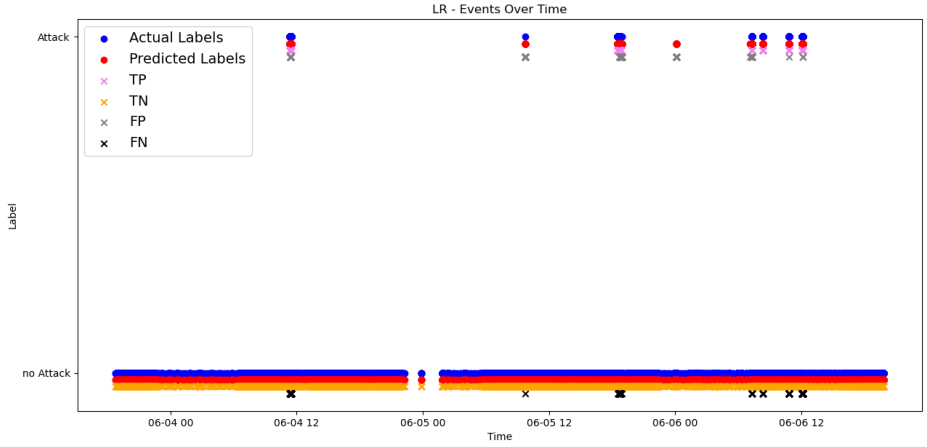


(b) MLP

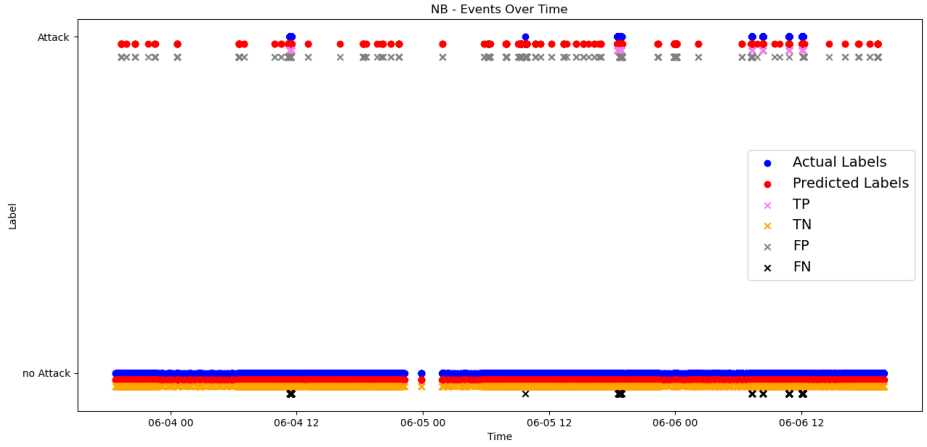


(c) SVM

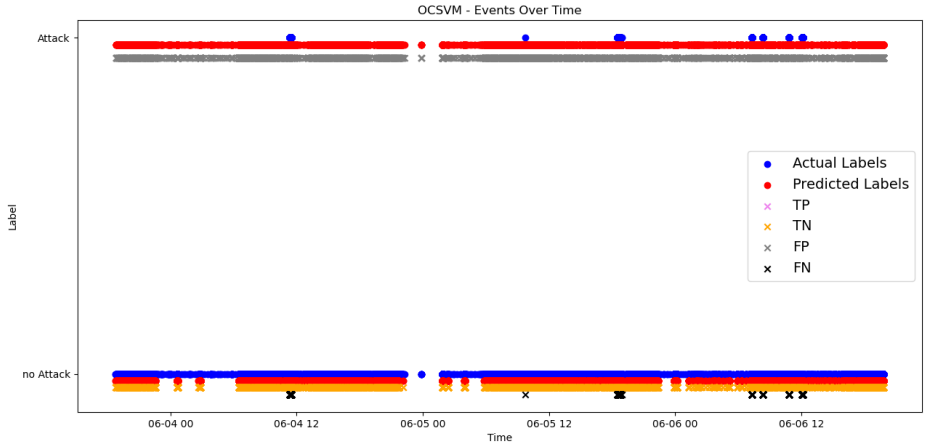
Figure B.10: Results of the TP, TN, FP, and FN predictions of selected Brute Force ML approaches with their correctly and misclassified labels (1/2)



(d) Logistic Regression



(e) Naive Bayes



(f) OCSVM

Figure B.11: Results of the TP, TN, FP, and FN predictions of selected Brute Force ML approaches with their correctly and misclassified labels (2/2)

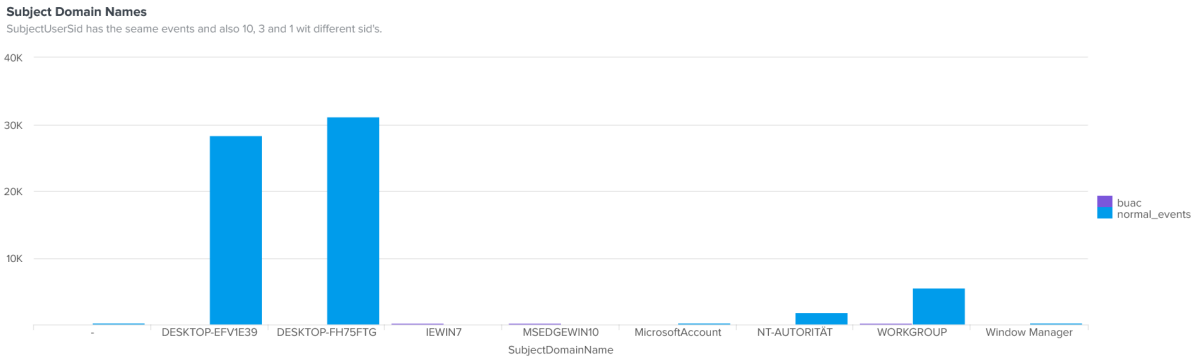


Figure B.12: SubjectDomainNames of the Bypass User Account Control attack events and normal events, the color shows the indices

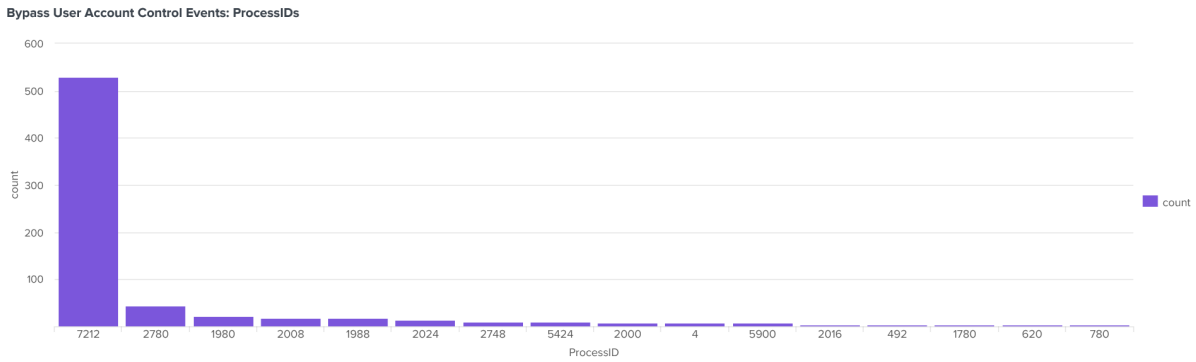


Figure B.13: All ProcessID values of the Bypass User Account Control (corresponding code in Listing A.29)

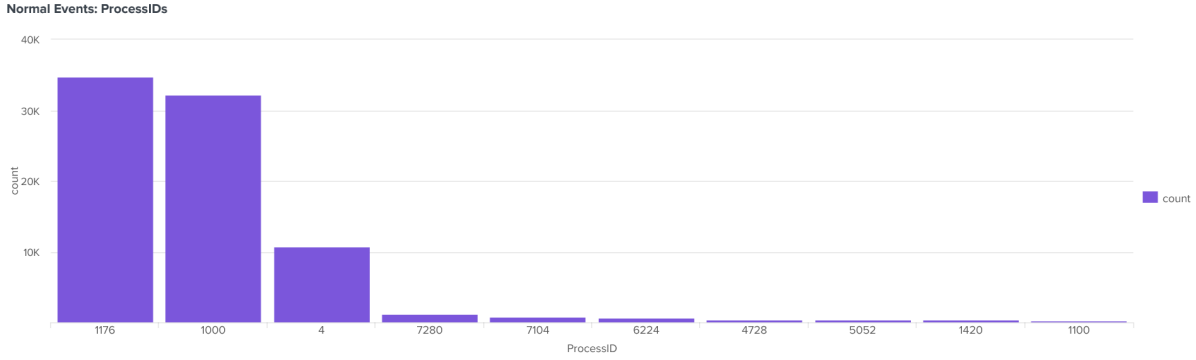


Figure B.14: Top ten ProcessID values of the normal events (the code for this plot is given in Listing A.30)

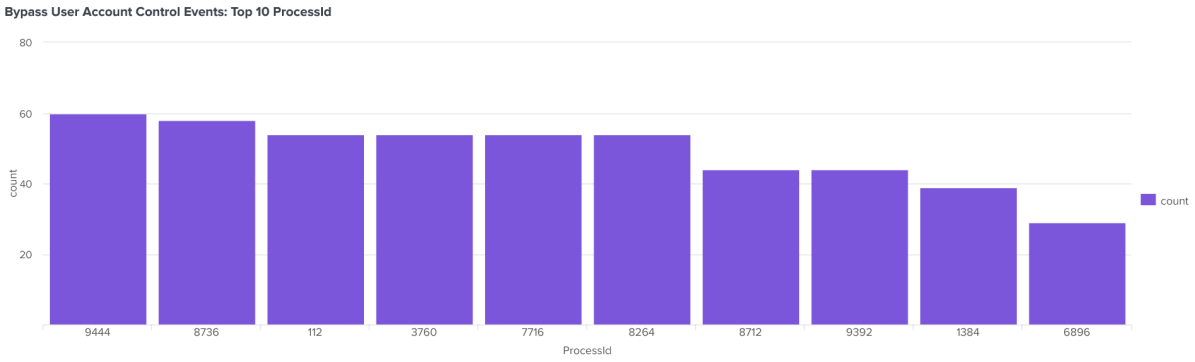


Figure B.15: Top ten ProcessId values of the attack events (code in Listing A.31)

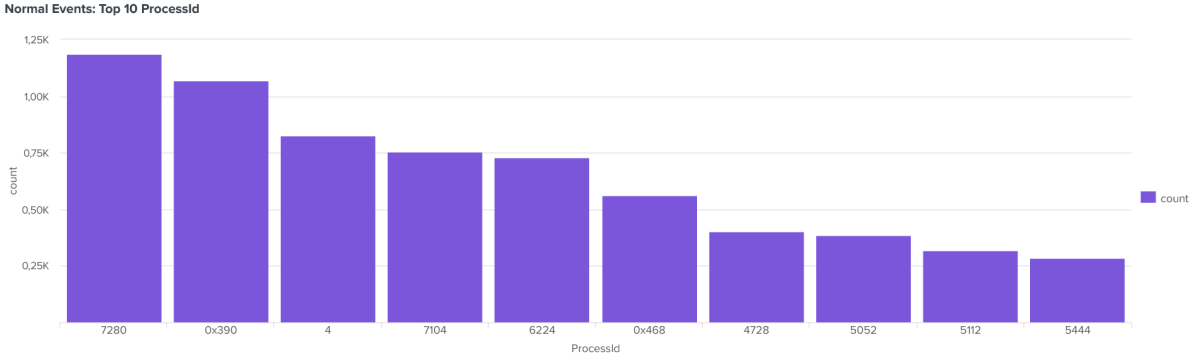
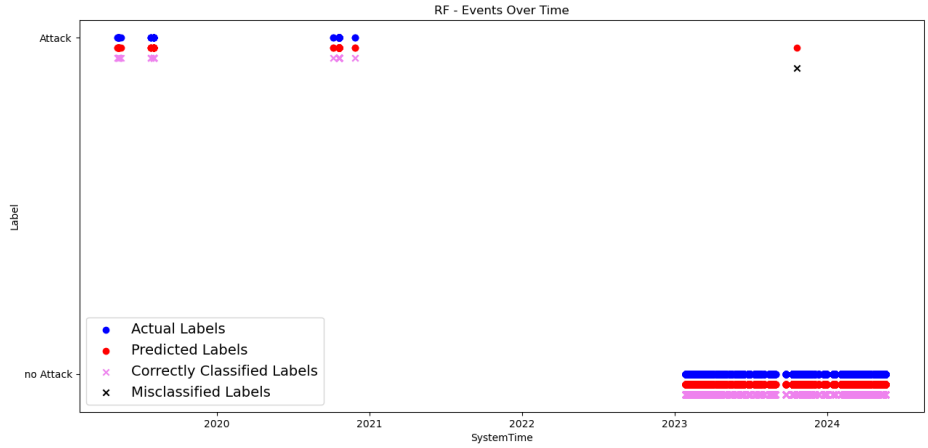
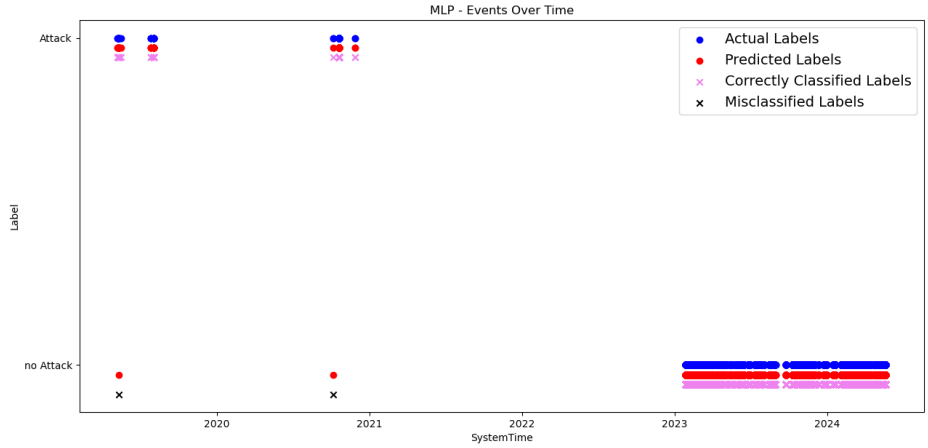


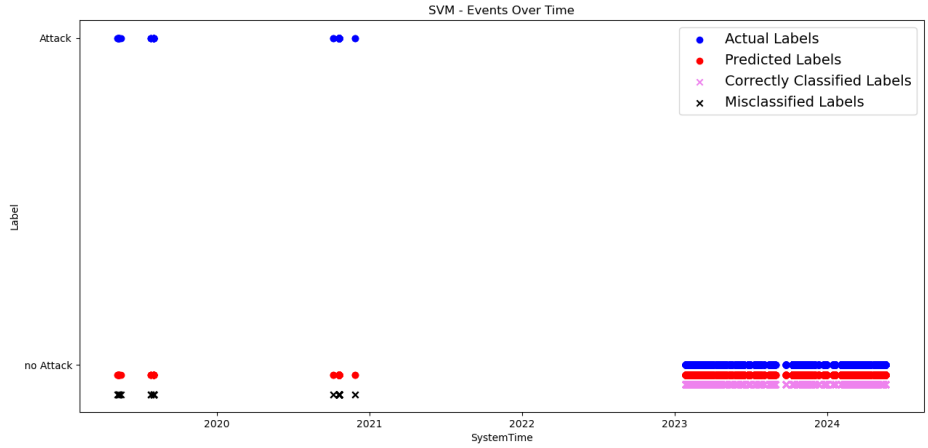
Figure B.16: Top ten ProcessId values of the normal events (find the corresponding code in Listing A.32)



(a) Random Forest

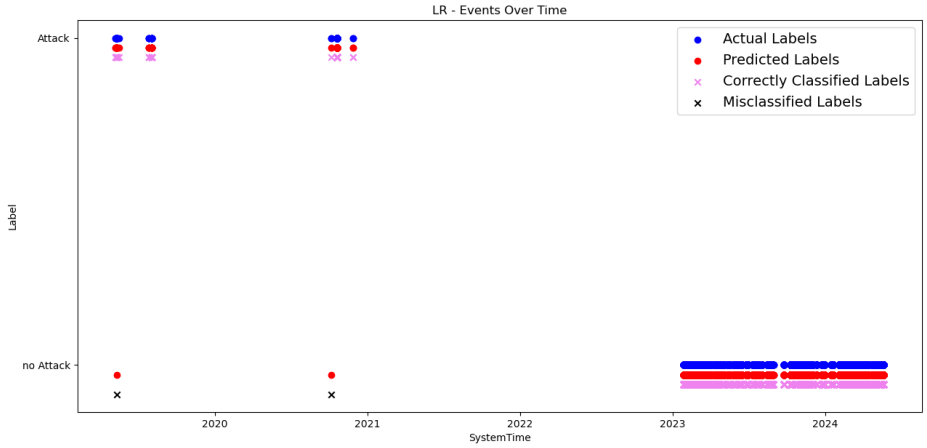


(b) MLP

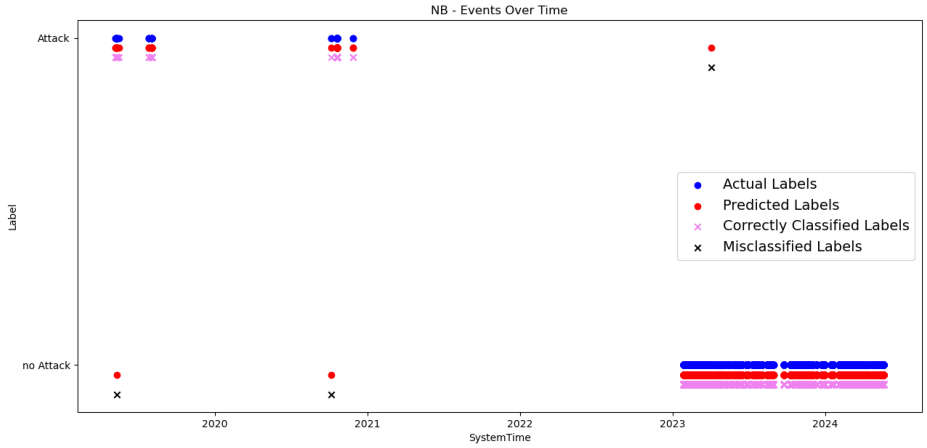


(c) SVM

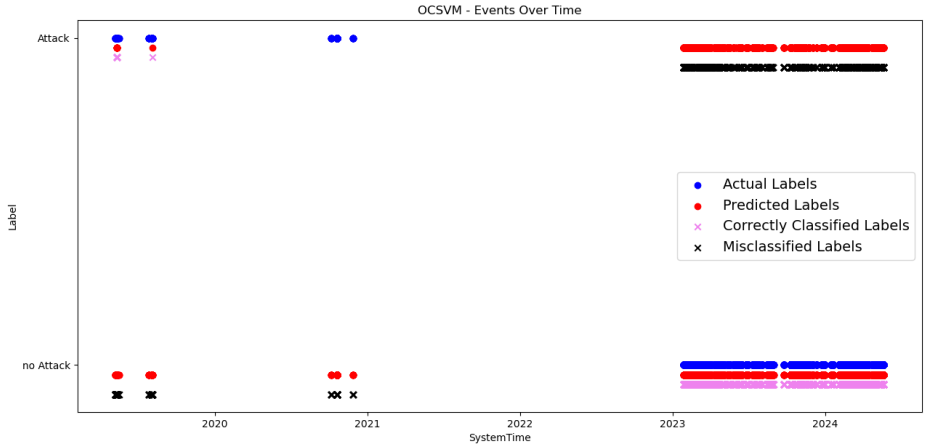
Figure B.17: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Bypass User Account Control attack scenario (1/2)



(d) Logistic Regression

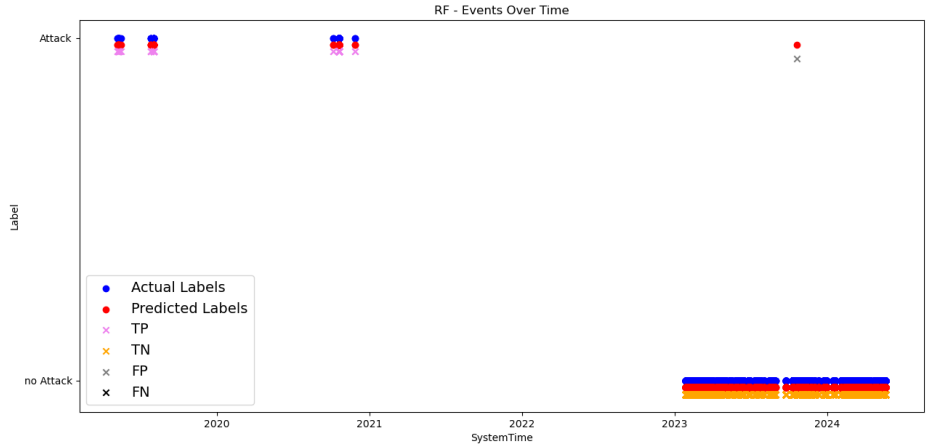


(e) Naive Bayes

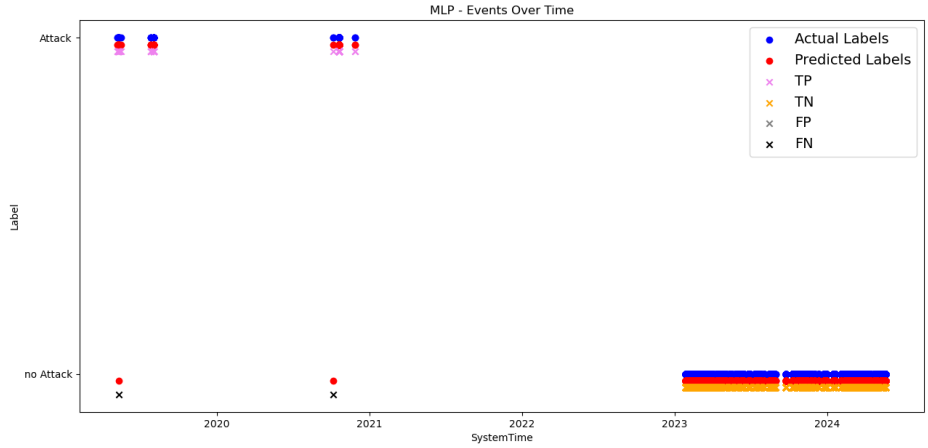


(f) OCSVM

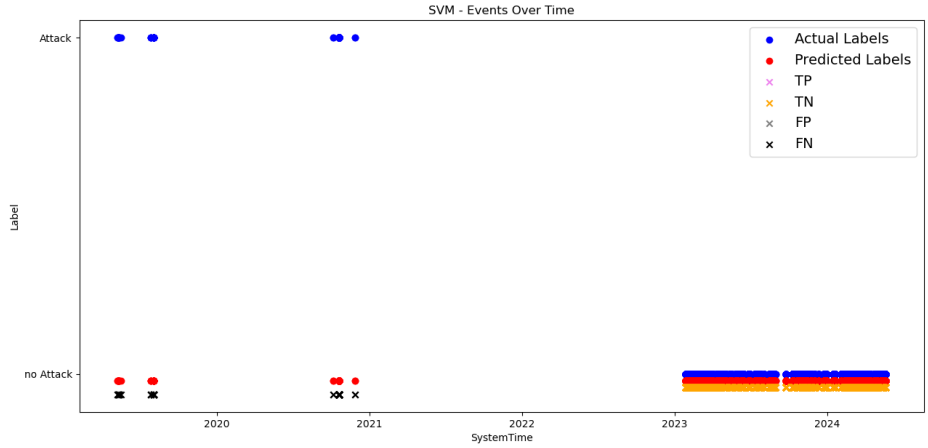
Figure B.18: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Bypass User Account Control attack scenario (2/2)



(a) Random Forest

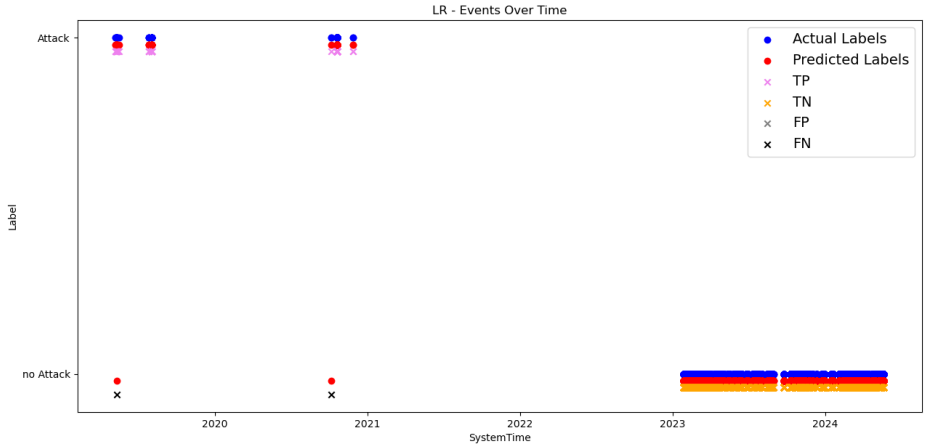


(b) MLP

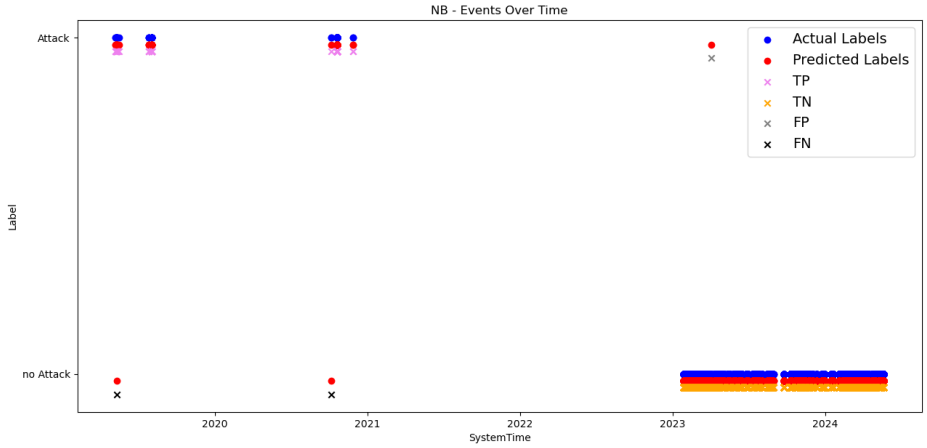


(c) SVM

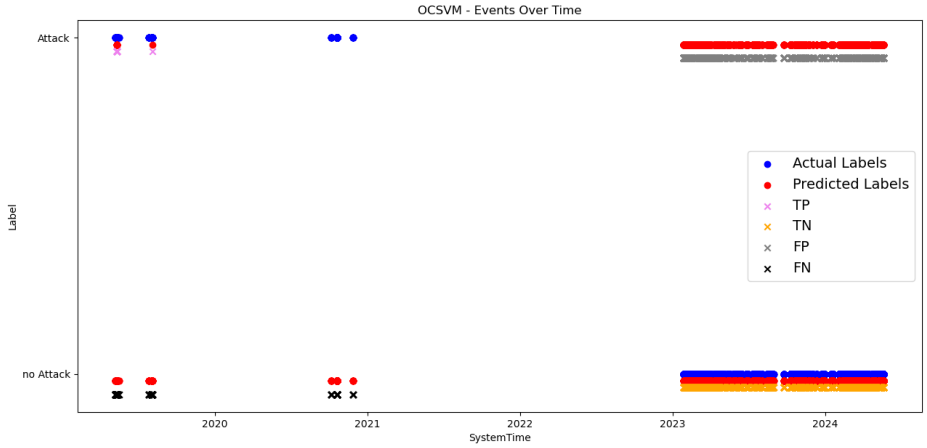
Figure B.19: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Bypass User Account Control attack (1/2)



(d) Logistic Regression

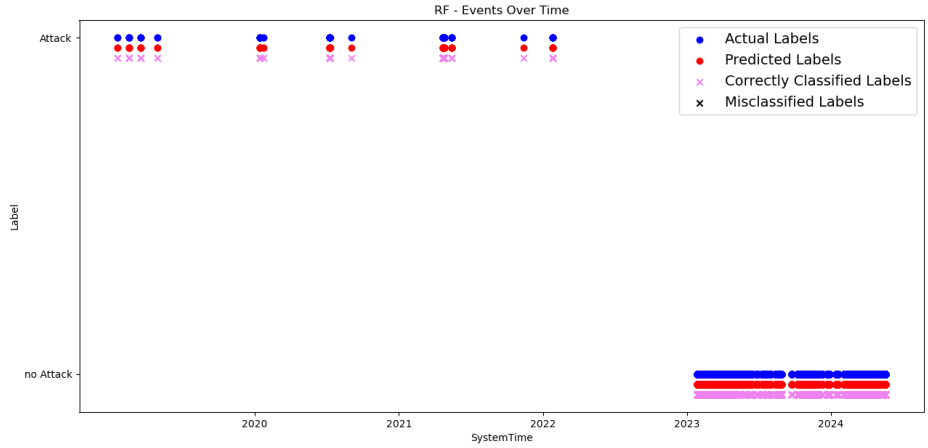


(e) Naive Bayes

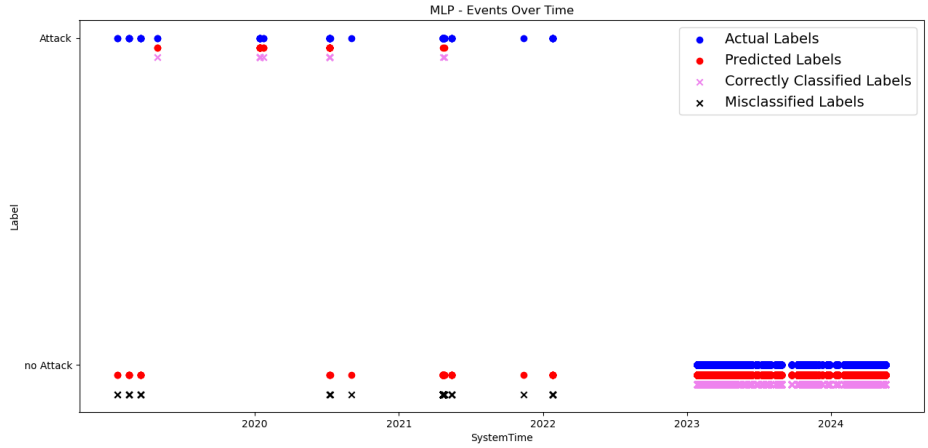


(f) OCSVM

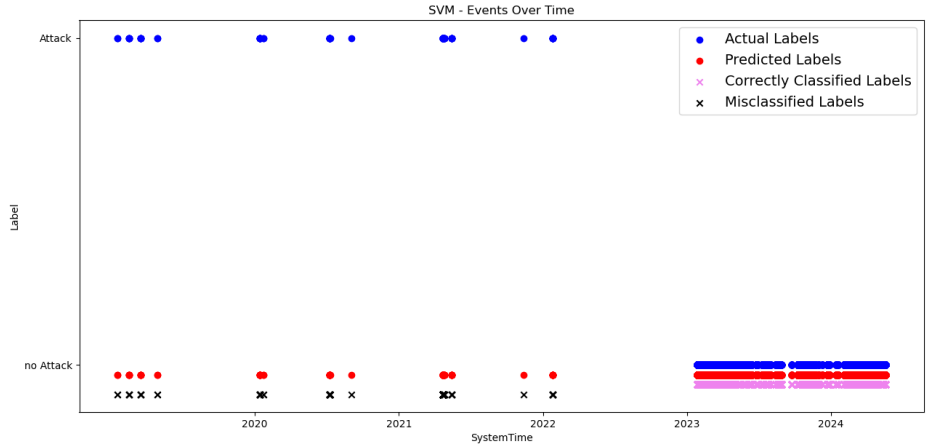
Figure B.20: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Bypass User Account Control attack (2/2)



(a) Random Forest

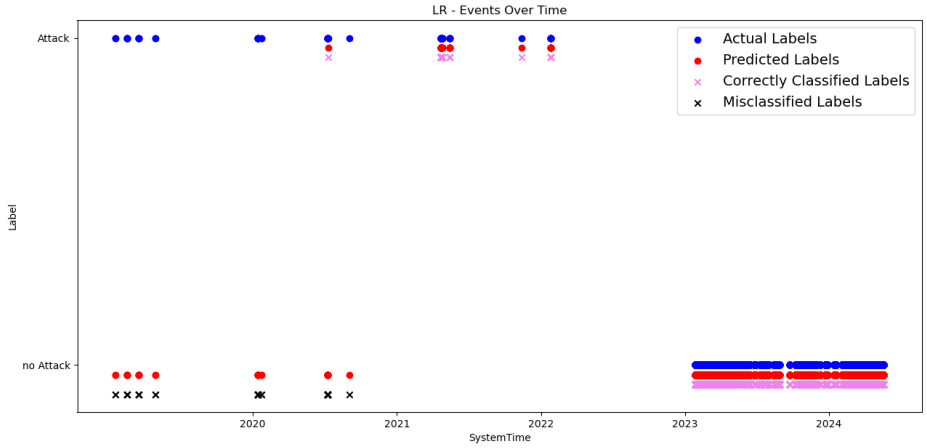


(b) MLP

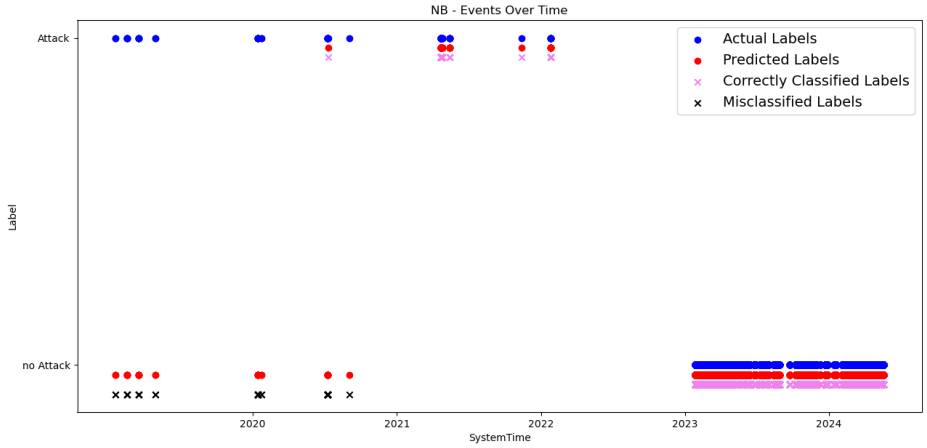


(c) SVM

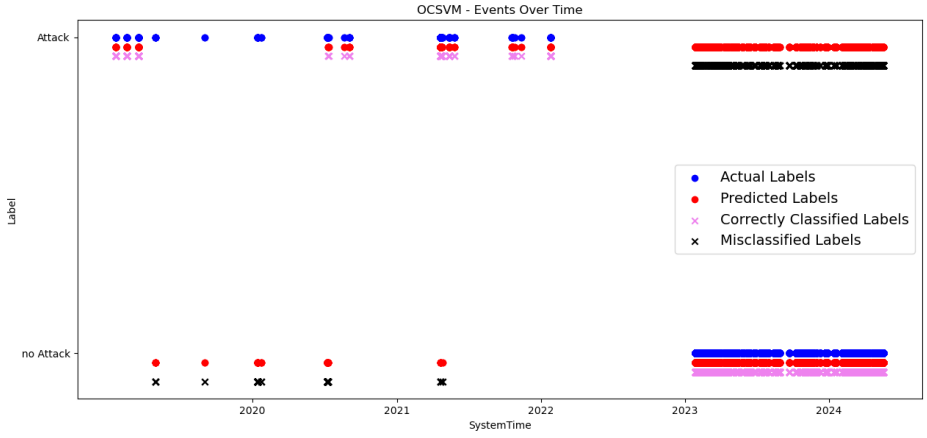
Figure B.21: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Remote Services attack scenario (1/2)



(d) Logistic Regression

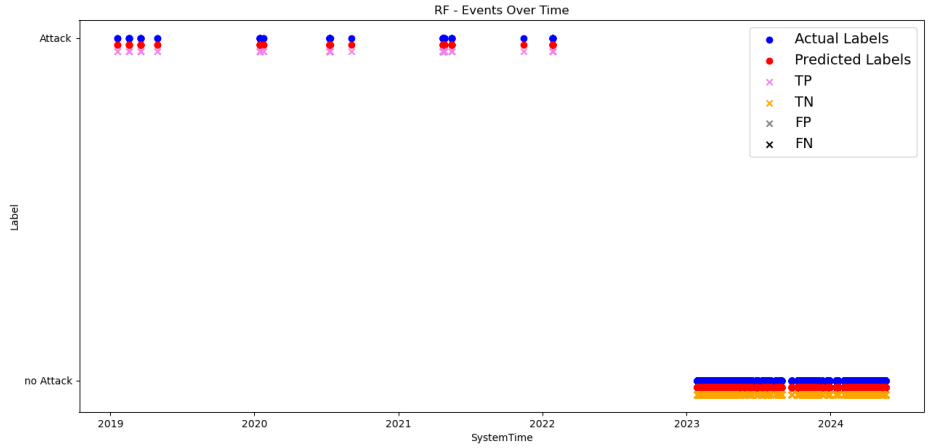


(e) Naive Bayes

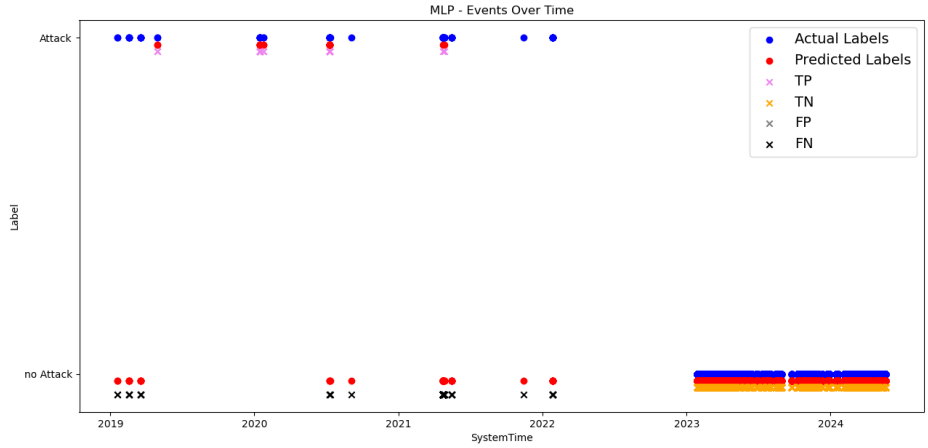


(f) OCSVM

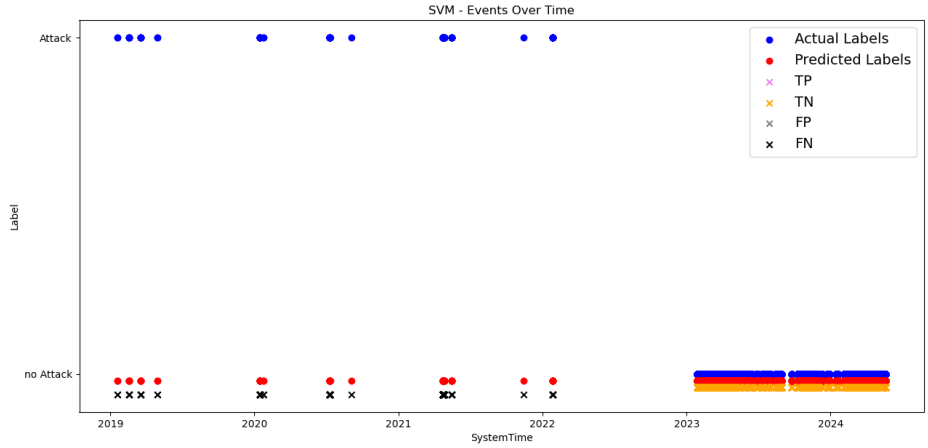
Figure B.22: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Remote Services attack scenario (2/2)



(a) Random Forest

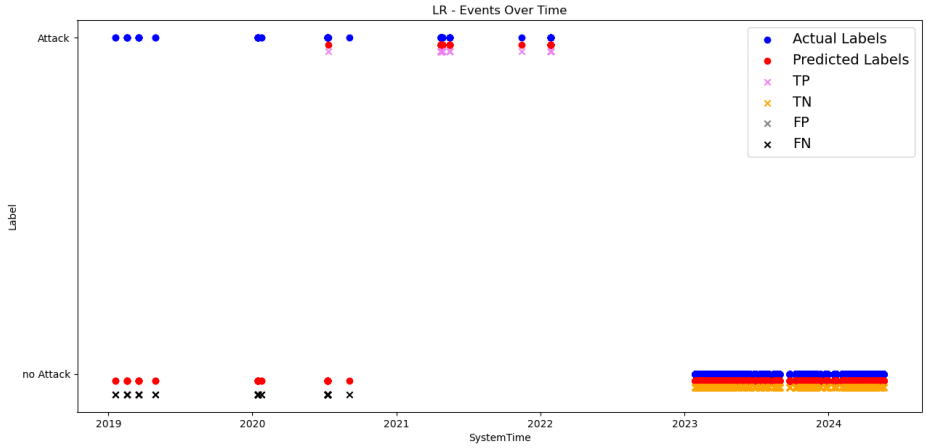


(b) MLP

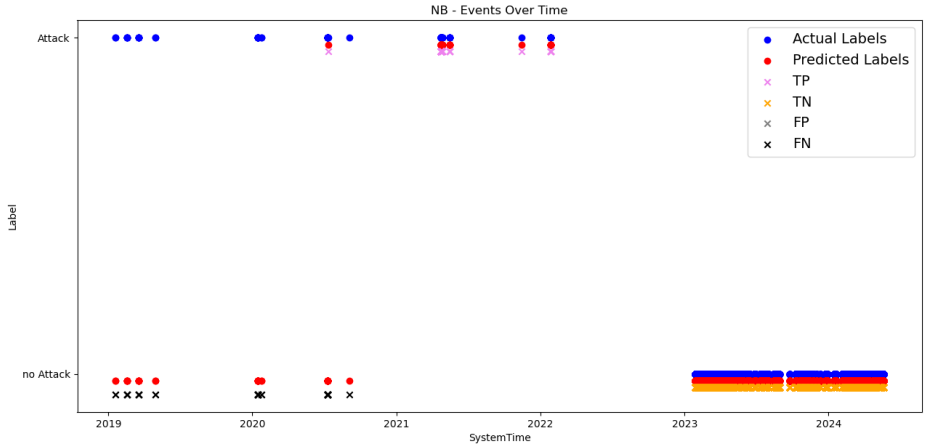


(c) SVM

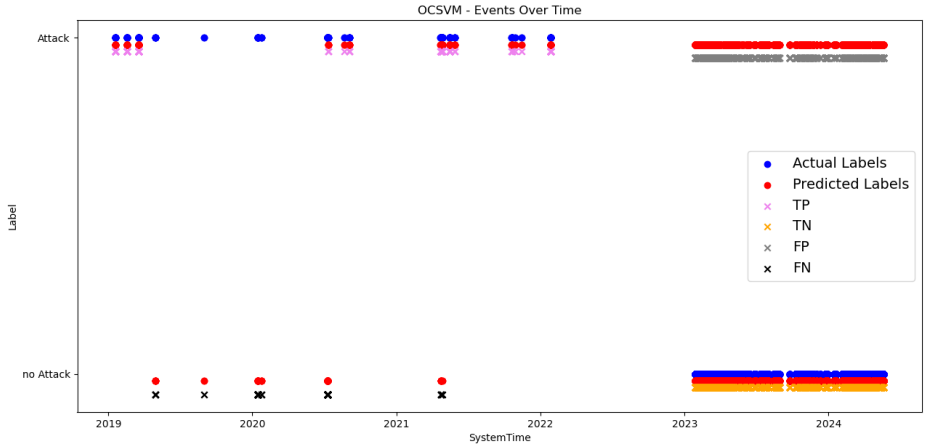
Figure B.23: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Remote Services attack (1/2)



(d) Logistic Regression

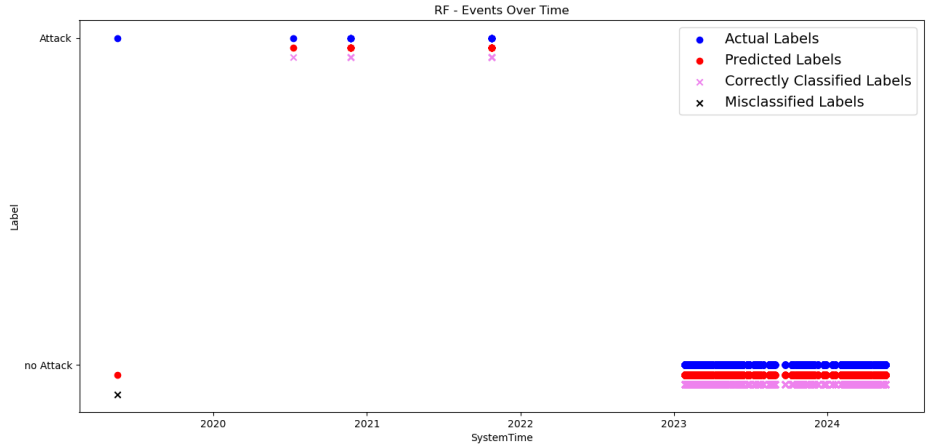


(e) Naive Bayes

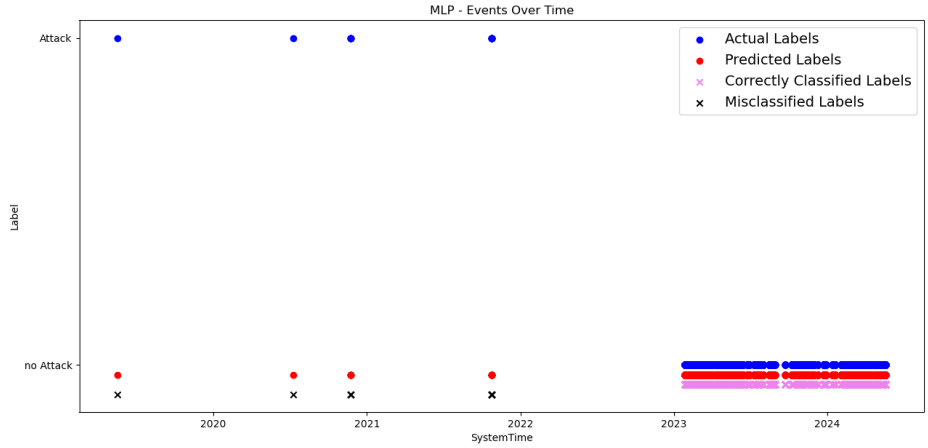


(f) OCSVM

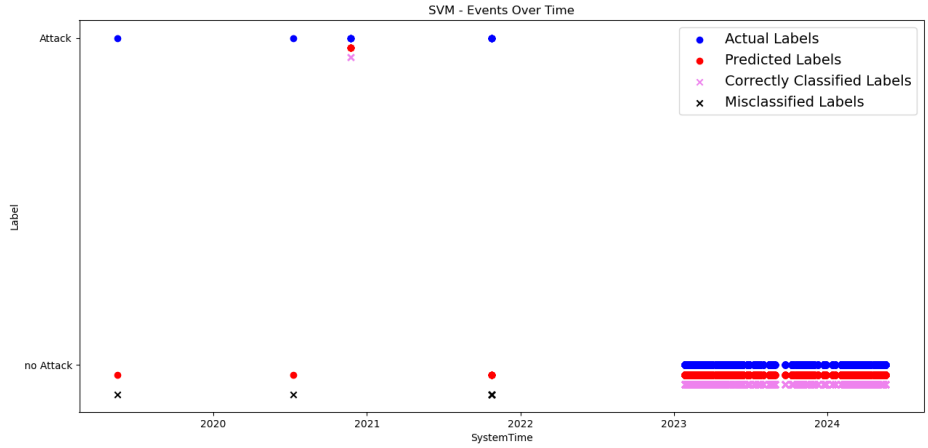
Figure B.24: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Remote Services attack (2/2)



(a) Random Forest

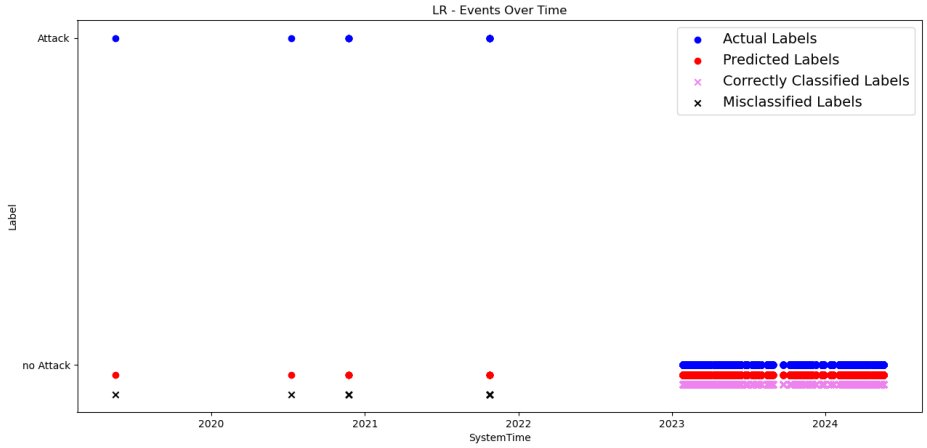


(b) MLP

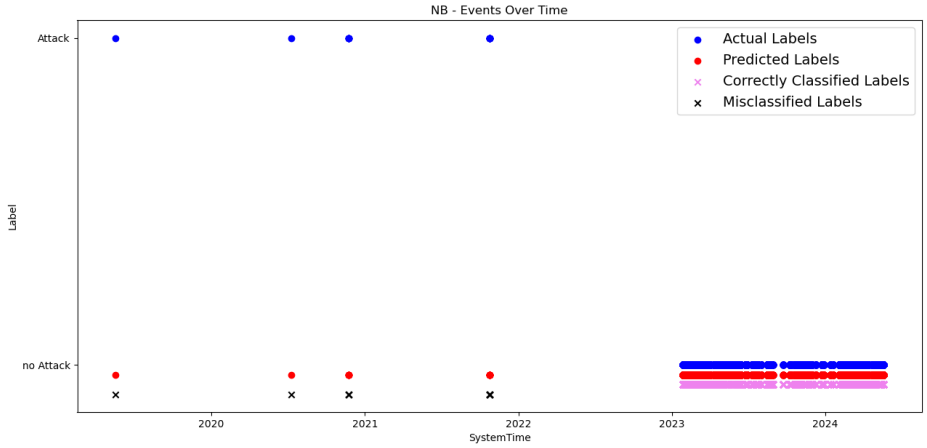


(c) SVM

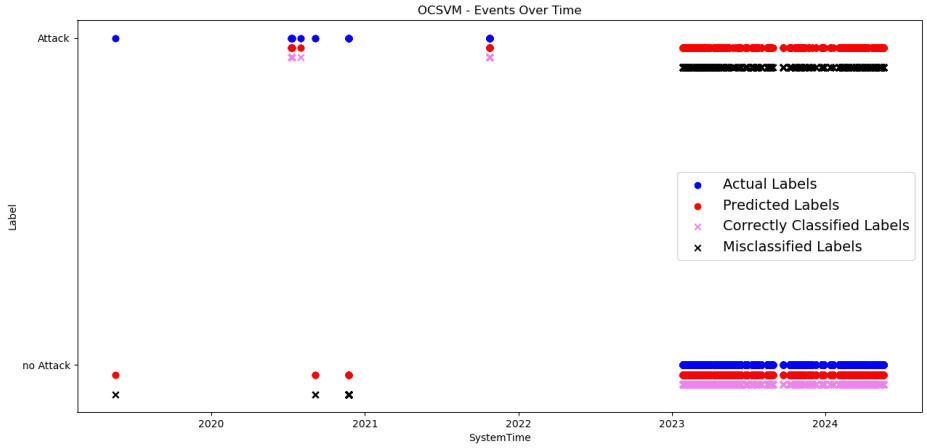
Figure B.25: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Disable Windows Event Logging attack scenario (1/2)



(d) Logistic Regression

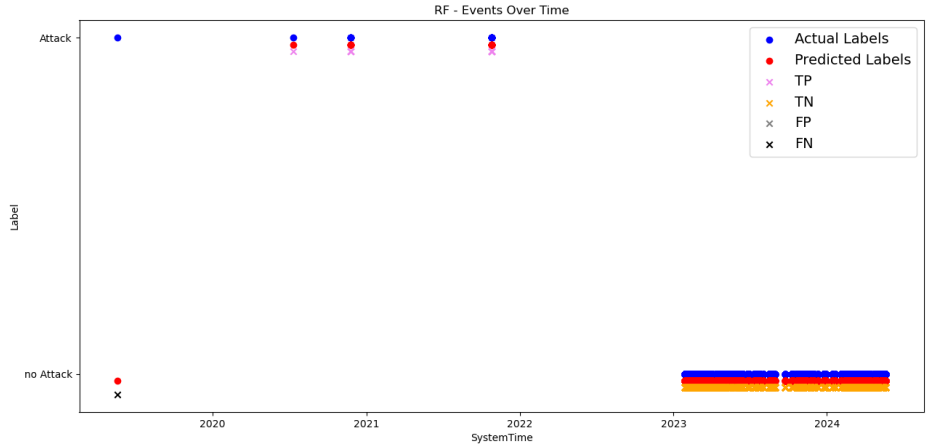


(e) Naive Bayes

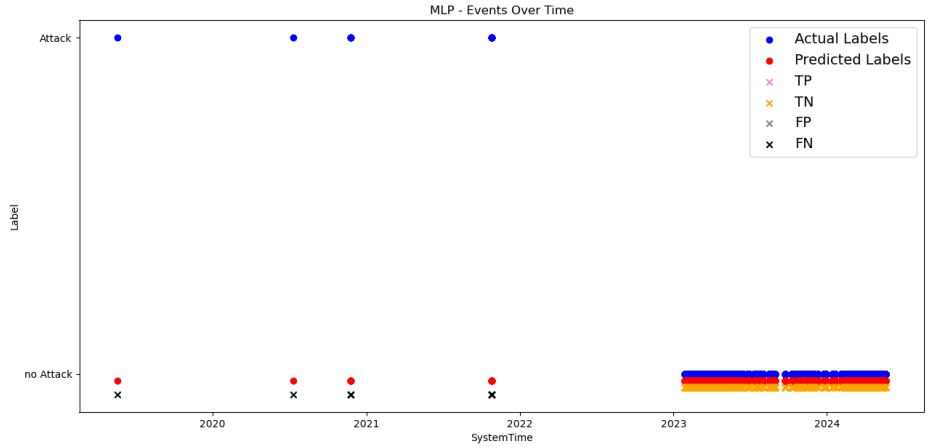


(f) OCSVM

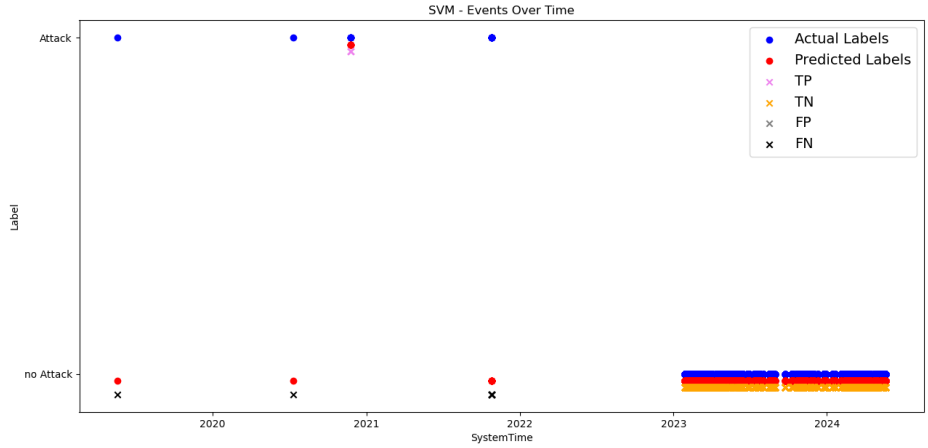
Figure B.26: Results of the prediction of all ML approaches with their correctly and misclassified labels for the Disable Windows Event Logging attack scenario (2/2)



(a) Random Forest

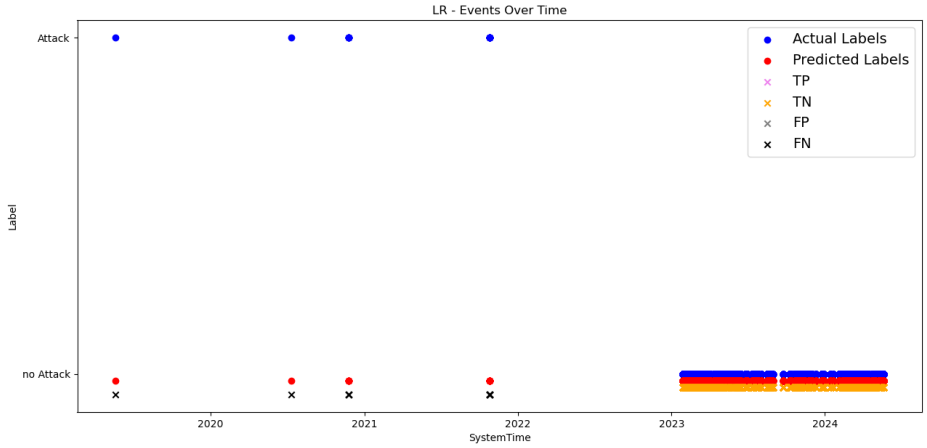


(b) MLP

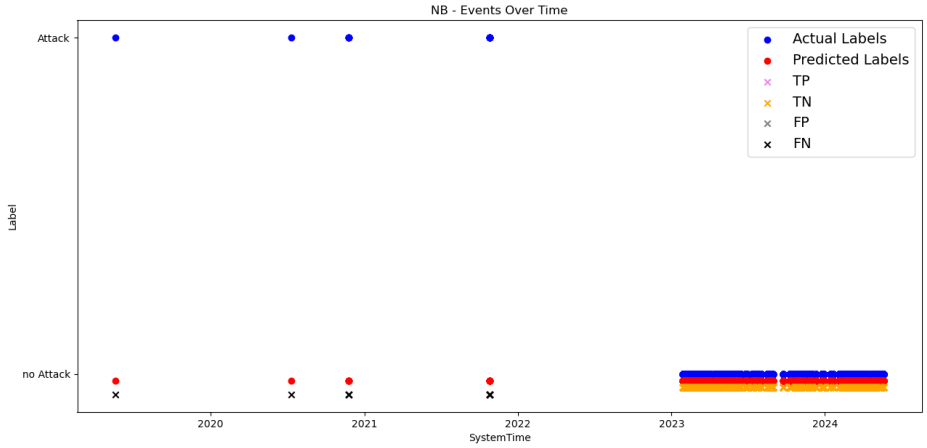


(c) SVM

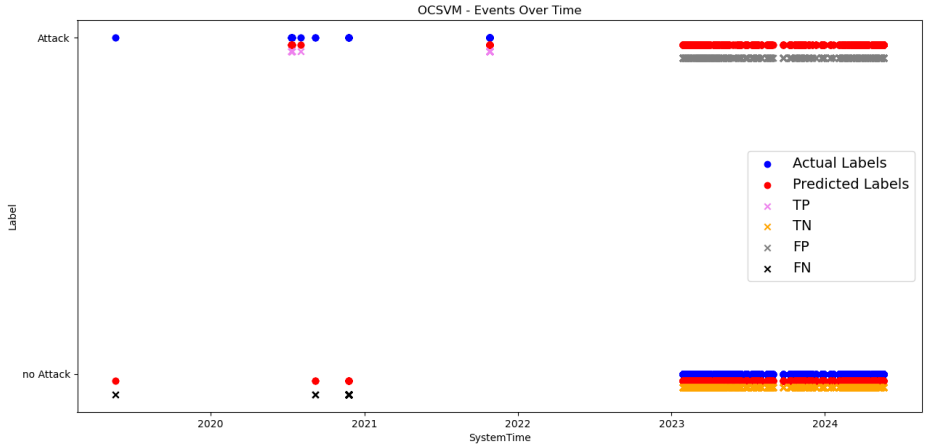
Figure B.27: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Disable Windows Event Loggings attack (1/2)



(d) Logistic Regression



(e) Naive Bayes



(f) OCSVM

Figure B.28: Results of the TP, TN, FP, and FN predictions of all ML approaches with their correctly and misclassified labels of the Disable Windows Event Logging attack (2/2)