Hasso-Plattner-Institut
Universität Potsdam
Internet-Technologien und -Systeme

High-Speed Security Log Analytics Using Hybrid Outlier Detection

Dissertation
zur Erlangung des akademischen Grades
"doctor rerum naturalium"
(Dr. rer. nat.)
in der Wissenschaftsdisziplin
IT-Systems Engineering

eingereicht an der
Digital Engineering Fakultät
der Universität Potsdam

von
Andrey Sapegin

Betreuer:
Prof. Dr. Christoph Meinel

Zweitbetreuer:
Prof. Dr. Andreas Polze

Potsdam, 13. März 2019

Prüfungskomission:

Vorsitzender:	Prof. Dr. Andreas Polze	(Hasso-Plattner-Institut, Universität Potsdam)
1. Gutachter:	Prof. Dr. Christoph Meinel	(Hasso-Plattner-Institut, Universität Potsdam)
2. Gutachter:	Prof. Dr. Andrei Gurtov	(Universität Linköping)
3. Gutachter:	Prof. Dr. Florina Ciorba	(Universität Basel)
5. Mitglied:	Prof. Dr. Felix Naumann	(Hasso-Plattner-Institut, Universität Potsdam)
6. Mitglied:	Prof. Dr. Robert Hirschfeld	(Hasso-Plattner-Institut, Universität Potsdam)

Tag der Disputation: 13. März 2019

Published online at the
Institutional Repository of the University of Potsdam:
https://doi.org/10.25932/publishup-42611
https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-426118

Abstract

The rapid development and integration of Information Technologies over the last decades influenced all areas of our life, including the business world. Yet not only the modern enterprises become digitalised, but also security and criminal threats move into the digital sphere. To withstand these threats, modern companies must be aware of all activities within their computer networks.

The keystone for such continuous security monitoring is a Security Information and Event Management (SIEM) system that collects and processes all security-related log messages from the entire enterprise network. However, digital transformations and technologies, such as network virtualisation and widespread usage of mobile communications, lead to a constantly increasing number of monitored devices and systems. As a result, the amount of data that has to be processed by a SIEM system is increasing rapidly. Besides that, in-depth security analysis of the captured data requires the application of rather sophisticated outlier detection algorithms that have a high computational complexity. Existing outlier detection methods often suffer from performance issues and are not directly applicable for high-speed and high-volume analysis of heterogeneous security-related events, which becomes a major challenge for modern SIEM systems nowadays.

This thesis provides a number of solutions for the mentioned challenges. First, it proposes a new SIEM system architecture for high-speed processing of security events, implementing parallel, in-memory and in-database processing principles. The proposed architecture also utilises the most efficient log format for high-speed data normalisation. Next, the thesis offers several novel high-speed outlier detection methods, including generic Hybrid Outlier Detection that can efficiently be used for Big Data analysis. Finally, the special User Behaviour Outlier Detection is proposed for better threat detection and analysis of particular user behaviour cases.

The proposed architecture and methods were evaluated in terms of both performance and accuracy, as well as compared with classical architecture and existing algorithms. These evaluations were performed on multiple data sets, including simulated data, well-known public intrusion detection data set, and real data from the large multinational enterprise. The evaluation results have proved the high performance and efficacy of the developed methods.

All concepts proposed in this thesis were integrated into the prototype of the SIEM system, capable of high-speed analysis of Big Security Data, which makes this integrated SIEM platform highly relevant for modern enterprise security applications.

Zusammenfassung

In den letzten Jahrzehnten hat die schnelle Weiterentwicklung und Integration der Informationstechnologien alle Bereich unseres Lebens beeinflusst, nicht zuletzt auch die Geschäftswelt. Aus der zunehmenden Digitalisierung des modernen Unternehmens ergeben sich jedoch auch neue digitale Sicherheitsrisiken und kriminelle Bedrohungen. Um sich vor diesen Bedrohungen zu schützen, muss das digitale Unternehmen alle Aktivitäten innerhalb seines Firmennetzes verfolgen.

Der Schlüssel zur kontinuierlichen Überwachung aller sicherheitsrelevanten Informationen ist ein sogenanntes Security Information und Event Management (SIEM) System, das alle Meldungen innerhalb des Firmennetzwerks zentral sammelt und verarbeitet. Jedoch führt die digitale Transformation der Unternehmen sowie neue Technologien, wie die Netzwerkvirtualisierung und mobile Endgeräte, zu einer konstant steigenden Anzahl zu überwachender Geräte und Systeme. Dies wiederum hat ein kontinuierliches Wachstum der Datenmengen zur Folge, die das SIEM System verarbeiten muss. Innerhalb eines möglichst kurzen Zeitraumes muss somit eine sehr große Datenmenge (Big Data) analysiert werden, um auf Bedrohungen zeitnah reagieren zu können. Eine gründliche Analyse der sicherheitsrelevanten Aspekte der aufgezeichneten Daten erfordert den Einsatz fortgeschrittener Algorithmen der Anomalieerkennung, die eine hohe Rechenkomplexität aufweisen. Existierende Methoden der Anomalieerkennung haben oftmals Geschwindigkeitsprobleme und sind deswegen nicht anwendbar für die sehr schnelle Analyse sehr großer Mengen heterogener sicherheitsrelevanter Ereignisse.

Diese Arbeit schlägt eine Reihe möglicher Lösungen für die benannten Herausforderungen vor. Zunächst wird eine neuartige SIEM Architektur vorgeschlagen, die es erlaubt Ereignisse mit sehr hoher Geschwindigkeit zu verarbeiten. Das System basiert auf den Prinzipien der parallelen Programmierung, sowie der In-Memory und In-Database Datenverarbeitung. Die vorgeschlagene Architektur verwendet außerdem das effizienteste Datenformat zur Vereinheitlichung der Daten in sehr hoher Geschwindigkeit. Des Weiteren wurden im Rahmen dieser Arbeit mehrere neuartige Hochgeschwindigkeitsverfahren zur Anomalieerkennung entwickelt. Eines ist die Hybride Anomalieerkennung (Hybrid Outlier Detection), die sehr effizient auf Big Data eingesetzt werden kann. Abschließend wird eine spezifische Anomalieerkennung für Nutzerverhaltens (User Behaviour Outlier Detection) vorgeschlagen, die eine verbesserte Bedrohungsanalyse von spezifischen Verhaltensmustern der Benutzer erlaubt.

Die entwickelte Systemarchitektur und die Algorithmen wurden sowohl mit Hinblick auf Geschwindigkeit, als auch Genauigkeit evaluiert und mit traditionellen Architekturen und existierenden Algorithmen verglichen. Die Evaluation

wurde auf mehreren Datensätzen durchgeführt, unter anderem simulierten Daten, gut erforschten öffentlichen Datensätzen und echten Daten großer internationaler Konzerne. Die Resultate der Evaluation belegen die Geschwindigkeit und Effizienz der entwickelten Methoden.

Alle Konzepte dieser Arbeit wurden in den Prototyp des SIEM Systems integriert, das in der Lage ist Big Security Data mit sehr hoher Geschwindigkeit zu analysieren. Dies zeigt das diese integrierte SIEM Plattform eine hohe praktische Relevanz für moderne Sicherheitsanwendungen besitzt.

To my mother, Elena, to my father, Andrey, to my wife, Nataliia, and to my son, Alexey.

Acknowledgements

The PhD thesis finishes the exciting and important period of my life, which I went through with enormous support from many great people, whom I would like to acknowledge here.

First, I am very thankful to my family, especially for my parents and my wife, for all the inspiration, support and motivation they provided to me during this long time. I have to express gratitude to my mother, who had awoken my interest to computers and Information Technology, when I was at school, and who encouraged and supported me to apply for a PhD position.

Next, I remember my great-grandfather, Prof. Leonid Diukov, Candidate of Sciences, who was the first member of my family with a post-graduate degree and hence inspired me for a PhD study. I also always remember my grandfather, Grigoriy Binder, Docent, Candidate of Sciences, a mathematician, who passed away more than a decade ago. He has shown me the beauty of math and is kept in my thoughts inspiring me to learn and solve the exciting scientific problems.

Special thanks I want to give to my best friend, Mark Kibanov, PhD Student and Research Assistant at the University of Kassel, for influencing me to move to Germany and constantly helping me with advice for my life and research. Special thanks should be also given to my other friend and ex-chef, Christian Beutenmueller, a Senior Researcher at ExB Research & Development GmbH, who helped me with an understanding of the most sophisticated Data Mining and Machine Learning methods.

I am very grateful to my supervisor Prof. Dr. Christoph Meinel for giving me an opportunity to start this fascinating PhD journey and providing me his support during all the time of my study. I would also like to deeply thank the head of my research group, Dr. Feng Cheng, for the mentorship, advice and support he provided for my research work.

Finally, I need to say thanks to my team members, first of all, David Jaeger, who co-developed the SIEM system prototype together with me and took over the import and normalisation of data in this system, which allowed me to concentrate on the high-speed data analysis. I am very thankful to my colleagues, Martin Ussath and Marian Gawron and many others for their help, advice and inspirations.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The security of Information Technology (IT) systems becomes more and more important within last decades, following rapid evolution of these systems. Many institutions are constantly reporting alarming rates of cyber-security events, where enterprise losses from data breaches reach 4 million US dollars on average [1]. The rate of data compromises also remains quite high, depending on the industry, e.g. 23% for retail in 2015 [2]. Besides corporate sector, nowadays cybercrime also affects almost 600 million people worldwide [3].

The development of new software services, the growth of computer networks, constantly increasing number of devices, such as computers, laptops and smartphones, Internet of Things trend, extensive usage of wireless communications - all these factors lead to the continuous creation of new attack vectors and blur the enterprise security perimeter. Altogether, these networks, devices and systems generate a huge amount of heterogeneous security-related information, including log messages, NetFlow data and traffic captures. To properly control it and achieve acceptable security level, any modern enterprise should gather and analyse such data. Of course, these data cannot be processed manually, and are therefore collected and processed by Intrusion Detection Systems (IDS) as well as Security Information and Event Management (SIEM) systems. The efficiency of these systems became a vital part of the enterprise security infrastructure. The next section reviews the development history of these systems and describes their major technology challenges.

1.1 History of SIEM and IDS

Historically, attacks on the electronic systems have been emerged together with the appearance of these systems themselves. Examples of such attacks in the history are disruption of public telegraph system demo in 1903 [4], development of various

phreaking boxes for telephone systems in the 1960s [5] and installation of a tapping device on the underwater Soviet cable by US submarine in the 1970s [6].

With the development of computer systems, the security issues were initially recognised by manually looking on the printed log messages [7]. However, the number of such messages grew up constantly and in 1980, Anderson offered an automated threat monitoring and surveillance system [8]. This system was the first preimage of Intrusion Detection Systems (IDS), which were propagated in the early 1990s. E.g. Heady et al. were first to define an intrusion as "any set of actions that attempt to compromise the integrity, confidentiality or availability of information resources" in 1990 [9].

However, the scope of IDSs was rather focused on recognition of attacks on hosts and networks, while there was also a need in software for management of log messages and security alerts. Therefore, Log Management systems emerged to process various log messages centrally, whereas Security Information Management (SIM) systems were offered to collect and manage security-related data. Next, Security Event Manager (SEM) tools were developed to work together with IDS, SIM and Log Management systems and to produce high-level alerts for security operators. Finally, Security Information and Event Management Systems (SIEM) appeared in the late 1990s and unified SIM and SEM approaches for gathering, storing and analysing security-related data and events. A definition of SIEM was made in 2005 by Gartner: "SIEM technology provides real-time event management and historical analysis of security data from a wide set of heterogeneous sources" [10].

From that point in time, SIEM systems have been further evolved in a variety of systems with different scope and methods used. However, these systems still face a bunch of challenges, which are reviewed in the section below.

1.2 Major challenges for SIEM Technology

Nowadays, the major challenges for SIEM are mainly related to the constantly growing data volumes that need to be processed by such systems. For example, Gartner defines small SIEM deployment as "one with 300 or fewer event sources, a sustained EPS rate of 1,500 events per second or less, and a back store sized at 800 GB or less" [11]. Middle deployments have a rate of at least 15,000 events per second, while large ones need to be able to process at least 25,000 events per second and store about 50 Tb of data. In practice, the data *volumes* of the big multinational enterprises could even exceed this rate by several times. Taking into account heterogeneous nature of these data (*variety*), and the need to correlate between different events or log messages to detect complex attacks in real time (*velocity*), it is possible to classify security log messages coming into SIEM system

as Big Data [12] or *Big Security Data*, whereas its processing becomes a non-trivial task. The major challenges of Big Security Data processing are described in details below:

- **Heterogeneous log messages.** This issue is faced by every SIEM processing data from more than one source, such as Microsoft Active Directory Domain Controllers, GNU/Linux systems, or application logs in a custom format. Each system or application has its own log format with custom fields. This problem is also mentioned by other researchers, e.g. in the survey of Zuech et al. [13]. To analyse such data, a SIEM system should support all these formats. One approach to deal with it implies the creation of rules, filters and algorithms for each monitored application or system. A classic example of this approach is Snort IDS, which has more than 3000 rules written by the community for different use cases [14]. A contrary approach supposes conversion of all custom log messages into one common format, such as IODEF [15] or CEF [16]. However, such a conversion is also not an easy task, since the resulting format should, on the one hand, contain all information from original sources, and stay relatively compact to allow high-speed event processing on another.

- **The high amount of events to be processed.** Already mentioned above, this is the key issue for any SIEM system. With billions of log messages generated daily [17, 18], a SIEM system should still be able to collect, process and analyse all these messages in almost real time to satisfy market requirements [19, 11]. However, slow dashboard actions and operators waiting for a system response are reported by many sources. A slowness of SIEM systems is described as a common issue by Seekintoo [20], Balcerek et al. mention unacceptable usability of Prelude IDS due to slow reactions [21], Zadrozny et al. report about Splunk search times up to 18 minutes on 100 million events data set [22], McAfee describes slow performance of Enterprise Security Manager and its Graphical Interface in the Knowledge Base [23], SanDisk claims query times of 30 minutes for an ArcSight SIEM system [24].

- **The computational complexity of attack detection and event correlation algorithms.** Even though a SIEM system could at least manage to collect and store large volumes of data, analysis of such Big Security Data becomes a much harder task. A lot of research was done in last decades on machine learning methods for security analysis like unsupervised outlier detection, analysis of user behaviour, etc. [25, 26, 27]. However, most SIEM systems on the market still do not offer such functionality, sticking to the signature-based approach. Of course, signature-based detection fits perfectly for known attacks and issues but fails to detect novel or previously unknown

attacks [28, 29, 30]. The main reason for not implementing outlier detection in SIEM systems could be poor scalability and relatively bad performance of machine learning methods on Big Data [31, 32, 28].

- **Lack of automatic threshold selection.** Normally, a SIEM system issues alerts based on a simple threshold, if a parameter exceeds a value, which is pre-defined by the operator. For example, such option, called anomaly detection, is provided by NetIQ Sentinel software [33, 7.5 Configuring Anomaly Detection]. An alert could be issued on exceeding threshold, change in average values, deviation from baseline, etc. However, to the best of author's knowledge, no SIEM system tries to automatically identify or at least offer an appropriate threshold value to the system operator, even though there are some known techniques available [34]. The same issue applies to existing outlier detection algorithms, which often require providing a parameter for data analysis. In case the outlier detection method is based on clustering, it could be k for k-means [35] and k-NN [36] algorithms, or $MinPts$ and ϵ parameters for DBSCAN [37] algorithm.

1.3 Contributions of this thesis

The global contribution of this thesis is a development of high-speed SIEM system, capable of processing security events in nearly real-time. The prototype of such SIEM system, called Real-time Event Analytics and Monitoring System (hereafter REAMS), is being jointly developed at Hasso Plattner Institute [38]. The thesis contributions are therefore related to the part of this SIEM system, responsible for high-speed processing and analysis of security events after they were gathered and normalised, while gathering and normalisation of security data are held out of scope for this thesis (except the evaluation of log formats for normalisation as a part of architecture for high-speed event processing).

Of course, development of such a system is not possible without addressing major challenges for SIEM technologies described in the previous section. Thus, contributions of this thesis focus on providing solutions for the problems mentioned above:

- Development of an architecture for high-speed analysis of security events.
- Integration of in-memory and in-database processing technologies into SIEM system.
- Evaluation of log formats for security analytics to select the best option for high-speed normalisation of security events.

- Development and evaluation of novel hybrid high-speed outlier detection with ranked output for SIEM system.
- Development and evaluation of novel outlier detection method for analysis of user behaviour in a SIEM system.
- Integration of automatic threshold detection methods, as well as methods for automatic selection of outlier detection parameters.

Altogether, the contributions allow to build a *SIEM system capable of application of computationally heavy outlier detection methods on Big Security Data*, which makes this work novel. All these contributions are described with more details in the subsections below.

1.3.1 Architecture for high-speed security analytics

To satisfy challenging performance requirements for SIEM systems (e.g. ones from Gartner, mentioned in the previous section), an architecture for accurate and high-speed analysis of such events was developed and evaluated. To achieve high performance of the system, in-memory, in-database and parallel processing methods were used together with lightweight log format for event normalisation. All these solutions were tested to measure their performance and prove effectiveness.

1.3.1.1 Integration of in-memory and in-database processing technologies into SIEM system

The classical way of data analysis utilises a database as information storage, while processing of stored data is usually done after the information is queried and retrieved from this database. The performance of the database is limited by the type of the storage used, which is usually either Hard Disk Drive or Solid State Drive.

To increase the performance of data processing in our system, it was decided to utilise in-memory and in-database processing techniques. Therefore, an in-memory SAP HANA SQL database [39] was used as the data storage for normalised security events. Different from the classical approach, SAP HANA stores all the data in the main memory of the database server and offers various tools for in-database data processing. Examples of such tools are SAP HANA Predictive Analysis Library [40] (hereafter PAL) with many machine learning algorithms available or integration with R programming language [41]. Both are accessible through database SQLSCRIPT procedures [42]. Thus, the data could be processed without retrieving it from the database, while high-speed data access is ensured with the usage of in-memory approach.

To prove the benefits of this solution, it was compared with a classical approach based on popular SQL database, namely PostgreSQL [43]. The performance of both – classical and in-memory – approaches was measured on the same hardware. The results clearly demonstrated that in-memory and in-database processing brings a significant increase of performance, which is critical for the SIEM system.

1.3.1.2 Choosing lightweight log format for security analytics

The performance of the database backend also depends on the data model being used. This data model should be on the one hand lightweight enough, to ensure fast data processing. On the other hand, the data model should include enough fields, so that any security-related log format could be converted/normalised into it. This thesis compares our own log format, developed at Hasso Plattner Institute (which is called Object Log Format [44, 45]) with other existing formats, and selects the best one for high-speed processing of security events.

1.3.2 High-speed outlier detection for multivariate security data

Besides the high-speed collection and normalisation of security-related events, a SIEM system also needs to analyse collected and normalised data in the reasonable time. For REAMS, being developed at Hasso Plattner Institute, there are 3 types of data analysis techniques combined: signatures, complex queries and outlier detection. This thesis focuses on the outlier detection part and provides several novel algorithms for it.

Whereas queries and signatures are being widely used in SIEM and IDS systems on the market[1], the computational complexity of outlier detection makes its integration a challenging task for developers of SIEM system.

In this thesis, the novel unsupervised outlier detection algorithm is described. Different to many existing approaches, the offered algorithm does not require extraction of data-specific features or metrics from the data set; rather it could be applied on the raw normalised log messages. This is achieved through conversion of log message fields into vector space model [47] and through applying spherical k-means [48] on these data, stored as a sparse matrix.

The thesis provides both performance and effectiveness estimation of the algorithm, as well as compares it with other outlier detection approaches, such as k-NN Anomaly Detection developed by Goldstein et al. [49].

[1] A classic example of signature-based system could be Snort IDS [14], while ManageEngine EventAnalyzer [46] shows an example of query-based SIEM system.

Besides the algorithm itself, this thesis describes several other useful techniques, which were developed during its realisation.

1.3.2.1 Clustering data subsets in parallel using spherical k-means and one-class Support Vector Machine

Since the original k-means problem is NP-hard [50], it makes it almost impossible to apply the k-means algorithm on bigger data volumes. This thesis describes a possible workaround for this problem. Instead of clustering all data at once, the data are first split into subsets, which are clustered with spherical k-means. All data subsets could be clustered in parallel, which makes an algorithm highly scalable. Next, to determine outliers among all clusters in all subsets, one-class Support Vector Machine (SVM) [51] is applied on concept vectors [52] of identified clusters.

1.3.2.2 Applying automatic threshold detection for generic outlier detection

To reduce the number of parameters that should be supplied to the algorithm, the automatic threshold detection technique is applied twice. First, it is being used to determine the number of clusters (k) for k-means. Second, it is applied to select a threshold between outliers and non-outliers based on the distribution of outlier scores.

1.3.2.3 Detecting and ranking clusters of suspicious events

Another advantage of the offered outlier detection method is that its output consists of ranked clusters with anomalies. Compared to the classical approach, where an operator of a SIEM system gets a list with thousands of anomalies, the novel algorithm allows an operator to concentrate on several top-ranked clusters of anomalies, instead of digging through the endless list of standalone log messages.

1.3.3 Analysis of user behaviour

Another type of the outlier detection offered in this thesis is more specific but focuses on the very important problem: analysis of user behaviour. The developed algorithm models user logon events following a Poisson distribution. In case user logons do not follow a Poisson distribution but are rather overdispersed (mean number of logon events is less than a variance), the offered algorithm recognises it and applies the negative binomial model instead. This model was proved on the simulated Windows Events [53] data and extended to work with real data, where

interactive user logon/logoff data are often absent (i.e., only events with network logon type are presented).

1.3.3.1 Special case: intrusion without access violation

The developed outlier detection for user behaviour allows analysing special cases, like an intrusion without access violation. These cases are always especially complicated to catch for classical – signature-based and query-based – approaches. Indeed, if a user suddenly starts to connect to all available resources, it looks suspicious, although there is no access violation. To capture such cases, a two-level (group and user) outlier detection approach was developed and tested. First, a group of users with suspicious activity on some resource is identified. Next, an algorithm identifies a user responsible for such a suspicious activity within a group. Thus, this approach allows reducing an amount of false positive alerts significantly. For example, it correctly classifies situations, when a user accesses a resource, which he never used before, but which was used by other members of the corresponding user group.

All in all, the contributions presented in this thesis offer a set of solutions for the most challenging problems of modern SIEM systems. Implemented in the REAMS developed at Hasso Plattner Institute, these solutions build up a prototype of high-speed SIEM system, capable of processing and correlation of security events nearly in real-time.

1.4 Thesis organisation

The thesis is organised as follows. Chapter 2 describes modern IDS and SIEM systems, their classification and methods used. The two types of attack detection – misuse detection and anomaly detection – are reviewed in details. It also tells about data formats being used and describes data processing and event correlation techniques. Next, Chapter 3 provides information about how to create an architecture for high-speed processing of security events. The proposed architecture and its performance are tested and verified on simulated data. Compared to classical architecture, the benefits of the new approach are highlighted. Taken the proposed architecture as a basis for data processing, Chapter 4 describes classical outlier detection algorithm on the example of Anomaly Detection from SAP HANA Predictive Analytics Library. Next, a novel type of outlier detection is offered, namely hybrid unsupervised outlier detection, which combines spherical k-means clustering with one-class SVM. As a result, the output of the algorithm contains ranked clusters of outliers/anomalies, which simplifies the interpretation of results for

an operator of SIEM system. This novel outlier detection algorithm also utilises automatic threshold detection and could be applied on any textual fields after conversion to vector space model, without any feature extraction needed. Estimation of performance and effectiveness, as well as comparison with other algorithms are also provided. The offered outlier detection is also proved on the real data set from a big multinational company and all detected issues are reviewed and classified. Although the offered outlier detection works well for analysis of generic security events, an analysis of user behaviour requires a more specific approach, which is presented in Chapter 5. Here the special case of intrusion without access violation is described and analysed. The challenging task to detect such intrusions is solved with novel outlier detection based on Poisson and negative binomial models of user behaviour. Results are received and evaluated on both simulated data set and real data. Finally, Chapter 6 concludes the thesis. It discusses the implementation questions for outlier detection in SIEM systems and provides the overview of issues discovered in the data with different threat detection methods. Finally, the contributions of the thesis are reviewed from the point of implementation of high-speed anomaly detection methods in modern SIEM system.

Chapter 2

SIEM and IDS: state of the art

Modern SIEM systems vary a lot in their functions and applications, as well as IDSs. The latter usually provide monitoring at a single point, such as a host or network connection, whereas SIEM systems aim to process as much information as possible from the entire enterprise network. However, within their scope, both IDS and SIEM utilise very similar data gathering and processing techniques.

The focus of these systems could be diverse as well. While some IDS play a role of sophisticated firewall and detect attacks on network entry points, others could monitor the network internally to catch insiders, or even collect information from the entire network to analyse it centrally [54]. Thus, the SIEM technology often uses the same methods as IDS and should, therefore, be reviewed together with IDS.

Thus, this chapter describes the classification of both SIEM and IDS and reviews key technologies and methods used widely in such systems.

2.1 Classification of IDS

Nowadays there are many different SIEM systems and also many types of IDS that could be classified in different ways. SIEM systems usually have a broader scope than IDS, and collect and correlate information from multiple sources, including an IDS. However, there is still a lot in common between IDS and SIEM, especially in terms of data processing techniques. Therefore, the classification and methods described in this Chapter are relevant for both SIEM and IDS. The brief classification, describing the most widely adopted systems, is provided below.

- According to the location within IT infrastructure, IDS could be *host-based* and *network-based*. *Host-based* systems monitor single or multiple hosts, while *network-based* systems control the security of the whole network.

- Next, both *network-based* and *host-based* systems are classified by the type of data used for analysis. These types include network traffic, log messages from various monitored points, system call traces and even registry of Microsoft Windows Operating System.

  Since *network-based* systems usually process either network traffic or log messages, they are classified into *log-based* and *traffic-based*.

  Different to this, two most common types of *host-based* systems are *log-based* and *'system call trace' - based*. Besides that, some *host-based* IDSs for Microsoft Windows analyse *registry access* traces.

  Here different classes of IDSs serve different purposes and complement each other, so the comparison between classes is not always sensible. In case of *host-based* systems, "log file-based IDS are almost always less accurate than a comparable system call based algorithm due to several issues with the log entry data format" [55]. However, in case of *network-based* systems, such comparison would not be possible. This is because *traffic-based* IDSs are rather focused on the traffic at the monitored point itself, while *log-based* systems, which are usually SIEM and not IDS, try to gather as much information as possible from different log sources, which often include alerts from *traffic-based* IDSs as well.

- By detection method, IDS and SIEM are usually distinguished between *misuse detection* and *anomaly detection* systems. Modern systems tend to utilise both of these approaches and are called *hybrid*.

- Finally, it is also possible to classify IDS and SIEM by its focus. Some systems analyse *user behaviour*, while others are focused mainly on the host and network *processes* [56]. Additionally, there are *database* IDS, which monitor database transactions for malicious activities [57], or *wireless* IDS, which controls the radio range [58].

From the provided classification, it is possible to conclude that SIEM and IDS vary a lot in their functions and purposes. However, most of them use the same or similar methods for data gathering, storing and analysis. These common methods are reviewed in the sections below.

2.2 Data gathering and analysis techniques for Intrusion Detection

The purpose of the Intrusion Detection System is to detect malicious activity in the monitored network or at the monitored point. Obviously, the quality of

the detection depends on the quality of the collected data. Thus, data collection becomes an important step in the intrusion detection process.

2.2.1 Data collection

There are different ways used for the data collection by both IDS and SIEM. In case of the *host-based* IDS, the data for the detection can be collected directly from the system, where an IDS is installed. However, in case of *network-based* IDS, the data collection methods vary depending on the location of the IDS.

The basic scenario considers that an IDS is used to monitor the single available connection to the Internet. Placed next to the firewall, an IDS system monitors all Internet traffic and does not need any extra sensors in this case as well.

However, monitoring the modern network perimeter is not as easy as monitoring single connection point. Due to the growing number of mobile and wireless devices, the network perimeter in modern enterprises becomes more and more blurred or even disappears [59]. Therefore, an IDS should monitor not only all regular Internet connection points and VPN servers, but also Wi-Fi connection points and end-user devices like laptops and smartphones [60, 61].

Such networks can be monitored using *distributed network-based* IDS that have not only IDS nodes or sensors, but also agent applications installed on every monitored point, such as user computers. Another type of an agent are third-party firewalls or even other IDSs that support external monitoring or can send the monitoring data to the distributed IDS.

The *distributed* IDS also analyses the data at all its nodes, which makes the correlation of data between nodes almost impossible. Different to this, the *centralised network-based* IDS first collects all the data from nodes and agents centrally. The Central Analysis Server is the place where events are analysed, correlated, visualised and reported [54].

Generally, the data collection for all types of agents, including IDS nodes, sensors, application agents and third-party devices could be implemented in two ways. Either the data are pushed from all agents to the central server, or the server every time initiates the data transfer by querying each agent with the predefined time interval to pull the needed information. As IDS and SIEM aim the high-speed processing of security data, the data are usually pushed to the central server immediately. However, in this case, the Central Analysis Server becomes a bottleneck in the large network and should be able to process and store all the messages it receives. As mentioned in Section 1.2, the large deployments need to process 25,000 events per second and store at least 50 Tb of data, which becomes a non-trivial task and requires an efficient data processing architecture to be developed.

Thus, sometimes there is almost no difference between *centralised network-based* IDS and SIEM system that also collects and correlates the security-related

data from agents, such as network devices, user computers, other SIEM systems and IDSs.

The analysis and detection techniques used in both SIEM and IDS are also similar and will be described in the sections below.

2.2.2 Misuse detection

Misuse detection is currently the most common detection approach used in IDS and SIEM. Under this approach, analysed data (network packets, log messages, system calls, registry access traces) are checked against a specific set of rules or patterns that match a malicious activity.

In the simplest case, such rules represent filter masks applied on the specific data fields or characteristics, such as packet header, payload, TAG part of syslog [62] message, etc. These signatures are used in many popular IDS such as Snort [14] and Bro [63]. The difference of this simple signature detection approach from firewall rules is that firewall rules are applied on the packet header only, while systems like Snort apply their rules on any part of the data, including payload [64].

Of course, signature detection is not limited to such simple filtering. Using special languages such as STATL [65], LAMBDA [66] or SHEDEL [67], it is possible to create much more complicated attack models and specifications [68]. The more advanced approach, namely Event Description Language, allows describing signatures for multi-step attacks using signature-nets (similar to Petri-nets) [69]. The recent work of Jaeger et al. shows how to detect multi-step attacks with this language [70].

Besides signature detection, misuse detection also utilises machine learning and data mining algorithms, which are applied to learn malicious patterns from existing data [71, 72, 73]. This type of detection requires samples with malicious patterns. These patterns are processed by data mining algorithm to create a model of malicious behaviour. At the next step, incoming data are checked against the learned model. Thus, a similarity between known malicious patterns (contained in the sample) and new incoming data is measured [74]. Finally, if the similarity between new data and pattern is high enough, an alert will be issued.

A classic example of machine learning based misuse detection was provided by Mukkamala et al. [75] on the KDD Cup 1999 data set [76] (hereafter KDD Cup data). First, training and test data sets were created by selecting samples of normal and attack data from original labelled data. Based on these samples, they have used Support Vector Machine and Artificial Neural Network to train models of both normal and attack activities. Next, authors applied models on the different testing data sets, reaching 99% or even higher accuracy.

Since machine learning and data mining methods could be applied not only for misuse detection but for anomaly detection as well, the relevant techniques will be reviewed in more details in Section 2.2.5.

Although misuse detection and, in particular, signature-based detection is considered an efficient approach and adopted widely, it still has some limitations. The main limitation of misuse detection is the inability to detect previously unknown attacks if no signature or model covers them [25, 26, 29]. This implies that misuse detection systems should be actively maintained and updated on a regular basis in order to function properly. Besides this, the variety of monitored applications and protocols generate many different traces that should be covered by signatures or modelled.

One approach to deal with these heterogeneous data is to create a separate signature for each protocol or application. For example, in Snort [14] it results in more than 3,400 rules maintained by the community. A high number of rules negatively affects the performance of the whole system, since it becomes complicated to find a corresponding rule for each piece of data being analysed.

An alternative to this approach is to apply signatures or create models based on the normalised data. For example, for log-based IDS or SIEM, it means that each log line should be converted into one common format. Then, each type of attack signature should be written only once for this common format. Although it allows reducing the number of signatures, a normalisation of heterogeneous data is not always straightforward. For example, Jaeger et al. utilise a hierarchical knowledge base for normalisation of log lines [77]. Based on regular expressions, it achieves high performance in both normalisation and searching corresponding signature or rule.

All in all, misuse detection and especially signature detection remains to be a fast and reliable method for IDS and SIEM. The main limitation of misuse detection approach — inability to detect novel attacks — could be compensated by the use of anomaly detection, which is described below.

2.2.3 Anomaly detection

Different to misuse detection techniques that check data against predefined rules or signatures to capture malicious patterns, anomaly detection tries to build a model or pattern of "normal" system state or behaviour and then alerts on the deviations from this "normal" model. Since an anomaly detection approach allows capturing all deviations from "normal" state, it can detect any kind of novel attacks, which is not possible with misuse detection. However, this approach also results in two problems. First, it is a very tough task to build a model of "normal" system state. Second, anomaly detection considers any kind of deviation from "normal" model as an issue, while these deviations could be benign [78]. For example, false alerts

could be caused by connections of a new user, changing roles of users and user groups, rare cases of authorized remote access, software updates and so on.

Although there are some heuristic, signal-processing or rule-based anomaly detection models [79], nowadays most commonly used anomaly detection approaches are based on statistical methods as well as machine learning or data mining techniques [72]. These methods are reviewed in details in Section 2.2.5.

2.2.4 Hybrid detection systems

To avoid disadvantages of misuse and anomaly detection, modern IDS and SIEM normally try to incorporate both of these approaches. Such systems, called *hybrid*, apply anomaly detection methods on data together with misuse detection. The results from both approaches are either aggregated or hierarchically integrated with each other [80]. Thus, misuse detection will report all detected known attacks, avoiding any false positive alerts. Meanwhile, novel attacks could be recognised by anomaly detection approach, among some false positive alerts.

2.2.5 Machine learning and data mining methods for intrusion detection

Machine learning and data mining methods rely on statistical techniques and try to extract knowledge or learn models from data itself [81, 82]. Such knowledge, models and other characteristics extracted from data are then used to solve different kinds of problems and tasks. Two major classes of machine learning and data mining algorithms — based on the type of the problem to solve — are:

- **Classification algorithms**, which classify new events according to the model learned from data (to solve so-called classification problem).
- **Regression algorithms**, which predict a value of a variable based on one or more parameters (to solve so-called regression problem).

Of course, apart from this two generic problem classes, machine learning algorithms are applied to a wide variety of tasks from very different areas. Based on these tasks, machine learning and data mining methods could be grouped into classes as follows:

- **Clustering** methods that cluster "similar" events together. Such algorithms also solve classification problem, however, clustering algorithm needs to classify all available data (and not only new events) into clusters.

- **Dimensionality reduction**, which is used to reduce the number of dimensions in the data, to simplify forthcoming data processing.
- **Outlier/novelty detection** methods that are often based on clustering algorithms. These methods are used to find unusual, rare or novel data points, i.e. to identify suspicious activity.
- **Natural language processing** techniques which are used to extract semantic from the text, classify documents by topic, recognise spam, perform automatic translation etc.
- ...

Here the same algorithms can be applied to the different problems. For example, a clustering algorithm could be used to classify data points into several classes. When the clusters are found, they could be used for outlier detection purposes. E.g. points located far from all clusters are considered outliers. Next, the same clustering algorithm could be applied to cluster texts into topics for the purposes of natural language processing.

Besides the classification based on the type of the problem, machine learning approaches are also classified depending on how the algorithm learns from the data, i.e. on the properties of data [83]:

- **Supervised learning** where a data set for training contains labels for each example. A machine learning algorithm tries to build a model, which will correctly predict a right label for new-coming samples.
- **Unsupervised learning**, when there is no training data set, i.e. data do not contain labels. Therefore, a machine learning algorithm tries to group data points together and separate them into different categories.
- **Semi-supervised learning**, when only a few samples from the training data set have labels. During training, machine learning algorithm will first try to cluster samples together and assign missing labels based on similarity with labelled samples.
- **Reinforcement learning**, when, instead of getting a label, a machine learning algorithm gets feedback for each answer it produces for specific input (reward in case of right answer). However, this feedback does not contain information, how to improve an answer.

In addition to these classes, the training data set can be also prepared in different ways. For example, to apply some machine learning techniques on unlabelled

data set for outlier detection, such a data set could be divided into two parts, where the first one will be considered or labelled as "normal". The model trained on this part of the data will not be able to find outliers in it. However, such model can be applied on the second part of the data to highlight all novel data points or outliers, relative to the first part of the data. This approach is often utilised for outlier detection in the area of SIEM and IDS. As soon as it does not require any information, labels or examples of outlier records during the training phase, it will be considered unsupervised in this thesis, whereas different researchers could classify it differently: as supervised [28] or also as unsupervised [84].

In the area of SIEM and IDS, generally, two types of machine learning algorithms are used: (1) supervised classification for misuse detection and (2) unsupervised outlier/novelty detection for anomaly detection. The review of these machine learning techniques for SIEM and IDS is provided in subsections below.

2.2.5.1 Supervised classification methods for misuse detection

Nowadays, recent machine learning and data mining methods for misuse detection can be applied on security data with very high accuracy.

An example of such advanced misuse detection with machine learning methods is provided by Wang et al. [85]. By applying fuzzy clustering and Artificial Neural Networks on the KDD Cup data set, authors achieved 99.91% accuracy with 98% F-score. The total training time on 1.8 GHz dual-core CPU (1.8 GB RAM was used) was approximately 35 minutes for 18,285 randomly selected records.

Another research by Dhakar et al. [86] has shown 99.96% accuracy and 0.03% False Positive Rate (on the KDD Cup data) by combining Tree Augmented Naive Bayes and Reduced Error Pruning.

Li et al. [87] apply advanced data set deduplication (based on k-means clustering), training data set construction (using Ant Colony Optimisation), as well as feature selection and reduction methods (Gradually Feature Removal) before training an SVM on KDD Cup data. This allows authors to reach 98.62% accuracy for SVM classifier. The prediction time for the whole KDD Cup data set was less than 9 seconds on single core CPU (1.8 GB RAM was used), depending on the number of features.

Farid et al. [88] proposed a combination of Naive Bayes and Decision Tree, which allowed them to achieve 99% accuracy also on KDD Cup data.

Kevric et al. [89] also achieved an accuracy of 99.53% by training NBTree and Random Tree classifiers on 20% of KDDTrain+ data from NSL-KDD [76] data set. The classifiers were combined with sum rule and tested on the same data with tenfold cross-validation.

Peddabachigari et al. [90] reached 100% accuracy by combining decision trees and SVM algorithms first hierarchically, and then in the ensemble.

Besides the direct application of machine learning methods for misuse detection, some researchers utilise these methods to create/extract signatures for misuse detection. For example, W. Li applies Genetic Algorithm to evolve rules for IDS from the pre-classified data set with network traffic [91].

Rastegari et al. [92] also utilise Genetic Algorithm for the creation of rule sets. Authors compare their approach not only with existing genetic algorithm based methods for evolving rule sets but also compare the accuracy of the classifier based on the evolved rules with non-genetic algorithms, such as k-nearest neighbours and decision trees. The results show similar accuracy (up to 98.4%) for all methods on the constructed data sets.

Thus, supervised machine learning and data mining methods are comparable to signatures at least in terms of the accuracy of misuse detection. The estimation of analysis' speed for these methods is, however, not so straightforward since most of the algorithms were evaluated on the relatively small KDD Cup data set (less than 5 million records) or its subsets. The study of Chauhan et al. [93] shows that some of machine learning methods can be applied on the data at relatively high speed, e.g. 0.04 seconds for Instance Based Learning or 0.6 seconds for Random Tree on 25,192 records from NSL-KDD data set. Taking into account the nearly linear computational complexity of Instance Based Learning or Random Trees, one could assume that processing of much bigger data set should be possible as well. Moreover, since the supervised machine learning techniques could also be used to evolve signatures or rules, the processing speed does not play such a big role. Once evolved, the rule sets or signatures can be directly added into an Intrusion Detection System to extend an original set of signatures.

Of course, supervised approaches, even having the high accuracy, are by definition not able to detect previously unknown attacks. Such detection is usually performed with unsupervised machine learning and data mining methods, reviewed in the next subsection.

2.2.5.2 Unsupervised outlier detection methods for anomaly detection

Within last decades, anomaly detection methods based on machine learning and data mining were significantly improved. These methods often allow authors to reach a high precision (90-100%) and relatively low false positive ratio.

Papalexakis et al. used Sparse Matrix Regression (SMR) Co-clustering and Information Theoretic Co-clustering for anomaly detection on KDD Cup data set [94]. The authors did not use any special feature selection techniques and, therefore, do not rely on any special characteristics of the particular data set. The data labels were also used only for evaluation of the results. These properties make their approach fully unsupervised compared to other related work (reviewed in the paper), where researchers either use custom features (specific for the par-

ticular data set) or train the algorithm on the "normal" subset of the data. The computational complexity of SMR Co-clustering did not allow its application on the selected subset of the data, which contained less than 500,000 events. The authors applied it on the smaller data subset and extracted the parameters for Information Theoretic Co-clustering, which was "not able to isolate a consistent set of parameters". After the Information Theoretic Co-clustering was applied on the subset of almost 500,000 events, it returned 2 clusters with anomalies and 3 clusters with normal data (checked with the data labels), whereas the prevalence of the main class in each cluster was between 94.7% and 99.79%. This result was comparable with the winner of KDD Cup 1999 competition, who used a supervised approach.

Salem and Stolfo [27] applied one-class SVM for anomaly detection in user search behaviour data. The authors analysed user file searches (from the specially collected Windows data set with 20 million records) and created custom features, such as "number of file touches" or "percentage of file system navigation user actions". Using 80% of normal user data for training of one-class SVM, they achieved 100% masquerade detection rate with only 1.1% of false positive alerts, whereas the execution time was from 18 to 69 minutes on a regular desktop with 2.66GHz Intel Xeon processor (24 GB RAM was used).

Viswanath et al. [95] propose to use Primary Component Analysis (PCA) for anomaly detection of user behaviour. The goal of the authors is to find anomalous users in the data sets from Facebook, Yelp and Twitter. The authors create custom features for social network users, such as a number of likes per day. The features are forwarded to PCA and then the threshold on the L2 norm in the residual feature subspace is set to detect anomalies. This approach achieves a detection rate of 66% but with less than 0.3% false positive rate.

In [96], Ye et al. perform analysis of various data sets containing audit events from the BSM security extension for the Solaris OS (up to 100,000 events of 284 types, each data set is related to single host machine) with Markov Chain Model. The authors evaluate the effectiveness with Receiver Operating Characteristic (ROC) curve, showing nearly 100% True Positive Rate and small False Positive Rate. The execution time for their approach is not estimated, but authors mention that all referenced anomaly detection techniques are computationally intensive.

Ni et al. [97] describe the challenges for Big Data Intrusion Detection. To elaborate on these challenges, they develop an approach to improve the performance of the anomaly detection on large-scale data. Authors offer an "Unsupervised Feature Selection based Density Peak clustering", which is based on the Maximal Information Coefficient. It allows processing different types of data (both numeric and textual). Authors utilise this algorithm for feature selection on KDD Cup

data set. Next, they evaluate classification accuracy of various algorithms, such as Support Vector Machine or Random Forest applied to selected features. All in all, the offered feature selection method allowed to reduce the time needed for the analysis by 14.44% on average, whereas the loss of accuracy was less than 1%. However, due to the memory constraints, the authors were only able to apply their approach on 10% of the original KDD Cup data set.

Another anomaly detection method was developed by Goldstein et al. [49]. This method relies on "data views" that aggregate various statistics for users and workstations from Windows Events. Next, kNN-based outlier detection is used to analyse these statistics. Thanks to the usage of "data views", the outlier detection is performed on the relatively small volume of aggregated data instead of the full data set. Authors evaluated the results by manually checking the outliers and considered them useful. However, this approach is limited to the Windows Events only and cannot be utilised (without changing the "data views") on any other data. Besides that, the offered implementation of the kNN outlier detection implements linear search and has rather high computational complexity ($\mathcal{O}(n^2)$), which can be an issue for the data sets with a high number of users and workstations.

Chawla and Gionis [98] propose discovering outliers during clustering, and implement an extension of k-means algorithm accordingly. This algorithm is proved to converge to the locally optimal solution and has linearly growing computation time for a number of processed records if other parameters (dimension, number of clusters and number of outliers to return) are fixed. Authors evaluated it on 10% of KDD Cup data, but only analysed numerical attributes. Next, they compared their results with kNN-based outlier detection, which was not able to run on the selected data set. The proposed algorithm achieved a high purity of outlier clusters and better precision in comparison with kNN outlier detection, however, the precision does not exceed 60%.

Song et al. [99] offered to combine clustering with one-class SVM for outlier detection. The algorithm heuristically prepares training data by filtering out possible attacks. This step is performed by analysing feature space and filtering out all data points from sparse regions. After the training data was prepared, the algorithm heuristically divides it into clusters, each of which is forwarded as an input into separate one-class SVM. The testing data are also filtered in the same way and then classified into existing clusters, to select the corresponding SVM model for outlier detection. This idea allowed authors to implement the outlier detection process in parallel. The algorithm was evaluated on self-collected traffic data from Kyoto University's honeypots with approximately 60,000 records. The results show 75-85% precision for normal data, 100% precision for attack data and 15-25% false positive rate. The authors claim to keep the time complexity comparable with their earlier work, which was evaluated on KDD Cup data set [100].

In that paper, an offered method took approximately 500 seconds for training and 50 seconds for testing on 600,000 records, while the execution time grows linearly.

Some researchers also managed to apply classical neural networks for unsupervised outlier/novelty detection, even though the neural network approach is often not considered as the first choice for outlier detection [101]. This is because neural networks could have problems with detection of completely novel classes due to open decision boundaries [102]. Compared to one-class SVM, neural network does not produce such a clear outlier score like decision values from SVM, which are basically the distance to the hyperplane around "normal" class. Therefore, different heuristic techniques are applied to create a metric for outlier detection, such as "threshold on the highest output value" from the neural network or average reconstruction error for Replicator Neural Networks [103].

For example, Hawkins et al. [104] constructed the Replicator Neural Network (RNN), which is a multi-layer perceptron with three hidden layers. To apply it for anomaly detection purposes, authors define an Outlier Factor as "average reconstruction error over all features" and rank outliers accordingly to this measure. The approach was tested on the subset of KDD Cup data set, which contained 4 out of 41 features for 703,066 records, where only 0.48% were attacks. This subset was further divided into smaller subsets by the type of the application ('http', 'smtp', etc.). Each subset was used for training of a corresponding RNN. To train it within a feasible time, authors decided to further reduce the subset by sampling it so that the maximum size of the training subset never exceeds 6,000 records. The results are evaluated in different ways, including an analysis of the middle hidden layer of the RNN. The best results are obtained for analysis of 'http' data. The middle layer analysis shows that one of the value patterns covers 2202 of 2211 intrusions, whereas the precision is 99.6%.

Other neural network-based approaches, such as Self-Organising Maps could be more suitable for the task of anomaly detection. Lei and Ghorbani [105] used unsupervised machine learning methods, including Self-Organising Map and "Improved Competitive Learning Network" on the subset of KDD Cup data. First, these methods were applied to cluster the data set into 6-15 clusters. Next, authors labelled the clusters as either intrusion or normal using the label information from KDD Cup data (if more than 50% of the records in the cluster belong to one of 7 intrusion classes from the testing data set, then the cluster is labelled as intrusion). Finally, the testing data were classified into these clusters. The applied methods allowed them to achieve 97.8% accuracy with 98.4% precision and 98.97% recall. Training was performed on 101,000 records and took from 500 to 3,000 seconds, depending on the selected algorithm and parameters. The testing time was from 454 to 3,308 seconds for 400,000 events in the testing data set. However, the authors do not provide any hardware details of the execution environment and do

not share the reasons why only the part of KDD Cup data set was selected for the analysis.

All in all, the reviewed anomaly detection methods can indeed reach high effectiveness, but only under some circumstances and for particular data set. In [27, 95, 49, 104, 96], the authors use custom data-specific features or feature selection based on the data set knowledge. In [99], authors prepare the training data set with 'normal' data based on the assumption that dense regions in the feature space represent the normal state. In [105], the label information is used to distinguish clusters with the prevalence of normal and attack data. Similarly, the algorithm used in [94] returns the clusters with a high prevalence of the main class, but the knowledge about the data set is required to identify which of these clusters contain attacks. Thus, these approaches are not fully unsupervised but achieve high effectiveness. On the other hand, the more generic approaches, like one offered in [106], achieve much lower precision (60%).

Considering the performance of the reviewed approaches for anomaly detection, authors often mention the high computational complexity of their methods ([96, 94]). In [94, 97, 104], authors were forced to reduce the data set size due to computational complexity or memory requirements of their algorithms. Salem and Stolfo [27] do not claim any performance issues, but this is due to their use of custom features. The similar manual feature construction allows Goldstein et al. [49] to apply kNN-based anomaly detection even based on linear search (which is quite inefficient).

High performance was claimed by Lei and Ghorbani [105]. However, for the training and following classification of testing data, their system needs attack labels. Taking into account the fact that 80% of records of KDD Cup data set are labelled as intrusions, it could be relatively easy to create clusters with intrusions during the training phase. Also Ni et al. consider relying on data label information as impractical for large-scale network data [97].

The approach offered by Song et al. also has quite high performance, is easy to parallelise and could indeed be more suitable for Big Data analysis. However, it relies on the heuristic filtering and clustering of training data. Applied on the relatively small data sets (up to 600,000 records from KDD Cup data), it shows good performance. Nevertheless, it is hard to predict, how good the filtering and clustering will work on the Big Data with hundreds of millions of events from heterogeneous sources.

Thus, the processing of Big Data is challenging for existing unsupervised outlier detection methods in terms of both effectiveness and time complexity. Nowadays, both SIEM and IDS should deal with large data sets that contain data from heterogeneous sources. The heterogeneous nature of these data (data collected from different sources in different formats, and containing different types of informa-

tion about monitored systems) makes it hard to extract data-specific features and apply anomaly detection algorithms developed for particular data set. Finally, the volume of the data makes application of some unsupervised algorithms almost impossible.

The key to the processing of large volumes of heterogeneous data from various sources is the data normalisation, which will be reviewed in the next section.

2.3 Data normalisation and event correlation[2]

The Security Information and Event Management systems collect and process security-related information and alerts from the entire enterprise network. This information comes from different data sources, such as network-level devices (routers and firewalls), web servers, domain controllers, computers of regular users, IDSs and even other SIEM. The correlation of these diverse data is hardly possible without conversion or *normalisation* into the same format. The existing normalisation formats and correlation techniques are reviewed in this chapter.

2.3.1 Data normalisation formats

Each modern software or application usually writes the information into the log file or system socket (in Unix-based systems) [107], or Event Log (in Microsoft Windows Operating System) [108]. In any case, this information could be forwarded to the Log Management server and in the end collected by SIEM or IDS. However, the existence of a high number of custom log formats makes the log analysis more complicated [109]. Many system administrators, developers and operators of security systems are therefore experiencing problems with analysis of such data [110, 111]. To elaborate on this issue, several log formats were proposed as a standard: CEF [16], CEE [111], IDMEF [112] and IODEF [15].

Nevertheless, the proposed formats are often avoided by many software developers in their projects. This applies to the modern IDS as well. For example, systems like BlackStratus LogStorm [113], Sawmill LogAnalyzer [114] and OSSEC [115] use a custom self-developed log format. The causes for this situation originate from the architecture of IDS, SIEM and Log Management systems. All such systems handle huge amounts of data [116] and need to store and process them efficiently. The existing generic log format standards do not perfectly fit for this purpose. These formats contain too many standardised fields that do not compose together the suitable schema for normalisation of security-related log messages.

In particular, to parse log messages from the most common sources, such as Windows Events [53] and syslog [62], one needs to add many custom fields to

[2]Some contents of this section have been published as [44, 45].

existing formats. At the same time, many defined fields often remain unused [44].

To address this issue, a novel Object Log Format (OLF) [44] was jointly developed specially for normalisation of log messages within SIEM and Log Management systems.

The above mentioned log normalisation formats, including the proposed OLF, are now review in the subsections below.

2.3.1.1 Common Event Format

The Common Event Format (CEF) [16] was originally developed by HP for its former ArcSight SIEM system [117]. This format is specially designed for SIEM and other security-related information systems. Based on the syslog, it contains eight mandatory fields. These fields include meta-information, for example, version, device vendor, severity and name. The last mandatory field is an extension field, which contains a collection of other (optional) fields, stored as key-value pairs [118]. The documentation describes an Extension Dictionary with 165 optional fields defined. The fields could be mapped to each other. Besides that, custom extensions of CEF are also supported.

2.3.1.2 Common Event Expression

The Common Event Expression (CEE) format was offered by MITRE [111], but the work on this format was discontinued since 2014. The CEE is a hierarchical format containing 58 fields, 7 of which are objects (each object consists of several fields). In case the number of defined fields is not enough, CEE supports custom fields and custom syntax for events.

2.3.1.3 Intrusion Detection Message Exchange Format

The Intrusion Detection Message Exchange Format (IDMEF) is defined in the RFC 4765 [112]. IDMEF is mainly proposed for the exchange of the messages between security systems but can be also used for reporting and normalisation. IDMEF is a hierarchical class-based format. The top class called “Message” has several subclasses, each of which describes a particular message type. The major message types or subclasses also have subclasses, namely 5 core classes (“Analyzer”, “Source”, “Target”, “Classification”, and “AdditionalData”). The classes at the lowest level of the class hierarchy consist of attributes. All in all, the whole class hierarchy contains 118 elements. The RFC also defines the way to extend classes and attributes.

2.3.1.4 Incident Object Description Exchange Format

The Incident Object Description Exchange Format is defined in the RFC 5070 [15]. Different to IDMEF, this format is proposed for sharing information between Incident Response Teams, and not Intrusion Detection Systems. The data model is similar to the IDMEF and has a multi-level class hierarchy. All in all, this format is backwards compatible with IDMEF but has more pre-defined classes for the metadata, which describes security incident. IODEF has 19 top-level classes and 83 attributes.

2.3.1.5 Object Log Format

During the joint development of HPI REAMS, the existing log formats have shown some disadvantages. Not all log messages being analysed in REAMS could be normalised into existing formats without the use of custom fields or another extension [45]. The most problematic part was a precise parsing of the textual event descriptions. The existing formats by default usually have an only single field for text messages, such as "authentication failure", "Relaying denied. IP name lookup failed" and "BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity [Classification: A Network Trojan was detected]". Such fields are, for example, "Classification" class of the IDMEF or "message" field in the CEE. Next, before an extension of IODEF was offered by IETF in RFC 7203 [119], existing formats did not have enough fields for storing security meta-information, such as vulnerability classifiers, the correlation between events and other metrics [44].

Besides that, an extension of existing formats by adding all necessary fields was not an optimal solution, taking into account huge data volumes that need to be processed by SIEM system nowadays. Adding new fields, classes and attributes blows up the data model, which could have a negative impact on the overall system performance.

Therefore, a new lightweight format was offered to optimise the data model for normalisation of log messages, and thus improve an attack detection. The offered format, called Object Log Format [44], is presented in Figure 2.1.

The proposed hierarchical format has 3 levels, starting with 19 top-level elements. All in all, OLF has 107 fields that fully satisfy the requirements of REAMS for normalisation of log files.

2.3.2 Event correlation

After the log messages are normalised into the same log format, the correlation of events between log traces from different sources becomes much easier. Indeed, the data stored in the different fields of different original formats is now saved in the

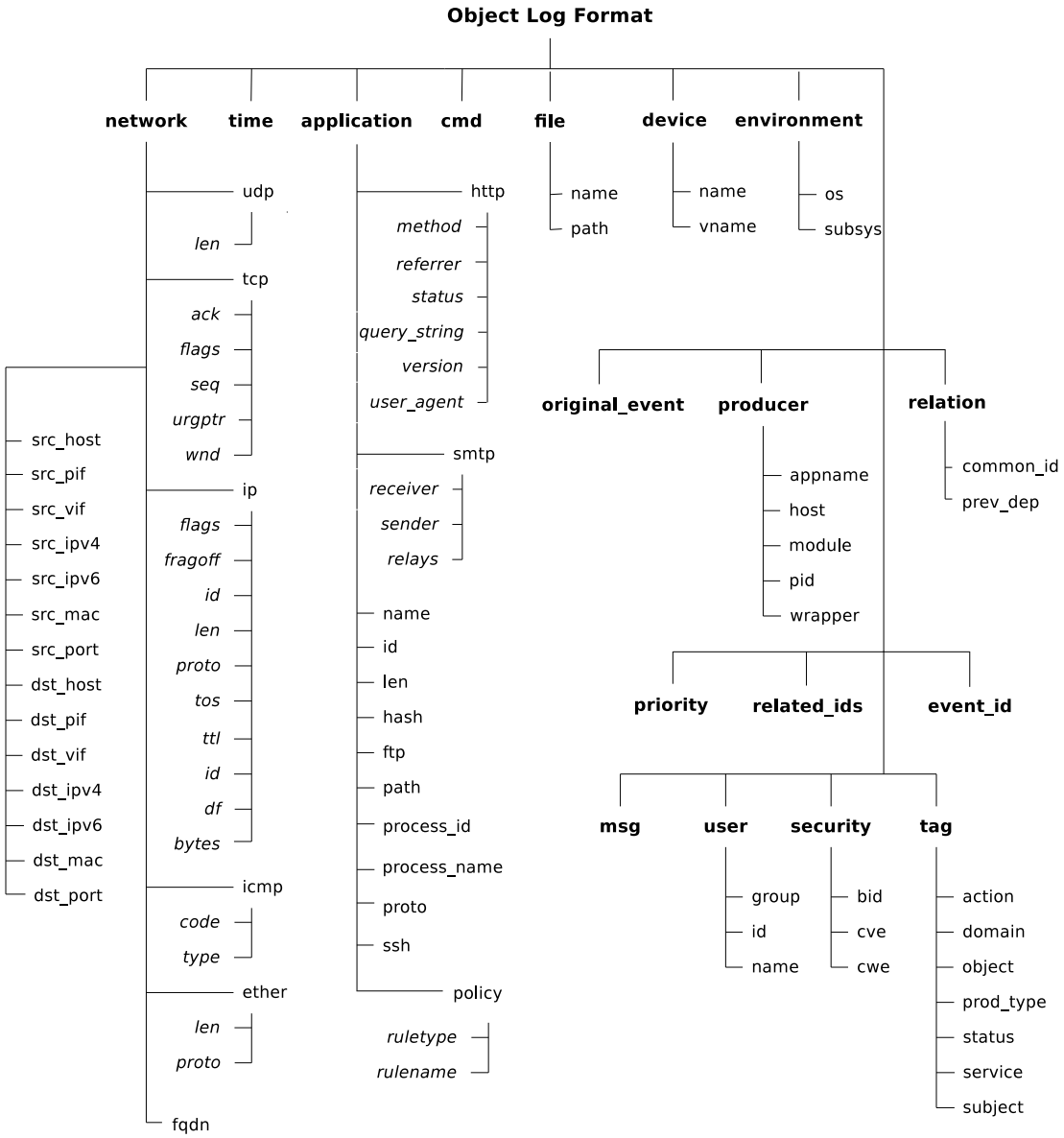


Figure 2.1: Tree of properties in Object Log Format

same attributes of the selected common log format.

The correlation of the normalised data can be performed in the different ways. One example of the simplest correlation method would be queries in SQL or similar language to get general overview or statistics about the data. Examples of such basic queries are listed below as follows:

- number of events, users and hosts in the network or subnetwork

- user logon/logoff events calculated per time interval
- number of log messages during a predefined time period
- types of security-related events
- statistics on failed events
- ...

The example of a little bit more complicated query in SQLSCRIPT [42] language is presented in Listing 2.1.

Listing 2.1: Users, accessing network shares of other users

```
1  select count(*), to_varchar(file_path),
2  targetuser_username, min(time), max(time)
3  from EVENT
4  where event_type_id = '5140' and
5  to_varchar(file_path) like '%.%' and
6  targetuser_username like '%.%' and
7  upper(to_varchar(file_path)) not like
8  concat(concat('%',upper(targetuser_username))
9  ,'%')
10 group by to_varchar(file_path),
11 targetuser_username
12 order by targetuser_username, count(*) DESC
```

Listing 2.1 shows the query for Windows Event logs. This query selects Windows users, accessing network shares of other users. Even though this query works on single source only — Windows Events — the analytics of such data would hardly be possible without data normalisation. The original Windows Events have different XML schemas for different Event IDs and/or different providers that complicates parsing and correlation of events within this single source [120].

The correlation of events takes into account not only the data from log messages themselves but also the metadata, which can be extracted during the normalisation in common log format. The examples of such metadata are event status (failure or success) and type of the action described in the log message (a logon event, access to the file share, a start of the process). If extracted from all events and sources, the metadata simplify filtering of events. The example of such filtering is shown in Listing 2.2.

Listing 2.2: Sample query capturing all failure events

```
1 select distinct(event_type_id)
2 from EVENT where tag_status = 'Failure';
```

The query in Listing 2.2 selects failure events by checking their status. Since the metadata (event status) were extracted from all events, now the simple query is enough to find out all failure events from different sources, including Windows, generic syslog and various application logs (like a web server, SSH and so on).

Of course, queries are the simplest correlation method that could be applied to normalised data. The normalisation and conversion of all incoming data into one common log format also simplify data analysis and correlation with other methods, such as misuse and anomaly detection that were already reviewed in Sections 2.2.2 and 2.2.3.

2.4 Chapter summary

In this chapter, the current state of SIEM and IDS was described, including data collection methods, intrusion detection approaches and normalisation formats.

The blurring network perimeter requires IDS and SIEM to be capable of collecting data from various systems and devices from all over the network. This, in turn, increases the data volumes that such systems should process on a regular basis. Therefore, the special architecture design is needed to enable high-speed processing of high data volumes.

Another important point for high-speed processing and correlation of heterogeneous log messages from various sources is the normalisation into common log format. This common format defines the data model, which in the end has an impact on the overall performance of the system. The existing normalisation formats differ in the structure and size, so the choice of the optimal data model becomes a challenge as well.

Provided examples of machine learning algorithms for unsupervised outlier detection reflect the existing challenges mentioned in Section 1.2, in particular, high amount of heterogeneous log messages and computational complexity of machine learning and data mining methods. These challenges prevent modern SIEM and IDS systems from wide adoption of machine learning and data mining methods and application of anomaly detection for Big Security Data.

To solve all these challenges, a jointly developed prototype of the SIEM system is proposed in this thesis (Real-time Event Analytics and Monitoring System). According to the classification described in Section 2.1, the offered system could be classified as *centralised network-based log-based hybrid SIEM*. Within this system, the thesis focuses on the high-speed processing of log messages as well as their high-speed analysis. For the analytical part, the thesis focuses on anomaly detection

techniques based on unsupervised machine learning and data mining algorithms[3].

Next chapters describe the contributions of this thesis in details, starting with developed SIEM architecture for high-speed analysis of log messages.

[3]Hereafter the terms anomaly detection and (unsupervised) outlier/novelty detection will be used interchangeably. Although there are other anomaly detection techniques that do not use machine learning and data mining methods (please see Section 2.2.3 for details), nowadays the anomaly detection in SIEM and IDS is normally presented with unsupervised outlier detection algorithms only.

Chapter 3

SIEM architecture for high-speed event analysis[4]

High-speed processing of security events requires a special architecture to deal with large volumes of data. Before the development of such architecture, one needs to define what is "high-speed" related to SIEM systems. Since the goal of the SIEM is the "real-time collection and processing of security events", in this thesis a SIEM system is called "high-speed" if it is able to process events with all analytical methods (including data mining) faster than they come into the system. According to Gartner [11], the large SIEM deployment should be capable of processing at least 25,000 events per second. For historical data analysis, it implies the ability to process more than 2 billion events per day. However, data sets with such a huge size are still a challenging task for current data mining and machine learning methods [124]. Nevertheless, a novel system architecture proposed in this chapter is capable of handling even bigger data sets with data mining methods.

3.1 Classical approach

To build an architecture for high-speed event analysis, the classical SIEM system architecture should be reviewed at first. Such classical basic architecture is presented in Figure 3.1.

The SIEM includes the gatherer component, which collects information from the monitored network. This component receives log messages with from Log Management servers, its own installed agents, sensors and third-party devices and services, such as inventory systems or IDS. Next, the received data are normalised into common log format and stored in the relational database management system

[4]Some contents of this chapter have been published as [44, 45, 121, 122, 123].

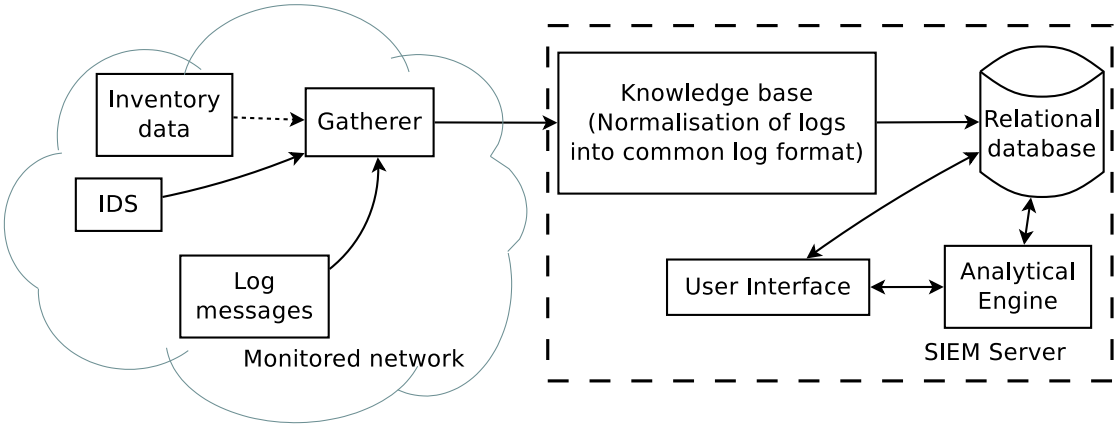


Figure 3.1: Basic SIEM system architecture

(RDBMS). Finally, the data can be queried from the database to be analysed and visualised in the User Interface.

This classical design has some implications on the performance. E.g. Zuech et al. mention that the "traditional RDBMSs are a performance bottleneck for SIEM systems" [13]. Indeed, in case of RDBMS, the data are stored on the disk of the database instance and need to be taken from the database for the analysis, i.e. read into the memory of the Analytical Engine for the analysis or returned to the User Interface as a query result.

The other components, such as Gatherer, Knowledge Base and Analytical Engine, should also be capable of performing high-speed processing. The improvements, proposed for the classical architecture, are reviewed in the section below.

3.2 Proposed approach

To improve the performance of the classical SIEM architecture, nearly every component was reviewed and redesigned. The updated architecture is presented in Figure 3.2.

The architectural changes shown in Figure 3.2 are listed below:

- First of all, the Gatherer and Normalisation (Knowledge base) components were improved and can now reach the speed of 145,000 events per second [125].
- Next, the common log formats were tested to select optimal data model for the normalised events.
- Then, the database backend was replaced with in-memory SAP HANA [39] database.

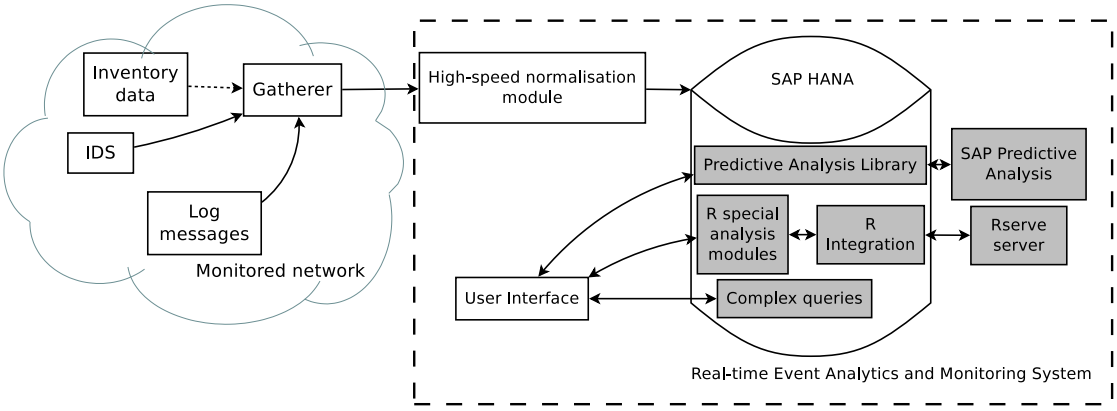


Figure 3.2: Proposed SIEM system architecture

- Finally, the Analytical Engine was redesigned and all analytic modules were moved to the database platform. Thanks to the in-memory database backend, all data mining and machine learning-based analytics, as well as query-based analytics, can be executed either directly in the memory of the database backend, or in the memory of Rserve server [126]. Thus, all the data are stored and analysed in-memory, which significantly improves the processing speed.

As mentioned earlier, the high-performance event gathering and normalisation are out of the scope of this thesis. These topics were reviewed by Jaeger et al. in [125]. All other improvements are reviewed in details in the subsections below.

3.2.1 Optimal log format for security events normalisation

To find out the best data model for normalised security events, the common log formats reviewed in Section 2.3.1 should be compared with each other. To perform such a comparison, existing log formats are evaluated using the following criteria:

- *Scalability.* The log format for normalisation of security messages should be scalable, i.e. there should be an ability to add new custom fields for those log messages, which cannot be fully parsed without it.

- *Lightweight.* For the purposes of high-speed normalisation, the log format should have a relatively small number of fields.

- *Multi-level schema.* The multi-level schema allows categorisation of different log message fields into classes, which could simplify event correlation and analysis.

Table 3.1: Existing common log formats

Format name	Organisation	Size estimation	Format structure
CEE (Common Event Expression)	MITRE	58 fields, 7 objects	two levels, hierarchical, object-based
IDMEF (Intrusion Detection Message Exchange Format)	IETF	118 elements, 5 core classes, 53 attributes	multi-level, class-based
IODEF (Incident Object Description Exchange Format)	IETF	53 elements, 19 top-level classes, 83 attributes	multi-level, class-based
CEF (ArcSight Common Event Format)	Micro Focus	8 mandatory fields, 1 extension field with 165 optional fields	one-level, key-value pairs
OLF (Object Log Format)	HPI	107 fields	3 levels, 19 top-level elements

The overview of the existing log formats is provided in Table 3.1. Taking into account the first criteria (scalability), all formats support adding custom fields. Reviewing the second criteria (lightweight), the CEE format has fewer fields in comparison to other formats. However, it is unclear whether just 58 fields are already enough to parse real log messages. Therefore, during the initial development of REAMS architecture, it was decided to test log formats on the real data set from "Scan of the Month" Honeynet Challenge: "Scan 34 - Analyze real honeynet logs for attacks and activity" [127], which is presented in Table 3.2.

Table 3.2: Overview of the log files from the "Scan of the Month" Honeynet Challenge ("Scan 34")

Server	Service	Date	Total number of records
n/a	HTTP Server	Jan 30 - Mar 16	3925
bridge	iptables firewall	Feb 25 - Mar 31	179752
bastion	snort IDS	Feb 25 - Mar 31	69039
combo	syslog	Jan 30 - Mar 17	7620

The Honeynet data set contains real attack data and other log messages collected from different honeypots between 30 January and 31 March 2005. The

Honeynet challenge analyses multiple log files related to different services, which makes the data set a perfect example of heterogeneous log data a SIEM system should deal with. Ideally, the SIEM data model should allow normalisation of all security log messages without adding new custom-defined fields. The Table 3.3, however, presents the parts of the log messages that do not have corresponding fields in the log format.

Obviously, the Object Log Format, which was developed specially for REAMS taking into account multiple log sources, including the Honeynet challenge, can parse every log message without adding any custom fields. Unfortunately, all other log formats need extra custom fields to be able to fully normalise the log messages from the Honeynet challenge. For example, CEE needs 59 extra fields, which doubles its size [45]. Compared to other formats with added custom fields, OLF becomes the most lightweight one, at least for the types of log messages that are regularly processed in REAMS.

Analysing the last criteria (multi-level schema), all formats except CEF, have a hierarchical schema, which simplifies correlation and analysis of normalised events.

Taking into account all three criteria, it was decided to use OLF for normalisation within REAMS. The benefit of the log normalisation into OLF could be demonstrated with the simple attack detection example on the log messages from the Honeynet challenge. To perform the attack detection, the challenge winners used some custom scripts as well as the manual analysis [128, 129]. However, after the data are normalised into OLF, one could use rather simple queries to identify some of the attacks. For example, the query provided in Listing 3.1 searches for Simple Mail Transfer Protocol [130] (SMTP) failures, which are usually caused by SMTP scans.

Listing 3.1: Simple query for detection of SMTP failures in normalised OLF data

```
1 select * from event where application_protocol = 'smtp' and
      tag_status = 'failure'
```

Using this query, it is possible to identify log lines related to SMTP scans from multiple sources, including Snort alerts from the logging server and mail server.

Table 3.3: Parts of log messages without a corresponding field in the log format

Log line	CEE	IDMEF	IODEF	CEF	OLF
81.181.146.13 - - [15/Mar/2005:05:06:53 -0500] “GET //cgi-bin/awstats/awstats.pl?configdir= —%20id%20— HTTP/1.1” 404 1050 “-” ”Mozilla/4.0 (compatible; MSIE 6.0; Windows 98)	HTTP/1.0, GET, 404	Mozilla/4.0, 404	81.181.146.13, GET, //cgi-bin/awstats/awstats.pl, 404, Mozilla/4.0	-	-
Mar 15 13:38:03 combo sshd(pam_unix)[14490]: authentication failure; logname= uid=0 euid=0 tty=NODEVssh ruser= rhost=202.68.93.5.dts.net.nz user=root	-	-	14490, 202.68.93.5 .dts.net.nz, user=root	14490	-
Mar 1 20:45:12 bastion snort: [1:2001439:3] BLEEDING-EDGE WORM Mydoom.ah/i Infection IRC Activity [Classification: A Network Trojan was detected] [Priority: 1]: TCP 11.11.79.67:2568 -> 129.27.9.248:6667	TCP	-	Priority: 1, 11.11.79.67, 129.27.9.248	Priority: 1	-
Mar 24 19:46:50 bridge kernel: INBOUND ICMP: IN=br0 PHYSIN=eth0 OUT=br0 PHYSOUT=eth1 SRC=63.197.49.61 DST=11.11.79.100 LEN=32 TOS=0x00 PREC=0x00 TTL=111 ID=1053 PROTO=ICMP TYPE=8 CODE=0 ID=512 SEQ=29421	ICMP, eth0, eth1, br0	-	SRC = 63.197.49.61, DST = 11.11.79.100, eth0, eth1, br0	br0	-
Feb 1 10:08:32 combo sendmail[32433]: j11F8FP0032433: ruleset=check_rcpt, arg1 = <china9988@21cn.com>, relay=[61.73.94.162], reject=550 5.7.1 <china9988@21cn.com>... Relaying denied. IP name lookup failed [61.73.94.162]	ruleset = check_rcpt, 550, china9988@ 21cn.com	ruleset = check_rcpt, 550, china9988@ 21cn.com	check_rcpt, relay= [61.73.94.162], 550, 61.73.94.162	32433, ruleset = check_rcpt	-

3.2.2 In-memory database backend

After normalisation, log messages in Object Log Format are persisted into the SAP HANA [39]. The SAP HANA is an in-memory platform that includes SQL database, where all data are stored in the main memory of the database instance in compressed form. SAP HANA supports both row- and column-based storage types. If the data are stored in the column-based table, the compression ratio could be higher, since the columns often contain identical values.

Besides the database itself, the platform offers multiple options for in-memory data analysis. First, SAP HANA PAL [40] contains the most popular data mining functions that can be executed directly from SQLSCRIPT [42]. Alternatively, one can use SAP Predictive Analysis[5] [131] to build and execute the analysis algorithms graphically based on the capabilities of SAP HANA PAL.

Another option for in-memory data analysis is SAP HANA R Integration [41], which allows calling R scripts from SQLSCRIPT. To enable it, SAP HANA should be able to connect to Rserve server [126], where all the data will be sent for the analysis. Of course, in this case, the analysed data/table should be transferred to Rserve. However, only the selected data will be copied from the database memory to the memory of the Rserve, which can be done relatively fast through the high-speed network connection. Moreover, both Rserve and SAP HANA could be installed on the same hypervisor, which means that the data should only be transferred from the memory of one virtual machine into another. Finally, even though the SAP HANA Integration Guide states that Rserve cannot be installed on the SAP HANA host [41], such a setup is technically still possible and can be used to run both SAP HANA and Rserve directly on the same host and without any hypervisor.

This connection to R provides access to any custom analytics using a variety of R libraries available. This covers not only machine learning and data mining libraries but also the libraries that allow parallelising R computations or executing them in the cluster environment.

All analytical options provided by SAP HANA platform allow operating on the data directly in the memory of the database instance, which improves the performance of the SIEM system in whole.

3.2.3 Hybrid detection approach

The REAMS utilises the hybrid detection approach (see Section 2.2.4) to gain the benefit of both misuse and anomaly detection methods.

The misuse detection, namely the signature-based analytics is the fastest detection method and can, therefore, be applied “on the fly”, directly after the data are

[5]Now renamed to SAP Predictive Analytics.

normalised into the same format. Signature-based analytics module in REAMS operates before the data are persisted into the in-memory database and allows getting the detection results "on the fly" as well. It is also able to detect multi-step attacks in different sources using the same generic signature [70]. This approach significantly reduces the number of signatures in the knowledge base and enables correlation of events coming from different sources.

After the normalised data are saved in the in-memory database, it can be analysed with all the variety of tools available from SAP HANA platform. The basic analytics can be done with queries using SQLSCRIPT. Such queries allow an operator of REAMS to perform custom searches, correlate events manually and calculate the statistics on the data. If an attack was identified using such a custom query but was not caught by a signature, an operator can easily create a new signature based on the query. All available signatures can also be re-applied on the data in the database at any time.

Both signatures and queries are indeed very important and provide the necessary level of attack detection, as well as an overview of the data. However, to detect the previously unknown attacks in an automated way, an anomaly detection should be implemented and applied on the data in addition to the signature- and query-based analytics. As mentioned in the previous section, the anomaly detection can be applied on the data directly in the memory of the database (using SAP HANA PAL and R Integration), which has a positive impact on the performance.

Thus, the data analysis in REAMS is organised in two steps. First, signature-based analytics is applied in the streaming mode on the all incoming data. Second, the data are saved in the database, where it becomes a subject of historical analytics (which includes anomaly detection methods). The aim of this historical analytics is to complement the signature-based detection and to extend the detection rate of the system overall.

3.3 Final architecture of the proposed system

The resulting structure of REAMS (based on the proposed architecture) is shown in Figure 3.3.

To gather the data from the monitored network, REAMS utilises several optional modules. First, the data could be collected from the other Log Management server or SIEM system, such as Syslog-ng [132] or Splunk [133]. Second, another module allows REAMS to gather the Windows Events from any Windows host, including the Domain Controller, where the Security Audit events from the whole Windows domain are usually collected. Finally, the REAMS agent installed on the end systems could also send the security-related events to the Log Gatherer component, running on the REAMS server.

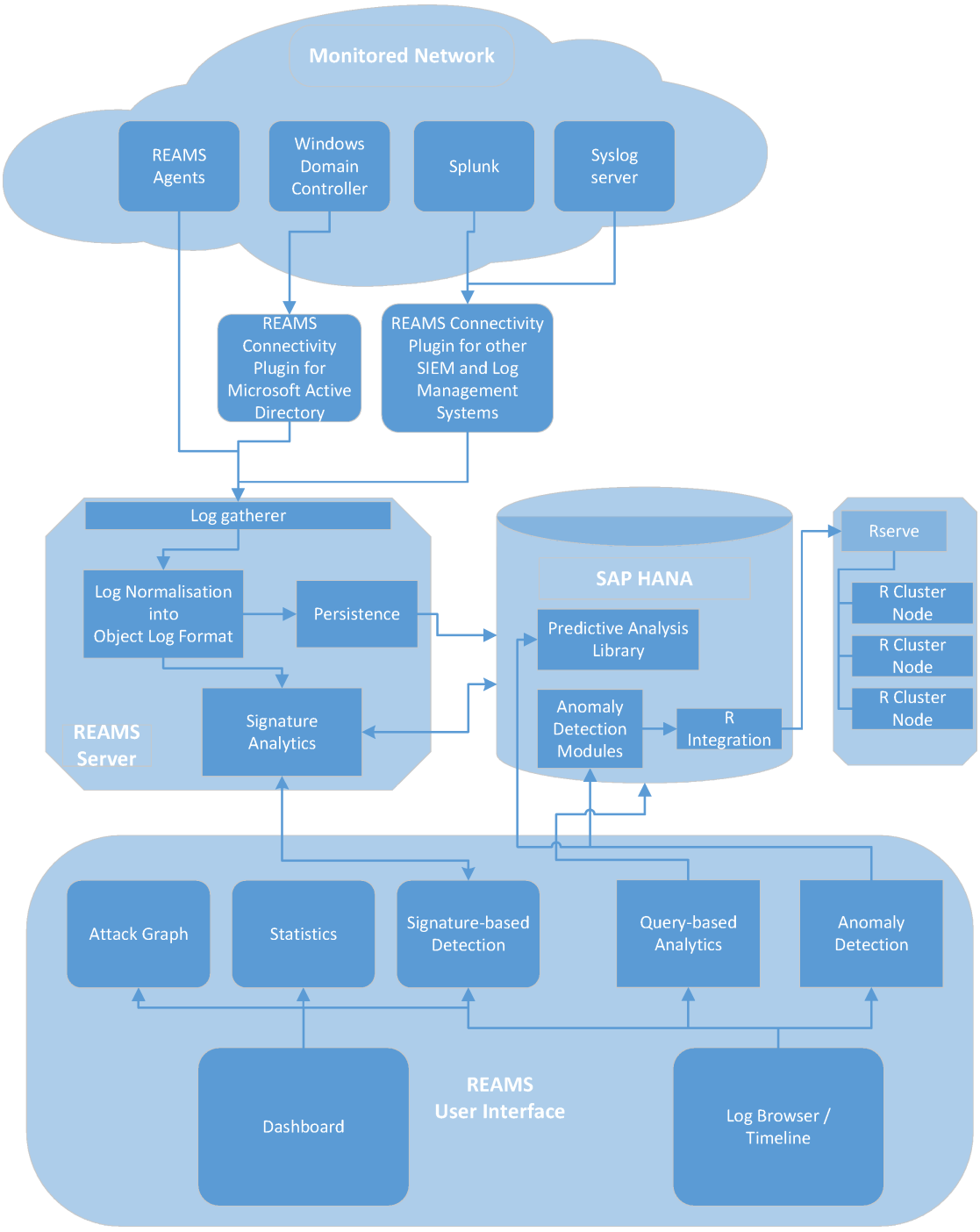


Figure 3.3: Real-time Event Analytics and Monitoring System (REAMS).

Besides its Log Gatherer component, the REAMS server also includes the Log Normalisation module, which normalises all supported source log messages into the Object Log Format. After the normalisation, all messages are immediately sent to both Persistence and Signature Analytics modules. The Persistence module sends the data to the SAP HANA in-memory platform, while Signature Analytics module applies all available signatures on the normalised messages. The detected attacks are immediately shown in the corresponding tab of the REAMS User Interface (on the Dashboard tab).

SAP HANA platform contains SQL procedures with analytical algorithms that utilise either SAP HANA PAL or custom Anomaly Detection Modules, implemented in R programming language. The algorithms utilising PAL are executed directly in the SAP HANA in-memory platform, while custom Anomaly Detection Modules are executed on the R cluster served by a Rserve server after the analysed data are transferred to the Rserve.

All analytical modules can be triggered from the REAMS User Interface, including the Signature Analytics module. The User Interface, therefore, acts as a control centre for the operator of REAMS. The screenshots of the REAMS User Interface are presented in Figures 3.4 and 3.5.

As shown in these Figures, an operator of REAMS can browse the log messages (on the Log Browser tab), filter them and run analytics on the selected subset of events. The Dashboard view allows an operator to see the statistics and results from Signature Analytics module, including an attack graph, which is generated based on the attacks detected using Signature Analytics module.

The screenshots of REAMS User Interface are provided for the reference only since the implementation of the User Interface is not a part of this thesis. Rather, this thesis is focused on the optimal architecture of SIEM system and high-speed anomaly detection. The proposed architecture should be evaluated to prove that it outperforms the classical approach. The performance evaluation provided in the section below answers the question whether the proposed architecture can be taken as a basis for high-speed anomaly detection on Big Data.

3.4 Performance evaluation

Although the in-memory database is expected to demonstrate higher computational performance by design, it is not clear whether the in-memory backend will help SIEM system to significantly outperform the classical architecture. To measure and compare the performance of anomaly detection algorithms with both architecture types, the test data set with injected attacks was generated in the virtual network. This data set was used to evaluate and compare the performance of existing anomaly detection methods in SAP HANA PAL and R.

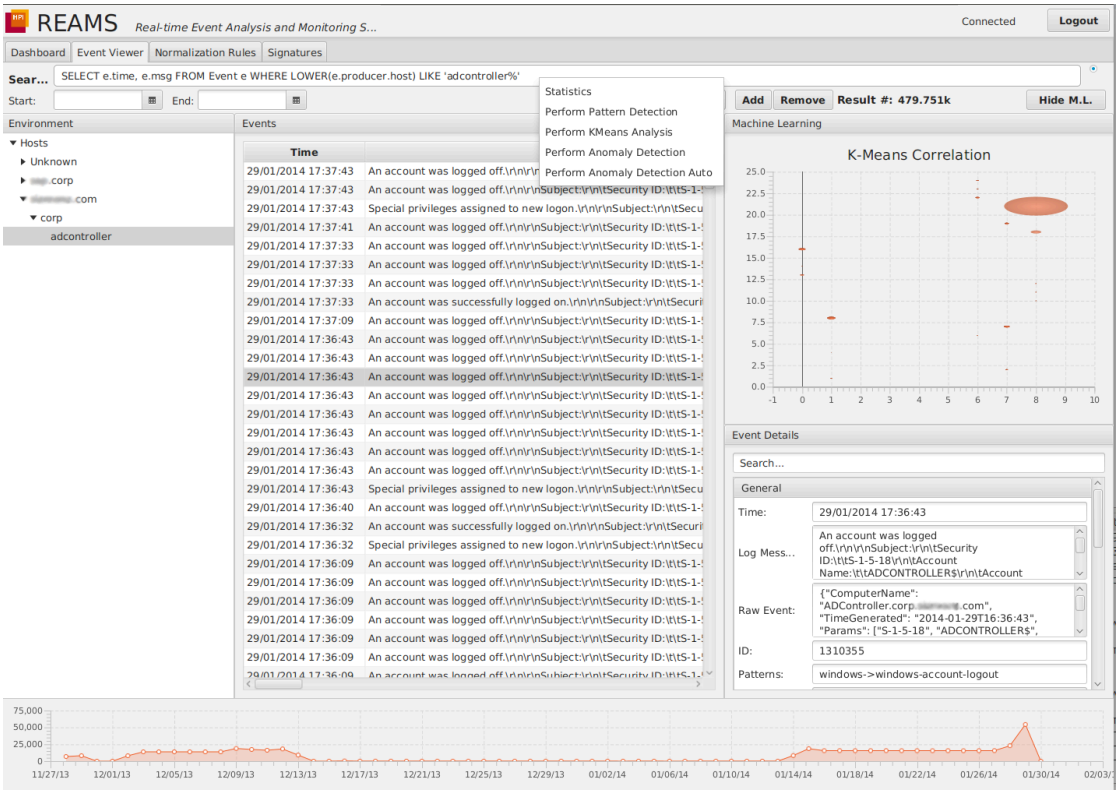
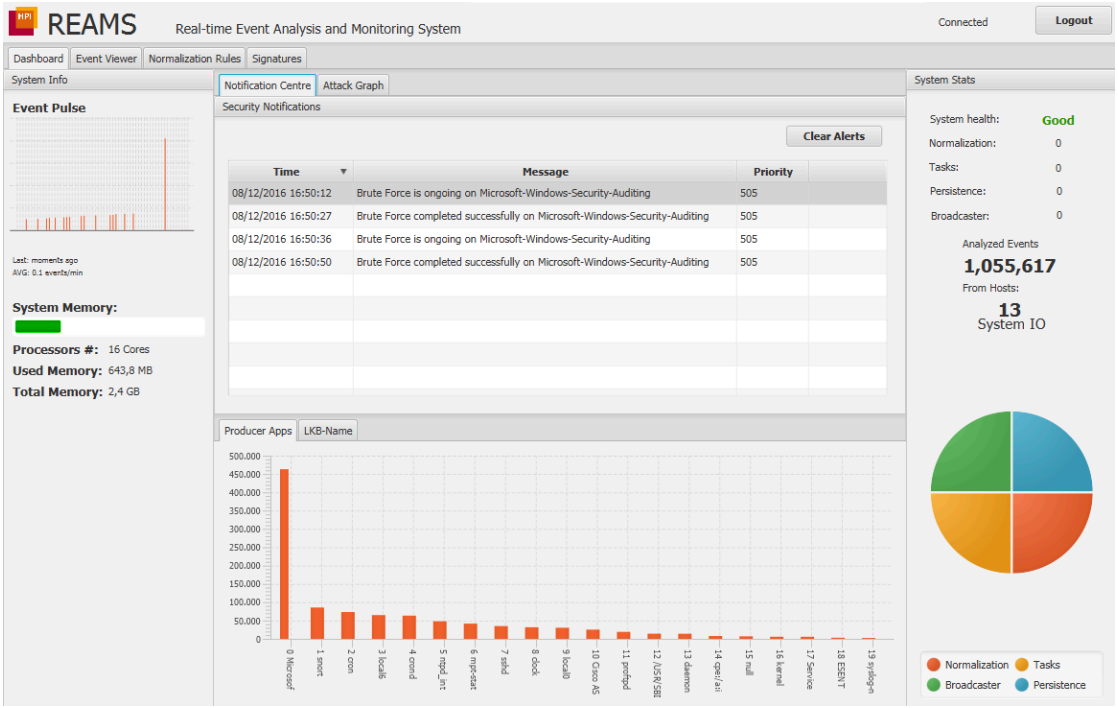
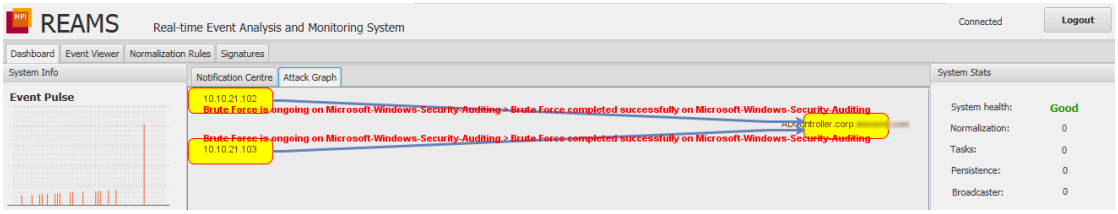


Figure 3.4: REAMS Event Viewer



(a) REAMS Dashboard: main screen



(b) REAMS Dashboard: attack graph tab

Figure 3.5: REAMS Dashboard with brute-force attack detected and shown in the attack graph

3.4.1 Generation of the test data set in the virtual testbed

To generate the test data set, a virtual testbed with Microsoft Windows domain was created. The testbed is shown in Figure 3.6.

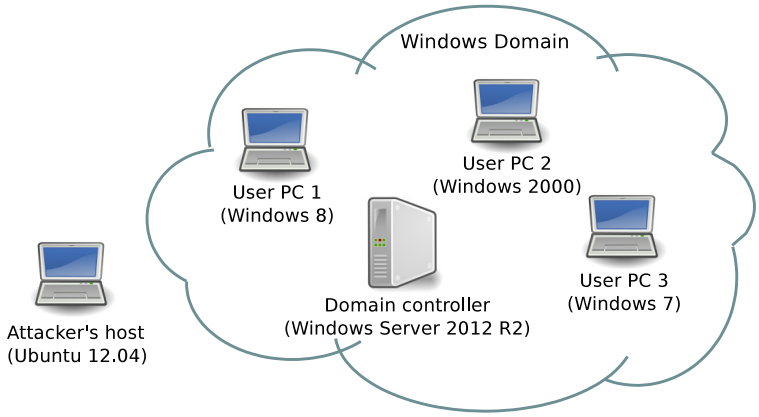


Figure 3.6: Active Directory testbed for generation of the test data set

In Figure 3.6, the virtual network contains 3 user machines and 1 Domain Controller. All 3 users and the administrator of the domain are able to log in to every virtual machine. To collect security-relevant log messages, the Audit policy was enabled on the Domain Controller. The REAMS client, installed on the Domain Controller, was used to forward Windows Events to the REAMS server.

The virtual testbed was used over two periods (first between 28.11.2013 and 13.12.2013; second between 14.01.2014 and 29.01.2014). During this time, several simple attacks were performed manually and from another virtual machine within the same network, but outside the Windows domain. The performed attacks are listed in Table 3.4.

All in all, 477,172 Windows Events were collected, including the attack data. This number includes all Windows Events from the Security log of the Domain Controller. However, some of these events could be filtered out, since only events related to the user behaviour need to be analysed to detect the performed attacks. Thus, only the events with the following Event IDs need to be selected for the performance testing purposes: 4768 ("A Kerberos authentication ticket was requested"), 4769 ("A Kerberos service ticket was requested"), 4771 ("Kerberos pre-authentication failed"), 4776 ("The domain controller attempted to validate the credentials for an account"), 4624 ("An account was successfully logged on") and 4625 ("An account failed to log on"). The distribution of these events is presented in Figure 3.7.

After filtering, the total number of logon-related events in the test data set is

Table 3.4: Attacks in the generated data set

Date	Attack performed
28.01.2014, 13:11 GMT+1	Unsuccessful password brute-force with Hydra [134] via RDP [135] on domain controller
29.01.2014, 10:00-10:10 GMT+1	Successful brute-force of LDAP [136] using Hydra
29.01.2014, 12:30-13:10 GMT+1	Manual password brute-force in the console of the User PC 2 (Windows 2000)

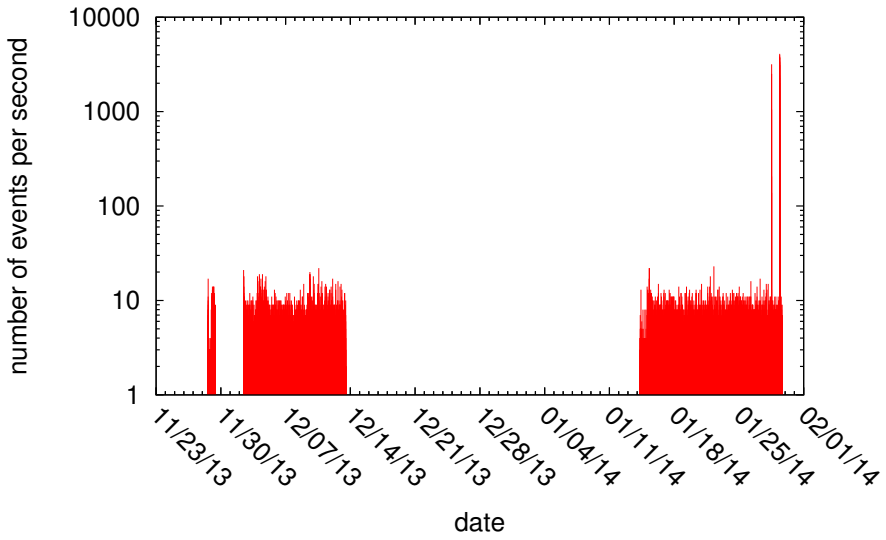


Figure 3.7: Distribution of logon-related events in the data set

188,457, which is a relatively small data set for performance tests. To increase the number of records for the performance tests, the data set was replicated. The replicated data sets are described in Table 3.5.

Since every Windows Event contains a timestamp, the replication should not generate the duplicate events with the same timestamp that can confuse anomaly detection algorithms. Therefore, during the replication, the timestamp was always updated.[6]

[6]Each event's timestamp was updated in such a way that the time span of unreplicated data set (from the previous replication step, originally 65 days between 28.11.2013 and 29.01.2014) in days was added to it.

Table 3.5: Size of the data sets for performance tests

Number of events	Number of events with selected Event IDs
477,172	188,457
954,344	376,914
1,908,688	753,828
3,817,376	1,507,656
7,634,752	3,015,312
15,269,504	6,030,624
30,539,008	12,061,248
61,078,016	24,122,496
122,156,032	48,244,992

3.4.2 Selecting features from Windows Events

The replicated data set was normalised into Object Log Format and persisted into the SAP HANA database. However, many fields could be excluded from the analysis of the user behaviour based on this data set. Taking Windows Event ID 4624 ("An account was successfully logged on") as an example, many fields, such as 'Provider Name', 'Opcode', 'Task', 'Keywords', 'LogonProcessName', 'Key Length', reflect system information or other event data and could not help to detect attacks described in Table 3.4.

Therefore, only the most relevant fields were selected as features for the performance evaluation. The mapping of these fields from Windows Event schema into Object Log Format is presented in Table 3.6.

As also shown in Table 3.6, different Windows Events have different schemas, depending on the Event ID and provider [120]. It makes the normalisation more complicated, for example, 'TargetSid' for EventID 4768 and 'TargetUserSid' for EventID 4771 should be mapped to the same field in the Object Log Format. In total, 13 features from the Object Log Format data model are selected to perform data analysis during the performance evaluation.

Initially, the selected textual features were simply mapped to numbers to evaluate the computational performance[7] of anomaly detection algorithms and system architectures using the environment described in the section below.

[7]The processing of textual features for outlier detection will be discussed in details in the next chapter of the thesis.

Table 3.6: List of selected features as stored in Object Log Format

	Windows Event ID					
Object Log Format fields	4768	4769	4771	4776	4624	4625
subjectUser .userId					Subject UserSid	Subject UserSid
subjectUser .username					Subject UserName	Subject UserName
targetUser .userId	TargetSid		Target UserSid		Target UserSid	Target UserSid
targetUser .username	Target UserName	Target UserName	Target UserName	Target UserName	Target UserName	Target UserName
additional [win.ad.login.type]					LogonType	LogonType
network.srcIpv4 / network.srcIpv6	IpAddress	IpAddress	IpAddress		IpAddress	IpAddress
network.srcHost				Workstation	Workstation Name	Workstation Name
eventTypeId	EventID	EventID	EventID	EventID	EventID	EventID
time	Time Created	Time Created	Time Created	Time Created	Time Created	Time Created
producer.host	Computer	Computer	Computer	Computer	Computer	Computer
application .statusCode	Status	Status	Status	Status		Status
additional [win.ad.sub.status]						SubStatus
additional [win.ad.failure.reason]						Failure Reason

3.4.3 Environment for performance measurements

To test anomaly detection algorithms with both classical and proposed architectures, two setups were prepared and compared to each other:

- *Classical architecture setup* is based on the PostgreSQL RDBMS [43]. This classical architecture requires the data to be read from the database for the analysis. Thus, in this setup, the main analytical module runs on the Rserve server and queries the data from PostgreSQL.
- *Proposed architecture setup* utilises SAP HANA in-memory platform. Under this approach, the data are analysed directly in the memory of the database. In the case when additional analytical modules should be executed on Rserve server, no data are queried by those modules from Rserve. Rather, SQLSCRIPT running on the SAP HANA sends the data to the Rserve server and initiates the analysis.

Both the analysis of the test data set and all performance measurements using classical and proposed setups were executed in the virtual environment described in Table 3.7.

Table 3.7: System configuration for performance tests

System	Operating System	CPU	HDD	RAM
Hypervisor	VMware ESXi 5.5.0	2x Intel Xeon E5-2660, 2.2GHz, 32vCPU	1,2TB SAS 6Gbit/s 10k	128 GB 1.600MHz RDIMM
Database VM SAP HANA SPS06 / PostgreSQL 9.4	SLES for SAP Applications 11.3 (x86_64)	16 vCPU	320 GB	80 GB
Rserve (R 3.0.2) VM	openSUSE 11.4 (x86_64)	8 vCPU	40 GB	16 GB

The database and Rserve servers were installed into virtual machines on the VMware ESXi hypervisor [137]. The database VM was initially configured to be able to run SAP HANA SPS06 database. Since SAP HANA stores all data in the memory, 80 GB RAM was assigned for the virtual machine. The disk space was allocated according to the SAP HANA installation guide (3 times bigger than

the amount of RAM). To implement the classical architecture, the PostgreSQL RDBMS was installed on the same VM and used when the SAP HANA database instance was shut down. The PostgreSQL settings were changed to utilise as much RAM as possible. In particular, the following modifications were done in the 'postgresql.conf' file: *shared_buffers = 32GB*, *temp_buffers=8GB*, *work_mem=32GB*, *max_stack_depth=4MB*.

The Rserve has lower hardware requirements and needs less RAM as well. During the data analysis, it was found that 16 GB RAM was sufficient for the Rserve server instance to perform the necessary analysis. The disk space was set to the default setting of 40 GB for GNU/Linux VM in VMware ESXi since the Rserve was not using a disk for the computations.

To estimate the performance and compare classical and proposed approaches with each other, several analysis algorithms directly available in SAP HANA PAL and R were selected.

SAP HANA PAL offers an Anomaly Detection algorithm, which is based on the k-means clustering. Since the implementation of this Anomaly Detection algorithm is proprietary and the same algorithm is not available in R, it was decided to also compare the performance of underlying k-means algorithms from both SAP HANA PAL and R ('kmeans' function from package "stats").

To run the Anomaly Detection and K-Means[8] algorithms in SAP HANA PAL, the SAP Predictive Analysis software was used. The example scenario for running Anomaly Detection on the data is presented in Figure 3.8.

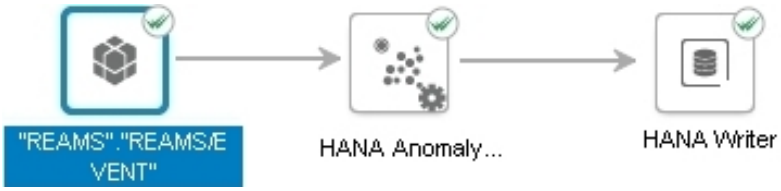


Figure 3.8: Anomaly Detection scenario in SAP Predictive Analysis

To perform the data analysis in SAP Predictive Analysis, the processing blocks should be dragged-and-dropped onto the document sheet. The first block accesses the data in the specific table in SAP HANA, the second one performs the Anomaly Detection, and the last one writes the results to another table in SAP HANA. On click on the Run button, the SAP Predictive Analysis transforms the scenario into the SQLSCRIPT commands with the corresponding SAP HANA PAL function calls and submits them to the SAP HANA platform, where the script is executed.

The Anomaly Detection module was configured with the following parameters:

- percentage of anomalies for Anomaly Detection: 10%

[8]Here the specific implementation of k-means from SAP HANA PAL is meant.

- anomaly detection: by the sum of distances from all centres
- normalisation type: based on row values
- distance measure: Euclidean Distance
- number of clusters: 8
- maximum number of iterations: 10
- number of threads: 8

The same options were used for K-Means function (except the percentage of anomalies and the type of anomaly detection since they are not applicable for the K-Means).

The k-means function available from R is executed with R script using the same parameters (except the number of threads, which is not applicable). The R script is presented in Listing 3.2.

Listing 3.2: Example of SAP HANA procedure using R integration

```
1 CREATE PROCEDURE KMeansR(IN table1 INPUTTABLE, OUT result
    RESULTTABLE)
2 LANGUAGE RLANG AS
3 BEGIN
4 cl <- kmeans(table1, centers = 8)
5 result <- as.data.frame(cbind(table1,CLUSTERNUMBER=
    cl$cluster))
6 END;
7 CALL KMeansR(INPUTTABLE, RESULTTABLE);
```

The whole R script contains just two lines (4 and 5), where the first one executes k-means, which returns cluster numbers. The second line of the R code adds identified cluster numbers to the original table.

To execute the same R functions in the classical architecture, the R script should contain code to query the data from the database. This updated script is shown in Listing 3.3

Listing 3.3: Example of R script processing data from PostgreSQL

```
1 library(RPostgreSQL)
2 drv <- dbDriver("PostgreSQL")
3 con <- dbConnect(drv, user="anomalies", password="anomalies
    ", host="192.168.0.1", port="5432", dbname="anomalies")
4
5 tab <-dbReadTable(con,"normalised_events")
```

```
6  features <- cbind(tab$SUBJECTUSER, tab$TARGETUSER,
       tab$FAILUREREASON, tab$LOGONTYPE, tab$IPADDRESS4,
       tab$IPADDRESS6, tab$WORKSTATION, tab$EVENTID,
       tab$UNIXTIME, tab$COMPUTER, tab$STATUS, NAOK=TRUE)
7
8  cl <- kmeans(features, centers = 8, iter.max = 10, nstart =
       1)
9
10 result <- as.data.frame(cbind(tab,CLUSTERNUMBER=cl$cluster)
       )
11
12 if (dbExistsTable(con,"cluster_with_r")){
13         dbRemoveTable(con,"cluster_with_r")
14 }
15
16 dbWriteTable(con,"cluster_with_r",result)
17
18 dbDisconnect(con)
```

The second R script is larger since it should create the database connection (lines 1-3), retrieve the data and prepare the features (lines 5-6), and write back the results (lines 12-16)[9].

To monitor the execution time and resources during the performance tests, various tools were used in the prepared environment, as listed below:

- SAP HANA Studio [138] was used to measure the execution time of scripts running in SAP HANA.
- SAP Predictive Analysis was used to measure the execution time of the Anomaly Detection and K-Means scenarios.
- R was used to print out the execution time of k-means clustering for the classic architecture scenario.
- 'top' tool was used on both Database VM and Rserve VM to measure CPU usage and memory usage.
- 'sar' tool (from 'systat' package) was used on both Database VM and Rserve VM to measure I/O Wait and CPU usage.

[9]Different to this, in the proposed architecture all steps except running the k-means clustering are performed directly in the database using SQL queries that were not shown in the Listing3.2.

After the environment for the performance tests was set up, one last step before running the tests is needed. It is important to check whether the selected anomaly detection and clustering algorithms are not only able to identify outliers and cluster the data fast, but also able to produce explainable results. The next subsection describes the initial results received with the algorithms selected for the performance testing.

3.4.4 Initial anomaly detection results

The Anomaly Detection from SAP HANA PAL was applied on the data, which were normalised into OLF. The results are shown in Figure 3.9.

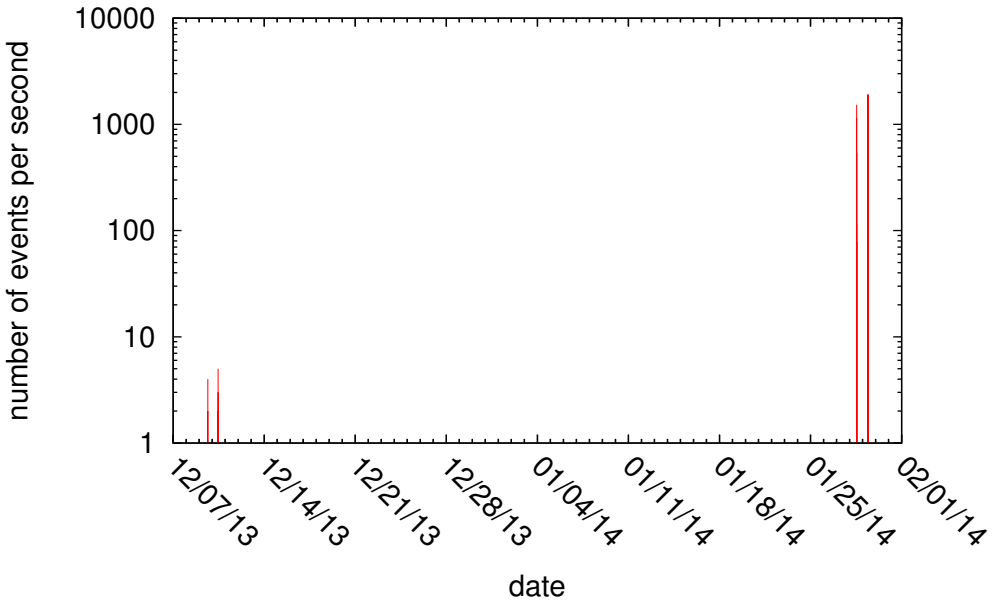
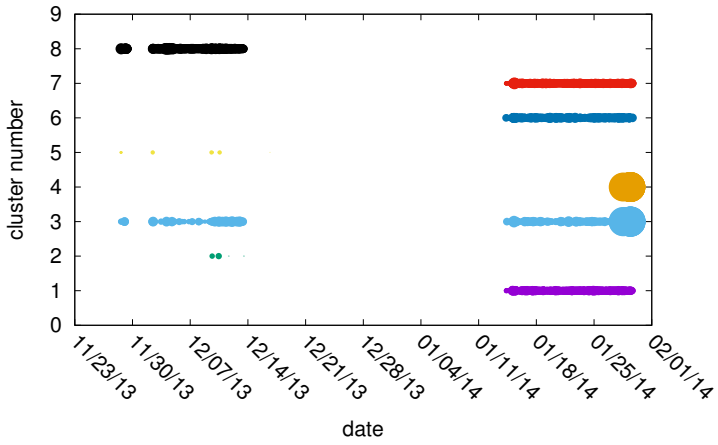


Figure 3.9: Results from Anomaly Detection algorithm

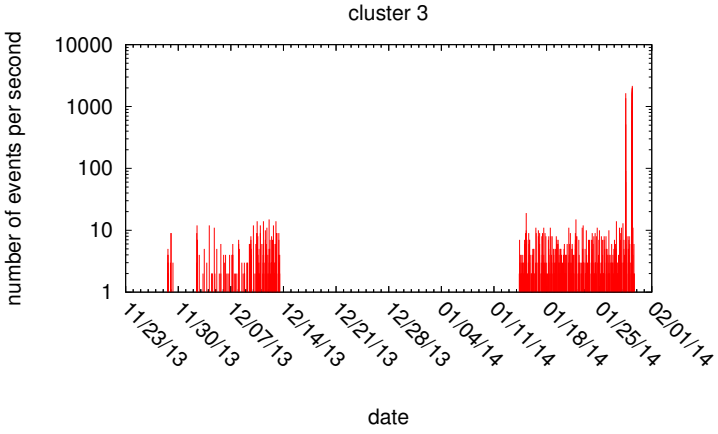
The Anomaly Detection algorithm was able to identify both password brute-force attacks from Table 3.4. These two attacks are represented with spikes in the right side of the chart. The two other small spikes on the left side are false positives and contain benign Windows Events. The appearance of false positive alerts is caused by the preset percentage of anomalies in the data (10%), which should be guessed for this algorithm.

Next, the K-Means algorithm from SAP HANA PAL was used to cluster the data.[10] The results of the clustering are presented in Figure 3.10.

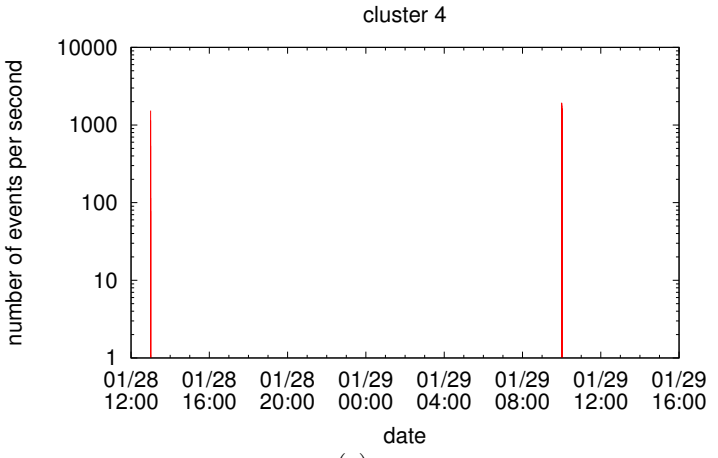
[10]K-means algorithm in R produced similar results, which are omitted.



(a)



(b)



(c)

Figure 3.10: Results from the k-means algorithm with 8 clusters

Figure 3.10(a) shows the distribution of Windows Events among 8 clusters. The size of the bubble represents the number of events (log-scaled) per second. The biggest bubbles in the chart correspond to the clusters 3 and 4. The cluster 3 is shown in Figure 3.10(b) and contains all brute-force attacks from Table 3.4 (including the manual one, which was not identified by the Anomaly Detection), as well as other benign events. The cluster 4, shown in Figure 3.10(c), contains events related to two automated brute-force attacks only.

Thus, both Anomaly Detection and K-Means algorithms from SAP HANA PAL, and also k-means from R are able to produce explainable results on the testing data set. With the parameters used, the Anomaly Detection algorithm correctly identified both automated brute-force attacks, but also produced a relatively small amount of false positive alerts (see Figure 3.9) and did not detect the manual password brute-force. The K-Means algorithm correctly clustered brute-force attacks together (similar to k-means from R). The attacks were classified into 2 clusters, even though one of the clusters with attacks contained a lot of benign events.

The performance of the selected algorithms in both classical and proposed architectures was tested in the prepared environment. The results of the performance evaluation are presented in the subsections below.

3.4.5 Performance of classical architecture

In the classical architecture, the SIEM system queries the data from the database for the analysis. In our environment, the R script (see Listing 3.3) gets the data for the k-means clustering from the PostgreSQL database. The performance of the k-means with PostgreSQL backend is shown in Figure 3.11.

The algorithm needed only 5 seconds to analyse the original unreplicated data set with 188 thousand of events. With the increase of the data set size, the execution time grows nearly linear (x-axis is log-scaled). However, the biggest data set size that could be analysed using classical architecture was 12 million events that were processed within 8 minutes. On the bigger data sets, the k-means was constantly failing due to an insufficient amount of memory. Increasing the memory of Rserve virtual machine from 16 to 32 GB did not solve the issue, and the performance tests were stopped on the data set with 12 million events. The possible cause of the failures could be the 'RPostgreSQL' library [139], which is required to query the data from PostgreSQL database.

The results of the server monitoring during the performance tests are presented in Figure 3.12.

From Figures 3.12(a) and 3.12(b), it is possible to conclude that PostgreSQL database always had many resources during the data analysis. Only maximum of 10% CPU was used and I/O Wait was also quite low. This shows that the database

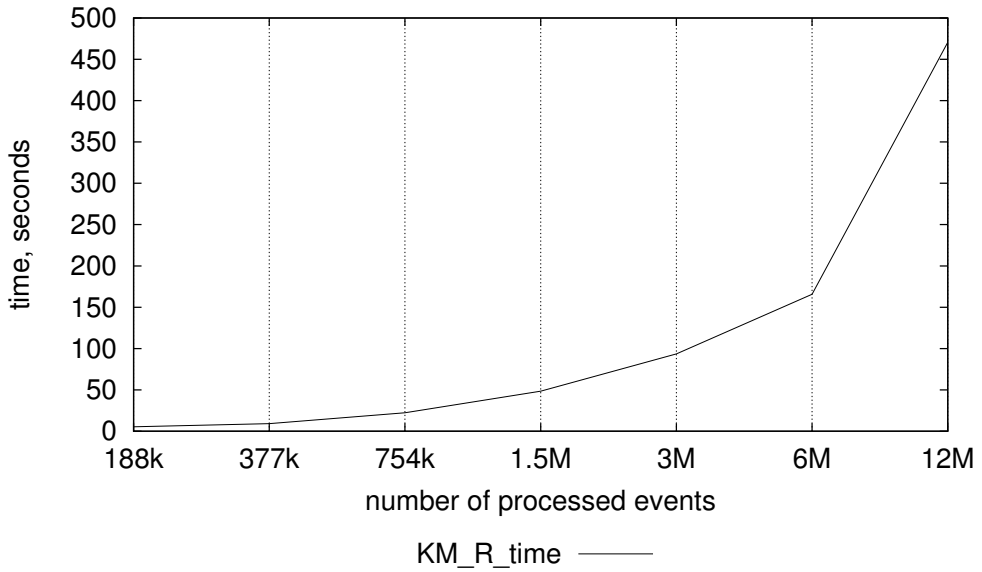


Figure 3.11: Performance of k-means algorithm on the test data set using PostgreSQL as a backend, x-axis log-scaled

was able to cache nearly all data in the memory and did not reach any resource limitations. The CPU usage on the Rserve VM was also relatively low and never exceed 13% (of 16 CPU cores available[11]), see Figure 3.12(c). Since all data were processed in the memory of the Rserve VM, the I/O Wait chart is provided just for the reference. In Figure 3.12(d), the I/O Wait values different from zero are caused by insufficient amount of RAM and indicate that the swap was being used during the data analysis.

3.4.6 Performance of proposed architecture

The performance of proposed architecture, where the data are either analysed directly in the in-memory database or sent to Rserve for the analysis, is demonstrated in Figure 3.13.

Figure 3.13 shows the execution time for 3 algorithms: Anomaly Detection in SAP HANA PAL, K-Means in SAP HANA PAL and k-means running on the Rserve server. Using the same hardware, it was possible to analyse the data sets with size up to 48 million events, compared with 12 million for classical approach. The k-means algorithm from Rserve also failed due to memory limitation, but this time only on the biggest data set. Even though it failed on the data set with 48 million events, it was possible to analyse the data set with 24 million events

[11]The k-means from "stats" package in R does not support parallel processing

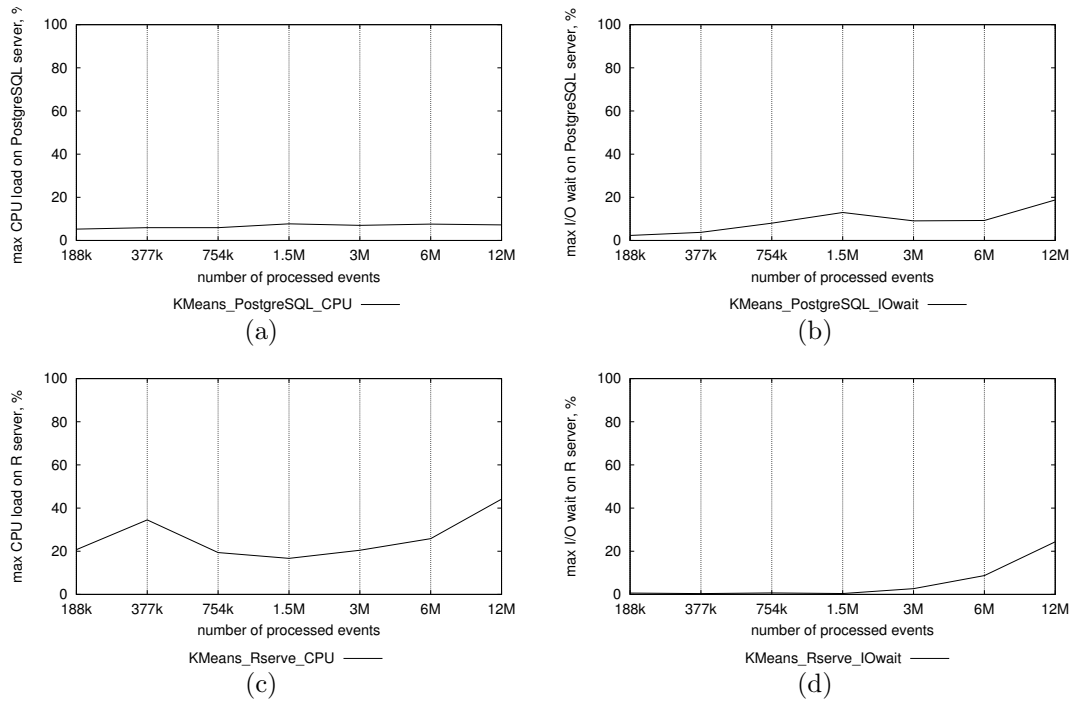


Figure 3.12: CPU usage and I/O Wait during performance measurements using PostgreSQL as a backend, x-axis log-scaled

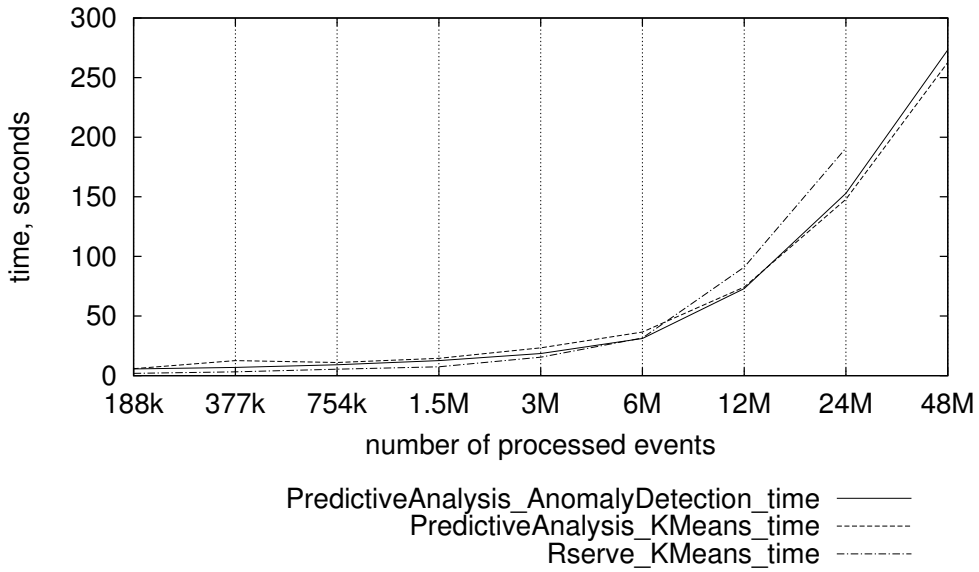


Figure 3.13: Performance of machine learning algorithms on the test data set, x-axis log-scaled

using the same configuration of the Rserve server, as was used to test the classical architecture. The only difference was the usage of 'RPostgreSQL' library [139] in classical approach, which seems to be the cause for the increase of the memory requirements.

The interesting observation is that k-means in R demonstrates similar performance to the K-Means from SAP HANA PAL, despite the fact that the data should be transferred to the Rserve server and the fact that the k-means from R used 1 CPU thread only, against 8 threads used by the parallel k-means implementation from SAP HANA PAL. This fact shows the potential to further improve the performance of algorithms in SAP HANA PAL.

The execution time of all three algorithms also grows linearly. The 48 million events were analysed within 5 minutes. On the data set with 12 million events, all algorithms are approximately 6 times faster than classical approach (91 against 470 seconds). However, on the smaller data sets the difference is not so high, e.g. classical architecture with PostgreSQL database backend is just 2.8-2.9 times slower than the proposed one with the in-memory backend.

Of course, the execution time of Anomaly Detection and underlying k-means algorithm grows linearly if only one parameter (e.g. data set size) is being changed[12]. In case of the growing data set size, it is often needed to increase the number of clusters for k-means. To check, how the performance of the data analysis will

[12]Since the k-means algorithm is NP-hard.

change with a different number of clusters, K-Means from SAP HANA PAL was tested on the data set with 24 million events. The results of these tests are shown in Figure 3.14.

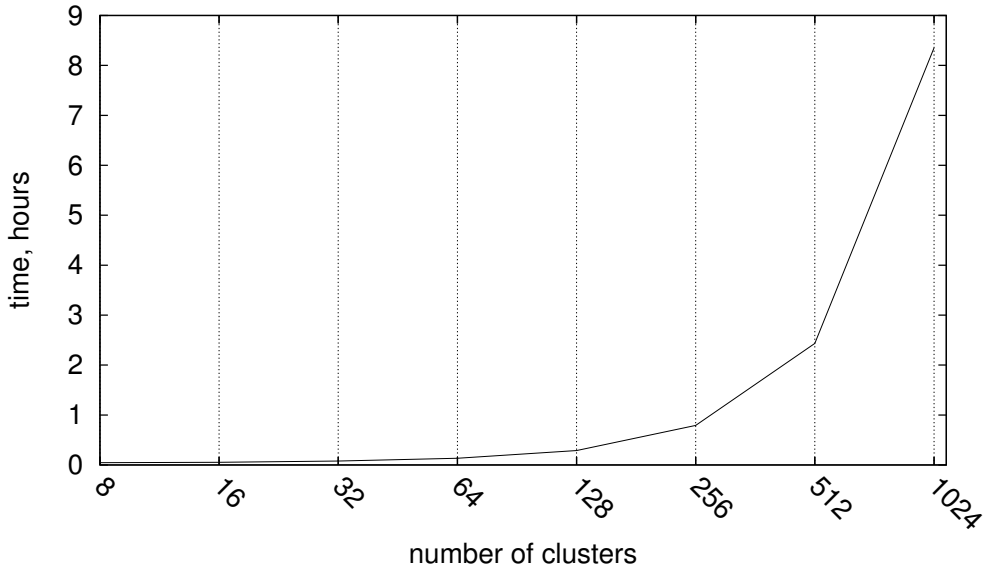


Figure 3.14: Performance of K-Means algorithm from SAP HANA PAL with different number of clusters, x-axis log-scaled

Here the execution time also grows linearly (x-axis is log-scaled), at least until 512 clusters. The chart shows that the high number of clusters requires much more execution time, e.g. 8 hours to cluster 24 million events into 1024 clusters. Still, if the data does not contain too many clusters, the proposed architecture and in-memory database backend allow processing big amounts of data with a high speed, e.g. 8 minutes for processing 24 million events with 64 clusters using only 8 CPU threads.

Even though only 48 million events could be analysed with selected algorithms due to the memory limit (80 GB RAM), it is worth to mention that the database stored much more data in its memory. To process 48 million events, 122 million replicated events with original log lines were stored at the same time in the database (see table 3.5). Besides that, there were tables with results and other auxiliary tables needed for Anomaly Detection. The utilisation of both database and Rserve virtual machines during the performance tests is presented in Figure 3.15.

The monitoring of server resources shows that only CPU can be a limiting factor for the algorithm's performance. However, as shown in Figure 3.15(a), the maximum of 100% of CPU usage is only reached on the bigger data sets,

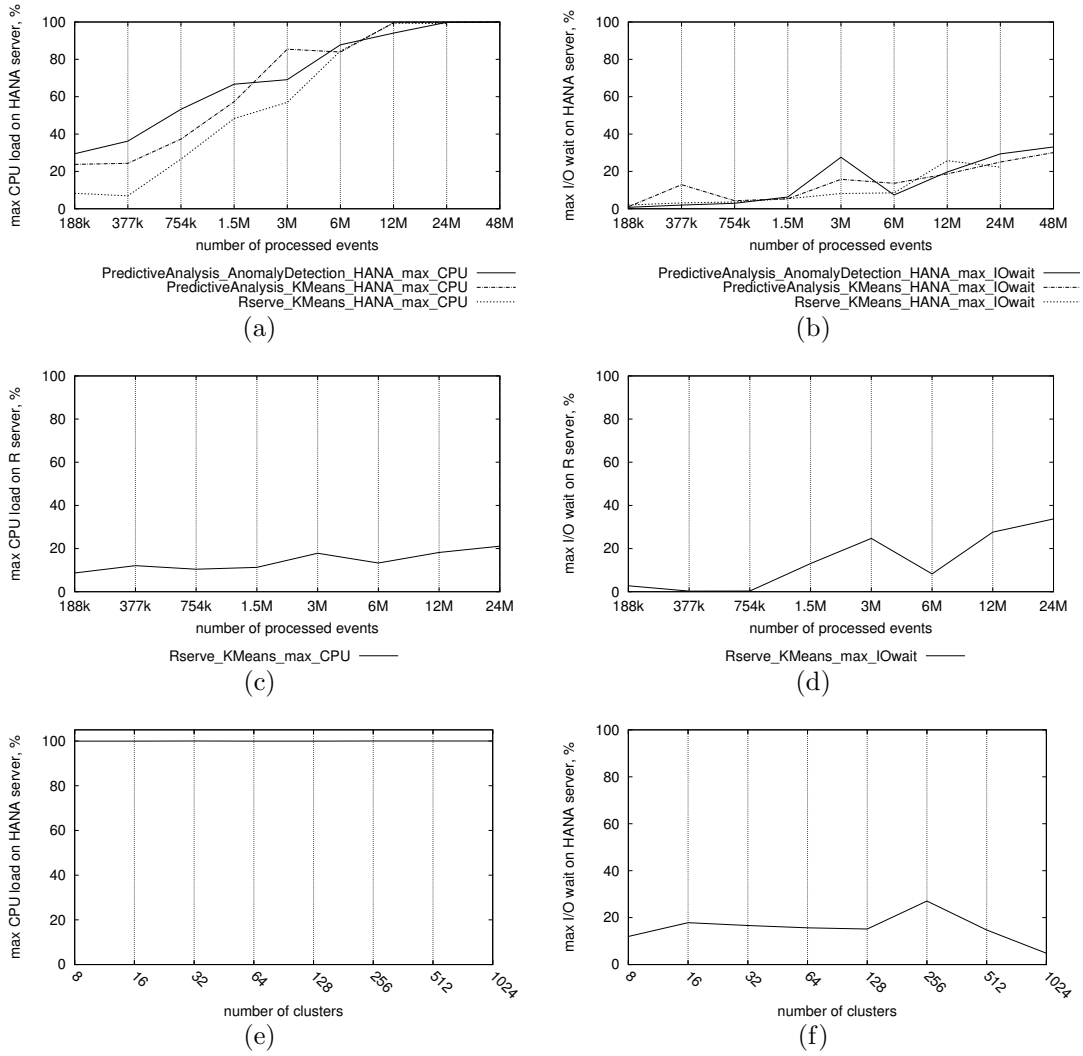


Figure 3.15: CPU usage and I/O Wait during performance measurements, x-axis log-scaled

starting from 12 million events. Even though the CPU utilisation always reaches the maximum for bigger data sets (see also Figure 3.15(e)), the execution time remains reasonable and grows nearly linear. The CPU usage on the Rserve server is always low (see Figure 3.15(c)), because the implementation of k-means from R is single-threaded. The I/O Wait is also not so high since all data processing is performed in the main memory (see Figures 3.15(b), 3.15(d) and 3.15(f)).

3.5 Chapter summary

This chapter presented the novel architecture of the SIEM system, based on the in-memory data processing, normalisation into common log format and hybrid detection approach.

The performance tests have proved the suitability of the proposed architecture for high-speed analysis of security-related events. The execution time for the Anomaly Detection and underlying k-means algorithm on the data set containing 48 million events is less than 5 minutes (approximately 175,000 events per second). If it will continue to grow linearly, such a performance allows processing of up to 15 billion events per day. This performance could be enough to satisfy needs of the large enterprises, such as EMC [18]. The enterprises that collect up to 1 trillion security events per day [17], but are able to analyse only up to 3% of it, can still benefit from the proposed architecture since it may allow them to increase the percentage of analysed events.

Compared with the classical architecture, the proposed one could achieve up to 6 times higher processing speed on the same hardware, especially on the larger data sets. The main limitation of the proposed approach is the relatively large amount of main memory needed for the data storage and the analysis. However, taking into account the fact that some additional data were stored in the database during performance tests (see Section 3.4.6 for details), the memory usage could be further optimised.

The comparison with the k-means algorithm from R shows that algorithms from SAP HANA PAL, namely Anomaly Detection and K-Means could potentially be optimised. For example, k-means implementation in R has nearly the same performance as K-Means from SAP HANA PAL, even though it uses only 1 CPU against 8 for SAP HANA PAL.

In the next Section, the Anomaly Detection algorithm from SAP HANA PAL will be analysed to propose a more advanced anomaly detection approach, in terms of both high-speed processing and usability of results for a human operator.

Chapter 4

High-speed outlier detection for heterogeneous security events[13]

The results of the performance tests have shown the possibility to analyse Big Security Data with machine learning and data mining methods, such as Anomaly Detection from SAP HANA PAL. To better understand whether such algorithm could be further improved to propose a more advanced, fast and usable approach, it is reviewed in the subsection below.

4.1 Anomaly Detection in SAP HANA Predictive Analytics Library

Anomaly Detection algorithm from SAP HANA Predictive Analytics Library works as follows:

- First, numerical data are scaled (optionally), so that the distance could be calculated properly even for multi-dimensional data sets.
- Next, scaled features are clustered using the well-known k-means algorithm. This algorithm requires to input the number of clusters in the data, number of iterations to perform and the distance measure (Manhattan, Euclidean and Minkowski distances are supported).
- After clustering, data records are ordered either by the distance from the centre of the corresponding cluster or by the sum of distances from all cluster centres.

[13]The results described in this chapter have been published as [122, 123].

- Finally, the top p% of all data records with the biggest distance are marked as anomalies

This outlier detection approach has several disadvantages. Foremost, the Anomaly Detection algorithm only works on the numerical data, whereas most log messages have textual data only. The examples of the textual data are usernames, protocol names, port numbers etc. Therefore, to prepare the data set for the performance tests, textual features were simply projected to natural numbers. This projection could be formally explained as follows [123]:

Let c_j be a multiset and let $\tilde{c_j}/ =$ be a set of all equivalence classes for c_j. Then, let $K = \{1, 2, ..., N\}$ be the index set of $\tilde{c_j}$, so that $\tilde{c_j} = \{a_1, a_2, ..., a_N\} = (a_k)_{k \in K}$. Let's define the function $g : c_j \to \tilde{c_j}$ that returns the equivalence class of an element in c_j, so that $g(x_i) = a_k$. And another function $h : \tilde{c_j} \to K$ that returns the index of that equivalence class, i.e. $h(a_k) = k$. Then the mapping will look like:

$c'_j = \{h(g(x_1)), h(g(x_2)), ..., h(g(x_m))\}$.

In other words, the data set is presented as a collection of columns:

$D = \{c_1, c_2, ..., c_n\}$, where each column is a vector of values:

$c_j = \{x_1, x_2, ..., x_m\}$.

Then each column c_j is converted to new column c'_j by mapping each value to its category. E.g. taking the column of usernames as an example,

$c_j = \{bob, alice, bob, carol, bob, bob, \ldots\}$

will be converted to

$c'_j = \{1, 2, 1, 3, 1, 1, \ldots\}$.

However, this simple conversion still does not guarantee that a clustering algorithm, such as k-means, can be meaningfully applied on the data. To calculate the Euclidean distance between two events/rows, all columns or features of the data set have to be numeric and of the same measure. Even though the text fields, such as usernames, port numbers, protocol names etc. are converted to numbers, they are still not numeric and are not of the same measure. Therefore, they cannot be used for Euclidean distance calculation. Obviously, after such a mapping, the number representation also does not follow a normal distribution, which does not allow applying Mahalanobis distance as well [140][14].

Besides this problem with processing textual fields, the Anomaly Detection algorithm has other issues as well. As indicated during the performance tests, the execution time for this algorithm grows linearly with increasing the number of events to be analysed. However, with increasing number of records, a number of clusters in the data often should also be increased. It significantly increases the computational complexity and potentially makes an algorithm inapplicable to Big Data, if the whole data set has to be clustered at once.

[14]Mahalanobis distance can be used for data sets with features/columns of different measures, but only if all features/columns are normally distributed.

Next, the Anomaly Detection from SAP HANA PAL requires a user to input a number of parameters, such as a number of clusters and percentage of anomalies in the data. The number of clusters is indeed required for k-means; however, one could try to automatically determine it using one of the existing methods [141, 142, 143].

The percentage of anomalies in the data required as input could be very confusing for the data analyst. First, the value of this parameter could be unknown for the data analyst. This is often the case for the unsupervised outlier detection when the knowledge about anomalies in the data is unavailable. To illustrate this problem, the Anomaly Detection from SAP HANA PAL was executed on the artificial sample data set with 2 features (V1 and V2) to detect outliers. The results are presented in Figure 4.1.

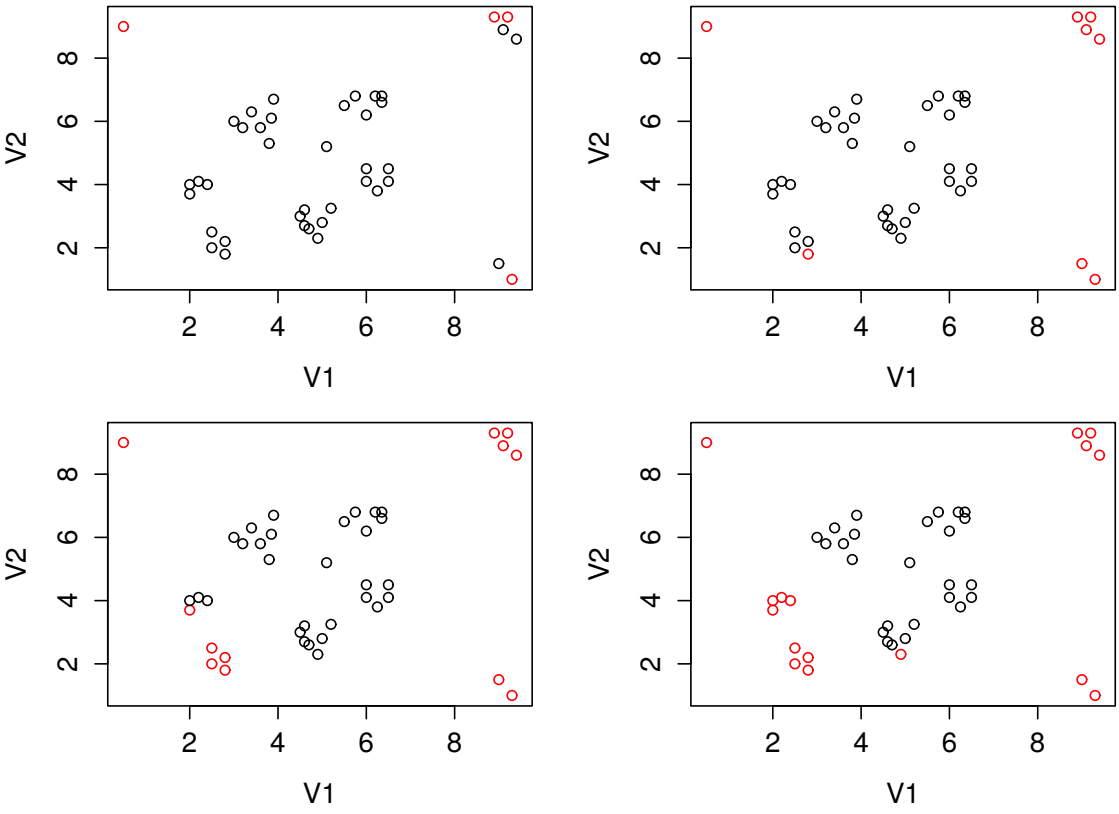


Figure 4.1: Outliers identified by Anomaly Detection from SAP HANA PAL with different percentage of anomalies in the data as the input parameter.

Figure 4.1 shows results of the Anomaly Detection with the percentage of anomalies in the data equal to 10%, 20%, 30% and 40% correspondingly (left-to-right and top-to-bottom), whereas the number of clusters was set to 10. From these figures, it is possible to conclude that the anomalies are not selected in an optimal

way. First, not all data points from the same cluster are marked as anomalies, even though they are located near each other. Second, if the value for the percentage of anomalies in the data is too high (30% or 40%), the points from clusters that does not look suspicious are marked as anomalies. This happens because the threshold is not set based on the distances from the cluster centres; rather it is set based on the number of outliers in the data set, which often stays unknown.

Thus, the Anomaly Detection algorithm from SAP HANA PAL needs to be improved to become more efficient for the analysis of high volumes of security-related events. In terms of processing security log messages, the weak points of the algorithm are the inability to process textual fields, high computational complexity on large data volumes with a high number of clusters and requirements to input number of clusters and percentage of anomalies in the data.

In the next section, the solutions for these problems are offered. These solutions are incorporated into the universal outlier detection approach, which is evaluated on the test data set.

4.2 Outlier detection for textual data based on spherical k-means

The outlier detection algorithm presented in this section is based on the k-means clustering, similar to the Anomaly Detection from SAP HANA PAL, but proposes several changes making it more universal and applicable for the analysis of textual fields. The proposed improvements are described in the subsections below.

4.2.1 Modelling of multivariate non-normal data

To deal with processing of textual/categorical features, all columns mapped to numeric categories (see Section 4.1) are further converted into a Vector Space Model (also called feature binarisation or one-hot encoding), which was introduced by Salton et al. in 1975 [47]. To formally describe this conversion, let us take the data set with all columns mapped to numeric categories:

$D' = \{c'_1, c'_2, ..., c'_n\}$.

Each column $c'_j = \{b_1, b_2, ..., b_p\}$ is converted to the sparse matrix

$$S_{p\times q} = \begin{pmatrix} s_{11} & s_{12} & & s_{1q} \\ s_{21} & & & \\ & & & \\ s_{p1} & & & s_{pq} \end{pmatrix}, \tag{4.1}$$

where $q = max(c'_i)$ and

$$s_{ij} = \begin{cases} 1 \; if \; b_i = j, \\ 0 \; otherwise \end{cases} \tag{4.2}$$

After this step, the sparse matrices for different data set columns are joined horizontally. It results in a matrix with p rows (number of rows in the data set) and $r = max(c'_1) + max(c'_2) + ... + max(c'_n)$ columns.

The vector space model conversion is also illustrated in Figure 4.2.

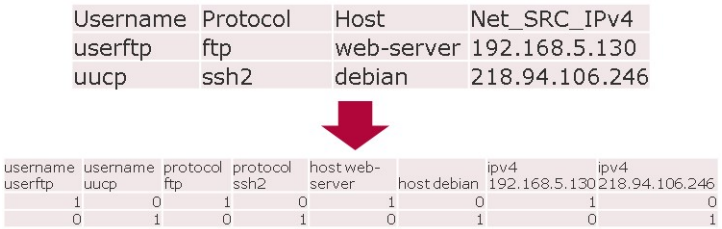


Figure 4.2: Illustrative example of vector space model representation.

As shown in Figure 4.2, each unique value or category from each column becomes a new column. If the original column has this value in row p, the corresponding column in the vector space model will have 1 as a value in row p and 0 otherwise.

This conversion allows correct calculation of the distance between every two data set rows, e.g. using the function of the angle between the corresponding pairs of vectors [47].

However, the resulting vector space matrix could be very large, since each unique IP address, username, protocol and so on becomes a separate column. For the large enterprise networks with tens of thousands hosts and users, this could result in several hundreds of thousand columns. Nevertheless, since this matrix is mainly filled with zeroes, it is possible to use sparse matrix structure to store it, if the sparse matrices are supported by the programming language. In R, this functionality is provided by the "Matrix" package [144], while "skmeans" package [145] provides the implementation of the spherical k-means which uses $d = 1 - cosine\_similarity$ as a distance function (a function of the angle between two vectors representing data set rows).

Still, to reduce the size of this matrix, some simple measures could be performed. For example, all hostnames should be converted to lower case, usernames cast to the same form ($user.name == user.name@exampledomain.edu$) and so on. The timestamp could be divided into multiple features, such as day of the week, hour and minute. In case there are too many unique IP addresses, one could only keep the network address (or 3 of 4 octets) to reduce the number of

columns. Thus, if the number of columns in the sparse matrix becomes too high, some features can be optimised or dropped to reduce it.

4.2.1.1 Incorporating continuous numerical features into the vector space model

Even though the security-related log messages normally contain only textual or categorical fields, such as IP address, username, hostname, protocol and so on, SIEM systems may need to process mixed data containing continuous numerical features as well. An example of such a data set is the KDD Cup 1999 data [146], which contains both categorical and numerical features [76]. Although the numerical features are actually discrete and not continuous (for example, "number of data bytes from destination to source" or "% of connections that have "SYN" errors"), the number of possible discrete values becomes too high for vector space model.

The common approach to process numerical features together with categorical ones is the feature discretisation. Under this approach, the numerical values are distributed into the finite amount of intervals that do not overlap with each other [147]. The original values from the same interval are then replaced with the same categorical value, e.g. the interval number. For the unsupervised outlier detection, which is the focus of this thesis, it is reasonable to utilise the unsupervised feature discretisation. The basic example of such unsupervised discretisation is based on k-means clustering [148]. First, each continuous numerical column is clustered into a predefined number of clusters. Then, each original value is replaced with the cluster number from k-means.

Written formally, before conversion to the vector space, each column containing continuous numerical values $c_j = \{x_1, x_2, ..., x_m\}$ is converted to a new column c'_j by mapping each value to the cluster number from k-means: $c'_j = \{f(x_1), f(x_2), ..., f(x_m)\}$, where function f returns the corresponding cluster number.

Using this *simple discretisation* method, mixed data sets with both numerical and categorical values can be converted to vector space and analysed with spherical k-means. However, the feature discretisation of numerical values implies data reduction/loss, because the similar but still different original values are replaced with the cluster number. These cluster numbers become a separate column in the vector space matrix and contain only zeroes or ones. Thus, if two rows in the data set are the same except the values of the one numerical continuous feature, and these values are clustered into the same category, the spherical k-means distance of these two rows will be calculated as 0.

For example, let's take a mixed data set with one categorical column $c_1 = \{a, b, b, b, a\}$, which is mapped to $c'_1 = \{1, 2, 2, 2, 1\}$ and one column with continu-

ous numerical values $c_2 = \{2.5, 3.7, 3.8, 3.9, 4\}$, which is discretised into 3 clusters as $c'_2 = \{1, 2, 2, 2, 3\}$. Both columns are converted to the following vector space matrix:

a	b	1	2	3
1	0	1	0	0
0	1	0	1	0
0	1	0	1	0
0	1	0	1	0
1	0	0	0	1

Then the cosine similarity, for example, between 2nd and 3rd row will be equal to 1, which means that the difference between 2nd, 3rd and the 4th elements in the column c_1 (originally, 3.7, 3.8 and 3.9) is lost.

However, the cosine similarity used in spherical k-means can also take into account the values in the vector space matrix between 0 and 1, which reduces the information loss during the feature discretisation.

To reduce the information loss, each value of a continuous numeric column should be mapped not to the cluster number, but to the original value, scaled to the unit length with min-max normalisation within the corresponding cluster. Written formally, during the feature discretisation, each column containing continuous numerical values $c_j = \{x_1, x_2, ..., x_m\}$ is converted to k new columns of the same length as c_j, where each

$c'_{jr} = \{f(x_1), f(x_2), ..., f(x_m)\}$, where $r = 1..k$ and

$$f(x_i) = \begin{cases} (x_i - min(X_i)/(max(X_i) - min(X_i)) * C + (1 - C) \\ \qquad if\ min(X_i) <> max(X_i)\ AND\ x_i \in clust_r, \\ 1\ if\ min(X_i) == max(X_i)\ AND\ x_i \in clust_r, \\ 0\ otherwise, \end{cases}$$

where $X_i = \{x_1, x_2, .., x_n\}$ is a set of all values belonging to the same cluster r as x_i and $0 < C < 1$ is a coefficient to avoid discretised values close to 0.

Next, the conversion to the vector space model described in the Section 4.2.1 should be adapted as well, cause for the discretised numerical columns the sparse matrix should be filled not with zeroes and ones, but with values scaled to the unit length. Thus, for the discretised numerical columns, instead of the use of the Equations 4.1 and 4.2, the discretised numerical column $c'_j r = \{b_1, b_2, ..., b_p\}$ can be directly used as a sparse vector and be horizontally joined with the sparse matrices for other data set columns.

With this offered *advanced discretisation* (taking $C = 0.5$), the column $c_2 = \{2.5, 3.7, 3.8, 3.9, 4\}$ should be discretised to $\{c'_{21} = \{1, 0, 0, 0, 0\}, c'_{22} = \{0, 0.5, 0.75,$

$1, 0\}, c'_{23} = \{0, 0, 0, 0, 1\}\}$.

The original mixed data set is then converted to the following vector space matrix:

a	b	1	2	3
1	0	1	0	0
0	1	0	0.5	0
0	1	0	0.75	0
0	1	0	1	0
1	0	0	0	1

Different from before, the vector space matrix constructed with the *advanced discretisation* still distinguishes between the 2nd, 3rd and 4th element of the original numeric column c_1. E.g. the cosine similarity between 2nd and 3rd row is now equal to 0.98 (and not to 1), which shows that the original values were different.

Thus, the offered *advanced discretisation* allows conversion of continuous numerical features into the vector space model with reduced information loss during discretisation. This, in turn, improves the accuracy of the distance function $(1 - cosine\_similarity)$ used in the outlier detection for mixed data sets with both categorical and continuous numerical features.

4.2.2 Clustering security events in parallel

In Section 3.4, the performance tests have shown that the execution time grows nearly linearly if only one of two parameters — the number of events or the number of clusters — is changed. However, the high-speed implementation of Anomaly Detection for Big Security Data with both high number of events and a high number of corresponding clusters is still non-trivial due to the fact that the underlying k-means algorithm is NP-hard. The conversion of textual fields to the vector space model also requires a lot of memory and slows down the computation, even though it could be stored and processed as a sparse matrix.

To limit the number of events and clusters for k-means clustering and increase the processing speed, the data set is divided time-wise into the subsets with the same number of events. Each subset is clustered with k-means in the separate thread. Then the outliers can be determined by the distance (cosine dissimilarity) from the concept vectors [52] of all clusters from all subsets.

Of course, the clustering of the subsets is not as precise as the clustering of the full data set. However, it enables parallel clustering, limits the number of events and clusters for each processed subset and increases the overall algorithm's performance.

4.2.3 Outlier threshold based on the distribution of distances

Anomaly Detection from SAP HANA PAL defines the threshold for outliers based on the percentage of the anomalies in the data. Different to this, it is possible to select outliers based on their distance from concept vectors of all clusters, which has a geometrical sense. In particular, the threshold is set to the nth percentile of the distribution of the distances of outliers from all concept vectors of all clusters.

For each cluster $C \subset S$, where $C = \{sc_1, sc_2, ..., sc_r\}$, the vector of column means is calculated as $cm = \{\overline{sc_1}, \overline{sc_2}, ..., \overline{sc_r}\}$. Next, the concept vector [52] cv is calculated as well:

$$cv = \frac{cm}{\sqrt{\sum_{j=1}^{m} cm_j^2}}$$

Then, for every event e its cosine dissimilarity to each concept vector of each cluster from all subsets is calculated:

$$dissimilarity_{e,cv} = 1 - \frac{cv \times e}{\sqrt{(cv \times cv) * (e \times e)}}$$

Thus, the outlier score of each event is equal to the sum of its cosine distances to all concept vectors:

$$outlier\_score_e = \sum_{i=1}^{t \times k} dissimilarity_{e,cv_i},$$

where t is the number of time intervals (or data subsets) and k is the number of clusters in each subset corresponding to the time interval.

Next, the threshold is set to the nth percentile of the outliers' distance distribution:

$$threshold = max(outlier\_score) - \frac{100 - n}{100}(max(outlier\_score) - min(outlier\_score)),$$

where n could be selected by the data analyst.

This approach is illustrated in Figure 4.3 (please also see the artificial example from Section 4.1, as well as Figure 4.3 for reference).

The distance-based threshold could be represented by circles around the geometric centre of all cluster centroids, as shown in Figure 4.3. The Figure shows the data from artificial example and possible thresholds based on the nth percentile of distance distribution (drawn as red circles). Such a threshold allows selecting

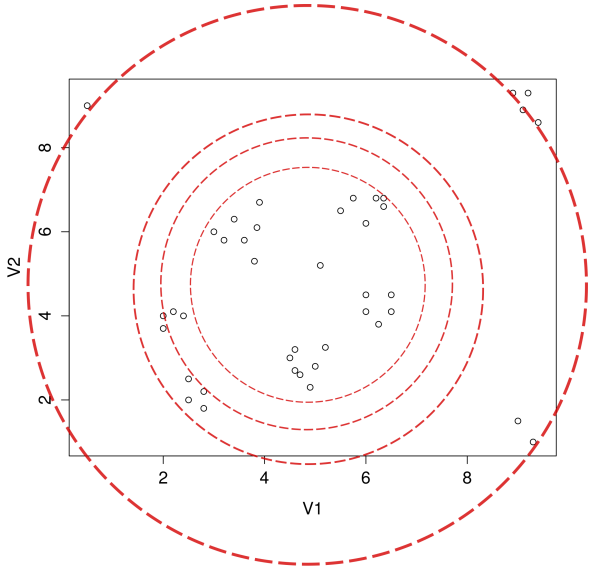


Figure 4.3: Outlier detection thresholds based on the distribution of distances from concept vectors

outliers by their distance, e.g. to select all events with the outlier score (distance from all concept vectors), which is higher than 99% of events have. Thus, the threshold has geometrical sense and is much more intuitive for the data analysis.

4.2.4 Detecting optimal number of clusters for k-means

As mentioned in Section 4.1, a variety of methods is available to automatically determine the number of clusters in the data [141, 142, 143]. To prove the concept of finding an optimal k for Big Security Data, the most simple and natural generic approach called "Elbow Method" was selected and adapted to apply on the data converted to the vector space model. This approach is very similar to one mentioned by Sugar and James in [141] or Salvador and Chan in [149]. The method analyses the "distortion curve" or evaluation function, which represents the measure of cluster dispersion (for example, average Mahalanobis distance between each data point and the corresponding cluster centre) for a different number of clusters. The point where the distortion curve levels off (or the maximum curvature point) is taken as the optimal number of clusters.

The particular implementation of the "Elbow Method" used for security-related data converted into the vector space model is described below.

First, t subsets are selected from the original data (see Section 4.2.2) to execute spherical k-means for each of them with a different number of clusters (k). When clusters are determined, average cosine similarity is determined for all clusters.

For each cluster C from data subset S, where $C = \{sc_1, sc_2, ..., sc_r\}$, the vector of column means $cm = \{\overline{sc_1}, \overline{sc_2}, ..., \overline{sc_r}\}$ is determined to calculate the concept vector [52] cv:

$$cv = \frac{cm}{\sqrt{\sum_{j=1}^{m} cm_j^2}},$$

where m is the number of rows in the data subset.

Then, for every cluster row cr, its cosine similarity to its concept vector is calculated as follows:

$$similarity_i = 1 - \frac{cv \times cr}{\sqrt{(cv \times cv) * (cr \times cr)}}$$

Next, the mean cosine similarity of each row with its concept vector for a defined k is calculated and used as a similarity measure:

$$similarity\_measure = \frac{1}{m \times t \times k} \sum_{i=1}^{m \times t \times k} similarity_i,$$

where m is the number of rows in the data subset, t is the number of data subsets and k is the number of clusters in each subset.

The result of these calculations is a set of tuples {k,similarity_measure} for all values of k. The optimal k is calculated as the maximum curvature point of the interpolated function based on this discrete set of tuples. To find the maximum curvature point based on a discrete set of tuples, the second-order central difference is calculated for each discrete point (tuple):

$$\delta^2_{sm_i} = sm_{i+2} - 2 * sm_{i+1} + sm_i,$$

where sm is a similarity measure.

The maximum value $max(|\delta^2_{sm}|)$ indicates the point with the maximal curvature ("elbow/knee point"). Then, the value of k from the corresponding tuple can be taken as an optimal number of clusters, please see Algorithm 1 for details.

Algorithm 1 calculates the value of the second-order central difference for first $n - 2$ points from *similarities* vector and finds its absolute maximum — point with maximum curvature (lines 1-4). The corresponding k value (line 5) would be optimal according to the described method.

4.2.5 Proposed Universal Outlier Detection approach

The improvements for Anomaly Detection algorithm described in Sections 4.2.1-4.2.4 are merged together into the Universal Outlier Detection algorithm, which is

Algorithm 1 ElbowPoint(similarities,k_values)

for i = 0 **to** length(similarities)-2 **do**
  $\delta^2_{similarities_i} = |similarities[i+2] - 2 * similarities[i+1] + similarities[i]|$
  if max_difference $< \delta^2_{similarities_i}$ **then**
    max_difference $= \delta^2_{similarities_i}$
    OptimalK = k_values[i]
  end if
end for
return OptimalK

Algorithm 2 Universal_Outlier_Detection

determine optimal number of clusters
divide data into N subsets
for all {subsets} from data **do**
  sparse_subset = convert_to_vector_space(subset)
end for
for all {sparse_subset} from sparse_subsets **do**
  clusters = spherical_kmeans(sparse_subset,num_clusters, method='pclust',nruns=3)
  for all {cluster $C[x_{ij}]\_mn$} from clusters **do**
    colmeans = $\{colmean_1, .., colmean_n\}$, where $colmean_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$
    concept_vector = $colmeans / \sqrt{\sum_{j=1}^{m} colmeans_j^2}$
  end for
end for
for all {sparse_subset} from sparse_subsets **do**
  for all {subset_row} from sparse_subset **do**
    for all {concept_vector} from concept_vectors **do**
      similarity = $= 1 - \frac{concept\_vector \times subset\_row}{\sqrt{concept\_vector \times concept\_vector} * \sqrt{subset\_row \times subset\_row}}$
    end for
    Distance = $\sum similarity$
  end for
end for

presented in Algorithm 2.

The Algorithm 2 calculates the distance for each event from all concept vectors of all clusters for all data subsets, whereas the clustering can be executed in parallel. Thanks to the conversion into the vector space model and feature discretisation, the algorithm now works with any type of data: continuous numerical, textual and mixed.

To select the outliers, the threshold should be defined as a percentile of the distances distribution (see Section 4.2.3 for details). All events with the distance higher than the threshold will be then marked as outliers. The proposed algorithm was evaluated on the same testbed data set with (please see data set described in Table 3.5 from Section 3.4.1) with 188 thousand events. The next section shows the results of the evaluation of the proposed algorithm.

4.2.6 Threats detected in the testbed data set

The proposed Universal Outlier Detection algorithm was able to detect the attack from the second row in Table 3.4. When the threshold was set to a 99,99th percentile of distances' distribution, the algorithm returned 18 events that were all related to this attack. One of these events is listed below:


```
{"ComputerName": "ADController",
"TimeGenerated":"2014-01-29T10:04:43",
"Params":["MICROSOFT_AUTHENTICATION_PACKAGE_V1_0", "Administrator",
"ADCONTROLLER", "0xc000006a"], "Sid": null, "EventID": 4776,
"TimeWritten": "2014-01-29T10:04:43",
"SourceName": "Microsoft-Windows-Security-Auditing",
"EventType": "AUDIT_FAILURE", "EventCategory": 14336,
"LogType": "Security", "Message": "The computer attempted
to validate the credentials for an account.
Authentication Package:
MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
Logon Account:Administrator
Source Workstation:ADCONTROLLER
Error Code:0xc000006a",
"RecordNumber": 953526,
"Data": ""}
```


The error code '0xc000006a' in the event means that the password was wrong. Thus, the proposed outlier detection algorithm was able to identify the failed authentication events even though the Error Code indicating that the authentication failed due to the wrong password was not included as a feature into the data set for outlier detection (see Section 3.4.2).

If the outlier threshold is decreased to the 99,9th percentile of distances' distribution, 189 events are marked as anomalies. The summary of these events is shown in Table 4.1.

Table 4.1: Results from Universal Outlier Detection on the testbed data set.

Number of events	Type	Relation to attacks
56	Audit failure, Event ID 4776	Brute-force attack on LDAP
16	Logon failure, Event ID 4625	Brute-force attack on LDAP
13	Audit success, Event IDs 4768 and 4769	False positive
104	Successful logon, EventID 4624	False positive

Table 4.1 contains both events related to the brute-force attack and benign events marked as outliers (false positives). The false positives are, however, quite easy to filter out, as soon as these events are only of two types: audit success and successful logon.

All in all, it is possible to conclude that the proposed Universal Outlier Detection algorithm was able to detect malicious behaviour in the data set.

However, the performance of the proposed approach is not so high. The algorithm divided the data set of 188,427 events into 19 samples of 10,000 records each (except the last sample, which contained 8,427 events). These samples were then clustered with 19 threads during the algorithm's execution.[15] The execution time was 24 minutes[16], which is much slower than 3-4 seconds execution time needed for Anomaly Detection from SAP HANA PAL.

Since the determined number of clusters was equal to 8, which is not a high number, the reason for the low performance is the conversion of textual features into vector space model and its further processing. Even though the offered Universal Outlier Detection allows processing bigger data sets, since it is easy to parallelise (for example, with 100 threads, the same period of 24 minutes would be enough to process the data set containing approximately 1 million events), the

[15]The Universal Outlier Detection was executed not on the environment for performance measurements described in Table 3.7, but on the environment with Intel Xeon E7-8870 CPU, due to technical reasons.

[16]The computation of the optimal number of clusters is not taken into account since it should be only performed once and only for limited number of subsets before the execution of main algorithm.

measured performance should be further improved to make this algorithm applicable to large data volumes.

Both Anomaly Detection from SAP HANA PAL and proposed Universal Outlier Detection have one more disadvantage, which is generic for outlier detection algorithms and especially relevant for the Big Data. The outlier detection often produces too many anomalies and the large volume of such events makes their analysis more time-consuming for the human operator. Even though these anomalies are ranked by distance from concept vectors or another outlier score, the similar events are still not grouped together. In the simple case of brute-force attack, it could easily lead to the situation when the output of the outlier detection algorithm is flooded with failed authentication events. These events are often mixed with other types of anomalies, which makes the analysis of the outlier detection results especially uncomfortable for the human operator.

Both issues — low performance and a large number of ungrouped outliers — are addressed in the next section, which proposes the Hybrid Outlier Detection algorithm.

4.3 High-speed detection of clusters with anomalous events

To group similar outliers together in the outlier detection output, the algorithm can be changed in the way that it will detect the whole clusters of events as outliers. Such implementation of outlier detection requires a chain of at least two machine learning / data mining methods. The first one determines clusters in the data, whereas the second one finds outliers among these clusters (for example, taking cluster centres or concept vectors to describe each cluster).

This approach can be theoretically faster than Universal Outlier Detection on Big Data since it avoids calculating the distance between each record and all cluster centres. Instead, the outlier detection itself is done on the smaller data set that includes cluster centres only. To further speed-up the algorithm, the training of outlier detection algorithm can be performed on the reduced data set, selected, for example, by random sampling without replacement [150].

The implementation of this approach is described in the section below.

4.3.1 Hybrid Outlier Detection: one-class SVM ensemble trained on clusters from spherical k-means

The base part of the Hybrid Outlier Detection algorithm is similar to the proposed Universal Outlier Detection: the data are divided into subsets, which are clustered

in parallel using spherical k-means. However, to be able to detect clusters of outliers, this clustering should be extended with another outlier detection method.

For this purpose, a one-class SVM outlier detection algorithm was selected. One-class SVM produces a clear outlier score, namely the decision value, which is a distance from the hyperplane around the "normal" class (data used for the training). It is also possible to create a separate SVM model for the concept vectors of clusters from each training data subset in parallel. The ensemble of such SVM models will produce a joint outlier score based on scaled decision values. The resulting Hybrid Outlier Detection algorithm is presented in Figure 4.4.

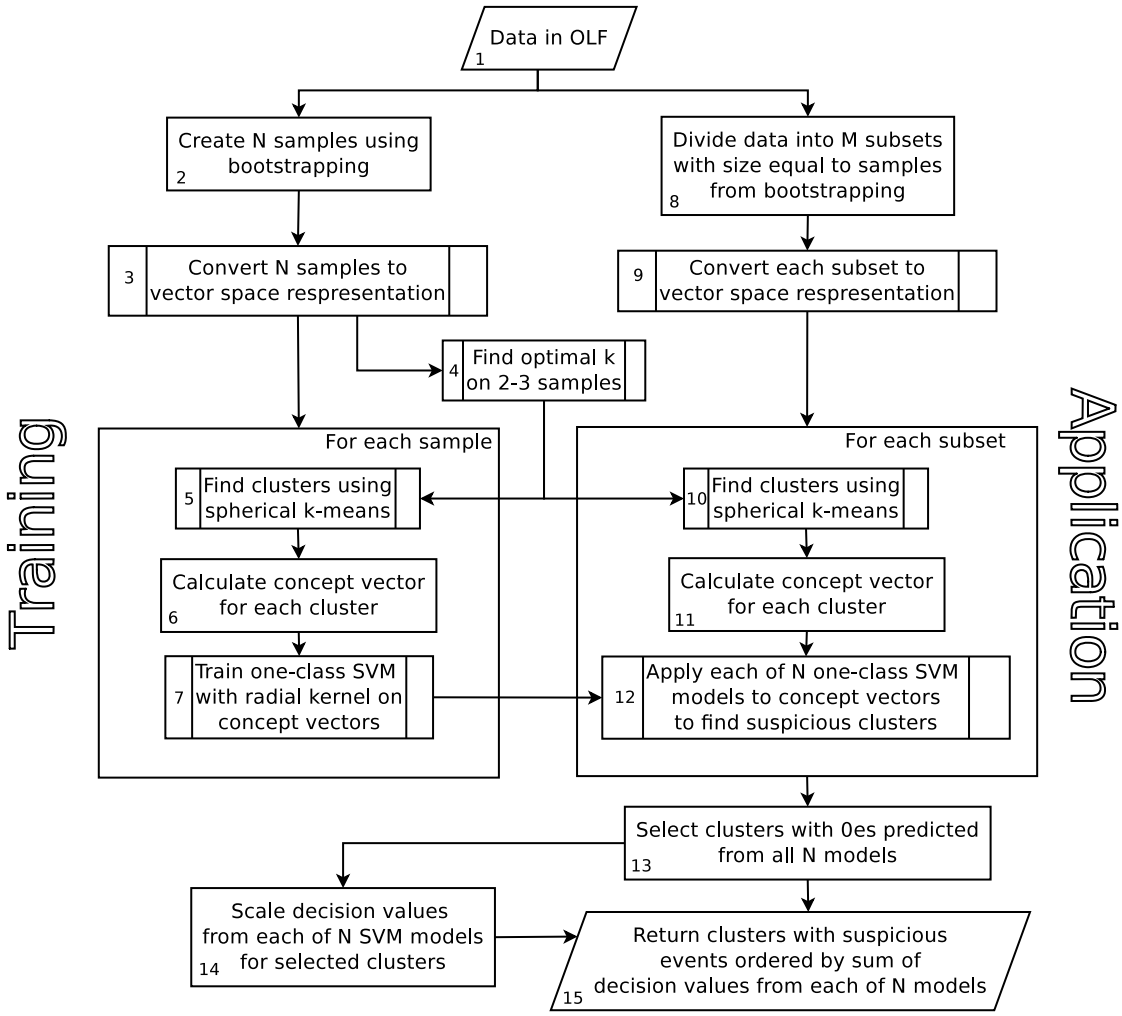


Figure 4.4: Hybrid Outlier Detection scheme

The Hybrid Outlier Detection algorithm showed in Figure 4.4 consists of 15 steps that could be described as follows:

- In **step 1**, the data, which is stored in OLF, are selected for the analysis.
- Using these data, N training samples are generated in **step 2**. The sample generation mechanism could be different, e.g. one could take first 50% (or less) of the full data set and use it for training to find outliers in the second 50% of the data set. However, to improve the speed of the training phase, the random sampling without replacement is proposed. The N randomly generated samples could cover a smaller part of the data set and thus require less training time. On the other hand, the randomly selected samples should be still sufficient to train the model, which is precise enough[17], and even improve the quality of the outlier detection compared to the one trained on the full data set [150, Section 3.2].
- In **step 3**, the samples are converted to the vector space model, as described in Section 4.2.1, which allows processing of both textual and continuous numerical (after its discretisation) data.
- In **step 4**, the optimal value of k (the number of clusters in a sample) is determined based on several samples. To determine the k, the approach described in Section 4.2.4 is used.
- In **steps 5-7**, the N training samples are used to train an ensemble of N one-class SVM models. Each training sample is clustered with spherical k-means. Next, the concept vectors are calculated for each cluster (see Section 4.2.4 for details). Finally, to train each of N one-class SVM models, the concept vectors from the corresponding training samples are used.
- The **steps 8 and 9** are executed to prepare subsets for the detection of outliers. Each subset (maybe except the last one, if the full data set size is not divisible by sample size) should have the same size and the same number of clusters as N training samples have.
- **Steps 10-11** are executed for each subset and are identical to steps 5-6.
- After clustering and calculation of concept vectors, each concept vector (corresponding to one cluster) is evaluated by each of N one-class SVM models at **step 12**. An SVM model classifies the concept vector and returns 0 if it is an outlier, together with the decision value.
- In step **13**, only clusters classified as outliers by all N SVM models are selected. It means that out of N training samples, no sample contains a cluster with the concept vector similar to the classified one. Such an outlier

[17]The effectiveness estimation will be described later in this chapter.

cluster should be relatively rare in the data set, since it was not captured by the random sampling without replacement.

- In **steps 15 and 16** the scaling of the decision values is performed to rank clusters by the sum of decision values (this step will be described in details in the next subsection). Thus, the clusters with higher rank have a higher dissimilarity with clusters from all N training samples and should contain the rarest events in the data set.

The offered Hybrid Outlier Detection is easy to parallelise and should be much faster than Universal Outlier Detection since the outlier detection is now performed on the reduced data set with concept vectors only. Another important feature of this algorithm is the ability to output the ranked clusters, which is described in the subsection below.

4.3.1.1 Ranking of clusters with anomalies

Each cluster is characterised by its concept vector, which is evaluated by an ensemble of SVM models at step 12 of the Hybrid Outlier Detection. To rank these clusters, it is required to calculate the outlier score, which is an SVM decision value in case of a single SVM model. However, N SVM models applied to each concept vector produce decision values of a different scale, due to the fact that these models were trained on N different samples. To be able to sum up the decision values from different models, the scaling should be performed [151].

For this purpose, the decision values from the SVM models can be presented as a matrix

$$DV_{t\times N} = \begin{pmatrix} dv_{11} & dv_{12} & & dv_{1N} \\ dv_{21} & & & \\ & & & \\ dv_{t1} & & & dv_{tN} \end{pmatrix},$$

where elements are SVM decision values and t is the number of outliers.

These decision values are then scaled by the standard deviation without centring using function $f : \mathbb{R} \to \mathbb{R}$, where

$$f(dv_{ij}) = \frac{dv_{ij}}{\sigma_j}$$

and σ_j is the standard deviation for column j of matrix DV, so that:

$$\sigma_j = \sqrt{\frac{1}{t}\sum_{i=1}^{t}\left(dv_{ij} - \frac{1}{t}\sum_{i=1}^{t} dv_{ij}\right)^2}$$

The resulting matrix is

$$DV'_{t\times N} = \begin{pmatrix} dv'_{11} & dv'_{12} & dv'_{1N} \\ dv'_{21} & & \\ & & \\ dv'_{t1} & & dv'_{tN} \end{pmatrix},$$

where $dv'_{ij} = f(dv_{ij})$.

This matrix is used to get a set of summarised scaled decision values for each outlier from different SVM models $SumDV = \{sdv_1, sdv_2, ..., sdv_3\}$, where

$$sdv_i = \sum_{j=1}^{N} dv'_{ij}.$$

The set $SumDV$ can be used as a set of outlier scores to rank the outliers accordingly.

Thus, the Hybrid Outlier Detection allows an operator of the SIEM system to look on the most suspicious events, which are grouped together into clusters and ranked by the sum of scaled decision values from an SVM ensemble.

To prove the benefits of the proposed Hybrid Outlier Detection, it was compared with other outlier detection methods on the real data set from the large multinational enterprise, as well as on the KDD Cup 1999 data. Both of these data sets are described in the next section.

4.3.2 Windows Events data set from enterprise network and KDD Cup 1999 data

The ability of the proposed Hybrid Outlier Detection to identify suspicious events is checked on the real data set from the large multinational company. This data set mainly includes Windows Events from a network segment of this company covering the period of three months. The Windows Events were originally collected by Domain Controllers with enabled Security Audit policy and then forwarded to the ArcSight SIEM system. Besides the Security Windows Events, the data set also includes "System and Application Events", as well as ArcSight messages. The overview of the data set is provided in Figure 4.5.

The data set presented in Figure 4.5 includes 163,725,150 log messages. The number of messages per day is varying due to the weekly working cycle and due to the failures of the event filtering. The three spikes (on 07.09-08.09, 15.09 and 25.11) are caused by "Denial of service event filtering triggered" (according to the ArcSight log messages). During this failure, the events were collected from 103 to 735 Domain Controllers instead of 14 on average for the rest of the data set.

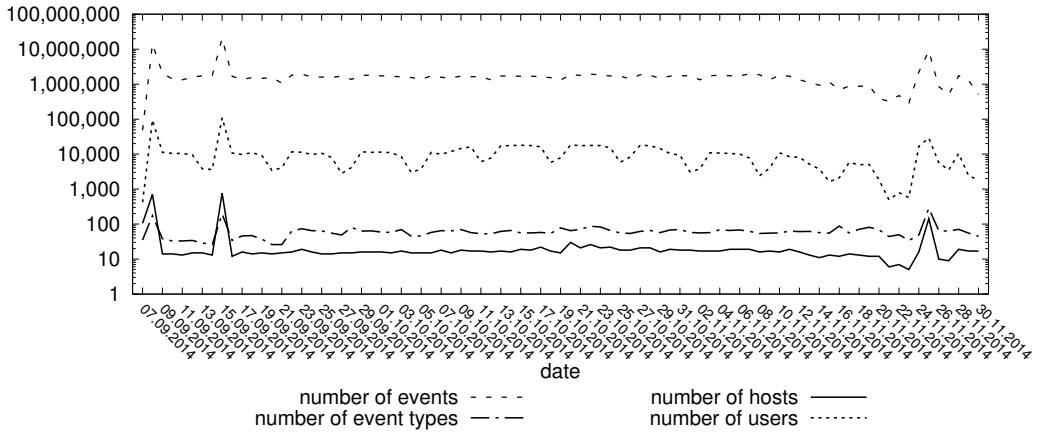


Figure 4.5: Overview of Windows Events data from the large enterprise, y-axis log-scaled

The set of the monitored Domain Controllers is also not consistent. Most of Domain Controllers were monitored during 1-3 days only, whereas only 16 Domain Controllers were monitored during a longer period (between 34 and 81 days, 64 days on average).

To apply Hybrid Outlier Detection, it is reasonable to filter out all log messages from Domain Controllers that were monitored less than 3 days (so that the outlier detection algorithm will be able to build the clusters based on the long-term patterns in the data). After such filtering, the data set contained 123,552,537 events from 200,699 users (71,537 of which are computer accounts).

One limitation of the data set is the nature of the monitored events. The collected Security Audit events include mainly the automated activity of the user accounts rather than real user actions. Taking a Windows Event ID 4624 ("An account was successfully logged on") as an example, all such events in the data set have logon type 3 or 5. These logon types reflect automated network activity and service start-up, whereas the logon type 2 (recorded when the user enters his/her password) was never reported to the ArcSight SIEM system. Further, only 5,826,900 of 31,481,180 Windows Events with Event ID 4624 are related to accounts of real users, while the rest are related to computer accounts that do not directly reflect the user activity.

Nevertheless, even though the most of the collected events represent automated activity in the network, such events still can reflect the activity of real users. All in all, this Windows Event data set provides a real example of log messages in the SIEM system of a large enterprise that should be processed and analysed.

Besides the Windows Events data set, the benefits of the Hybrid Outlier Detection were also proved on the public KDD Cup 1999 data set [146]. Even though

it contains redundant and duplicated records [76], it is still the most popular data set for evaluation of intrusion detection algorithms. Although the cleaned NSL-KDD version of this data set was offered by Tavallaee et al. [76], the original KDD Cup 1999 data could better reflect the real scenarios, since the real-world data collected by IDS or SIEM system also often contain redundant or duplicated events. However, due to the fact that the data set has attack ratio of 80%, there is a high probability that unsupervised outlier detection methods will classify attacks as normal since they represent the majority of the data. To avoid this issue, only first 400,000 records (with an attack ratio of 9.8%) of the KDD Cup 1999 data set will be used to evaluate Hybrid Outlier Detection.

The next sections describe the results of Hybrid Outlier Detection applied on the data and evaluate both accuracy and performance of the proposed algorithm in comparison with existing approaches.

4.3.3 Threats detected in the Windows Events data set

To analyse the Windows Events data with Hybrid Outlier Detection, the following columns from Object Log Format were selected:
event_id, subjectuser_username, targetuser_username, net_src_ipv4, net_src_host, net_src_port, producer_host, net_dst_ipv4, net_dst_host, net_dst_port, tag_action, tag_status, file_path, event_type_id, time.

The number of training samples and number of events per each sample were selected heuristically taking into account optimal sample size (from 10% to 20% of the data set size according to Zimek et al. [150, Section 3.2]), the algorithm's performance and available resources (will be described in the next section). To apply Hybrid Outlier Detection, a number of samples and events were set to $N = 80$ and $m = 100,000$.

According to the step 4 of the Hybrid Outlier Detection, an optimal value of k (the number of clusters per sample) was determined based on 2 training samples, as well as 2 of M subsets from original data (see Section 4.3.1 for details). Figure 4.6 shows the function of calculated similarity measure based on the value of k for both training and original data samples.

For both types of samples, the optimal k was determined as 96 using the technique described in Section 4.2.4. Taking $k = 96$, the average similarity (for all clusters) between all events in the cluster and the corresponding concept vector is 84% and 90% for the training samples and samples from original data correspondingly. This proves that the selected value of k allows the spherical k-means to produce clusters with a relatively small variance of events within the cluster.

The Hybrid Outlier Detection returned 12,759 clusters with 10,822,312 Windows Events marked as outliers. Obviously, such a number of outliers is very hard to process for the human operator. However, since all outliers are clustered and or-

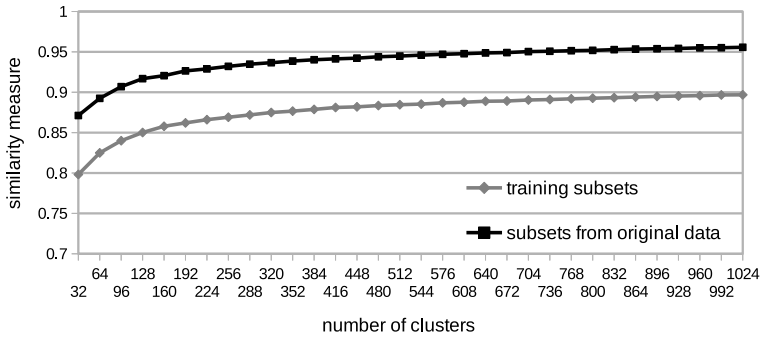


Figure 4.6: Average cluster similarity for different numbers of clusters (k).

dered by the sum of scaled decision values from different SVM models, it is possible to check the most highly ranked clusters/outliers only. Within top 100,000 events from the most suspicious clusters, approximately 98% are authentication failures. These are mainly Windows Events with Event IDs 539, 560, 4625 and 4776. Thus, the Hybrid Outlier Detection algorithm was able to identify authentication failures as anomalies and place them at the top of the output.

If instead of looking at top 100,000 outliers, one will check the 100 most suspicious clusters, these clusters contain 7439 events, 7376 of which are authentication failures. Thus, some clusters with authentication failures that can be caused by misconfiguration (as described in Section 4.3.2, the Windows Event data set reflects automated activities only) also contain a few events where the authentication was successful. Since the username, IP addresses and other parameters of failures and successes are same, it is hard to explain the reason for the authentication failures in such cases.

Besides that, the top 100 clusters from the output of Hybrid Outlier Detection contain two types of events that are different from the rest authentication failures:

- 26 authentication failures in clusters 1-9 (top to bottom in the ranking) related to the target user 'snapdrive'. All these authentication failures were created by one server only. Among 123 million of Windows Events, there were only 206 events of this type, and the Hybrid Outlier Detection placed 26 of them within the 10 most highly ranked clusters with outliers. These failures could indicate a failure of the SnapDrive storage management software or other misconfiguration. The detection of such issues proves the ability of Hybrid Outlier Detection to identify rare and unusual events in the data set.

- The Hybrid Outlier Detection identified 422 events with authentication failures for computer accounts accessing network shares (Event ID 5140). These

events were places in clusters 10-14 in the outlier ranking. In total, the Windows Events data set contains 330,363 network share access events and only 1483 of them (including 422 placed within 15 most suspicious clusters) are failures. It is possible to assume that this rare type of event also indicates a misconfiguration preventing specific servers from accessing the network share.

All in all, the Hybrid Outlier Detection was able to identify authentication failures in the data set and place them at the top of the output. The algorithm was also able to find out several rare suspicious events and place them within top 10-15 clusters with outliers. It proves the concept mentioned in Sections 1.3.2.3 and 4.3.1.1, which states that the operator of the SIEM system should be able to concentrate on the several clusters with most suspicious events ranked by the outlier score, instead of checking the endless list of all outliers returned by the algorithm.

The Hybrid Outlier Detection is just one algorithm in the REAMS that implements hybrid detection approach, as mentioned in Section 3.2.3. According to this approach, the Windows Events data set was also analysed with other methods. The full overview of issues detected with all analytical methods available in REAMS will be provided in the conclusion of this thesis. The next sections of this chapter concentrate on the Hybrid Outlier Detection and provide performance and effectiveness estimation, as well as comparison with existing methods.

4.3.4 Performance estimation

The estimation of the execution time was done on the IBM System x3850 X5 server with 6 TB RAM and 8 Intel Xeon E7-8870 CPUs providing 80 physical cores and 160 threads with enabled Hyper-Threading. Due to the fact that this server was always shared to execute multiple tasks[18], the Hybrid Outlier Detection was executed with 50% of available CPU threads. Each thread used up to 27 GB RAM, resulting in 2.1 TB RAM required to perform outlier detection in parallel with 80 threads.

The Hybrid Outlier Detection needs approximately 39 hours to process all 123 million events from the Windows Event data set. The training phase took 3 hours, whereas 36 hours were spent to finish the application phase. Taking into account the fact that these events cover 85 days with an approximate rate of 60,000 events per hour, the Hybrid Outlier Detection was able to process them at more than 50 times higher rate. Indeed, to apply the model on 123 million events, it took 36 hours, which implies the rate of 3.4 million events per hour. Besides that, the

[18]Even though the server was shared, there were always free resources, both RAM and CPU, during the data analysis.

server hardware has a capacity to further increase the processing speed since the Hybrid Outlier Detection used only 50% of available CPU resources and 30% of available RAM.

On the one hand, even though all available hardware resources would be used, the processing rate will not exceed 160 million events per day, which is not too high (but still enough to process the whole 3 months of Windows Event data within 1 day). On the other hand, the Hybrid Outlier Detection should be much faster than other existing generic outlier detection methods. In the next section, both the effectiveness and performance of Hybrid Outlier Detection will be compared with other outlier detection methods.

4.3.5 Effectiveness on Windows Events data set and comparison with other outlier detection algorithms

The initial effectiveness estimation of Hybrid Outlier Detection algorithm can be performed based on the results provided in Section 4.3.3. The 98% of the top 100,000 outliers are authentication failures. If compared with the number of authentication failures in the original data set (4,618,649 events or 3.7% of the full data set with 123 million events), it can be considered a good result and proves that Hybrid Outlier Detection is able to at least identify authentication failures.

For the proper effectiveness estimation, it could be possible to plot the Receiver Operating Characteristic (ROC)[19] based on the ground truth. However, the Windows Event data set does not contain the knowledge about the ground truth, i.e. there is no information whether events are malicious or not. Therefore, it is only possible to take the status field of Windows Events as ground truth. Thus, all events with "Failure" in the status field should be considered malicious to create a ROC curve based on this data set. However, this approach could lower the real effectiveness of the algorithm due to the fact that not only "Failure" events could be considered malicious in the real situations. The output of the Hybrid Outlier Detection also includes clusters with outliers that contain a number of authentication failures followed by authentication success. In such cases, the events of successful authentication preceded with failures should not be classified as false positive since they could be a part of successful password brute-force attack. However, in this section, such condition is still counted as false positive just for evaluation purposes. Figure 4.7 shows the ROC curve for the top one million ranked outliers from the Hybrid Outlier Detection output.

The number of "false positives" (Windows Events with 'success' status) within the top one million outliers is 64.7%, which is much higher than 2% for top 100,000. However, the absolute number of false positives is not so important for the outlier

[19]Standard evaluation method for unsupervised outlier detection algorithms [152].

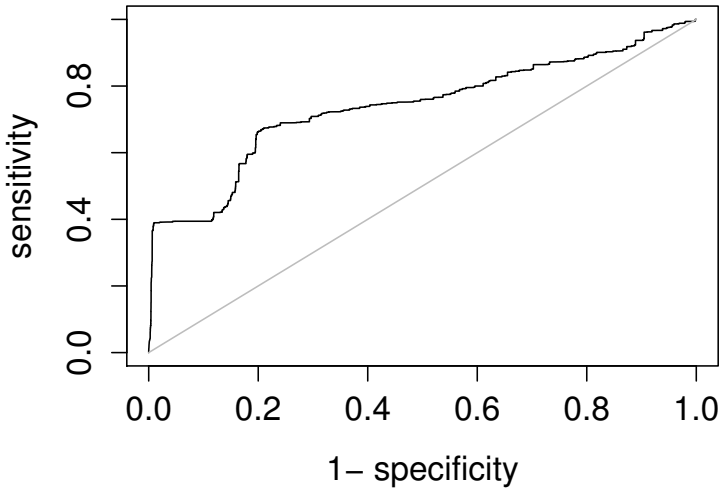


Figure 4.7: ROC curve for the top-ranked million anomalies returned from Hybrid Anomaly Detection algorithm

detection, if the algorithm produces ranked results and is able to place true positive alerts at the top of the output [152]. The ROC curve shows how good the Hybrid Outlier Detection is able to place "true positives" (events with "Failure" status) higher in its output, i.e. how much higher is the True Positive Rate (y-axis) than False Positive Rate for each outlier threshold value. The Area Under Curve (AUC) characteristic represents this rate difference with the single number, which is equal to 73.9%. Although this AUC value is not too high for unsupervised outlier detection in general, taking into account the specifics of the Windows Events data set (described in Section 4.3.2) and other benefits of the Hybrid Outlier Detection (such as performance, ranking of clusters with outliers and ability to work on any type of data), this result can be considered a good one.

To compare the effectiveness of Hybrid Outlier Detection with another outlier detection method on the Windows Event data set, another method should ideally also return ranked clusters with outliers. However, the literature search does not reveal any examples of similar approaches. Moreover, many existing unsupervised outlier detection methods cannot work on textual data (like [153] and [154]) or are bounded to the specific data set, due to custom pre-processing, mappings and other feature preparation techniques [155, 27, 156].

Therefore, it was first decided to compare Hybrid Outlier Detection with the Universal Outlier Detection from Section 4.2.5, in order to be able to compare the

ROC curves of two generic outlier detection methods. However, the initial tests of the Universal Outlier Detection proved that it is much slower than Hybrid Outlier Detection and requires more than 3 weeks for the analysis of Windows Events data set on the same hardware. To reduce the execution time, the Windows Events data set size was also reduced to the first 8 million events for Universal Outlier Detection.

In addition to Universal Outlier Detection, another approach [49], based on custom feature selection but designed specially for Windows Events, was selected for the efficiency comparison. The kNN-based outlier detection offered by Goldstein et al. works on so-called "data views" that are aggregation views for different types of Windows Events. An example of such data view would be "events of single users" that "are aggregated with a time unit of one day" [49]. Each data view is processed with the kNN outlier detection to identify anomalies (for example, the unexpectedly high number of authentication failures for a single user within one day). This algorithm was implemented by Amer and provided by Goldstein in the RapidMiner [157]. However, this implementation utilises the straightforward approach to find k nearest neighbours, which needs to calculate the distance between every two events. On the large data sets, this method becomes inefficient. The Windows Event data set contains 164,840 users, which makes this algorithm almost impossible to apply on these data. To solve this problem, the kNN-based outlier detection was completely re-implemented using cover tree [158] under this thesis. The re-implemented algorithm was able to apply outlier detection on 51 of 85 data views, while the computation for the rest of the data views was not finished even within 3 days.

The results of the comparison of Hybrid Outlier Detection with both Universal Outlier Detection and re-implemented kNN-based Outlier Detection are presented in the subsections below.

4.3.5.1 Comparison with Universal Outlier Detection

Similarly to Hybrid Outlier Detection, the Universal Outlier Detection also returns authentication failures as outliers after being applied on first 8 million events from the Windows Events data set. To compare the results of these two algorithms, the ROC curve was also calculated for Universal Outlier Detection. To keep the same ratio between the number of outliers and the full size of the analysed data, as for Hybrid Outlier Detection results, the ROC curve was the plot for top 64,750 outliers[20]. The ROC curve is shown in Figure 4.8.

[20]Top 64,750 outliers from data set with 8 million records for Universal Outlier Detection correspond to the same ratio as top 1 million outliers from data set with 123 million records for Hybrid Outlier Detection.

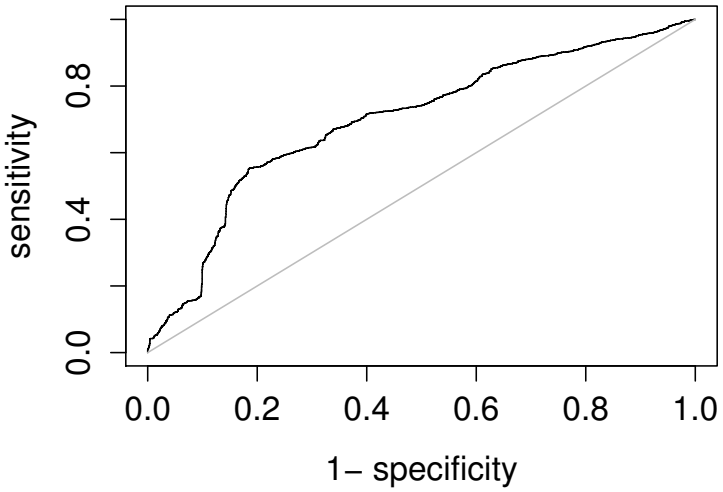


Figure 4.8: ROC curve for the top-ranked 64,750 anomalies returned from Universal Outlier Detection algorithm executed on 8 million events.

The AUC value for Universal Outlier Detection is 69.8%, which proves that this outlier detection method is also able to rank outliers higher in the output. However, this value is 4% less than for Hybrid Outlier Detection. Thus, the Hybrid Outlier Detection has both higher performance and better AUC value / higher effectiveness than Universal Outlier Detection.

4.3.5.2 Comparison with kNN-based outlier detection

In the case of kNN-based outlier detection, it is not possible to plot the ROC curve for comparison with Hybrid Outlier Detection. The reason for this is the absence of ground truth for Windows Events data set. As mentioned in Section 4.3.5, the events were considered malicious/benign based on their status field. All authentication failures were therefore marked as malicious. This approach cannot be applied to kNN-based outlier detection due to the fact that this algorithm works on the data views containing failures only[21]. Thus, there are no benign

[21]Original data set is pre-filtered with queries that select different types of failures and aggregate them into the data views. The kNN-based outlier detection is then executed on failures only, just to find the most abnormal values within various access failure events. Therefore, the direct comparison with Hybrid Outlier Detection (which works on all events and should be able to distinguish "malicious" authentication failure events from "benign" authentication success events) is not possible.

events within the data views, which makes ROC curve pointless.

To compare two algorithms without the possibility to plot a ROC curve, it was decided to check, how many users identified as suspicious by kNN-based outlier detection are also included into the top 100,000 outliers from the Hybrid Anomaly Detection. The results of kNN-based outlier detection from 51 data views are presented in Table 4.2 below.

Table 4.2: Issues detected with kNN-based outlier detection (Goldstein et al.)

Type of issue detected	Number of users/events
Account locked out	2 users with more than 100 account lockouts within 1 hour
Account locked out	1 computer user with 308 account lockouts on the same workstation
Special privileges assigned to new logon	7 users with 100 or more assignments per hour
Special privileges assigned to new logon	6 users with 600 to 6000 assignments per day
Special privileges assigned to new logon	5 users with several thousand assignments on one workstation
Kerberos pre-authentication failed	6 users with more than 70 failures per hour
Kerberos pre-authentication failed	4 users with more than 100 failures per day
Kerberos pre-authentication failed	10 users with more than 1000 failures on the same workstation
Unknown username or bad password	3 users with 50-100 bad username/password events per hour
Unknown username or bad password	2 computer accounts and 2 real users with 800 to 36,000 bad username/password events on one workstation
Account currently disabled	1 computer account with 325 logon failures within 1 hour
Account currently disabled	7 computer accounts with more than 100 logon failures on one workstation

In total, the kNN-based outlier detection returned 56 users with an abnormally high number of authentication failure events.

The top 100,000 outliers from Hybrid Outlier Detection include 22 of these 56 users, which can be considered as a good result. On the one hand, the kNN-

based outlier detection is focused on the authentication failures only and is able to detect more suspicious failures (related to the pre-defined data views) than Hybrid Outlier Detection. On the other hand, the Hybrid Outlier Detection is a generic algorithm, which is able to identify various issues, not limited to the authentication failures. Besides that, some events were identified by Hybrid Outlier Detection only, for example, SnapDrive authentication failures or failed network share accesses. These events cannot be detected with kNN-based outlier detection by definition since there were no data views for these types of events.

All in all, kNN-based outlier detection and Hybrid Outlier detection complement each other, while each approach has its own advantages. The kNN-based outlier detection can be used for detailed analysis of authentication failures, whereas the Hybrid Outlier Detection works on the original data set without any preprocessing and returns different types of outliers, which cannot always be covered by data views.

Thus, the comparison of Hybrid Outlier detection with two existing approaches on Windows Events data set clearly shows its benefits: support for any types of data, high performance and effectiveness, clustered output. To make the comparison with other approaches also possible, the Hybrid Outlier Detection was tested on the public KDD Cup 1999 data. The effectiveness estimation on this public data set is provided in the section below.

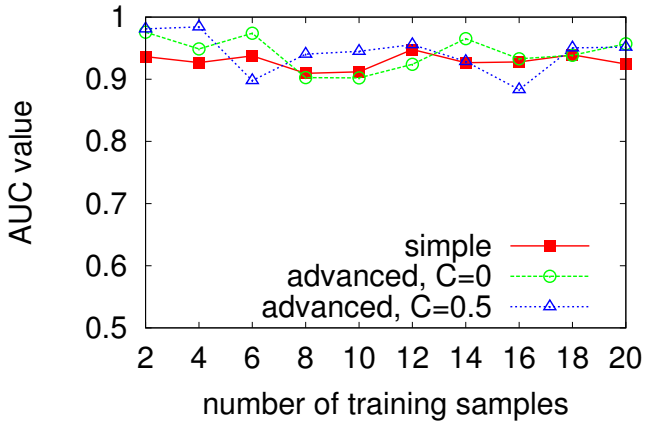
4.3.6 Effectiveness of Hybrid Outlier Detection on public KDD Cup 1999 data set

The KDD Cup 1999 data is an example of the data set containing both textual and continuous numerical features. The Hybrid Outlier Detection is able to process all of them thanks to the techniques described in Sections 4.2.1-4.2.4 and 4.3.1.1. The results of the Hybrid Outlier Detection on this data set were evaluated with AUC values, which are shown in Figure 4.9.

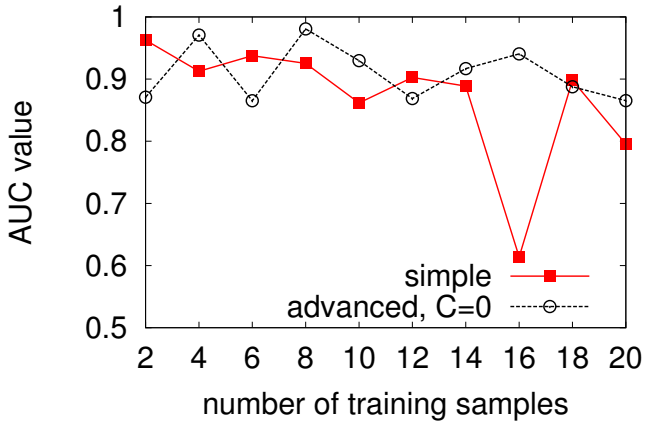
Figure 4.9 shows the AUC values calculated for multiple ROC curves, each of which was created using a different combination of parameters for Hybrid Outlier Detection: number of training samples (on the x-axis), number of clusters per sample (k) and type of discretisation[22] (marked with line colour).

The AUC values were also calculated for a different number of training samples, up to the half of the data set (20 samples 10,000 records each). From Figure 4.9, it is possible to conclude that the AUC value has a relatively small variance for different parameter combinations. Both relatively small fluctuations and the drop-down of AUC value in Figure 4.9(b) can be explained with the fact that the k-means algorithm (with random initialisation) is applied multiple times to the data

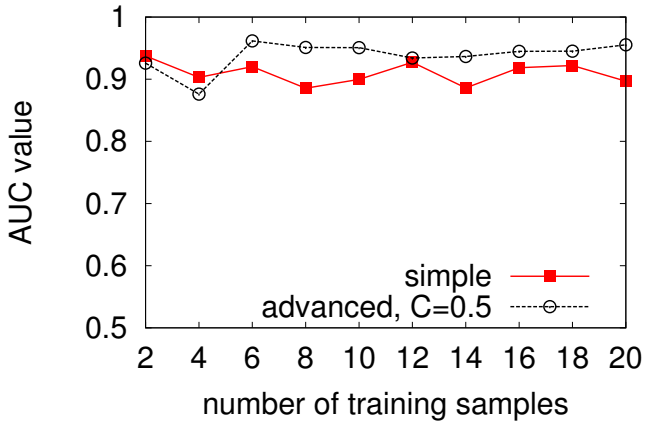
[22]See Section 4.2.1.1 for details.



(a) AUC for k=22



(b) AUC for k=10



(c) AUC for k=43

Figure 4.9: AUC values for the different number of training samples, discretisation methods and clusters per sample (k)

during execution of Hybrid Outlier Detection (first, during feature discretisation, then during clustering of training samples and later for clustering data subsets during model application phase).

For each number of training samples, the measurements of AUC value was performed with different feature discretisation methods (simple and advanced). On the samples created with simple discretisation, an optimal number of clusters was $k = 22$ (Figure 4.9(a)), whereas when the advanced discretisation was used, the optimal number of clusters was $k = 10$ for $C = 0$ ((Figure 4.9(b))) and $k = 43$ for $C = 0.5$ (Figure 4.9(c)). To compare discretisation methods with each other, the Hybrid Outlier Detection was also applied on the data using advanced discretisation and $k = 22$, as well as using $k = 10$ and $k = 43$ and simple discretisation.

Figures 4.9(a)-4.9(c) prove that advanced discretisation allows reaching higher AUC values on average. For example, with $k = 22$ and advanced discretisation, the Hybrid Outlier Detection reaches average AUC value of 94.2%, whereas with simple discretisation the average AUC value is 92.9%. For $k = 10$, the average AUC value equals 91% and 87% with advanced ($C = 0$) and simple discretisation correspondingly. For $k = 43$, the average AUC value is 93.8% for advanced discretisation ($C = 0.5$) and only 91% when using simple discretisation.

The results also prove that the Hybrid Outlier Detection is able to identify outliers even when trained on the relatively small subset of the data (2 or 4 training samples with 10,000 records each, which is 5-10% of the data set size). Indeed, with 4 training samples, $k = 22$ and advanced discretisation with $C = 0.5$, the AUC value is 98.4%. This value is comparable to other existing unsupervised outlier detection methods evaluated on the same data set. For example, Goldstein et al. [152] evaluated a number of such methods with AUC values between 73.7 and 99.9% on KDD Cup 1999 data set.

4.4 Chapter summary

In this chapter, the existing classical outlier detection algorithm was reviewed to propose the various improvements for modern unsupervised outlier detection methods. These improvements result in two new outlier detection algorithms, namely Universal Outlier Detection (defined in Section 4.2.5) and Hybrid Outlier Detection (described in Section 4.3.1). Both algorithms were tested on different data sets, including real Windows Event data set with 123 million events from the multinational enterprise. The Hybrid Outlier Detection was also evaluated on KDD Cup 1999 data and compared with other existing approaches for unsupervised outlier detection.

The results of the evaluation and comparison show the clear benefits of the proposed algorithms. In particular, the Hybrid Outlier Detection does not require

feature preprocessing, produces ranked clusters of outliers and works on any type of the data. The performed tests also show that the techniques offered in this thesis allow the proposed algorithms to reach higher performance on the same data. For example, thanks to the training on randomly selected data subsets, the Hybrid Outlier Detection is approximately 7 times faster than Universal Outlier Detection (which is, in turn, based on the classical k-means outlier detection, similar to the one implemented in SAP HANA PAL). Thanks to the special techniques, such as advanced feature discretisation and conversion of the features into the vector space model, the Hybrid Outlier Detection also has high effectiveness, which was proved on different data sets. Trained on 5-10% of the data set it shows high efficacy, e.g. AUC of 98.4% on KDD Cup 1999 data.

Taking into account all evaluation results provided in this chapter, the Hybrid Outlier Detection can be claimed as a most relevant generic unsupervised outlier detection approach for the analysis of Big Security Data within a SIEM system.

However, besides the generic outlier detection methods, SIEM systems sometimes need to detect special types of outliers, for example, the activity of intruders that do not produce any authentication failures or other errors traces in the log messages (which is therefore hard to detect with generic methods). The next chapter focuses on such cases and proposes the anomaly detection method specially for the analysis of user behaviour.

Chapter 5

Outlier detection for malicious user behaviour without access violation[23]

Modern SIEM and IDS systems usually concentrate on various hacking scenarios, including access violation cases, policy violations, protocol anomalies and suspicious network traffic [160, 29, 30, 161, 162, 163, 26]. The existing solutions for the user behaviour analysis are also rather focused on specific use cases, such as local user activity or VPN access [164, 156, 165]. The malicious user behaviour that does not trigger these scenarios often stays out of focus for SIEM and IDS. For example, in case of espionage, the insider (or intruder who uses the stolen identity of the employee, including the password) can try to collect the company data as silently as possible, accessing only the resources he/she is authorised to access. In this case, his/her actions can stay undetected by SIEM and IDS. The Data Leak Protection systems can also have difficulties in detection of such espionage cases since they are generally focused on the data itself and usually do not control data flows within the enterprise [166, 167]. Then, to stay undetected by a Data Leak Protection system, an attacker can try to avoid large data transfers to the untrusted locations.

Thus, detecting such cases of malicious user behaviour is a rather complex task, especially in the large enterprises with tens of thousands of employees. The existing anomaly detection methods based on user behaviour models have high precision (more than 90%) but are rather sophisticated and too heavy for large data volumes. The examples of these models are Markov chains and Hidden Markov Models [96, 168], Principal Component Analysis [95, 169], Support Vector Machines and Neural Networks [27, 75, 170], and finally hybrid techniques [90, 171]. All in all,

[23]The results described in this chapter have been published as [159, 123].

the existing solutions are either too specific and analyse a particular scenario, data source or system, or do not consider the described espionage scenario at all.

In this chapter, the case of malicious user behaviour without access violation is analysed in details to develop the high-performance user behaviour model, suitable for the analysis of Big Security Data. To develop such a model, the described scenario was implemented using the special simulation tool. Based on this simulated scenario, the high-performance and lightweight user behaviour model was proposed. Next, this model was tested on both simulated and real data. The detailed description of the scenario, user behaviour model and the data are provided in the sections below.

5.1 Scenario of malicious user behaviour without access violation

In order to implement the described scenario and generate the simulated data set, the scenario itself should be defined with more details. These details should cover the "normal" case, when nothing suspicious happens in the company's network, as well as "abnormal" case, when, for example, stolen credentials of an employee are utilised to collect all information available in the company's network, but without access violations. Both the benign and malicious user behaviour cases are presented with necessary details in Figures 5.1 and 5.2.

Both Figures 5.1 and 5.2 describe the network with two user groups: users (Alice, Bob and Carol) and administrators (Administrator). To access any of the servers, the user first needs to enter credentials on his/her PC. All users have rights to logon on any server or computer in the network, however, only Server_1 is accessed by users in the normal case. Different to this, the Server_2 is normally never used by anybody except administrators.

In the "abnormal" case (shown in Figure 5.2), the user Bob accesses both Server_2 and Admin PC, which were never used before neither by him nor by his user group. Although he is authorised (according to the domain policy) to use these resources, such case should be classified as malicious. On the other hand, when Alice accesses Server_1, her behaviour should be classified as benign due to the fact that other users from her group were using the same server before.

Both cases of "normal" and "abnormal" user behaviour were implemented using a special simulation tool in the virtual testbed with Windows domain. The testbed and the simulation tool are described in the section below.

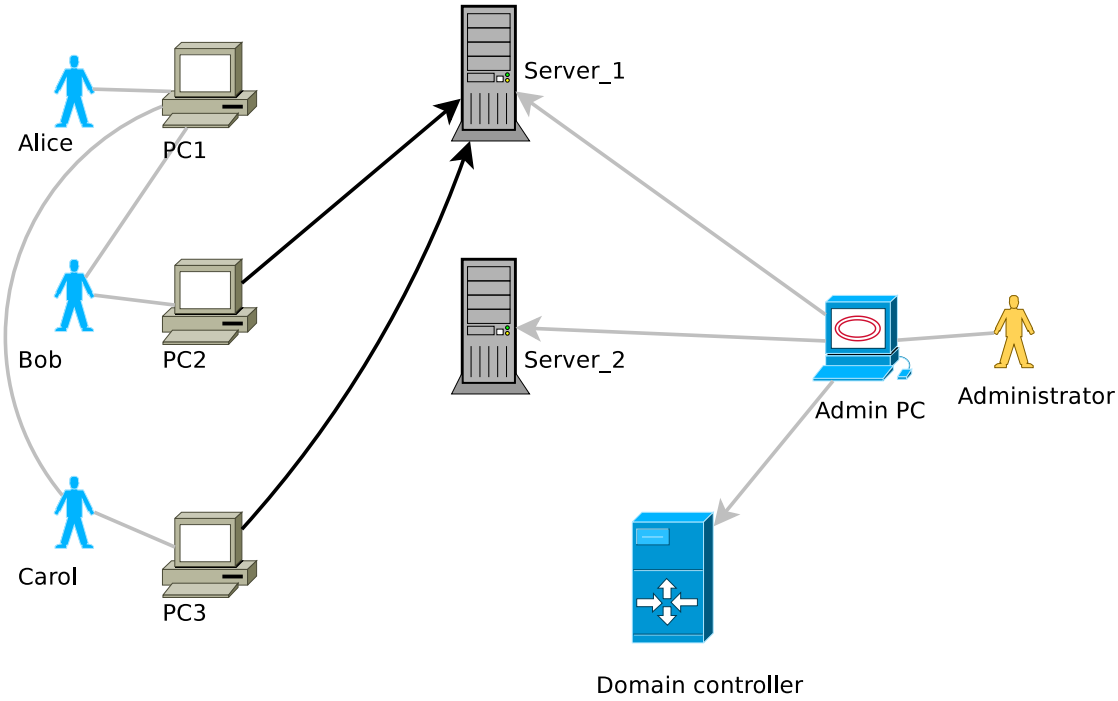


Figure 5.1: Example of normal user behaviour in the network

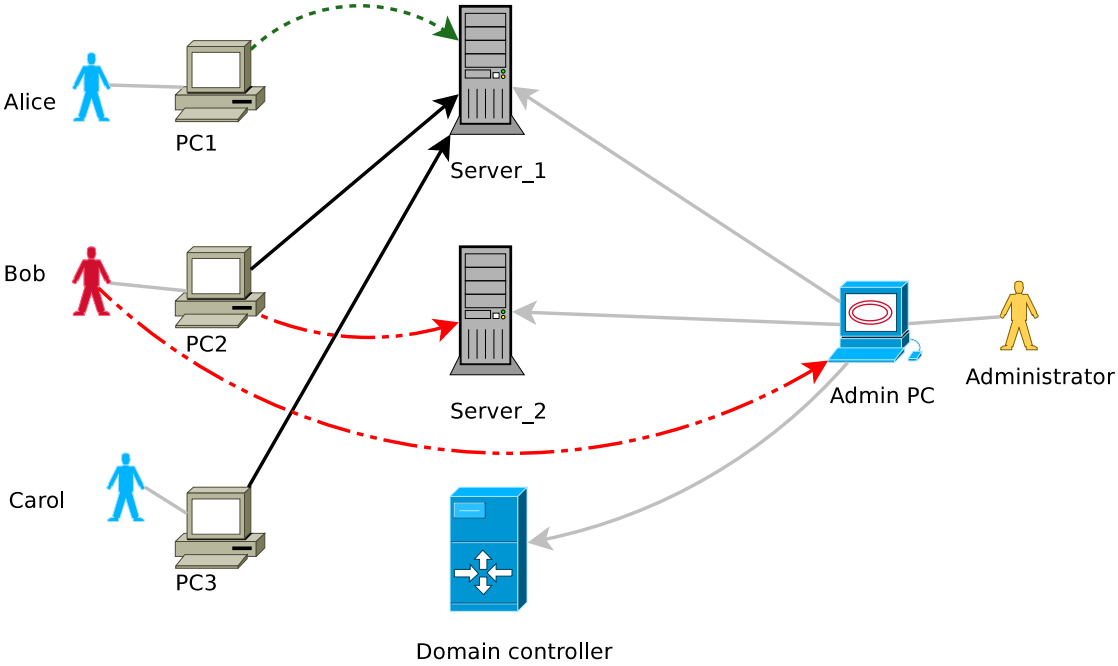


Figure 5.2: Example of malicious user behaviour in the network

5.2 Simulation of user behaviour for evaluation purposes

To simulate the described user behaviour scenario, the testbed with virtual machines running Microsoft Windows 7 (for the user and administrator PCs), Windows Server 2003 (for Server_1 and Server_2) and Windows Server 2012 (for Domain Controller) was created on the VMware ESXi hypervisor [137]. The Remote Desktop [135] was enabled on all virtual machines, whereas the Domain Controller virtual machine was also running DHCP, DNS and Active Directory services. The simulation of user activities was implemented utilising VNC [172] remote access functionality supported by VMware ESXi. The special simulation tool was implemented in Python (based on Python Imaging Library [173]) to execute user actions through VNC using screen recognition functions [174]. Thus, the simulation software connects in parallel to multiple virtual machines via VNC and recognises the screen state. Based on the screen state and scenario being executed, the simulation tool is able to lock or unlock the Windows session (by sending key combinations or typing the password into the corresponding input field), as well as to run a PowerShell [175] script, which is placed on the Desktop of each virtual machine in advance. This script, in turn, connects to another virtual machine over Remote Desktop according to the pre-defined scenario.

The collection of user behaviour events is organised in the same way as described in Section 3.4.1. The enabled Audit policy allows the Domain Controller to receive all Windows Events related to user actions, such as logon and logoff events.

The simulation tool is controlled with scenario files containing the actions to be executed in order. The overview of the scenario files format is presented in Figure 5.3. All scenarios with the schema shown in Figure 5.3 are stored in the .csv[24] files. The main file is the 'Scenario', which describes how many times and for how long each user should login on different computers. All entries from this file are executed in parallel, but can lock each other (i.e., only one user can access one computer at the same time). Besides this, each entry can be executed several times, if defined in the 'TIMES' field.

The computers, where users log in, are defined in the 'Computers' file, which contains all data needed to establish the VNC connection to the virtual machine: hostname of the VMware ESXi hypervisor, password and port (different for each virtual machine).

The simulation tool also supports execution of "inner scenarios" that are used if the user should perform a remote connection from the computer where he/she is logged in to another computer or server. The descriptions of inner scenarios are

[24]Comma-separated values.

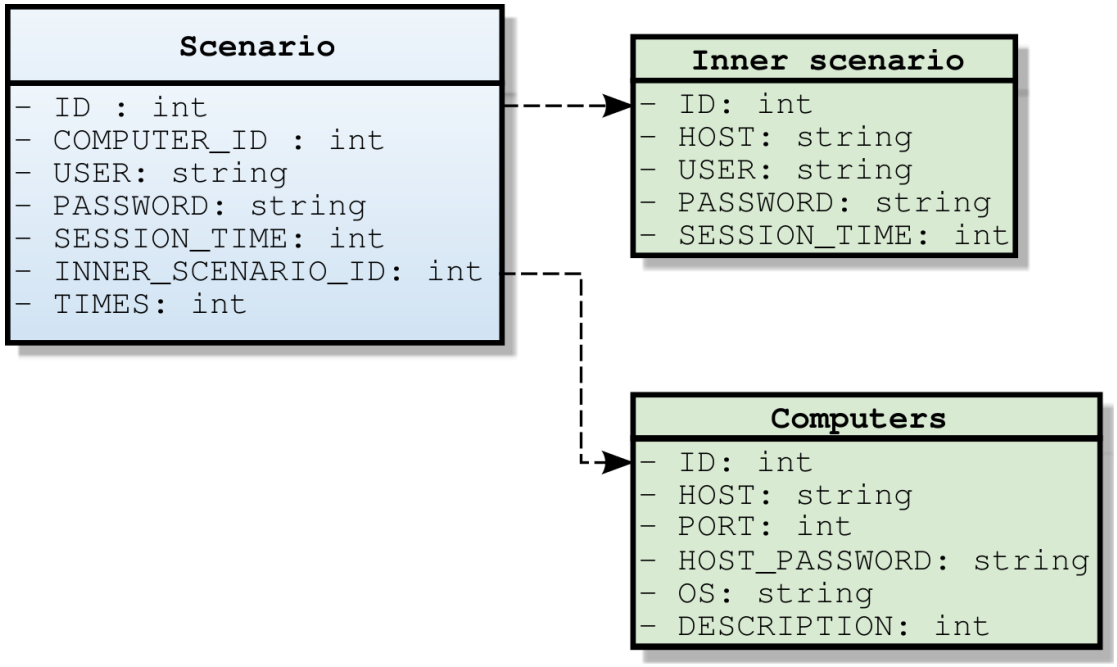


Figure 5.3: Scenario schema used in simulation software

stored in the 'Inner scenario' file and contain parameters for the PowerShell script used to establish the Remote Desktop connection to another VM, as well as the time of the Remote Desktop session. During the execution of the inner scenario, a user still holds the lock for his/her computer.

The use cases of benign and malicious behaviour described in Section 5.1 were simulated according to the scenario files provided in Tables 5.1-5.3 below.

Table 5.1: Hosts used for simulation of user behaviour

ID	HOST	PORT	HOST PASSWORD	OS	DESCRIPTION
1	192.168.42.20	5908	<PC1pass>	win7	PC1
2	192.168.42.20	5905	<PC2pass>	win7	PC2
3	192.168.42.20	5906	<PC3pass>	win7	PC3
4	192.168.42.20	5901	<PC4pass>	win7	Admin PC

Table 5.2: Simulation scenario containing malicious user behaviour

ID	COMP. ID	USER	PASSWORD	SESSION TIME	INNER SCE-NARIO ID	TIMES
1	4	Administrator	<adminpass>	300	1	15
2	4	Administrator	<adminpass>	200	4	10
3	4	Administrator	<adminpass>	300	5	20
4	4	Administrator	<adminpass>	215	0	30
5	1	Alice	<pass1>	205	0	40
6	1	Alice	<pass1>	205	6	40
7	2	Bob	<pass2>	195	2	66
8	3	Carol	<pass3>	300	3	45
9	2	Bob	<pass2>	220	0	30
10	3	Carol	<pass3>	200	0	44
11	1	Alice	<pass1>	205	6	40
12	4	Administrator	<adminpass>	300	0	11
13	2	Bob	<pass2>	245	7	31
14	1	Alice	<pass1>	250	0	22
15	4	Bob	<pass2>	200	0	25

Table 5.3: Inner simulation scenario containing malicious user behaviour

ID	HOST	USER	PASSWORD	SESSION TIME
1	10.10.21.1	Administrator	<adminpass>	235
2	10.10.21.110	Bob	<pass2>	240
3	10.10.21.110	Carol	<pass3>	250
4	10.10.21.110	Administrator	<adminpass>	250
5	10.10.21.109	Administrator	<adminpass>	230
6	10.10.21.110	Alice	<pass1>	229
7	10.10.21.109	Bob	<pass2>	230

Table 5.4: Scenario for simulation of normal user behaviour

ID	COMP. ID	USER	PASSWORD	SESSION TIME	INNER SCE-NARIO ID	TIMES
1	4	Administrator	<adminpass>	300	1	15
2	4	Administrator	<adminpass>	200	4	10
3	4	Administrator	<adminpass>	300	5	20
4	4	Administrator	<adminpass>	215	0	30
5	1	Alice	<pass1>	205	0	40
6	2	Bob	<pass2>	195	2	66
7	1	Bob	<pass2>	199	0	6
8	3	Carol	<pass3>	300	3	45
9	1	Carol	<pass3>	280	0	25
10	2	Bob	<pass2>	220	0	30
11	3	Carol	<pass3>	200	0	44
12	4	Administrator	<adminpass>	300	0	11
13	1	Alice	<pass1>	250	0	22

Table 5.5: Inner scenario for simulation of normal user behaviour

ID	HOST	USER	PASSWORD	SESSION TIME
1	10.10.21.1(Domain Controller)	Administrator	<adminpass>	235
2	10.10.21.110 (Server_1)	Bob	<pass2>	240
3	10.10.21.110 (Server_1)	Carol	<pass3>	250
4	10.10.21.110 (Server_1)	Administrator	<adminpass>	250
5	10.10.21.109 (Server_2)	Administrator	<adminpass>	230

The scenarios of benign and malicious user behaviour described in Tables 5.1-5.3 were executed with the simulation tool. During the execution time, 38,568 Windows Events were collected by the Domain Controller. However, to proceed with the analysis of user behaviour, these events need to be filtered first, as described in the next section.

5.2.1 Filtering user behaviour data

For analysis of user behaviour, the special filtering was applied to focus on the logon-related events only. Initially, only the following Windows Events related to user behaviour were selected:

- **4768** "A Kerberos authentication ticket (TGT) was requested"
- **4771** "Kerberos pre-authentication failed"
- **4776** "The domain controller attempted to validate the credentials for an account"
- **4624** "An account was successfully logged on"
- **4625** "An account failed to log on"

The selected events capture all events related to user login and logout for the whole Windows domain. However, these events still need to be cleaned up due to specifics of Windows logging process. The clean-up steps could be described as follows:

- Filter out all events with target user name 'DWM-2'. Events logged for DWM-2 user just reflect Desktop Window Manager activity. Even if this event is connected to the user logon process, there should be another logged event (4624) containing complete information about the real user.
- Filter out all events with ID 4624 and Security ID 'NULL'. Such events are not related to the user logon; rather they represent some local system activity [176].
- Filter out all events with logon type 5. Logon type 5 means start of some service and is not related to user logon.
- Filter out all events with logon type 3. Logon type 3 reflects Windows network activity and is not directly related to user logon.
- Filter out all events with Impersonation level 'impersonation' and without workstation data. It is not necessary to process such events since there is always a duplicated event containing workstation data.
- Filter out all other events where the subject user name is 'NULL' and target user name is 'NULL' or does not contain one of the real domain users. Such events do not need to be processed as well since they are caused by the normal system activity and are not related to any real domain users.

The data set containing these filtered events was used to develop a novel high-speed outlier detection approach for the analysis of user behaviour, which is described in the section below.

5.3 Outlier detection for user behaviour data

Since the data set mainly contains logon events, the user behaviour model should be based on such events. To create this model, the data are divided into equal time intervals. Next, the number of logon events is calculated for each {user,workstation} pair (tuple) per time interval. The optimal time interval should be selected heuristically, since each user behaviour data set can have different properties, such as the number of users in the company, time range, user activity (number of logon events per day) and so on. For small companies with 10-20 users that access 1 or 2 servers a couple of times per day, the optimal time interval could be one or more days. For large enterprises with thousands of users and a high number of servers, the optimal time interval can be set to several minutes, depending on user activity. Based on the simulated data set, the time interval of 15 minutes was chosen as optimal.

The resulting statistics for the number of logon events for each {user, workstation} tuple per time interval is used to create the user behaviour model. Since the data describe the discrete number of logon events per time interval, it is possible to suggest that it could be modelled with the Poisson's distribution, which is widely used for modelling user arrival events and network traffic [177, 178]. The Poisson's model is also quite simple (see Formula 5.1), which reduces the computational complexity of the outlier detection and allows the high-speed event processing.

$$P\left(X = x\right) = \frac{e^{-\lambda}\lambda^{x}}{x!} \tag{5.1}$$

However, for the simulated data set, the variance of the number of logon events per time interval is higher than the mean for the most of {user,workstation} tuples, which does not allow direct application of the Poisson's model. In the real data[25], such cases of over-dispersion can be mixed with series of logon events that follow the Poisson distribution. To handle this problem, the negative binomial model (which is the generic case of the Poisson model and covers the over-dispersion cases, when the variance is higher than mean) can be applied, according to the Formula 5.2.

[25]The real data set will be described in details later in this chapter.

$$P(X = x) = \frac{\Gamma\left(x + \frac{\mu^2}{\sigma^2 - \mu}\right)}{x! * \Gamma\left(\frac{\mu^2}{\sigma^2 - \mu}\right)} * \left(\frac{\mu}{\sigma^2}\right)^{\left(\frac{\mu^2}{\sigma^2 - \mu}\right)} * \left(\frac{\sigma^2 - \mu}{\sigma^2}\right)^x \tag{5.2}$$

Although the negative binomial distribution also covers the cases when mean equals to variance, it still makes sense to model such event series with Poisson's formula to process the corresponding event series at a higher speed.

To apply the outlier detection, the data are divided into training and testing subsets (normal and abnormal scenarios). Both data sets are divided into time intervals to calculate the number of events for each {user,workstation} tuple. Next, the mean and variance are learned from the training data set and then applied to the testing data set to calculate the probability for each number of logon events there. This approach is described in the Algorithm 3 in details.

Algorithm 3 Outlier_detection_users($threshold_{user}$)

1: **for all** {user,workstation} from training_data **do**
2: $\mu_{user,workstation}$ = average number of logon events per time_interval
3: ${\sigma^2}_{user,workstation} = E[(X - \mu)^2]$
4: **end for**
5: **for all** time_interval in testing_data **do**
6: **for all** {user,workstation} from testing_data **do**
7: **if** ${\sigma^2}_{user,workstation} > \mu_{user,workstation}$ **then**
8: Probability({user,workstation}) =
= PoissonsProbability({user,workstation},$\mu_{user,workstation}$)
9: **else**
10: Probability({user,workstation}) =
= NegativeBinomialProbability(
{user,workstation},
$\mu_{user,workstation}$,
${\sigma^2}_{user,workstation}$
)
11: **end if**
12: **if** Probability({user,workstation}) < $threshold_{user}$ **then**
13: **mark** {user,workstation,time_interval} as *anomaly*
14: **end if**
15: **end for**
16: **end for**

Thus, in the lines 1-4 of the Algorithm 3, the mean and variance are identified for all {user,workstation} tuples. Next, all time intervals for the testing data are

processed in lines 5-16. Each {user,workstation} tuple from each time interval is evaluated in lines 6-15. First, in line 7, the type of the learned logon event distribution is checked by comparing mean with variance. If mean is equal or slightly less than variance[26], the probability of a number of logon events within the time interval is calculated according to Poisson distribution in line 8, otherwise, it is calculated according to negative binomial distribution in line 10. If the probability is less than the predefined threshold, the tuple {user,workstation,time_interval} is marked as an anomaly in line 13.

This approach detects an anomalous number of events related to user logon. However, all {user,workstation} pairs are evaluated individually, which will result in the misclassification of benign cases, when a user accesses the resource that was regularly used not by him/her, but by his/her user group (like described in Section 5.1). To classify such user behaviour cases correctly, the two-step probability check is introduced in the section below.

5.3.1 Two-level probability check

To model both user and group behaviour, the Algorithm 3 can be applied twice to the data. First, instead of modelling events for {user,workstation} tuples, {group,workstation} tuples are analysed to identify anomalous (or *suspicious*) user groups. The suspicious groups are the {group,workstation,time_interval} tuples with the probability less than the pre-defined threshold, which means that the number of events from the particular group to the particular workstation is abnormal within the particular time interval (according to the Poisson or negative binomial distribution). The Algorithm 3 applied to user groups is presented below as Algorithm 4.

Second, the same algorithm is applied to {user,workstation} tuples. However, the probability of a number of events is calculated only for users from suspicious groups. Thus, it becomes possible to identify the users which caused the abnormal number of events for each suspicious {group,workstation,time_interval} tuple. This modified version of Algorithm 3 is described as Algorithm 5 below.

Different to the Algorithm 3, Algorithm 5 calculates the probability of a number of events only for those {user,workstation} tuples, where the user belongs to the suspicious user group (see line 6).

[26]The data set does not contain {user,workstation} tuples where the mean value is less than 90% of variance value.

Algorithm 4 Outlier_detection_groups($threshold_{group}$)

1: **for all** {group,workstation} from training_data **do**
2: $\mu_{group,workstation}$ = average number of logon events per time_interval
3: ${\sigma^2}_{group,workstation} = E[(X - \mu)^2]$
4: **end for**
5: **for all** time_interval in testing_data **do**
6: **for all** {group,workstation} from testing_data **do**
7: **if** ${\sigma^2}_{group,workstation} > \mu_{group,workstation}$ **then**
8: Probability({group,workstation}) =
= PoissonsProbability({group,workstation},$\mu_{group,workstation}$)
9: **else**
10: Probability({group,workstation}) =
= NegativeBinomialProbability(
{group,workstation},
$\mu_{group,workstation}$,
${\sigma^2}_{group,workstation}$
)
11: **end if**
12: **if** Probability({group,workstation}) < $threshold_{group}$ **then**
13: **mark** {group,workstation,time_interval} as *suspicious*
14: **end if**
15: **end for**
16: **end for**

Algorithm 5 Outlier_detection_users($threshold_{user}$,$suspicious\_groups$)

1: **for all** {user,workstation} from training_data **do**
2: $\mu_{user,workstation}$ = average number of logon events per time_interval
3: ${\sigma^2}_{user,workstation} = E[(X - \mu)^2]$
4: **end for**
5: **for all** time_interval in testing_data **do**
6: **for all** {user,workstation} from testing_data where {user_group,workstation,time_interval} is *suspicious* **do**
7: **if** ${\sigma^2}_{user,workstation} > \mu_{user,workstation}$ **then**
8: Probability({user,workstation}) =
= PoissonsProbability({user,workstation},$\mu_{user,workstation}$)
9: **else**
10: Probability({user,workstation}) =
= NegativeBinomialProbability(
{user,workstation},
$\mu_{user,workstation}$,
${\sigma^2}_{user,workstation}$
)
11: **end if**
12: **if** Probability({user,workstation}) $< threshold_{user}$ **then**
13: **mark** {user,workstation,time_interval} as *anomaly*
14: **end if**
15: **end for**
16: **end for**

Thus, the offered approach is able to identify an abnormal number of events for {user,workstation} tuples. Besides that, the algorithm also takes into account the group behaviour and uses it to correctly classify the cases when the user connects to the workstation, which was previously used only by his/her group, but not by him/her directly.

This approach for detection of user anomalies is based on the predefined $threshold_{user}$ and $threshold_{group}$ that are relatively hard to determine without the knowledge about each particular data set. The next section describes the automatic detection of the optimal values for both thresholds, which simplifies the application of the algorithm to different data sets.

5.3.2 Optimal threshold detection

To detect the threshold automatically, the same method as mentioned in Section 4.2.4 can be used. To apply this method, the number of suspicious groups should be calculated for different threshold values.

To perform this calculation, the maximal probability of a number of events for each {group,workstation} tuple within the time interval is identified based on the training data. Depending on the type of the distribution for each particular tuple, either Poisson or negative binomial distribution is used to calculate this probability value. The highest probability value is used as upper bound for the possible threshold value. Then, N threshold values between 0 and $highest\_probability/2$ are used in the Algorithm 4 to find out the number of detected suspicious groups for each threshold value. Finally, the Algorithm 1 from Section 4.2.4 is applied to identify the optimal threshold value.

The optimal values for $threshold_{user}$ can be identified in a similar way. However, the Algorithm 5 requires the list of suspicious user groups, and only calculates the probabilities for the users from those groups. To include all users into the optimal threshold calculation, the Algorithm 3 should be used instead.

Based on these optimal threshold values, all {user,workstation} tuples with the probability of the number of logon-related events below the threshold will be marked as anomalies. After that, the calculated probability will be used to rank anomalous events, as described in the next section.

5.3.3 Ranking of anomalous user behaviour cases

The output of the proposed outlier detection will contain all anomalous {user,workstation} tuples ranked by the probability of the number of events within the time interval in ascending order. Thus, the events with the lowest probability will be placed on the top of the algorithm's output to attract the attention of the

operator of the security system. However, it is important to distinguish between two types of anomalies by their probability score.

The first type are events with the probability equal to zero. The events with zero probability arise due to the fact that the proposed outlier detection requires the training data (which can be simply the part of the data set being analysed[27] or simulated data following the "normal" user behaviour scenario). The probability equal to zero means that the anomalous {user,workstation} tuple never appeared in the training data. Depending on the analysed data set and the training data, such events could either be the false positive alerts (new benign connections of users to workstations) or reflect an attack or insider activity (new connection between user and workstation that was not expected by the security policy and therefore was not modelled or included into the training data).

The second type of the anomalies are events with a probability higher than 0 (but still less than the threshold). The low probability value means that the {user,workstation} tuple was included into the training data set, but the data contains an abnormal number of logon-related events, which was very different from the training data.

To make the difference between these two types of the anomalies clear for the operator of the system, the algorithm's output contains two lists with anomalies: one with events that have zero probability, and another one with events that have a probability higher than zero.

Thus, the proposed outlier detection for user behaviour is able to automatically determine optimal thresholds, take into account both group and user behaviour and distinguish between two distribution types for each {group,workstation} or {user,workstation} tuple. The ranked output helps to highlight the most suspicious events to the security expert. The effectiveness and performance of the proposed method are evaluated on both simulated and real data in the sections below.

5.4 Threats detected in the simulated data

To check the detection rate of the proposed set of algorithms, the simulated data set (please see Sections 5.1 and 5.2 for details) is reviewed. Figures 5.4 and 5.5 show the distribution of filtered (according to the Section 5.2.1) Windows Events related to user behaviour for training and testing subsets (normal and abnormal scenarios) correspondingly.

The distribution of user behaviour events in both Figures is shown as bubbles, where the size of the bubble represents the number of events for {user,workstation}

[27] In this case, the offered outlier detection still will be unsupervised according to the classification from Section 2.2.5.

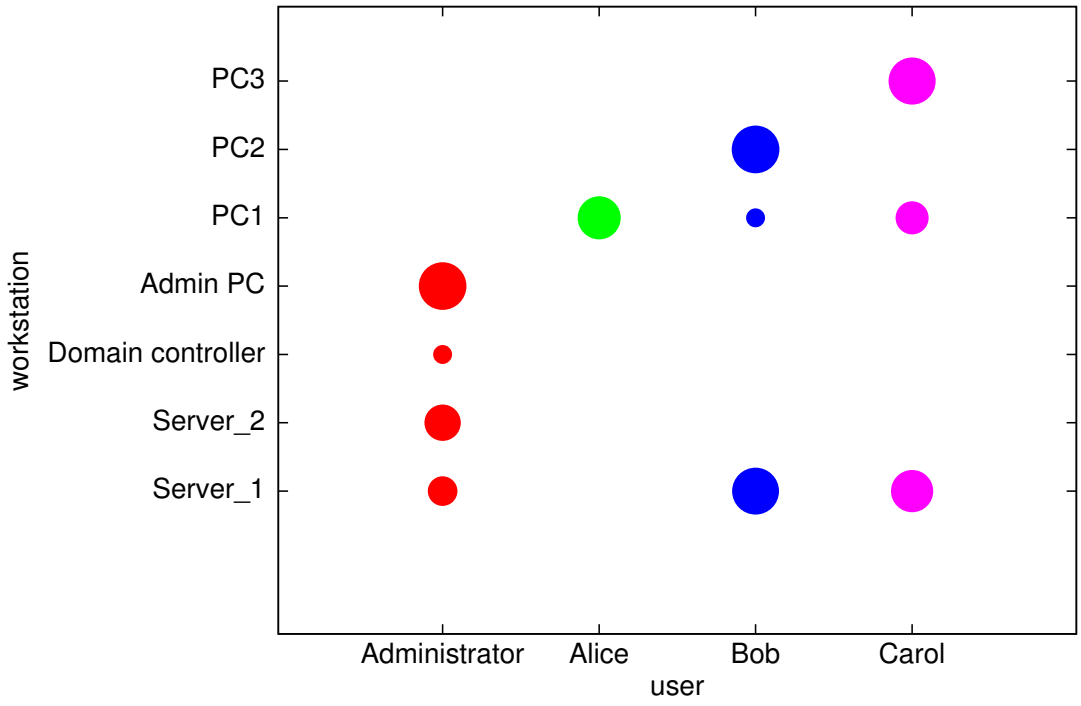


Figure 5.4: Distribution of user logon events in the training data set

tuple (users and workstations are marked on the x- and y-axis correspondingly). The bubbles are logarithmically scaled.

The outlier detection learned the mean and variance from the training data and then was applied on the testing data. The {user,workstation} tuples that were marked as anomalous due to the low probability of the number of events are shown in Figure 5.6.

The results presented in Figure 5.6 include all user behaviour events for user Bob connecting to Admin PC and Server_2, which are malicious according to the original scenario (see Section 5.2 for details). Besides this, the connection of user Alice to Server_1 was not marked as anomalous, since this server was used by the user group of Alice in the training data. Thus, the offered outlier detection algorithm was able to identify all malicious cases of user behaviour and produced no false positive alerts. This proves the high effectiveness of the offered approach and, in particular, the two-step probability check.

To prove that the proposed algorithm is also effective on the real data, it was applied on the Windows Event data set from Section 4.3.2, as described below.

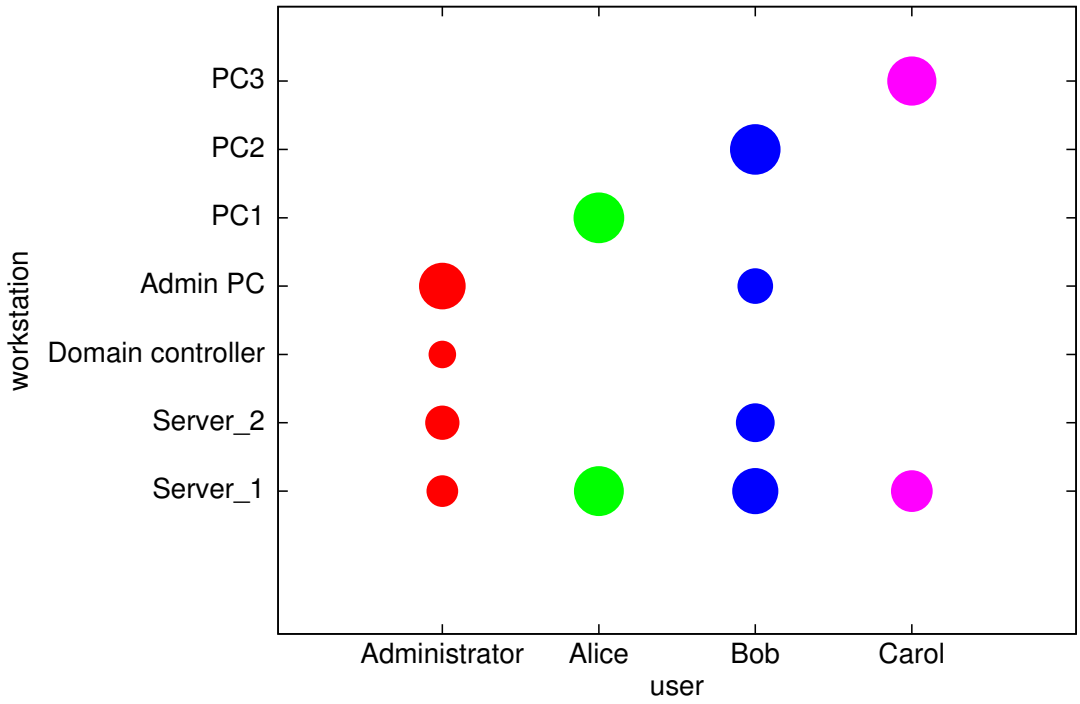


Figure 5.5: Distribution of user logon events in the testing data set

5.5 Application of the proposed outlier detection on the real data

The Windows Event data set from Section 4.3.2 was selected since it contains the events reflecting the user behaviour. Even though these events are mainly related to the automated activity of the user accounts, the traces of this automated activity can be still used for the outlier detection. However, the offered approach should be adopted to better fit to the real data. Since the monitored network covered by the real data set is much bigger (in terms of a number of users and hosts) than the simulated one, it could be reasonable to increase the number of features for the outlier detection. The additional features are the time of day and the day of the week (that can help to identify user activity in the unusual time), as well as the source IP address (to distinguish between the logon-related events of the same user coming from different computers and/or locations). Thus, instead of {user,workstation} tuples, the {source,user,day,hour,destination} tuples will be used for the outlier detection on the real data.

Besides the additional features, the outlier detection cannot be applied with proposed two-step probability check to the data set as is, since the Windows Event

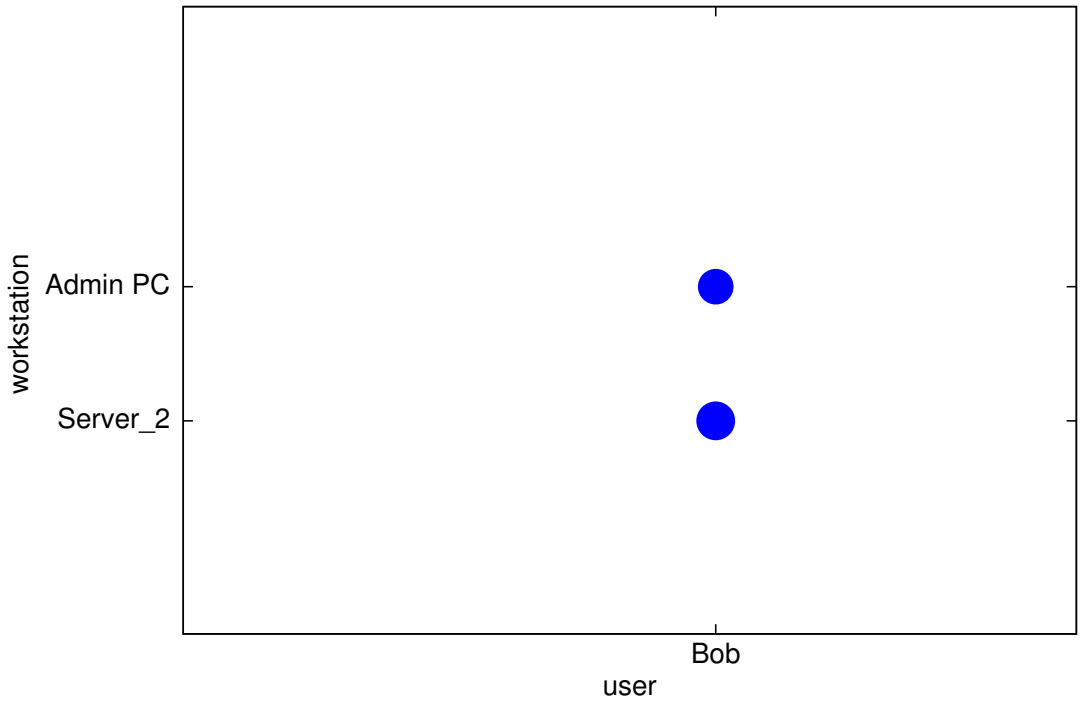


Figure 5.6: Detected user behaviour anomalies

data set does not contain the information about the user groups. So, to analyse this data set, the original Algorithm 3 from Section 5.3 was used.

Next, to apply the proposed outlier detection, it is important to check that the real data follows either Poisson or negative binomial distribution. In this data set, 63% of {source,user,day,hour,destination} tuples have a distribution of the number of logon-related events per time interval with variance higher than mean by more than 1.5 times, which can be modelled with the negative binomial distribution. For the rest of these tuples, the variance is slightly less than mean ($0.9\mu < \sigma^2 < \mu$) and the Poisson distribution can be applied for the probability calculation.

Finally, the real data also requires the extra filtering in addition to one proposed in Section 5.2.1. Due to the algorithm specifics, the part of the data set should be used to train the user behaviour models. In this case, all user behaviour events (combinations of {source,user,day,hour,destination}) that were not captured by the training data will be automatically classified as anomalies. Due to specifics of this particular data set, it contains a lot of events that were produced by users or computer accounts that were active for just several days within the 3-months period. To avoid the enormously high number of false positive alerts caused by such issues, the outlier detection will be executed only for those users and computer accounts that were active for at least 45 days within 3 months cov-

ered by the data set. After this extra filtering, there are 618,043 events covering 679 unique user accounts. However, this filtered data subset still contains ≈29,000 unique combinations of {user,source,destination} and ≈191,000 unique combinations of {user,source,destination,weekday,dayhour} features. This fact makes it very complicated to apply the proposed outlier detection to the data. However, the outcomes of the outlier detection prove that it is still capable of producing explainable results. The statistics for the outliers are presented in Table 5.6.

Table 5.6: Statistics for both types of detected issues in the data set

Type of issue detected	Number of events/issues
Events with 0 probability (IP address change, destination host change)	675 of 679 users, 184,521 events
Events with a probability less than the optimal threshold (unexpected number of logon events at destination host at specific time/-date)	328 of 679 users, 56,997 events

Table 5.6 shows the number of events for both types of outliers (described in details in Section 5.3.3). Altogether, there are 241,518 events marked as anomalies, which is a relatively high value. The reason for such high number of outliers can be found in the data set specifics. The high number of unique combinations in {user,source,destination,weekday,dayhour} tuples and constantly changing IP addresses (due to the settings of DHCP server), as well as changes of destination hosts (that are not malicious on their own) created a lot of events that were new for the trained models. However, since all such records have 0 probability, they can be easily filtered out by the operator of the security system. Of course, these filtered events can also contain some unexpected connections that could be malicious. Based on the available data set, it seems impossible to distinguish such events from noise created due to IP address changes and benign unique connections. However, if the model can be trained over the longer time period and include the user group information, the noise level should be reduced, which would allow detecting such events more easily.

The rest of 56,997 anomalies with low probability is still a high number, even though the optimal threshold value was automatically set to 0.09% (which is relatively low). These events are classified as anomalies mainly due to the abnormal number of logon events. The fact that the output is ordered (ranked) by the probability of the number of logon events helps the operator of the security system to check the most suspicious anomalies placed on the top of the list. By checking the

top of this list, two users with an average number of logon/logoff events per day exceeded by 100 times were identified, which clearly indicates either a security or a misconfiguration issue[28]. Other suspicious user behaviour cases were related not only to the high number of logon events but also to the unusual logon time. The first 10 users with the lowest probability are described below:

- **user 1**: an unusual number of logon events
- **users 2-4**: an unusual number of logon events on Sunday, 03:00
- **users 5-9**: an unusual number of logon events
- **user 10**: an unusual number of logon events on Wednesday, 04:00

Thus, both high number of logon events and suspicious user activity in the night time were identified and highly ranked by the proposed outlier detection algorithm, which proves its effectiveness.

To estimate the performance of the algorithm, it was executed on the same hardware, as described in Section 4.3.4. Utilising 40 CPU threads, the analysis of logon-related events took 14 minutes (including data selection and filtering procedures), which can be considered a good result taking into account the full size of the data set (as described in Section 4.3.2, the full data set contains 123 million events that need to be processed to extract 618,043 relevant user behaviour events suitable for the analysis).

5.6 Chapter summary

The User Behaviour Outlier Detection algorithm developed in this chapter solves the problem of detecting malicious user behaviour cases without access violation. The offered approach was able to detect all attacks in the simulated data set without producing any false positive alerts (thanks to the two-level probability check described in Section 5.3.1). On the real data, the proposed algorithm demonstrated both high effectiveness and good performance. Besides that, the optimal threshold detection and ranking of anomalies make the offered approach easier to apply and to parse for the operators of SIEM systems.

The application of the offered approach requires the training phase, which would need a clean data set to learn the normal user behaviour from it. The creation of such a data set is a rather sophisticated task, which was also addressed in this chapter. The first option to create the clean data set is to develop a scenario

[28]These 2 users had approximately 1500 logon events per day, whereas the mean value was equal to 15.

of normal user behaviour and use a special simulation tool to implement it. The log messages needed for the training phase are then generated during execution of the simulation tool. Although this approach was used to create a small data set with both normal and abnormal scenarios for the proof-of-concept, it could be rather complicated to create such a scenario for large enterprises with thousands of employees. Another option to create a training data set would be cleaning of the log messages (generated during the user activity) from attacks and other malicious behaviour. However, to get such a cleaned data set, one should be able to identify all attacks and malicious user behaviour cases in it, which is technically almost impossible. Even if this option would be available, it will make the further application of anomaly detection useless, since all attacks would be detected with the data cleaning methods. The last option to create a training data set (which was used in this chapter to apply the proposed User Behaviour Outlier Detection on the real data) is to take a subset of the data to learn the model of normal user behaviour. The main disadvantage of this approach is that all attacks and malicious user behaviour cases from the training data subset will be learned as normal. However, all new attacks, not included into the training subset, will still be detected by the algorithm. The last approach can be classified as unsupervised, since it does not require any data labels for training (please see Section 2.2.5 for classification of machine learning approaches).

Trained on the subset of the real data, the offered unsupervised User Behaviour Outlier Detection was able to identify several cases of suspicious user behaviour, which were reported to the enterprise partner. Thus, the offered approach was proved suitable for analysis of user behaviour within large enterprises and helps to detect special cases of unusual user activity, which were not covered by SIEM systems before.

Chapter 6

Conclusion

Nowadays, the major challenges for existing SIEM systems are related to the huge data volumes that these systems collect, store and analyse. The data analysis is the most computational heavy of these tasks, due to the high complexity of data processing techniques used in the SIEM systems. The need to process the data in a nearly real-time or at least with high speed[29] makes the application of data analysis methods based on machine learning[30] extremely complicated. At the current state of SIEM systems, the high-speed application of anomaly detection methods was only possible either with manual feature construction for each particular data set and source or for limited data volumes[31]. Due to these circumstances, anomaly detection approaches are not widely implemented in the modern SIEM systems, which, in turn, leads to the limitations in data analysis options for the security experts and operators of such systems.

This thesis solves the problem of high-speed analysis of Big Security Data in SIEM systems. The offered solution is based on the contributions of the thesis, such as the new architecture for SIEM system, log format for high-speed normalisation of security events, integration of in-memory and in-database processing techniques and high-speed outlier detection methods. The special focus of this thesis is given to the development of high-speed unsupervised anomaly detection methods. The proposed Hybrid Outlier Detection has high performance, is easy to parallelise, can be applied on the heterogeneous data (including textual and continuous numerical features), automatically determines optimal parameters for clustering and thresholds, returns ranked clusters of anomalies and has better effectiveness compared to other outlier detection methods. This outlier detection method is an example of a generic algorithm that can be applied on the data with

[29] As mentioned in Section 3, a high-speed SIEM system should have the processing rate higher than the event input rate.

[30] The data processing methods were reviewed in details in Chapter 2.

[31] The review of modern anomaly detection methods was provided in Section 2.2.5.2.

minimal input from the security expert or an operator of a SIEM system. Another type of the algorithm proposed in this thesis is the User Behaviour Outlier Detection algorithm. Different to the generic Hybrid Outlier Detection, it solves the previously uncovered problem of detecting malicious user behaviour that does not cause the access violation alerts. The offered User Behaviour Outlier Detection has a lightweight model (which ensures the high performance), automatic threshold detection and returns ranked lists with outliers. The two-level probability check covered in Section 5.3.1 reduces the number of false positive alerts for users with group information.

The performance of both outlier detection methods proposed in this thesis allows their integration into SIEM system for the purpose of high-speed processing of Big Security Data, which brings new analysis options for security experts and operators of SIEM systems. The section below discusses in details the integration and application of outlier detection methods, as well as their benefits and limitations.

6.1 Implementation of anomaly detection in modern SIEM systems

The main advantage of anomaly detection methods over the misuse detection is that the first ones are able to detect previously unknown issues or attacks through capturing the deviations from the system state, which is assumed normal. These deviations are measured based on probability, score from a statistical model or some heuristic metrics. By their nature, such probabilistic outlier scores imply the presence of both type I and II errors, i.e. false positive and false negative alerts. When anomaly detection is applied on Big Data, it often produces a huge number of alerts, including both the false positives and false negatives.

To reduce the number of alerts in general, the outlier score produced by the model should be kept and used for ranking of anomalies. In this way, the simple threshold can be used to select only the most suspicious anomalies. However, taking into account the heterogeneous nature of Big Security Data, the optimal threshold value varies between different data sources, time intervals and data subsets. To simplify the task of the security experts and operators of SIEM systems, it is important to automatically detect the optimal threshold value as proposed in this thesis (please see Sections 4.2.3 and 5.3.2 for details).

Besides this, grouping of outliers into clusters also makes it easier to deal with false positive and false negative alerts. Even though the clustering does not reduce the number of alerts on its own, it becomes much easier for the human operator to look at clusters of similar outliers (for example, a group of failed logon events for

the same user and workstation, where only the timestamps differ) rather than to dig through a mixture of different events placed next to each other due to similar outlier score.

Other options for reducing the number of false positive and false negative alerts are data specific and require special techniques that are based on the knowledge of data analyst. An example of such technique is the two-level probability check proposed in this thesis for User Behaviour Outlier Detection. Even though such approaches usually have higher efficiency than the generic ones, they cannot be easily extended to other data sources and data subsets. On the one hand, it becomes the main limiting factor for the integration of such approaches into Big Data processing pipeline. On the other hand, such customised approaches should still be available for the security expert, since they take into account more data-specific details and allow reaching higher effectiveness through focusing on the particular data type.

Another major problem for application of anomaly detection is the processing of textual data. Whereas the signatures can be easily applied to the textual data (for example, capturing messages such as "ssh login failure" with regular expressions), the anomaly detection methods try to calculate the distance or similarity between normal system state and the potential outlier. Finding such similarity or distance between two textual log messages is rather complicated. Therefore, the classical solution for this problem requires knowledge about each specific data set and source. Based on this knowledge, specific numerical features (for example, number of failed logon events per user for Windows Events data set) can be constructed out of textual data. However, this approach is not widely applicable on Big Data due to the variety of data sets and data sources. In this thesis, the novel Hybrid Outlier Detection is proposed to solve the problem of textual data processing. The Hybrid Outlier Detection utilises one-hot encoding to convert textual features into vector space model stored as the sparse matrix. This measure enables the possibility to apply outlier detection methods on original fields of log messages, such as username, IP address, etc. The thesis also solves the problem of conversion mixed data containing both textual and continuous numerical features into vector space model by applying customised feature discretisation method. Thus, the proposed outlier detection supports all possible types of security-related data and can be applied to any Big Security Data contexts.

The last but not the least issue for integration of anomaly detection techniques into a SIEM or IDS system is the high computational complexity of these techniques. The application of these methods for Big Data processing requires a lot of resources, i.e. a high amount of CPU cores or a large volume of RAM if the results have to be calculated in a reasonable time. However, this limitation should not be the reason to avoid integration of such methods into security analytics

systems. According to the definition of "high-speed" from Chapter 3, a security analytics system should be able to process the data with the speed comparable to the event input rate. This thesis has shown that such high-speed anomaly detection is still possible even for the Big Security Data. To reach this goal, a number of architectural solutions are proposed in this thesis, such as in-memory and in-database processing, or lightweight log format for event normalisation. The more sophisticated solution requires the development of high-speed and scalable outlier detection algorithms, which is also covered in this thesis (please see Chapters 4 and 5 for details). Lastly, the operator of SIEM system can decide to apply the heaviest anomaly detection algorithms only on selected relevant data feeds to reduce the number of events for the analysis.

All in all, this thesis proposes to apply anomaly detection techniques *in addition* to the misuse detection, and not instead of it. In REAMS, the normalised data are first analysed using signatures (misuse detection) and then stored in the in-memory SAP HANA database backend. Next, at the database level, the stored data are correlated and processed using predefined SQL-based queries (which can be also classified as a misuse detection). These queries are used to detect suspicious events and user behaviour by ranking events according to different metrics (e.g. top 10 users with the highest number of authentication failures; top 10 users with high file share activity, etc.). Finally, the anomaly detection methods are applied to identify the suspicious events that were not captured by signatures and custom queries. The application of anomaly detection methods can be also divided into two phases. First, the generic Hybrid Outlier Detection is executed to highlight the most suspicious clusters of events over all data sources. Second, the data-specific approaches such as User Behaviour Outlier Detection are applied in order to identify more suspicious events in selected data sets.

The task of the operator of REAMS is to react on the alerts from misuse detection modules and regularly check the top of the output of the anomaly detection modules, i.e. to prove the alerts on the most highly ranked outliers and clusters of outliers.

The next section reviews the results detected by REAMS in real data with both misuse and anomaly detection methods and demonstrates the add-on value from the usage of anomaly detection methods, as well as describes the divergence in the detected issues between different outlier detection algorithms.

6.2 Overview of issues detected in the data using different analysis methods[32]

The hybrid detection of attacks (with both misuse and anomaly detection methods) in REAMS is illustrated on the real data set with Windows Events, which was described in Section 4.3.2. Some properties of the data set, such as the absence of events recorded for user interactive logon actions and a high number of events related to the activity of computer accounts (which do not directly reflect the activity of real users), makes the detection of real malicious activity a very complicated task. Nevertheless, using the hybrid detection, many suspicious users and events were identified by REAMS. The overview of these issues is provided in Table 6.1.

Table 6.1: Types of issues detected by REAMS in the real data

N	Method	Type of issue detected	Number of events/issues
1	Signatures	Failed login brute-force attack ($\geq$ 10 failed logins by the same user)	2,784 alerts
2	Signatures	Successful login brute-force attack ($\geq$ 10 failed logins by the same user followed by successful login)	24 alerts, 3 of which are related to real users (non-computer accounts)
3	Queries	Failed network share access (Event 5140)	1,483 of 330,363 network share access events
4	Queries	Logon failure (Event 4625)	2,411 users including computer accounts
5	Queries	Kerberos replay attack (Event 4,649)	2 events
6	Queries	Users accessing hidden private shares of other users	1,612 users
7	Hybrid Outlier Detection	Authentication failures for SnapDrive	26 of 206 events ranked as most suspicious events (9 of 10 most suspicious clusters) in the data set
8	Hybrid Outlier Detection	Failed network share access (Event 5140)	422 of 1,483 events are included in the top 15 suspicious clusters

[32]Some of the results described in this section have been published in [123].

9	Hybrid Outlier Detection	Various authentication failures (Events 539, 560, 4625, 4776, ...)	98% of top 100,000 suspicious events
10	kNN-based Outlier Detection	Users with a high number of various authentication failures (account lock outs, Kerberos authentication failures, disabled accounts)	34 users including computer accounts
11	kNN-based Outlier Detection	Users with a high number of "special privileges assigned to new logon" messages	18 users including computer accounts
12	User Behaviour Outlier Detection	Events with 0 probability (IP address change, destination host change)	675 of 679 users, 184,521 events
13	User Behaviour Outlier Detection	Events with a probability less than the optimal threshold (unexpected number of logon events at destination host at specific time/-date)	328 of 679 users, 56,997 events
14	User Behaviour Outlier Detection	Average number of logon/logoff events per day exceeded by 100 times	2 users

Table 6.1 shows that each analytical method was able to identify some issues or suspicious activity in the data.

An exemplary signature implemented in REAMS for detection of brute-force attacks[33] was applied at the time of data normalisation (please see Section 3.2.3 for details). Using this signature, both unsuccessful and successful brute-force attacks were identified (see lines 1-2 of Table 6.1). The most suspicious of detected issues are the 3 series of failed logon events (related to non-computer user accounts) followed by successful ones.

The queries implemented in REAMS using SAP HANA SQLSCRIPT (please see Sections 2.3.2 and 3.2.3 for details) also captured various authentication failures, which are listed in rows 3-6 of Table 6.1. The most suspicious of these issues

[33]The REAMS is a prototype of a SIEM system and does not have a large signature knowledge base, rather it only contains several signatures for proof-of-concept. From the exemplary signatures, only the signature for brute-force attack detection was applicable on Windows Events data set.

are the Kerberos replay attacks and connections to the private file shares of users, whereas logon failures and failed share accesses are the results of misconfiguration in the Windows domain policies.

The anomaly detection methods were applied after misuse detection to search for suspicious events that were identified neither by signatures nor by custom queries implemented in REAMS. First, the generic outlier detection method (Hybrid Outlier Detection) was executed. The detected issues shown in lines 7-9 of Table 6.1 partly overlap with the issues identified using queries and signatures. E.g. failed network share access cases were detected with both queries and Hybrid Outlier Detection. The Hybrid Outlier Detection also highlighted various authentication failures (line 9 of Table 6.1), whereas some of them were already captured with queries and signatures (lines 1, 2 and 4 of Table 6.1). Together with the fact that the 98% of top 100,000 suspicious events from the output of the outlier detection are authentication failures, it proves that the proposed method is able to identify abnormal activity in the data[34]. The Hybrid Outlier Detection also identified many events that would stay undetected otherwise, including authentication failures for SnapDrive storage management software described in line 7 of Table 6.1. These events are very rare for the data set (206 of 123 million events) and were not highlighted by signatures or queries due to their relatively small volume.

The next anomaly detection method applied on the data set was the kNN-based Outlier Detection. This method also detected various authentication failures (lines 10 and 11 of Table 6.1), but focused more on the specific subtypes of authentication failures, such as, for example, "special privileges assigned on logon". The results from this outlier detection method are also partly overlapping with the results of signatures, queries and Hybrid Outlier Detection. For example, 22 of 56 suspicious users were also identified by Hybrid Outlier Detection[35]. However, the specific types of authentication failures were not always detected by other methods (due to the lack of specific signatures or the probabilistic nature of Hybrid Outlier Detection), which makes this type of anomaly detection valuable for analysis of Windows Events.

Finally, the User Behaviour Outlier Detection was used to identify suspicious users without access violation (please see Section 5.5 for details). The results from this type of anomaly detection include two types of outliers (described in details in Section 5.3.3), which are listed in lines 12-13 of Table 6.1. The User Behaviour Outlier Detection highlighted several very suspicious cases of a high number of logon events at the unusual time. This method also allowed to identify 2 users with unusually high activity (100 times more logon/logoff events per day,

[34]Please see Section 4.3.3 for details on issues detected in the Windows Events data set with Hybrid Outlier Detection.

[35]Please see Section 4.3.5 for details.

compared to the average value for all users), which is mentioned separately in line 14 of Table 6.1.

Thus, both misuse and anomaly detection methods produced the relevant results on the selected data set. Even though the results of all methods overlap with each other, the proposed outlier detection algorithms identified a number of issues, which would stay undetected otherwise. Due to the different focus of the developed outlier detection methods (Hybrid Outlier Detection for generic analysis, kNN-based Outlier Detection for analysis of specific types of authentication failures in Windows Events, User Behaviour Outlier Detection for capturing malicious user behaviour without access violation), the results obtained from these methods complement each other and enrich the detection capabilities of REAMS.

All anomaly detection algorithms have demonstrated high performance. The Hybrid Outlier Detection processed 123 million events within 36 hours (3.4 million events per hour). Compared to its predecessor (Universal Outlier Detection), it is capable of processing events at 14 times higher rate (please see Sections 4.3.4 and 4.3.5 for details). The kNN-based Outlier Detection (re-implemented in this thesis) needs the maximum of 65 hours to process 51 data views in parallel [123], whereas the original version of this algorithm [49] was unable to run on such data set at all, even though its implementation was included into RapidMiner [157]. Finally, the User Behaviour Outlier Detection needs 14 minutes to filter 123 million events and process 618,043 events (please see Section 5.5 for details), which implies the rate of at least 2.6 million events per hour (or 527 million events per hour before filtering), thanks to the lightweight user model proposed in this thesis. Compared to the input rate of 60,000 events per hour (for Windows Events data set from the large multinational company), the proposed algorithms can process the 50 times bigger data sets at high speed without allocation of extra hardware resources.

The next section summarises the contributions of this thesis that allowed to develop the high-speed SIEM system (REAMS) and integrate the reviewed outlier detection methods enabling processing of Big Security Data with high performance and accuracy.

6.3 Thesis contributions

The efficient and high-speed log analytics demonstrated in the section above was made possible despite many existing challenges for SIEM system technology (please see Section 1.2 for details). These challenges were addressed and overcome thanks to the contributions of this thesis that were listed in Section 1.3. These contributions are now reviewed with the focus on the technical and implementation details, as well as the performance and accuracy improvements:

- **Architecture for high-speed analysis of security events.**

  This thesis proposes the new architecture for the high-speed analysis of security-related events in a SIEM system. The proposed architecture is based on in-memory and in-database processing, as well as high-speed data normalisation and lightweight log format. In Chapter 3 of this thesis, the proposed architecture is compared with classical architecture, where the data should be retrieved from the database before processing. The performance evaluation has shown that the proposed architecture is up to 6 times faster than the classical one (on the same hardware). The performance gain is especially high on the larger data set, which makes the proposed architecture even more attractive for SIEM systems.

- **Integration of in-memory and in-database processing technologies into SIEM system.**

  To build an architecture for the high-speed analysis, both in-memory and in-database processing techniques were integrated into the prototype of the SIEM system (REAMS). In this thesis, the SAP HANA SQL-database [39] was utilised as in-memory storage and in-database processing engine. All data processed by the SIEM system were stored in the main memory of the database. There the data were analysed with SQLSCRIPT [42] queries and various outlier detection algorithms. These algorithms are available either from SAP HANA PAL [40], or through R integration [41]. This solution was validated on the replicated data set with up to 48 million records, which was processed with different algorithms based on k-means clustering (please see Section 3.4 for details). The comparison with classical architecture (based on PostgreSQL database) has proved that the proposed solution has much higher performance. On the same data set, the performance of the classical approach — which requires retriving the data from the database — was enough to process only 12 million events within 8 minutes, whereas the in-memory solution processed 48 million events within 5 minutes.

- **Evaluation of log formats for security analytics to select the best option for high-speed normalisation of security events.**

  When a SIEM system stores and analyses all data in the main memory, the optimal log format becomes extremely important for efficient data processing. In this thesis, 5 different log formats[36] were evaluated (according to the developed criteria) on the Honeynet Challenge data set [127]. The analysis (please see Section 3.2.1 for details) was performed to select the best option for the SIEM system architecture and to improve the performance of REAMS.

- **Development and evaluation of novel high-speed outlier detection methods for SIEM system.**

  Besides the architecture for high-speed data processing, the data analysis methods should also have high performance to enable the Big Security Data processing in a SIEM system. The modern hybrid SIEM systems, such as HPI REAMS, combine two data analysis techniques, namely misuse and anomaly detection methods. The first ones are not computationally heavy and can be applied on the large data volumes. The second ones require many computational resources and are therefore not widely used for Big Data processing in SIEM systems.

  This thesis focuses on high-speed outlier detection methods and proposes the novel Universal Outlier Detection and Hybrid Outlier Detection algorithms (please see Chapter 4 for details), which are capable of Big Security Data analysis. To develop these novel approaches, the original Anomaly Detection algorithm from SAP HANA PAL was reviewed to identify and mitigate its disadvantages. Different to PAL Anomaly Detection, the proposed Universal Outlier Detection introduces the ability to work on heterogeneous data (including both textual/categorical and continuous numerical features) and implements parallel processing techniques. In Universal Outlier Detection, the categorical data are converted into the vector space model using one-hot encoding, whereas the numerical features are first mapped into categories using improved feature discretisation method (described in Section 4.2.1.1). The resulting vector space is divided into subsets that are clustered in parallel using spherical k-means to perform outlier detection on the data.

  The proposed Hybrid Outlier Detection further improves the speed and accuracy of the Universal Outlier Detection by training an ensemble of one-class

[36]The evaluated formats (also reviewed in details in Section 2.3.1) are: CEE [111], CEF [16], IODEF [15], IDMEF [112] and OLF (Object Log Format, which was jointly developed at HPI [44]).

SVMs on clusters (represented by concept vectors) from the output of spherical k-means clustering.

The algorithms are evaluated on the real Windows Events data set containing 123 million events after filtering (the data set is described in details in Section 4.3.2). Both algorithms were able to detect issues in the data, whereas the Hybrid Outlier Detection was approximately 10 times faster and had the AUC value equal to 73.9% against 69.8% for Universal Outlier Detection. The Hybrid Outlier Detection was also compared with newly re-implemented (to increase its performance) kNN-based Outlier Detection proposed by Goldstein et al. [49]. The results have shown that the generic Hybrid Outlier Detection was also able to detect the same issues as kNN-based Outlier Detection. Besides this, on the KDDCup 1999 Data, the Hybrid Outlier Detection reaches AUC of 98.4%. Thus, the proposed Hybrid Outlier Detection allows high-speed analysis of Big Security Data and outperforms other existing methods without loss of accuracy.

- **Development and evaluation of novel outlier detection method for analysis of user behaviour in a SIEM system.**

  Besides the proposed generic high-speed outlier detection algorithms (Universal Outlier Detection and Hybrid Outlier Detection), this thesis also focuses on the special case of user behaviour without access violation. As mentioned in Chapter 5, this scenario often stays uncovered by existing detection methods. Of course, there is still a chance that generic outlier detection algorithms will be able to capture the most evident deviations in user behaviour resulting in high numbers of log messages, even without access violation. However, to detect most of suspicious user behaviour cases, a special targeted outlier detection method based on the user behaviour model is required.

  This issue was thoroughly analysed in this thesis. The specified user behaviour case was simulated using the specially developed tool, as described in Section 5.2. The simulated data were analysed to propose the most efficient user behaviour model capable of the high-speed data analysis. As a result, the User Behaviour Outlier Detection was developed and proposed in this thesis. This algorithm builds a unique user behaviour model based on a number of user activities/connections per time interval. The algorithm also models the behaviour of user groups to correctly classify the cases when abnormal or unusual user actions are rather normal in the context of his/her user group. This technique, namely the two-level probability check, significantly reduces the number of false positive alerts (please see Section 5.3.1 for details).

Both high performance and effectiveness of the proposed User Behaviour Outlier Detection were proved on the same Windows Event data set, as other outlier detection approaches. Running in parallel with 40 CPU threads, the proposed algorithm was able to filter 123 million log messages and process 618 thousand of user-related events within just 14 minutes. The output of the User Behaviour Outlier Detection, described in Section 5.5, shows that the algorithm is able to identify suspicious user activities that were reported to the partner enterprise.

- **Usability and accuracy improvements for outlier detection algorithms through ranking and clustering of outliers and integration of automatic parameter detection.**

  All novel outlier detection methods proposed in this thesis (Universal Outlier Detection, Hybrid Outlier Detection and User Behaviour Outlier Detection) were developed taking into account their usability for operators of SIEM systems. To apply outlier detection methods, an operator often needs a specific knowledge about the data or the selected algorithm to configure various parameters and thresholds. Further, the output of the outlier detection algorithms often contains a large number of events including many false positive alerts. In the end, outlier detection becomes a rather sophisticated and time-consuming task for an operator of a SIEM system.

  This thesis provides solutions to these usability issues. First, the proposed generic clustering-based outlier detection algorithms utilise a method to automatically detect an optimal number of clusters whenever possible and thus reduce the amount of the input parameters (please see Section 4.2.4 for details). Second, all offered outlier detection algorithms also support automatic detection of outlier threshold, as mentioned in Sections 4.2.3 and 5.3.2. Third, all proposed algorithms implement the ranking of outliers (mentioned in Sections 4.3.1.1 and 5.3.3), which helps the operator of SIEM system to concentrate on the several most suspicious outliers on the top of the output, instead of scrolling through the full list of outliers. Finally, the Hybrid Outlier Detection algorithm clusters the outliers together in the output (as also mentioned in Section 4.3.1.1), which makes the output much easier to parse for the security expert. Thus, instead of looking on a long list with different standalone outliers mixed with each other, a security expert can analyse several clusters with the highest outlier score.

All contributions of this thesis are integrated into the jointly developed prototype of SIEM system (HPI REAMS). This system resolves the most challenging problems of modern IDS and SIEM systems and becomes a proof-of-concept solution capable of Big Security Data processing.

Besides the high-speed outlier detection methods that were the main focus of this thesis, nowadays there is a number of novel emerging technologies that can be utilised to further improve existing SIEM systems. The continuous development of HPI REAMS considers these technologies to define the directions for the future work, which is described in the section below.

6.4 Future work

The emerging modern technologies and architectural trends for future SIEM systems can be grouped into several directions. These directions include (1) data enrichment approaches, (2) new data analysis and detection methods and (3) scalable Big Data processing technologies. This section discusses these directions in details, including the ways of their integration into HPI REAMS.

- **Data enrichment approaches.** At the current state, HPI REAMS supports the direct collection of log messages from individual workstations and domain controllers. More important, REAMS can be connected to various Log Management servers, as well as other IDS and SIEM systems. These systems forward all collected log messages from the enterprise network to REAMS, including data from workstations, proxy and web servers, firewalls and so on. All these data are correlated and analysed in REAMS to identify malicious activities. To improve the detection capabilities of the SIEM system, log data can be enriched with other information sources. These new sources include both external Threat Intelligence and vulnerability data, and internal intermediary devices within the enterprise network. First, correlation of proxy and firewall logs with domain blacklists from Threat Intelligence can be used to identify suspicious user behaviour cases or infected workstations that regularly connect to malicious web sites. Next, the correlation of internal inventory with vulnerability data can highlight the weak spots in the enterprise infrastructure. In turn, this information can be used to adjust the score of related alerts in the output of the SIEM system, so that the alerts from most vulnerable systems will have a higher ranking. Finally, collecting data from intermediary devices should broaden the scope of the SIEM system. In addition to security monitoring of intermediary devices themselves, it will allow monitoring availability problems caused by disruptions in network protocols [179]. Indeed, many routing protocols have security issues as well [180], which stay unmonitored even though the routers themselves could be controlled by a SIEM (via syslog messages or Simple Network Management Protocol). The related work [181, 182] shows an example, how the routing protocol can be analysed to identify the reasons for large bursts in the protocol messages.

- **Novel data analysis and detection methods.** To further improve the detection capabilities of HPI REAMS, more high-speed outlier detection methods should be developed and integrated. However, the task of generic outlier detection is already covered with Universal Outlier Detection and Hybrid Outlier Detection algorithms, proposed in this thesis. Therefore, the main focus of new outlier detection algorithms should be specific use cases, such as analysis of user behaviour (also covered in this thesis), data flows, process creation logs, etc.

  Besides the classical outlier detection methods, the graph-based analysis also becomes a very promising direction. Under this approach, various information sources are used to construct a single graph representing the relations between entities in the enterprise network, as well as outbound communications. The resulting graph can be analysed with various graph inference techniques to perform threat detection. For example, in the related work such graph was built based on DNS and proxy server logs to determine malicious nodes in the network [183]. This type of analytics can be also re-implemented using in-memory and in-database processing techniques to reach higher performance, thanks to SAP HANA Graph engine [184].

- **Scalable Big Data processing technologies.** While the parallel and distributed (using R cluster) processing was widely used in this thesis, more scalable distributed approaches may be needed for Big Security Data in the future SIEM systems. The higher scalability can be reached, for example, with MapReduce [185] algorithms on Apache Hadoop [186] or Apache Spark [187]. All these technologies can be easily integrated into REAMS alongside SAP HANA, as shown in Figure 6.1.

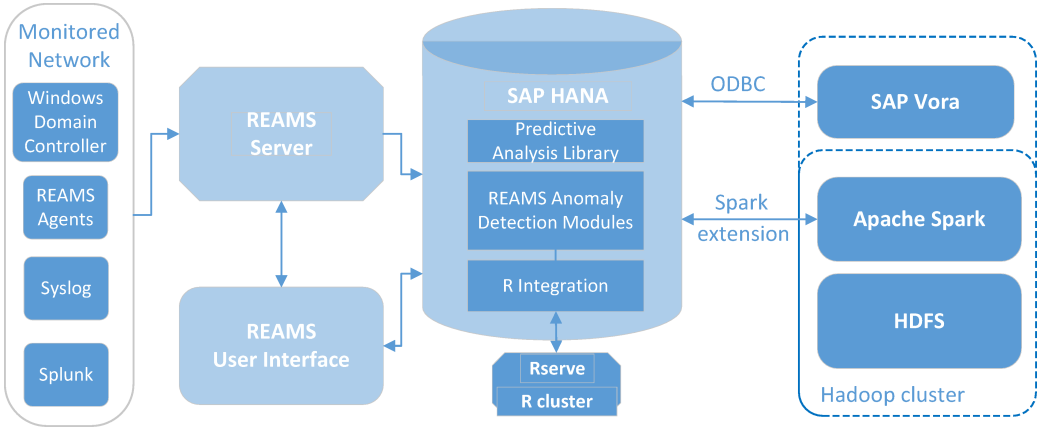


Figure 6.1: Integration of Apache Spark into REAMS for scalable Big Data processing

Figure 6.1 demonstrates the possibilities for the integration of MapReduce-based algorithms for distributed data processing into REAMS (the actual REAMS architecture is shown in Figure 3.3). First, the integration can be performed through SAP Vora distributed computing solution [188] using ODBC connection to SAP HANA database. Second, Apache Spark can be directly connected to SAP HANA through Spark extension [189]. Both options allow accessing the data stored in SAP HANA database and move it to the HDFS [190] for distributed analysis.

Both the proposed distributed analysis and the outlier detection methods developed in this thesis are related to the high-speed batch processing. Indeed, many outlier detection algorithms need to cluster batches of data in order to identify outliers. However, to react on security issues in real-time, a SIEM system should also be able to process Big Security Data in streaming mode. The actual version of REAMS contains two stream processing modules, namely Normalisation and Signature Analytics. In the future work, the system can be extended to include more stream processing modules, for example, Threat Intelligence correlation module. Thus, REAMS combines various batch and stream processing methods for better threat detection, as shown in Figure 6.2.

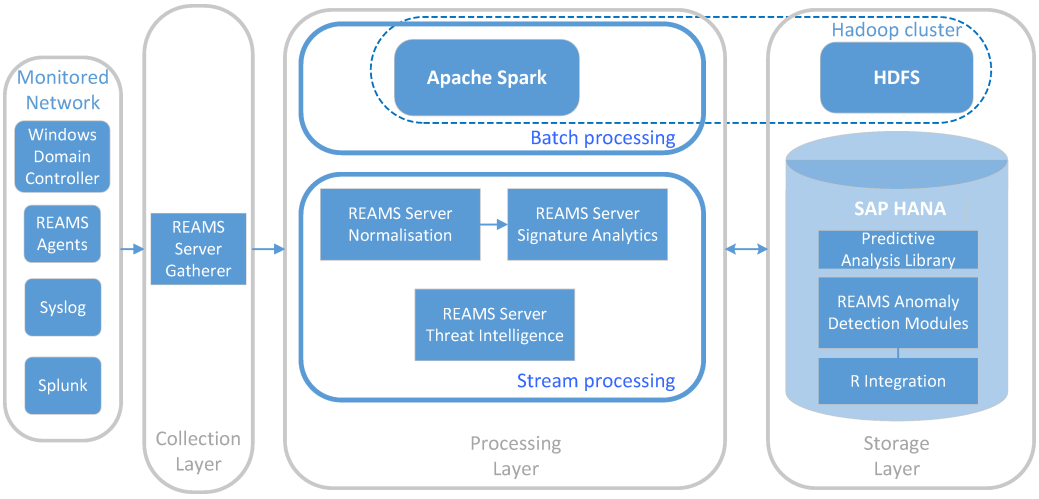


Figure 6.2: Combination of stream and batch processing in REAMS

Figure 6.2 presents how the stream and batch processing can be combined in a SIEM system. The data collected with the REAMS gatherer comes into processing layer, where it can be analysed in parallel using both batch processing in Apache Spark and stream processing in REAMS modules. Afterwards, the data are forwarded to the storage layer, where developed in-memory anomaly detection algorithms can be applied in the batch mode.

Finally, the data from the storage layer can be sent back to Apache Spark for distributed outlier detection.

Thus, all novel technologies can be easily integrated into HPI REAMS in the future, thanks to the system architecture, which was developed in this thesis. The reviewed modern high-speed data processing methods, analytical modules and data enrichment techniques ensure the sustainable development of this SIEM system prototype in the future.

Bibliography

[1] 2015 Cost of Data Breach Study: Impact of Business Continuity Management. Ponemon Institute, May 2015. http://public.dhe.ibm.com/common/ssi/ecm/se/en/sew03053wwen/SEW03053WWEN.PDF. Visited on 03-05-2018.

[2] Trustwave Global Security Report. Trustwave, 2016. https://www2.trustwave.com/GSR2016.html. Visited on 03-05-2018.

[3] Symantec Corp. Norton Cybersecurity Insights Report. Norton, 2016. http://us.norton.com/norton-cybersecurity-insights-report-global. Visited on 05-02-2018.

[4] Dot-dash-diss: The gentleman hacker's 1903 lulz. https://www.newscientist.com/article/mg21228440-700-dot-dash-diss-the-gentleman-hackers-1903-lulz/. Visited on 03-05-2018.

[5] Ron Rosenbaum. *The Secrets of the Little Blue Box*. Esquire, October 1971.

[6] Sherry Sontag and Christopher Drew. *Blind Man's Bluff: The Untold Story of American Submarine Espionage*. PublicAffairs, 2016.

[7] R.A. Kemmerer and G. Vigna. Intrusion detection: a brief history and overview. *Computer*, 35(4):27–30, April 2002.

[8] James P. Anderson. Computer Security Threat Monitoring and Surveillance, April 1980.

[9] R. Heady, G. Luger, A. Maccabe, and M. Servilla. The architecture of a network level intrusion detection system. Technical Report 9, Los Alamos National Laboratory (LANL), Los Alamos, NM, August 1990.

[10] Amrit T. Williams and Mark Nicolett. Improve IT Security with Vulnerability Management. Gartner, 2005. ID:G00127481.

[11] Oliver Rochford Kelly M. Kavanagh. Magic Quadrant for Security Information and Event Management. Gartner, 2015. ID:G00267505.

[12] Jonathan Stuart Ward and Adam Barker. Undefined By Data: A Survey of Big Data Definitions. *Computing Research Repository*, abs/1309.5821, 2013.

[13] Richard Zuech, Taghi M. Khoshgoftaar, and Randall Wald. Intrusion detection and Big Heterogeneous Data: a Survey. *Journal of Big Data*, 2(1):1–41, February 2015.

[14] Snort, an open source network intrusion prevention and detection system. `http://www.snort.org/`. Visited on 03-05-2018.

[15] R. Danyliw, J. Meijer, and Y. Demchenko. RFC 5070 - The Incident Object Description Exchange Format. *Network Working Group, IETF*, December 2007.

[16] ArcSight Common Event Format, July 2010. `https://kc.mcafee.com/resources/sites/MCAFEE/content/live/CORP_KNOWLEDGEBASE/78000/KB78712/en_US/CEF_White_Paper_20100722.pdf`. Visited on 03-05-2018.

[17] Sandeep Bhatt, Pratyusa K. Manadhata, Loai Zomlot, and Loai Zomlot Hewlett-packard Laboratories. The Operational Role of Security Information and Event Management Systems. *IEEE Security & Privacy*, 12(5):35–41, September 2014.

[18] Ting-Fang Yen, Alina Oprea, Kaan Onarlioglu, Todd Leetham, William Robertson, Ari Juels, and Engin Kirda. Beehive: Large-Scale Log Analysis for Detecting Suspicious Activity in Enterprise Networks. In *Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC '13*, ACSAC '13, pages 199–208, New York, New York, USA, 2013. ACM Press.

[19] Oliver Rochford Kelly M. Kavanagh, Mark Nicolett. Magic Quadrant for Security Information and Event Management. Gartner, June 2014. ID:G00261641.

[20] Seekinto solution brief. `https://web.archive.org/web/20160402160725/http://www.seekintoo.com/pdf/siem.pdf`. Visited on 03-05-2018.

[21] Bartłomiej Balcerek, Bartosz Szurgot, Mariusz Uchroński, and Wojciech Waga. ACARM-ng: Next Generation Correlation Framework. In *Building a National Distributed e-Infrastructure–PL-Grid*, pages 114–127. Springer, 2012.

[22] Peter Zadrozny and Raghu Kodali. *Big Data Analytics Using Splunk*. Apress, Berkeley, CA, 2013.

[23] McAfee KB82563. Common causes for slow ESM performance. `https://kc.mcafee.com/corporate/index?page=content&id=KB82563`. Visited on 03-05-2018.

[24] Enhance ArcSight 6.0 ESM Security with Fusion ioMemory ™ Performance Density and High Throughput. SanDisk, 2014. `https://www.sandisk.com/content/dam/sandisk-main/en_us/assets/resources/enterprise/overviews/Fusion_ioMemory_HP_ArcSight_Solution.pdf`. Visited on 03-05-2018.

[25] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[26] Shelly Xiaonan Wu and Wolfgang Banzhaf. The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, 10(1):1–35, January 2010.

[27] Malek Ben Salem and Salvatore J. Stolfo. Modeling User Search Behavior for Masquerade Detection. In *Proceedings of the 14th International Conference on Recent Advances in Intrusion Detection*, volume 6961 of *Lecture Notes in Computer Science*, pages 181–200. Springer Berlin Heidelberg, 2011.

[28] P. Gogoi, D. K. Bhattacharyya, B. Borah, and J. K. Kalita. A Survey of Outlier Detection Methods in Network Anomaly Identification. *The Computer Journal*, 54(4):570–588, 2011.

[29] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Hiren Patel, Avi Patel, and Muttukrishnan Rajarajan. A survey of intrusion detection techniques in Cloud. *Journal of Network and Computer Applications*, 36(1):42–57, January 2013.

[30] Ahmed Patel, Qais Qassim, and Christopher Wills. A survey of intrusion detection and prevention systems. *Information Management & Computer Security*, 18(4):277–290, 2010.

[31] Shan Suthaharan. Big Data Classification: Problems and Challenges in Network Intrusion Prediction with Machine Learning. *SIGMETRICS Perform. Eval. Rev.*, 41(4):70–73, April 2014.

[32] Samuel Marchal, Xiuyan Jiang, Radu State, and Thomas Engel. A Big Data Architecture for Large Scale Security Monitoring. In *Proceedings of the 3rd IEEE Congress on Big Data*, pages 56–63. IEEE, July 2014.

[33] NetIQ Sentinel User Guide. `https://www.netiq.com/documentation/sentinel-73/pdfdoc/s73_user/s73_user.pdf`. Visited on 03-05-2018.

[34] Muhammad Qasim Ali, Ehab Al-Shaer, Hassan Khan, and Syed Ali Khayam. Automated Anomaly Detector Adaptation using Adaptive Threshold Tuning. *ACM Transactions on Information and System Security (TISSEC)*, 15(4):1–30, April 2013.

[35] Tapas Kanungo, D.M. Mount, N.S. Netanyahu, C.D. Piatko, Ruth Silverman, and A.Y. Wu. An efficient k-means clustering algorithm: analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):881–892, July 2002.

[36] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[37] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, pages 226–231, 1996.

[38] Real-time Event Analysis and Monitoring System. `https://hpi.de/en/meinel/security-tech/security-analytics/reams.html`. Visited on 03-05-2018.

[39] SAP HANA. `https://www.sap.com/products/hana.html`. Visited on 03-05-2018.

[40] Predictive Analysis Library. `http://help.sap.com/hana/SAP_HANA_Predictive_Analysis_Library_PAL_en.pdf`. Visited on 03-05-2018.

[41] SAP HANA R Integration Guide. `https://help.sap.com/hana/SAP_HANA_R_Integration_Guide_en.pdf`. Visited on 03-05-2018.

[42] SAP HANA SQLScript Reference. `https://help.sap.com/viewer/de2486ee947e43e684d39702027f8a94/2.0.00/en-US`. Visited on 03-05-2018.

[43] PostgreSQL: The world's most advanced open source database. `http://www.postgresql.org/`. Visited on 03-05-2018.

[44] Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, and Christoph Meinel. Hierarchical object log format for normalisation of security events. In *2013 9th International Conference on Information Assurance and Security (IAS)*, IAS '13, pages 25–30. IEEE, December 2013.

[45] Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, and Christoph Meinel. Normalisation of Log Messages for Intrusion Detection. *Journal of Information Assurance and Security*, 9(3):167–176, September 2014.

[46] ManageEngine EventLog Analyzer. `https://www.manageengine.com/products/eventlog/`. Visited on 03-05-2018.

[47] Gerard Salton, Andrew Wong, and Chung-Shu S. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[48] Kurt Hornik, Ingo Feinerer, Martin Kober, and Christian Buchta. Spherical k-Means Clustering. *Journal of Statistical Software*, 50(10):1–22, 2012.

[49] Markus Goldstein, Stefan Asanger, Matthias Reif, and Andrew Hutchinson. Enhancing Security Event Management Systems with Unsupervised Anomaly Detection. In *Proceedings of the 2nd International Conference on Pattern Recognition Applications and Methods*, pages 530–538. SciTePress - Science and and Technology Publications, 2013.

[50] Meena Mahajan, Prajakta Nimbhorkar, and Kasturi Varadarajan. The Planar k-Means Problem is NP-Hard. *Theoretical Computer Science*, 442:274–285, July 2009.

[51] Larry M. Manevitz and Malik Yousef. One-Class SVMs for Document Classification. *Journal of Machine Learning Research*, 2:139–154, December 2001.

[52] Inderjit S. Dhillon and Dharmendra S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1-2):143–175, 2001.

[53] Windows Events. `https://msdn.microsoft.com/en-us/library/windows/desktop/aa964766(v=vs.85).aspx`. Visited on 03-05-2018.

[54] Royce Robbins. Distributed Intrusion Detection Systems: An Introduction and Review. SANS Institute, 2003. `https://www.sans.org/reading-room/whitepapers/detection/distributed-intrusion-detection-systems-introduction-review-897`. Visited on 03-05-2018.

[55] Gideon Creech. *Developing a high-accuracy cross platform Host-Based Intrusion Detection System capable of reliably detecting zero-day attacks*. PhD thesis, University of New South Wales, 2013.

[56] Anup K. Ghosh and Aaron Schwartzbard. A Study in Using Neural Networks for Anomaly and Misuse Detection. In *Proceedings of the 8th Conference on USENIX Security Symposium - Volume 8*, SSYM'99, pages 12–12, Berkeley, CA, USA, 1999. USENIX Association.

[57] Ricardo Jorge Santos, Jorge Bernardino, and Marco Vieira. Approaches and Challenges in Database Intrusion Detection. *ACM SIGMOD Record*, 43(3):36–47, 2014.

[58] Ismail Butun, Salvatore D. Morgera, and Ravi Sankar. A Survey of Intrusion Detection Systems in Wireless Sensor Networks. *IEEE Sensors Journal*, 14(5):1370–1379, 2014.

[59] Buecker Axel, Andreas Per, and Paisley Scott. Understanding IT Perimeter Security. IBM, 2008. `https://www.redbooks.ibm.com/redpapers/pdfs/redp4397.pdf`. Visited on 03-05-2018.

[60] Michael T. Raggo. *Mobile Data Loss: Threats and Countermeasures*. Elsevier Inc., 2016.

[61] Arun Madan, Sridhar Muppidi, Nilesh Patel, and Axel Buecker. Securely Adopting Mobile Technology Innovations for Your Enterprise Using IBM Security Solutions. IBM, 2013.

[62] R. Gerhards. RFC 5424 - The Syslog Protocol. *Network Working Group, IETF*, 2009.

[63] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-time. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, page 3, Berkeley, CA, USA, 1998. USENIX Association.

[64] Mike Fisk and George Varghese. Applying Fast String Matching to Intrusion Detection. DTIC Document, 2002. `www.dtic.mil/get-tr-doc/pdf?AD=ADA406266`. Visited on 03-05-2018.

[65] Steven T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer. STATL: An attack language for state-based intrusion detection. *Journal of Computer Security*, 10(1-2):71–103, 2002.

[66] Frédéric Cuppens and Rodolphe Ortalo. LAMBDA: A Language to Model a Database for Detection of Attacks. In *Recent Advances in Intrusion Detection*, volume 1907, pages 197–216. Springer-Verlag Berlin Heidelberg, 2000.

[67] Michael Meier, Niels Bischof, and Thomas Holz. SHEDEL — A Simple Hierarchical Event Description Language for Specifying Attack Signatures. In *Security in the Information Society*, pages 559–571. International Federation for Information Processing, 2002.

[68] Michael Meier. A Model for the Semantics of Attack Signatures in Misuse Detection Systems. In *Information Security*, pages 158–169. Springer Berlin Heidelberg, 2004.

[69] Ulrich Flegel and Michael Meier. Modeling and Describing Misuse Scenarios Using Signature-Nets and Event Description Language. *it - Information Technology*, 54(2):71–81, April 2012.

[70] David Jaeger, Martin Ussath, Feng Cheng, and Christoph Meinel. Multi-Step Attack Pattern Detection on Normalized Event Logs. In *Proceedings of the 2nd IEEE International Conference on Cyber Security and Cloud Computing (CSCloud'15)*, pages 390–398. IEEE Computer Society, November 2015.

[71] P. Arun Raj Kumar and S. Selvakumar. Distributed denial of service attack detection using an ensemble of neural classifier. *Computer Communications*, 34(11):1328–1341, July 2011.

[72] Carlos A. Catania and Carlos García Garino. Automatic network intrusion detection: Current techniques and open issues. *Computers & Electrical Engineering*, 38(5):1062–1072, September 2012.

[73] Pavel Laskov, Patrick Düssel, Christin Schäfer, and Konrad Rieck. Learning Intrusion Detection: Supervised or Unsupervised? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 3617 LNCS, pages 50–57. Springer-Verlag Berlin Heidelberg, 2005.

[74] Sumeet Dua and Xian Du. *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications, Boston, MA, USA, 2011.

[75] Srinivas Mukkamala, Guadalupe Janoski, and Andrew Sung. Intrusion detection: support vector machines and neural networks. In *Proceedings of the IEEE international joint conference on neural networks (ANNIE)*, pages 1702–1707, 2002.

[76] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. *IEEE Symposium on Computational Intelligence for Security and Defense Applications, CISDA 2009*, 2009.

[77] David Jaeger, Amir Azodi, Feng Cheng, and Christoph Meinel. Normalizing Security Events with a Hierarchical Knowledge Base. In *Information Security Theory and Practice*, volume 9311 of *Lecture Notes in Computer Science*, pages 237–248. Springer International Publishing, Cham, 2015.

[78] Steven Noel, Duminda Wijesekera, and Charles Youman. Modern Intrusion Detection, Data Mining, and Degrees of Attack Guilt. In *Applications of Data Mining in Computer Security*, pages 1–31. Springer, Boston, MA, 2002.

[79] Ali A. Ghorbani, Wei Lu, and Mahbod Tavallaee. *Network Intrusion Detection and Prevention*, volume 47 of *Advances in Information Security*. Springer US, Boston, MA, 2010.

[80] Gisung Kim, Seungmin Lee, and Sehun Kim. A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, 41(4 PART 2):1690–1700, 2014.

[81] Daniel T. Larose. *Discovering Knowledge in Data*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2005.

[82] Yoav Benjamini and Moshe Leshno. Statistical Methods for Data Mining. In *Data Mining and Knowledge Discovery Handbook*, pages 565–587. Springer-Verlag, New York, 2010.

[83] Stephen Marsland. Machine Learning: An Algorithmic Perspective, 2009. `http://books.google.com/books?id=n66O8a4SWGEC`. Visited on 03-05-2018.

[84] Charu C. Aggarwal. Supervised Outlier Detection. In *Outlier Analysis*, pages 169–198. Springer New York, 2013.

[85] Gang Wang, Jinxing Hao, Jian Ma, and Lihua Huang. A new approach to intrusion detection using Artificial Neural Networks and fuzzy clustering. *Expert Systems with Applications*, 37(9):6225–6232, September 2010.

[86] Mradul Dhakar and Akhilesh Tiwari. A Novel Data Mining based Hybrid Intrusion Detection Framework. *Journal of Information and Computing Science*, 9(1):37–48, 2014.

[87] Yinhui Li, Jingbo Xia, Silan Zhang, Jiakai Yan, Xiaochuan Ai, and Kuobin Dai. An efficient intrusion detection system based on support vector machines and gradually feature removal method. *Expert Systems with Applications*, 2012.

[88] Dewan Md. Farid, Nouria Harbi, and Mohammad Zahidur Rahman. Combining Naive Bayes and Decision Tree for Adaptive Intrusion Detection. *International journal of Network Security & Its Applications*, 2(2):12–25, April 2010.

[89] Jasmin Kevric, Samed Jukic, and Abdulhamit Subasi. An effective combining classifier approach using tree algorithms for network intrusion detection. *Neural Computing and Applications*, 28(S1):1051–1058, December 2017.

[90] Sandhya Peddabachigari, Ajith Abraham, Crina Grosan, and Johnson Thomas. Modeling intrusion detection system using hybrid intelligent systems. *Journal of Network and Computer Applications*, 30(1):114–132, January 2007.

[91] Wei Li. Using genetic algorithm for network intrusion detection. *Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference, Kansas City, Kansas*, pages 24–27, 2004.

[92] Samaneh Rastegari, Philip Hingston, and Chiou-Peng Lam. Evolving statistical rulesets for network intrusion detection. *Applied Soft Computing*, 33:348–359, August 2015.

[93] Himadri Chauhan, Vipin Kumar, Sumit Pundir, and Emmanuel S Pilli. A Comparative Study of Classification Techniques for Intrusion Detection. In *2013 International Symposium on Computational and Business Intelligence*, pages 40–43. IEEE, August 2013.

[94] Evangelos E. Papalexakis, Alex Beutel, and Peter Steenkiste. Network Anomaly Detection Using Co-clustering. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 403–410. IEEE, August 2012.

[95] Bimal Viswanath, Ahmad M. Bashir, Mark Crovella, Saikat Guha, Krishna P. Gummadi, Balachander Krishnamurthy, and Alan Mislove. Towards Detecting Anomalous User Behavior in Online Social Networks. In *23rd USENIX Security Symposium (USENIX Security)*, April 2014.

[96] Nong Ye, Yebin Zhang, and C.M. Borror. Robustness of the Markov-Chain Model for Cyber-Attack Detection. *IEEE Transactions on Reliability*, 53(1):116–123, March 2004.

[97] Xiejun Ni, Daojing He, Sammy Chan, and Farooq Ahmad. Network Anomaly Detection Using Unsupervised Feature Selection and Density Peak Clustering. In *Applied Cryptography and Network Security*, volume 2846, pages 212–227. Springer International Publishing Switzerland, 2016.

[98] Sanjay Chawla and Aristides Gionis. k-means–: A Unified Approach to Clustering and Outlier Detection. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 189–197. SIAM, 2013.

[99] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, and Koji Nakao. Toward a more practical unsupervised anomaly detection system. *Information Sciences*, 231:4–14, 2013.

[100] Jungsuk Song, Hiroki Takakura, Yasuo Okabe, and Yongjin Kwon. Unsupervised Anomaly Detection Based on Clustering and Multiple One-Class SVM. *IEICE Transactions on Communications*, E92-B(6):1981–1990, 2009.

[101] M. F. Augusteijn and B. A. Folkert. Neural network classification and novelty detection. *International Journal of Remote Sensing*, 23(14):2891–2902, 2002.

[102] Markos Markou and Sameer Singh. Novelty detection: a review - part 2: neural network based approaches. *Signal Processing*, 83(12):2499–2521, 2003.

[103] Marco A. F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, June 2014.

[104] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. Outlier Detection Using Replicator Neural Networks. In *Data Warehousing and Knowledge Discovery*, volume 4654 of *Lecture Notes in Computer Science*, pages 170–180. Springer Berlin Heidelberg, December 2002.

[105] J.Z. Lei and a. Ghorbani. Network intrusion detection using an improved competitive learning neural network. *Proceedings. Second Annual Conference on Communication Networks and Services Research, 2004.*, pages 190–197, 2004.

[106] Sanjay Chawla and Aristides Gionis. k -means–: A unified approach to clustering and outlier detection. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 189–197. Society for Industrial and Applied Mathematics, Philadelphia, PA, May 2013.

[107] Angela D. Orebaugh, Simon Biles, and Jacob Babbin. *Snort Cookbook - Solutions and Examples for Snort Administrators*. O'Reilly Media, Inc., January 2005.

[108] About Event Tracing. `https://msdn.microsoft.com/en-us/library/windows/desktop/aa363668(v=vs.85).aspx`. Visited on 03-05-2018.

[109] D Casey. Turning log files into a security asset. *Network Security*, 2008(2):4–7, 2008.

[110] Liu Yang, Pratyusa Manadhata, William Horne, Prasad Rao, and Vinod Ganapathy. Fast submatch extraction using OBDDs. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, ANCS '12, pages 163–174, New York, NY, USA, 2012. ACM.

[111] Common Event Expression White Paper, June 2008. `https://cee.mitre.org/docs/Common_Event_Expression_White_Paper_June_2008.pdf`. Visited on 03-05-2018.

[112] H. Debar, D. Curry, and B. Feinstein. RFC 4765 - The Intrusion Detection Message Exchange Format (IDMEF). *Network Working Group, IETF*, March 2007.

[113] BlackStratus' LOG Storm. `http://www.blackstratus.com/enterprise/supported-technologies/log-storm/`. Visited on 03-05-2018.

[114] Sawmill log analysis tool. `http://www.sawmill.net/`. Visited on 03-05-2018.

[115] Open Source Host-based Intrusion Detection System. `http://www.ossec.net/`. Visited on 03-05-2018.

[116] Alvaro A. Cardenas, Pratyusa K. Manadhata, and Sreeranga P. Rajan. Big Data Analytics for Security. *IEEE Security & Privacy*, 11(6):74–76, 2013.

[117] Micro Focus ArcSight Data Platform. `https://software.microfocus.com/fr-ca/software/siem-data-collection-log-management-platform`. Visited on 03-05-2018.

[118] HPE Security ArcSight Common Event Format. Hewlett Packard Enterprise, May 2016. `https://www.secef.net/wp-content/uploads/sites/10/2017/04/CommonEventFormatv23.pdf`. Visited on 03-05-2018.

[119] T. Takahashi, K. Landfield, and Y. Kadobayashi. RFC 7203 - An Incident Object Description Exchange Format (IODEF) Extension for Structured Cybersecurity Information. *Internet Engineering Task Force (IETF)*, 2014.

[120] PowerShell Get-WinEvent XML Madness: Getting details from event logs. `http://blogs.technet.com/b/ashleymcglone/archive/2013/08/28/powershell-get-winevent-xml-madness-getting-details-from-event-logs.aspx`. Visited on 03-05-2018.

[121] Andrey Sapegin, Marian Gawron, David Jaeger, Feng Cheng, and Christoph Meinel. High-Speed Security Analytics Powered by In-Memory Machine Learning Engine. In *2015 14th International Symposium on Parallel and Distributed Computing*, pages 74–81. IEEE, June 2015.

[122] Andrey Sapegin, Marian Gawron, David Jaeger, Feng Cheng, and Christoph Meinel. Evaluation of in-memory storage engine for machine learning analysis of security events. *Concurrency and Computation: Practice and Experience*, 29(2):e3800, 2016.

[123] Andrey Sapegin, David Jaeger, Feng Cheng, and Christoph Meinel. Towards a system for complex analysis of security events in large-scale networks. *Computers & Security*, 67:16–34, June 2017.

[124] Alessandro D'Alconzo, Pere Barlet-Rosb, Kensuke Fukudac, and David Choffnes. Machine learning, data mining and Big Data frameworks for network monitoring and troubleshooting. *Computer Networks*, 107(1):1–4, 2016.

[125] David Jaeger, Andrey Sapegin, Martin Ussath, Feng Cheng, and Christoph Meinel. Parallel and distributed normalization of security events for instant attack analysis. In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, pages 1–8. IEEE, December 2015.

[126] Binary R server. `http://rforge.net/Rserve/`. Visited on 03-05-2018.

[127] Anton Chuvakin. Scan 34 - Analyze real honeynet logs for attacks and activity. `http://old.honeynet.org/scans/scan34/`. Visited on 03-05-2018, February 2005.

[128] Matt Richard, Michael Ligh, Andy Magnusson, Syd Seale, and Kelly Standridge. Project Honeynet Scan of the Month 34, May 2005. `http://old.honeynet.org/scans/scan34/sols/1/index.html`. Visited on 03-05-2018.

[129] Christine Kronberg and Agleia Freeworld. Analysis of the logfiles given in SotM34, 2005. http://old.honeynet.org/scans/scan34/sols/2/proc.pdf. Visited on 03-05-2018.

[130] J. Klensin. RFC 5321 - Simple Mail Transfer Protocol. *Network Working Group, IETF*, 2008.

[131] SAP Predictive Analysis. https://www.sap.com/products/predictive-analytics.html. Visited on 03-05-2018.

[132] syslog-ng - Open Source log management solution. https://syslog-ng.org/. Visited on 03-05-2018.

[133] Splunk Enterprise. http://splunk.com/. Visited on 03-05-2018.

[134] THC-Hydra, network logon cracker. https://www.thc.org/thc-hydra/. Visited on 03-05-2018.

[135] Remote Desktop Protocol. http://msdn.microsoft.com/en-us/library/aa383015.aspx. Visited on 03-05-2018.

[136] J. Sermersheim. RFC 4511 - Lightweight Directory Access Protocol (LDAP): The Protocol. *Network Working Group, IETF*, 2006.

[137] Charu Chaubal. The Architecture of VMware ESXi. White Paper, 2008. https://microage.com/wp-content/uploads/2016/02/ESXi_architecture.pdf. Visited on 03-05-2018.

[138] SAP HANA Studio Installation and Update Guide. https://help.sap.com/viewer/a2a49126a5c546a9864aae22c05c3d0e/2.0.01/en-US. Visited on 03-05-2018.

[139] RPostgreSQL: R interface to the PostgreSQL database system. https://cran.r-project.org/web/packages/RPostgreSQL/index.html. Visited on 03-05-2018.

[140] Rik Warren, Robert E. Smith, and Anne K. Cybenko. Use of Mahalanobis Distance for Detecting Outliers and Outlier Clusters in Markedly Non-Normal Data: A Vehicular Traffic Example, June 2011.

[141] Catherine A. Sugar and Gareth M. James. Finding the Number of Clusters in a Dataset. *Journal of the American Statistical Association*, 98(463):750–763, September 2003.

[142] D. Pelleg and A.W. Moore. X-means: Extending K-means with efficient estimation of the number of clusters. In *Proceedings of the 17th International Conference on Machine Learning*, pages 727–734, November 2000.

[143] Greg Hamerly and Charles Elkan. Learning the k in k-means. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*, NIPS'03, pages 281–288, Cambridge, MA, USA, 2003. MIT Press.

[144] Matrix: Sparse and Dense Matrix Classes and Methods. `https://cran.r-project.org/web/packages/Matrix/index.html`. Visited on 03-05-2018.

[145] skmeans: Spherical k-Means Clustering. `https://cran.r-project.org/web/packages/skmeans/index.html`. Visited on 03-05-2018.

[146] KDD Cup 1999 Data. `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Visited on 03-05-2018.

[147] Salvador Garcia, Julián Luengo, José Antonio Sáez, Victoria López, and Francisco Herrera. A Survey of Discretization Techniques: Taxonomy and Empirical Analysis in Supervised Learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):734–750, April 2013.

[148] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. *Machine Learning: Proceedings of the Twelfth International Conference*, 54(2):194–202, 1995.

[149] Stan Salvador and Philip Chan. Determining the number of clusters/segments in hierarchical clustering/segmentation algorithms. In *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 576–584. IEEE, 2004.

[150] Arthur Zimek, Matthew Gaudet, Ricardo J.G.B. Campello, and Jörg Sander. Subsampling for efficient and effective unsupervised outlier detection ensembles. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, pages 428–436, New York, NY, USA, 2013. ACM.

[151] Hans-Peter Kriegel, Peer Kroger, Erich Schubert, and Arthur Zimek. Interpreting and Unifying Outlier Scores. In *Proceedings of the 2011 SIAM International Conference on Data Mining*, pages 13–24. Society for Industrial and Applied Mathematics, Philadelphia, PA, April 2011.

[152] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PLOS ONE*, 11(4):1–31, April 2016.

[153] Mei Ling Shyu, Shu Ching Chen, Kanoksri Sarinnapakorn, and Liwu Chang. Principal component-based anomaly detection scheme. *Studies in Computational Intelligence*, 9:311–329, 2006.

[154] Wei Wang and Roberto Battiti. Identifying intrusions in computer networks with principal component analysis. In *First International Conference on Availability, Reliability and Security (ARES'06)*. IEEE, April 2006.

[155] Jonathan J. Davis and Andrew J. Clark. Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security*, 30(6-7):353–375, September 2011.

[156] Michael J. Chapple, Nitesh Chawla, and Aaron Striegel. Authentication anomaly detection: A case study on a virtual private network. In *Proceedings of the 3rd Annual ACM Workshop on Mining Network Data*, MineNet '07, pages 17–22, New York, NY, USA, 2007. ACM.

[157] Rapid Miner. Predictive Analytics Platform. `https://rapidminer.com/`. Visited on 03-05-2018.

[158] Li Shengqiao, Alina Beygelzimer, Sham Kakadet, John Langford, Sunil Arya, and David Mount. Package 'FNN', 2015. `https://cran.r-project.org/web/packages/FNN/FNN.pdf`. Visited on 03-05-2018.

[159] Andrey Sapegin, Aragats Amirkhanyan, Marian Gawron, Feng Cheng, and Christoph Meinel. Poisson-based anomaly detection for identifying malicious user behaviour. In *Lecture Notes in Computer Science*, volume 9395, pages 134–150, 2015.

[160] Susanta Nanda and Tzi Cker Chiueh. Execution trace-driven automated attack signature generation. In *Proceedings - Annual Computer Security Applications Conference, ACSAC*, pages 195–204, 2008.

[161] P García-Teodoro, J Díaz-Verdejo, G Maciá-Fernández, and E Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1-2):18–28, February 2009.

[162] Karen A. Scarfone and Peter M. Mell. Guide to Intrusion Detection and Prevention Systems (IDPS). National Insitute of Standards and Technology, 2007. `https://www.nist.gov/publications/guide-intrusion-detection-and-prevention-systems-idps`. Visited on 03-05-2018.

[163] Alexander Ihler, Jon Hutchins, and Padhraic Smyth. Adaptive event detection with time-varying poisson processes. In *Proceedings of the 12th ACM*

SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06, pages 207–216, New York, NY, USA, 2006. ACM.

[164] Robin Berthier, Will Rhee, Michael Bailey, Partha Pal, Farnam Jahanian, and William H Sanders. Safeguarding academic accounts and resources with the University Credential Abuse Auditing System. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, pages 1–8. IEEE, June 2012.

[165] Sang Hyun Oh and Won Suk Lee. An anomaly intrusion detection method by clustering normal user behavior. *Computers & Security*, 22(7):596–612, October 2003.

[166] Simon Liu and Rick Kuhn. Data Loss Prevention. *IT Professional*, 12(2):10–13, March 2010.

[167] A Shabtai, Y Elovici, and L Rokach. *A survey of data leakage detection and prevention solutions*. Springer, 2012.

[168] Rahul Khanna and Huaping Liu. System approach to intrusion detection using hidden markov model. In *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*, IWCMC '06, pages 349–354, New York, NY, USA, 2006. ACM.

[169] Haakon Ringberg, Augustin Soule, Jennifer Rexford, and Christophe Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '07, pages 109–120, New York, NY, USA, 2007. ACM.

[170] Wun-Hwa Chen, Sheng-Hsun Hsu, and Hwang-Pin Shen. Application of SVM and ANN for intrusion detection. *Computers & Operations Research*, 32(10):2617–2634, October 2005.

[171] You Chen, Yang Li, Xue-qi Cheng, and Li Guo. Survey and Taxonomy of Feature Selection Algorithms in Intrusion Detection System. In *Information Security and Cryptology*, volume 4318 of *Lecture Notes in Computer Science*, pages 153–167. Springer Berlin Heidelberg, 2006.

[172] Virtual Network Computing. `http://www.hep.phy.cam.ac.uk/vnc_docs/index.html`. Visited on 03-05-2018.

[173] Python Imaging Library. `http://www.pythonware.com/products/pil/`. Visited on 03-05-2018.

[174] Aragats Amirkhanyan, Andrey Sapegin, Marian Gawron, Feng Cheng, and Christoph Meinel. Simulation user behavior on a security testbed using user behavior states graph. In *Proceedings of the 8th International Conference on Security of Information and Networks - SIN '15*, pages 217–223, New York, NY, USA, 2015. ACM Press.

[175] Windows PowerShell. `https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting`. Visited on 03-05-2018.

[176] Event 4624 null sid - Repeated security log. `http://www.morgantechspace.com/2013/10/event-4624-null-sid-repeated-security.html`. Visited on 03-05-2018.

[177] Hongliang Yu, Dongdong Zheng, Ben Y. Zhao, and Weimin Zheng. Understanding User Behavior in Large-scale Video-on-demand Systems. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 333–344, New York, NY, USA, 2006. ACM.

[178] Balakrishnan Chandrasekaran. Survey of network traffic models. *Washington University in St. Louis CSE*, pages 1–8, 2009.

[179] Umar Javed, Italo Cunha, David Choffnes, Ethan Katz-Bassett, Thomas Anderson, and Arvind Krishnamurthy. PoiRoot: Investigating the Root Cause of Interdomain Path Changes. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, page 183, New York, NY, USA, 2013. ACM Press.

[180] Kevin Butler, T.R. Farley, Patrick McDaniel, and Jennifer Rexford. A Survey of BGP Security Issues and Solutions. *Proceedings of the IEEE*, 98(1):100–122, January 2010.

[181] Andrey Sapegin and Steve Uhlig. On the extent of correlation in BGP updates in the Internet and what it tells us about locality of BGP routing events. *Computer Communications*, 36(15-16):1592–1605, September 2013.

[182] Andrey Sapegin, Feng Cheng, and Christoph Meinel. Catch the Spike: On the Locality of Individual BGP Update Bursts. In *2013 IEEE 9th International Conference on Mobile Ad-hoc and Sensor Networks*, pages 78–83. IEEE, December 2013.

[183] Pejman Najafi, Andrey Sapegin, Feng Cheng, and Christoph Meinel. Guilt-by-Association: Detecting Malicious Entities via Graph Mining. In *Security*

and Privacy in Communication Networks, pages 88–107. Springer International Publishing, 2018.

[184] SAP HANA Graph Reference. https://help.sap.com/doc/21574acf46fe45a8ae9def213f2c4d9e/2.0.00/en-us/sap_hana_graph_reference_en.pdf. Visited on 03-05-2018.

[185] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, pages 137–149, 2004.

[186] Apache Hadoop. https://hadoop.apache.org/. Visited on 03-05-2018.

[187] Apache Spark - Lightning-fast luster computing. https://spark.apache.org/. Visited on 03-05-2018.

[188] SAP Vora. https://www.sap.com/product/data-mgmt/hana-vora-hadoop.html. Visited on 03-05-2018.

[189] Using the SAP Vora Spark Extension. https://help.sap.com/doc/0991e2320f5940d988ed32b995d28a44/2.1/en-US/b289656d890f410698ff7e82871f6451.html. Visited on 03-05-2018.

[190] Apache Hadoop HDFS. https://hortonworks.com/apache/hdfs/. Visited on 03-05-2018.