
Enabling Big Data Security Analytics for Advanced Network Attack Detection

DISSERTATION

ZUR ERLANGUNG DES AKADEMISCHEN GRADES

“DOKTOR-INGENIEUR”

(DR.-ING.)

IN DER WISSENSCHAFTSDISZIPLIN

DIGITAL ENGINEERING

EINGEREICHT AN DER

FAKULTÄT FÜR DIGITAL ENGINEERING

DER UNIVERSITÄT POTSDAM

von
David Jaeger

Betreuer:
Prof. Dr. Christoph MEINEL

Potsdam, 5. Dezember 2018

This work is licensed under a Creative Commons License:
Attribution – Non Commercial – Share Alike 4.0 International.
This does not apply to quoted content from other authors.
To view a copy of this license visit
<https://creativecommons.org/licenses/by-nc-sa/4.0/>

Prüfungskommission:

Vorsitzender: Prof. Dr. Andreas Polze	(Hasso-Plattner-Institut)
1. Gutachter: Prof. Dr. Christoph Meinel	(Hasso-Plattner-Institut)
2. Gutachter: Prof. Dr. (TU NN) Norbert Pohlmann	(Westfälische Hochschule)
3. Gutachter: Prof. Dr. Michael Meier	(Universität Bonn)
5. Mitglied: Prof. Dr. Robert Hirschfeld	(Hasso-Plattner-Institut)
6. Mitglied Prof. Dr. Felix Naumann	(Hasso-Plattner-Institut)

Tag der Disputation: 13. September 2019

Published online at the
Institutional Repository of the University of Potsdam:
<https://doi.org/10.25932/publishup-43571>
<https://nbn-resolving.org/urn:nbn:de:kobv:517-opus4-435713>

Diese Arbeit ist meiner Mutter gewidmet.

Acknowledgments

I would like to thank my supervisor Prof. Dr. Christoph Meinel for his guidance through my research and his constant support throughout my Ph.D. studies. I am also grateful to Dr. Feng Cheng for his valuable guidance and always providing helpful advice as my mentor.

Further, I would also like to thank my dear colleagues and friends - Martin Usath, Marian Gawron, Pejman Najafi, Andrey Sapegin, and Chris Pelchen for the fruitful discussions, inspirational brainstorming, and the great teamwork. Additionally, I thank all members of the Security Engineering team that accompanied me on this journey at the HPI.

A special thank goes to my parents, Maureen and Bernd, for their loving support and inspiring me to always take one step further. Finally, I also thank my loving partner Linda for always supporting me and encouraging me throughout the last phase of this work.

Abstract

The last years have shown an increasing sophistication of attacks against enterprises. Traditional security solutions like firewalls, anti-virus systems and generally Intrusion Detection Systems (IDSs) are no longer sufficient to protect an enterprise against these advanced attacks. One popular approach to tackle this issue is to collect and analyze events generated across the IT landscape of an enterprise. This task is achieved by the utilization of Security Information and Event Management (SIEM) systems. However, the majority of the currently existing SIEM solutions is not capable of handling the massive volume of data and the diversity of event representations. Even if these solutions can collect the data at a central place, they are neither able to extract all relevant information from the events nor correlate events across various sources. Hence, only rather simple attacks are detected, whereas complex attacks, consisting of multiple stages, remain undetected. Undoubtedly, security operators of large enterprises are faced with a typical Big Data problem.

In this thesis, we propose and implement a prototypical SIEM system named Real-Time Event Analysis and Monitoring System (REAMS) that addresses the Big Data challenges of event data with common paradigms, such as data normalization, multi-threading, in-memory storage, and distributed processing. In particular, a mostly stream-based event processing workflow is proposed that collects, normalizes, persists and analyzes events in near real-time. In this regard, we have made various contributions in the SIEM context. First, we propose a high-performance normalization algorithm that is highly parallelized across threads and distributed across nodes. Second, we are persisting into an in-memory database for fast querying and correlation in the context of attack detection. Third, we propose various analysis layers, such as anomaly- and signature-based detection, that run on top of the normalized and correlated events. As a result, we demonstrate our capabilities to detect previously known as well as unknown attack patterns. Lastly, we have investigated the integration of cyber threat intelligence (CTI) into the analytical process, for instance, for correlating monitored user accounts with previously collected public identity leaks to identify possible compromised user accounts.

In summary, we show that a SIEM system can indeed monitor a large enterprise environment with a massive load of incoming events. As a result, complex attacks spanning across the whole network can be uncovered and mitigated, which is an advancement in comparison to existing SIEM systems on the market.

Zusammenfassung

Die letzten Jahre haben gezeigt, dass die Komplexität von Angriffen auf Unternehmensnetzwerke stetig zunimmt. Herkömmliche Sicherheitslösungen, wie Firewalls, Antivirus-Programme oder generell Intrusion Detection Systeme (IDS), sind nicht mehr ausreichend, um Unternehmen vor solch ausgefeilten Angriffen zu schützen. Ein verbreiteter Lösungsansatz für dieses Problem ist das Sammeln und Analysieren von Ereignissen innerhalb des betroffenen Unternehmensnetzwerks mittels Security Information and Event Management (SIEM) Systemen. Die Mehrheit der derzeitigen SIEM-Lösungen auf dem Markt ist allerdings nicht in der Lage, das riesige Datenvolumen und die Vielfalt der Ereignisdarstellungen zu bewältigen. Auch wenn diese Lösungen die Daten an einem zentralen Ort sammeln können, können sie weder alle relevanten Informationen aus den Ereignissen extrahieren noch diese über verschiedene Quellen hinweg korrelieren. Aktuell werden daher nur relativ einfache Angriffe erkannt, während komplexe mehrstufige Angriffe unentdeckt bleiben. Zweifellos stehen Sicherheitsverantwortliche großer Unternehmen einem typischen Big Data-Problem gegenüber.

In dieser Arbeit wird ein prototypisches SIEM-System vorgeschlagen und implementiert, welches den Big Data-Anforderungen von Ereignisdaten mit gängigen Paradigmen, wie Datennormalisierung, Multithreading, In-Memory-Speicherung und verteilter Verarbeitung begegnet. Insbesondere wird ein größtenteils stream-basierter Workflow für die Ereignisverarbeitung vorgeschlagen, der Ereignisse in nahezu Echtzeit erfasst, normalisiert, persistiert und analysiert. In diesem Zusammenhang haben wir verschiedene Beiträge im SIEM-Kontext geleistet. Erstens schlagen wir einen Algorithmus für die Hochleistungsnormalisierung vor, der, über Threads hinweg, hochgradig parallelisiert und auf Knoten verteilt ist. Zweitens persistieren wir in eine In-Memory-Datenbank, um im Rahmen der Angriffserkennung eine schnelle Abfrage und Korrelation von Ereignissen zu ermöglichen. Drittens schlagen wir verschiedene Analyseansätze, wie beispielsweise die anomalie- und musterbasierte Erkennung, vor, die auf normalisierten und korrelierten Ereignissen basieren. Damit können wir bereits bekannte als auch bisher unbekannte Arten von Angriffen erkennen. Zuletzt haben wir die Integration von sogenannter Cyber Threat Intelligence (CTI) in den Analyseprozess untersucht. Als Beispiel erfassen wir veröffentlichte Identitätsdiebstähle von großen Dienstleistern, um Nutzerkonten zu identifizieren, die möglicherweise in nächster Zeit durch den Missbrauch verloren gegangener Zugangsdaten kompromittiert werden könnten.

Zusammenfassend zeigen wir, dass ein SIEM-System tatsächlich ein großes Unternehmensnetzwerk mit einer massiven Menge an eingehenden Ereignissen überwachen kann. Dadurch können komplexe Angriffe, die sich über das gesamte Netzwerk erstrecken, aufgedeckt und abgewehrt werden. Dies ist ein Fortschritt gegenüber den auf dem Markt vorhandenen SIEM-Systemen.

Contents

1	Introduction	1
1.1	Problem Statement	3
1.2	Thesis Contributions	4
1.3	Thesis Organization	7
I	Big Data Terminology and Concepts	9
2	The Anatomy of Big Data	11
2.1	Definition of Big Data	12
2.1.1	The 7 V's	13
2.2	Data Representation	15
2.2.1	Data Structure	15
2.2.2	Common Data Types	16
2.3	Challenges of Big Data	17
3	Big Data Processing	19
3.1	Data Processing Paradigms	19
3.1.1	Aiming for Scalability	19
3.1.2	Methods of Performance Scaling	20
3.1.3	Batch- and Stream-Processing	22
3.1.4	Data Immutability	24
3.1.5	Processing Reliability	26
3.2	The Big Data Workflow	27
3.2.1	Known Processing Architectures	27
3.2.2	Traditional ETL/ELT-Process	30
3.2.3	Our Processing Workflow	31
4	Tools and Libraries for Big Data	35
4.1	Parallel Processing	35
4.1.1	Inter-Process Communication (IPC)	35

CONTENTS

4.1.2	Distributed Processing Frameworks	37
4.2	Persistence	40
4.3	Conclusion	42
II	Big Data Security Analytics on Event Data	43
5	Approaching Enterprise Sec. with Big Data Proc.	45
5.1	Current State of Attack Detection	46
5.1.1	Attack Categorization	46
5.1.2	Attack Detection Systems	48
5.1.3	Detection Methods	49
5.1.4	Data Sources for Attack Detection	50
5.2	Enterprise Security as Big Data Challenge	52
5.3	A Big Data SIEM for Enterprise Security	55
5.3.1	System Approaches	55
5.3.2	System Architecture	58
5.4	Conclusion	60
6	Stream-based Processing of Big Event Log Data	61
6.1	Data Collection	62
6.1.1	Types of Event Log Sources	63
6.1.2	Existing Methods of Collection	63
6.1.3	Proposed Log Collection	64
6.2	Basic Normalization of Raw Events	64
6.2.1	Event Representation	64
6.2.2	Event Formats	67
6.2.3	Existing Event Normalization Methods	70
6.2.4	A Customized Rule-Matching Approach	71
6.3	Enrichment	77
6.3.1	Intra-Event Enrichment	77
6.3.2	Inter-Event Enrichment	79
6.3.3	Extra-Event Enrichment	81
6.4	Persistence	82
6.4.1	Storage Systems	83
6.4.2	Mapping of the Event Model to a Schema	85
6.5	Conclusion	86
7	Improving the Speed of Event Normalization	87
7.1	Normalization Performance Factors	88
7.2	Application of Regular Expressions	88

CONTENTS

7.2.1	Finite Automatons	89
7.2.2	Implementation Issues of Finite Automatons	89
7.2.3	Evaluating Regex Implementations	90
7.3	Unified Event Assignments	92
7.3.1	Property Access Approaches	92
7.3.2	Object Pooling and State Resets	94
7.3.3	Evaluation	94
7.4	Rule Organization	95
7.4.1	Homogeneity and Structure of Real-World Events	96
7.4.2	Organization of the Knowledge Base	97
7.4.3	Matching with a Hierarchical knowledge base (KB)	99
7.5	Searching Rules Efficiently	102
7.5.1	Rule Indexing	102
7.5.2	Candidate Rule Caching	103
7.5.3	Rule Selector	103
7.5.4	Priority Lists	103
7.5.5	Evaluation	104
7.6	Conclusion	105
8	Scaling Event Stream Processing	107
8.1	Vertical Parallelization by Multi-Threading	107
8.1.1	Locking Implementation with a Blocking Queue	109
8.1.2	Lock-Free Implementation with the Disruptor	110
8.1.3	Limitation of Scaling	111
8.2	Horizontal Parallelization by Distributed Processing Nodes	112
8.2.1	Data Exchange	114
8.2.2	Processing Reliability	117
8.2.3	Single-Node Performance of Distributed Frameworks	118
8.2.4	Multi-Node Performance using the Disruptor Pattern	119
8.3	Applying the Parallelization to Persistence	120
8.3.1	Tuning the Database Access for Parallelization	121
8.3.2	Distribution to Multiple Nodes	124
8.4	Conclusion	126
9	Advanced Attack Analysis on Normalized Events	129
9.1	The Role of Cyber Threat Intelligence (CTI)	131
9.2	Simple Data Correlation	133
9.2.1	Correlation Queries on the Database	133
9.2.2	Correlation Queries on HDFS with Spark	134
9.2.3	Evaluation	135
9.3	Single-Step Signature Detection	135

CONTENTS

9.3.1	Our Signature Model	136
9.3.2	Normalization	137
9.3.3	Matching	141
9.3.4	Evaluation	142
9.4	Multi-Step Signature Detection	142
9.4.1	Model Based on EDL	143
9.4.2	Signature Sources	145
9.4.3	Matching and the SAM Implementation	146
9.4.4	Evaluation	150
9.5	Machine Learning and Anomaly Detection	151
9.5.1	Implemented Algorithms	151
9.5.2	Execution Environment	152
9.6	Conclusion	153
10	Using Identity Leak Data as Threat Intelligence	155
10.1	Public Identity Leaks	157
10.1.1	Quantity	157
10.1.2	Data Distribution	158
10.1.3	Data Origin	160
10.1.4	Representation	161
10.2	Leak Processing Workflow	163
10.2.1	Data Collection	164
10.2.2	Identity Extraction	165
10.2.3	Hash Recognition	170
10.2.4	Subscriber Filter	173
10.3	Leveraging Leak-Related Threat Intelligence	175
10.3.1	Alerting	175
10.3.2	Signature Derivation	176
10.4	Conclusion	176
III	Case Study	179
11	Real-Time Event Analysis and Monitoring System (REAMS)	181
11.1	Real-World Deployment	181
11.1.1	Agents	182
11.1.2	Core Server	183
11.1.3	Graphical User Interfaces	184
11.2	Practical Use-Cases	188
11.2.1	HPI Infrastructure	189
11.2.2	Event Data of Big International Companies	190

CONTENTS

11.3 Conclusion	191
12 Identity Leak Checker (ILC)	193
12.1 Real-World Deployment	193
12.1.1 Backend	193
12.1.2 Web Interface	194
12.1.3 Client	195
12.2 Practical Use-Cases	195
12.2.1 Identity Leak Checker Service	196
12.2.2 Monitoring for Federal States and Companies	197
12.3 Conclusion	198
13 Conclusion	199
Bibliography	XIX

List of Figures

3.1	Types of scaling	20
3.2	Lambda Architecture	28
3.3	Kappa Architecture	30
3.4	ETL-Process	30
3.5	Processing workflow used in this work	32
4.1	The blocking queue concept	36
4.2	The disruptor concept	37
5.1	Example of a more complex multi-step attack	47
5.2	REAMS architecture being derived from our Big Data workflow	59
6.1	Stream-based part of our SIEM described in this chapter	62
6.2	Levels of event sources	63
6.3	Event Structure of the Object Log Format (OLF)	70
6.4	Process of a Rule-based Deep Normalization	76
6.5	Extracting missing time information from related logs	79
6.6	Idea of the master table model	85
7.1	Slot concept as used for assigning event fields to an OLF event	93
7.2	Normalization throughput with different methods of field assignments	95
7.3	Two log events produced by Snort with a similar structure	97
7.4	Hierarchy of event formats on the example of Snort events	98
7.5	The matching of the two events with a flat KB	98
7.6	The matching of the two events with a hierarchical KB	99
7.7	Comparison of matching in flat and hierarchical KB	101
7.8	Direct rule selection for a Snort event	103
7.9	Evaluation of various KB optimizations on 69030 Snort events	104
8.1	Vertical parallelization with a message distributor	108
8.2	Normalization performance with a blocking queue on a 16-core machine	109

LIST OF FIGURES

8.3	Thread behavior during normalization	109
8.4	Disruptor normalization with varying buffer sizes, number of worker threads, and CPU cores	111
8.5	Normalization performance with disruptor (buffer size 2^{13}) and blocking queue on a 16 core machine	112
8.6	Horizontal parallelization of normalization with network distribution	113
8.7	Bitset of set fields in an OLF object	116
8.8	Normalization throughput in kevt./s for common distribution frameworks on a single node with 16 threads	119
8.9	Performance of distributed normalization with varying number of normalization nodes	120
8.10	Persistence throughput with multiple nodes	125
9.1	Example of a signature in our model for one of the indicators of APT28137	133
9.2	Relevant indicator information in a Snort rule	138
9.3	Our signature model for the Snort signature	140
9.4	Example of a simple signature for a brute-force attack with /etc/-passwd access in the EDL	144
9.5	Comparison of existing jSAM implementation with extended version	147
9.6	EDL signature graph for login brute-force attacks	150
10.1	Number of leaked identities in the past five years with trend line	157
10.2	Distribution of leak formats for 100 of the largest public leaks	162
10.3	Workflow used for processing leaks	164
10.4	Proposed data model for leaked digital identities	166
10.5	Usage of top passwords	173
10.6	Filtering of the workflow outputs for a specific feed subscriber	174
10.7	EDL signature for finding suspicious logins	177
11.1	REAMS deployment running live in our lab	182
11.2	Command-line interface of REAMS	183
11.3	Dashboard of the REAMS desktop client	184
11.4	Event listing of the REAMS desktop client	185
11.5	Context menu for available analysis methods	186
11.6	Dashboard of the Kibana-based REAMS GUI	187
11.7	Dashboard of the ITOA-based REAMS GUI	188
11.8	Setup of REAMS in our infrastructure	189
11.9	Detecting attacks in our environment	190
12.1	Main query interface of the ILC web service	194
12.2	Native client interface for the ILC with demo data	195
12.3	Number of identities available in the ILC service	196

List of Tables

- 3.1 Comparison of processing modes 24
- 6.1 Identification of time drifts in context logs 80
- 6.2 Derivation of event fields from context events 81
- 7.1 Performance of regular expression implementations 91
- 8.1 Persistence throughput with various optimizations 124
- 9.1 Field mappings from Snort to OLF 139

Chapter 1

Introduction

With the ongoing digitization and datafication [1, p. 15] of our society, more and more parts of our daily lives are moving from the physical into the digital and connected world. Companies, organizations and public authorities are migrating their services, computer systems, and data into the open Internet. Related concepts like e-government, e-health and Industry 4.0 are more prevalent than ever. While this trend bears big opportunities, it also induces significant threats. As data is processed and stored in networks accessible from the Internet, attackers can more easily manipulate or disrupt this data processing as well as steal private or confidential data.

Attackers are performing their malicious activities in ways that are getting more sophisticated and harder to trace. Two large groups of attackers are mostly responsible for this development. On the one hand, there is a relatively low-skilled group of attackers, e.g., *script kiddies* and average cybercriminals that make use of highly sophisticated attack tools created by others. With these tools, they can perform many, mostly automated, advanced single-step attacks on various targets. On the other hand, there is a group of highly-skilled attackers, e.g., nation-states or state-sponsored, that are performing complex multi-step attacks to penetrate selected high-level targets. We call an attack to be single-stepped, if there is only a single attack activity on a single machine, whereas multi-step attacks involve multiple attack activities on potentially multiple machines. The group of highly-skilled attackers uses their knowledge to create sophisticated attack tools for themselves and low-skilled attackers. Both groups of attackers pose a significant threat for large enterprises. The sheer amount of attackers in the first group can find the less protected systems on the Internet. The second group of attackers is able to circumvent security software and can find ways into a network that seems to be fully protected at first glance.

Looking at recent media reports and statistics of security firms and organizations allows the conclusion that the number of sophisticated and large-scale attacks is increasing. According to the *Munich Re* insurance company, 90% of businesses have experienced cyberattacks in the past years [2]. The project *Information is Beauti-*

CHAPTER 1. INTRODUCTION

ful [3] takes a look at data breaches in particular and visualizes some of the worst incidents over the past years. Frighteningly, a considerable number of incidents caused the breach of more than 10 million records of personal identifiable information, such as from Equifax [4], the US Office of Personnel Management (OPM) [5] or JPMorgan Chase [6]. Additionally, a new group of sophisticated attacks, the so-called Advanced Persistent Threats (APTs) [7], is emerging [8]. In contrast to traditional cyberattacks, the employed attack methods are highly advanced, and the attackers are moving very carefully in the target's environment, often for an extended amount of time. These properties make APTs almost impossible to detect with normal intrusion detection mechanisms. Popular cases of APTs are the APT1 attack [9], targeting various international companies, and the attack on the German parliament (Bundestag-Hack) [10].

The current situation of an ever-increasing number of security incidents seems hopeless at first glance. According to the Ponemon Institute [11], the mean time to detect a data breach is about 197 days. However, like crimes in the physical world, attackers leave behind digital traces during their break-ins that can be used for tracking and the mitigation of their attacks. On the one hand, software and hardware sensors on computer systems and networks monitor various activities and produce *events*, especially *security-related events* or *security events*, as manifestations of these activities. Each produced event is written into a so-called *event log*, a list of all previously occurred events. On the other hand, many contextual data sources complement the previously mentioned event data with security-related information. The correlation of these data sources can help to reveal the traces of an attacker. Nevertheless, the processing and analysis of *security-related data*, also referred to as *security data* in the following, is still a challenging task. The reasons for that are manifold, as listed in the following.

1. As networks are growing in size, there is an overwhelming number of sensors that produce an even more overwhelming number of security-relevant data. (Volume)
2. To detect attacks as they are conducted, it is desirable to process all the produced data in near real-time. (Velocity)
3. As data sources are very heterogeneous, the provided data is represented in various formats. The data may be fully structured, partially structured or completely unstructured. Furthermore, the provided information differs significantly in the level of detail. (Variety, Veracity)
4. Many data sources keep their data stored locally and require a consumer to actively ask for the data. This makes a comprehensive overview of all ongoing activities and security-relevant data difficult.

CHAPTER 1. INTRODUCTION

5. To detect advanced network attacks, the data of multiple sources has to be correlated. In many cases, existing data has to be enriched to be correlated. (Veracity)

In today's network environments, heaps of data from various sources are not effectively used. Even worse, security data that could support the detection and prevention of network attacks lies dormant on various systems in the network. Therefore, it is not uncommon after a major data breach that special security firms and professionals are hired for long-lasting manual investigations [12, 13] of raw security events to assess what has happened. Nevertheless, as networks get bigger and attacks more complex, this cannot be handled in a timely manner anymore.

1.1 Problem Statement

A new direction in handling the processing and analysis of security data, particularly security events, are so-called SIEM systems. The 1st generation of these systems was able to gather and manage alerts, i.e., security-critical events, from various sensors. The 2nd generation of SIEMs has the goal to incorporate all kind of security and event data into one big system [14]. This 2nd generation of SIEMs seems promising for solving the current problems of massive security data. However, looking at products on the market in the SIEM sector [15] reveals that the development is lagging far behind. Systems that are attributed to the 2nd generation, cannot handle the vast amounts of data fast enough, making a real-time attack detection infeasible. Additionally, it is not ensured that these systems can handle the load of the most critical security event sources, i.e., servers and routers, in large enterprises. The main issue of current SIEMs, as well as general security solutions, is the process of gathering, normalizing and persisting unstructured security data, which is also commonly known as the ETL-process in data processing [1, 16].

The previously described problems in the real-time attack detection can be seen as a typical *Big Data* challenge [1]. The processing of security data is characterized by the popular 4 V's of Big Data, i.e., *volume*, *velocity*, *variety*, and *veracity* [17]. These characteristics make it hard for traditional computer systems and technologies to handle the data on time. Fortunately, this situation is currently changing with the advent of plentiful memory, fast processing and fast networks for small money. Technologies that were not imaginable, mainly because of limited resources, are now becoming a reality [14, 18]. Concrete developments in this direction are in-memory databases, such as SAP HANA, and massive parallel and distributed processing clusters, which are realized with technologies like Google's MapReduce or solutions like Hadoop, Spark, or Storm.

1.2 Thesis Contributions

In this thesis, we show how the deluge of security-related data can be addressed with the efficient use of new Big Data technologies and the optimization of data processing, i.e., mainly the ETL/ELT-process with normalization and persistence. Our primary approach is the comprehensive normalization of security data, the persistence into an in-memory database, and the parallelization of all event processing. As the data is normalized and persisted in an in-memory database, we propose further techniques for correlation and analysis, also partly known as Complex Event Processing (CEP) [16, 19, 20], of normalized security events and external cyber threat intelligence (CTI). The analysis can be applied to the data in near real-time, meaning as soon as the events have been generated in the monitored network. This enables immediate mitigation of ongoing or future attacks.

To show that our approaches work practically, we have created two prototypical implementations of Big Data analysis systems. The first system, named REAMS, can be categorized as a SIEM system and analyzes security event logs to detect attacks from various activities in a network. The second system, called ILC, collects and analyzes publicly leaked identity data as contextual security information for the REAMS system.

In particular, this thesis provides four main contributions to the research community.

1. A Scalable High-Performance Event Processing Workflow:

The analysis of security-related events is an integral part of security investigations but is also extremely challenging to conduct because of the huge amount and diversity of available event data. At the moment, these challenges make it almost impossible for security investigators to detect ongoing attacks in time. As support for this challenging task of security investigators, we have created a highly efficient event processing workflow that performs the gathering, full normalization, enrichment, persistence of event data in near real-time [21, 22, 23, 24, 25]. All these steps together enable advanced event analytics that was not feasible before, such as complex search queries, pattern detection, and machine learning.

The main focuses of our proposed workflow are an accurate and complete normalization of available event information, an efficient persistence into an in-memory database for fast event access, and the best possible event throughput with parallel processing.

In the first part of the workflow, we normalize the event data by first extracting all relevant event information and then put it into the Object Log Format

(OLF) [26, 27], a fully-structured and comprehensive event format. The normalization procedure works with a knowledge base that contains normalization rules for each available event type. As part of this knowledge base, we came up with an efficient algorithm to find the corresponding rule for a given raw event [28]. Once the extracted information is stored in OLF, missing event information is enriched, and existing information is unified [29, 30], e.g., by categorizing the event semantics.

In the second part of the workflow, we are persisting normalized OLF events in a column-based in-memory database, namely the SAP HANA¹. The persistence has been optimized by using minimal SQL statements and the adjustment of performance controls within the SAP HANA Database Management System (DBMS) [25].

In addition to the implementation of the processing steps, we have put another focus on the maximal parallelization of the entire workflow and each of its steps to make normalized events available in near real-time for further processing. The workflow has been made horizontally scalable by supporting the distribution to multiple processing nodes and vertically scalable by making use of multi-threading in conjunction with fast inter-thread communication [24, 25]. Altogether, we can achieve event throughputs for normalization and persistence of around 280 000 evts./s [25], which is considerably faster, i.e., factor 9-10, than most SIEM solutions on the market and is more than sufficient to handle the load of events in large enterprise networks [15, 31].

Our presented approach shows that large data volumes can be normalized and persisted with high speed. This functionality is a foundation for further analytics on normalized security data and enables the detection of advanced attacks in the following work.

2. Attack Detection on Normalized Events:

Full normalization of event data is the cornerstone for efficient security analysis. It makes the information of raw events easily accessible and allows fine grained access to common data fields like IP addresses, domain names, ports, application names, and usernames. Furthermore, the common data model allows the correlation of events by a selected number of data fields. As part of our work, we show different techniques to detect attacks from normalized events.

The simplest technique is a complex search query on the structured data. It can deliver a broad overview of the data and gives an idea of existing nodes in a network and what kind of activity is going on them [32]. We also show

¹SAP HANA - <https://hana.sap.com/about/hana.html>

CHAPTER 1. INTRODUCTION

how these queries can be used to find some obvious attack attempts within a network.

Another approach is the employment of attack signatures on the data. Such signatures are either manually created or are derived from CTI or various existing signature languages. We show that simple attacks or suspicious activities can be automatically identified with single-event signatures. On top of that, we also show how more complex attacks, identifiable by a chain of events, can be detected with multi-event signatures [33]. For this, we have modified an existing signature language and engine to describe behavior on normalized events. An attack pattern detection on normalized events makes it possible to formulate typical attack behavior only once in a signature and then reuse it across all kind of applications. This is an improvement over traditional attack signatures, which were customized to each application.

3. Searching the Dark- and Deep-Web for Leaked Identity Data to Mitigate Attacks:

Every day, many companies and service providers on the Internet are breached, and their user data is released to the public, which is also known as a *leak*. These data leaks contain identity data such as email addresses, usernames, passwords, and financial data.

Releasing this data puts the victims and many related companies at a high risk since the data is usually widely misused by cybercriminals. There are different strategies to make use out of the data. Firstly, an attacker uses the detailed identity information to impersonate the victim, which is known as *identify theft*. Secondly, criminals use the credentials to login into other services and private networks, such as company networks. This is possible as many users are reusing usernames and passwords across multiple services and even at private or corporate networks.

Identity leaks are a major threat that has to be considered for personal and enterprise security. We have carefully investigated the topic of identity leaks and acquired an insight into the underground economy of leak sharing and trading.

During this investigation, many publicly accessible leaks have been collected manually, and common leak locations are automatically monitored for future leaks [34]. We are now able to collect hundreds of leaks each month. Based on this collected leak data, we have created a workflow to normalize the identity information in these leaks, extract affected identities, warn the victims and generate meaningful statistics on password security on the user and service provider side [34, 35, 36, 37].

4. Implementation of Prototypical Big Data Security Analytics Systems for Event Logs (REAMS) and Leak Data (ILC):

As a result of our research on the normalization and analysis of security-related Big Data, we created two large prototypical systems that prove our claims.

We have created the *Real-Time Event Analysis and Monitoring System (REAMS)*² for monitoring security-related event logs. The main idea of this system is the combination of capabilities from Intrusion Detection Systems (IDSs) and SIEMs into one system, meaning that common attack detection techniques, i.e., *anomaly-* and *signature-based* detection, are applied on centrally managed event information. Another idea is to apply parallel processing and use in-memory storage to reach near real-time processing of event information. Our system has been already practically used to analyze log data with billions of events from two well-known global companies.

Another system we have created is the *Identity Leak Checker (ILC)*³, the goal of which is to warn victims of identity theft and produce statistics on password security for security awareness. This system consists of three components, a *backend component* that collects publicly accessible and leaked identity data from the Internet, a *web interface* that allows victims to find out whether their data is affected by these leaks and how secure their password is, and a *client* that enables domain owners to monitor all users in their domain. Because identity breaches are more prevalent than ever, and people are aware of their threat, we have registered more than 7.5 million requests in our service.

1.3 Thesis Organization

This thesis is organized into three parts.

Part I introduces the general topic of Big Data. Chapter 2 defines the Big Data term and covers the fundamental properties of Big Data. The following Chapter 3 then shows current paradigms and strategies to deal with such data. The last Chapter 4 shortly introduces some popular tools for Big Data processing that are currently used in the community.

The second part is the main part of the work and brings the concepts of Big Data processing into the security monitoring context. Chapter 5 describes the challenges of implementing security in enterprise networks and presents an architecture and workflow for a Big Data SIEM. Chapter 6 then goes deeper into the workflow's stream-based processing steps, which are essential for the provision of events in a common format to later analysis steps. Chapter 7 solely focuses on the optimization of the

²HPI REAMS - <https://sec.hpi.de/reams>

³HPI Identity Leak Checker (ILC) - <https://sec.hpi.de/ilc>

CHAPTER 1. INTRODUCTION

most time-consuming step of the workflow, i.e., the normalization. This optimization is the basis for the desired real-time processing. The scaling of the entire workflow to multiple processors and nodes is discussed in Chapter 8. Chapter 9 goes over from the preparation of the data to the analysis on top of it. Multiple techniques are described on how attacks can be detected within log events. In the last chapter of Part II, details about the nature of identity leaks are revealed. It is further described how information from these leaks can be automatically processed to warn users of identity theft later.

The third part brings the theoretical considerations of the last chapters into the practice. More specifically, two prototypical Big Data platforms are presented that are realizing previous ideas on event analysis and the monitoring for identity leaks. Chapter 11 describes REAMS, the system that collects and analyzes huge amounts of log events in an enterprise network to detect advanced attacks. Chapter 12 presents our productive implementation of a system that gathers and provides information on compromised accounts of public identity leaks to individuals and enterprises.

The last Chapter 13 concludes our work and proposes ideas for further work on the topic.

Part I

Big Data Terminology and Concepts

Chapter 2

The Anatomy of Big Data

The collection and analysis of data was always an important process to understand phenomena and find correlations in sciences. In the past centuries, data has been carefully collected and analyzed manually to gain new knowledge. As this task was so time-consuming, scientists were limited to rather small datasets and could only focus on selected problems. Since computers had entered our lives a few decades ago, the way we look at data and information has fundamentally changed.

With the help of computers and digital sensors, the process of collection and analysis of data can be fully automated and performed in short time with high accuracy. As a result, we are not limited to selected problems anymore. We can monitor various phenomena simultaneously with ease and collect data points with a precision of nanoseconds or less. The datasets we have now at hand are getting bigger and bigger. We have moved on, from a world of small and limited datasets to a world of data abundance, where huge datasets are available on almost any imaginable facet of our lives. Trends like the *Internet of Things* (smart objects, Industry 4.0), *e-health*, *e-government*, *cloud computing* and of course *event log monitoring* are supporting this development. The value of the collected data can be immense. Refrigerator data can be used to derive product preferences of customers, smartwatch data can give details on the wearer's health status, security events can help to uncover attacks, and so on.

The availability of large amounts of data on almost everything is a valuable chance to understand our world better. Nevertheless, the analysis of all this data, the extraction of information and the derivation of new knowledge is still a major challenge to be faced. Our traditional analysis methods are not working efficiently anymore, because they were mostly designed for small datasets and are not using the potentials of new hardware. Instead, new methods have been created that can process Big Data fast by making use of new processing paradigms and current developments in hardware, such as multi-processing, in-memory technology, and data partitioning.

This chapter gives an overview of the problems related to this new data deluge, which is also referred to as *Big Data*.

2.1 Definition of Big Data

Before we dive deeper into the technologies and possible processing paradigms of Big Data, a general definition of the term should be established. In fact, there are many discussions on what Big Data really means and what the term covers [1]. Although many people are talking about Big Data, there is no single definition everyone has agreed on, yet. Therefore, we have selected some common definitions that can be found when dealing with Big Data.

1. *“Big data refers to things one can do at a large scale that cannot be done at a smaller one, to extract new insights or create new forms of value, in ways that change markets, organizations, the relationship between citizens and governments, and more.”*
(Mayer-Schönberger et al. [1, p. 6], 2013)
2. *“Big data is high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation.”*
(Gartner, IT Glossary)
3. *“Datasets whose size is beyond the ability of typical database software tools to capture, store, manage, and analyze”*
(McKinsey [38], 2011)
4. *“Big data technologies describe a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling high-velocity capture, discovery, and/or analysis.”*
(Gantz et al. [39], IDC, Sponsored by EMC Corporation, 2011)

The four definitions show that the Big Data term can be interpreted differently and that they focus on multiple key points. The first definition focuses on different types of data processing and its potential for revealing new insights. Definition 2 puts a focus on the properties of the data that is being processed. It makes clear that there is usually a large amount of data and that its structure makes it hard to be processed. Definition 3 is of a relative nature, as it puts the data into relation to the current processing capabilities and concludes that there are not enough resources, yet. Definition 4 covers the specific technologies and software components that are required to handle the new kind of data. It implies that the algorithms and systems we were using before cannot be used anymore, because they are not efficient enough. Additionally, this quote reiterates that the data has specific properties that make it hard to process.

CHAPTER 2. THE ANATOMY OF BIG DATA

A reasonable classification for the various types of definitions has been given by Hu et al. [40]. They classify definitions into *attributive*, *comparative*, and *architectural definitions*, which together cover the most important key points of Big Data.

Attributive: The key point in these definitions are the various properties of the data, which are usually referred to as the V's, such as volume, velocity, and variety.

Comparative: The main part of these definitions is that the data is compared to the data we can already handle. In this case, Big Data is mostly considered to be not processable with existing means, such as hardware, software or algorithms.

Architectural: These kinds of definitions are focused on special frameworks required to process the data, such as frameworks handling parallel processing and the distribution of the processing.

For sure, typical definitions do not fit into only one class. For example, Definition 2 by Gartner fits into the attributive and architectural group, because it requires the 3 V's (attributive) and talks about new forms of information processing (architectural).

2.1.1 The 7 V's

The attributive definition describes Big Data by its typical characteristics that distinguish it from traditional data. In literature, these characteristics are often referred to as the V's, since all of them can be written with the letter *V* at the beginning.

Volume: The volume indicates the size and quantity of the data. While traditional datasets were in the range of mega- or gigabytes and a few thousand records, newer datasets are moving into the range of tera- to petabytes with billions or even trillions of records.

Velocity: The velocity describes the speed at which data is arriving. In the past, datasets were once collected and not changed afterwards. This has changed with Big Data, since data sources, especially any kind of sensors, are continuously producing new data records. The speed at which the data is arriving is getting higher and higher and makes it almost impossible to put in a single storage system.

Variety: The variety describes the fact that the structure and content of the produced data are more and more diverse. While traditional data was produced by a single data source with a predefined representation, there are now various heterogeneous data sources that produce different sorts of data in very different, mostly unstructured, representations.

CHAPTER 2. THE ANATOMY OF BIG DATA

Veracity: The veracity is related to the quality of the data. In the past, data was once produced and checked for its integrity and quality. Incorrect data was discarded to keep a consistent dataset. With Big Data, this level of single record quality is not necessary anymore, because the volume and velocity can compensate for minor mistakes.

Value: The potential of Big Data is enormous as it allows to derive insights that were not easy to obtain before. The mere volume and diversity of the data alone can reveal new correlations. Because of these capabilities, the estimated value of the new kind of data is immense. However, the value is only unleashed when the data is carefully prepared for further analysis.

Variability: The variability describes differences in the meaning of information within the data. It means that an extracted piece of information can be interpreted in multiple ways, but only one interpretation is right. Only in conjunction with the context of the information, the true meaning can be derived.

Visualization: Especially with a vast amount of data, it becomes difficult to understand the meaning behind the data. Therefore, a clear visualization of the data and its analysis results is crucial to derive new insights. However, finding a good visualization is a challenging task that still has to be faced.

The main characteristic of Big Data, as the name implies, is that the data is big or has a large *volume*, as mentioned in Definition 2. Looking at some big Internet companies reveals that datasets of many petabytes (PB) up to a few exabytes (EB) are not unusual. Netflix maintains around 60 PB (2016) of content [41], Facebook manages a dataset of around 300 PB (2014) [42], and Google is estimated to even store around 10–15 EB (2014) in their data centers [43]. Concerning the worldwide data, Mayer-Schönberger et al. [1] estimate that there are 1.2 zettabytes (ZB) of data in 2013. The satellite data expert Richard Currier [44] even believes that there are 4ZB of data only generated in 2013. Dragland [45] guesses that around 90% of all data was generated in the past two years and according to IBM [46], every day around 2.5EB of new data is created.

Gartner has introduced a broader set of properties as the *3 V's*, i.e., *volume*, *velocity*, and *variety*. The importance of *velocity* becomes clear when we look at the user interactions that big Internet companies have to handle. Each user interaction is related to a piece of data that is generated on the service side and can be used for analytical purposes. The company Visual Capitalist has assembled an overview [47] of what happens in one minute on the Internet. According to their statistic, each minute, Google receives ≈ 3.7 M search queries, Facebook experiences ≈ 973 k login attempts, and ≈ 187 M emails are sent. For one day, these numbers sum up to more than a billion actions to handle. Such a large amount of incoming data is well above

CHAPTER 2. THE ANATOMY OF BIG DATA

the figures we are familiar with. Handling such speed of arriving data is a typical problem of Big Data. The *variety* of data is another key factor of Big Data. Large datasets, such as from Facebook, Google and Twitter, are mostly user-provided and therefore unstructured and in textual form. Additionally, various data sources are producing data without employing a common representation for their data. Analyzing such data is another major challenge for today's Big Data algorithms.

The 3 V's are the fundamental attributes of Big Data. Additional attributes have been added later, for example by IBM [48]. They have proposed to add *veracity* for the 4 V's, since with a large variety and volume the quality of the data becomes an inevitable challenge. Shortly after that, IBM added another V, namely *value*, to the attributes, making it the 5 V's. The newest definitions are even referring to 7 V's, as they are all listed above.

Our Definition on Big Data As a conclusion of all the above definitions of Big Data, we have created our own understanding of the term that will also be the basis for the considerations in the following part of this work. Two facets are particularly prominent in all discussions about the Big Data term, i.e., volume and velocity. We consider these two properties as the foundation of our view on Big Data. Furthermore, we share the view of Mayer-Schönberger et al. that Big Data does not have to be necessarily big in absolute numbers, but it constitutes a *big part* of a comprehensive set of data [1]. However, for a typical Big Data problem, we consider datasets with hundreds of millions of records as regular size and expect throughputs of more than 10 000 records/s for live datasets. Due to these extreme conditions, we consider it as necessary that new technologies and processing architectures are inevitable. Another important factor in our view of Big Data is the variety and variability. In particular, the information lying dormant within the data cannot be easily extracted and needs special treatment, i.e., prior indexing or normalization, and solutions, such as flexible storage and query mechanisms.

2.2 Data Representation

One key attribute of Big Data is variety, indicating that the data has a broad spectrum of representations. A representation can be considered as the methods used to store or exchange information. The following subsections give a short overview of the representational characteristics of Big Data, such as the data structure and data types.

2.2.1 Data Structure

Data has to be structured to make it understandable for machines and available for algorithms, so that it can fit into program's data structures or a database's table struc-

CHAPTER 2. THE ANATOMY OF BIG DATA

ture. When new data arrives, it can be in three different states, i.e., structured, semi-structured and unstructured.

Structured: The data is fully structured, and each piece of information can be directly mapped to a data field. In other words, the data is ready to be put in a table structure. Due to these properties, structured data can be processed efficiently and without prior preparation. According to a study by the company IDC [39], 5-10% of all data is structured.

Semi-Structured: Semi-structured data has some elements that are structured, but most of it is unstructured. Usually, the structured part is some meta-information. However, the actual content is mostly unstructured. The structured metadata can be easily represented in a relational database, but the unstructured part makes a reasonable analysis of the data difficult and therefore requires some pre-processing. According to the IDC study, also 5-10% of all data is semi-structured.

Unstructured: Unstructured data has no distinct format and the information encoded in this data is not easily extractable. Examples of such data are images, videos, audio, but also any kind of text. Today, unstructured data is the most common and makes up 80-90% of all data.

Unstructured data is considered as one of the main challenges of Big Data but also contains much valuable information. Consequently, methods on how to make use of arbitrary text, images and videos are heavily researched.

2.2.2 Common Data Types

The overview of data representations has revealed that a large part of available Big Data is unstructured. This nature of the data can be better understood with a look at some common data types [17].

Text Data: Text is used for the transfer of human knowledge and information and can be found in books, documents, websites, and more. Incorporating all this information into data analysis can bring entirely new insights and chances. While a single person is not able to read all existing literature about a topic, a machine that understands how to handle text could easily do. However, the understanding of text is still a challenge, because it is unstructured and involves a complex grammar. New methods like Natural Language Processing (NLP) and machine learning allow the interpretation of human language by machines.

CHAPTER 2. THE ANATOMY OF BIG DATA

Sensor and Log Data: A sensor is a device or software that measures a specific physical or chemical parameter or detects occurrences of predefined *events*. Upon a measurement or detection, a sensor emits a textual *event log entry* that specifies which parameter has been measured or what kind of event has been observed. With the currently ongoing datafication, sensors are measuring almost any imaginable parameter and their use is growing unstoppably. The challenge of sensor data is the partial or missing structure and a missing common format to express the observed event or parameter. In this thesis, we will show how to solve the problem of unstructured or semi-structured log data.

Multimedia Data: Audio, music, photos, graphics, and video are becoming increasingly popular on the Internet and are used to document many parts of our life. As the saying “*A picture is worth a thousand words*” implies, multimedia data can contain important information that can be useful for analysis. The challenge of multimedia data is to interpret what it represents, since all available data consists of encoded physical signals, e.g., acoustic waves. Techniques like object detection for pictures or voice recognition for audio are addressing this issue.

Handmade Data Tables: This data type covers information that was collected manually and combined in a table or register, such as for car owners, residents (census data), products (catalogs), companies (yellow pages and commercial registers) and landlines (white pages). Much of this data was created before the era of Big Data, but is still relevant for Big Data analysis because of the data coverage and completeness. Since such data tables are created manually and were initially intended for reading by humans, they often contain both, written text and information fields.

2.3 Challenges of Big Data

The presented characteristics make clear that the handling of Big Data is connected to many challenges. Many of these challenges can be directly derived from the 7 V's. In the following, we present an overview of the main challenges to understand why certain technologies are used for the processing of Big Data and to which processing phases special attention should be paid.

Huge Amount of Data and High Throughput The most prevalent challenges for Big Data are its volume and velocity. They make it particularly difficult to process datasets in real-time and require new strategies for processing and storage. On the one hand, traditional mechanical hard drives can provide plenty of storage space but are

CHAPTER 2. THE ANATOMY OF BIG DATA

sometimes not fast enough to handle incoming data. On the other hand, in-memory storage can handle incoming data very fast but has only limited space available. Even if data can be stored fast enough, the fast access to it also has to be ensured to enable further complex analytics.

Missing Structure and Different Formats The interpretation of the data contents are another major challenge in the processing of Big Data. The interpretation is regarded difficult, as the more significant part of data is unstructured (80-90%) and different formats for the representation are used. This fact requires complicated pre-processing steps, such as indexing or normalization, which extract relevant information from the raw data. The extracted information then has to be converted and put into a common data structure to enable access for comprehensive analytics and queries.

Data Quality Data originates from many different sources and is produced by a large variety of sensors and sensor types. Not all the provided data is as accurate as one would wish, be it because of misconfiguration, missing calibrations, software or hardware deficiencies or simply failures. Sometimes, the sheer amount of data can overcome some of these inaccuracies. Nevertheless, not all these issues, particularly missing information, can be solved with data volume. For the remaining mistakes, another pre-processing step needs to be deployed that can complete missing information from the context and can identify and fix deviating or wrong values.

Correlation The power of Big Data lies in the correlation of multiple data sources to create a big picture. This correlation can be realized with manual queries, automated correlation algorithms or machine learning approaches. Still, before such methods can operate efficiently, the data needs to be moved to a structured form and a certain level of quality has to be established. Only then, a data analyst can start to find the right features within the data that can reveal new insights.

In this thesis, we show how many of the above challenges can be solved in the context of event-based security analytics.

Chapter 3

Big Data Processing

Big Data is characterized by properties that make it difficult to process. The mechanisms and tools we have used in the past to process data are mostly not capable of handling this new kind of data, which is huge, has no clear structure and has deficiencies in quality. Nevertheless, making use of Big Data can give us new insights we were not able to obtain before.

In this chapter, we present different existing approaches and technologies to deal with Big Data and present a processing workflow covering the steps from raw data to the analysis of structured data.

3.1 Data Processing Paradigms

Traditional data was typically in the range of thousands of data records, which did not require performance-optimized algorithms. Big Data is often in the range of millions or even billions of records, which requires a new elaborate strategy for processing. In particular, it becomes inevitable to use all available resources for processing and to come up with new best practices and paradigms for handling data volume and velocity.

3.1.1 Aiming for Scalability

Scalability can be seen as the capability of a system to handle a growing amount of work or data. It is a necessity for the processing of large and increasing data volumes. Unfortunately, many existing algorithms and processing paradigms were not designed with scalability in mind, making their adoption to Big Data difficult. In general, scalability can often be achieved by adding more resources and making algorithms or programs ready for parallel processing. The methods to implement scalability can be grouped into two categories: horizontal and vertical scalability.

Horizontal Scalability This category includes methods that add additional computer systems or nodes to achieve higher performance. The distribution of the processing task to multiple nodes is typically coordinated over the network. Horizontal scalability is known to be a cost-effective and easily implementable solution since many relatively low-end nodes can be combined into huge computing clusters, which can even outperform supercomputers.

Vertical Scalability This type of scaling adds computing resources or additional hardware to a single node. The most common types of resources added or improved are CPUs or main memory. Vertical scaling has the advantage that smaller scale-ups are easier to achieve and could be cheaper, because no network is needed and only one hardware component with better performance has to be added. Nevertheless, as soon as larger scale-ups are required, the cost of better hardware explodes and makes the scaling economically unattractive.

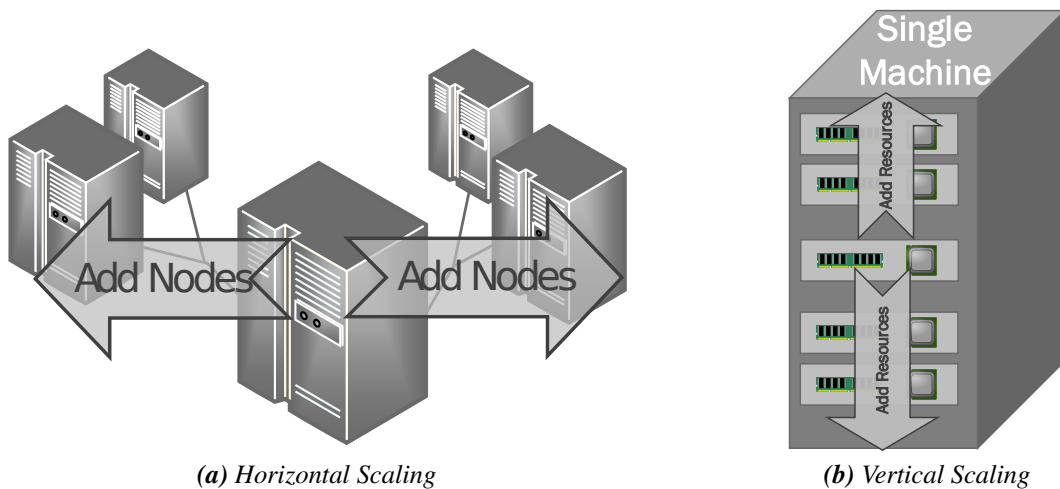


Figure 3.1: Types of scaling

Both types of scaling are desirable for high-performance processing infrastructures. Vertical scaling allows the efficient use of a single machine whereas horizontal scaling combines multiple such single machines to a more powerful cluster. A system architect has to find the right balance between both solutions.

3.1.2 Methods of Performance Scaling

A large part of the scaling for Big Data can be achieved with a small set of methods. Some of these methods are already used for decades to scale applications, but

CHAPTER 3. BIG DATA PROCESSING

there are also methods that have just been introduced with the rise of Big Data. The following describes five of the most popular methods.

Multi-Threading A thread is a unit of code that is executed in a processor core. A single process can have multiple threads, and each processor core can execute one thread at a time. With the advent of multi-core processors, one computer can run multiple threads in parallel and thus one process can have multiple parallel execution paths. The term multi-threading means that a program executes multiple tasks simultaneously to achieve better performance. According to Amdahl's law, a program can be sped up by the number of available processor cores if the code is fully parallelizable [49]. Multi-threading is a typical method of vertical scaling, because it relies on the addition of more processor cores to an existing system.

Distributed Processing A single machine is limited in the processing resources it can provide. The idea of distributed processing is to use the processing resources of multiple computers being connected over a network, a so-called cluster. The computing nodes do not even need to be very powerful, because processing power can also be reached with many low-end computers. Additionally, an existing cluster can be easily expanded with more nodes. Distributed processing is a typical method of horizontal scaling and is one of the primary methods to keep up with Big Data.

In-Memory-Databases Traditional DBMS have used hard drives to store data and are slow when they access data records. To overcome the long-lasting hard drive access, they are keeping hot data, i.e., data that is often accessed, in caches within the main memory. Due to the slow access and hardware limitations, these DBMSs are not suitable for large amounts of data. In-memory databases are a new generation of DBMS that move the storage of data to main memory and the caching to the CPU-caches, making data access and analysis extremely fast [50]. This approach is feasible, since a large main memory and powerful processors are getting popular and cheap. In large companies, machines with 1 TB and more of main memory are not unusual anymore. In-memory databases can be grouped into the category of vertical scaling, as the performance improvement is achieved by adding more main memory.

Data Partitioning Another method for improving the performance of a DBMS is to use data partitioning. The idea is to split up the table data into so-called partitions, which contain independent subsets of the columns or rows of the original table. For row-based partitioning, a partitioning scheme decides into which partition a row is put. A DBMS can handle its partitions on a single machine or can distribute them to multiple nodes. The distribution of the data partitions, also called sharding, allows a

CHAPTER 3. BIG DATA PROCESSING

DBMS to handle data requests independently on different machines and can, therefore, reduce loads significantly. One of the major challenges of sharding is to find an efficient partitioning scheme that supports the separation of the data in a way that each request can be processed in a dedicated partition. Due to the distribution of data to multiple nodes, sharding is considered as a method of horizontal scaling.

Message Brokers and Queues In a distributed processing cluster, nodes and workflow components are exchanging messages to coordinate processing steps and to transmit working tasks. As a consequence, the fast and reliable transfer of messages to all cluster components is utterly important for a well-functioning Big Data workflow. A message broker or queue is a system that coordinates the message transfer between so-called *publishers* and *subscribers*. A message queue receives message streams from multiple publishers and assigns the messages to a topic queue, according to the requirements of the publisher. On the other side, a subscriber asks the broker for messages from a specific topic queue. The publisher/subscriber concept allows asynchronous communication between workflow components and supports their decoupling. The message queue itself is a fault-tolerant list of messages where each message is addressable over an index. Accordingly, a subscriber can rewind and reprocess data as needed. A retention policy defines how long a queue should hold the received messages. Since a message queue supports the fast exchange of messages in clusters, it can be seen as an example of horizontal scaling.

3.1.3 Batch- and Stream-Processing

In the context of Big Data, there are two modes of processing that are often mentioned, i.e., batch- and stream-processing. The modes are distinguished by the way the data is consumed, which is either one data point at a time or the whole dataset at once. Each of the modes has advantages in particular use cases. The main differences are what kind of data can be processed, the performance in terms of volume and velocity and what kind of operations can be applied during the processing.

Batch-Processing This mode is a popular way of data processing, where a dataset is processed as a whole and in one finite run. It was already used in the past to process smaller finite datasets but is now also commonly used to process Big Data, such as historical data, i.e., data gathered and stored in the past for possible future analysis.

The performance of batch-processing is in favor of data volume. It can process large chunks of data in a relatively short time, meaning it has a high throughput, because it has access to the complete dataset and can fully parallelize the processing in the best case. On the other hand, the processing results take rather long to produce, meaning it has a high latency, because the complete dataset has to be processed first.

CHAPTER 3. BIG DATA PROCESSING

Depending on the size of the dataset, it could take from hours up to days until the final results are delivered. In many cases, there are also no intermediary results available.

Batch-processing can be applied multiple times on the same data and keeps the original data sound. This kind of reprocessing is a feature that is often required for Big Data applications that collect all the data they receive in one large dataset. To get the newest analysis results on the data, they apply the processing again and again on the whole dataset. In case there are problems with any processing step, results can be recalculated when the problem is solved. Batch-processing is also powerful in the kind of operations it can apply on the data, since the size of the dataset is known from the beginning and data records can be accessed many times, in contrast to stream-processing. This allows even very complex analytics on the data, as resources can be well estimated and do not need to be kept in use infinitely.

Stream-Processing This is a new mode of processing that is designed for live-data or, in other words, data that is arriving in a continuous infinite stream. Each record within the stream is handled individually without access to the whole dataset. Since many Big Data applications produce new data every second, stream-processing is necessary to provide real-time analysis of this data.

The performance of stream-processing behaves in the opposite way of batch-processing, meaning it has low latency paired with low throughput. It is optimized for velocity and designed to deliver analysis results in real-time. However, as a consequence of that, it takes longer to process larger datasets since records are processed one-by-one.

In the context of stream-processing, the one-by-one scheme has limitations for advanced analytics, because stream-operations do not have random access to all available data. All applied operations are limited to a single record. Another problem is the volatility of the processed data. Since already processed data is not kept, an error during the processing can lead to the loss of previous results and therefore inaccuracies. This is a contrast to the batch-processing approach, where there is always a chance of reprocessing.

Micro-Batch Processing This mode of processing is a mixture of the previously described modes and combines their advantages. Micro-batching is processing rather small batches, i.e., a few hundred or thousand records, in one run. In other words, it is processing a stream of small batches. The small batches enable it to produce preliminary analysis results in very short time and the processing is considerably faster than the one-by-one processing used for stream-processing.

The processing capabilities of micro-batch-processing are similar to stream- and batch-processing. Micro-batch-processing can be used to analyze large batches with quick intermediary results and reliable processing. Nevertheless, the accessibility

CHAPTER 3. BIG DATA PROCESSING

to all available data is still limited to this approach and makes complex analytics challenging.

Concluding, it can be said that all of the processing modes have their advantages and disadvantages, as shown in Table 3.1. Batch-processing is well-suited for performing complex analytics on massive datasets. Stream-processing is designed to work on live-data and is capable of delivering intermediary results on the data. Micro-batch-processing is a compromise between the previous two modes but is still not flexible enough to perform any kind of complex analytics. According to Akidau [51], well-designed stream-processing systems are not only faster than batch-processing, but they can even provide a superset of batch-processing functionality. This is why some people already regard stream-processing as the processing mode of the future.

Mode	Processing Entity	Latency	Throughput	Operations
Batch	Huge batches	Very High	Very High	Complex
Stream	One record	Very Low	Low	Lightweight
Micro-Batch	Tiny batches	Low	High	Normal

Table 3.1: Comparison of processing modes

3.1.4 Data Immutability

One of the main bottlenecks for a scalable Big Data architecture is the data persistence. An important role for the performance of that persistence plays the (im)mutability of a storage system, which defines how persisted data records are accessed and updated.

Mutable Storage Systems In architectures that were developed in the past, all data was stored in relational databases where a table row represents one data record. Whenever a record has to be updated with new values, the corresponding row in the table has to be found in an index and the changed data fields are overwritten with new values. During the time where a field is overwritten with a new value, the database makes sure that the updated field and its relational dependencies are locked and not read or modified by concurrent database accesses. Only these locks can keep the system in a consistent state and prevent damage to the data. The property that a data field is overwritten in place is called *mutable*. The mutability of traditional database systems is one of the significant hurdles for scalability, because index lookups and locks cause considerable delays when records are inserted and updated. While the delay is negligible for small datasets that are only accessed and changed occasionally, it becomes a big problem for huge datasets that are accessed very frequently.

CHAPTER 3. BIG DATA PROCESSING

Immutable Storage Systems Mutability and scalability are hard to achieve together. Therefore, data scientist and engineers came up with new models of databases that are focusing on immutability for better scalability. An immutable database has the property that any changes to the data are not realized through updates, but inserts, so that no more locking is required. Although an immutable database system only supports the addition of new data, modifications on existing data can be achieved with timestamping. With this method, each database operation is represented with a new database entry that includes a timestamp on when the operation was performed. So, the database consists of the whole history of operations ever performed on the data. The current state of the data can be derived by applying all the operations in order of their insertion. Additionally, a user can recover any historical state of the database. Due to the fact that no records are deleted or changed, and it is known which operation has been performed first, the database system can present a consistent state when no further operations are applied. Still, in an intermediary state where some operations are still outstanding, it might not be possible to present consistent results, especially on different nodes of a distributed database system. This type of consistency of a database system is also called *eventual consistency*.

Examples of immutable DBMSs are several NoSQL databases, such as Cassandra, the SAP HANA database as well as the HDFS from Hadoop.

CAP-Theorem The CAP-theorem, also known as Brewer's theorem [52], states that a distributed system can only fulfill two of the three properties *consistency*, *availability*, and *partition-tolerance* at the same time. So, when designing a distributed database system, one can pick two of the properties but has to sacrifice the third. Since no distributed system can prevent network failures, partition-tolerance is considered as a must-have property. So, in reality, there is a choice between availability and consistency.

An article by Marz [53] summarizes the effects of both choices quite well. When availability is chosen over consistency, a read may return a different value than what was just written. A system that aims to achieve consistency at some point, i.e., eventual consistency, can perform repairs on the data and the database will be sound in the end. When consistency is chosen over availability, it can happen that the database system either does not perform operations on time or that errors are returned on an operation because a part of the system is not available. A user then has to wait until the full system gets up again. However, any changes to the data are immediately available, which is also called immediate consistency.

Both presented combinations are very hard to realize and have much potential for errors. In his article, Marz sees the use of immutable databases as a solution to simplify the development of distributed database systems, because there are no incremental updates anymore that make the implementation of the consistency prop-

CHAPTER 3. BIG DATA PROCESSING

erty difficult. He also proposes a database architecture that he claims can beat the CAP-theorem. The details of this architecture will be described in more detail in Section 3.2.

The mutable model of database systems was designed for small datasets in environments with low memory resources. In the Big Data era, with abundant memory and huge datasets, mutable database systems are replaced with more and more immutable database systems. These immutable databases do not only perform much better, but can, at the same time, provide immediate consistency easier and allow recomputation on errors.

3.1.5 Processing Reliability

So far, we have focused on the scalability of a processing architecture. Another factor that plays into the design of a processing architecture is the fault tolerance and the met processing guarantees. When running a Big Data infrastructure with a large number of nodes, there is a relatively high chance that one of the nodes or the communication between the nodes fails. The fault tolerance of the system defines whether the interrupted execution caused by failure will be repeated and proper execution is guaranteed. However, it should be considered whether a guaranteed execution is needed or whether it is sufficient just to repeat the processing of the entire dataset, if necessary.

According to many definitions of Big Data [1], huge amounts of data can compensate for potential errors and losses of single records. Thus, even if a few records are failing during the processing, the eventual results of an analysis will still be similar. Furthermore, it has to be considered that many data analysis routines are repeated regularly, in particular for batch-processing systems, which make the guaranteed execution redundant. In the end, it is the choice of the architect or analyst, whether the processing of each individual record is important and analysis results must be correct at each time. In any case, the complexities that come with a guaranteed processing cause performance degradation.

In the literature, there are three common error semantics for the reliability of distributed systems, i.e., *at-most-once*, *at-least-once* and *exactly-once* [54, 55].

At-Most-Once This is the simplest form of reliability guarantee and does not provide fault tolerance. In short, a record is distributed once and is not replayed in an error case, so no additional mechanisms are required to implement it. If a record is lost within the transmission or its processing fails, then it would be as if the record never existed.

CHAPTER 3. BIG DATA PROCESSING

At-Least-Once This semantic already provides some basic level of fault tolerance. It means that each record will be processed eventually, but it cannot be prevented that a record is processed multiple times. A mechanism that can ensure this semantic is the acknowledgment. Whenever a record has been fully processed, it is acknowledged, and the processing of further records can go on. If a record is lost or the processing fails, there will be no acknowledgment and the record will be replayed after a timeout. In the worst case, some operations are performed twice for the same record.

Exactly-Once This is the strictest form of reliability and ensures that each record is processed at least once but prevents duplicate processing. Consequently, a record must be replayed in case the current processing failed, but the results of the processing are only considered once. Generally, exactly-once semantics are difficult to achieve and put limitations on the performed operations. However, lost records can be handled with acknowledgments. To ensure that a record is not processed multiple times, each operation performed on it must have no side effects or, in other words, be idempotent. Even if a record would be processed twice, the result of the processing will be as if the record was only processed once. As a conclusion, a workflow with non-idempotent operations cannot fulfill exactly-once semantics.

3.2 The Big Data Workflow

The analysis of Big Data can be considered as a complex workflow that transforms raw data from various data sources into some final analysis result. This workflow is not a single big transformation algorithm but consists of multiple processing steps that each contribute an essential part to the overall transformation. The processing steps are usually executed sequentially, and each step is either executed as a single instance or is run in parallel in multiple instances to achieve better performance. In the following, we first present some common architectures for Big Data processing to show in which environments the workflow is executed. Afterwards, we introduce and describe common processing steps as they are found in Big Data workflows.

3.2.1 Known Processing Architectures

The efficient processing of large amounts of data requires a sophisticated software architecture that can make use of parallel computing resources and is highly scalable. The key factor for such an architecture is a clever combination of the two processing paradigms stream- and batch-processing in a way that only their advantages come into play.

CHAPTER 3. BIG DATA PROCESSING

The design and creation of efficient Big Data processing architectures have been the interest of many companies, especially Internet companies like Twitter and LinkedIn, and has been discussed in various blog posts of researchers. Over time, two approaches have gained the most reputation in the community and have found their way in many Big Data analysis infrastructures. These two approaches are described in the following.

3.2.1.1 Lambda Architecture

This architecture was initially introduced by Marz in a blog post [53] in 2011 and was later refined in his book [56]. It is meant to eliminate the influence of the CAP-theorem in data processing and can handle complex operations on any kind of data and still produces results with low latency. The Lambda architecture consists of three layers, i.e., the speed-, batch-, and serving-layer. Figure 3.2 shows the composition of these layers in the architecture.

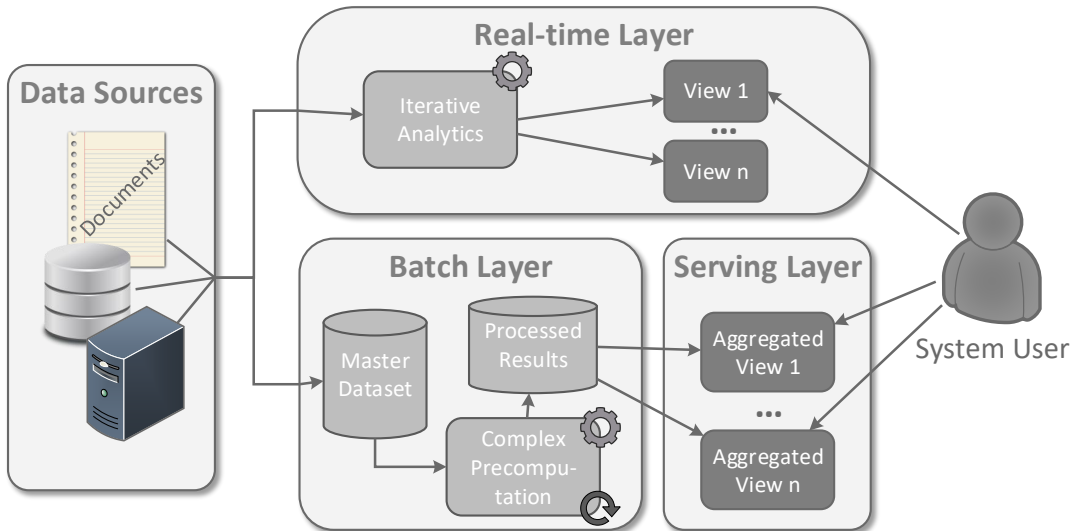


Figure 3.2: Lambda Architecture

The batch-layer is the core of the architecture and precomputes complex views on the master dataset, which is a copy of all data that ever arrived and is constantly growing with new data. Since a master dataset can easily cover multiple billions of records and the batch-processing can only deliver results after all records have been processed, there are usually large delays of multiple minutes up to hours for a single run of the batch-layer. As soon as the pre-computed results are available, they are written into a mutable database for the access of the serving-layer and the batch-layer starts a new run over the master dataset to incorporate new data that has arrived since the last run.

CHAPTER 3. BIG DATA PROCESSING

The speed-layer, also referred to as real-time-layer, is the complement of the batch-layer and is responsible for processing the data of the last few hours with low latency. It puts all its processing results as views into a mutable database, so that a system user can see the state of analysis in near real-time. Nevertheless, the incremental character of the performed operations and the difficulties connected to mutable databases makes the speed-layer susceptible to processing, hardware, and human errors.

The serving-layer acts as an access point to the data previously prepared by the batch-layer. The batch-layer already provides intermediary results that take longer to process within its database, but these results are not consumable for a system user. Therefore, the serving-layer applies further queries and provides aggregated views on the preliminary data. Eventually, it delivers the results with minimal latency to the system user.

According to Marz [53], the combination of speed- and batch-layer can reduce the influence of the CAP-theorem. When choosing availability over consistency, the batch-layer can easily provide eventual consistency because of the immutability. In the speed-layer, the consistency of the data is not as relevant, because the batch-layer eventually overwrites all the results produced by this layer after a few hours.

3.2.1.2 Kappa Architecture

The Lambda architecture was introduced as a solution to support the processing of large volumes of data with low latency and with the capability of reprocessing. The key concept of the architecture is to use stream- and batch-processing together. However, exactly this concept brings some difficulties in practice. Both processing modes are rather different in the way they handle data and need different logic to achieve the same results. For the Lambda architecture, this means that each processing task needs to be implemented twice, once for the batching and once for streaming.

An approach that addresses the problem of reimplementation in the Lambda architecture was proposed by Kreps in a blog post in 2014 [57] and is called the *Kappa architecture*. The main idea of the Kappa architecture, which is illustrated in Figure 3.3, is to eliminate the batch-processing and incorporate the batch data into the stream-processing. The reprocessing of data is now also handled by stream-processing, and higher throughput is achieved with parallelism and new hardware capabilities.

The processing in the architecture works as follows. Similar to the Lambda architecture, all arriving data is first written into an immutable master dataset that allows fast reading. Then, a single stream-processing instance works on the master dataset and writes the processed results into a table. Whenever the reprocessing of the data is requested, a new stream-processing instance is started on the master dataset again. As soon as the new processing instance has caught up with the master dataset, the previous processing instance is stopped. The serving-layer of the Kappa architec-

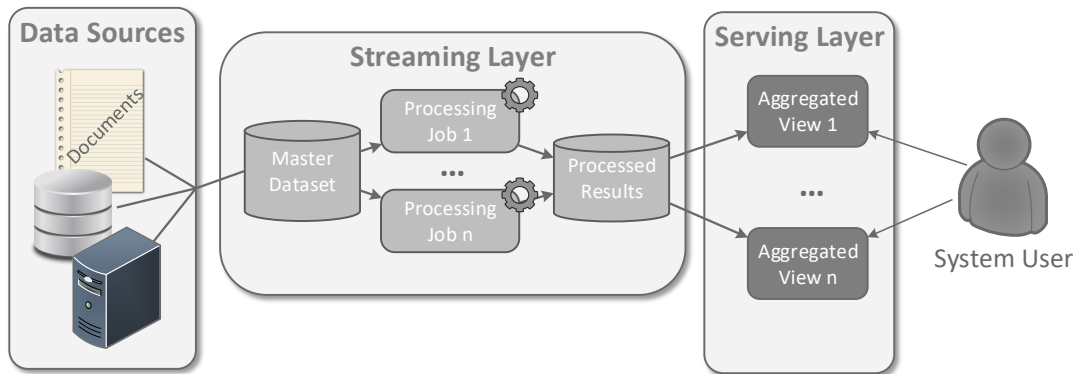


Figure 3.3: Kappa Architecture

ture is responsible for providing different views on the result table of the most recent processing instance.

Interestingly, the Kappa architecture shows that Big Data processing can be handled entirely with stream-processing. It confirms the idea of Akidau [51] that stream-processing can be seen as a superset of batch-processing if enough resources are available.

3.2.2 Traditional ETL/ELT-Process

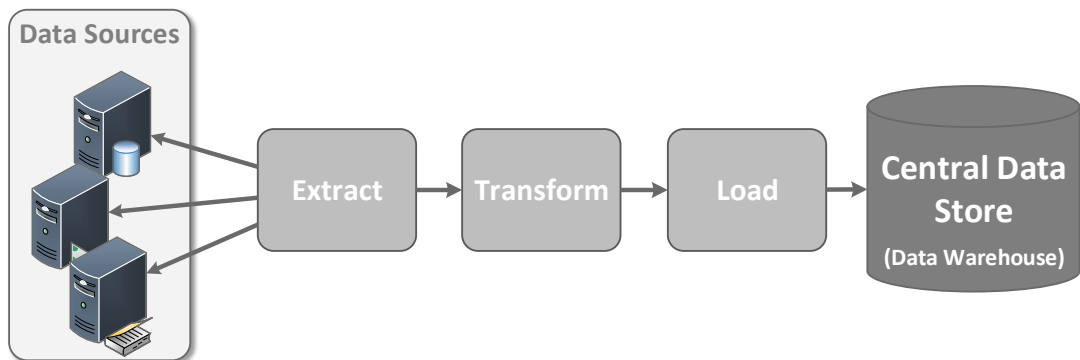


Figure 3.4: ETL-Process

The so-called Extract-Transform-Load (ETL) process is a fundamental data processing workflow that moves bulk data from various operational systems into a data warehouse, a single source of data on which all analysis is applied [58]. While data warehouses become less relevant as they are too slow and inflexible [59], the ETL-process is still relevant for feeding large amounts of data from various sources into a central data store, as it is common for Big Data processing. The defined process is

CHAPTER 3. BIG DATA PROCESSING

rather simplistic as it only consists of three high-level steps that give the ETL-process its name: *extract*, *transform* and *load*. The tasks of the steps are listed below, and Figure 3.4 illustrates the interaction between them.

1. **Extract:** Collect the data from all available data sources. A challenge of this step are the heterogeneous systems, their incompatible interfaces and the high variance in the formats of the provided data. To correctly extract the data from all sources, the extraction component needs to support all variations of interfaces and data formats.
2. **Transform:** Apply multiple transformation rules and functions on the raw extracted data in a way that the data can later be readily used for complex queries and analysis. Typical transformation functions are normalization, translation, cleaning, aggregation, filtering, or ordering.
3. **Load:** Move the previously transformed data into the final target, which can be a database, a message queue or a data warehouse.

There are many discussions on the eligibility of the ETL-process in today's Big Data architectures. People argue that the transformation is too heavy-weight to be performed in real-time and propose to pass the raw data directly to the data store. Any transformation would then either be provided within the data store or when the data is read. This kind of processing workflow is also commonly referred to as Extract-Load-Transform (ELT)-process, because it swaps the two steps *load* and *transform*.

Although the ETL/ELT-processes cover an essential part of the Big Data processing workflow, namely moving the data from their sources to a single data store instance, they do not further specify when and how analysis is performed on the stored data. Due to these limitations, a more detailed processing workflow is desirable, which is introduced next.

3.2.3 Our Processing Workflow

The processing workflow is the basis to understand which operations and transformations have to be applied on a given set of raw data. For now, there is no common workflow that fits all purposes, but rather many different workflows for various specific tasks to be performed on the data. In the following, we introduce a workflow that is more comprehensive than the presented ETL-process and is based on the typical processing steps that can also be found in literature about Big Data processing [16, 40, 60, 61, 62]. This workflow is used as the underlying framework for the main contributions of the following work and is therefore focused on the fast processing of high volumes of event data and incorporates data correlation and complex analytics.

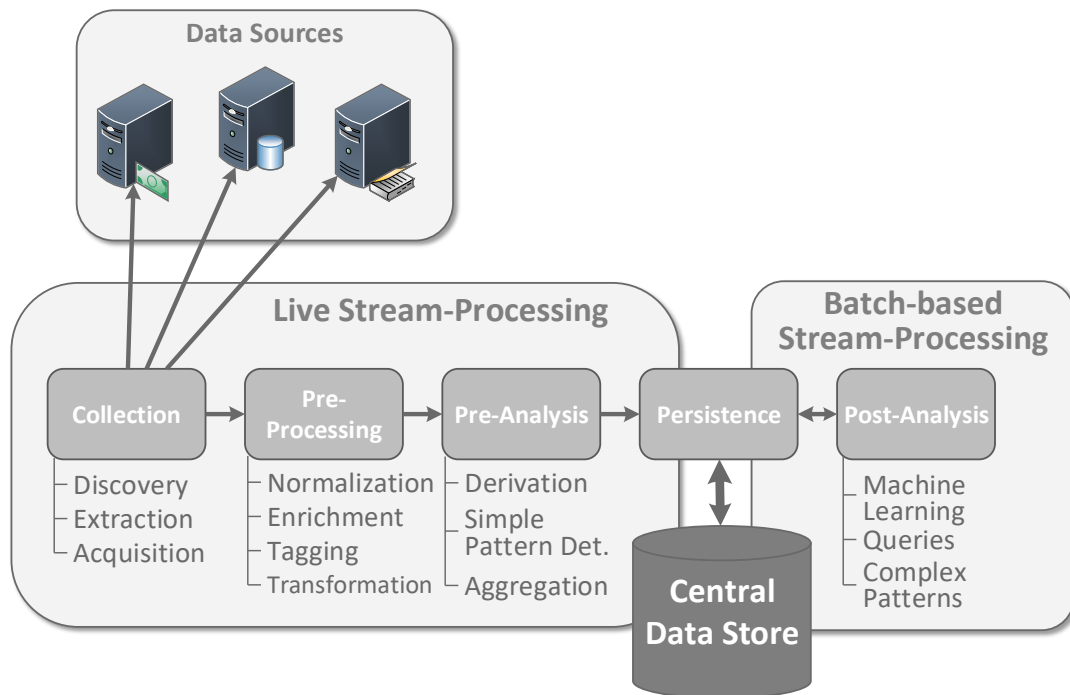


Figure 3.5: Processing workflow used in this work

A first overview of the workflow is depicted in Figure 3.5. The lower part of the architecture reveals that there are altogether five major processing steps, each having some common associated operations. The following list describes each of them in more detail.

1. **Collection:** This step is comparable to the *extract* step of the ETL-process. It is the first step in the workflow and handles the collection of raw data from the various data sources. Three major tasks have to be performed for the collection. The first task is the *discovery* of possible sources by looking for systems that are offering data. The second task is the *extraction* of data records from the found data. The last major part is the retrieval and *acquisition* of the extracted data so that it can be passed to the next processing steps.
2. **Pre-Processing:** The second step in the workflow prepares the received raw data for further analysis and persistence. One of the most important tasks in this area is the normalization of the raw unstructured data into a common structured format so that the following steps can easily access all available data fields. Since many of the received data records are incomplete or inaccurate, the data further needs to be enriched and tagged with contextual knowledge from a maintained knowledge base. Additionally, some data types are trans-

CHAPTER 3. BIG DATA PROCESSING

formed for easier processing.

3. **Pre-Analysis:** The pre-analysis step performs the first rudimentary checks on the data records. A possible task in this step is the derivation of additional knowledge from all passing records, such as the frequency and variation of values for specific data fields. Another possible task is to find simple patterns in the data and make findings of these patterns immediately visible to the following steps. To simplify analysis on the data at a later time, the pre-analysis can already perform rudimentary aggregations of records with similar properties.
4. **Persistence:** The persistence is similar to the load phase of the ETL-process and is a central part of the processing workflow. It moves incoming live records into a permanent data store. The data store is an immutable storage that is later also used as the master data for more complex analysis and enables recomputation as well as fast batch-based processing.
5. **Post-Analysis:** This step allows to apply complex analytical methods on all available data in the data store. Such analytical methods could be complex queries with aggregation, filtering, and ordering, or machine learning for finding anomalies in the data, or the identification of complex patterns by correlating multiple records with each other. The post-analysis step is also a suitable place for an analytical interface with which researchers and analysts can query and correlate data.

The five processing steps are distributed over two different phases in the workflow. The first phase is the stream-processing phase that can handle the live data with low latency and transforms the records into a structured form. Essentially, the operations executed in the first phase of the workflow have much in common with the operations of the ETL-process, but they are now handled with stream-processing. In comparison to the Lambda architecture, the pre-processing is not repeated for real-time and historical data but is only performed once for the live-data and is then stored for later batch-processing. In the unlikely event that some new pre-processing or pre-analysis operations become available, the raw events in the master dataset are reprocessed and existing records are overridden with timestamping, as mentioned for the modification of immutable database records.

The second phase focuses on the processing of historical data from the central data store. However, instead of using the typical batch-processing approach for master datasets, we propose to use stream-processing as used in the Kappa-architecture and mentioned by Akidau [51]. The data that is processed in this phase is already structured, so that operations can focus on the pure analysis. On top, all the processing is error-tolerant, because the processed data originates from the central data store that is immutable and can be read again and again without losing data.

Chapter 4

Tools and Libraries for Big Data

The last two chapters have introduced the theoretical aspects of Big Data. We now take a closer look at tools and libraries that bring this theory into practice. In particular, we are focusing on the tools and libraries from the two areas of parallel processing and data persistence.

4.1 Parallel Processing

Parallel processing is an important mean for scaling programs vertically on the same machine and horizontally over a number of computing nodes. Vertical scaling is achieved through multi-threading, which is already built into modern operating systems, whereas horizontal scaling is achieved through work distribution in the network. The coordination of work between multiple threads or nodes is a challenging task, as it requires efficient communication between processors. In the following subsections, we present known ways of so-called Inter-Process Communication (IPC) and further introduce frameworks that support a developer to parallelize programs vertically and horizontally.

4.1.1 Inter-Process Communication (IPC)

The coordination of processors for parallel processing encompasses status messages and the passing of tasks to processors with free resources. Depending on the data to be processed, hundreds of thousands of messages are exchanged within a second. Obviously, this communication becomes a bottleneck for Big Data processing.

Netty Netty is an event-driven network library for the Java programming language¹ that is optimized for high-performance network communication. In contrast to many

¹Netty Project - <https://netty.io>

standard network libraries, Netty focuses on asynchronous communication, also known as non-blocking I/O (NIO), and reduces unnecessary copying of network data between device and operating system memory.

The performance, simplicity and large active community make Netty popular for the communication within distributed Big Data systems, such as HBase, Storm, and Spark, which are described later.

LMAX Disruptor The LMAX disruptor is a software pattern and library that allows a high-speed exchange of messages between concurrent threads [63, 64]. It was originally created for the purpose of high-frequency trading where low-latency and a high message throughput are the key to success.

Traditionally, messages between threads were exchanged by so-called *blocking queues* as shown in Figure 4.1, a concurrent queue implementation where a consumer thread waits for incoming messages to be put into the queue by a producer thread. As with many concurrent data structures, the blocking queue has the disadvantage that any write access to the queue needs locks. Thus, whenever a producer thread puts new messages to the queue, the full data structure needs to be locked and the consumer threads have to wait for the locks to be released [64]. Particularly this locking causes small but remarkable latencies when many messages have to be exchanged in a short time.

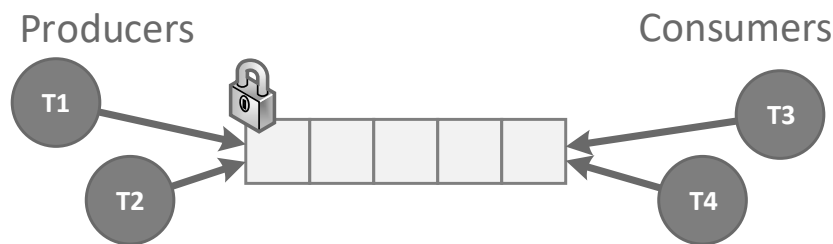


Figure 4.1: The blocking queue concept

The disruptor is a solution that reduces the number of locks to a minimum and consequently allows message exchange with tiny latencies. It mainly consists of two components, a ring buffer, and multiple sequences. Figure 4.2 demonstrates the concept of the disruptor. The ring buffer is a bounded data structure where each slot of the buffer represents a message to be exchanged. When a producer wants to publish a new message, it asks the disruptor for a free slot in the ring buffer and puts in its message. Then the disruptor tells the consumers that a new slot with a message is available for consumption. The slots that are filled by the producers and emptied by the consumers are tracked with sequences. When a new slot is filled, the sequence is incremented, and the consumers know that there is a new message they have not con-

sumed. When the sequence number exceeds the slots of the ring buffer, the sequence number is wrapped, and the disruptor begins at the first slot of the ring buffer.

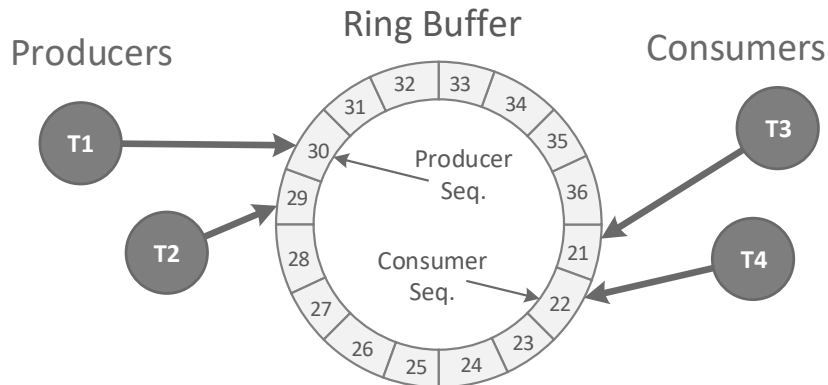


Figure 4.2: The disruptor concept

The sequences are one of the few points that require concurrent access within the disruptor. This concurrent access can be safeguarded with a lock on the increment operation of the sequence, which can be performed in a very short time. In case of the existence of only a single consumer or producer, this locking can even be entirely skipped. An advantage of the disruptor, in comparison to the blocking queue, is that a message in a slot can be processed not only by a single consumer, but by a chain of different consumers.

4.1.2 Distributed Processing Frameworks

The distribution of processing tasks to multiple nodes and their processor cores is a complex problem that can be handled with many already existing processing frameworks. The frameworks are distinguished by the type of processing they support, which is either stream-, batch- or micro-batch-processing. Depending on the use case, i.e., whether live- or historical data should be processed, one framework can have significant advantages over the others, especially in terms of latency and throughput. The following selection of frameworks is currently most popular for Big Data processing.

Apache Hadoop (Batch-based) Apache Hadoop is one of the first and most popular frameworks for the distributed processing of Big Data in a cluster and was initially published in 2006. It is an implementation of the MapReduce algorithm in combination with the Hadoop Distributed File System (HDFS).

The MapReduce algorithm was originally proposed by Google in 2004 [65] and consists of the *map* operation, which transforms the values of a dataset to multiple

CHAPTER 4. TOOLS AND LIBRARIES FOR BIG DATA

intermediate key-/value-pairs, and the *reduce* operation, which combines multiple intermediary values with the same key to a final value for that key. Since each of the operations can be independently run on a subset of the dataset, MapReduce can be executed in parallel on multiple nodes and is highly scalable. The HDFS, which is based on the Google File System (GFS) [66], is used by Hadoop to initially distribute the full dataset to all cluster nodes. As a result, each worker node can retrieve the data faster from multiple nodes over the HDFS or can even access the local copy of the data because it is itself part of the HDFS. Because of the initial migration of the data to a distributed file system and the type of operations that are applied to the data, Hadoop is primarily seen as a batch-processing system.

From today's view, Hadoop has various limitations that are related to its use of MapReduce and the HDFS. Firstly, the Map Reduce operations are limited in their functionality and it can be hard to express a task only with these abstractions. Secondly, Hadoop works in an inflexible and slow iterative manner, meaning that each operation is handled in a separate run that consists of reading from HDFS, processing with MapReduce and writing of the results to HDFS.

Apache Spark (Batch- and Micro-Batch-based) Spark is a new kind of processing framework that was initially proposed by researchers from the University of Berkeley in 2012 to overcome the limitations of Hadoop [67].

The Spark engine is based on the concept of Resilient Distributed Datasets (RDDs), i.e., intermediary processing states. An operation is performed on top of an existing RDD and the results are output to a new RDD. A complete task can, therefore, consist of a sequence of multiple RDDs and their operations. In contrast to Hadoop, the available operations are more comprehensive than *map* and *reduce* and the results of an operation are written to an RDD instead of into HDFS. In the end, the RDD acts as a cache which is either kept in memory or on the hard drive.

The various optimizations allow Spark to be up to 100x faster than Hadoop and let it be more flexible in its operations. Also, Spark comes in two versions to satisfy the need for batch- as well as stream-processing. The core of Spark is tailored to batch-processing, but there is an integrated streaming component, being called *Spark Streaming*, that works with micro-batches. In fact, Spark Streaming uses the existing implementation of the batch component but splits the incoming dataset into multiple smaller chunks beforehand. As a side effect of the combination of batch- and stream-processing in one framework, Spark can be easily used to create Lambda architectures, because the same analysis code can be used for the stream- and batch-layer.

Apache Storm and Trident (Stream- and Micro-Batch-based) Storm is one of the first stream-processing frameworks for Big Data and was created by Nathan Marz

CHAPTER 4. TOOLS AND LIBRARIES FOR BIG DATA

from Twitter. It was initially published in 2011 and was moved to the Apache Foundation in 2013.

The central concept of Storm is to distribute a *topology*, which is the terminology for a task's processing pipeline, with so-called *spouts* and *bolts* over multiple nodes. A spout represents a data source of a topology that continuously produces data and a bolt can be considered as a single processing step within the pipeline. Since Storm is a streaming framework, a topology is once started and then runs infinitely, if not explicitly stopped. As new data arrives in the spouts, it is passed through the whole topology. In comparison to Hadoop and Spark, a majority of the data is passed between the nodes via network instead of accessing data from HDFS. This is because the whole workflow is split over multiple nodes instead of distributing the workflow as a whole to multiple nodes.

The implementation of Storm already uses some of the technologies we have presented before. In particular, Netty is used for the communication between different nodes and the LMAX Disruptor is employed to pass data records from the network to the worker threads of a bolt and vice versa [68].

Trident is another processor that is integrated with the Storm release but is micro-batch-based with exactly-once processing. Trident makes use of the Storm core, but further abstracts the operations similar to SQL. Instead of expressing how something is achieved, it is expressed what has to be achieved in terms of sorting, grouping or aggregation [54]. As a result of the micro-batching and the abstraction of operations, Trident is claimed to be faster and easier to use than Storm.

Twitter Heron (Stream-based) Heron is a new stream-processing framework that is developed at Twitter and was published in early 2016. It is the successor of Apache Storm and is mostly backwards compatible with the Storm API.

Heron was completely reworked in its architecture in comparison to Storm. In the Storm architecture, a record has to pass multiple queues and threads before it reaches a bolt. In Heron, the number of threads is reduced to two, one that receives and sends records and another that performs the processing. Another objective of Heron is to improve scheduling, debugging and resource allocation of the workers. So, in comparison to Storm, Heron only handles a single task for each worker.

According to Ramasamy [69], the performance of Heron is significantly better than that of Storm and the management of topologies is much more comfortable. In experiments, the throughput of a Heron topology was around 10-14x higher than with Storm.

Each of these frameworks provides a comprehensive and flexible platform for analysis algorithms with a wide range of capabilities, e.g., fault tolerance, centralized management and adaptability to existing systems. The main point in which

these systems are different is their way of processing, whether they perform batch- or stream-processing or work on historical or live data, respectively. Although processing frameworks can simplify the handling of Big Data, there is also an important downside to be considered. The flexibility and adaptability make these frameworks susceptible to errors and potentially decreases the performance because of the many possible alternatives that need to be considered during processing.

4.2 Persistence

The persistence or storage of data creates a bridge between stream- and batch-processing. Based on the type of data and the analysis that is performed on it, special persistence frameworks can be used that support an analysis use case. In the following, a list of common storage systems with their individual capabilities is presented.

HDFS and HBase The Hadoop Distributed File System (HDFS) is a common distributed file system that is based on the Google File System (GFS) [66] from 2003. It provides scalability, reliability, and availability and allows to store huge files, which would be too big to be placed on a single hard drive, across multiple cluster nodes with commodity hardware.

A distributed column-based NoSQL-database system that works on top of the HDFS is HBase. It is based on the BigTable database initially proposed by Google [70] and is optimized for schema-less sparse data. It breaks up a row by its columns and stores each non-empty column with its value in a separate entry in the database. This allows a dynamic schema as well as an efficient compression, because empty values are dropped, and equal values can be combined. Each entry in HBase is indexed by its row, which allows fast access to all values of a row. A special characteristic of HBase is its immutability with immediate consistency, which makes it to a CP(Consistency,Partition-Tolerant)-system according to the CAP-theorem.

Cassandra Cassandra is another column-based distributed NoSQL-database that was inspired by Google's BigTable and Amazon's Dynamo. The data structure that Cassandra uses to store a row is very similar to that of HBase. However, it is possible to create secondary indices to access subsets of rows faster. In contrast to HBase, Cassandra is more focused on availability and only provides eventual consistency, which makes it to an AP(Availability,Partition-Tolerant)-system.

MongoDB This is a popular document-based NoSQL-database that tries to incorporate typical concepts of traditional Relational Database Management Systems

CHAPTER 4. TOOLS AND LIBRARIES FOR BIG DATA

(RDBMSs). It stores each data record as a JSON/BSON object and is not bound to a fixed schema. Indexing in MongoDB is not limited to a record's primary key. Instead, indices are supported on arbitrary attributes, enabling fast querying of data. MongoDB is implementing CP according to the CAP-theorem.

SAP HANA SAP HANA is a commercial in-memory data platform by SAP that is designed for real-time Big-Data processing. It combines an RDBMS with various tools for advanced data analytics, enterprise data management, and a dedicated application server. The core of HANA is the in-memory database engine. This engine operates purely in-memory and implements various optimizations on the hardware as well as software level to deliver an optimal processing performance.

On the hardware level, HANA leverages the available hardware to not only scale horizontally but also vertically, e.g., by making optimal use of main memory, CPU caches and supporting different processing architectures as Massive Parallel Processing (MPP) and Single Instruction, Multiple Data (SIMD) [50].

On the software level, HANA uses a columnar storage and an insert-only scheme. In the *column-based* approach, the rows of a table are separated by their columns and all values of a column are stored in a sequence. This is a shift from the traditional model, where all values of a row are stored in a sequence. Looking at common analytics tasks, also known as Online Analytical Processing (OLAP), it turns out that usually only a small number of columns is used for analysis, so that a columnar access can significantly improve performance. The *insert-only* principle is HANA's immutability concept. It means that each insert, update or delete on the data is represented as a new record with an implicit time-stamp. The immutability is not directly visible to a user but is internally used to manage data. To keep the memory footprint of databases small, HANA comes with a built-in mechanism, called merging, to combine multiple updates on one record to an eventual record. Due to its immutability and the provided immediate consistency, HANA is classified as a CP-system according to the CAP-theorem.

It can be observed that there is a development of storage systems to rather targeted solutions. Depending on the use case, there are DBMSs for documents, key-value pairs, and structured relational data. A large part of the systems is going into the direction of a schema-less NoSQL architecture. The scalability of these systems is mostly achieved with the distribution of storage and processing in a cluster. A special case in the group of distributed databases is SAP HANA. This RDBMS does not only scale horizontally, but also scales vertically by retaining the data in the main memory and making massive use of parallel processing.

4.3 Conclusion

Various libraries and tools support the analysis of Big Data. They can either be used as a basis for a more complex Big Data analysis infrastructure or can themselves be used to build new analysis tools.

A group of readily available tools are distributed processing frameworks, such as Hadoop, Spark, and Storm. They can simplify the handling of Big Data by providing a platform to host analysis code. However, it has to be considered that their high complexities are prone to bugs and their adaptability causes performance loss. In other words, for use cases that require high performance and scalability, it might be worth looking into a customized solution, e.g., under the use of the presented IPC solutions.

On the persistence side, there is a large variety of different systems that have their special field of application. While most of the systems are focusing on the distribution to a cluster only, the SAP HANA also enters the way of vertical scaling with its in-memory approach. This development is interesting, because rather complex analysis, such as machine learning and general statistics, can be performed on a single machine without extensive data exchange over the network.

Part II

**Big Data Security Analytics
on Event Data**

Chapter 5

Approaching Enterprise Security with Big Data Processing

The protection and security of our computer networks are becoming increasingly important, as more and more of our personal data is moved into the digital world. Already, the amount and sophistication of attacks are dramatically rising, as attackers are discovering the value of our digital data. In addition, to protect networks nowadays, one has to deal with highly complex and heterogeneous infrastructures. While small networks could be handled by a team of a few administrators in the past, the complexities of modern network infrastructures are easily overwhelming all available resources.

One problem is the limitation of existing attack detection tools to only a small part of the overall infrastructure. They are missing the big picture of what is going on in a network and are not able to correlate multiple suspicious activities for a multi-step attack. Furthermore, these tools are often tailored to support the detection of well-known attack patterns but are not capable of detecting novel attack techniques.

A solution to the above problem is already in sight, but some serious issues need to be addressed. Already today, there is a large variety of sensors that are monitoring and recording each activity within and outside of our networks and there are even more data sources that provide general context information about all imaginable facets of a network. Each step of an attacker and the details of his malicious activities can be traced. Nevertheless, due to a large number of activities happening in- and outside of a network, these traces are going under in the flood of event data and make it a challenge for traditional tools to detect attacks.

The development of Big Data technologies can be a way out of the misery of log management in enterprise networks. It can enable the analysis of extensive, mostly still dormant, data sources and makes more sophisticated attack detection possible. In the following of this chapter, we first show the current state of security systems and their methods of attack detection and then present our approach on how security

can be handled in growing networks with the help of Big Data processing.

5.1 Current State of Attack Detection

The detection and prevention of computer attacks is a topic since the late 1980s with the appearance of the Morris worm. Since then, network security measures have evolved from perimeter security, i.e., building a barrier around the network, and individual computer protection to security suites that monitor each activity in a network. It can be observed that each development step of these measures has incorporated more data into the analysis because it is required to detect the increasingly sophisticated attacks. The following sections introduce the categories of attacks and then follow with existing detection systems and their incorporated data sources.

5.1.1 Attack Categorization

A computer network, be it small or large, that is connected to the public Internet is a potential target for cyberattacks. How significant this number of network attacks is, can be seen in reports from various IT-security organizations [71, 72, 73]. They are pointing out to thousands of attack incidents every year in enterprise as well as government networks. The attacks security operators of these networks are confronted with, can be categorized into two groups, *simple single-step* and *advanced multi-step attacks*.

5.1.1.1 Simple Single-Step Attacks

The more significant parts of attacks on the Internet are unsophisticated and automated, such as mass-phishing campaigns, automatic account takeovers with default credentials, or scripted SQL injection attacks. These *simple attacks* are mostly conducted with attack tools that are downloadable from public websites and are easy to use for everyone. Although these tools are publicly available, they can still be highly sophisticated if they were written by skilled professionals. Consequently, unsophisticated attackers can exploit complex vulnerabilities and cause severe damage.

The attack methods of simple attacks are usually well known and are limited to the exploitation of a single target. Because of the latter property, simple attacks are also referred to as *single-step attack*, because they only use one attack technique to exploit a target. As the employed attack methods of simple attacks are well known, there are existing protective measures that can be deployed against them. However, these measures are often not employed over the entire network, so that attackers can find remaining loopholes in the protected network.

5.1.1.2 Advanced Multi-Step Attacks

As companies and governments are protecting their networks better and are putting increasingly valuable information and systems into their networks, attackers are upgrading and are putting more effort into their attacks to gain unauthorized access to these networks and their confidential data. Such advanced attacks consist of multiple attack steps that gradually lead the attacker deeper into the targeted network. Each attack step uses its dedicated attack technique and has one specific target. The employed attack techniques can either be similar to a simple attack or are employing customized attack tools or exploits only adapted to the selected target. Because of the multi-step character of the attack, advanced attacks are also referred to as *multi-step attacks*.

One popular type of a more sophisticated attack is the so-called APT [7, 8, 74], which is known for its highly motivated and organized threat actors, such as cyber armies of governments [9] or professional spy companies [75], that are employing advanced, customized attack tools and techniques with the goal to stay as long as possible in the target environment to steadily exfiltrate valuable information. Figure 5.1 shows a scenario for a multi-step attack, as it could be part of an APT attack.

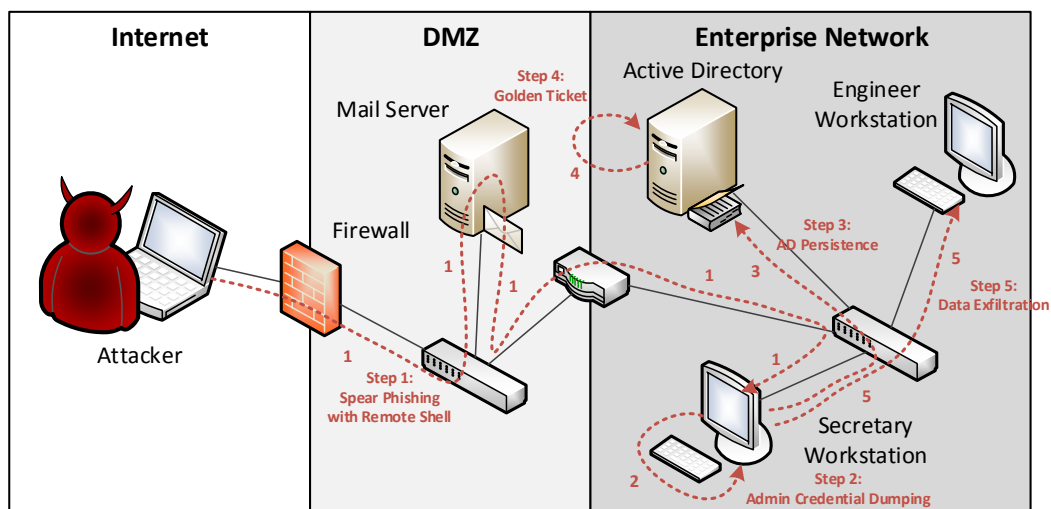


Figure 5.1: Example of a more complex multi-step attack

At the first step of the attack, the attacker sends a phishing email message to the secretary of a leading employee. The email address for this secretary could be extracted from the public website of the targeted enterprise. With the opening of the message, a malware is executed and a remote shell from the secretary's computer to the attacker is established. As a result, the attacker gains access to the secretary's computer and, thus, access to the enterprise network. With access to the sec-

retary's email software, the attacker convinces an administrator of the network via spear phishing to log into the secretary's computer. The attacker then obtains the domain credentials of the administrator by using a tool like Mimikatz. In step 3, the attacker uses these administrator credentials to get access to the central Active Directory server of the company, which then allows him to issue a golden ticket with Mimikatz in the fourth step. With this golden ticket, the attacker can access the workstation of a leading engineer in the company, which gives him access to confidential documents of the newest developments of the company.

5.1.2 Attack Detection Systems

Over the years, many types of security tools with different strategies for the detection of attacks have been developed. A categorization of these tools is possible with regards to their deployment and their detection approaches.

5.1.2.1 Intrusion Detection Systems (IDSs)

This group of systems is performing their detection directly on the local system by monitoring all ongoing activities. In the case a malicious activity is detected by an IDS, it produces an alert that can be consumed by a security operator or another system. Generally, there is a distinction between IDSs that monitor an endpoint (HIDS) and the ones that monitor an intermediary network node (NIDS). Due to the fact that an IDS is only deployed on a single system, it would only be able to identify a simple single-step attacks.

Host-based IDSs (HIDSs) This version of an IDS is deployed on a network endpoint (host) and monitors all kind of operating system resources. A popular representative of this group are anti-virus programs that detect malicious software by searching for known malware signatures. A new type of HIDS is the so-called endpoint protection solution that monitors the behavior of running programs to identify malicious activities.

Network-based IDSs (NIDSs) The Network-based Intrusion Detection System (NIDS) is deployed on an intermediary network node and monitors network traffic that passes by. It is primarily used to detect remote attacks on network endpoints in an intranet and not on the deployed device itself. The monitoring of the network traffic typically covers the inspection of packet headers as well as the payload of the packets and its effects on the target. This is a slight but important difference to the firewall, which is only inspecting packet headers with IP addresses and ports to

decide whether it is allowed to pass through. However, some newer generation firewalls, especially application layer gateways/firewalls, can be counted into the group of NIDSs.

Intrusion Prevention Systems (IPSs) A special form of IDS is the so-called Intrusion Prevention System (IPS). In contrast to an IDS, it additionally mitigates and prevents attacks as they are detected.

5.1.2.2 Security Information and Event Management (SIEM) Systems

A SIEM is a combination of Security Information Management (SIM) and Security Event Management (SEM). On the one hand, SIM specifies the management of all available security-related information, such as asset, alert and vulnerability information, in one platform. On the other hand, SEM specifies the collection of all available event data in one centrally deployed system. A SIEM is centrally deployed in the network and, according to Gartner's initial definition, provides real-time event management and historical analysis of security data from a wide variety of heterogeneous sources [76]. In practice, a SIEM mainly retrieves event data from networking hardware, such as routers and switches, operating systems, applications, and security solutions, such as firewalls and IDSs. Furthermore, information about the network infrastructure and common security threats, so-called cyber threat intelligence (CTI) being described later in this chapter, can be incorporated into the SIEM for more advanced analytics. With all this information at hand, a SIEM system can observe single-step attacks at a specific point in the network and multi-step attacks involving multiple network nodes. A drawback of SIEMs is their limit to only detect attacks without the option to mitigate them.

5.1.3 Detection Methods

There are two main techniques on how security investigators and automated security systems detect attacks [77].

5.1.3.1 Signature-based Detection

An attack signature is a formal description of an attack pattern from previously observed attacks. The signature-based detection, also referred to as *misuse detection*, uses these signatures to identify known attack patterns on the monitored system. In practice, there are many different forms of signatures, from simple indicators, over heuristics and neural networks to multi-step signatures that describe complex attack strategies.

Indicator Of Compromise (IOC) One particular trend in the development of signatures are so-called indicators of compromise (IOCs), which describe activities or artifacts that were observed by third parties during an evident attack. If someone observes the specified activity or artifact in the own network, then this is an indicator that the same type of attack is ongoing as previously observed by the third party. Examples for IOCs are a single IP address, a URL or a file hash.

5.1.3.2 Anomaly-based Detection

An anomaly is the deviation of the system behavior from the norm. The anomaly-based detection assumes that an uncompromised system is the norm and that an anomaly in the system behavior points out to malicious activity. Before a security system can use the anomaly-based approach, it must either learn the current state via machine learning or have the normal behavior pre-programmed. As the anomaly-based detection is not based on static attack patterns, it can reveal attacks that were previously unknown. However, the approach is also known for delivering false-positive results, because, in fact, not all anomalous behavior is necessarily malicious.

5.1.4 Data Sources for Attack Detection

The presented security solutions are working on different types of data sources to reveal attacks. Each data source can deliver a different view on the current security status. We are distinguishing the following three groups of data sources.

5.1.4.1 System Parameters and Behavior

This is the traditional data source for attack detection and covers information about the resources and behavior of the local system. On a typical system, there are resources like files, processes and network sockets and behavior like API calls, command-line calls that can be monitored to reveal malicious activities.

A HIDS is a good example of a tool that heavily relies on monitoring of the local system. For example, it observes the creation of processes and network sockets to detect the execution of malicious programs and their connection to a remote-control server. A NIDS also monitors the network traffic passing through the host system.

The system behavior and system resources are volatile in nature. Security tools working on this data are acting on live data and alert the user as soon as an attack was detected. Once an alert has been raised, it is not possible to revisit what other activities took place at that time.

5.1.4.2 Event Logs

Event logs are a solution for the previously described volatility problem of system information. The idea is to create events for each observed activity or status change in a hardware component, an operating system or software application, which are all collected in a system-local event log. This event log acts as a repository of past activities and records important activities as they occur. In comparison to volatile system information, event logs allow to perform historical attack detection and enable a detailed investigation of past incidents.

To create a common understanding, we are using the following terminology throughout this thesis.

Event This is an activity that has taken place and has been observed by a system.

Event Record This is the digital representation and description of the observed event. Depending on the context, we also commonly refer to an event record as *event*, *log event*, *log entry* or *log message*.

Event Log An event log is a list of records of events that have been observed over a longer period.

5.1.4.3 External Cyber Threat Intelligence (CTI)

External intelligence is a new kind of data source for attack detection. Instead of only relying on information that was produced by the own systems, information on threats from third parties is included in the detection process. These third parties might already have experienced a new kind of attack that is not widely known, yet. This advantage in knowledge of third parties can protect another network from similar attacks. In the following, we distinguish two types of external knowledge.

Known Weaknesses One type of external information is intelligence about potential weaknesses being exploited in the wild. Knowing these weaknesses can give defenders an advantage by installing dedicated protection mechanisms against these weaknesses. Examples for knowledge bases of weaknesses are vulnerability databases (HPI-VDB¹), platforms for extended port and service scans (Shodan²) and directories of breached services or accounts (Zone-H³ and the later presented HPI-ILC).

¹HPI Vulnerability Database - <https://hpi-vdb.de>

²Shodan - <https://shodan.io>

³Zone-H - <https://zone-h.org>

Known Attack Indicators IOCs are another type of external intelligence that consist of simple observable activities or artifacts, such as IP addresses, domain names, URLs and file hashes.

5.2 Enterprise Security as Big Data Challenge

Enterprises and organizations are facing a difficult challenge by keeping their networks clean from script kiddies, threat actors and other types of attackers. Especially the fact that many larger networks are encompassing thousands of devices, some of them even directly connected to the Internet, complicates the monitoring of a network in its entirety. Each device that is added to a network needs to be monitored because it creates another potential attack vector that can be leveraged by an attacker.

To establish a protection against cyberattacks, an enterprise can set up security software that monitors their networks. However, there are also limitations in the capabilities of traditional security software, i.e., mainly IDSs and firewalls, in the monitoring of complex networks. In particular, these solutions were not designed to operate in a network, but on a single machine. Thus, the correlation of monitoring results, i.e., alerts, from individual segments of a network is not easily possible. A solution that addresses this shortcoming are the relatively new SIEM systems, which are correlating the monitoring results from multiple machines by gathering event logs of applications, systems, and security solutions, i.e., IDSs and firewalls, from all over the network. By this, they are becoming the single source for all kind of security-related investigations. Some new generations of SIEMs are even integrating CTI to reveal IOCs in their logs. The SIEM development is an important step to be able to detect threats in all parts of a network and to identify and trace sophisticated multi-step attacks. However, also with SIEMs, the reality shows that existing solutions have not reached the point of development where all devices in a network can be monitored simultaneously. Current SIEMs are barely able to monitor the most critical parts of a network, such as operating servers [31], because they are overwhelmed by the sheer quantity and diversity of the data to be processed. To get an understanding of the nature of this data, the following subsections give a more detailed picture of the volume/velocity and variety of security-related information in large networks.

Volume of Security-Related Information Event logs have become one of the most valuable resource for the tracking of attacks in networks. Many companies have realized this and are now building up their log management and are gradually increasing the number of log sources to monitor each corner of the network. According to a survey [31] conducted by the SANS Institute in 2014 with 522 participants, around 97% of the companies already collect logs, and 85% see the detection/tracking of malicious behavior as the main reason to do so. In addition, the respondents reveal

that the Big Data nature and the correlation of the logs are their main challenges for log management. Looking at a typical company network, the reason for these challenges becomes clear. Even small networks are having dozens or hundreds of devices that need to be incorporated into the monitoring. In larger enterprise networks, the number can grow to more than 100 000 active devices and more than 10 000 operating servers. Considering that each of these devices can generate logs, then the sum of all collected logs in a network can easily exceed a billion log events a day. Processing and storing all of them simultaneously seems to be an insurmountable task.

Examples from the industry show that a billion events are not even the end. Gartner, in its annual “*Magic Quadrant for SIEM*” [15] report, defines that a very large SIEM deployment has a sustained event rate of ≈ 2.1 G evts./d (25 k evts./s) that needs more than 50 TB of backing store. Furthermore, there are reports from companies like Hewlett-Packard (HP) [78] that claim to produce security events with a rate of 1 T evts./d (11.5 M evts./s) in a network of 300 000 employees. With their security solution, HP is merely able to process a tiny fraction of around 3 G evts./d of these. Also Barclays [79] and Goldman Sachs are bringing up event rates of multiple G evts./d and log volumes of a few TB/d for their current deployments. Barclays was able to process only a small fraction of 500 M evts./d with their SIEM, too. The fact that companies are not able to process all their logs can also be deduced from the SANS survey. According to them, only 50% of the respondents monitor desktops/laptops and less than 20% are monitoring mobile devices, although breach investigation reports show that many data breaches are caused by phishing on employee devices [73].

Apart from event logs, which make up the most substantial part of security-related information, there is also a significant amount of CTI that a SIEM can use to find known threats. According to a survey [80], around 55% of the companies are using their SIEM to correlate events with CTI. The demand for CTI has also led to a wide range of CTI providers, and many security companies are providing their own threat feeds. However, the challenge in the handling of CTI is not necessarily the huge number of indicators, but the fact that all these indicators need to be compared to all incoming events. Therefore, already a few million indicators can overwhelm the processing of the whole SIEM. Some of the more influential representatives on the CTI market, such as IBM X-Force⁴, Anomali ThreatStream⁵ or AlienVault OTX⁶, already have around a million indicators each in their data feeds.

Variety of Security-Related Information Another challenge for SIEMs is the large variety in representation, content, and exactness of security-related informa-

⁴IBM X-Force - <https://www.ibm.com/security/xforce/>

⁵Anomali ThreatStream - <https://www.anomali.com/platform/threatstream>

⁶Alien Vault OTX - <https://www.alienvault.com/open-threat-exchange>

tion [31]. As long as the collected data is not available in a common format, i.e., the data is normalized, it cannot be further processed and analyzed, which means it is not usable for security investigation. The reason for the large variety is mostly due to heterogeneous data sources and an insufficient standardization of the formats that represent the information.

Event logs are again the primary type of security-related information with a high variety. According to Gartner's annual report, a typical SIEM has to handle 300 event data sources for a small deployment and 900 data sources for a very large deployment [15]. Any of these data sources could implement a different event format. At the moment, there are mainly two formats that are used across multiple data sources, i.e., *Syslog* for the UNIX world and *Windows Event Log* for the Windows world. Both formats have fundamental problems. Syslog is more of a format wrapper and is only semi-structured. It represents some meta-information, such as time and source, in a structured way, but leaves the description of the observed activity, i.e., the message, in a raw unstructured form. Windows Event Log is also a wrapper for Windows-based log events and is tailored to the Windows ecosystem. Other approaches to bring the normalization of event formats forward, such as Common Event Format (CEF) or Common Event Expression (CEE), have never brought it to wide use.

The situation that there is a wide variety of event formats that do not use a common data model, makes the work of SIEM systems extremely hard. A SIEM that wants to handle all these logs needs to implement special rules that can extract and map relevant information to a common model. Unfortunately, currently existing SIEM systems are not able to handle this variety and only focus on the structured information of a few common formats, such as Syslog. However, most of the unstructured information, which is considered the most valuable for security investigation, remains in its original form.

The diversity of data is not only limited to event logs but can be found for CTI, too. Similarly to event logs, there are many different sources for threat intelligence that represent their indicators in different formats. Luckily, the biggest providers are delivering their IOCs in either *Structured Threat Information Expression (STIX)*, *Cyber Observable eXpression (CybOX)* or *Open Indicators of Compromise (OpenIOC)* [80]. However, some smaller sources are not adhering to these standards and are using Comma Separated Values (CSV) or plaintext lists. Furthermore, the usage of the standards does not always guarantee that all data is available in a structured form, because some vendors put detailed information in descriptive fields. As a result, SIEMs that want to incorporate CTI need to support the three common standards and be prepared to further normalize information.

Based on the above points, the processing of security-related information in a large enterprise network can be seen as a Big Data challenge. In essence, there is a huge

volume of data, i.e., event logs and CTI, that needs to be collected from a wide range of sources and this data is mostly semi- or unstructured and is not represented in a common data format. SIEMs in their current form are not able to process such big security data, either because they are not fast enough to handle the event throughput, or they cannot comprehensively normalize the incoming data.

The examples from the industry show that current SIEM systems are even far away from handling all upcoming data. While Gartner [15] defines a SIEM with a sustained throughput of 25 k evts./s big enough for very large deployments, HP has to cope with throughputs of up to 10 M evts./s. Other reports from the industry reveal that even a throughput of around 50 k evts./s cannot be handled with their current SIEM deployments, although they are only including event logs from the most critical systems and leave out all kind of activity on desktops and laptops. Including all event logs from such a network could easily increase this throughput to over 1 M evts./s.

Surely, something has to be improved for SIEM systems to cope with these issues. Presumably, the current generation of SIEMs is not able to process all this data because they are not making use of already established Big Data processing paradigms. Our goal is to show that a SIEM with these techniques is indeed able to handle all data in a large network.

5.3 A Big Data SIEM for Enterprise Security

Current SIEMs are not capable of handling the Big Data challenge well, because they are much designed like traditional software systems, meaning they are processing data sequentially, are using disk-based single-node RDBMSs for data storage and analysis, and are expecting a common data format for incoming data. Together, these shortcomings lead to an overall limited processing performance as well as inefficient access to relevant event information. As a result, today's SIEMs are only able to handle a small fraction of security-related data and cannot perform further analysis because they do not adequately normalize their data. Already, there are technologies and approaches available to address these typical Big Data problems, but many vendors are not taking the necessary steps to integrate them into their products.

In the following, we propose approaches and an architecture for a prototypical real-time SIEM system that makes use of existing Big Data technologies to handle security-related event data for enterprise networks.

5.3.1 System Approaches

The main objectives of such a Big Data SIEM are high-performance event processing and the preparation of event information for further complex security analytics. In addition, the SIEM should not only be able to gather event data at a central place, as

many existing SIEM solutions already do, but should come with established attack detection methods that are known from IDSs, i.e., signature- and anomaly-based detection as well as machine learning (ML) approaches. The strategy to achieve these objectives is to combine the concepts of Big Data processing, which were presented in Chapter 3, with optimized data models and algorithms for security-relevant event data. In particular, the following optimization strategies are leveraged for the system.

Processing Workflow The high throughput of incoming data is one of the biggest challenges of real-time event processing in enterprise networks. We are addressing the high throughput of events with a specialized Big Data processing workflow that is inspired by the general workflow presented in Section 3.2.3. It comes as an extension to the well-known ETL process and as a slight variation of the general-purpose processing architectures, i.e., *Lambda* and *Kappa*, which is more optimized to the requirements of event processing. In particular, our workflow is designed for parallel processing and distinguishes between a *preparation phase* where incoming data is collected, pre-processed and persisted and an *analysis phase* where the prepared and persisted data is analyzed in bulk.

We consider the massive parallelization of the processing workflow as one of the key performance factors because multi-core CPUs and fast network links are becoming the standard. Thus, our parallelization incorporates multi-threading for vertical scalability and network-based task distribution for horizontal scalability. The system is flexible to either be on a single high-end machine or a range of network-connected low-end machines. Today, many distributed *processing frameworks* are available to parallelize processing tasks multiple processing nodes, such as the solutions mentioned in Section 4.1.2. We want to make use of these distribution frameworks in our systems and see whether they can push the performance of a SIEM to the necessary levels. As an alternative to these frameworks, we also want to evaluate whether the processing performance can be further improved by tailoring the parallel processing paradigms to our concrete use case of a given event structure. One approach we want to review in particular is *disruptor-based multi-threading*.

Data Normalization In large company environments, event data originates from a broad range of heterogeneous event sources that produce highly disparate event formats. As typically with Big Data sources, this information is semi-structured or even unstructured, which makes automated processing difficult. At the moment, many SIEM systems are indexing unstructured event data they receive and perform a rudimentary normalization that extracts important meta-information, such as time or data source. In fact, these SIEMs are like a large and central event repository where a security operator issues text searches. Unfortunately, the possibilities for further analysis on indexed data is limited, because a majority of relevant information is hidden

in unstructured parts that can be queried with text searches. Advanced techniques, like field-based correlation or machine learning, are simply not feasible. A possible reason, why SIEMs do not perform more accurate normalization, could be the high processing overhead of a comprehensive normalization. Current systems would not be able to perform a full normalization with high event loads.

As part of our SIEM system, we concentrate on the full and fast normalization of events, which means that all relevant event information is extracted and represented with common data fields. This full normalization has the effect that all relevant event information can be accessed and stored more efficiently than unstructured data. In the case of data access, individual event properties are directly addressable and can be correlated with each other. As the same type of information is always represented with the same normalized field, we can always refer to this normalized field to access a type of information in all events. In the case of storage, the normalization results in fully structured information that can be accessed faster and could even be compressed at a high rate, depending on the complexities of the event fields. The described full normalization goes far beyond what is currently implemented in existing SIEMs and allows entirely new ways of attack analysis. For example, an attack detection algorithm can correlate multiple security-related events by their source and target IP address to perform a root cause analysis. To solve the throughput problem that many SIEMs have, we are integrating the normalization into the pre-processing step of the workflow that operates on multiple events simultaneously. In addition, we are applying optimizations to the normalization algorithm, which consider the specific hierarchical structuring of many event formats.

In-Memory Data Persistence A bottleneck in Big Data systems is the data persistence because all new data has to be persisted as soon as it arrives, and complex analytical functions potentially need to read the majority of past data. Existing SIEM solutions, such as Splunk [81], ArcSight [82], fluentd [83], and IBM QRadar [84], are relying on distributed, schema-less and disk-based databases, which are either self-developed or open-source solutions, such as HDFS, MongoDB, Elasticsearch, and Cassandra. While these persistence solutions allow scaling the capacity of storage easily, a study by Rabl et al. [85] shows that these systems have problems in their I/O performance in general and with their write performance in particular.

For our SIEM, we want to try a new persistence approach that leverages the normalized and structured event data and enables its faster access for analytical purposes. Looking at the persistence methods from Section 4.2, then the in-memory approach seems to fit this requirement the best because all persisted data is immediately accessible, and the data can be stored with a schema that can hold the structured event information. Especially with the growing size of main memory in today's systems, live and large parts of historical event data could be kept in the main memory. In case

the event data exceeds the size of available memory, it is moved into a distributed storage, such as HDFS. *SAP HANA*, as currently one of the furthest developed in-memory database, comes with advanced analytical capabilities, such as machine learning and graph analytics, operating directly on the database without prior exports. Thus, we consider *SAP HANA* as a suitable candidate to implement the primary persistence for our SIEM.

IDS-based Attack Detection The eventual idea of event collection and analysis is to reveal and prevent ongoing attacks on an enterprise network. In the industry, security solutions are mostly focused on the searching and browsing of event data, but the step to an automated reporting of incidents or threats is missing. This could mainly be grounded in the Big Data properties of event data. As a matter of fact, our previously presented approaches have already paved the way for the handling of such characteristic event data, so that we can now go the next step to an automated detection of malicious behavior in event data.

Automated attack detection with signature- and anomaly-based techniques is traditionally performed in IDSs, which mainly operate on system behavior and resources of a single machine. We propose to combine the advantages of SIEM and IDS into one system, which, foremost, means that the detection is applied to event data.

This has the major advantage that suddenly not only attacks on each system in an enterprise network can be detected, but also complex attacks spanning over a whole network. In addition, in contrast to normal IDSs, the operation on event data allows revealing malicious activities as they occur or have happened long in the past, assuming that the event data from this time is still at hand.

5.3.2 System Architecture

Now that different processing approaches have been presented, we propose a SIEM architecture that combines all these approaches into one system. This new SIEM is called *Real-Time Event Analysis and Monitoring System (REAMS)* and alludes to the English word *reams*, meaning a massive amount of something. An overview of the core components of REAMS is depicted in Figure 5.2.

The core system is represented by the large box in the middle of the picture. At the outer boundary of that box, there are inputs and outputs of the system. The primary input (left) into REAMS is from data sources within the monitored network infrastructure. It encompasses event data from different kinds of sources and additional infrastructure information, such as installed software and registered users, collected by dedicated host agents. Another input (bottom left) is *external threat intelligence*. It provides the system with security-related information that is utilized for event annotations and incorporated into the detection of attack patterns and anomalies. A

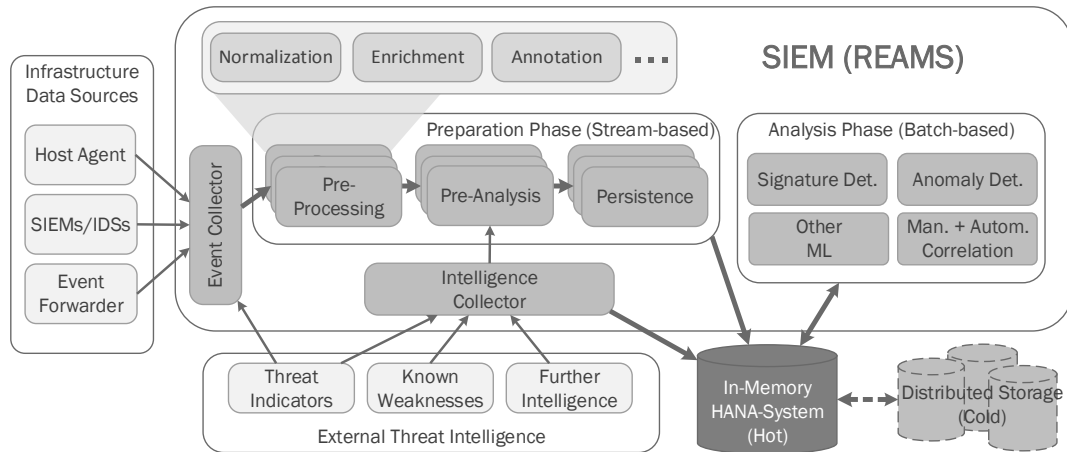


Figure 5.2: REAMS architecture being derived from our Big Data workflow

small fraction of this intelligence, namely event information that directly affects the monitored infrastructure, is fed as event input into the system. The two database systems (bottom right) act as the storage system for the SIEM. The focus of this storage system is the HANA platform as a hot storage, which can perform event storage and analysis directly in main memory. Optionally, a distributed DBMS can be attached to the HANA platform for historical data and data that exceeds the size of available memory.

At the core of REAMS, the previously mentioned processing workflow is implemented, which means there are a stream-based preparation phase and a batch-based analysis phase which are capable of processing multiple events in parallel either on the same machine or on a processing cluster.

Within the *preparation phase*, the event data goes through three steps, i.e., *pre-processing*, *pre-analysis* and *persistence*. The *pre-processing* prepares the event data to be easily consumable for analysis algorithms and covers *normalization*, *enrichment*, and *annotation* of the original event. The *pre-analysis* step comes directly after the *pre-processing*. It performs some preliminary analysis on the normalized and structured event information, such as single-event signature detection or the counting and grouping of events by selected criteria. However, it should be noted that this analysis must be fast enough to handle the incoming streaming throughput. In the last step, the normalized event data is persisted to the storage system, i.e., primarily the HANA platform, and made available to the *analysis phase*.

Within the *analysis phase*, the persisted and structured events from the storage system are correlated with each other or with external threat intelligence. As primary methods of analysis, we are relying on the signature- and anomaly detection as typical methods of IDSs as well as additional ML and data correlation approaches.

5.4 Conclusion

In this chapter, we have presented the current state of attack detection in general and enterprise networks in particular. While there are many different solutions to detect attacks on individual systems, there are only limited solutions for attack detection in the entire network. As networks and attacks are becoming more complex, it is crucial to correlate security-related information from as many sources as possible to get a complete picture of the activities in a network. We have analyzed the challenges that are related to the collection from such sources and the correlation of their data and conclude that enterprises are confronted with a Big Data problem when it comes to their security. As a solution to the problem, we propose approaches that can deal with the challenges of analyzing the activities in enterprise networks and come with an architecture that incorporates the introduced approaches into one system.

In the following chapters, we describe the details of our approaches and present how they interact with each other to perform attack analysis in enterprise networks.

Chapter 6

Stream-based Processing of Big Event Log Data

Related Publications

- *Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, and Christoph Meinel. “Hierarchical Object Log Format for Normalisation of Security Events”. In: Intl. Conference on Information Assurance and Security. 2013 [26]*
 - *Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Runtime Updatable and Dynamic Event Processing using Embedded ECMAScript Engines”. In: Intl. Conference on IT Convergence and Security. 2014 [29]*
 - *David Jaeger, Feng Cheng, and Christoph Meinel. “Enriching Normalized Security Event Logs for Deeper Security Analytics”. 2018 [30]*
-

The processing of security-related data in enterprise networks is considered a Big Data problem because it fulfills multiple of the well-known 7V's. Currently, existing security products, in particular SIEM systems, are not able to handle this challenge well in their current state of development. Because of these shortcomings, we have proposed an architecture for a new type of SIEM in Chapter 5 that utilizes Big Data paradigms and our specific Big Data workflow to better handle volume, velocity, and variety of security-related data. In this architecture, we are employing our proposed Big Data workflow (see Section 3.2.3)) that separates the processing into a stream-based *preparation phase* and batch-based *analysis phase*. In the following of this

chapter, we are diving deeper into the stream-based phase of this architecture and concentrate on the processing of event logs as security-related data. The full stream-based workflow is shown in Figure 6.1.

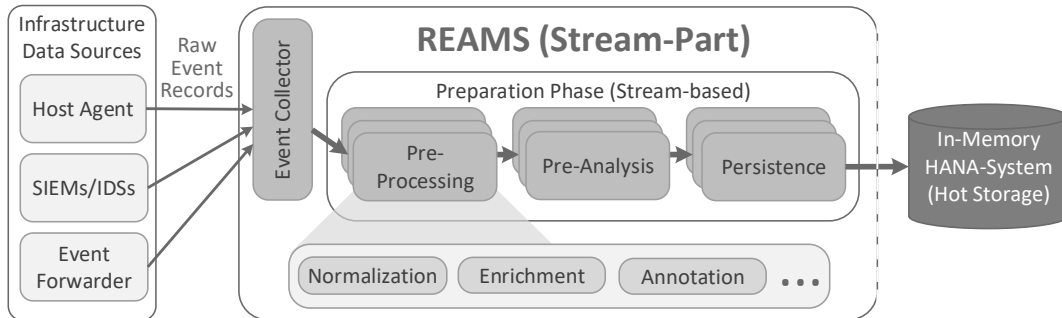


Figure 6.1: Stream-based part of our SIEM described in this chapter

The workflow starts with the collection of the events from various sources. In the following step, the raw event data from the source is transformed into a common event format by normalization, enrichment, and annotation. Afterwards, the normalized event undergoes some quick checks that deliver some additional value to the event, such as a comparison to known IOCs. In the last part of the workflow, the event is persisted to the database, where it can be further analyzed in the batch-processing phase, which is further described in Chapter 9.

As a goal of the stream-based phase, we want to fully normalize and persist log events with a throughput that goes far beyond the typical 50 k evts./s of currently existing SIEMs and be able to handle the log management in large enterprise networks with a single SIEM. Furthermore, the performed normalization has to be comprehensive enough to perform complex attack detection, such as multi-step signature detection, anomaly detection, and further machine learning.

6.1 Data Collection

As the first step in the processing workflow is the collection of data from various data sources in the enterprise network. In larger networks, the number of sources can go into thousands or even hundreds of thousands. As the goal is to get an overview of all potentially malicious activities in the network, it is desirable to capture events from all these sources.

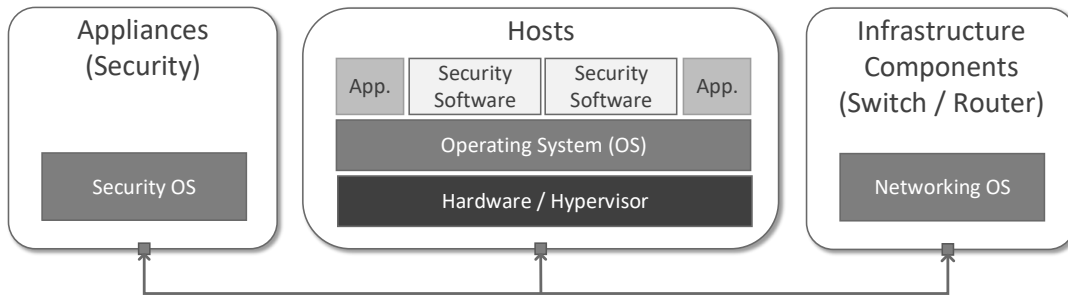


Figure 6.2: Levels of event sources

6.1.1 Types of Event Log Sources

Event logs are produced everywhere where actions are performed and observed. Therefore, sources of event data can be found at various locations and on different levels of abstraction within a network. On the highest layer, event logs can be found on systems like *hosts*, *infrastructure components* and additional *appliances* that are ensuring the security of the network. On each of these systems, there are multiple layers of event sources, like *hardware components or hypervisors*, the *operating system*, *applications*, like web, file or email servers, and in particular *security solutions*, such as IDSs, firewalls and other SIEM systems [86].

6.1.2 Existing Methods of Collection

The methods in which sources provide their logs are as different as the sources themselves. Altogether, there are three methods to obtain logs from a source.

Agent (Push) An agent is deployed on the system where the source is located. It forwards the logs it reads directly from the source, such as from a file or a local database. This method is most flexible but requires some effort for setup. Also, there must be a server to receive the logs on the SIEM side.

Client (Pull) The client remotely connects from the SIEM to the log source and receives all logs over an API. Obviously, the source must provide an API and the client must implement the same protocol. Some typical sources that provide such APIs are other SIEMs or SNMP-devices.

Server (Push) This is the most convenient method of log retrieval, but also the most limited. The server waits for incoming events that are sent by a corresponding log client at the source. The most popular server implementation is a Syslog server [87].

Many existing SIEM products come with a Syslog server and have some custom clients to read logs directly on the source. In addition, there is usually a functionality to forward all arriving logs to another Syslog server.

6.1.3 Proposed Log Collection

In order to receive as much data as possible, we propose to use a Syslog server and another log server with a customized communication protocol. The customized server is used to receive logs from a custom agent or a LogStash [88] agent. In addition to the two servers, there are modules to receive logs from Splunk [81] and Graylog [89]. By supporting the ingestion of logs from other SIEMs, we can easily connect our system to an existing log management environment and test on productive log entries.

6.2 Basic Normalization of Raw Events

An event record that arrives at the collector is in a raw state, meaning that its data representation and context are unknown and that the information it bears cannot be extracted straightaway. To be able to extract further information from a raw event, the SIEM has to analyze the event record's representation and determine the data format. As soon as the format is known, the SIEM can start to *normalize* the provided information to a unified data model that is easily consumable.

Currently existing SIEMs have major shortcomings in the normalization of events because they are not able to perform a thorough analysis of event formats in real-time. As a consequence, less performant SIEMs only support a small set of very popular event formats, such as Syslog, or are not able to provide their extracted information in a comprehensive data model. Furthermore, formats like Syslog are just wrappers for more elaborated formats and are only providing rudimentary meta-information about the event. The extraction of essential information about the activity, which we call *deep normalization*, remains out of reach for current SIEMs.

In the following, we introduce the basics of event representation and then propose the deep event normalization used within our SIEM, which includes the determination of the event format and the extraction of relevant information into our unified event format OLF, as presented in Section 6.2.2.2.

6.2.1 Event Representation

An event itself is an abstract concept that has no particular representation. To serialize an event, e.g., as a record in a file or database or to transfer it to another server, it has to be transformed into a binary or textual representation. In addition, if multiple event records are combined in an event-stream, as in a log file or network transfer,

additional separation of records has to be implemented. Following variations of event representations are typical.

6.2.1.1 Single-Line vs. Multi-Line

Events that are represented in a textual form come in two variations. They are either limited to a single line or spread over multiple lines. The single-line representation is the most prevalent form of representation and is preferred because it is easier to parse and interpret. Usual text editors and many programming libraries already come with a functionality to distinguish lines in a file and can present them accordingly. Multi-line event records are mainly used in two special cases. Firstly, they are used to make large events, which would exceed a reasonable length of a single line, more readable. Secondly, they are used for events that are gradually updated, e.g., if a progress should be represented. When dealing with the interpretation of multi-line log entries, it is challenging to identify all lines belonging to one entry, because the writing of one record can be interrupted by the writing of another event. This interference can lead to the intermixing of lines from two separate events. To handle this problem, some event representations introduce a *relational identifier* that signifies the event to which a line belongs.

6.2.1.2 Event Separation

There are various concepts for the separation of event records in a list or stream of events. The primary goal of the separation is that a human reader or machine can easily derive the start and end of a given record.

Delimiter-based Separation A simple but also widespread method of separation is by a unique delimiter between two records. Single-line event records are using a simple newline character (`\n`) to delimit different entries. Examples of a line-based separation are the Syslog and Apache log formats as shown in Listing 6.1. For multi-line records, there are delimiters like doubled newlines (`\n\n`) or a sequence of dashes (`- - -`). Examples of these formats are audit logs of Apache's mod-security.

Listing 6.1: Delimiter-based separation in standard Syslog (shortened)

```
Mar 13 13:07:06 ws5 sshd: Failed password for john from 10.0.3.1 port 5631 ssh2\n
Mar 14 05:33:22 ws5 sshd: Accepted password for fred from 10.0.3.2 port 5416 ssh2\n
```

Pattern Separation Another form of separation for multi-line event records is by an implicit pattern of the record. This pattern must be so unique, that it is possible

to identify the beginning of a new event easily. A common case where such pattern-based separation is necessary are logs from the Apache Tomcat web server, as shown in Listing 6.2.

Listing 6.2: *Pattern-based separation for Java exceptions in Syslog (shortened)*

```
05-Mar-2017 10:30:04.124 SEVERE [main] o.a.coyote.AbstractProtocol.init Failed to
  initialize end point...
  java.lang.NullPointerException
    at o.a.tomcat.util.net.NioEndpoint.bind(NioEndpoint.java:360)
    at o.a..tomcat.util.net.AbstractEndpoint.init(AbstractEndpoint.java:730)
    at o.a.coyote.AbstractProtocol.init(AbstractProtocol.java:456)
    ... 14 more
05-Mar-2017 10:30:04.125 SEVERE [main] o.a.c.core.StandardService.initInternal
  Failed to initialize connector...
  o.a.catalina.LifecycleException: Failed to initialize component...
    at o.a.catalina.util.LifecycleBase.init(LifecycleBase.java:106)
    at o.a.catalina.core.StandardService.initInternal(StandardService.java:567)
    ... 20 more
```

The two event records can be separated by their unique date and time format (dd-*MMM*-yyyy hh:mm:ss.SSS¹) and the severity tag (SEVERE). The previously mentioned Syslog format is also suitable for pattern separation, as it has a prominent date and time (MMM dd hh:mm:ss) at the beginning and some distinct fields for the host and application afterwards.

Structured Separation A different method of separation is the use of well-defined structures, such as XML or JSON, for the records. As structures are self-contained, they do not need a special delimiter or pattern to define the boundaries of a record. Listing 6.3 shows an event record in the XML-format, which has an element <Event> that highlights the start and end.

Listing 6.3: *XML-based event records in Windows (shortened)*

```
<Event>
  <System>
    <Provider Name="Microsoft-Windows-Security-Auditing" />
    <EventID>4648</EventID>
    <TimeCreated SystemTime="2017-04-02T17:23:37.000Z" />
    <EventRecordID>4326</EventRecordID>
    <Channel>Security</Channel>
  </System>
  <EventData>...</EventData>
</Event>
```

6.2.1.3 Event Structure

There are three variations on how events are represented, i.e., structured, semi-structured and unstructured.

¹Format according to Java's SimpleDateFormat

Structured A fully structured representation keeps each piece of information separate and is designed for machine-readability. When interpreting events in this format, no complicated parsing is necessary as all information is distinguishable right away. An example of a fully structured log format is the Common/Combined Log Format (CLF) [90], as shown in Listing 6.4, which is used in web servers and proxies.

Listing 6.4: A CLF-record of an Apache Web Server

```
10.0.3.1 - - [02/Apr/2017:14:08:39 +0200] "GET /index.htm HTTP/1.0" 200 6762 "https://google.de" "Mozilla/5.0 (Windows NT 6.1; rv:52.0) Gecko/20100101 Firefox/52.0"
```

Unstructured A fully unstructured representation puts all event information in a textual form or in a form that is hard to interpret by a machine but easier to consume by a human. Furthermore, an unstructured format is more expressive than a strict structured format. Listing 6.5 shows an unstructured log record that describes a failed login.

Listing 6.5: Unstructured event record from an OpenSSH server

```
Failed password for john from 10.0.3.1 port 5631 ssh2
```

Semi-structured A semi-structured representation combines machine-readability of the structured and the flexibility of the unstructured representation. Typically, the structured part of the record holds important meta-information while the unstructured part describes the observed activity in a detailed textual form. A popular log format that uses the semi-structured representation is Syslog [87]. Listing 6.6 shows an example of a Syslog record that integrates unstructured application logs.

Listing 6.6: Semi-structured event record in the Syslog format

```
Mar 13 13:07:06 ws5 sshd[431]: Failed password for john from 10.0.3.1 port 5631 ssh2
```

6.2.2 Event Formats

The different ways of representing an event are precisely defined in event formats. These formats describe how a group of events is transformed into the digital form and should be comprehensive enough to express each property of an event. At the moment, there is a large variety of different event formats, and many applications or devices are even using their custom format to serialize their events. This fact becomes a challenge when serialized events need to be normalized. Fortunately, there have been efforts to standardize the representation of events, at least slightly, by providing event formats for common use cases.

6.2.2.1 Known Standardized Event Formats

Syslog Syslog is a standard around message logging that includes a structured log format as well as a transport protocol for the transmission of log messages between systems. There are two versions of the protocol, the older informational BSD Syslog from RFC 3164 [87] and the newer standardized Syslog from RFC 5424 [91]. Since BSD Syslog is still mostly used today, we are focusing on BSD Syslog in the following. The goal of the format is to provide an easily parseable message wrapper that standardizes the representation of necessary meta-information. As can be seen in Listing 6.6 and 6.7, a Syslog message consists of a header and an application message. The header contains meta-information like the event priority, creation time and information about the host and application where the event occurred. The message part is free to use for the event-producing application.

Listing 6.7: Structure of a Syslog message

```
PRI TIMESTAMP HOSTNAME APP-NAME[PROC-ID]: MSG
```

Common Event Expression (CEE) This format is an open specification by the MITRE Corporation that provides a unifying event format with a corresponding transport protocol and was developed together with industry and academia [92] as a result of missing unified event formats. The format has the goal to provide a unified event structure with high flexibility. The main idea is an object-oriented representation of the event information. The high flexibility is achieved by using *CEE Profiles*. A profile consists of a field dictionary, i.e., the set of supported fields, and an event taxonomy, i.e., a vocabulary for common event tags. The *Core Profile* is the foundation of CEE and contains the most basic fields and tags. Further profiles can be provided by product vendors, communities of interest and others and are laid on top of the *Core Profile*. Listing 6.8 shows an example of a CEE event.

Listing 6.8: Example of a CEE message for the event in Listing 6.6

```
{ "time": "2017-03-13T13:07:06Z",
  "host": "ws5",
  "app": { "name": "sshd" },
  "user": { "name": "john" },
  "src": { "ipv4": "10.0.3.1", "port": 5631 },
  "action": "login", "status": "success" }
```

As of November 2014, the funding for the development of CEE was stopped and the project is suspended by MITRE.

Common Event Format (CEF) This format [93] is an effort by the SIEM vendor of ArcSight to standardize and unify existing logs into one common log format. It aims to simplify the integration of various logs into one large log management and

analysis platform, the ArcSight ESM. CEF is fully-structured and consists of key-value pairs of relevant event information and is also flexible to be extended with new information. By default, CEF is integrated into a Syslog message and therefore extends it to a structured logging alternative. The structure and an example of a CEF message is presented in Listing 6.9.

<pre>Jan 18 11:07:53 host CEF:Version Device Vendor Device Product Device Version Signature ID Name Severity [Extension]</pre>	<pre>Sep 19 08:26:10 host CEF:0 Security threatmanager 1.0 100 worm successfully stopped 10 src=10.0.0.1 dst=2.1.2.2 spt=1232</pre>
--	--

Listing 6.9: The CEF message format and example [93]

Other Formats In addition to the previously mentioned formats, there are further efforts to standardize log formats. In particular, SIEM vendors are often using their own formats to unify many existing formats into a common format. We have already mentioned the *CEF* by ArcSight, but there are others like the *Log Event Extended Format (LEEF)* from IBM QRadar [84], the *Graylog Extended Log Format (GELF)* from Graylog [89], and the *Common Information Model (CIM)* from Splunk [81]. Another group of formats are domain-specific formats, such as the various formats for IDSs, like the *Intrusion Detection Message Exchange Format (IDMEF)*, *Incident Object Description Exchange Format (IODEF)*, *CybOX*, and the *Common Intrusion Specification Language (CISL)*, and web servers, like *Common Log Format (CLF)* and the *Extended Log Format*.

6.2.2.2 OLF: Our Object-based Event Format

The practice of event management reveals that the above formats have some shortcomings that make their use difficult. In fact, some of these formats never made it across the standardization phase, such as CISL or CEE, since they are not used in real-world systems. Another problem is that the formats represent events in a very different level of detail. While the Syslog format only provides basic meta-information about an event in a structured form, the CybOX format is over-specified and gets lost in details.

Because of these shortcomings and the difficulty to change the organization of existing formats, we have created the Object Log Format (OLF) in our research team [26]. OLF unites the flexibility and object-orientation of the CEE and the variety of fields from CEF and other application specific logs. Furthermore, we have put a focus on the security aspects of an event by integrating information about vulnerabilities as *Common Vulnerabilities and Exposures (CVE)*, weaknesses (*Common Weakness Enumeration (CWE)*) and the mapping to an attack kill-chain phase (*Adversarial Tactics, Techniques & Common Knowledge (ATT&CK)*), which are used for

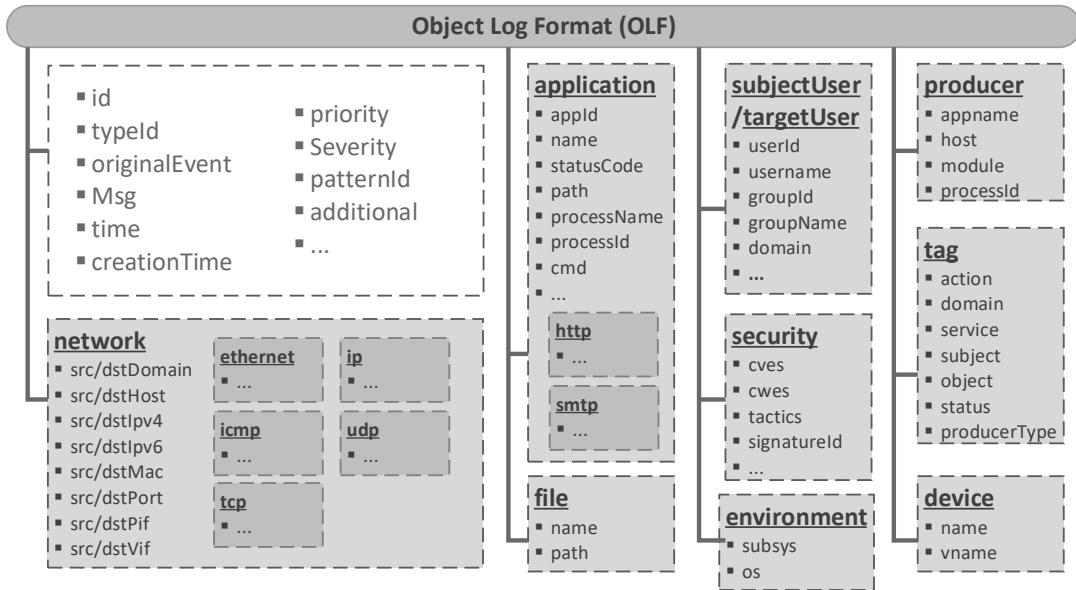


Figure 6.3: Event Structure of the Object Log Format (OLF)

APTs [9, 94]. An overview of the event structure of OLF is presented in Figure 6.3 and described in more detail in the work by Sapegin et al. [26].

6.2.3 Existing Event Normalization Methods

The transformation of raw events into a common format is achieved via a dedicated normalization method. This method can be considered as the core of a SIEM because it decides how far the collected events will be usable for later analysis. The research community, as well as productive log management/SIEM systems, have brought up a range of such normalization methods that have been utilized with varying degrees of success. Altogether, these existing methods can be categorized into four major groups.

Rule-Matching This method is based on predefined *normalization rules* that describe how relevant information can be extracted from events of a specific format or type. The most popular implementation for rule-matching is based on a normalization with regular expressions (regexes), where each regex matches one specific event format and uses matching groups to reference relevant information from it. In a special form of regex, the so-called named-group regular expression (NGRE) [95], the matching groups are further accompanied with a name that is used to map a group to a field in the unified event format. A challenge of the rule-matching approach is the overhead of finding the right rule

that matches the format of a given event record.

Tokenization/Indexing The idea of this method is to identify and extract distinct fields within a raw event record. For this purpose, an event record is split into its structural components, also known as tokens. Each token is then checked against known static parts of common event formats and against frequent data types in log events, such as IP addresses, hostnames or notations of time. A common implementation for tokenization is Apache Lucene². A limitation of the tokenization approach is the loss of the context in which the event information was embedded.

Natural Language Processing (NLP) This normalization method is based on the fact that many event formats are employing human-readable text to represent their content. As a consequence, this text can be decomposed by its language structure, such as subject, object, and verb. The components then help to deduce the meaning of the event similarly to how a human reader would see it. Examples of NLP implementations are Stanford’s CoreNLP library or SAP HANA’s text analysis features [96]. An example of the NLP technique was described by Kobayashi et al. in their work [97]. The problem with the NLP approach is its dependency on human-readable event formats and the difficult interpretation of the language components.

Custom Methods The custom implementation for the normalization can be most efficient because it can be tailored to the properties of a log format. For example, a custom solution could have two dedicated pieces of code to normalize CSV data and Syslog events. Due to the effort that is necessary to support a particular format, custom normalization is not applicable to very heterogeneous environments with many log formats. Examples of an implementation of the custom method are LogStash [88] and Sawmill [98].

In productive SIEM systems, the *rule-matching* approach has probably gained the most attention. It can be found in products like ArcSight [82], Splunk [81], AlienVault USM [99], and Graylog [89]. Regular expressions can reliably extract pieces of information from different formats, and the rules can be extended easily since regular expressions are a well-known concept in IT.

6.2.4 A Customized Rule-Matching Approach

An investigation on currently existing rule-matching implementations shows that they have big variations in their rule design, normalization capabilities and matching performance. There is no universal implementation that works best for all of them. One

²Apache Lucene - <https://lucene.apache.org>

thing that is rather consistent among the implementations is the utilization of regexes to identify an event format and extract relevant event information. As part of our Big Data SIEM, we have created a rule-matching variant that uses concepts from the existing implementations and is based on an advanced form of regex, namely the NGREs. It further realizes a form of deep normalization that extracts as much information as possible from an event record.

6.2.4.1 Concept of Information Categories

The regex-based rule-matching approach is based on the idea that event records of the same type have organized their information in the same way, even if the event content is in a textual form and unstructured. We distinguish three information categories for an event, i.e., *static*, *dynamic* and *semantic*, as shown in the example in Listing 6.10.

Listing 6.10: Log Event from Listing 6.6 with information categories (*static*, *dynamic*, *semantic*)

```
Mar 13 13:07:06_ws5_sshd[431]:_Failed password_for_john_from_10.0.3.1_port_5631_ssh2
```

The *static* portion acts as a structural frame of the event and is distinct for each type of event. As a result, the static part can be regarded as a suitable candidate to identify an event format. Some typical static parts of an event record are textual phrases, white spaces, and punctuations. In the above example, there are static words like `for`, `from`, and `port` that join the informational fields to a human-readable text snippet. Additionally, spaces, brackets, and colons are used to structure the provided information better.

The *dynamic* portion of a format is the content that changes with each instance of an event. It is usually the most valuable part of an event record because it contains the information that distinguishes an event from other events. Although each piece of dynamic content is changing from event instance to event instance, it has a fixed data type and is represented in the same format, such as a common time format or a number representation. In Listing 6.10, the IP address and port number are examples of dynamic event components.

The *semantic* portion of an event is an implicit piece of information that classifies the event into various categories, such as the performed `action` and the `outcome` of the observed event. Semantic information is usually difficult to extract from an event instance in an automated fashion, but it is sufficient to determine it manually once for each event type. When the event has been identified as being of a specific type, the event is tagged with the previously determined semantic information. For our example, the event could be tagged with `login` as the `action` and `failed` as the `outcome`.

Listing 6.11: Regular expression for the event in Listing 6.10

```
(\w+ \d+ \d+:\d+:\d+) (\S+) (\S+?)\[ (\d+)\]: Failed password for (\S+) from (\S+)
port (\d+) (\S+)
```

6.2.4.2 Matching Different Information Categories

Assuming that a large number of events is following the information categories presented above, then the static and dynamic parts of an event, i.e., the explicit parts of the content, can be identified with the help of a regex. To create such a regex, the static parts of an event can be directly copied over, and the dynamic parts can be captured with the concept of *matching groups*. Whenever this regex matches an event, meaning that the static parts in the event are corresponding to the static parts of the regex, the information in these groups can be retrieved with an associated identifier. For the above example, the regex from Listing 6.11 can be constructed.

The static parts of the event are directly represented in the regex with a bold font. These parts ensure that the typical characteristic of the login event can be recognized, and the format be determined. The dynamic parts of the event are represented with sub-regexes and are embraced with pairs of parentheses that constitute the matching group. In the regex from Listing 6.11, the first pair of parentheses is the matching group for the date and time of the event. The second pair is reserved for the host that produced the event.

An extension of the regular regex, as shown in Listing 6.11, is the *named-group regular expression (NGRE)*. It allows to label a matching group with a name that can be used to retrieve its value. Although this labeling is a rather simple change in the addressing of a matching group, it can bring a significant benefit for event normalization. The labeled name can act as a mapping key for the field name in the unified OLF event. Listing 6.12 shows how the previous regex would have to be changed.

Listing 6.12: A regex using named-groups to address event fields

```
(?<time>\w+ \d+ \d+:\d+:\d+) (?<producer.host>\S+) (?<producer.appname>\S+?)\[ (?<
producer.processId>\d+)\]: Failed password for (?<subjectUser.username>\S+) from
(?<network.srcIpv4>\S+) port (?<network.srcPort>\d+) (?<application.proto>\S+)
```

Now there are group names like `time` and `producer.host` that correspond to fields of OLF.

6.2.4.3 Integration of Regular Expression Templates

A drawback of regex-based rules is that the creation of more complex regexes is seen as a challenging task by developers. Especially the definition of the dynamic parts of the regex requires knowledge on the concept of character classes, matching groups,

and multiplicity of sub-regexes. Furthermore, the way in which a regex is written has a major impact on the matching performance. In particular, an inexperienced regex writer could produce constructs that lead to excessive backtracking with very low execution performance. To take the burden of defining regexes for rather complex, but still common, parts of events, a concept of regex templating is introduced. With this, an expert only creates a common regex once. After that, the template can be referenced by name in the full regex of the normalization rule. Ideally, the SIEM would even ship with a set of common templates so that a SIEM operator does not have to prepare templates by himself.

Taking the login example from above, there could be templates for time, IPs, ports, and hostnames that handle the complexity of the corresponding regex and include the right matching groups for the unified event. Listing 6.13 shows an example of a template for the typical Syslog time.

Listing 6.13: A regex template for the Syslog time

```
<regex-template name="G_SYSLOG_TIME">
  <regex><![CDATA[ (?<time>(?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec) \s
    +(?:3[01]|[1-2][0-9]|0?[1-9]))\s+\d{1,2}:\d{1,2}:\d{1,2}} ]></regex>
</regex-template>
```

This template, together with templates for the other dynamic fields, can then be referenced in the full regex by a special notation with two curly braces. Listing 6.14 shows an adjusted regex that utilizes templates for the more complex fields.

Listing 6.14: A regex using named-groups to address event fields

```
{{:G_SYSLOG_TIME}} (?<producer.host>\S+) (?<producer.appname>\S+?) \[ (?<producer.
  processId>\d+)\]: Failed password for (?<subjectUser.username>\S+) from {{:G_SRC
  }} port {{:G_SRC_PORT}} {{:G_PROTO}}
```

This method of writing a regex is considerably simpler than before, because a rule writer does not have to know how data fields as IP addresses and time values have to be represented. The rule writer is simply using a placeholder at the position of the data field to be matched.

6.2.4.4 Extending Normalization Rules with Static and Semantic Information

The NGRE is an integral part of our normalization rule because it allows to determine the event format and to extract the dynamic parts of an event instance. Still, there is more to an event than its dynamic information, such as semantic and other implicit information. To extract these parts, the event format to be matched needs to be interpreted and understood, for example by a human.

Our approach for a normalization rule is taking care of this additional type of information that is not directly extractable. Additionally to an NGRE, there are so-called *static-fields* that hold information that is added to the unified event once the

given NGRE matches. These static-fields are always the same for a particular rule and are assigned manually after a thorough interpretation of an event format. Listing 6.15 shows an XML-based representation of such an extended normalization rule.

Listing 6.15: A normalization rule that support implicit event information

```
<rule name="ssh-failed-login">
  <program-name>cpe:/a:openbsd:openssh</program-name>
  <pattern><![CDATA[(?<time>\w+ \d+ \d+:\d+:\d+) (?<producer.host>\S+)...]]></
    pattern>
  <static-fields>
    <static-field key="tag.action" value="login" />
    <static-field key="tag.status" value="failure" />
    <static-field key="network.dstPort" value="22" />
  </static-fields>
</rule>
```

The rule highlights that the previous login event is not just having a source IP and port, but actually represents an event covering a login that has failed. In addition, the rule can even go as far as identifying the software that produced that particular event, i.e., OpenSSH, and implying that this software listens on the well-known TCP port of 22 by default. All this information cannot be extracted automatically by matching against a regex.

6.2.4.5 Working with Rulesets

So far, only a single normalization rule has been considered for the matching of a given event, but there are many different event formats that have to be handled with a multitude of normalization rules in a real-world SIEM environment. Depending on the heterogeneity of the monitored network, a SIEM easily has to deal with dozens or even hundreds of rules at the same time. The collection of all these rules is called a *knowledge base (KB)*, because it holds the knowledge necessary to normalize an arbitrary event.

A big challenge in the normalization process is the management of the knowledge base and the selection of the normalization rule that matches the event. In a simplistic approach, the SIEM runs through all rules in the KB until it finds the first one that matches. Obviously, this solution costs performance, as many unnecessary NGREs are applied without a match. Nevertheless, there are SIEMs on the market that match their regexes exactly with this approach but fail to keep up with the velocity of incoming events.

As a solution to the rule selection, we propose a *rule indexing* for all rules in the KB. The indexing delivers, based on the current context, a prioritized list of candidate rules in which one rule most likely matches the given event. Only in the worst case, where no candidate rule matches, all rules have to be checked against the given event. In the case where no rule matches at all, the event remains unnormalized.

Our default indexing approach is derived from the fact that event logs often contain many similar events, whereas only a relatively small number of events are absolutely distinct. As a result, many events can already be matched with a small number of common rules. Only by ordering the rules by the number of times they have successfully matched an event, a list of prioritized rules has been created for the indexing. An even smarter approach of rule indexing is shown in Section 7.4.3 of the next chapter.

6.2.4.6 The Full Matching Process

Putting the previous steps together, a basic normalization process can be created, as shown in Figure 6.4.

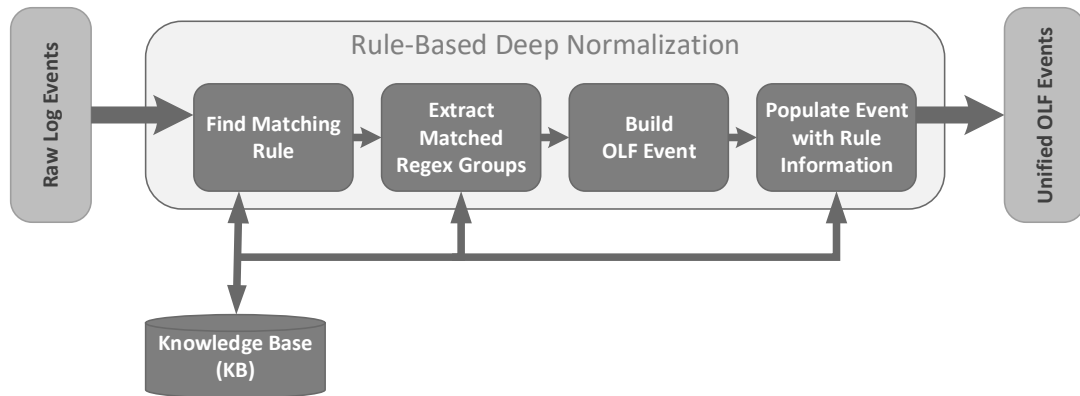


Figure 6.4: Process of a Rule-based Deep Normalization

In the first step of this process, a matching rule is searched in the KB by utilizing the rule index. The first matched rule is then passed onto the next step that extracts the named groups from the match. These extracted fields are then passed to the third step where the named groups are mapped to fields of the unified OLF event. In the last step, the unified event is further populated with the static fields that are specified in the passed rule. As a result of the process, an OLF is provided that contains the relevant event information. Taking our sample event from Listing 6.10 as a source, an OLF event like in Listing 6.16 is created.

Listing 6.16: The unified event in Object Log Format (OLF)

```

{ time: "2017-03-13T13:07:06",
  network: { srcIpv4: "10.0.3.1", srcPort: 5631, dstPort: 22 },
  producer: { host: "ws5", appname: "sshd", processId: 431 },
  user: { username: "john" },
  application: { proto: "ssh2" },
  tag: { action: ["login"], status: ["failure"] }
}
  
```

6.3 Enrichment

The normalization of raw events into a common event format is the first step of the pre-processing and is crucial for even the most basic types of security analytics. However, even if most of the relevant information is extracted and put into dedicated fields of the common format, there are some pieces of information missing to make a comprehensive correlation and analysis of events possible. Some examples are:

- Web server logs usually do not contain information about the producing host.
- Many events either have the hostname or IP address, but not both.
- Events in the Syslog format do not specify a year in their time information.
- Many event formats do not specify a time zone in their time representation.

Although the information is not available in the event itself, it would be derivable from other sources, such as events from the same network environment, inventory systems or publicly available knowledge bases. In existing research, the topic of enrichment is not deeply investigated. For example, Knight [100] proposes to resolve hostnames with DNS-lookups and obtain more information from WHOIS queries. However, there is no discussion about how further information could be obtained. Casey [101] addresses the problem of unsynchronized times and missing time-zones by implementing Network Time Protocol (NTP) clients and enforcing the UTC time-zone on all producing hosts. This solution does not seem to be practical in a large network environment, because some hosts might have requirements on the time-zone or have a closed operating system that does not allow such configuration.

As a solution to all of the mentioned scenarios of missing information, we propose three major methods of event enrichment, i.e., *intra-event*, *inter-event*, and *extra-event* enrichment.

6.3.1 Intra-Event Enrichment

This method enriches a normalized event with information that is derivable from its log entry but could be extracted by a regex-based normalization rule. Looking back at the normalization process, then it can be derived that a regex is limited to extracting only static information from a log entry because of its design as a finite automaton. Any information that is encoded with a more complex logic cannot be interpreted by it. As a solution to even extract such information, we propose lightweight processing scripts that are attached to each normalization rule.

6.3.1.1 Lightweight Processing Scripts

The handling of more complex data structures in a log entry requires processing logic that goes beyond that of a regex. One solution to provide this logic could be to modify the normalization code for each supported event format, but this would not scale for a larger number of formats. Whenever there is a new format, the code would have to be changed, and the SIEM needs to be redeployed. A more practical approach is to provide additional logic as external scripts that can be executed at runtime in an interpreter. In the case of our implementation, the Java scripting engine allows executing JavaScript and Python code within the program context. As a script is dependent on the type of event, such a script would be attached to a normalization rule. If an event is normalized with the rule that has an attached script, then this event is additionally passed through the attached script. The normalized event object is made available to the script as a context object named `evt`, whose properties can be read and modified in the script.

For a better understanding, we demonstrate the example of Cisco ASA firewalls. These firewalls are using a Syslog variant that specially encodes Syslog's application specifier, as shown in Listing 6.17. In this sample log entry, the application field (`%ASA-6-302014`) does not only indicate that the event originates from an ASA firewall, but it also reveals the log priority (6) and event type (302014).

Listing 6.17: *Cisco ASA log in a Syslog variant containing encoded data*

```
Jun 11 12:26:34 10.10.10.1 %ASA-6-302014: Built outbound TCP connection 8236059 for
outside:192.168.7.4/443 (192.168.7.4/443) to inside:10.10.10.5/49350
(192.168.5.4/49350)
```

A dedicated normalization rule for such Cisco events is not desirable, because it causes overhead in the rule selection process although the log is compatible with the known Syslog format. Rather, a JavaScript snippet like in Listing 6.18 can be used to extract the encoded fields. It reads the application field from the `evt` object and writes the priority and event type back to it.

Listing 6.18: *Extraction script that extracts relevant Cisco ASA information*

```
var match = /%ASA-(\d+)-(\d+)/.exec(evt.getProducer().getAppname());
evt.getProducer().setAppname("Cisco_ASA");
evt.setPriority(parseInt(match[1]));
evt.setEventTypeId(parseInt(match[2]));
```

To reduce the impact of the script on any normalization other than Cisco ASA logs, this script would only be executed for events previously normalized with an ASA normalization rule.

6.3.2 Inter-Event Enrichment

Sometimes the information that is given in an individual log entry is incomplete and not sufficient for further analytics. Still, such missing information can potentially be found in contextual log entries, i.e., events occurring at the same time or in the same environment. So, by correlating such contextual events with each other, some of the incomplete information can be completed. Since information is derived from other events, we call this kind of enrichment *inter-event enrichment*. We propose two methods of inter-event enrichment, which are either focused on time or general information.

6.3.2.1 Time Enrichment

The time at which an event occurred or was processed is a central field of information in an event and allows seeing an event in the context of other events that happened at the same time. Therefore, any time fields in an event should be as exact as possible. Unfortunately, event producers can have different system times and even different time-zones. Furthermore, the time information in an event can be incomplete, such as in a typical Syslog event where the year and time-zone are missing.

The correlation of multiple events can eliminate the shortcomings of time fields, such as missing or wrong information. In the case of missing time information, log events with a more detailed time format from the same producer host and hence with the same system time can be incorporated into the completion of time information. For example, an event that only exhibits a Syslog time could be completed with time information from an event with Apache's CLF. Figure 6.5 illustrates this example.

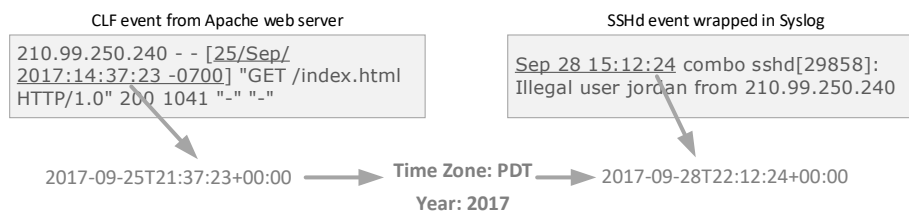


Figure 6.5: *Extracting missing time information from related logs*

Events that are received in real-time, meaning they were not imported as a batch or buffered anywhere, can have the retrieval time at the SIEM as their creation time, including its time-zone. It is possible to determine whether a group of events has been received in a batch by checking the deltas between the creation times and retrieval times of two individual events. If these deltas differ significantly, then the events might have been imported as a batch. Considering the retrieval time of an event, a small time drift to the creation time is caused by network delays when the log entry

is sent to the central log server. This time difference, however, can be corrected by determining the network delay time and subtracting it from the retrieval time.

$$\Delta t_{Delay} = t_{Retrieval}(e_1) - t_{Creation}(e_1) \quad (6.1)$$

$$t_{Creation}(e_2) = t_{Retrieval}(e_2) - \Delta t_{Delay} \quad (6.2)$$

The approach of analyzing the retrieval time for time differences is not applicable to batch imports, because that time solely represents the time of import. Nevertheless, there is another approach that is not that accurate but still works in most of the cases. Typically, during batch import, the natural order of events is often very close to the chronological order of the events. Table 6.1 gives an example of events as they might have been imported through a batch.

Table 6.1: Identification of time drifts in context logs

Event ID	Producer	Creation Time	Derived Time	Derived Difference
1	System 1	14:37:22	14:37:22	
2	System 2	12:15:29	14:37:43	$\Leftarrow 2:22:14$
3	System 3	14:38:04	14:38:04	
4	System 1	14:39:02	14:39:02	
5	System 2	12:16:49	14:39:03	$\Rightarrow 2:22:14$
6	System 1	14:39:04	14:39:04	

In this table, *Event ID* corresponds to the import order. Although all events are in chronological order, there are two events, i.e., 2 and 4, that show a creation time that is significantly off the other four events. These four events have creation times that confirm the chronological order. Looking at the producer systems, it seems that *System 1* and *System 3* have similar system times. Only *System 2* is producing events with lagging times. Based on the assumption that the majority of systems show the right time, the creation time of event 5 is estimated from its surrounding events, which would be 14:39:03. Continuing with the system time delta for *System 2*, the time for all its other events, namely event 2, can now also be corrected.

6.3.2.2 General Context Enrichment

In addition to time, many other fields are derivable from contextual logs. The idea is to take information from verbose events to complete information from less verbose events. Table 6.2 lists examples of fields that can be correlated and derived between events.

The rows in the table can be interpreted as follows. We assume there are two normalized events e_1 and e_2 , where e_2 should be completed with information. If e_1 has values for all required and derived fields and e_2 has the same values in its required

Table 6.2: Derivation of event fields from context events

Required Fields	Derived Fields
producer ID / stream ID	host
host, application, user ID / username	first name, last name, user ID, user domain, username
host, application	application ID (CPE), destination port, process ID
host, netw. interface (NIC)	ethernet address
host, IP / DNS name	IP, DNS name
host	timezone

fields as e_1 , then the values of e_1 's derived fields can be copied to e_2 . For example, if e_1 has values for the IP and DNS name of a source host, then e_2 with only a DNS name can be enriched with an IP address. Of course, this enrichment is only possible where a unique mapping from DNS to IP and no round-robin DNS entries are employed.

In practice, there is a variety of log formats that are mostly missing information and formats that mostly act as information providers. CLF, as the most popular format for web logs, is taking both roles, that of an information provider but also information consumer. On the one hand, as we showed for the time enrichment, CLF provides very detailed timestamps. On the other hand, CLF does not have details about the destination host. To complete this piece of information, a Syslog event from the same producer host can be utilized. Listing 6.19 shows two such events for a better understanding.

Listing 6.19: Syslog event that completes the destination host of a CLF event

```
# CLF event
192.168.14.2 - - [25/Sep/2015:14:37:23 -0700] "GET /index.html HTTP/1.0" 200 1041
"_" "_"
# More detailed Syslog-event
Sep 25 23:37:24 sec.hpi.de http-monitor: HTTP-access from 192.168.14.2
```

Both of the events are known to be produced at the same web server. The Syslog event indicates that this server has a hostname of `sec.hpi.de`, while this information is missing in the CLF event. By enriching the CLF event with the hostname, the source and destination of each web access become clearer.

6.3.3 Extra-Event Enrichment

The third type of enrichment is incorporating external intelligence to fill missing information in event records. This intelligence can originate from the public domain, also called Open Source Intelligence (OSINT) [102], or from internal systems.

6.3.3.1 Data from Open Source Intelligence (OSINT)

OSINT constitutes information that has been produced from publicly available information and helps in the completion of specific event fields that are often unavailable in a log entry. The following list is a selection of useful OSINT sources.

- **Domain Name System (DNS) servers** are resolving IP addresses to DNS names or vice versa. There are also special passive DNS databases that are keeping track of all past mappings between IPs and DNS names.
- **WHOIS servers** are mapping domain names to their geographical location and give information about the domain registor and registrar.
- **Vulnerability Databases** are providing an overview of all known vulnerabilities of software. They can map a vulnerability ID (CVE) to a detailed vulnerability description as well as a classification of its severity (Common Vulnerability Scoring System (CVSS)) and affected weaknesses (CWE).

6.3.3.2 Data from Internal Systems

Another form of enrichment information is available from systems within the own environment. The advantage is that this information is most specific and directly related to the emitted events. We see two types of internal systems that are available in many IT infrastructures.

- **Authentication Systems** are a valuable source for gaining information about a single user, such as his permissions or group affiliation. Windows Active Directory (AD) is one authentication system that is available in many enterprise networks.
- **Inventory Systems** are maintaining a detailed overview of the monitored network and its nodes. Furthermore, it is keeping track of installed software, hardware, and application users for known systems.

6.4 Persistence

The persistence step is responsible for writing normalized and enriched events into some data storage. This data storage acts as the connection point between real-time and batch-processing, meaning that it enables to apply analysis on all events and not on just a single event. Furthermore, all the data is kept for the record and can later be reprocessed in any kind of log analysis tool. When persisting events, it has to be considered how they are later accessed and analyzed, because this has an impact on the employed type of storage system and the data representation.

6.4.1 Storage Systems

At the SIEM market, solutions are using a wide range of mostly custom developed storage systems. The main differences are whether the data is queryable with SQL and whether a static schema is required. MicroFocus ArcSight and IBM QRadar are using custom SQL databases, whereas Splunk and LogStash count on a distributed NoSQL database. As a distinction to the mentioned systems, LogStash uses the schema-free storage system ElasticSearch, which works similar to a search engine. Our experiments with three of these systems showed that the employed storage systems have their disadvantages when it comes to very large datasets. In particular, there are limitations in either writing or querying the data, because it is mostly kept on slower hard drives storage and records are stored sequentially instead of in a columnar structure, which can be read and written more efficiently. In addition to these custom solutions of the SIEM vendors, there are also other Big Data storage solutions, such as Cassandra, MongoDB, and Hive. Although they are distributed over multiple nodes, they also do not perform well in writing as the review by Rabl et al. [85] suggests.

6.4.1.1 SAP HANA as Primary Storage

As an alternative to the above approaches, we are proposing the column-based in-memory RDBMS SAP HANA as the primary storage in REAMS. All incoming events and a large amount of recent historical events are retained in this storage, making it also to a kind of *hot storage*. The main difference of SAP HANA to other storage systems is that all tables are kept in the main memory. Therefore, all table writes and queries are executed on top of the main memory, making the access multiple magnitudes faster.

As an RDBMS, HANA follows the concept of tables that are structured according to a given schema. We think that a schema has an advantage in a SIEM, especially if a common log format like OLF is used. A set of common fields that can be accessed by columns allows an efficient correlation and querying. As a column-based RDBMS, HANA organizes its tables by columns instead of rows and all values of a column are stored sequentially. Using this organization has benefits in querying tables with many columns because then only the columns relevant to the query have to be accessed. Another advantage that comes with a column-based structure is that a new column can be easily added to a table, without touching all existing records. Such flexibility in the schema is desirable if OLF is extended later.

In the context of Big Data processing, RDBMSs are often seen as unsuitable, because they rely on the ACID (Atomicity, Consistency, Isolation, Durability) principles. Generally, ACID has a negative performance impact due to locking that is related to immediate consistency. However, especially for the handling of log events,

an event in the database does not have to be changed anymore, and locking is not required. So, the typical disadvantages of ACID do not take effect. Rather, if the immutability of events is ensured, HANA can act similarly to the well-known Big Data storage systems.

6.4.1.2 Open-Source Storage Systems as Secondary Storage

In addition to the hot in-memory storage, we propose three optional storage systems, i.e., *Kafka*, *ElasticSearch*, and *HDFS*, that increase compatibility with other security analytics systems.

Kafka acts as a temporary message queue that is well supported as a data source in many existing Big Data solutions. Since all data in Kafka is written as a sequential file to the file system, relatively high throughputs can be achieved with cheap hardware. The bottleneck is more the preparation of messages for the serialization. A consumer for the events in Kafka would read the events sequentially and can therefore also relatively quickly read all available data. However, especially for ad-hoc analysis, it would be necessary that each event is read and deserialized, which is a rather time-consuming task.

HDFS is another way to improve accessibility for external tools, such as Apache Spark. The distributed file system is considered as a cold storage since the backing is realized by cheap hard drives. Especially considering larger network environments that produce petabytes of events per year, such storage is inevitable as a backup for the in-memory database. In comparison to other storage systems, a user is free to choose any format for storing his data as long as it can be written to a file. Nevertheless, the two file formats CSV and Parquet are most commonly used and are well supported as import format in Big Data applications. The disadvantage of HDFS is similar to Kafka, i.e., any kind of analysis requires the reading and deserialization of each event.

As an experiment for the adaptability of our persistence implementation, we have also integrated **ElasticSearch** as a hot storage location into REAMS. ElasticSearch has a schema-less data model and is meant for text indexing of the model fields. Due to this indexing, all fields of an event are searchable with a variety of different search options, including possibly unnormalized parts. Another benefit of ElasticSearch is the availability of powerful user interfaces, such as Kibana³ and Grafana⁴, which come with all sorts of dashboards and graphs applied to the data stored in ElasticSearch.

³Kibana - <https://www.elastic.co/de/products/kibana>

⁴Grafana - <https://grafana.com/>

6.4.2 Mapping of the Event Model to a Schema

The persistence of events to a storage system requires that their event model is mapped to the corresponding storage data model. SAP HANA, as our primary storage system, is a RDBMS that is based on columnar tables, so the event model has to be represented in multiple tables. The main question is whether an event should be put into a single large table or split into multiple related tables.

Relational Model In the sense of a relational database, the data model should be in accordance with the normal forms [103]. A data model in normal form reduces data redundancies and improves data integrity, which at the same time reduces the required storage space and improves data query times for specific use cases. In the concrete case of our event model, the normal form separates the event model into multiple tables representing the root object as well as the child objects of OLF, such as application, user, and network. The records in the table for the root object refer to the primary keys of the records of its child objects by so-called foreign keys. If a child record with the same values already exists in a table, then, in accordance to the normal form, the root record refers to the existing child record and no new record would be written. From our experience, a typical event has around 4-5 child objects in OLF initialized, hence requiring the writing of records to 5-6 tables. Also, it might be necessary to check for previously existing child records to set the foreign keys of the root record. Adding this together, there could be around 10 SQL queries to the database to persist one event. This number of queries is so high that a reasonable persistence throughput is difficult to achieve.

Master Table Model Another data model approach, which is preferable from the view of persistence performance, is to reduce the number of tables. We propose to use a single master table that contains all relevant fields of the event model, including the fields of OLF's child objects. The object hierarchy is flattened to fit into a table structure. Only additional fields, which are not part of the default model, are stored in a separate property table. The relation of the tables is shown in Figure 6.6. In practice, many events would only require a single write instead of 10. For the rare case where additional fields are present, a few more writes would be necessary. In terms of analysis, the usage of a single table in HANA does not have such an impact as in traditional RDBMSs, since the columnar tables and HANA's field compression still allow a fast table scan.

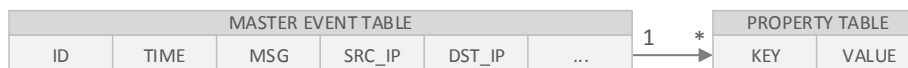


Figure 6.6: Idea of the master table model

Type Mapping When mapping the event model into the master table, the field types are adopted to the storage system. In particular, fields with strings, numbers, and dates are represented as corresponding field types in the database. This is important for a more in-depth analysis that involves grouping and aggregation of values in the model.

6.5 Conclusion

In this chapter an event processing workflow was presented that goes from the raw event to its representation in the database. At the beginning, there is a large variety of events that are scattered over a wide range of heterogeneous event sources. We have categorized these common event sources and have outlined methods of gathering events from these sources into a single SIEM system. In the second step, we normalized log entries to a common event format, so that they can be used in a structured manner for further analysis. Mostly, such normalization can be challenging, because there are differences in how entries are represented, such as in their level of structure, their separation from each other and their encoding of information. Our investigation on known normalization techniques has shown that there are currently limitations in extracting all available information from a log entry. However, for comprehensive security analysis, each event should provide as much information as possible. We propose a custom normalization technique that addresses this limitation by utilizing extensible normalization rules that are based on the known technique of regexes matching. As a result of the normalization, the events are transformed into our common Object Log Format (OLF). Since the information contained within a log entry is often incomplete or not normalizable with a regex, we propose multiple methods of data enrichment from contextual information directly after normalization. We have identified three sources of enrichment information, i.e., information within the same event, from other events of the monitored environment and external sources. Each category of information can provide different pieces for a more complete event record. In the last step, we investigated the persistence of events in OLF. We have proposed SAP HANA as the primary storage of events because its design fits well to our use case and has various advantages over existing storage systems. Nevertheless, we also consider three open-source storage systems, i.e., Kafka, HDFS, and Elastic-Search, from the Big Data domain for the purpose of a cold storage as well as for compatibility to third-party systems.

The writing of events into a storage system has paved the way for fast access to event data and the correlation of multiple events. It is ensured that events in their variety of formats are transformed into OLF. However, when dealing with Big Data, not only the volume and variety is a challenge, but also the velocity in which events are arriving. In the following two chapters, we further investigate how the workflow from this chapter can be improved to handle the throughput of large enterprise networks.

Chapter 7

Improving the Speed of Event Normalization

Related Publications

- *David Jaeger, Amir Azodi, Feng Cheng, and Christoph Meinel. “Normalizing Security Events with a Hierarchical Knowledge Base”. In: Intl. Conference on Information Security Theory and Practice. 2015 [28]*
 - *David Jaeger, Andrey Sapegin, Martin Ussath, Feng Cheng, and Christoph Meinel. “Parallel and Distributed Normalization of Security Events for Instant Attack Analysis”. In: Intl. Performance Computing and Communications Conference. 2015 [24]*
 - *David Jaeger, Feng Cheng, and Christoph Meinel. “Accelerating Event-Based Attack Detection with a Distributed In-Memory Platform”. In: Intl. Conference on Dependable, Autonomic and Secure Computing. 2018 [25]*
-

Deep normalization of events to an OLF object is one of the most time-consuming parts in the event processing workflow. Therefore, to speed up the overall processing workflow, also the normalization process has to keep up with the velocity of incoming events. Many currently existing SIEMs are not fast enough to handle the event load of large enterprises, because their normalization routines are not optimized for speed. In this chapter, we have a look at several optimization strategies for the normalization algorithm.

7.1 Normalization Performance Factors

Considering that events are normalized with the proposed process, then each of the processing steps is a potential point of optimization. Furthermore, since the process is executed sequentially, each step can become a bottleneck of its own that slows down the overall workflow. Therefore, it is necessary to carefully analyze all processing steps for potentially processing-intensive tasks. By reviewing the different processing tasks during basic normalization, we can identify three potentially processing-intensive candidates that may impact the throughput.

- The *application of regexes* is employed to find a matching rule in the KB and to extract information from the match. As each operation involves the application of a regex for each normalization rule, there is much potential for overall performance improvement.
- The *assignment of information* to the unified event is used during the extraction of event information and the population of the event with static information from the matched rule. The assignment involves the identification of the right field in the unified event to which the information should be assigned and the copying of the data into the unified event object.
- The efficiency of the *indexing and searching* within the KB decides how many rules have to be checked and how quick a matching rule is found. Depending on the size of the KB, a proper search algorithm can reduce the number of rules to be checked from multiple thousands to a small, countable number.

In the basic normalization, all these tasks have not been especially considered for their performance. The following subsections take a closer look on approaches for the optimization of these steps together with their impact on the event normalization throughput. All the approaches have been integrated into REAMS and are thus focused on the implementation with Java.

7.2 Application of Regular Expressions

Regular expressions are a well-known concept of pattern-matching in computer science and have been covered in a large number of research works. Various algorithms and implementations for the application of a regex against a given text have been developed. In the end, most of these solutions are evaluating regular expression patterns in finite automaton, more specifically in NFA or DFA. Each type of automaton has its advantages and disadvantages. In particular, there are differences in their runtime, memory consumption, and features. To come up with an efficient event matching

algorithm that uses regexes, both types of finite automaton and their corresponding implementations have to be understood and checked for their applicability to the special domain of event matching.

7.2.1 Finite Automaton

There are two types of finite automaton that recognize regexes, the DFA and NFA. Both take a string as input and decide whether it is accepted by the designed expression.

The *deterministic finite automaton (DFA)* is constructed in a way that the transition from one state to the following state is always *deterministic*, meaning that at a given state there is only a single transition that triggers for the current position of the input. Building an automaton with these properties is time-consuming and memory-intensive, because all possible intermediate states of a regex have to be determined and represented in the automaton. Especially for long and complex patterns, the computational complexity required to convert the pattern to a DFA and the representation of all its states explodes. On the other side, the runtime of the DFA is linear to the input length ($\mathcal{O}(n)$, n is number of characters in input), because there is only a single transition per input character. These two characteristics make the DFA particularly fast in matching but difficult to handle for more complex regexes.

The *non-deterministic finite automaton (NFA)* is a superset of the DFA and allows nondeterministic transitions from one state to the following state, meaning that for a given state there can be multiple transitions that trigger for the current position of the input. Building an automaton with these properties is significantly less time-consuming and memory-intensive than the building of a DFA because the states and transitions can be directly derived from the regex. However, due to its nondeterminism, the runtime of the NFA is much longer ($\mathcal{O}(m^2n)$, m is number of states in the regex) than the one of a DFA, because each possible path in the graph has to be checked.

7.2.2 Implementation Issues of Finite Automaton

The two types of finite automaton are the theoretical concepts of regex matching, but there are more issues to consider in the implementation of these concepts.

One challenge in the implementation of an NFA is the realization of its nondeterminism because there must be an algorithm that is able to try all transition alternatives for a state at once. *Backtracking* is one such algorithm, which employs a try-and-error method that iterates over all transition alternatives to find a matching path in the NFA for a given input. Due to its iterations, backtracking should be able to find the matching path if it exists, although it is not clear how long the algorithm will take to find it. In fact, there are some pathological cases of regex constructions, such as “(a*)*b”,

that cause the backtracking-based NFA to get stuck in trying an almost infinite number of alternatives if longer inputs are provided. As a consequence, regexes for NFAs should be created carefully to prevent these pathological cases.

An implementation property that is also closely related to the way the backtracking works is greedy and non-greedy/reluctant matching for the quantifiers “*”, “+”, and “{min, max}”. In the greedy matching variant, the matching algorithm tries to consume as much input as possible for a single quantifier, whereas the algorithm tries to consume as few as possible for the non-greedy variant. Greedy matching is the default mode for a quantifier and means that the backtracking first consumes all possible characters until it tries the next alternative. Due to many possibly unnecessary consumptions for the greedy variant, non-greedy quantifiers should be used in favor of greedy quantifiers, which are denoted with an additional “?”. In contrast to the NFA, a DFA only supports the default greedy matching, because all transitions are deterministic with no choice of being reluctant in the state traversal.

Another factor in the implementation of automata is the extension of pattern matching with additional features. While early implementations, such as Perl Compatible Regular Expressions (PCRE) [104] before version 2.0, were merely meant for checking a pattern against a string, later implementations added a variety of features that allowed to reference parts of the match, i.e., back references and named capturing groups (NGRE), and specify conditions for a match, i.e., lookarounds. Unfortunately, many of these features are only supported for NFA implementations, because the design of the NFA is very close to the regex that specifies these features, whereas the DFA is close to the input string and implicitly checks various parts of the regex in parallel.

The described issues make clear that implementations with the same type of automaton share similar properties. An NFA-implementation usually has a broad range of special features, such as capturing groups or lookarounds, and a small memory footprint, but a relatively slow matching speed. A DFA-implementation, in contrast, has a high matching speed, but a limited feature set and large memory footprint. Furthermore, the regexes that are supported by a DFA are significantly different to what is generally understood to be a common regex. Thus, the writing of regexes that are compatible with a DFA is challenging.

7.2.3 Evaluating Regex Implementations

To improve the rule-matching of our SIEM, a regex implementation with high matching performance and powerful regex features has to be found. For this purpose, seven known regex implementations, which are written in the Java programming language, have been examined for their employed type of automaton and evaluated for their matching performance. As part of the performance evaluation, each implementation normalizes a set of 5 M events from a productive web system. All the events are rep-

CHAPTER 7. IMPROVING THE SPEED OF EVENT NORMALIZATION

resented in CLF and are normalizable with a single predefined regex that only uses basic features. The results of the evaluation can be seen in Table 7.1.

Implementation	Class	kevt/s	Rate	Automaton
Java 8 Regex	java.util.regex.Pattern	100.7	100%	NFA
Named-Regexp	com.google.code.regex.Pattern	99.1	100%	NFA
JINT	kmy.regex.util.Regex	89.7	94.5%	NFA
JRegex	jregex.Pattern	72.9	100%	NFA
PAT	com.stevesoft.pat.Regex	44.7	100%	NFA
RE2J	com.google.re2j.Pattern	24.3	100%	DFA/NFA
BRICS	dk.brics.automaton.RegExp	0.0	0%	DFA

Table 7.1: Performance of regular expression implementations

The table shows that the majority of the implementations are employing an NFA for regex matching. This distribution is explainable with the much higher expressiveness of allowed regexes. In turn, the limitation in the regex expressiveness of the DFA-implementation, i.e., the *BRICS* library, has the consequence that the provided CLF regex is not supported. Since a matching regex for *BRICS* would require significant changes to the provided regex, it is not feasible to compare it against the others. However, the matching speed for the NFA-implementations could indeed be measured and is specified in the table. Five of the NFA-implementations were able to normalize 100% of the 5 M events, while one implementation had problems with events that contained IPv6 addresses. Surprisingly, the fastest of all the implementations is the *Java 8 Regex* (`java.util.regex`), which is already integrated into the Java standard library. All the other specialized libraries for regex matching are slower than the default regex library of the Java standard library. Also, the *Named-Regexp* on the second position is closely related to the Java 8 library. It uses the matching capabilities of Java 8 but provides some additional features for NGRE matching. One of the slowest libraries is *RE2J*, which is inspired by the Google RE2 library. In comparison to the other libraries, the RE2 library is a hybrid library that is more focused on the DFA-implementation, but also comes with an NFA-implementation for all regexes that utilize special features.

In the end, the standard Java 8 regex library and its extended Named-Regexp library seem to be the most promising implementations for regex matching, because they are the best performing libraries and still have rich features. The matching throughput of both libraries is as high as 100 kevt/s with only a single thread, which is four times as fast as the least performing regex library, i.e., *RE2J*. A DFA-implementation, such as *BRICS*, would be too restricted in its supported regexes and would complicate the creation of normalization rules for complex event formats. For the REAMS implementation, the concrete matching will be performed with the

Named-Regex library, because it has one of the best matching throughputs and all common regex features. In particular, the library has extended support for NGREs, which are especially important for the mapping of event content to a unified event.

7.3 Unified Event Assignments

A normalized event is created by assigning previously extracted pieces of information, either from a matched NGRE or from the static fields of the normalization rule, to fields in the unified event format. This assignment consists of three steps, i.e., the identification of the field in the unified format to which the information belongs, the conversion of the information to the data type of the identified field, and the setting of the value to the unified event object.

The variety of event data demands that a unified event format remains adaptable to new types of information. Therefore, a SIEM should be extensible in its event model and support new information fields as they appear. However, this dynamic in the event model also becomes a challenge for assignments.

7.3.1 Property Access Approaches

The support of dynamic event models, such as in *REAMS* with the OLF, can be implemented with two different assignment mechanisms, which are inspired by the two ways of accessing properties in a programming object, i.e., static or dynamic.

Static access to an object is the simplest form of access, because the properties, i.e., variables or getters/setters, to be used are statically specified in the program code. In contrast to that, dynamic access to an object is much more flexible, because the properties can be selected at runtime by their name or other properties. Although the dynamic approach enables flexible programming, it is still considerably slower than the static access to an object [105]. We have implemented and evaluated both, the static and dynamic, assignment approaches.

Dynamic Property Access In the dynamic approach, the unified event object is represented as a JavaBean where each property is addressable over an object notation string. Therefore, each named matching group or static field has to have a name that is compatible with the object notation and points to a field in the unified event format. For example, if the name `net.srcIpv4` is used in a named group of a rule's NGRE, then the extracted value will be assigned with the setter `setSrcIpv4` in the `net` object of the unified event object. The actual lookup of the setter by name and the invocation of that setter is performed over Java's Reflection API.

Since each group name in an NGRE or name of a static field is used as a path to the right field in the event object, there is no further need to specify mappings

between groups and field names to unified fields. Furthermore, specialized libraries like *Commons BeanUtils* make it even easier to assign values over the JavaBeans concept. The library comes with features like caching of property lookups and automated conversion of data types if they do not match between value and final object property.

Static Property Access In the static approach, each object property is directly accessed in code, without the help of the performance-degrading reflection. The challenge with this approach is the dynamic of fields, because based on the current information field, the corresponding property has to be called directly in the code. At first glance, this only seems possible with a rather big `switch`-statement or a `LambdaMetafactory` that decides, based on the name of the field, to which property the field value should be assigned. However, this still requires a rather fixed code section that is not flexible to changes in the model.

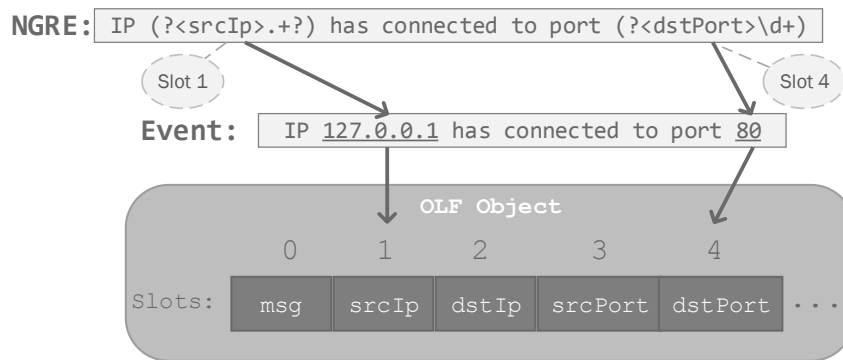


Figure 7.1: Slot concept as used for assigning event fields to an OLF event

As an alternative solution to the direct calling of functions, we are proposing to keep the values of the event object in an array of data slots. Each data slot is dedicated to an event field and is accessible either by a field's setter function or by a generic setter that allows setting a field by its name or by its index. Figure 7.1 shows how the assignment of extracted NGRE fields is achieved with the slot concept. As a preparation, each NGRE is analyzed for extractable fields and the dedicated slot indices for these fields are obtained. When the indices for the fields are resolved, the NGRE is used for the matching and the matched fields are extracted. For each of the extracted fields, the values are assigned to the data slot with the previously determined index. When this procedure has been finished for all the matching groups of the NGRE, the normalization rule is fully applied, and all fields in the event object are set.

7.3.2 Object Pooling and State Resets

Another factor that plays into the assignment is the way how unified event objects are recreated for each new raw event instance. In the default case, a new event object would be created with each incoming event. With this, each object would be freshly initialized, and all fields would have the default state. However, each time a new object is requested from the Java Virtual Machine (JVM), around 1 k B of memory for more than 100 fields has to be requested from the operating system. Assuming an event throughput of multiple $M \text{ evts./s}$, then the requesting of so many relatively large memory chunks can become a bottleneck. Furthermore, each of the requested objects needs to be garbage collected.

The performance impact resulting from the object creation is addressable with the *object pooling* concept. The idea is to create a set, or pool, of objects once at the beginning of the program and then reuse these objects throughout the entire runtime. Although object pooling is sometimes considered outdated, since Java's object creation overhead has been minimized over the years, the allocation of large amounts of memory is still a performance factor that pooling can influence.

One disadvantage of pooling is the effort needed to handle the pooled resources and the need to reset the state every time an object is reused. In the approach of dynamic property access, the reset is accomplished by iterating over all properties and setting these properties to their default value. For the static approach, however, we propose a more efficient approach that leverages the slot concept. In addition to the object array, we are introducing a bitset for which each bit signifies whether a specific field slot in the object has been set or changed. Since for a typical event only a small number of fields is set, the resulting bitset only has a rather small number of bits set, usually around 10-20. This property and the exact information about set fields is leveraged at the static reset, because there only the fields are reset to their default state that actually have been set or changed before. So, instead of resetting more than 100 fields, only around 10-20 fields are reset. Details about the bitset idea are described in Section 8.2.1.2.

7.3.3 Evaluation

To prove the performance benefits of the static approach with object pooling and the bitset-based reset, an evaluation for all the different variations of approaches has been conducted on 5 million Apache web server events. In particular, each type of access has been tested with the default creation of event objects for each incoming event, the reuse of an existing object without reset and the reuse with reset. The results of the evaluation are shown in Figure 7.2.

The bars in the diagram show the average event throughput per second. The static property access is more than 35% faster for all combinations, because no reflection is

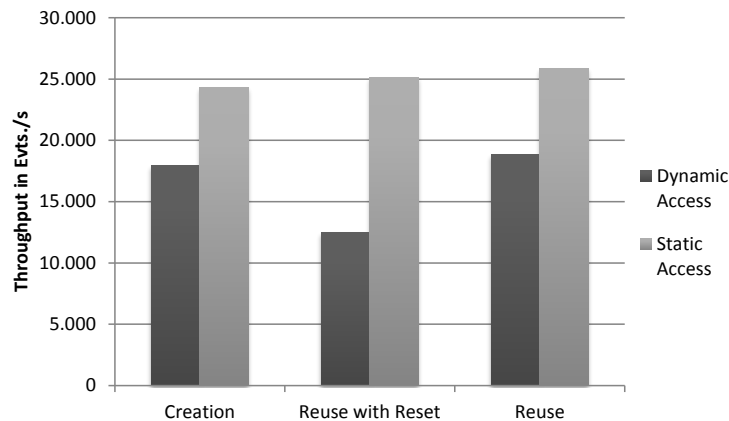


Figure 7.2: Normalization throughput with different methods of field assignments

involved. For the static access, also the two reuse variants are faster than the creation variant. This improvement is because no new memory has to be requested with each event. However, the reuse with a state reset is slightly slower for the static access and significantly slower for the dynamic access. Such behavior is expected, because the reset requires more work, but is necessary for a functioning normalization procedure. Therefore, static property access with reuse and reset is the preferable variant for field assignments.

7.4 Rule Organization

One of the main challenges for the rule-based normalization is still the finding of the rule that is able to normalize a given event. Assuming the deployment of a SIEM in an enterprise network, there are potentially hundreds if not thousands of different event formats, each requiring a custom normalization rule. Within this large pile of rules in the KB, the SIEM must find a matching rule for each incoming event record, ideally fast enough to handle the extremely high throughput.

A straightforward method of finding a matching rule for a given event record would be to test the regex of each rule against the record. If there is a rule matching the event, then the right rule would eventually be found. On average, it would take the SIEM $\frac{n}{2}$ tries to find the right rule among n rules in the KB. Assuming a matching throughput of 100 k evt./s for the Named-Regexp library from Section 7.2 and a set of 1000 normalization rules in the KB, then the SIEM would be able to normalize as few as 200 evt./s per thread, which is unacceptable for a SIEM. For sure, a better solution than the simple iteration of all rules has to be found.

A more efficient solution than the iterative testing of all normalization rules requires that the structure and appearance of events in real-world environments are

incorporated into the KB design. Only if the unique properties of the events are regarded, a better organization of the rules can be achieved, which the rule search algorithm of the KB can leverage to find a matching rule faster.

7.4.1 Homogeneity and Structure of Real-World Events

In theory, the log events in an enterprise environment can originate from a wide range of sources and can have a variety of different, rather unstructured, formats. The reality, however, shows that the majority of events comes from a few sources that use a small number of different formats with a hierarchical structure. Only a minority of the events comes from a broader range of sources and is entirely unstructured [31].

Homogeneity The observation that a majority of events comes from a few sources can be explained with the fact that the main operations within an IT-infrastructure are performed by a few services or systems. If these key systems are properly monitored and their logs are gathered, then many activities in the network can already be comprehended. As an example of a typical company infrastructure, the following setup can be considered.

The employees in a company use the company's *DHCP* service to connect their devices to the network and use the *DNS* to resolve hostnames to IP addresses. The Internet is accessed over a *web proxy* and the connections are checked by a *firewall*. The authentication of users and the management of IT-assets in the company domain is provided over a central *directory service*. In a service-oriented company, there is also a *web server* with the company's online presence and an *email service* to receive messages. To remotely access computers in the intranet of the company, services like *VPN* and *SSH* are used. Furthermore, each workstation and server is equipped with an *operating system* that additionally has some *HIDS* installed.

The described setup can be seen as the core network of a more complex network. A thing that adds up to the homogeneity of logs is that some of these services are provided by only a few different products or vendors. A DNS service is often delivered by the BIND software. A web server is typically implemented with an Apache web server. On the OS side, many companies are relying on Microsoft Windows. From this, it becomes clear that a productive SIEM only has to support a small number of different event formats to derive what is happening in the network.

Structure The structure of events is another factor that is not as random as it first seems. In fact, the observation of real-world events shows that a majority of events is either wrapped in the Syslog or Windows event formats, because the previously mentioned services mostly run on a type of UNIX or Windows system that produces the corresponding formats. The wrapping formats hold a more application-specific

CHAPTER 7. IMPROVING THE SPEED OF EVENT NORMALIZATION

format, which again has its own structure. If the application-specific format is even further structured, there is a chance that there is another sub-format included in it. Such a case of altogether three layers of event formats is especially typical for applications that produce many different event types.

An example of an application with a deeply structured format is the Snort NIDS [106] on a Linux machine. Snort has many different event types due to its high number of alert signatures. Each alert signature produces a particular event type that has to be distinguished by an event consumer. Figure 7.3 shows two event records as Snort has produced them. Both records are wrapped in the mentioned Syslog format (dark gray) and contain a structured Snort-specific format (lighter gray). The Snort-specific format, again, is structured by itself and wraps a more concrete message (lightest gray) for the alert information.

Mar 1 16:02:40 bastion snort:	Mar 1 16:02:40 bastion snort:
[1:648:7] SHELLCODE x86 NOOP	[1:1807:10] WEB-MISC Chunked-Encoding transfer attempt
[Classification: Executable code was detected]	[Classification: Web Application Attack]
[Priority: 1]:	[Priority: 1]:
{TCP} 4.152.207.238:3521 -> 11.11.79.83:80	{TCP} 4.152.207.238:3718 -> 11.11.79.84:80

(a) Snort event of type 648, i.e., an attempt to execute shellcode

(b) Snort event of type 1807, i.e., an attempt to inject commands

Figure 7.3: Two log events produced by Snort with a similar structure

The type of format outlined in Figure 7.3 can be called hierarchically structured, because there are many different formats wrapped in each other. Together, many application logs from Linux systems and the Snort NIDS can be sorted into this hierarchy. Figure 7.4 shows an exemplary format hierarchy from the Snort perspective.

The fact that many events are hierarchically structured can help in the normalization of events. Instead of seeing an event as one bulky piece of text, an event can be divided into multiple pieces that can be handled individually.

7.4.2 Organization of the Knowledge Base

The KB in the SIEM is one of the bottlenecks for normalization since it has to identify a matching normalization rule for a given event from a large number of available

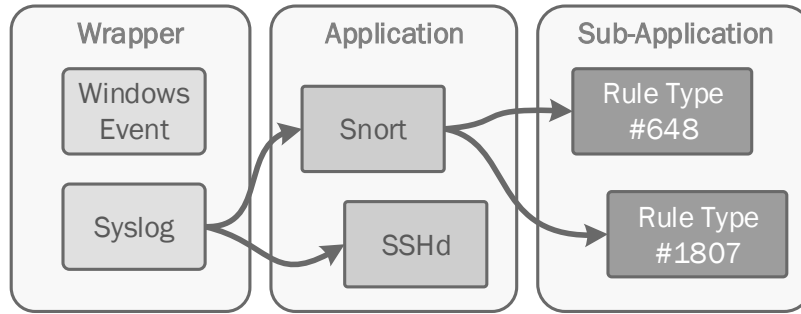


Figure 7.4: Hierarchy of event formats on the example of Snort events

normalization rules. With the insights about the special homogeneity and structure of real-world events, there are multiple ways on how the organization of matching rules in the KB can be improved to find a rule faster and to match an event more efficiently. In the following, one way of optimization is proposed that is based on the decomposition of event formats by their hierarchical structure.

Flat Knowledge Base In the flat form of a KB, a normalization rule consists of an NGRE and multiple static fields that cover an event in its entirety and do not consider its hierarchical structure. As a result of this design, a single normalization rule can only normalize one specific event type. Furthermore, the event log of an application that potentially has a multitude of different event types can only be normalized with the same amount of normalization rules. Figure 7.5 demonstrates how the two Snort events from Figure 7.3 would be normalized with a flat KB.

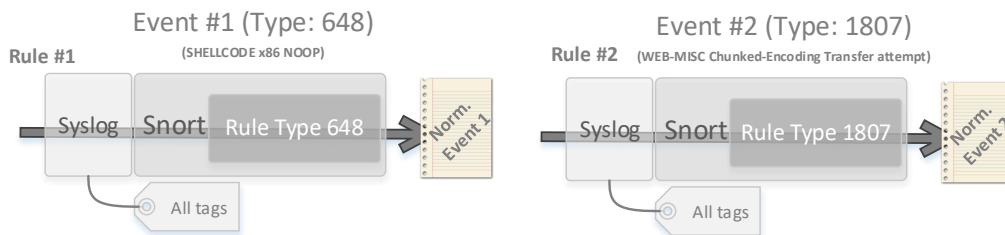


Figure 7.5: The matching of the two events with a flat KB

The Figure shows that the two different types of events are normalized with two distinct normalization rules, namely Rule #1 for type 648 and Rule #2 for type 1807, which are illustrated as a set of gray boxes. Within each of the larger boxes, multiple smaller boxes with different shades of gray represent the structural parts, i.e., Syslog, Snort, and rule type, of an event type. Looking at both rules, it becomes clear that they are sharing the same Syslog and Snort components with similar tags. The only

CHAPTER 7. IMPROVING THE SPEED OF EVENT NORMALIZATION

difference in the rules are their inner parts that are specific to the event type. Obviously, a flat rule comes with many redundancies that can be reduced with a more fine-grained organization of normalization rules in the KB.

Hierarchical Knowledge Base The redundancy of the flat KB can be prevented by applying the hierarchical structure of the event format to the normalization rules. In such a hierarchical KB, normalization rules are organized in multiple levels like the Snort events in Figure 7.3. In other words, an event is not mapped in its entirety to a normalization rule, but every structural part of it is mapped to a separate normalization rule. Thus, similar structural parts of events can be reused across many event types and the structural parts that are already known and have been defined do not need to be reformulated for a new type of event. Figure 7.6 shows how the two Snort events can be normalized with a hierarchical KB.

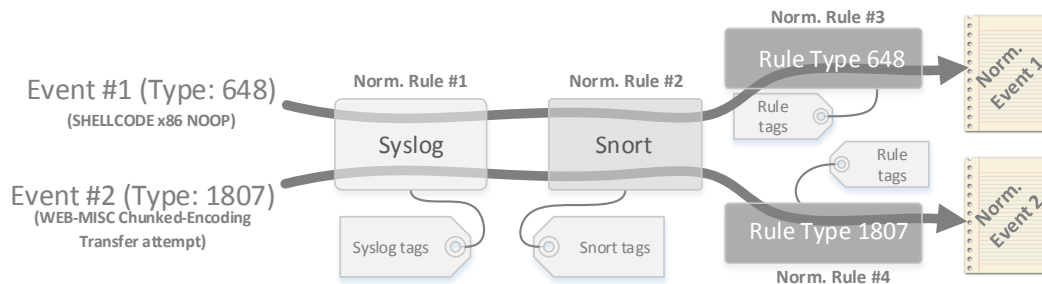


Figure 7.6: The matching of the two events with a hierarchical KB

The difference to the flat matching can be found in the way the normalization rules are organized in levels and how they are applied to the two events. While in the flat matching each event was only handled by a single rule, the events are now handled by the rules #1 and #2 for the first two steps, because they share the same wrapper and application-specific format. Only in the last step, they are handled by two separate normalization rules that respect their differences. All kind of tagging as well as the regexes solely apply to the currently matched sub-format and are independent of each other.

7.4.3 Matching with a Hierarchical KB

The organization of rules in the hierarchical KB is inspired by the typical structure of event formats and can, therefore, more efficiently match events. In the following, the matching algorithm and the resulting differences to the flat KB are outlined.

7.4.3.1 Matching Algorithm

In general, a KB-based matching algorithm has the task to find an applicable rule to a given event according to a given *rule context*. The set of rules in which the algorithm looks for the matching rule is called *candidate rules*, which indicates that these rules are candidates for a possible match.

In the case of a flat KB, the rule context is the given event without any additional information. The candidate rules are all available rules in the KB. Consequently, a matching rule is found by going through all available rules and testing their regexes for applicability on the given event.

The hierarchical KB defines its matching context as the remaining unnormalized part of the given event and the level of the format that will be matched next, such as wrapper, application, or sub-application level. Based on the current level, the candidate rules are all the rules that are categorized into the current format level. To normalize an event with a hierarchical KB, the entire event is first tested against all rules on the *wrapper* level and the remaining unstructured message is extracted for further normalization. In the following step, the previously extracted message is tested against all the rules on the application level and the remaining unstructured message is extracted again. So, over multiple steps, the matching goes deeper and deeper into the hierarchy of the KB until an event without a remaining unstructured part is found. Only then, the event has been fully normalized.

In the concrete example from Figure 7.4 and 7.6, both Snort events are checked against the Syslog and Windows rules. When Syslog has been identified as the matching rule, the relevant Syslog information is extracted, and the given tags are applied to the unified event. Based on the extracted message part of the Syslog event, the Snort rule is found as the matching rule and all Snort-relevant information is extracted and assigned to the unified event. In the last step, the remaining Snort message is used to find the rule in the sub-application level, where either the rule with type #648 or #1807 applies. Both rules do not have a message part on their own, meaning that no further matching is performed, and the event is fully normalized.

7.4.3.2 Comparison of Approaches

The hierarchical KB has several advantages over the flat KB in terms of structure and performance.

The hierarchical approach is generally more organized than the flat approach in the case where multiple event types with a very similar structure have to be normalized. For the flat case, all the similar event types have to be expressed in independent rules that share a significant part of their regexes. In the hierarchical case, the parts of an event type that are similar, are expressed in joint rules that are shared for the normalization of the overall event. For Snort events, the two similar event parts of

Syslog and Snort can be shared across all event types that use the Snort or even Syslog format. Another benefit of separate rules for event parts is that static fields can be specified at the event part where they apply. For example, the Snort-rule can be tagged with information that generally applies to Snort, such as the fact that the event is network-based (`tag.domain = net`), whereas the Snort alert-specific rule could be tagged with information that only applies to that particular alert.

Also from the performance view, the hierarchical structuring of normalization rules has benefits. As already outlined in the description of the matching algorithm, the hierarchical approach works with a significantly reduced set of candidate rules. For each matching level, only the rules are applied that have the previously matched format as a parent format. For example, once an event has been identified as Syslog event, only the application formats that are wrapped by Syslog are checked against the event, but not any sub-application formats. In the flat approach, an event would be checked against all event types, including specific application events. An illustration of the two types of matching is shown in Figure 7.7.

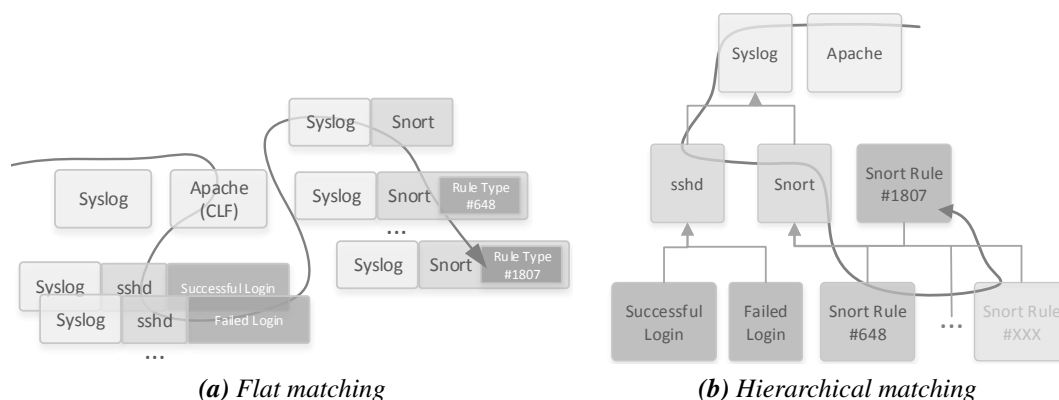


Figure 7.7: Comparison of matching in flat and hierarchical KB

On the left-hand side is the flat matching, which shows that many unrelated rules, such as multiple for the SSHd application, have to be checked before the right Snort rule is found. On the right-hand side, the hierarchical matching only checks the topmost rules and only goes deeper into the hierarchy when one of those matches. The other sub-application formats of Snort are only checked, if the event originates from Snort.

The example shows that the number of rules that have to be checked in the hierarchical case can be significantly smaller, especially when there are many applications with many different event types. How the hierarchical approach compares against the flat approach in a real-world setup is shown in more detail in the evaluation later in this section.

7.5 Searching Rules Efficiently

The reorganization of the KB already has a major performance benefit for the normalization. However, the problem that the normalization algorithm has to search through all candidate rules with different rule contexts still persists. Therefore, the searching algorithm is another suitable point for optimizations.

Altogether, we propose three different optimizations of the candidate search algorithm that should again significantly reduce the event matching.

7.5.1 Rule Indexing

The first optimization aims for a more efficient selection of the candidate rules among all available rules. Especially in the case of a hierarchical KB, the candidate rules are a subset of all available rules and are filtered out by several criteria.

- **Standalone:** A rule with this property matches an event that can stand on its own and is not wrapped into another event format. In other words, these rules do not have a parent rule.
- **Level:** The rules in a KB are organized hierarchically by multiple levels. Each level handles a different fragment of an event. Generally, a rule does only have to be checked if it has an equal or lower level than the already matched level of the event context.
- **Parent:** Many rules can only appear as a sub-format, also called a child, of another format, i.e., the parent format. A rule that is not standalone and not a child of an already matched rule is not applicable to the current event context.

The selection of rules with these criteria from the overall set of rules can be seen as a typical indexing problem. Three different strategies for the rule indexing are proposed and are later evaluated for their performance.

- **Iteration:** The simplest form of indexing, which is actually no indexing at all, is the iteration through all rules during which each rule is checked for the required criteria. Since each rule has to be touched to check the criteria, this method is the slowest.
- **Criteria Indexing (CQEngine):** The indexing of rules by their criteria is an easy way to access individual rules faster. We have selected the Java-based CQEngine [107] library as a concrete indexing implementation, because it promises to be a high-performance engine with a powerful query interface.

- **Text Indexing (Lucene):** A special form of indexing is provided by the Lucene indexing library, which is designed for high-performance text indexing. In addition to the simple indexing of the criteria for each rule, also the rule's regex content is indexed. This approach has the effect that candidate rules can be further filtered by checking the current rule context against the criteria and as well as the event content's similarity to the indexed regex. A more detailed description of that approach can be found in the work by Azodi et al. [23].

7.5.2 Candidate Rule Caching

The concept of rule indexing already allows to improve the selection speed for the candidate rules, but it is still required to select them from many rules in the KB. An approach to further improve access to the rules is to cache selection results for different sets of indexing criteria.

7.5.3 Rule Selector

There are wrapping formats that are incorporating information about the type of event they are wrapping. Syslog is a good example of such a format, because it has a specifier for the application that produced the wrapped format. Such a specifier, which can also be seen as a selector, can help to reduce the number of candidate rules or can even directly point to the exact rule that would apply. Figure 7.8 has an example of a Snort event and shows how the rule-matching can be reduced to a minimum.



Figure 7.8: Direct rule selection for a Snort event

7.5.4 Priority Lists

In cases where no rule selection is available and where still many candidate rules are left to check, a more efficient method of rule iteration is desirable. An approach that addresses this need is the prioritization of rules by the number of times they successfully matched.

Since the creation of a prioritized list of rules with each incoming event and set of candidate rules would consume too much time, we propose to prioritize only the rules that are stored in a cached result. Furthermore, the reordering of the rules by their frequency should be performed after some time interval that takes the processing overhead and actuality of the frequency values into account.

7.5.5 Evaluation

The above approaches have been implemented and evaluated for their performance as part of our prototypical REAMS. The goal is to find the optimal combination of approaches to achieve the best speed. For the evaluation of the approaches, we have performed two experiments that serve different normalization scenarios. Within the experiments, each combination of optimization approaches was run 10 times per event dataset and on 8 parallel normalization threads.

Experiment 1: Normalization of Hierarchical Logs The first experiment focuses on the normalization of log events that are highly hierarchical. As such, we have taken 69 030 individual Snort events from a network security challenge called Honeynet Challenge #34 [108]. Since all these Snort events have a rather wide range of event types, many normalization rules for Snort rules are required. Instead of specifying all these rules manually, we have generated appropriate normalization rules for all common Snort rules that are either specified in Snort’s default *rule snapshot* [109] or the open rules from *Emerging Threats* [110]. With these generated rules, we were able to normalize all available events with various combinations of our normalization approaches. The results can be read from the diagram in Figure 7.9.

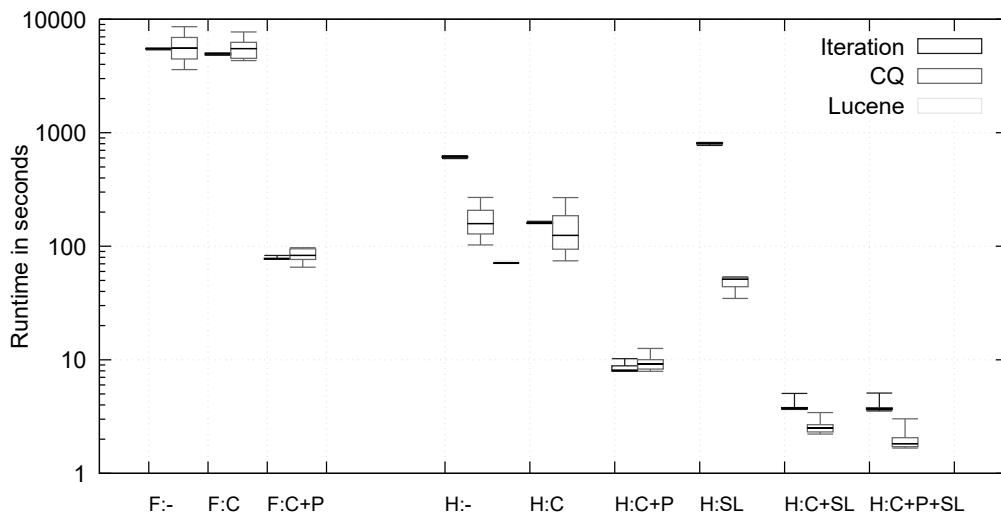


Figure 7.9: Evaluation of various KB optimizations on 69030 Snort events, **left (F)** - flat knowledge base, **right (H)** - hierarchical knowledge base (**C** - Cache, **P** - Prioritization, **SL** - Rule Selector)

The diagram reveals that the hierarchical implementation performs at least a magnitude better than the flat implementation. Although more rules have to be applied for

the hierarchical approach, since there are additional wrapper and application formats, the flat rules have much more complex regexes that need to match larger parts of the events. What can also be deduced from the performance results is that the approach with CQ rule indexing is generally faster than the iterative approach, but only when used with the hierarchical KB, as in the flat KB no criteria are there to be indexed. Lucene, as the third indexing approach, performs better than the iterative and criteria indexing approach. However, due to its employed mechanisms, it cannot make use of rule caching and prioritization and is also not able to outperform these optimizations for the other approaches

From the remaining optimizations, the rule prioritization and the rule selectors achieve the biggest performance benefits. In comparison to caching, these optimizations are effectively reducing the number of rules that have to be checked from the candidate rules. Furthermore, the runtime of both approaches is most stable between test runs, because the order in which rules are checked is deterministic now. In the basic hierarchical KB, the rules are randomly ordered with each test run, resulting in widely varying rule search times per run. The caching of rules has a performance benefit if many different candidate lookups have to be performed. Therefore, it has the most impact in combination with rule selectors (e.g., $H:SL \leftrightarrow H:C+SL$), because they produce a wide range of candidate rules. For smaller sets of candidate rules, the impact of caching is only marginal (e.g., $H:- \leftrightarrow H:C$).

Experiment 2: Normalization of Mixed Logs The second experiment evaluates the normalization performance when different types of application logs are used. Therefore, the events to be normalized consist of the Snort events from the previous experiment and additional Apache web log events that were shuffled into the Snort events. Altogether, there are now 76 659 events that again originate from the Honeynet Challenge #34. Comparing the test results with the results from experiment 1, then there are almost no changes in the normalization performance for the different combinations of approaches. This is because there are only 11% more events that are already normalized with one additional CLF rule. However, this also shows that even intermixed hierarchical log events can be normalized without significant performance impacts.

7.6 Conclusion

The different types of optimizations for the normalization algorithms show that considerable speedups in the normalization throughput can be achieved. The application of an NGRE to an event is one of the main operations that is performed during the matching and can be improved by choosing a flexible and still fast regex implementation. Already the standard Java regex implementation can achieve a matching

CHAPTER 7. IMPROVING THE SPEED OF EVENT NORMALIZATION

speed of around 100 k evts./s with a single thread for a preselected normalization rule, which is three times as fast as other regex implementations. As a second optimization point, the extraction and assignment of event information to the unified event format can be improved by switching from a dynamic field assignment with around 18 k evts./s to a static field assignment with object pooling to around 26 k evts./s for a preselected rule. As the last point, the optimization strategies for the rule organization in the KB can bring some additional performance benefits. In the case in which fully structured events have to be normalized, which is often the case, the finding and matching of the right normalization rule can be improved by multiple orders of magnitude by using a hierarchical KB in combination with rule prioritization, rule selectors, and result caching.

At the moment, the optimization is targeted to executing the normalization on a single thread. In the following chapter, we are going into the direction of parallelizing the normalization process by using multiple CPU cores and machines.

Chapter 8

Scaling Event Stream Processing

Related Publications

- David Jaeger, Andrey Sapegin, Martin Ussath, Feng Cheng, and Christoph Meinel. “Parallel and Distributed Normalization of Security Events for Instant Attack Analysis”. In: Intl. Performance Computing and Communications Conference. 2015 [24]
 - David Jaeger, Feng Cheng, and Christoph Meinel. “Accelerating Event-Based Attack Detection with a Distributed In-Memory Platform”. In: Intl. Conference on Dependable, Autonomic and Secure Computing. 2018 [25]
-

The increasing number of incoming events is one of the main challenges a SIEM has to face in an enterprise network. Of course, it is important that each of the workflow steps is optimized to deliver the best performance, but the sole optimization of algorithms is not sufficient to handle growing volumes and velocity. An essential requirement for a Big Data system is to scale in both ways, horizontally and vertically. In this chapter, we present a scaling approach that enables multi-processing for all workflow steps by employing a multitude of threads and machines.

Due to the fact that the parallelization should apply to any processing step, we are just focusing on the essential steps of normalization and persistence in the following.

8.1 Vertical Parallelization by Multi-Threading

With the vertical parallelization, multiple simultaneous threads are utilized to improve the event throughput. The central component in this parallelization is a so-called *message distributor*, which collects all incoming events and then dispatches

CHAPTER 8. SCALING EVENT STREAM PROCESSING

them to worker threads that have free capacity. Theoretically, the parallelization is linearly scalable, but the reality shows that this is difficult to achieve. In an enterprise environment, such a distributor could be confronted with millions of messages per second, so that even the tiniest inefficiency in this component has a significant impact on the event throughput. Figure 8.1 shows which role the distributor plays for the task of normalization.

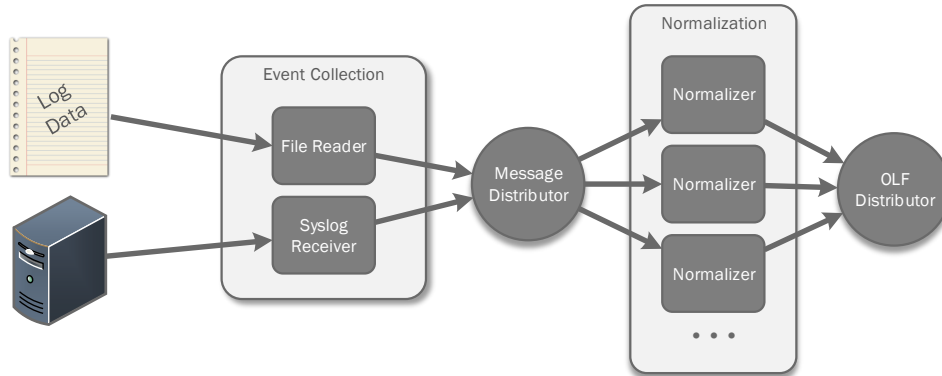


Figure 8.1: Vertical parallelization with a message distributor

The organization of the components resembles that of a typical producer-consumer scenario, in which the *event collectors* are producers and the *event normalizers* are consumers. The *blocking queue* and the *disruptor* pattern presented in Section 4.1.1 are implementing the producer-consumer scenarios. Although the blocking queue is a well-known software pattern, it struggles with high objects throughputs far beyond 100 k evts./s, because each time an object is consumed, all consumer threads have to wait for the finalization of the operation to ensure the queue's consistency. In contrast to that, the disruptor allows that multiple consumers and producers can access various buffer slots simultaneously without blocking each other. Furthermore, the disruptor has the concept of object pooling already integrated by design, because each slot acts as a pooled object that neither has to be recreated nor disposed of. Consequently, a thread that accesses a slot does not need to lock the ring buffer, because there are no modifications required.

Since the message distributor takes a key role in the parallelization, we evaluate both distribution approaches for the normalization step. In our experiment, we are normalizing 10 million Apache Web server log entries with a varying number of worker threads. Each configuration of worker threads is run three times so that an average normalization throughput can be determined that is less prone to outliers. The normalization procedure is using the previously presented optimization strategies from Section 7.4 in combination with dynamic object access.

8.1.1 Locking Implementation with a Blocking Queue

The blocking queue is part of the Java standard library and does not require further configuration. It is therefore easy to integrate into our implementation. The reader and the normalizers all share a single blocking queue with a capacity of 2^{14} (16384). Figure 8.2 shows the throughput of the normalization with a varying number of normalizer threads.

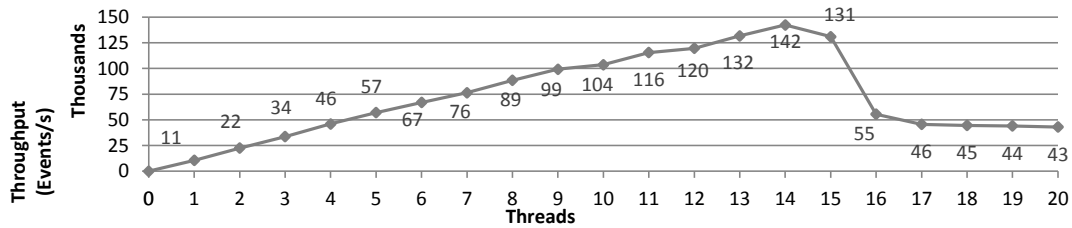


Figure 8.2: Normalization performance with a blocking queue on a 16-core machine

Up to the point of 14 threads, the throughput is growing almost linearly with the number of threads. However, starting from 15 threads, the throughput is declining. This behavior is unexpected, because it implies that 2 out of the 16 cores on the machine are not utilized. One factor that might play a role is the reader thread that partially occupies one of these cores. Still, there is one more core that does not seem to be utilized. To better understand what is happening with this setup, we have monitored the runtime behavior of 15 normalizer threads, as shown in Figure 8.3.

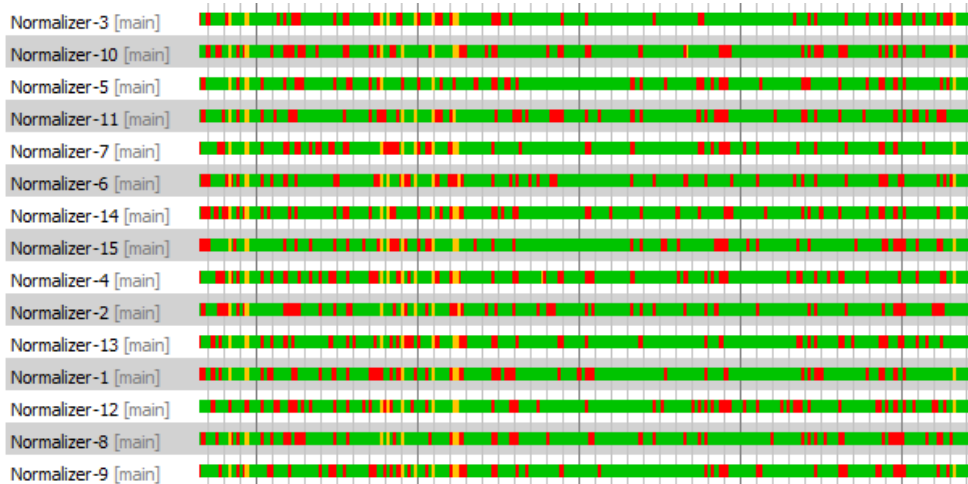


Figure 8.3: Thread behavior during normalization (green - thread running, red - thread blocking, yellow - thread waiting)

At first glance, the small red intervals in between the green bars of the normalizer

threads are most apparent. Green indicates that the thread is running, while red shows that the corresponding thread is currently in a blocking state, which means that it is not further normalizing events. Considering the number of red intervals, it can be inferred that a significant amount of time is spent on waiting and switching between the two thread states. Furthermore, since the graph has rather low precision, it can even be assumed that there are many more thread interruptions than visually shown.

The number of thread interruptions increases with the number of threads, because every single thread then has to wait for more other threads to take data from the queue. For this reason, the throughput also starts to drop with a certain number of threads. The blocking becomes so extensive that each thread consumes most of its time for waiting. Due to the way the blocking queue is designed, there does not seem to be a solution to this inherent problem.

8.1.2 Lock-Free Implementation with the Disruptor

The disruptor is more challenging to integrate, as it requires a thoughtful configuration and needs to be adapted to the type of object it passes. As main options, there are the multiplicity of producers and consumers as well as the size of the ring buffer. Both depend on the way the disruptor is accessed and have a direct impact on its throughput. For multiplicity, a single producer and consumer are least flexible but achieve the highest throughput per thread, while a multitude of producers and consumers have the worst throughput per thread but enable complex and highly parallelized processing scenarios. For the buffer size, the documentation [64] of the disruptor implementation specifies that the ring buffer size should be a power of 2 and tailored to the average throughput and number of producers and consumers.

Instead of running our experiment on just one particular configuration, we are running it on multiple combinations of configurations to find the most optimal setting. In each run, the combination of ring buffer size, number of consumer threads, and utilized CPU cores is changed and the resulting normalization throughput evaluated. The values for the configuration parameters are changed as follows:

- **Buffer sizes** are iterated by their exponents from 7 ($2^7 = 128$ slots) to 22 ($2^{22} = 4\,194\,304$ slots)
- **Thread numbers** are tested from 1 to the number of available CPU cores
- **CPU cores** are varied by conducting the experiments on two different machines, one with 4 CPU cores and 4GB RAM and another one with 16 CPU cores and 32GB RAM¹

¹Deployed on VMware ESXi host with 256GB RAM and 8x Intel Xeon X7560 CPUs @ 2.27GHz

CHAPTER 8. SCALING EVENT STREAM PROCESSING

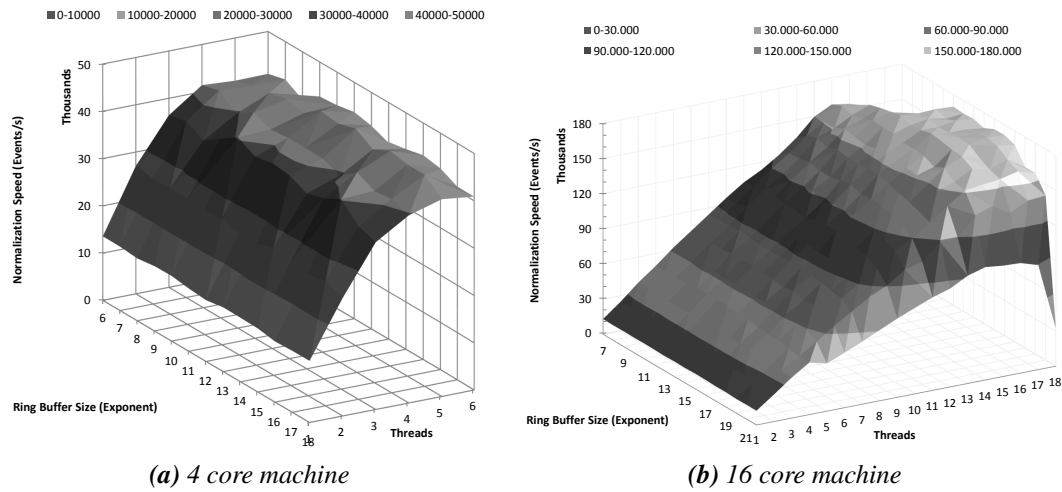


Figure 8.4: Disruptor normalization with varying buffer sizes, number of worker threads, and CPU cores

Figure 8.4 shows the results of the experiments in two surface charts. Similarly to the blocking queue, the throughput grows almost linearly with the number of threads. Other than with the blocking queue, the throughput keeps on growing to the number of available cores, which is what we would have expected from the beginning. This indicates that the disruptor better utilizes the cores and is not hindered by locking.

The size of the ring buffer is an extra configuration parameter that influences the throughput. Beginning from the smallest size exponent, the throughput is fluctuating or rising with a higher size exponent. However, starting from an exponent of 18, the normalization performance is decreasing dramatically. On average, the best performance is achieved with size exponents between 13 (8192 slots) and 17 (65 536 slots). Considering all the configuration parameters of the experiments, we could reach the highest throughput of 172 k evts./s with 17 threads and a buffer size of 2^{17} on 16 cores. The highest mean throughput was reached with a buffer size of 2^{13} .

As a comparison, we have put the throughput graphs of the blocking queue and disruptor into Figure 8.5. As it was already expected from the conceptual comparison of both approaches, the disruptor throughput is generally higher than that of the blocking queue. At the point of their highest speed, there is a difference of 28 166 evts./s ($\approx 20\%$).

8.1.3 Limitation of Scaling

Although the vertical parallelization allows scaling the normalization by just adding more hardware to a single machine, there is one drawback that has to be considered.

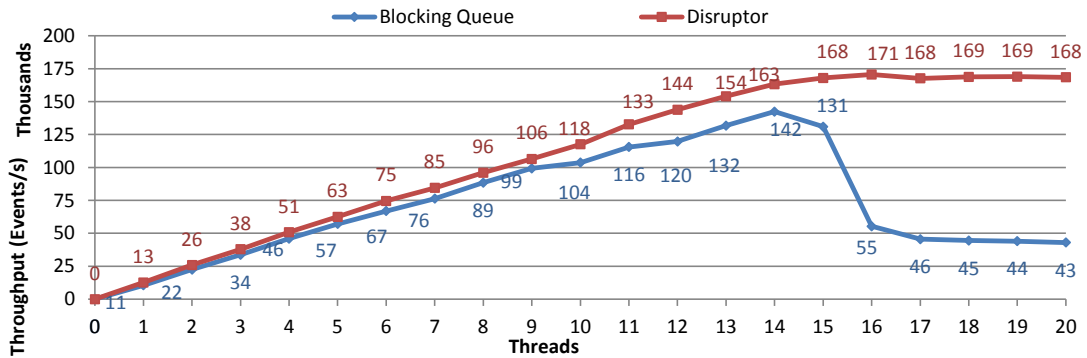


Figure 8.5: Normalization performance with disruptor (buffer size 2^{13}) and blocking queue on a 16 core machine

The more hardware resources a single system already has, the more expensive any additional resources become. For example, as of 2017, a machine with at least 48 physical CPU cores costs more than \$10 000. As a consequence, it is difficult to reach normalization throughputs necessary for an enterprise setup for an affordable price.

8.2 Horizontal Parallelization by Distributed Processing Nodes

The horizontal parallelization solves the problem of scaling by distributing work to multiple processing nodes. Usually, these nodes are low-end machines that have cheap hardware and are easier accessible than one big machine for vertical scaling. Still, a larger number of such low-end nodes can form a compute cluster that is more capable than any single machine could be. In the following, we show how a well-performing normalization can be accomplished in such a cluster.

As the main idea, we propose an architecture similar to that of Figure 8.6. In comparison to the vertical parallelization, the event collection is divided from the normalization by using two types of nodes, i.e., a single forwarding node and multiple normalizer nodes. The forwarding node is responsible for collecting all raw events and is forwarding them to one of the normalizer nodes with free capacity. The normalization nodes are similarly constructed as the master node in the vertical parallelization, i.e., they consist of a normalization component that manages multiple normalizer threads.

The normalizer nodes are working independently of each other, meaning that they do not require any synchronization between each other. This is possible because the normalization of an event is a self-contained operation that does not need information

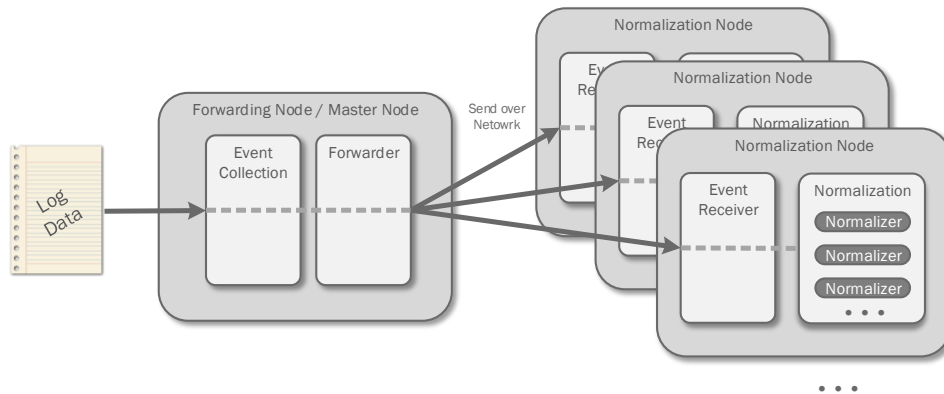


Figure 8.6: Horizontal parallelization of normalization with network distribution

from the outside or the context. This fact majorly reduces the complexity of the distributed normalization to an even dispatching of work tasks to the processing nodes and efficient processing of the tasks on these nodes. Both problems are not new and have already been addressed by the Big Data community. The solution are so-called distribution frameworks, such as Hadoop, Storm, or Spark, which were introduced in Section 4.1.2. Another way of solving these two problems is a custom parallelization approach that is based on our disruptor pattern from the last section. Although the disruptor is initially designed for fast inter-thread communication on a single machine, it can also be used to parallelize tasks on multiple machines simultaneously.

Generally, pre-existing frameworks have the advantage that much effort was put into making them adaptable to many different use cases and improving their processing performance. However, this generic design also makes these frameworks rather bloated and peak performance is sacrificed for some corner use cases. Our disruptor approach, on the other hand, is only developed as part of work and crafted for the normalization use case. Therefore, it is more performance optimized, but is rather difficult to be reused elsewhere.

To make the integration of existing distribution frameworks into REAMS possible, our normalization algorithm needs to be reimplemented as a module for each of the frameworks. We have performed this reimplementations for the four most common frameworks, i.e., Storm, Spark, Heron, and Trident. In the case of the disruptor pattern, no further reimplementations were required. To find the most efficient parallelization for multiple hosts, we compare these known distribution frameworks with the disruptor approach through multiple properties, i.e., data exchange, reliability, and processing speed on a single node.

8.2.1 Data Exchange

In a distribution framework, the data to be processed has to be made available to the worker nodes. Depending on the framework, this can be accomplished in two different ways.

Passing Any incoming data resides at the master node and is managed by it. The processing is distributed by splitting up the incoming data into small chunks of work that are then forwarded to the corresponding worker nodes. The worker nodes themselves only receive their small data chunks. This method of data exchange is usually used with stream-processing or sometimes with micro-batch-processing. The frameworks Storm, Trident, and Heron as well as the disruptor approach use *passing* to distribute work.

Distributed Reading All data to be processed is readable from the worker nodes. A master node is responsible for distributing chunks of work to dedicated workers. It does so by telling the worker from which offset the data has to be read and how the processing partitions look like. Since this method requires that all data is previously synced with the workers, it is mainly used by batch or micro-batch processors that are not that time-critical. A possible alternative to a previous sync would be the provision of the data in a message queue. Among others, Spark uses *distributed reading* to distribute work.

The data exchange by *passing* is more suitable to stream-processing than *distributed reading*, which also makes *passing* the most suitable option for our REAMS implementation. All the presented frameworks already have one of the data exchange methods implemented. For our customized disruptor approach, we propose an optimized passing mechanism that is based on the implementations of the stream-based frameworks Storm, Trident and Heron.

8.2.1.1 Data Serialization

To exchange events between nodes, the event objects need to be serialized into a byte stream that can be transmitted over the wire. How an object is represented in a byte stream is specified over a serialization format and has a significant impact on the serialization time and the space needed on the wire for transmission. When looking at the landscape of distribution frameworks, then the serialization with *Kryo*, *Avro*, and *Protobuf* seems to be most common among them. A more complete list of serializers with a comparison of their key performance factors has been compiled by Smith [111]. The libraries Avro and Protobuf provide a rather exact specification of the serialization format, whereas Kryo comes with a default serialization but is also flexible enough to be adapted to any other kind of serialization. Storm and Heron

are using Kryo as their default library for object serialization and allow to optionally switch to either Avro or Protobuf.

A closer investigation of the three common formats reveals that there are two major methods in which object fields are brought into a byte stream. One, which is used by Protobuf, is by generating a class according to a defined data model. This class has getters and setters for each model field and later directly transforms these fields into the byte stream. The other way, which is used by Avro and Kryo, is to dynamically read the fields of an existing class either by introspection or from a description of the data model. Both approaches have disadvantages when it comes to serialization performance. The first way theoretically would perform well, but the necessity to use a generated class results in time-consuming copying of data from our custom event object that allows direct property access, which was explained in Section 7.3.1, into that class solely to transmit an object. In detail, this leads to an additional allocation of memory and the creation of a new object. The second way uses introspection to get and set values of an already existing class. Also, this strategy is generally slower than static access. Another disadvantage of Kryo and Avro serialization is that optional data fields are not allowed. Especially in the case of event normalization, only a small number of fields, like 10-20, are actually used and need to be transmitted. Requiring the transmission of all fields, therefore, results in a size overhead of around 80-90%.

Our goal is to overcome the performance faults of the existing formats. First of all, the dynamic access of object fields with introspection has to be prevented. This is, in fact, easy to achieve with our proposed slot concept, because there, all fields can be accessed at the same time through an object array. The information about the type and length of the fields, which are required to serialize the fields, are available through a metadata helper that is generated together from our event data model. As a serialization, we use the encoding strategies of the Kryo library, such as a length-encoding for strings and numbers. As a second step, fields with a *null* value are not serialized. Of course, to indicate which field is encoded where, each field has to be prepended with an identifier. Although this requires more space for a single field, it still compensates for the size that would be necessary for adding all optional fields to the serialized stream. Altogether, these two steps combine the advantages of the existing libraries, i.e., the size and speed of Kryo's manual serialization as well as the optional fields of Protobuf, into one serialization approach.

8.2.1.2 Concept of Minimal Data Access

The serialization of an object into our data model still requires that all fields of the object are read, even if they are not transmitted over the wire as they have no value assigned. In fact, the information whether a field is relevant for transmission is already derivable during the event construction, because there all available object data

is assigned to the corresponding fields. In the case there is no value for a field, the accessor of the field is not invoked, and the field remains unset in the object. Our idea to improve the serialization is by keeping track of which fields have been set and later only reading these set fields. We propose to track this state in OLF via a bitset in which each bit corresponds to one slot in the field buffer. The bits are set whenever a setter for a field is invoked and reset when the object goes back to the object pool or a *null* value is assigned. As an example, Figure 8.7 shows how such a bitset could look like for an SSH event.

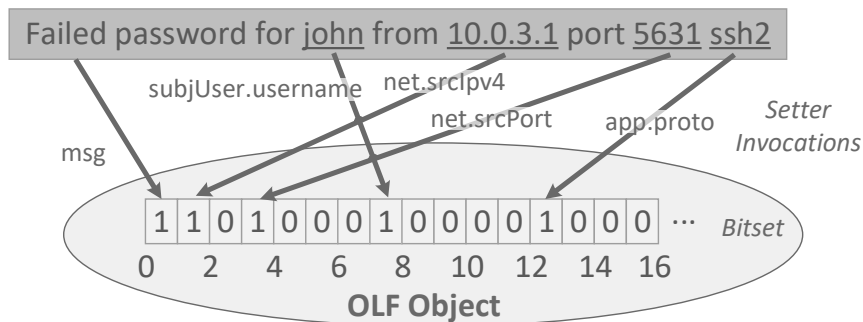


Figure 8.7: Bitset of set fields in an OLF object

The setting as well as the reading of the bitset is an operation provided by the CPU and requires minimal overhead. The fields that have to be serialized to the wire can thus be found out faster than iterating through all field accessors of the OLF object. In an experiment in which we are accessing the fields of already normalized web events, we could reach a throughput of ≈ 1.87 M evts./s by iterating over all fields in an OLF object. In contrast to that, the reading of only set fields as indicated by the bitset could reach as much as ≈ 2.94 M evts./s, which is $\approx 57\%$ faster. In other words, if a throughput of 500 k evts./s for the entire processing workflow is assumed, then the usage of the bitset concept can save almost 10% of the overall processing time. This time can efficiently be used for other tasks, such as a more complete data enrichment.

The bitset concept is useful for more efficient data transmission and for other parts of the processing workflow. Every component that is accessing all fields of the event can benefit from reduced access times. In particular, the persistence component, as well as the signature analysis module, are making use of the bitset, as shown later.

8.2.1.3 Data Transmission

After the serialization of an OLF object into a byte stream, this stream is transmitted to a remote processing node. In a computer network, this transmission is accomplished over the TCP/IP-stack. As all the existing distribution framework, we choose

the connection-oriented TCP protocol for the transport in REAMS, because the transmission is more reliable, especially with increasing network bandwidth.

The Java programming language is shipped with two major network libraries that allow building application protocols on top of the TCP/IP-stack. The Socket API is the simplest library to use, but also has the worst performance. In a first transmission prototype, we were able to only transmit ≈ 5 k evts./s per connection. Even assuming 10 processing nodes, then not more than ≈ 50 k evts./s can be normalized because of the limitations in transmission. Research by Welsh et al. [112] from 2000 confirms this bad performance of the standard library. However, they also show that native access to C sockets can achieve significantly higher throughputs. As a consequence of this shortcoming, the Java community introduced an extended library for scalable I/O in 2002 with the so-called Java NIO specification (JSR51 [113], JSR203 [114]). The crucial difference of this specification to the standard Socket API is that fast buffers are used to directly communicate between user code and I/O code in the operating system and the possibility to perform asynchronous non-blocking operations.

We have adapted our transmission prototype implementation to use these fast buffers and could achieve a throughput of 50 k evts./s for a single connection without further tuning of the socket parameters. This is an improvement by a magnitude by merely using data buffers more efficiently. The next step for further improvement is the switching to asynchronous calls for sending the data. In this mode, the sending thread can push a piece of data to the socket and check the outcome of the send operation later. In the meantime, the thread can prepare further event objects for transmission, e.g., by serializing them to a byte buffer.

The efficient handling of asynchronous socket calls is a challenging task and not straightforward to implement. Netty is a third-party network library that has the goal to simplify implementations against Java NIO by providing an abstraction that is easy to work with. It supports asynchronous socket calls and additionally reduces the copying of buffers to the necessary minimum. Therefore, at the time of writing this thesis, Netty has become one of the most used network libraries when speed is required. Also existing distribution frameworks, such as Storm and Spark, are using Netty as their network library. Due to the promising features of Netty, we have also moved our implementation to Netty and have implemented our serialization format as a module for Netty. In our experiment with the Netty version, we could reach a throughput of 150 – 200 k evts./s on a single sender thread, which is even higher than our pure NIO implementation.

8.2.2 Processing Reliability

Another factor that has to be considered for distribution frameworks is how they guarantee that each event is processed. Especially with network communication and dynamic addition and removal of worker nodes, events may be lost during transfer

or when a worker goes down. In the area of distributed processing, there are three semantics distinguished for reliability, i.e., *at-most-once*, *at-least-once*, and *exactly-once* semantics. Generally, implementing any of the two latter semantics costs a significant amount of performance and therefore reduces event throughput and delay. The exactly-once semantics does not even seem to be achievable for our use case, as we rely on stream-processing and the normalization is not a fully idempotent operation. Mainly for reasons of performance and simplicity, we will focus on the at-most-once semantic for the evaluation of the frameworks. If at-least-once semantics is really necessary, then a message queue at the beginning of the workflow can be used. The events in the queue would be acknowledged at the end of the workflow.

8.2.3 Single-Node Performance of Distributed Frameworks

The performance of the distributed frameworks on a single node can best be determined in an experiment. We have prepared an experiment in which we are normalizing around 76 million Apache Web server logs on a single machine² with 32 worker threads and 24GB of Java heap space. This single machine has a local setup of all solutions so that no remote network communication between multiple nodes is needed. This limitation in communication is important because we are only interested in the sole processing performance but not in the efficiency of the data exchange. Also, to create equal conditions for all frameworks, we have only applied basic configuration options without further tweaks. Figure 8.8 shows the normalization throughputs from all experiments.

The bar diagram reveals a clear difference in the performance of the solutions. The two slowest are the stream-based Storm with around 158 k evts./s and Spark Streaming with 182 k evts./s. Heron, which is the successor of Storm and also stream-based, is already significantly faster than the latter because it employs a new processing model and fixes performance faults of Storm. After Heron, there is no other common stream-based framework that can handle higher throughputs. Rather, it requires a framework with batch-processing to reach even higher throughputs. One such candidate is Trident, an extension of Storm, which is slightly faster than Heron by using micro-batch-processing. Another candidate is the pure batch-based Spark framework, which achieves around 330 k evts./s. What comes as a surprise is that Spark Streaming, which is based on the batch-based Spark, does not even come close to the performance of Spark. We assume this is the case because Spark Streaming chunks its stream into micro-batches and reinitializes the state of the normalizer with each batch. To reduce the processing overhead of the initialization, we have set the time-frame for a batch to 5 seconds. Still, this tweak does not seem to be enough to

²Dedicated machine, 64GB RAM, 2 x Intel Xeon E5-2630v3 (2.4GHz) with 16 physical cores using hyper-threading

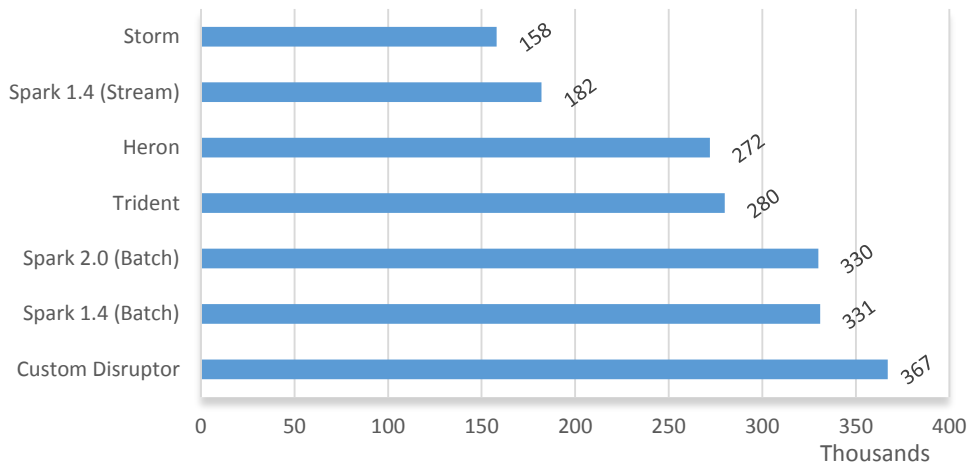


Figure 8.8: Normalization throughput in k evts./s for common distribution frameworks on a single node with 16 threads

reduce the impact of the reinitialization. However, at the top of all solutions, there is our custom stream-based disruptor approach that normalizes with up to 367 k evts./s.

The results of our experiments make clear that a customized solution can be faster if certain features, such as adaptability and flexibility, are sacrificed. The disruptor is around 100 k evts./s faster than the best stream-based framework and does even slightly outperform the fastest batch-based framework. Also, since we are interested in real-time event processing, the utilization of a batch-based framework, such as Spark, would not be ideal because of significant processing delays. As a consequence, we will focus on the use of the disruptor approach for distribution.

8.2.4 Multi-Node Performance using the Disruptor Pattern

We have described the building blocks for distributing processing steps over multiple nodes, i.e., the processing of events on a single node, the transmission of events between the nodes and the different forms of reliability. As the final step, we are combining these parts into an overall solution. The solution uses the disruptor pattern to process and transmit events in our custom serialization format, which is implemented as a Netty module. Together, we expect that these approaches can scale the performance with the number of processing nodes.

Our experimental setup consists of altogether 11 nodes. There is one master node that reads raw logs and forwards them to up to 10 worker nodes for normalization. The master node is a rather powerful machine that has 24 cores with 20 GB RAM. However, all normalization nodes are running on virtual machines that are comparable to commodity hardware, i.e., 4 virtual cores with just 4 GB RAM. According

CHAPTER 8. SCALING EVENT STREAM PROCESSING

to the results from Section 8.1, the disruptor is configured to have 4 normalization threads with a buffer size of 2^{13} . The results of our experiment are shown in Figure 8.9.

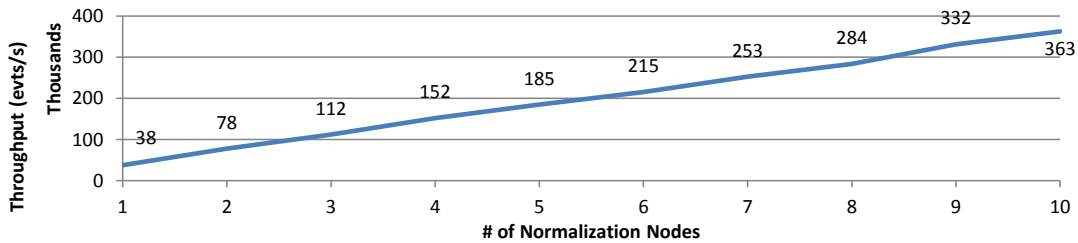


Figure 8.9: Performance of distributed normalization with varying number of normalization nodes

The graph in the diagram shows an almost linear growth of the normalization throughput with an increasing number of nodes. On average, each node contributes around 36 k evts./s to the throughput, reaching its highest with 363 k evts./s on 10 parallel nodes. With just 5 nodes, together having 20 cores, the distributed setup can already reach a higher throughput (+ 14 k evts./s) than the single 16 core machine from Figure 8.5. Consequently, a setup with multiple nodes is a viable alternative to a single-node setup, assuming that a significantly larger system is not affordable and multiple smaller commodity systems are at hand.

While running the experiment with 10 nodes, we have observed the CPU and network bandwidth utilization of the master node to identify potential performance bottlenecks. Regarding the CPU, the machine only has 3-4 cores of the 24 available utilized, which shows that the master can be operated with many more normalization nodes. The network bandwidth was a more critical factor because the transmission of the data to the other nodes almost fully occupied the 1 Gb/s Ethernet connection. Therefore, to further increase the normalization throughput, the Ethernet connection would have to be upgraded to 10 Gb/s.

8.3 Applying the Parallelization to Persistence

The parallelization of the processing workflow with multiple threads and nodes is the foundation for scaling event processing. In combination with various performance optimization, we can handle the event load of a large enterprise up to the point of normalization. A special role in the parallelization takes the persistence. It is the outlet of the workflow and depends on the performance of the storage system. If the storage system cannot keep up with the speed of normalization, then the persistence becomes the bottleneck and thus prevents real-time analysis. Therefore, it has to be

ensured that the storage system can scale with the workflow and that the persistence is aware of how the scalable capabilities of the storage system can be used.

As outlined in the overview of storage systems for REAMS, we consider SAP HANA to be a possible enabler for a scalable persistence, because it efficiently uses the fast main memory as storage and can also be deployed in a distributed manner on multiple nodes. The immutability concept of HANA additionally enables a faster write throughput and the reliance on a schema-based data model simplifies and increases the speed of advanced analysis.

In the following, we take a closer look at scaling the persistence with SAP HANA. As a first step, we focus on query optimization and the database interaction using HANA's Java Database Connectivity (JDBC) API. The goal is to build table schemas and queries that can be easily parallelized and interact with HANA in a way that multiple queries do not interfere with each other. In the second step, the persistence is expanded to multiple nodes.

8.3.1 Tuning the Database Access for Parallelization

The main part of RDBMS persistence consists of the issuing of SQL `INSERT` statements to the database. Each such query is responsible for transferring one event object into a database table. Considering our data model from Section 6.4, then this table will be the master table that contains a column for each field in the OLF model. Altogether, there should be 117 columns in this table.

A promising starting point for the optimization of SAP HANA queries is a comprehensive blog entry by Appleby on the *SAP Community Network* [115]. In this article, he covers multiple performance topics, such as how to create an efficient data model as well as how to construct SQL queries in a way that a high insert throughput can be achieved. Based on our ideas and the ideas of Appleby, we have created a list of potential improvements.

1. *Prepared Statements and Batching*: A well-known method to improve the insert performance of an application is to batch multiple similar queries with a so-called *prepared statement*. This statement is a template for a specific query and has placeholders for parameters to be provided for each instance of the query. The DBMS only needs to parse a prepared statement once when it is created and from then on only passes the parameters to execute this statement. In this way, the submission of a query to the DBMS is much more space and time efficient than a regular query. Furthermore, multiple instances of a statement can be collected in a batch before they are submitted to the database. As soon enough instances have been collected, they can all be sent to the database in a large data chunk and can be processed more efficiently, because only one type of statement has to be executed.

CHAPTER 8. SCALING EVENT STREAM PROCESSING

2. *Column-based vs. Row-based Tables:* A distinctive feature of the HANA database are column-based tables. In a column-based table, the values of each column are stored sequentially in contrast to a row-based table that stores records sequentially. The storage in columns has two major advantages. Firstly, values can be compressed more efficiently if a column has recurring values. Secondly, the sequential and compressed storage allows faster read and write access to the table if a query only addresses a selection of its columns.
3. *Minimal Insert Statements:* We have pointed out earlier that an OLF event usually only has a small fraction of all its fields set with values. An insert statement that covers all OLF fields would consequently have a lot of `NULL` fields that are transmitted to the DBMS and are processed there with the actual fields set. We propose to minimize the number of fields in each insert statement to reduce the amount of unnecessarily transmitted `NULLS`. Our idea to achieve this minimization is to replace the default insert statement that covers all OLF fields with many much smaller insert statements that cover a subset of commonly set OLF fields. The following steps would be necessary on the client side to implement this idea.
 - (a) The slot concept with the bitmap of set fields from Section 8.2.1 enables us to immediately deduce which fields are set within an OLF object. When the object is persisted, we also know which fields are relevant for the insert statement.
 - (b) For each object and its set fields, a separate insert statement is prepared and cached for later use. If there is a cached statement with the same set of fields, then the previously prepared statement is used. Using the insert statement, the current object is added to its batch.
 - (c) If enough objects have been grouped in a statement's batch, the statement is submitted to the database with only set fields and no `NULLS`.

Even though the described procedure would be optimal in terms of reducing empty fields in statements, there is the problem that the statements of some rare sets of fields would never have their batch filled. These statements would either be sent half empty or with considerable delays. A solution to this situation is to combine statements with similar sets of fields and to a single statement that uses the superset of all their fields.

4. *Partitioning:* The distribution of a table into multiple partitions is another possibility of better leveraging the processing speed of the DBMS. Each partition is handled independently by the DBMS so that operations on them are parallelizable. The parallelization is most efficient if the rows are evenly distributed between all partitions and queries involve records from all partitions.

CHAPTER 8. SCALING EVENT STREAM PROCESSING

5. *No Constraints*: Constraints, such as uniqueness of a field or a primary key, ensure the integrity of a table and are enforced with each change on the database. Whenever a query is about to violate a constraint, the DBMS prevents the execution of the query and keeps the table in a consistent state. Unfortunately, although constraints can ensure requirements on the data model, they also have a remarkable impact on the query performance. As a consequence of that fact, Appleby proposes to remove primary keys from tables whenever they are not urgently required. Since our master table is insert-only and no updates are required, the removal of the primary key is a viable option. An ID field that acts as the primary key could solely be assigned within the persistence component.
6. *Number of Connections*: Instead of just maintaining a single connection to the DBMS, multiple simultaneous connections can be established to transmit information in parallel. According to Appleby, HANA requires one physical core for each connection. Thus, on appropriate hardware, HANA could handle dozens of connections simultaneously. Of course, it has to be ensured that the queries issued over these connections do not interfere with each other, e.g., by locking tables or records. A drawback of multiple connections is that each connection usually has to be maintained by an individual thread on the client side, too. This can lead to a higher resource consumption that impacts the overall performance. Although this could be solved with asynchronous network connections, the JDBC API does not support this type of connection handling.
7. *Database Field Types*: the employed data type for a database field can have an impact on the query performance. This is because there are differences in the size and its predictability for each type. Appleby proposes to replace variable length fields of `NVARCHAR` with fixed length fields of `NCHAR`. Using fixed length fields indeed results in a higher amount of used memory, but also speeds up inserts as HANA can better predict the size of fields in advance.

We have implemented a persistence module for REAMS with all of the previously described improvements. Since it is not clear which performance impact each improvement has, we have evaluated multiple combinations of these improvements in an experiment. In this experiment, we have one master node with 24 cores and 20 GB RAM that takes over the task of reading, normalizing and persisting of logs. The master node is deployed as a virtual machine on the same host as our experimental setup for normalization. The persistence goes into an SAP HANA instance with 80 physical cores and 6 TB RAM. Table 8.1 shows the throughput values that were achieved during the experiment.

The usage of prepared statements in combination with batches indeed has a notable impact on the persistence throughput. It can be deduced that the batch size should neither be too small, i.e., below 1000, nor too big, i.e., above 8000, to achieve

Batch	Connections	CHAR Type	Table Type	Partitions	Throughput
8000	10	NCHAR	Column	5 (Hash)	159 133
8000	10	NVARCHAR	Row	5 (Hash)	100 016
8000	10	NVARCHAR	Column	5 (Hash)	166 226
8000	10	NVARCHAR	Column	1	172 342
8000	5	NVARCHAR	Column	1	134 877
8000	15	NVARCHAR	Column	1	171 599
15 000	10	NVARCHAR	Column	1	170 672
4000	10	NVARCHAR	Column	1	177 150
2500	10	NVARCHAR	Column	1	179 065
1000	10	NVARCHAR	Column	1	176 234

Table 8.1: Persistence throughput with various optimizations

the best results. An optimal throughput was observed with batches of 2500 queries. Also the number of connections supposedly is a possible point of improvement, as HANA can handle the load of multiple connections in parallel. Also here, the number should neither be too small nor too high. A set of 10 connections eventually resulted in the highest throughput. The first two rows in the table confirm that the column-based table surpasses the row-based table in persistence speed, as described by Appleby. In comparison to that, the partitioning and the fixed-length string type NCHAR have a slightly negative impact on the throughput. Nevertheless, the use of partitioning might still be an important performance factor for further analysis tasks on the persisted data.

Our experiment shows that all the proposed improvements of Appleby have an impact on the performance, even if it is not as big as in his experiments and sometimes even negative. The gap in the efficiency can be explained with the experimental setup of Appleby, because he copies the data to be persisted onto the database server and then reads this data as a CSV file directly into the database. Unfortunately, this procedure is not applicable to a stream-based event processing. All in all, the best combination of improvements could achieve a throughput of 179 k evts./s by combining 10 connections with a batch of 2500 queries on a column-based table.

8.3.2 Distribution to Multiple Nodes

The optimizations on the database connection have prepared the expansion of persistence to multiple nodes. We have made sure that inserts are independent of each other and do not block each other. Additionally, each node creates multiple connections to distribute the load to multiple threads on the DBMS.

As a next step, we integrate the persistence into the model we have proposed for

CHAPTER 8. SCALING EVENT STREAM PROCESSING

horizontal scaling. Each node gets a share of the overall event stream and passes its events through the proposed stream-based processing workflow. Since all events are independent of each other, it is not necessary to rearrange events to different nodes at any point in the workflow. At the end of the workflow, each node performs the persistence of its events. As an extension of our experiment from Section 8.2.4, we are now normalizing and persisting with a varying number of nodes and node configurations. Figure 8.10 visualizes the results.

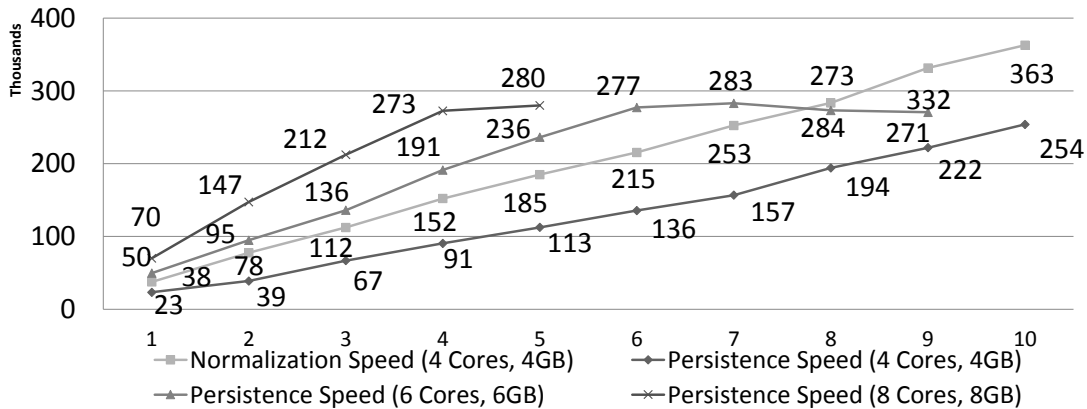


Figure 8.10: Persistence throughput with multiple nodes

As a comparison, we have also included the graph for the throughput of the normalization from Figure 8.9. For the configuration with 4 cores, we see that the throughput is increasing almost linearly with each additional node. The same can be observed for the nodes with a higher number of cores, but only to a certain number of nodes. Surprisingly, there seems to be a hard boundary at around 280 k evts./s that cannot be passed, regardless of available processing power. We have three possible explanations for this behavior.

1. There could be an I/O-bottleneck at the VM host we are using to deploy all the nodes. Even if enough processing is available, the handling of all the network connections results in too many I/O interrupts.
2. Another reason could be a limited network. In the normalization setup, we have observed throughputs of around 180 MB/s between the master node and the processing nodes. While this speed can be handled internally in the hypervisor, the persistence actually requires having this bandwidth on an outbound connection to the HANA server. However, our VM host only has a 1 Gb/s connection to this server. This strictly limits our database traffic to around 125 MB/s.
3. Although HANA is considered as an in-memory database, the data that is written to it must be backed up on the hard drive sooner or later. HANA has a

mechanism that performs these backups on a regular basis automatically. If the hard drives are not fast enough, then the writing to it could be the limiting factor for an ongoing persistence into the main storage with high throughput.

In an additional experiment, we have tried to address the points of the I/O-bottleneck at the nodes and network bandwidth. We have deployed the nodes on multiple physical machines that have a 10 Gb/s connection to our HANA server. Nevertheless, the throughput could not be raised above the boundary of 280 Gb/s. The possible limitations of the HANA server hardware could not be addressed, which makes it difficult to confirm that the last point applies.

One more observation that can be made from the above graphs is that larger nodes can use their resources more efficiently, i.e., the throughput per core becomes higher. This is expected, as each node needs a certain amount of processing resources to serve the operating system and some fundamental services. In the end, only the remaining resources can actually be used for processing. Therefore, if possible, larger nodes should be employed for the distribution.

8.4 Conclusion

In this chapter, we have focused on the scaling of the stream-based event processing workflow. Our goal was to achieve a throughput that is sufficient for monitoring large enterprises and that is far beyond Gartner's recommendation for a very large SIEM deployment, i.e., 25 k evts./s, as well as event throughputs of existing SIEM solutions, i.e., ≈ 50 k evts./s.

Our approach has focused on the horizontal and vertical scaling of the workflow steps by distributing the workload to multiple threads and processing nodes. Regarding vertical scaling, the main bottleneck is the distribution of the work between the threads. We could achieve the best throughput by using the disruptor pattern as a distributor. On a single machine with 16 cores, we could reach a normalization throughput of 171 k evts./s. As for the horizontal scaling, we have expanded our approach to also run on multiple hosts in parallel with the help of an efficient message exchange mechanism. This exchange mechanism in conjunction with vertical scaling enables a normalization throughput of 363 k evts./s on 10 nodes with commodity hardware. During our experiments, we could also observe that the throughput grows linearly with the number of threads for the linear as well as the horizontal approach. This means that even higher throughputs can be reached if more hardware is provided.

In the last step, we have focused on sustaining the normalization speed also for persistence, i.e., the outlet of the workflow. We have selected the HANA database as our primary storage system and have implemented various optimization strategies that are known from the SAP Community. In addition to that, we have integrated a

CHAPTER 8. SCALING EVENT STREAM PROCESSING

concept of minimal statements into the persistence module that could finally lead us to a peak throughput of around 280 k evts./s on a distributed setup on 7 nodes with commodity hardware. Although this throughput is smaller than with normalization only, it is still around 10 times higher than what is required by Gartner for a very large SIEM deployment.

All in all, the achieved throughput allows our prototypical REAMS to handle the sustained event load of large enterprises. In order to handle peak loads of even extremely large enterprises, such as a throughput of more than 1 M evts./s, multiple instances of REAMS can be deployed in parallel and more powerful processing nodes could be added. Furthermore, a proper buffering of events in a message queue can help in coping with extreme event peaks. As the events are now persisted in a normalized form, we are going over to the analysis phase of the processing workflow in the next chapter.

Chapter 9

Advanced Attack Analysis on Normalized Events

Related Publications

- *David Jaeger, Martin Ussath, Feng Cheng, and Christoph Meinel. “Multi-Step Attack Pattern Detection on Normalized Event Logs”. In: Intl. Conference on Cyber Security and Cloud Computing. 2015 [33]*
- *Andrey Sapegin, Marian Gawron, David Jaeger, Feng Cheng, and Christoph Meinel. “High-Speed Security Analytics Powered by In-memory Machine Learning Engine”. In: Intl. Symposium on Parallel and Distributed Computing. 2015 [116]*
- *Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Passive Network Monitoring using REAMS”. in: Intl. Conference on Information Science and Applications. 2015 [32]*
- *Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. “Pushing the Limits of Cyber Threat Intelligence: Extending STIX to Support Complex Patterns”. In: Intl. Conference on Information Technology: New Generations. 2016 [117]*
- *Andrey Sapegin, David Jaeger, Feng Cheng, and Christoph Meinel. “Towards a System for Complex Analysis of Security Events in Large-Scale Networks”. In: Computers & Security. 2017 [118]*
- *Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. “Identifying Suspicious User Behavior with Neural Networks”. In: Intl. Conference on Cyber Security and Cloud Computing. 2017 [119]*

In traditional security solutions, such as IDSs and firewalls, threats and intrusions are detected by monitoring the activities on a single system. Especially in the time of an increasing complexity of attacks, this limitation to a single machine is not feasible anymore [120]. The detection of complex attacks requires a big picture of all activities happening in a network and is based on the correlation of such activities with other available threat information. A SIEM system is the first step in this direction, because it collects events and alerts from a multitude of security sensors, being spread across the network, into one view. As a next step, the collected events need to be transformed into a unified event format, so that events from different sources and of different types can be correlated with each other.

In our experience, many SIEM systems on the market are currently limited in this regard, because they either cannot collect the events from all available sensors fast enough or are not comprehensively transforming events into a unified format. As a consequence, they are also not able to correlate events on time and therefore cannot trace attacks that are involving multiple steps or are spanning across multiple nodes in the network. REAMS, as our prototypical SIEM implementation, has a focus on providing both capabilities by transforming events from various sources into the unified OLF in near real-time, even for very large event volumes. Its storage into the in-memory DBMS HANA, and optionally into HDFS, further allows complex queries against the normalized event information. Therefore, REAMS is a suitable foundation for an event correlation platform that can be utilized for attack and threat detection.

In contrast to the detection of malicious activities on a single system by monitoring its behavior, the detection of malicious activities from events is not well researched. For sure, event logs have always been used to detect and understand attacks on a system, but that was more of a manual task conducted by a security operator. The automatic detection and correlation of malicious activities from multiple event logs, however, is a rather new topic to this date. Theoretically, the detection of attacks on event logs is not substantially different to the detection of attacks from system behavior as known from IDSs, since events are an abstraction of system behavior. Therefore, the detection approaches of an IDS apply to event logs, too.

In the following of this chapter, we take a closer look at analysis approaches of log data. At the beginning, the role of CTI is reviewed, because it acts as an intelligence source for all the detection approaches. After this, we are going over to the simple correlation of events and then continue with signatures as an implementation of misuse detection. Finally, we also shortly cover machine learning as an alternative detection approach.

9.1 The Role of Cyber Threat Intelligence (CTI)

An analysis of cyberattacks reveals that attackers are often reusing the same tactics, techniques & procedures (TTP), tools and infrastructures (attack assets) for various of their attacks [9, 10, 117]. On the one hand, unsophisticated attackers are reusing well-known public attack tools and weaknesses over a long time and are performing attacks in well-known schemes. On the other side, sophisticated hackers only have limited resources and do not want to spend the effort in creating new tools and setting up new infrastructures for each of their attack campaigns. They act by the thinking that as long as the old attack tools, techniques, and infrastructures work and were not detected by the victim, there is no need for them to change anything in future attacks. The reuse of attack assets is a weak spot in favor of the defender.

A defender that has knowledge and information of potential cyber threats, also referred to as CTI, is in an advantageous position, because he can detect and ward off such threats much earlier. Therefore, the sharing of intelligence and experience between affected parties, such as enterprises, institutions, security researchers or previous attack targets, is crucial for working cyber defense. The most common form of exchange are indicators of compromise (IOCs), which are characteristics observable in the monitored environment and point out to a potential threat. Today, there is an increasing number of security solutions, such as firewalls, SIEMs, and IDSs, that incorporate IOCs into their security analysis [80, 120]. According to a survey by the SANS Institute in 2015 [80], around 68% of the companies were using CTI as part of their security operations. Around 55% are even integrating CTI into their SIEM system or intrusion monitoring platform. Thus, the provision of CTI and particularly IOCs to our SIEM should not be a problem.

The automated sharing of CTI is a rather new approach to cybersecurity that just emerged from around 2010 [121]. Accordingly, many new types of sources and exchange formats for CTI have popped up over the recent years. Companies and institutions have created threat feeds that are providing intelligence easily consumable for security solutions. Among the most popular exchange formats for CTI are STIX, OpenIOC and Open Threat Exchange (OTX) [122]. In addition to that, plain listings of indicators in the form of text or CSV files are utilized. During our investigation of available data sources, we have identified three common groups of CTI with their typical exchange formats.

Blacklists / Reputation Lists This is the simplest and also the most common form of threat indicator. An indicator consists of a single observable characteristic, such as an IP address, domain name or file hash [120]. In a blacklist, the indicators are considered to be malicious, whereas in a reputation list each indicator has an additional tag about its level of maliciousness. Typically, blacklists and reputations lists are specifying hosts, services or files with observed negative

CHAPTER 9. ADVANCED ATTACK ANALYSIS ON NORMALIZED EVENTS

behavior and are created automatically from security companies through mechanisms like honeypots or malware sandboxing. However, since the information is automatically captured, blacklists/reputation lists often have questionable quality and frequently contain false positives. For the sharing of blacklists, vendors are providing the indicators in a simple text file, CSV file or enclosed in a standardized sharing format, such as STIX or OpenIOC.

Application Signatures These indicators are more detailed and are used as signatures for IDSs. As there are different types of IDSs, there are also different types of indicators in this group. For example, NIDS-based signatures are describing observable network behavior and HIDS-based signatures are describing the characteristics of malicious files. The formats in which they are represented are usually customized to the domain where they are employed. Among the most popular formats are Snort rules for NIDS signatures [109, 110] and YARA rules for anti-virus systems. Since signature rules are rather complex and require an understanding of the malicious behavior to be detected, they are written manually by security professionals and are then exchanged in formats like STIX or are combined in a large text file.

Threat Reports This category of indicators originates from concrete security incidents that were described in a threat report of a security vendor. Such threat reports are typically describing a more complex attack, so that the indicators are related to the stages of a cyber kill chain [9, 94]. The quality of indicators from threat reports is typically very high, as they have been investigated by a security professional and were written manually. In comparison to blacklists, the indicators of reports are more detailed and refer to a wide range of observable data fields, such as registry keys, existing files and opened ports. As a representation of this information, standardized formats like STIX, OpenIOC or OTX are used.

As can be seen from the above sources for CTI, the majority of threat intelligence is rather simple and quality information is scarce. In fact, the current intelligence focuses mainly on a single observable characteristic of a single event. The specification of sophisticated multi-step attack techniques, as known from APTs, and intelligence on the correlation of information is not covered and its representation in existing exchange formats is not supported [117]. However, according to the *CTI Pyramid of Pain* [123], the more complex and sophisticated an attack characteristic is, the harder it is to alter for the attacker. We see two reasons why such characteristics are currently not supported for detection. On the one hand, it is difficult to identify and express complex characteristics and, on the other hand, the currently existing SIEMs are not powerful enough to apply such intelligence on many events.

Although the currently available threat intelligence is improvable, it is still a valuable source for SIEM-based analytics. Blacklists and reputation lists create the foundation for signature-based detection on single events. Application signatures are harder to integrate, because they have custom formats and are close to observable characteristics of an operating system. Information in such detail is usually not available in events, so that only a smaller fraction of the application signatures can be utilized in a SIEM. Intelligence from threat reports covers the most sophisticated attacks that are causing the highest damage. Already, some of the threat reports, such as from Mandiant, are describing attacks with a high level of detail, including relations between different attack steps. Since the indicators from reports are represented in common formats, such as STIX, they can be imported into a SIEM with little effort and checked against single or multiple related events, depending on the signature engine. However, due to the limitations of expressing complex attack patterns, these indicators are not suitable for the correlation of events, as it would be required for the detection of multi-step attacks.

Our goal is to integrate all of the above-mentioned sources of threat intelligence into REAMS. In addition, we propose event-based detection approaches that are going beyond the currently existing intelligence and thereby show that the extension of threat intelligence formats should be considered.

9.2 Simple Data Correlation

As the stream-based workflow phase is making the incoming events available in a structured form, the most obvious type of event analysis is to apply structured queries on top of the persisted data. A structured query supports simple operations on single events, like filtering and sorting, but also more complex operations that set multiple events into relation, like grouping and joining. An implementation for queries is provided by the storage systems for the hot as well as cold storage over an SQL engine, so that no further code is required on the SIEM implementation itself.

We are distinguishing queries on the hot and cold storage. The primary purpose for queries on the hot storage, i.e., on the HANA database, is to get an overview of the currently ongoing activities in the monitored environment, whereas queries on the cold storage, i.e., on HDFS, are used to investigate past incidents.

9.2.1 Correlation Queries on the Database

A use case for queries on the databases are security dashboards that are provided in a user interface of the SIEM. The dashboard contains the most important security figures and is frequently updated. Examples of such figures are histograms of selected event types, selections of particularly critical events and the summation or counting of

CHAPTER 9. ADVANCED ATTACK ANALYSIS ON NORMALIZED EVENTS

event values. In addition to dashboards, the queries can check whether preset security thresholds, such as for the number of login attempts, is exceeded. Listing 9.1 gives an example of an SQL query on the event table that could be used for a dashboard. It sums up all the transmitted Hypertext Transfer Protocol (HTTP) data from GET requests for each combination of client and server. A security operator could use the results of this query to identify site dumps or the breach of data.

Listing 9.1: SQL Query for listing IP addresses with high HTTP traffic

```
SELECT PRODUCER_HOST, NET_SRC_IPV4, SUM(APPLICATION_LEN) AS PROD_SUM
FROM EVENT
WHERE HTTP_METHOD = 'GET' AND TAG_PRODUCER_TYPE = 'web_log'
GROUP BY PRODUCER_HOST, NET_SRC_IPV4
ORDER BY PROD_SUM DESC
```

9.2.2 Correlation Queries on HDFS with Spark

The cold storage can serve almost the same queries as the hot storage but is slower and does not cover the most recent events. If the runtime of a query is not a critical factor and the results are not immediately needed, as it would be in a dashboard, then database queries can be adopted and run over all live and historical event logs. Due to the fact that our proposed cold storage is a file system, i.e., HDFS, the queries cannot be run directly on the cold storage. Nevertheless, common distribution frameworks, such as Spark, can execute SQL-like queries on CSV or Parquet files that reside on HDFS. Although the event data resides on a disk, the frameworks can still reach high processing throughputs as the loading of the file and its processing is distributed to many processing nodes. Listing 9.2 shows how the SQL query from Listing 9.1 could be implemented in a Spark environment.

Listing 9.2: Spark query for listing IP addresses with high HTTP traffic

```
Dataset<Row> events = sparkContext.read.parquet("2017-10_events.parquet");
events = events
    .where(
        col("HTTP_METHOD").equalTo("GET"),
        col("TAG_PRODUCER_TYPE").equalTo("web_log")
    )
    .groupBy("PRODUCER_HOST", "NET_SRC_IPV4")
    .agg(sum(col("APPLICATION_LEN")).as("PROD_SUM"))
    .orderBy(desc("PROD_SUM"));
events.show();
```

At first, the Parquet file with the OLF events is loaded into a so-called Spark DataFrame. Then the query is applied to the DataFrame, which in turn converts the query parts to distributed tasks on the processing nodes. Since Spark is based on the MapReduce processing model, it can execute SQL queries and custom code in a distributed manner. As a result of this functionality, even complex analysis algorithms like signature- and anomaly-based detection could be executed on the cold storage.

9.2.3 Evaluation

To test the usability of the simple correlation approach, we have normalized ≈ 1.1 billion log events from a small IT infrastructure and have persisted these into a HANA database table with the OLF data schema. On the table, we have executed various queries that are typical for infrastructure monitoring, such as overviews of monitored systems, service clients and existing applications. The majority of these queries can run in a few seconds or even less. In addition to these infrastructure queries, we also applied queries for security investigation and could identify various attack attempts. At first, we have used the query from Listing 9.1 to identify unusually high network traffic. In the end, we could identify a small number of machines that downloaded multiple gigabytes of data from one of our servers to create a site dump. As a second test, we have searched for simple SQL injection and directory traversal attempts. This again delivered multiple thousand events.

9.3 Single-Step Signature Detection

An analysis approach known from IDSs is to detect common attack patterns with previously defined attack signatures. In the context of a SIEM, a signature would specify attack patterns observable in gathered events. A *single-step signature* is the simplest form of signature that only applies to a single log event. Thus, it can only cover one isolated attack step.

To a certain extent, single-step signatures are supported in existing SIEM solutions. However, there are limitations on how these solutions make use of the signatures provided from CTI, as outlined in Section 9.1. Our focus is to provide single-step signature matching capabilities on top of fully normalized OLF events and integrate signatures from the three presented sources of CTI into REAMS. In particular, we want to support as many application signatures from *Snort* and *mod-security* as possible, because they are two of the most common IDSs in the open source community and provide a large repository of signatures. If these signatures could be integrated, then the attacks previously only detectable in network traffic can also be detected in live and historical event logs.

The basis for our signature matching approach is a unified signature model that supports the expressiveness for the majority of available CTI signatures and applies to our OLF model. The existing signatures will be normalized and mapped to this model and are then applied to the incoming event stream. The next subsections describe the signature model, the normalization, and their application on events in more detail.

9.3.1 Our Signature Model

Existing Models In order to find a unified signature model, we have analyzed the features for the most predominant CTI formats, i.e., STIX/CyboX, OpenIOC, Snort signatures, and plain CSV. In STIX and OpenIOC, the signatures are referred to as indicators, where each indicator consists of a logical expression of multiple observables/indicator items, as shown in the example in Listing 9.3. One observable is expressed as a predicate and mainly consists of three components: the name of the observable property, a constant content value and an operator that specifies how the content value should be matched against the observable property.

Listing 9.3: Snippet of OpenIOC for APT28 [124]

```
<Indicator operator="OR">
  <IndicatorItem condition="contains" preserve-case="false" negate="false">
    <Context document="DnsEntryItem" search="DnsEntryItem/Host" type="mir"/>
    <Content type="string">nato-news.com</Content>
  </IndicatorItem>
  <IndicatorItem condition="contains" preserve-case="false" negate="false">
    <Context document="DnsEntryItem" search="DnsEntryItem/Host" type="mir"/>
    <Content type="string">ausameetings.com</Content>
  </IndicatorItem>
  ...
</Indicator>
```

Although Snort rules have a different structure for indicators, they also follow the idea that an indicator consists of multiple observables that have to match certain values. Similarly to the previous two formats, multiple observables, in the form of predicates, are joined to a logical expression of ANDs. In addition, there are modifier functions to be applied during matching, such as case insensitivity or matching offsets and lengths. Listing 9.4 has an example of a Snort signature that has multiple observables.

Listing 9.4: Trimmed Snort rule for detecting SQL injection in HTTP requests [109]

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"SQL union select -
possible sql injection attempt - GET parameter"; flow:to_server,established;
content:"union"; fast_pattern:only; http_uri; content:"select"; nocase; http_uri
; pcre:"/union\s+(all\s+)?select\s+/Ui"; metadata:policy max-detect-ips drop,
policy security-ips drop, service http; classtype:misc-attack; sid:13990; rev
:24;)
```

The information in CSV or text files is much simpler than that of the previous formats, because there are either lists or tables of observables without conditions, operators, or modifiers. If there are multiple observables in one record, then all of them must be present in the searched event.

Constructing a Unified Model As a result of our examination, we propose a signature model that has two main characteristics.

1. A signature consists of observables that are organized in a tree of nested logical operators. In other words, the signature is a logical expression of observables.
2. An observable consists of the *name* of the observable property, its *value*, a *predicate function (operator)*, and a *modifier* that specifies how the observable property is modified before passing it to the predicate function.

Both characteristics can be found entirely or in part in the examined signature formats, so that the proposed model is capable of bearing the information contained in them. To understand how such a signature would look like, we have converted the first part of the OpenIOC signature into our model, as shown in Figure 9.1. As an illustration of the logical concatenation of observables, we have additionally added the IP for `nato-news.com` and the default port 80 as observables to the signature.

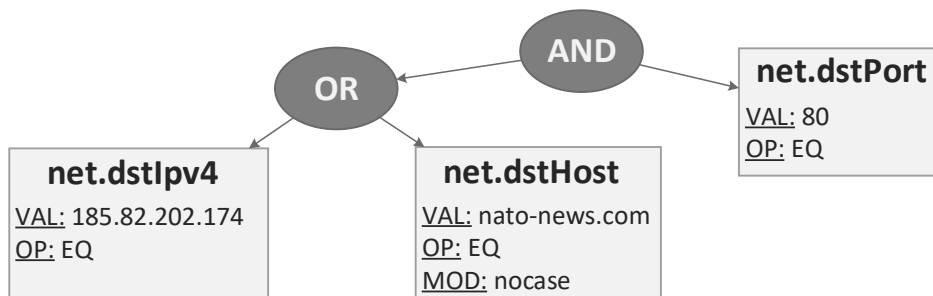


Figure 9.1: Example of a signature in our model for one of the indicators of APT28

The oval shapes are representing the logical operators that connect the observables. The rectangular boxes are representing the observables. They are referring to data fields, such as `net.dstIpv4` from the OLF model.

9.3.2 Normalization

Our signature detection approach relies on the fact that all signatures to be checked are available in the same unified signature model. This means that all the signatures retrieved from CTI need to be normalized to our model in order to make use of them. As the majority of signatures is represented in the few formats we have introduced, the normalization only has to handle a small number of different cases.

In order to normalize a format, we first have to understand the signature's meaning and structure. Once the structure is known, relevant indicator information is extracted and transformed into our signature model. In the last step, the fields that are referenced in observables have to be mapped to OLF fields. For OpenIOC, STIX and CSV/text files, the transformation of the structure is rather straightforward, because

it is similar to our model. For application signatures like Snort, this transformation is more challenging and needs to be described in more detail.

Transforming Snort Signatures to Our Model Snort rules are well-structured, so their parsing is least challenging. Each rule starts with information about the connection, the so-called `IPInfo`, which specifies the source and destination IP address and ports of the packets to be monitored. In parentheses, after the `IPInfo`, further properties of the rule are specified in key/value-pairs. These properties are categorized into generic rule information and rule options. The rule information contains meta-information about the rule, such as the alert message, the rule ID, its revision and a classification of the detected threat. The rule options contain the actual matching information for the rule, such as the fields and contents to match [125].

```
alert tcp $EXTERNAL_NET any -> $HOME_NET $HTTP_PORTS (msg:"SQL union select -
possible sql injection attempt - GET parameter"; flow:to_server,established;
content:"union"; fast_pattern:only; http_uri;|content:"select"; nocase; http_uri;
pcre:"/union\s+(all\s+)?select\s+/Ui"; metadata:policy max-detect-ips drop, policy
security-ips drop, service http; classtype:misc-attack; sid:13990; rev:24;)
```

Figure 9.2: Relevant indicator information in a Snort rule (dark gray - observables, light gray - metadata)

The illustration in Figure 9.2 again shows the same rule from Listing 9.4, but with the snippets marked that are relevant for the creation of a new signature according to our model. The parts marked with a dark gray color are used to build the rule structure. In particular, the first `IPInfo` part is translated to observables regarding the source and destination IP and port of our rule. In this concrete case, only the destination HTTP port deviates from the default values, so that the rule only has to check for port 80 or 8080. The flow information at the beginning of the second dark gray box distinguishes whether the signature applies to a server and consequently whether the destination in the `IPInfo` matches with the destination fields in OLF, which are `dstIPv4/6` and `dstPort`. So far, we have extracted the most important network parameters from the first part of the signature. The remaining signature parts in the dark gray boxes are providing additional observables as rule options. Usually, multiple rule options together make up one observable. There is a content or regex followed by an optional modifier and a field specification. In the figure, we have framed the options belonging to the same observable with a black border. All observables are referring to the HTTP Uniform Resource Identifier (URI) field. The first two use the keyword `http_uri` and the `pcre` has the flag `U` that indicates to search in the URI. After the rule options, there is even another field about the used service in the metadata that can be incorporated into the signature as an observable.

Snort fields	OLF fields
IPInfo Source IP	net.srcIpv4/6
IPInfo Source Port	net.srcPort
IPInfo Destination IP	net.dstIpv4/6
IPInfo Destination Port	net.dstPort
content	msg
http_header / http_raw_header / H/D flag	app.cmd
http_method / M flag	app.http.method
http_cookie / http_raw_cookie / C/K flag	app.http.cookie
http_uri / http_raw_uri / U/I flag	app.http.url
http_stat_code / S flag	app.http.status
http_stat_msg / Y flag	app.cmd
service	app.proto

Table 9.1: Field mappings from Snort to OLF

The light gray text is used as meta-information that is not directly integrated into the signature. As an example, the `msg` field could be taken as a subject for an alert that is created based on the matched signature. Also from the `classtype`, we can derive a CWE identifier for the alert event.

Mapping Data Fields to OLF fields After the observables have been extracted from the original Snort signature, they are still referring to data fields of Snort. To make them compatible with REAMS, they need to be mapped to corresponding OLF fields. By default, Snort works on the level of network packets, so that we are mainly limited to OLF's fields of the network layer. In addition to that, Snort has support for checking fields of a few application protocols, such as HTTP, SIP, SSL, GTP and DCE/RPC [125]. OLF in its current form only has support for HTTP, so that the signatures with an HTTP portion are most relevant for normalization.

In many cases, the fields in Snort have a direct counterpart in OLF. This means that there is no conversion or mapping of values required. Table 9.1 presents the mappings for most fields in Snort to OLF. The first four rows cover the mapping of the `IPInfo` fields to the IP addresses and ports in OLF. Content fields without a corresponding field specifier are not directly mappable to an OLF field, because they refer to packet content that is typically not available in an event. As the best try, such `content` could be mapped to OLF's message field. After the `content`, we list the HTTP field specifiers. In general, Snort uses multiple field specifiers for the same HTTP information. First, there is a distinction between raw and normalized content and secondly there is a difference whether the content is matched using the `content` field or a regex. For the regex, the field is indicated by a flag on the regex.

This results in the situation that, sometimes, four fields in Snort are mapped to only one field in OLF. The mapping itself is rather straightforward, since OLF refers to the same HTTP information. Only the HTTP header has a more generic mapping to OLF's `cmd` field, which holds the command issued over a protocol. In the last row of the table is an additional mapping for the service field in Snort. Also here, OLF has a corresponding field for the used protocol.

Combining the transformation of the rule from Figure 9.2 in the previous paragraph with the mapping of the Snort fields to OLF according to the above mapping table, we come to the rule in Figure 9.3.

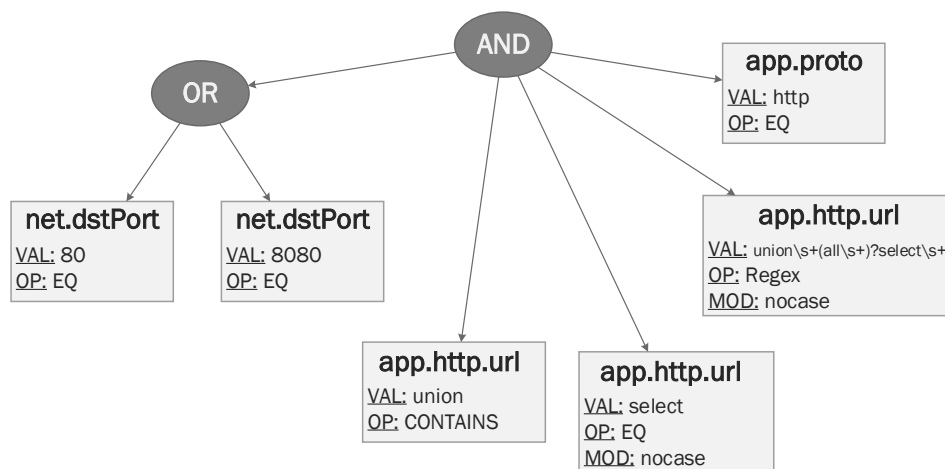


Figure 9.3: Our signature model for the Snort signature

Applying the Normalization on a Complete Ruleset After we have shown the conversion of a single rule from Snort to our signature model, we also want to apply the same normalization to as many rules as possible from the three largest available public Snort rulesets, which are the *community* and *registered* ruleset of Snort as well as the *Emerging Threats* ruleset from ProofPoint. Together, these rulesets have 75 291 different rules, of which 9973 are targeted to the HTTP protocol. Of all these rules, we were able to map 35 154 rules to our signature model and finally identified 5321 rules to be relevant for the matching within events. In the end, the majority is specific to HTTP.

Although the transformation of Snort rules to our model required us to create a customized normalizer, we can say that the effort was justified. Without writing any signatures ourselves, we can now identify many common threats and can also easily integrate the signatures of future upcoming threats.

9.3.3 Matching

The normalization procedure has ensured that all the indicators from CTI, which estimate multiple thousands, are available in our signature model. In the next step, these signatures need to be applied to the events. Remembering the processing workflow, we see two different locations where they can be applied. They can be checked during pre-analysis in the stream-based preparation phase or the batch-based analysis phase. The former allows identifying attacks in near real-time, whereas the latter enables detection in historical data.

Regardless of the location, we want to employ the same matching procedure. In the simplest procedure, the logical expression of each signature would be matched against each incoming event. Unfortunately, this would result in poor performance as thousands of signatures have to be checked. In previous research, people have dealt with the matching of many rules against a set of elements. In particular, Meier has summarized existing techniques and proposed two optimization strategies for single-step signatures [126]. Together with these two, we employ three optimizations that can dramatically increase the matching speed.

1. There are many signatures that only have a single observable with an equality check to a constant value, such as in blacklists. Instead of checking each of them individually, they can all be checked simultaneously by joining their content values into a hash list or search tree. As a result, many signatures can be checked in almost constant time.
2. There are many signatures with more complex structures that share the same sub-expressions for their observables. For example, there could be HTTP-based signatures that all check for port 80 or 8080. We can save evaluation time by identifying such shared expressions and evaluating them only once.
3. The evaluation of a logical expression, which has multiple predicates connected by logical operators, can be improved over logical shortcuts. A logical shortcut describes the fact that it is not always necessary to get the truth value of each sub-expression of a logical expression to derive the final truth of the whole expression. For example, if two predicates are connected with `AND` into an expression and the first predicate is known to be false, then the whole expression becomes false, regardless of the truth value of the second predicate. Therefore, in cases where the first predicate of an `AND` expression becomes false, only a single predicate has to be evaluated to derive the expression's outcome.

A further improvement of this principle is achieved by ordering sub-expressions/predicates by their selectivity, i.e., the chance of prematurely deciding the result of the overall expression. If we see that a sub-expression often results in a logical shortcut, then it makes sense to put this sub-expression at the very

beginning of the logical expression. This reordering is conducted dynamically as new events are matched against the signature.

The three optimizations enable us to apply single-step signatures in near real-time. In the streaming phase, we deploy multiple matchers in parallel as consumers of the disruptor. This is possible, as the matching runs independently for each event. Also in the batching phase, multiple matchers can run in parallel on multiple threads or even on multiple nodes. In order to keep track of generated alerts, the signature matcher represents alerts as special events that are fed back into the event processing workflow. Of course, these events do not need to be normalized again. An advantage of the representation as an event is that all analysis algorithms can build on top of these alerts. In other words, it would be possible to generate alerts based on another observed alert.

9.3.4 Evaluation

As an evaluation, we have applied all the HTTP-based signatures from Snort and Emerging Threats on a set of ≈ 10 million normalized OLF events. As an outcome, we could identify 1170 suspicious events that were related to 11 different signatures. The majority of these events were either targeted to the ShellShock vulnerability (CVE-2014-6271 [127]) (448 events) or accesses to various admin panels.

9.4 Multi-Step Signature Detection

We have presented single-step signatures as a method of revealing attack patterns in a single event. With them, we have realized a functionality that is known from traditional IDS systems. Due to the limitation to a single event, single-step signatures are not able to verify the success of an attack and cannot track attacks spanning over multiple hosts. This leads to the situation that some attack attempts are alerted as successful attacks (false positives) and that some complex attack campaigns are overlooked because each of the campaign steps was considered unsuspecting (false negatives). Therefore, as a next step, we are correlating multiple events. We call the description for such a correlation of events a multi-step signature, because it describes an attack that consists of multiple attack steps. As the name implies, a single step in a multi-step signature is comparable to the previously introduced single-step signature.

In the following, we present a signature model for multi-step signatures and explain how such signatures can be matched in REAMS.

9.4.1 Model Based on EDL

The research community has worked on a number of models and languages for the representation of complex signatures. Meier [128] has created an overview of these languages and proposes a categorization by their use case. He proposes a distinction between attack, exploit, event, detection, response, report, and correlation languages. His detection languages come closest to our signature model, as they describe the procedure and analysis steps required for the detection of attack patterns. Among others, he mentions the languages P-BEST [129], STATL [130] and LAMBDA [131] as representatives of this group. Due to the fact that these languages have mostly been developed as part of an IDS, they are also highly adapted to the features and the detection algorithms of these systems. This results in languages that stipulate the detection procedure and have a limited expressiveness. As an alternative, Meier et al. propose their abstract detection languages SHEDEL [132] and Event Description Language (EDL) [133] that combine the features of previous languages with a higher expressiveness and a lower dependency on a concrete IDS. We have decided to also use EDL as a foundation for our signature model, because it is the most advanced language and already comes with an implementation. For an understanding of EDL, we present its basic concepts below.

Concepts of EDL As the main idea, signatures are seen as a colored Petri net. The places of the net represent the current system or signature state. The occurrence of an event triggers the transition between states. The tokens that traverse the states and transitions of the net can be considered as instances of the signature. The color of a token stands for the variable state, consisting of *token features*, of a signature instance. Figure 9.4 shows an example of a signature net as it seen in EDL.

The signature example detects a brute-force attack and subsequent access to the `/etc/password` credential file. `Start` stands for the start state at which no malicious behavior was identified, yet. The final state is `Suspicious Access`. It indicates that the brute-force attack and the access to the credential file were successful. The remaining boxes stand for intermediate states of this attack, such as that two failed logins were seen. The transition between states is triggered by *account login* or *file access* events, as they are produced by Linux' auditing functionality. The conditions for the corresponding triggering event are expressed in the tabular boxes next to the transition boxes. There are three different types of entries in these tabular boxes. *Intra-event conditions* only refer to constant values and fields of the triggering event. In fact, they are comparable to the observables of a single-step signature. *Inter-event conditions* are referring to a field of the currently triggering event and a token feature, which was set by a mapping in a previous transition. The *mappings* are assigning values from the current context, i.e., from the triggering event or the current token state, to the state of the token passed to the destination place.

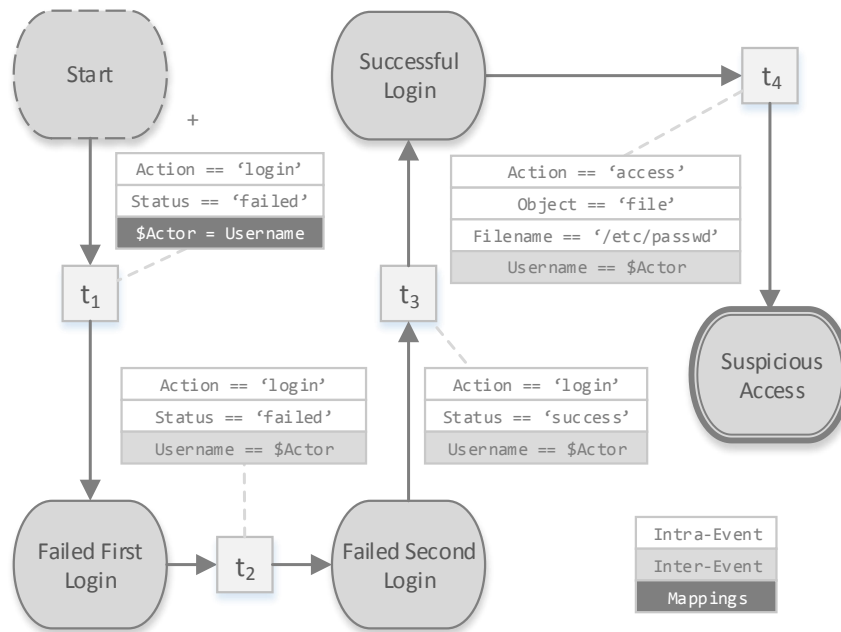


Figure 9.4: Example of a simple signature for a brute-force attack with `/etc/passwd` access in the EDL

Listing 9.5: Simple EDL signature for a Windows login

```

EVENT UserLogin
{
  PLACES
  Init {
    TYPE INITIAL
  }

  LoggedIn {
    TYPE EXIT
    FEATURES
    STRING username
  }

  TRANSITIONS
  Init(+) LoggedIn {
    TYPE NT_AUDIT_EVENT_4624
    CONDITIONS
    (True)
    MAPPINGS
    [LoggedIn].username=AccountName
    ACTIONS
    warnln("User " + AccountName + " successfully logged in")
  }
}

```

Listing 9.5 gives another example of an EDL signature, but this time in a textual form. It describes a successful login onto a Windows machine. The structure of the

event clearly separates the *places* and *transitions* of the signature net. The places section covers two places for the initial `Init` and the final `LoggedIn` state. Each place has the corresponding type specified with the `TYPE` keyword. In addition to the type, a place can also have multiple features for its token. In the transition section, there is a single transition that has a `(+)` directly after the `Init` state. This sign points out that this transition is *non-consuming*. In a regular consuming transition, the token would be moved from the source to the destination state, whereas in a non-consuming transition the token is just copied. This has the effect that there will always be a token in `Init` and there can be multiple instances of the signature at the same time. Inside the transition, there are four further specifications. The `CONDITIONS` and `MAPPINGS` are already known from the above signature net. The `TYPE` specification reveals what kind of event triggers the transition, so in our case Windows events of type 4624. The `ACTIONS` specification defines what should happen when the signature has matched. In our example signature, there would just be a text message that informs about the detected login.

A more detailed description of the concepts and language constructs of EDL can be found in Meier's book [126] or research papers [133, 134].

Extension of EDL for OLF In its current form, EDL can express a wide range of attack patterns, but is relying on the fact that these patterns are described with event types or formats that are known in advance. Consequently, EDL has a fixed set of supported event types and would be able to detect attacks that manifest in these events. Our primary goal is to make EDL compatible with OLF, so that events of any type can be processed with it. Since OLF is using nested objects, such as in `app.http.userAgent`, the field notations in intra- and inter-event conditions as well as mappings need to be extended to support the dot notation.

9.4.2 Signature Sources

As outlined in the section about CTI, threat indicators are kept simple due to the limited expressiveness of exchange formats. Only very few of those formats can incorporate more than one event into a pattern. The STIX format is one of those few, because it can represent relations between observables that are not belonging to the same event. Still, it is not possible to describe joint properties between these events, so that it is difficult to express event sequences as used in EDL. As a solution to this problem, Ussath et al. [117] have proposed an extension to STIX allowing the referencing of properties from other events.

Another exchange format that has support for simple multi-step patterns is Snort. Although it is mainly designed for single-step patterns, there are mechanisms for expressing the multiplicity of single events. For example, it can express a signature that

requires the occurrence of multiple failed logins in a given time-frame for detecting a login brute-force.

A completely different type of source for multi-step signatures are special correlation algorithms. They are extracting sequences of events that are connected to known threat indicators or are deviating from the normal system behavior. One such algorithm is proposed by Ussath et al. [135]. It is based on the idea that attackers are reusing the same infrastructure and tools within the same attack. The algorithm follows the propagation of observable malicious artifacts and is correlating these events to an attack sequence. As a result of its analysis, the algorithm creates an EDL signature that is able to detect attacks with the same characteristics.

Obviously, the choice of multi-step signatures as CTI is limited. The established exchange formats only provide rudimentary support for the description of such signatures. The only viable source are algorithms that generate signatures based on observed malicious behavior. In our opinion, the integration of EDL into a common exchange format like STIX could promote the use of multi-step signatures in CTI. As long this did not happen, manually created signatures are the main input for multi-step signature detection.

9.4.3 Matching and the SAM Implementation

To match multi-step signatures in the form of EDL against the incoming events, we need to create a dedicated module for our processing workflow. This module can be put into the pre-analysis step of the stream-based phase or the signature detection step of the batch-based phase. Fortunately, we do not need to implement EDL matching from the beginning, as Meier et al. have created a signature engine prototype for EDL, called SAM, together with his publications on the language. The original implementation was written in C++ (SAM), but there is also a newer version in Java (jSAM)¹. The newer Java version is most interesting, because our workflow implementation is written in Java, too.

As already mentioned, the EDL language itself is relying on predefined event types. Also, the jSAM implementation only comes with a few supported event formats, as Figure 9.5a illustrates. Solaris, IDMEF and CWS events do not seem to be up-to-date anymore and the NT Audits and Bro DNS events are not sufficient to express common attack patterns. For example, a traversal attack on an Apache Web server would remain undetectable if CLF is not supported. Also other popular event formats, like Syslog and a majority of other Windows events, are not supported.

One of the issues of SAM is its standalone nature, which requires that all event formats it should support have to be integrated into it. Furthermore, all generated alerts are kept within the system, meaning that they cannot be used for further anal-

¹jSAM Source Code - <http://sourceforge.net/projects/jsam-project/>

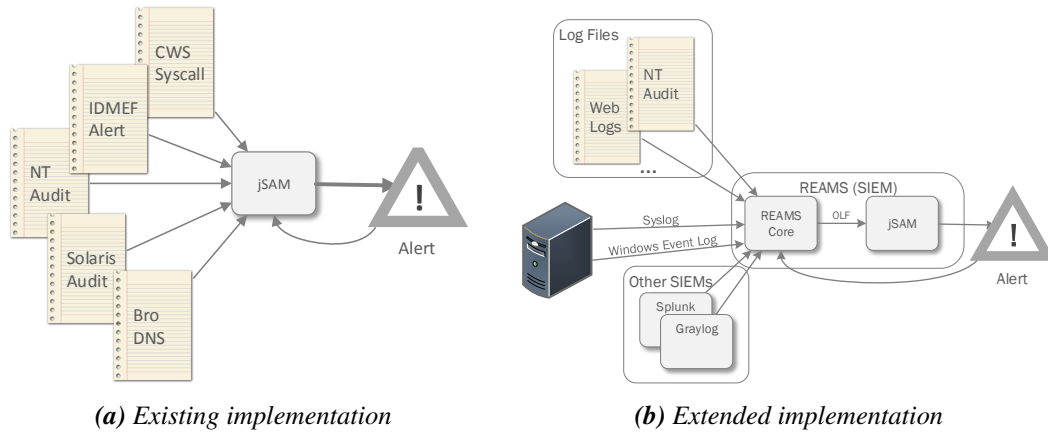


Figure 9.5: Comparison of existing jSAM implementation with extended version

Listing 9.6: Transition for failed login as OLF event

```

TRANSITIONS
FailedLogin(-) FailedLogin
{
  TYPE OLF
  CONDITIONS
    tag.action=="login", tag.status=="failure",
    [FailedLogin].username==[FailedLogin].username,
    [FailedLogin].sourceIP==[FailedLogin].sourceIP,
    [FailedLogin].target==[FailedLogin].target,
    [FailedLogin].app==[FailedLogin].app
  MAPPINGS
    [FailedLogin].count=[FailedLogin].count+1
  ACTIONS
    warnln("User "+[FailedLogin].username+" has another failed login on host: "
    +[FailedLogin].target)
}
    
```

ysis approaches. We propose to only integrate the core of jSAM and open up its interfaces to consume unified OLF events from REAMS and reintegrate produced alerts as OLF events into REAMS. Figure 9.5b demonstrates this new role of jSAM in our implementation. As a next step, we show how the proposed changes can be realized.

9.4.3.1 Supporting OLF in jSAM

All the supported event formats are described in a configuration file, called `AuditBaseTypes.xml`. As the simplest solution, we could add another section there for OLF and would have the required support. However, this means that the format once defined in REAMS needs to be copied over into this file, causing redundancies. As a more practical

Listing 9.7: Transition using report_alert function

```

TRANSITIONS
FailedLogin TenTimesFailedLogin
{
  TYPE SPONTANEOUS
  CONDITIONS
    [FailedLogin].count>=10
  ACTIONS
    report_alert (
      "LoginBruteForce",
      [FailedLogin].eventList,
      "Brute Force is ongoing on "+[FailedLogin].target,
      [FailedLogin].properties
    )
}

```

solution, we are retrieving the format description from REAMS and are integrating that directly into jSAM. In the original implementation, the formats are read from the configuration file by the so-called `BaseTypeReader` component. We have created an extension of this component, called `OLFEventTypeReader`, for the implementation of an alternative solution, which imports the OLF model of REAMS as a new model into jSAM.

In addition to the definition, each type also needs an `EventLoader` transforming incoming events into the event format of jSAM. We have created the `OLFEventReader` that receives OLF events from REAMS and transforms them for the use of the signature engine. Similarly to the persistence, this kind of event transformation benefits from the slot model of Section 7.3.1. Only set fields are read from the OLF event and are mapped to features of the resulting OLF object in jSAM.

As a result of our changes, we can now reference incoming OLF events in EDL signatures and detect patterns in the event stream. Listing 9.6 shows a transition for a failed login based on OLF. As a contrast to the transition from Listing 9.5, this transition is applicable to all kind of logins that manifest in logs, not just Windows-based logins.

9.4.3.2 Passing jSAM Alerts to REAMS

In jSAM, each triggered signature is represented as an event. This event can then be referenced in another signature, so that hierarchies of events and signatures can be built. Still, produced events are only considered internally to trigger transitions of other signatures, but are not exported out of jSAM. In addition to internal events, there are alerting functions in the transition actions that are executed whenever a transition is triggered. These alerting functions are used to send status messages to the system operator.

As a simple extension, we have created the `report_alert` function as a tran-

sition action. The function takes a message, a list of event IDs, and a list of event properties for the produced OLF event. In the end, the produced events are put back into the event stream of REAMS, where they can be used for further analysis, such as a single-step signature or query-based detection. Listing 9.7 shows the alerting of a failed login with `report_alert`.

As a more complete example, we have put the signature snippets from the previous Listings into the signature in Figure 9.6 for the detection of login-based brute-force attacks. The signature is solely based on OLF and internal `TimerEvents`. It is therefore applicable to all applications that indicate login attempts in log events.

Altogether, there are five places in the signature. The signature begins with the start place `init`. The first transition is leading to the `first_login` place, which indicates that a failed login was observed. Within the state, all important information about the login is stored, such as the target application and the user logging in. So far, the user is not yet suspicious, because a single failed login can happen to anyone. Only with the next failed login, the activity is considered as a possible ongoing attack and the signature changes to the `ongoing` state. Now, all consecutive login attempts are counted, regardless of whether they are successful or failed. Once a number of at least 10 failed logins is reached, the signature switches into the `exit` state. Together with the transition to this state, the incident is reported with the `report_alert` function and an alert is dispatched to REAMS. Based on the fact whether a successful login was observed before, the report can indicate the outcome of the brute-force attack. For all places of the signature, it is checked whether the last login happened more than 10 seconds ago. If this is the case, then the observed activities are not considered as a brute-force and the state is moved to the `escape` place.

9.4.3.3 Parallelization

Even though the matching of multi-step signatures is integrated as a module into the parallelized processing workflow, the matching itself is not easily parallelizable. The jSAM engine works with a signature net that keeps its state in the main memory. This state is maintained by one thread that performs operations on the net and its states. As the operations require access to many data structures, the concurrent access by multiple threads is not realizable without major performance faults. Therefore, the matching of a single multi-step signature cannot be performed with multiple threads in parallel. A strategy to distribute the matching of signatures is to distribute individual signatures to different threads or processing nodes. Each of those threads or nodes then gets a full copy of all incoming events to present to their signature nets.

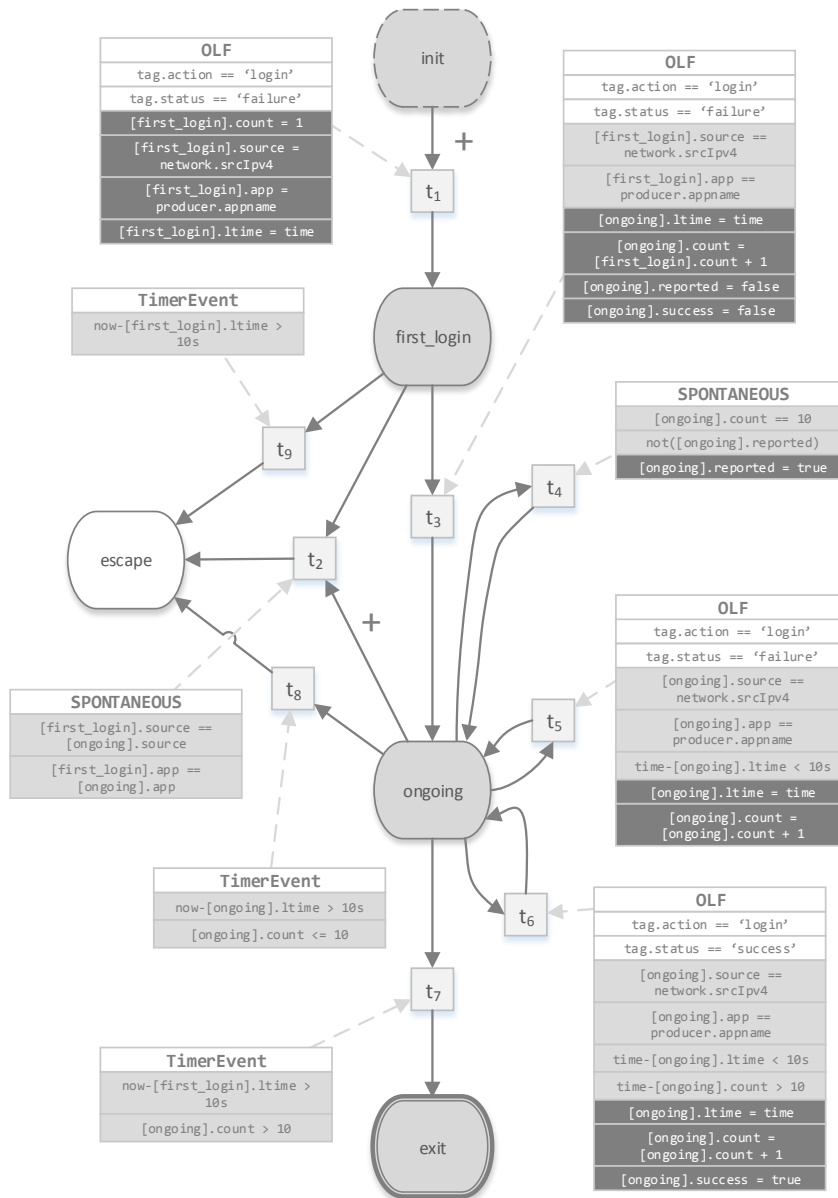


Figure 9.6: EDL signature graph for login brute-force attacks

9.4.4 Evaluation

As an evaluation, we have tested the above EDL signature against a lab environment with many different login-based services and our live-running IT infrastructure. In the lab environment, we have first installed a database, SSH, FTP, and web server.

Then we have attacked these services with the THC Hydra² brute-force tool. The signature engine was able to detect all attempts and could also distinguish successful from failed attempts. In our live-running IT infrastructure, we can regularly identify failed brute-force attacks against various SSH and web servers.

9.5 Machine Learning and Anomaly Detection

The previously presented approaches are mainly based on externally provided or self-created threat intelligence. This intelligence originates from the experience a security expert has gained by, e.g., investigating past incidents or observing known threats. Machine learning is another way of approaching the detection of attacks. Instead of relying on human expert knowledge, an algorithm gains its own experience by observing benign and malicious activities. With the obtained knowledge, the algorithm can later reason by itself whether an activity seems benign or not. On the one hand, such an algorithm can run independently of existing knowledge and can therefore also detect previously unknown attacks. On the other hand, the results of such algorithms are rather vague, meaning that they produce more false positives and negatives than misuse detection approaches. Still, machine learning algorithms are a viable complement to misuse detection approaches.

Machine learning algorithms are subdivided into *supervised* and *unsupervised* learning algorithms [136]. Supervised learning algorithms are deriving their knowledge from provided pairs of input and its desired output, which is also called labeled data. Unsupervised learning algorithms are deriving their knowledge solely from input data where the output is not known yet, which is also called unlabeled data. If we apply these concepts to security analysis, then a supervised algorithm would be provided with a set of features, such as the fields of an event, together with a label about the event's maliciousness. An unsupervised algorithm would be provided with a set of features that are associated with normal behavior. Unfortunately, due to the complexities of identifying attacks in huge amounts of security-related data, there are only very few datasets for labeled data. Furthermore, the few existing datasets are mostly based on set up scenarios where attacks have been simulated, such as the KDD Cup 1999 [137].

9.5.1 Implemented Algorithms

In our team, we have applied supervised and unsupervised machine learning algorithms on normalized OLF events. On the side of the supervised algorithms, Ussath et al. [119] have experimented with a feed-forward and recurrent neural network on

²THC-Hydra - <https://github.com/vanhauser-thc/thc-hydra>

a simulated login scenario. Neural networks have the advantage that they do not need special programming and usually have much higher accuracies than traditional supervised learning algorithms. Accordingly, the proposed algorithm could reach an accuracy of 97% in detecting suspicious login behavior in an enterprise network. Even though we have tested a supervised approach, our focus was more on unsupervised algorithms, as they can work on unlabeled data being available in REAMS. One of the algorithms, proposed by Sapegin et al., is an anomaly detection algorithm that is based on the Poisson distribution and has proven to work for the detection of anomalous login behavior [138]. Another algorithm, also proposed by Sapegin et al., runs k-means on chunked data and then applies a Support Vector Machine (SVM) algorithm on the ensemble of the k-means results. As an improvement, the algorithm automatically determines the optimal cluster number k and ranks the resulting clusters by their significance [118].

9.5.2 Execution Environment

There are four major ways in which the previously mentioned algorithms are run on the normalized events [116].

1. The basic algorithms, such as k-means, can be run directly within the HANA database engine. HANA provides this support over its Predictive Analysis Library.
2. Advanced algorithms can be run within the HANA database over its R integration.
3. Advanced algorithms can also be run on an external R module that reads the data either from the HANA database or any cold storage. The R module is either coming in the form of an R server or as a script in a distributed processing environment like Spark.
4. Neural networks are best executed on highly parallelized graphics processors and therefore require that the data is exported from the database or cold storage in advance. The execution is conducted on the server that has the appropriate hardware installed.

The execution within the database has a speed advantage, as the data does not have to be exported. Still, the external R servers are most flexible and can even distribute the processing tasks to multiple machines if solutions like Spark are employed. The most inflexible method is the execution on graphics cards, because the data has to be exported and there is only limited support for work distribution.

For a more detailed description of all the applied machine learning approaches and their runtimes in the REAMS environment, we refer to the work of Sapegin [116, 118, 138] and Ussath [119].

9.6 Conclusion

In this chapter, we have approached the topic of event analysis and its integration into REAMS. As a first step, we have introduced correlation queries as a simple and fast way of filtering, grouping, and joining event data. These queries can be used to create dashboards on the environment or to investigate incidents manually. As a second step, we have looked into the analysis approaches known from IDSs and have adopted them to normalized events. On the side of misuse detection, we have presented single- and multi-step signature detection. Single-step signatures are mostly based on CTI and are suitable for finding automated and unsophisticated attack activities in a quick and easy manner. Multi-step signatures are more applicable to sophisticated attacks that involve multiple stages. On the side of anomaly detection, we have introduced the role of machine learning and have referenced two different algorithms proposed by our team.

With the implementation of misuse and anomaly detection on top of event logs, we have combined the advantages of SIEMs and IDSs. We can follow activities from all monitored systems and can apply established analysis algorithms on them. While traditional IDS systems required that signatures be written for each application and event type individually, we can now have one signature for all applications from various domains due to a normalized event structure. By additionally integrating existing knowledge in form of CTI into the analysis process, the SIEM system is operational from the very beginning and does not require the provision of manually written signatures.

As an evaluation, we have also applied the proposed analysis approaches to the event data of our productive systems and that of our partners. In the end, we were able to identify numerous attack attempts with them, such as directory traversal, SQL injections, site dumps, and login brute-force.

Chapter 10

Using Identity Leak Data as Threat Intelligence

Related Publications

- *David Jaeger, Hendrik Graupner, Andrey Sapegin, Feng Cheng, and Christoph Meinel. “Gathering and Analyzing Identity Leaks for Security Awareness”. In: Intl. Conference on Passwords. 2014 [34]*
 - *David Jaeger, Chris Pelchen, Hendrik Graupner, Feng Cheng, and Christoph Meinel. “Analysis of Publicly Leaked Credentials and the Long Story of Password (Re-)use”. In: Intl. Conference on Passwords. 2016 [37]*
 - *David Jaeger, Hendrik Graupner, Chris Pelchen, Feng Cheng, and Christoph Meinel. “Fast Automated Processing and Evaluation of Identity Leaks”. In: Intl. Journal of Parallel Programming (2016) [36]*
 - *Hendrik Graupner, David Jaeger, Feng Cheng, and Christoph Meinel. “Automated Parsing and Interpretation of Identity Leaks”. In: Computing Frontiers. 2016 [35]*
-

Data breaches, also called data leaks, describe the break-in into a system or network and the subsequent stealing and possible distribution of sensitive data from it. They are affecting thousands of companies, services, and private users every year and are becoming a serious threat. The objective of these attacks is mostly the stealing of personally identifiable data, such as copies of customer databases, employee records,

CHAPTER 10. USING IDENTITY LEAK DATA AS THREAT INTELLIGENCE

or the phishing of credentials from individual users. The attackers use the stolen personal information to pretend another person's identity, perform social engineering or take over victims' accounts with their stolen credentials. The latter scenario is especially popular since many Internet users are reusing their passwords across multiple systems and services. As a consequence, the leakage of a user's credentials on one service can lead to the compromise of a multitude of the user's other service accounts.

For enterprises, the leakage of account credentials from their employees poses a significant threat, because the strong habit of reusing passwords could allow an attacker to gain access to the enterprise's internal network. According to a study by Verizon [73], around 40% of data breaches are successful due to the improper use of credentials, such as the reuse of password or a choice of weak passwords. On top of that, leaked personal information additionally simplifies social engineering of employees and can enable access to confidential data and sensitive systems.

In the following, we take a closer look at a significant subset of the problem, namely identity leaks that are published on the Internet for everyone to access. These public leaks are particularly relevant for two reasons. On the one hand, they put the victims at a higher risk because their easy-to-access data is misused by a larger number of even low-skilled attackers. On the other hand, they also give victims a chance to get to know that their data has been leaked. In fact, the awareness and knowledge about leaked services and credentials can give an enterprise as well as individual users an advantage over potential attackers. They can take precautions by resetting their passwords and closely monitoring activities of affected accounts. Unfortunately, at the moment, a large number of victims never get to know that their personal information or credentials have been leaked. This is because such leaks are mostly not reported in the mainstream media and the places where data is published are not always easy to find. However, an expert that has knowledge about the leak economy gets attention about such leaks early and can inform victims about new leaks in time. We propose to make publicly leaked identity data available as a kind of CTI that is also consumable in a SIEM like REAMS. An enterprise that subscribes to a corresponding intelligence feed could be informed as soon as one of their email addresses is affected.

In the next sections, we have a more in-depth look at the nature of public identity leaks and show how relevant identity information is extracted from it. Then, we describe how such information can be utilized for attack detection in the form of specialized threat intelligence.

10.1 Public Identity Leaks

10.1.1 Quantity

The relevance of data breaches is confirmed by a large number of leak incidents and their affected identities from the past years. A good impression of the volume and amount of identity leaks can be taken from the comprehensive list of public leaks by the leak monitoring service BreachAlarm [139]. Although they do not consider each public leak, they still try to list as many leaks as they can get their hands on. In Figure 10.1, we have created a chart that visualizes the number of leaked identities over each month of the past six years from as many as 21 321 individual leaks.

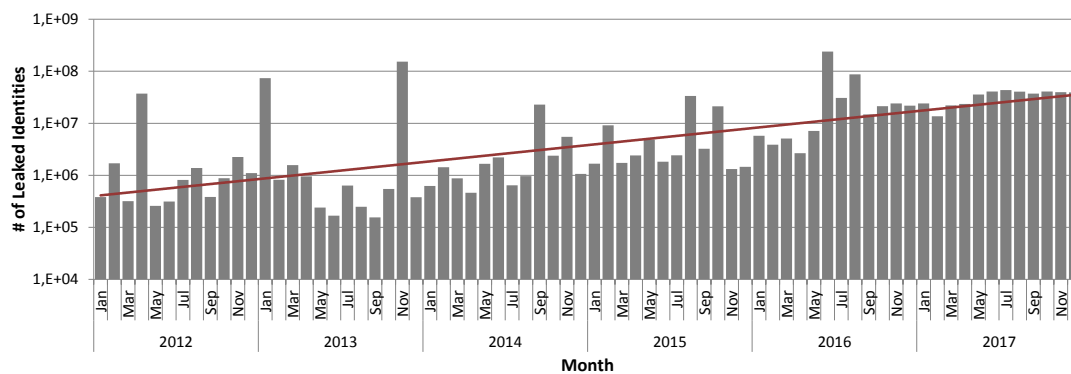


Figure 10.1: Number of leaked identities in the past six years with trend line (Source: BreachAlarm [139])

The graph clearly reveals the increase in monthly leaked identities in recent years. In fact, the growth is even close to exponential, because the scale of the chart is logarithmic, and the red trend line is linear in this chart. As an example, in 2013 the average volume of monthly leaks was less than 1 million identities, in late 2016, the average was already more than 20 million per month. Especially in 2016, there have been some impressively large leaks, such as from MySpace with around 360 million and LinkedIn with around 165 million affected identities. Yahoo has revealed an even larger leak at the end of 2016, which is estimated to affect one billion of their users [140]. Based on our investigations in the leak topic, we can say that already the largest 100 publicly available leaks are responsible for more than 2 billion leaked identities. Another site that monitors publicly accessible data breaches estimates the total number of affected identities to more than 4 billion identities at the beginning of 2018 [141]. Putting this into relation to the world population that uses the Internet, i.e., 4.1 billion people in December 2017¹, then, assuming that each user only has one digital identity, there is almost one leaked record for each Internet user.

¹Internet World Stats - <http://www.internetworldstats.com/stats.htm>

10.1.2 Data Distribution

There are three ways in which stolen identity data is distributed on the Internet by the original attacker, also referred to as *leaker*. The data is either shared within a small closed group, is sold, or published for everyone. Typically, data that is initially sold will eventually become public to everyone after some time. The motivation for the distribution of the data is manifold. Foremost, an attacker wants to prove his skills by publishing stolen data. He hopes to increase his reputation and recognition in the community. Then, some attackers mainly want to get a monetary benefit by either selling the data on forums or by misusing the stolen data to extract money from victims' financial services. Another group of attackers publishes the data to humiliate or damage the reputation of the attacked service or institution and its users.

For our consideration of identity leaks, we are focusing on the distribution by publication, because this data is easy to collect and potentially misused by many cybercriminals. The providers of published data can be distinguished by general-purpose and specialized leak providers. In the following, we list the most popular sites for leak distribution and estimate how many leaks and identities are distributed over these sites from our experience with the collection of leaks.

10.1.2.1 General Purpose Providers

These providers are hosting all kind of data and are not aware of what content they are delivering. Therefore, they also not aware that they are hosting sensitive identity data.

- **Paste Pages** ($\approx 1\text{-}1000$ ident., $\approx 10\text{-}100$ leaks/day) are websites where people can share their text snippets, so-called pastes, with other people. Paste pages are often used to distribute code snippets, log files or small texts, but some of the pages are also used to share small-sized identity leaks. Two of the most popular paste pages are *PasteBin* [142] and *GhostBin*.
- **File Hosting Providers** ($\approx 10000\text{-}100\text{M}$ ident., $\approx 1\text{-}10$ leaks/day) are used to share larger leaks, which do not meet the size constraints of paste pages, or leaks that consist of multiple files. Common sites in this area are the free file hosters *TinyUpload*, *Zippyshare*, or *Mediafire*.
- **Bit Torrent** ($>10\text{M}$ ident., ≈ 5 leaks/year) is an ecosystem that is employed for sharing leaks that are either too big to be uploaded to a file hoster or have a high level of public interest.

10.1.2.2 Leak Only Providers

This group of providers is exclusively used for the distribution of leaks and is therefore typically maintained by the leakers themselves to spread their successes and prove their skills.

- **Leak-Related Paste Pages** ($\approx 10\text{-}50000$ ident., $\approx 10\text{-}15$ leaks/day) have the same concept of general page pages but are solely meant for the distribution of leaks. To support a larger range of leaks, these leak-related sites allow larger leaks with more than 10 000 lines. An example of a leaks-only paste page was the now discontinued *QuickLeak*.
- **Leak Repositories** (>1000 ident., $\approx 50\text{-}1000$ leaks/repo., ≈ 5 repo./year) are providing a multitude of older leaks. They are maintained by security enthusiasts as well as cybercriminals as a source of research or for starting illegal activities.
- **Leak Forums** ($\approx 1\text{-}100$ ident., ≈ 1 leak/day) are places where leakers share their achievements with peers and provide either samples of their leaks or the entire leak. A popular forum for the sharing of leaks was *LeakForums*.

10.1.2.3 Leak Announcement

While leak providers are mostly used for hosting leak files, there are also special places where leakers announce their leaks and where they distribute links to the corresponding hosted files. From our experience with leaks, a large fraction of leaks is either announced on social media or special leak forums.

- **Social media** platforms, such as *Twitter* and *Reddit*, are a communication and announcement platform that is used by security researchers and journalists as well as the leakers themselves. Security researchers, like Brian Krebs or Troy Hunt, are reporting about notable data breaches but do not reference them directly. The leakers mainly use these platforms to announce which service they have leaked and sometimes even share links to the extracted data.
- **Leak forums** are an exchange platform for leaks and are also commonly used to announce and publish leaks, but also to redistribute already published leaks. The main user group of these forums are leakers as well as people interested in talking about leaks and sharing their databases. To get free access to the published data, it is often required to register for the forum. Some well-known examples for leak forums are the discontinued *LeakForums*, *DemonForums* or *Flashback Forum*.

10.1.3 Data Origin

The information that is disclosed in identity leaks is often of similar type. In theory, the information that is contained in identity leaks corresponds to the information that individuals leave behind during the use of a service or system. In reality, such data can be multiple terabytes of size and is therefore hard to handle for an attacker, because it needs to be downloaded from the victim and transferred to a public server in case of an intended publication. Therefore, published identity leaks are often stripped down to only the most sensitive information of a service. This includes, but is not limited to, *user credentials* (username, email, password), *personal information* (home address, phone numbers, birthdays), and *financial records* (digital currencies, credit card, and bank account information).

How the data in a leak is put together, is dependent on where the data was initially extracted from.

- **Database Dumps (Dumps)** mostly originate from data exfiltration after a hack, which is mostly associated with an SQL injection. According to a survey by Risk Based Security [143], 57% of the data breaches are caused by such hacking. Since the data directly comes from a database system, the information is rather full-fledged and contains information about many users.
- **Phished Credentials (Logs)** originate from attacks against individual users, such as phishing or the installation of spyware on their computers. When the user types his credentials on the phishing site or his keyboard, they are extracted and sent to the attacker's server. The result of such broad attacks are long lists of credentials, sometimes called *logs* that were collected from a multitude of victims.
- **Targeted Personal Leaks (DOXes)** are the most dangerous type of leak and are detailed compositions of all available information about an individual, such as address, birthday, phone numbers, relatives, all kind of accounts with credentials. The publication of the information is used for the humiliation of a victim and sets a low boundary for further identity theft. The sources for these compositions are previously published identity leaks, publicly accessible registers, and social media accounts.
- **Credential Combinations (Combos)** are verified compositions of already existing credentials. Because of the condition that many users are reusing their passwords in multiple places, attackers try to “stuff” (reuse) and verify previously leaked credentials on a service that is different from the original leak. They try common passwords for the user in the hope that users are employing weak passwords. The result of these attacks are so-called *combolists* that contain combinations of username and password.

10.1.4 Representation

The representation of identity leaks is as diverse as the different sources of the leaks. Since leaks act as a proof of an achieved hack, the leaker does not care about the concrete representation, but is solely interested in somehow transferring the extracted data to the recipient. To make use of the leaked data for either awareness or malicious purposes, it needs to be parsed and moved to a common format.

In fact, the representation of raw identity leaks is closely related to the type and their origin, whether they come from database dumps or are composed of individually collected records. We have identified four formats that are most common for leaks.

- **Comma Separated Values (CSV)** is a line-based format where each line represents a single data record. Each field of the record is encoded at a fixed position in a row and is separated from other fields with a delimiting character, such as “,” (*comma*), i.e., where the name CSV comes from, “;” (*semi-colon*), and “:” (*colon*). Listing 10.1 shows an example of a leak in the CSV-format.

Listing 10.1: Leak records in CSV-format with “:” (colon) as a separator

```
First>Last:Email:Pass:Birthday
John:Doe:jdoe@mail.com:password123:1978-11-03
Susan:Bennett:s.bennett@provider.net:sunshine:1966-05-24
Peter:Smith:pete56@online.org:football:1988-02-15
```

Due to the generic structure of CSV, it is commonly used as an exchange format between databases and applications. This makes it predestined for use in leaks, where information should be easily parseable and importable into other attack tools. Unfortunately, the reality of CSV-parsing is difficult, because leakers use many different delimiters and do not properly quote or escape fields.

- The **Structured Query Language (SQL)** is a standardized language for the communication with DBMSs. Many DBMSs allow storing backups of a database with this language, so that the resulting script can be used to reinsert the data into another database. Since the export into SQL-scripts is built into many DBMSs, it is an easy way for an attacker to get a full copy of a compromised database system. A major challenge when dealing with SQL is the variety of dialects that let representations differ between DBMSs. Listing 10.2 gives an example of a leak record in the SQL format.

Listing 10.2: Leak record in SQL-format

```
INSERT INTO user (fn,ln,email,pass,bday) VALUES ('John', 'Doe', 'jdoe@mail.com',
, 'password123', '1978-11-03');
```

- The **JavaScript Object Notation (JSON)** is an object-based key-value format that is used for data exchange on the Internet and can be directly interpreted

by the JavaScript engine of a web browser. To make data easily available to web browsers, many services and database systems serve their data in JSON right away. One example of a DBMS serving JSON is MongoDB, which is a popular database system in the web. However, due to a configuration issue in early MongoDBs that allowed unauthorized access to all data [144], there is a significant number of massive leaks in JSON. Listing 10.3 shows an example of two records from a MongoDB.

Listing 10.3: *Leak records in JSON-format as it appears in a MongoDB*

```
{ "_id" : {"$oid" : "4031ef453a"}, "first": "John", "last": "Doe", "email": "jdoe@mail.com", "password": "password123", "bday": "1978-11-03"}
{ "_id" : {"$oid" : "4031ef453b"}, "first": "Susan", "last": "Bennett", "email": "s.bennett@provider.net", "password": "sunshine", "bday": "1966-05-24"}
```

- In addition to the already listed formats, there are some other rarer formats in which leaks can be expressed, such as *HTML*, *ASCII-tables* or completely *customized* or *freeform* content. Such types of leaks are most difficult to be interpreted, because the formatting is difficult to infer, and information is organized arbitrarily. In the case of HTML, a leaker uses HTML's `table` or `div` elements to create a tabular structure for the data. ASCII-tables are similar to a formatting with CSV, but additional whitespace characters are used to visually align the data for a text editor. Although such tables are easy to read for a human, they are harder to interpret for a program.

10.1.4.1 Distribution of Formats

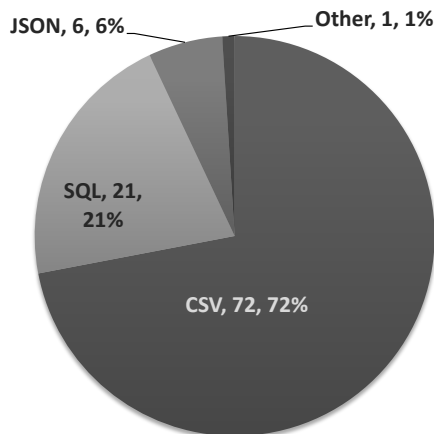


Figure 10.2: *Distribution of leak formats for 100 of the largest public leaks*

To make clear on how widespread the presented formats are for real-world leaks, we have analyzed the leak files of some of the largest 100 public leaks (as of May

2015) and have assembled the diagram in Figure 10.2. This diagram highlights that all major leaks can be grouped into the above formats and that all but one leak can be categorized into either CSV, SQL, or JSON. Furthermore, it becomes apparent that variations of CSV are most common with a share of 72%. On the second position are SQL formats, which take another 21% of the share. Only a rather small fraction is represented in the JSON format.

When looking at the detailed formatting of some of the largest leaks, there are differences in how the leaks comply with the above formats. In particular, many leaks have large deviations from a well-formed CSV format, which makes an automated parsing of the data difficult. Two of the main issues are an inconsistent use of delimiters within a single record and a varying number of fields between multiple records.

10.2 Leak Processing Workflow

As identity leaks come from various heterogeneous sources in a wide range of formats and with different types of information in them, it is a challenging task to find out who has become a victim and to what extent their personal information is affected. Even more, the volume of identity data with billions of individual identities is staggering and can be considered as a typical Big Data problem. To warn individual about their publicly leaked data, we need to be able to process this data efficiently. Our Big Data workflow from Section 3.2.3 is one option for the efficient processing of Big Data and is also used as a foundation for our leak processing workflow that transforms raw leak data to an intelligence feed for affected victims. Figure 10.3 illustrates a rough overview of this workflow.

In the beginning, raw leaks are collected from the previously mentioned sources. Each leak is then passed over to the identity extraction step, which consists of the syntactical analysis of the leaked records, the extraction of relevant identity information from each record and the interpretation of the extracted content as the properties of a digital identity. The outcome of this step is a digital identity object. If the extracted identity contains password information, there is a hash recognition step that determines the password storage method. This information is important to evaluate the severity of the leaked information. The easier it is to get the cleartext password from a record, the easier it becomes to misuse the information. In the end, the extracted digital identity with its severity is passed through a subscriber filter that selects only those records that are relevant for the subscribed entities, such as web domain or email address owners. Each subscriber receives an individual intelligence feed with all his leaked records.

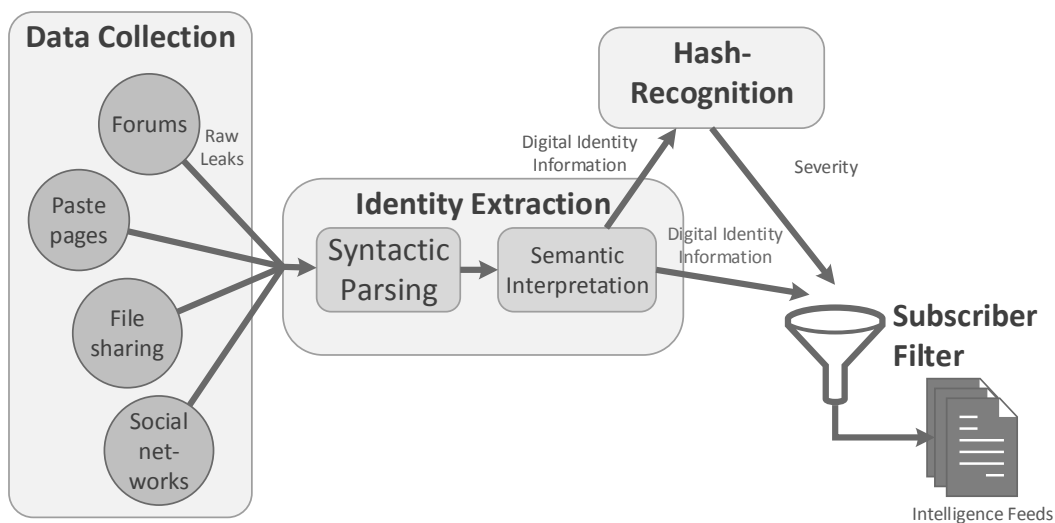


Figure 10.3: Workflow used for processing leaks

10.2.1 Data Collection

The majority of leaks today can be found on paste pages and through announcements on leak forums. Leaks from these two types of sources are either downloaded by an automatic crawler or manually over a web browser.

Automatic Crawler For paste pages, social networks and selected announcement pages, we are using a crawler that periodically checks for new publications and downloads potential leak files automatically. The crawler has two types of modules. In one module altogether 14 different paste and announcement pages are checked in a second or minute interval for new pastes or posts. In the other module, Twitter tweets of hundreds of accounts are analyzed, links to file sharing and paste pages extracted, and the corresponding files downloaded. The downloaded files are then further checked for sensitive information and are discarded if they do not contain such information. The check employs regexes that are matching common and distinguishable user information, such as email addresses or password hashes.

With the automatic crawler, we have mainly collected leaks at the beginning of our leak collection experiment in 2014, because at this time not many larger leaks were published and a large number of smaller leaks was spread over paste pages. As an example, in July 2015, our crawler collected 459 leaks with a median of 11 leaks per day. Eventually, we have gathered more than 20 000 leaks over two years. Among those are 4100 leaks with more than 1000 accounts and 50 leaks with even more than a million accounts. However, especially in recent years, the situation has changed, and the majority of leaked identities is now distributed otherwise.

Manual Collection We have recently moved from the automatic collection of leaks to the manual collection because there is an increasing number of larger leaks shared on either leak forums or repositories that are difficult to crawl automatically. Generally, leak forums only have a relatively short lifetime, because they are sooner or later attacked by the members of other forums or leak collectors that do not want that more exclusive leaks are further spreading into the public. Therefore, there is always a kind of primary forum used at a time for posting new leaks. By monitoring these primary forums, we can obtain links to the most recent leaks. The access to the links is either public or is granted by registering a user account in the forum. Since the start of our collection experiment, we collected more than 6000 identity leaks that usually hold 10 000 or more identities. As an estimation, all these leaks are exposing 10 billion individual user accounts.

10.2.2 Identity Extraction

The extraction step is responsible for obtaining all the digital identities from raw leaks. The extraction is organized into three sub-steps: the syntactic analysis, the data extraction, and the semantic interpretation. We are referring to the first two also as syntactic parsing.

Digital Identity Model Before we start with a detailed description of the extraction steps, we want to present our understanding of a digital identity based on the information we have observed in the collected leaks. This model is the foundation for the threat indicators contained in the resulting intelligence feed. Figure 10.4 illustrates the entity relations of the model.

At the center of the diagram is the `LeakRecord`, which stands for a single extracted digital identity. All the related objects represent certain aspects of this identity. Left to the `LeakRecord` entity is the `LeakSource`, which provides meta-information about the leak from which the record was extracted. The two entities `EmailAddress` and `Domain` at the top are giving information on the email address associated to a record and also act as the main identifier for an identity. Later, the email address is used to filter the records by a subscriber. The entities `BankAccountData` and `CreditCardData` at the upper right are meant for financial information and together indicate whether the identity is susceptible to immediate financial losses. The two entities at the bottom right give personal details about the identity, such as the name, contact data and the birthday. This kind of information is especially critical as it can be used for identity spoofing. The `Credentials` entity at the bottom left holds information on login credentials for a service and is most critical due to passwords reuse. To this point, the mentioned entities are populated by the identity extraction. The remaining `HashRoutine` entity is provided by the *hash recognition* step and pro-

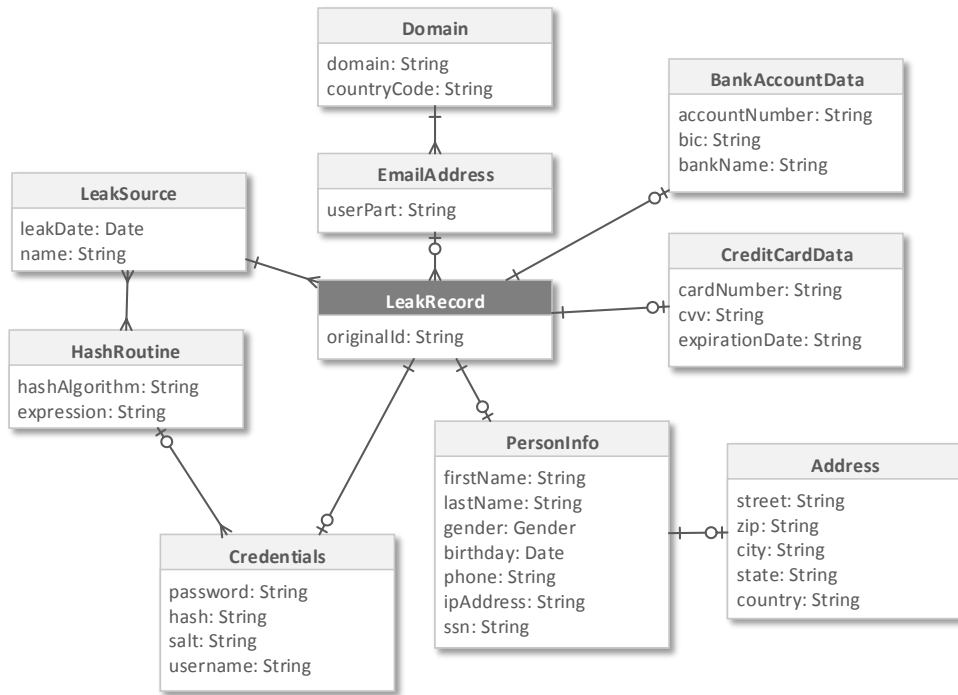


Figure 10.4: Proposed data model for leaked digital identities

vides information about the hash algorithm used to store the credentials in the leaked service. Later, the leak severity is derived from this entity.

10.2.2.1 Syntactic Parsing

The major challenge for the handling of identity leaks is the wide variety of formats they are represented in, such as CSV, SQL, and JSON. Similarly to the normalization of log events, we also want to bring records in these formats into our common digital identity model. However, in contrast to log normalization, the format in which a leak is represented is not known beforehand. Furthermore, there are slight variations in the formats that impede the simple application of a one-fits-all parser. We have subdivided the syntactical parsing into the classification of a leak’s format and the format-specific parsing. Still, there might be cases where the format or its particular variant cannot be determined automatically. In that case, a manual extraction becomes necessary.

Format Classification The classification tries to find common characteristics of a format and then makes an estimation in which formats the file could be. In the case of an SQL file, there have to be `INSERT INTO` statements. For a JSON file, every line

has to start with curly or square brackets. A CSV file is considered to be the default case. So whenever none of the above matches, we always assume a CSV file. A more detailed analysis follows in the dedicated parser. Generally, once the format has been determined, the corresponding parser is executed. If the parser fails, the next possible format in the above order is checked for applicability.

CSV The original CSV standard [145] specifies the delimiters, escaping characters, and quotations used for the encoding of data records in a line-based format. However, the reality of CSV encoding looks different, because there are significant variations from the standard. There are variants with different delimiters (`,` `;` `|` `\t` `_`), escaping characters (`\`), and quotations (`"` `'`). Listing 10.4 shows the same data record in different CSV variants as they appear in leaks.

Listing 10.4: Data records in various CSV variants

```
1956-07-04,null,john.doe@email.com,5f4dcc3b5aa765d61d8327deb882cf99
1956-07-04,null,"john.doe@email.com","5f4dcc3b5aa765d61d8327deb882cf99"
1956-07-04;<blank>;john.doe@email.com;5f4dcc3b5aa765d61d8327deb882cf99
1956-07-04 | null | john.doe@email.com | 5f4dcc3b5aa765d61d8327deb882cf99
1956-07-04 <blank> john.doe@email.com 5f4dcc3b5aa765d61d8327deb882cf99
```

A challenge in CSV parsing is the detection of the current variant with all its properties. Our approach is to look for characteristic fields with distinct formats, such as email addresses, hashes, or IP addresses. Once we find such field, we analyze the characters before and after it to reason about a possible delimiter character. The extracted candidate delimiter is then applied to the whole line and the number of fields it would separate is determined. This is done for the next consecutive lines and the deviation in the number of separated fields is calculated. If the deviation is sufficiently small, we have found a possible delimiter. Now we can continue to find the quotation characters which must always precede or follow a delimiter. Only when we have also found out the quotation characters, we can reason about the escaping that is used in the file.

SQL Theoretically, the SQL language is standardized in an ISO standard and would be easy to handle with an appropriate parser. In practice, however, there is a wide variety of dialects employed by all sorts of DBMSs that deviate from the standard. Furthermore, leakers that produce SQL dumps are not necessarily using compatible tools and thereby generate customized format variants.

A chance to deal with the variance of the format is to only focus on `INSERT INTO` and `CREATE TABLE` statements for data import. Especially the syntax of the `INSERT INTO` statement is rather stable across all dialects and allows to be parsed with only a few different parser variants. Listing 10.5 shows the main structure of an *insert* statement.

Listing 10.5: *Basic structure of an INSERT INTO statement*

```
INSERT INTO <tablename> (<field1>, <field2>, ..., <fieldn>)?
VALUES (<value1>, <value2>, ..., <valuen>)+;
```

The major points of variance in this structure are the quotations within the <tablename>, <fieldn> and <valuen> definitions. Also, in some variants, the field definitions are omitted in favor of a prior CREATE TABLE. Also, there can be multiple instances of a value definition.

In order to process SQL dumps, we have created a custom SQL parser that looks at lines containing the INSERT INTO phrase. After the phrases are found, the parser tries to parse their related statement. During the parsing, it can detect the currently used statement variant. For a more complete picture of the tables in a dump, the parser is also trying to find and analyze CREATE TABLE statements. Even though there are many custom modifiers in such statements, the core of the statement mostly has the same structure across dialects. Hence, the extraction of existing fields and their data types is mostly possible.

JSON Dumps in the JSON format are usually well parseable with a common JSON parser. There are two ways in which data is represented in JSON, either there is a single JSON object that bears all the information from a database, or there is one JSON object per record in the database. For dumps from MongoDB, which is the primary source of JSON dumps, the data is represented in the latter and can be pushed line-by-line into the parser. In the case of a hierarchical structure, we are mapping all records into a flat structure beforehand, so that fields can later be referenced with a single key.

Manual Parsing In the case where the automatic parsing does not work, we are falling back to the manual handling of the leak. Due to the similarity to event parsing, we are also using manually provided named-group regexes to extract records and their identity fields. In contrast to the normalization of event information, an individual regex per leak has to be provided, as the leak formats are simply too different. The extraction of information from the matched regexes is accomplished with the same principles as known from the event normalization, meaning that matched groups are assigned to equally named fields of a digital identity object. This also leads to the advantage of manual parsing over automatic, because now the extracted data records do not need to be further analyzed to find out which record field belongs to which digital identity field. Thus, a semantic interpretation, which would follow the syntactic parsing, is not necessary anymore.

Challenges Apart from the parsing of the above formats, there are three further major challenges that influence the success of parsing.

- **Character encoding:** As leaks originate from databases around the world, the encoding is a major issue when extracting information. While UTF-8 is the de-facto standard for the exchange of data in various character sets, some databases are still configured to use regional character sets, such as Windows-1251 for Cyrillic or GB23 for Chinese characters. Detecting which character set is used in a leak is currently one of the most challenging problems. Programming libraries that should be able to detect character sets, such as jChardet or jUniversalChardet, do not work reliably, because only a few data fields like usernames or addresses contain non-ASCII characters. The remaining of the file mostly consists of special characters or ASCII characters. Our current approach of finding the employed character set is to try a decoding with a variety of character sets and skip over those that produce errors during decoding. Only if multiple sets are matching, we are falling back to the default UTF-8.
- **Mixture of formats:** A leak cannot always be categorized into a single format and therefore be processed by a dedicated parser. We propose to solve the issue by parsing the data in multiple passes with different parsers. Each pass is removing the fractions from a file that it was able to process and hands the remaining file to the next parser pass. Ideally, in the end, the remaining file after all passes will be empty.
- **Leaker credit banners:** Some of the leaks published have the banner of its original leaker at the beginning or end of the data. These banners do not follow the usual format and therefore confuse the parser that then produces wrong records. To counteract that problem, the parser can check the read data for soundness and discard records that deviate, e.g., by the number of fields in a record, from the majority of the other records. Although banners are disturbing the parsing, they can also contribute meta-information about the leak, such as the original source.

10.2.2.2 Semantic Interpretation

The result of syntactical parsing is a tabular data structure covering the raw records from the leak. In many cases, this table does not have further information on the data columns and the meaning of the corresponding data fields. Still, to convert records to digital identity objects, the meanings of these fields are required. This is the point when the semantic interpretation comes into play. It analyzes each column of the tabular data structure and tries to derive its meaning from the type of values it presents. We are using two different field interpreters to determine the type and meaning of a field.

- **Word List Interpreters** are using a predefined wordlist to discover known terms in the values of a field. Use cases for such interpreter are usernames, person names, city names or clear text passwords.

To create specialized wordlists for these field types, candidate values can be extracted from leaks where the field semantic is already known. For example, one could start with a basic list of 100 first names to identify a name column in one leak. Once the column has been determined, new candidate values for first names can be extracted from it and added to the preexisting wordlist.

- **Regular Expression Interpreters** are detecting patterns that are recurring for certain data types. In particular, there are password hashes, birthdays, telephone numbers, email addresses, and credit card numbers in this category. The description of such patterns is provided as regexes. For an email address, the regex `(.+)@(.+)\.[a-z]{2,6}` could be used. It looks for the characteristic `@` at the middle of an address and checks whether there is a corresponding top-level domain in the email domain. For the detection of password hashes, such as MD5 and SHA1, the matching is even easier, because hashes have a fixed length and are written as a hexadecimal number. An MD5 hash could be matched with `[0-9a-fA-F]{32}`.

10.2.3 Hash Recognition

Credentials play a special role in the extracted identity data, since they are an attacker's key to a victim's account. While cleartext passwords allow the login into an account right away, hashes are a method to obfuscate the password and prevent direct misuse. However, the usage of plain hashing functions, like MD5, SHA1, and even SHA512, does not guarantee sufficient security, because plain hashes are susceptible to dictionary attacks. For more secure password storage, hashes should be extended with salts and multiple iterations of the hash calculation. The Password-Based Key Derivation Function 2 (PBKDF2) [146] is a standardized function that implements these security measures to securely store passwords or generally to derive shared keys securely.

The hash recognition step is responsible for determining the *hash formats* and *hash routines* used in a leak and derives a threat level for the affected users.

Hash Format This describes the way the hash is represented and is usually equal to the output of one of the common hash functions, because finally all hashes are produced as an output of a hash function.

Hash Routine This describes the way in which the hash is calculated, including the employed hash functions, the number of hash iterations and the fact whether a

salt or pepper is integrated. The routine specifies in detail how all these components are combined or concatenated with each other. As a description for hash routines, we are choosing the notation used by hashcat² and John the Ripper³. For instance, a routine that creates a SHA256 hash as the concatenation of a salt with the password would be represented as `sha256($s.$p)`. The website Vigilante [141] gives a broad overview of routines occurring in leaks over the past years.

Our recognition algorithm starts with an analysis of the hash format by matching against a number of regexes, such as `[0-9a-fA-F]{40}` for SHA1. Due to the fact that the routine must produce a hash that is conforming to the determined hash format, the hash format majorly limits the number of possible hash routine candidates. Sometimes the format is even so unique, that there is only one possible routine candidate left, such as `bcrypt` having a `$2a$` prefix.

As soon we know what kind of formats are used in a leak, we continue to try all applicable hash routines with a set of candidate passwords. If a threshold of n resolved passwords has been reached for a combination of routine and candidate passwords, the currently tried routine is most likely the hashing routine that was used to store the passwords in the leak. As different routines might create the same hash under certain circumstances, such as `MD5($p1)` and `MD5($p2$s)` if $\$p1 = password123$, $\$p2 = password$, and $\$s = 123$, n should be larger than one but still small enough to deliver a confident result in the shortest possible time. Algorithm 1 illustrates the checking routine in more detail with $n = 10$.

The backbone of the algorithm is the selection of appropriate candidate passwords by the `candidate_pws` function. Based on the current record, this function generates passwords that could most likely be a match for this record's hash. We propose the reuse, top-password, and username strategy for the candidate generation.

10.2.3.1 Reuse Strategy

The reuse strategy is relying on the fact that users are frequently using only a few or even a single password across all their service accounts. According to our research [37] on this topic, more than 20% of users are reusing the same password among two or more services. According to a survey by Das et al. [147], even 51% of the respondents specified that they are reusing passwords.

The advantage of the reuse strategy is that many hashes can be resolved in a short time, because 20% of all tried passwords are matching. Furthermore, even if a strong password policy was employed for the assignment of passwords, many reused passwords are strong enough to fulfill every policy. On the other side, there is a

²Hashcat - <https://hashcat.net>

³John the Ripper - <http://www.openwall.com/john/>

```

Data: parsed_leak
Result: routine
hash_format ← derive_hash_format(parsed_leak);
for routine in filter_routines(known_hash_routines, hash_format) do
  count ← 0;
  for record in parsed_leak do
    if count ≥ 10 then
      | return routine;
    end
    for password in candidate_pws(record) do
      | hash ← calculate_hash(routine, password, record[salt]);
      | if hash = record[hash] then
      | | increment count;
      | | break;
      | end
    end
  end
end
return null;

```

Algorithm 1: Finding routines of leaked password hashes

limitation to leaks that have a user identifier and there must be many previous leaks that are providing information on the users in the leak. According to our experiments, around 40% of the users in each leak also appear in a previous leak. Thus, in a leak with 1000 records, on average 80 passwords can be resolved.

10.2.3.2 Top Password Strategy

This strategy leverages the fact that users are using passwords that are easy to remember. Figure 10.5 shows the use of the most common passwords, such as *password* or *123456*, from altogether 2.1 billion leaked credentials. As an example, 7.36% of the users are using one of the top 1000 passwords.

Assuming that we are limiting the generation to the top 1000 passwords, we would have an average of 74 passwords recovered on a leak with 1000 passwords, but at a much lower match rate than the reuse strategy. An advantage of the top password strategy is that it is independent of user-related data like email addresses or usernames and therefore also works on a plain hash list. However, the strategy also has the disadvantage that there is a much lower success rate if a stronger password policy is enforced, because that would rule out the majority of the top passwords.

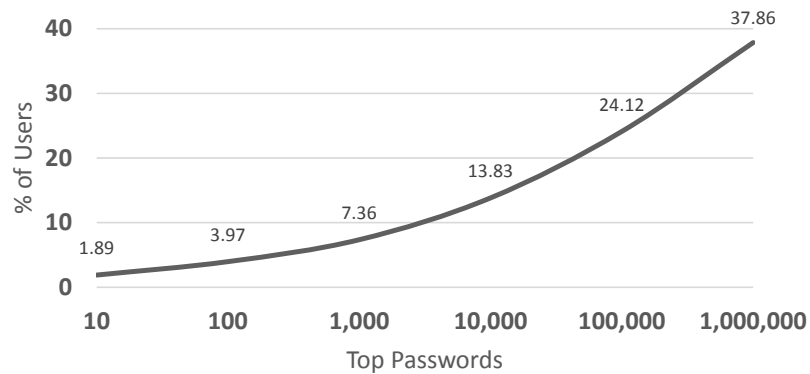


Figure 10.5: Usage of top passwords

10.2.3.3 Username Strategy

A last alternative to the above two strategies is the usage of the username as a password. According to Yampolskiy [148], 2.7% of users are using their username as their password. Therefore, a corresponding candidate generator could extract the username and the email address from the provided record.

The strategy has the advantage that it has a relatively high match rate, because there are only two possible candidates per record. However, similarly to the reuse strategy, there is the requirement that usernames and email addresses are provided in the clear.

10.2.3.4 Analysis on Strategies

Each of the above strategies has its own advantages. As a general solution, we are first trying the reuse strategy, because it has the highest match rate. In the case of smaller leaks, which means there are only very few candidates, the top password strategy is usually more successful. Only as the last solution, we are using the username strategy, since it has the lowest match rate. All in all, with a combination of all techniques, we can determine the used hash routines for almost all leaks in a few seconds or, in the worst case, in a few minutes, depending on the calculation complexity of the applicable routines.

10.2.4 Subscriber Filter

The subscriber filter prepares the information provided by the identity extraction and the hash recognition into an event feed for the subscribers of our identity-based threat intelligence. While the information from the identity extraction still incorporates all the leak data in a record-based data structure, the subscriber filter only delivers the

CHAPTER 10. USING IDENTITY LEAK DATA AS THREAT INTELLIGENCE

affected identity and the type of information that was leaked about it. Figure 10.6 shows this filtering process in more detail.

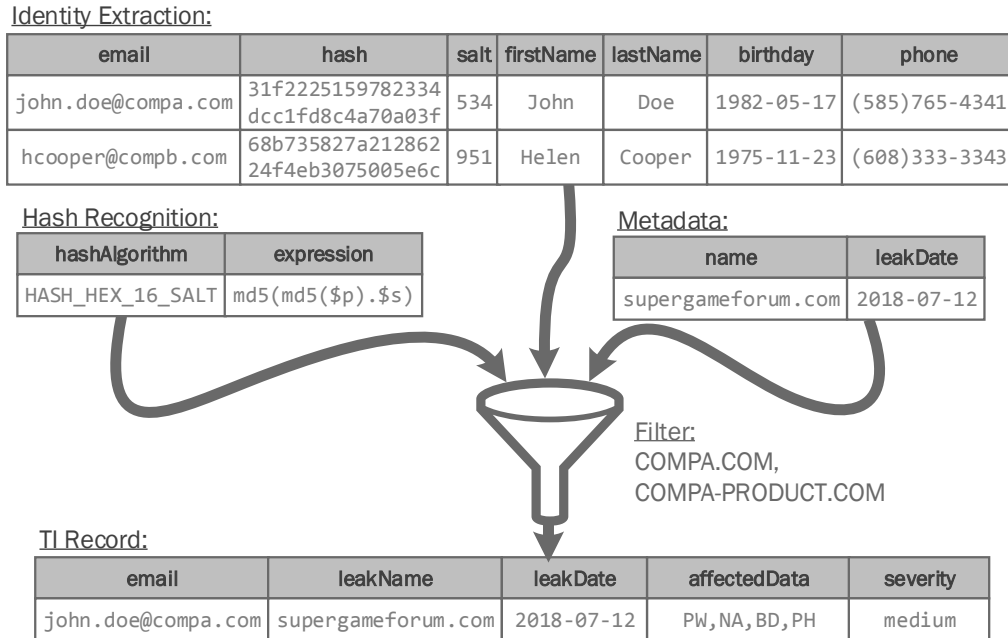


Figure 10.6: Filtering of the workflow outputs for a specific feed subscriber

As an example, there are two records for *John* and *Helen* within a leak. The leak contains credential information, meaning the email address and hashed password, the users' names, birthdays and phone numbers. The hash recognition has determined that the format is a 16-byte hex notation with a salt, which indicates that hash functions like *MD5*, *RIPEMD128* or *NTLM* were employed. Going deeper into analysis with the top password strategy, the recognition has eventually confirmed that the *vBulletin* hash routine was used, because both users have used weak passwords. As for the metadata, we know that the leak is from a site called *supergameforum.com* that was leaked at around July 2018. Taking all this information together, the filter now only looks for leak records that are relevant for the company called Company A, which owns the two domains *compa.com* and *compa-product.com*. Of the records in the leak, the first record of John is applicable to the domains of Company A, because John's email domain equals *compa.com*. Therefore, the subscriber filter assembles an intelligence feed record for John with the most important information. This information embraces the affected user by its email address, the name of the original leak, its date of leakage, the type of the data that was affected and the severity that is induced by the type of information affected. It has to be noted that the TI record does only contain the most important information necessary to react on a leaked information. In

particular, no concrete data fields are provided on top of a simple email address and the leak source.

In a real-world setup of the subscriber filter, of course, there would be a multi-tenancy setup where each subscriber only sees the records that affect him. A single subscriber can subscribe for multiple domains or even specific email addresses. In addition to events on the leakage of domain users, the filter also creates an event in case the domain itself is leaked and outputs all records that are part of this leak.

10.3 Leveraging Leak-Related Threat Intelligence

So far, we have described the nature of identity leaks and have presented a workflow that issues events for observed leaks in an automated fashion to corresponding subscribers. As a next step, we are integrating the provided information into a SIEM to improve the detectability of credential misuse within IT landscapes. We are proposing two methods in which leak information can be utilized in a SIEM.

10.3.1 Alerting

The simplest way is to convert the events from the feed to alerts in the SIEM system and show them in a dashboard or alert listing. In REAMS, these alerts are represented as OLF events with corresponding alert tags and can therefore even act as the basis for further correlations, such as in signatures or queries.

Based on the severity of the alerts, the operator can set up actions to be executed automatically on the occurrence of a leak or can manually act on the threat. For example, the following actions could be conducted.

- Send out an email to the affected user to ask for a password change, if credentials are leaked, or higher awareness towards spear phishing, if personal data was stolen.
- Reset the passwords of the affected user in the network.
- Disable the account for an affected user for very critical leaks.
- Start an investigation if the leak originates from one of the enterprise's domains.

Of course, for less severe leaks that do not have credentials included, it is not necessary to actively change configurations, such as passwords or accounts, in the environment. Still, even the leakage of personally identifiable information can lead to a larger attack surface for social engineering, which is known to be a frequent attack vector in highly sophisticated attacks [8]. To protect victims against the misuse of their data, they should be notified as early as possible about a raised alert.

10.3.2 Signature Derivation

As outlined earlier, signatures in a SIEM describe attack patterns that are observable in log events. As a method of initial compromise, credential misuse is commonly part of complex attack patterns, as they are occurring in APTs. Therefore, by integrating the leak events into signatures, we are able to detect attacks either in their early stages or that were previously under the radar of the IDS, mostly due to the fact that misused accounts were not known to be compromised in a recent leak. Considering that complex patterns are best represented with multi-step signatures, then the event on a leaked credential is best used as a type of precondition, because it is usually the requirement to perform elevated activities.

For a better understanding on the use of a leak event in multi-step signatures, we are taking the typical scenario where an attacker got his hand on a leaked credential for a user from his target network. Usually, the attacker then would use these credentials in the target network to gain access. Assuming that the target network has auditing features enabled for logins, then any login would be logged together with the user, the time of login and the IP address of the client from which the login was requested. This log is also the possible point of detection for a signature, because an attacker most likely will use a different computer and logs in from another location than the legitimate user. Therefore, if the login location, either geographically or network-wise, deviates significantly from the previous login, then this login is potentially malicious. Although such a signature can work independently, a leak event for the corresponding event gives the signature a much higher accuracy. For example, should an employee travel to another branch of his company, the signature without leak event would already indicate an alert if that user would log in from the branch's computer. If the leak event is considered in addition, there would be no alert. Figure 10.7 shows the state graph for such a signature with a leak event.

At the beginning, the signature looks at normal login behavior of the compromised user, meaning it looks at the login location of that user and remembers it for later. After that, if there will be leak events affecting this same user, the signature goes into the critical state and will look for other logins of that user. The trigger for the signature will be a login that has a different login location than the initial login before the leak. Of course, this signature only works if the credentials have not been exploited already in the monitored environment. Hence, it is crucial that new leaks are provided within the threat feed as early as possible.

10.4 Conclusion

Identity leaks are a major threat to the security of computer systems, networks, and their users, as criminals are misusing the contained identity records for either social

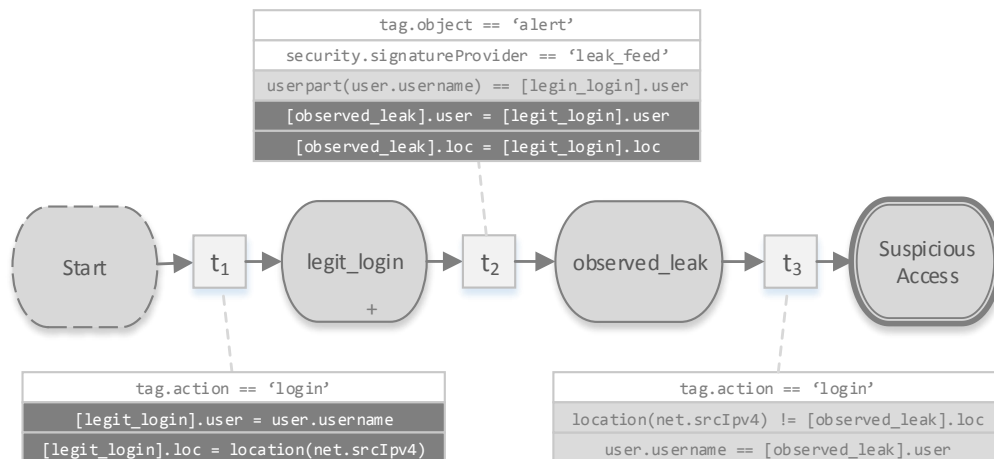


Figure 10.7: EDL signature for finding suspicious logins

engineering or unauthorized access to accounts. The victims of these leaks often do not even know that their data is floating around publicly and is actively misused. Only in the past few years, thousands of leaks with billions of affected digital identities have surfaced in the public Internet, putting a large fraction of the Internet community at risk.

In this chapter, we have tackled the problem of identity leaks and the related credential misuse. We have proposed a workflow that collects publicly available identity leaks and informs the identities included therein about the fact that their data might be misused. The core part of the workflow consists of the extraction of identity-related information from raw leak files and an estimation of the criticality of a leak for the affected identities. The extraction is achieved with the help of regexes that are either derived automatically from the data format or are provided manually by an operator. In the end, many of the normalization concepts developed for the event processing workflow could be reused for leak processing. The estimation of the criticality is based on the type of leaked information and the strength of the password storage. Hence, the criticality is a measure for the victim on how important it is to react to the leak in his environment, e.g., by resetting his password. The outcome of the two previous steps is a condensed leak event that gives the most important information about a particular leaked record, such as the affected identity, the type of affected data and the criticality score. Using our workflow, we can convert data leaks either fully automatically or semi-automatically into a threat intelligence feed for leaked records. The processing speed for both, the extraction and evaluation are fast enough to beat the speed of new leak publications.

The creation of the threat feed is the precondition for making leak information available for security analytics. To show that the information can also be utilized in a practical context, we have presented how leak information can be integrated into a

CHAPTER 10. USING IDENTITY LEAK DATA AS THREAT INTELLIGENCE

SIEM, or particularly in REAMS. The leak events can be presented as alerts to the operator of the SIEM, but the events are also helpful in creating advanced signatures being aware of leaks. The integration of leaks allows to now find more covert attacks related to credential misuse, which were previously only seen after their successful execution. As credential misuse is typical for the initial compromise phase of APTs, attacks are detected earlier in the kill chain and can thereby prevent major damage in the target network. At the moment, the information on a leaked email address cannot be expressed in existing CTI formats and is therefore also not supported in existing SIEM products. Hence, to widely make use of leak information in enterprises, the support for this field has to be extended in both, formats and products.

Part III
Case Study

Chapter 11

Real-Time Event Analysis and Monitoring System (REAMS)

In the previous chapters, we have mainly looked into the theoretical aspects of a SIEM that is capable of handling large amounts of event log data for security monitoring. In particular, we have proposed an event processing workflow that a modern SIEM can utilize to process and analyze security-related log events in near real-time. In this chapter, we are moving from the theoretical considerations to the practical use of our prototypical security analysis platform, named Real-Time Event Analysis and Monitoring System (REAMS). REAMS has been built up gradually from 2012 and is based on our experiences in the topic of event analysis. Since then, it has become a comprehensive system that is used for conducting experiments and trying out new approaches for processing and analyzing event logs. The platform is practically used within our research group for monitoring our IT infrastructure at the HPI as well as to analyze event logs of project partners. In the following, we are giving a rough overview of our REAMS deployment as it is used in our testbed environment and then follow with concrete use cases of the platform.

11.1 Real-World Deployment

The whole REAMS platform consists of multiple *agent* components, a *core server* and a selection of *user interfaces*. Such a deployment is generally popular for SIEM systems and has also been derived from our experiences with our previous prototypical SIEM implementation, called Security Analytics Lab (SAL) [149, 150]. It allows a more fine-granular separation of tasks and access levels of the system. Figure 11.1 gives an idea of how the current SIEM deployment in our lab environment looks like.

The main components have been implemented in the Java programming language and are running within the Java SE runtime environment. Therefore, REAMS is

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

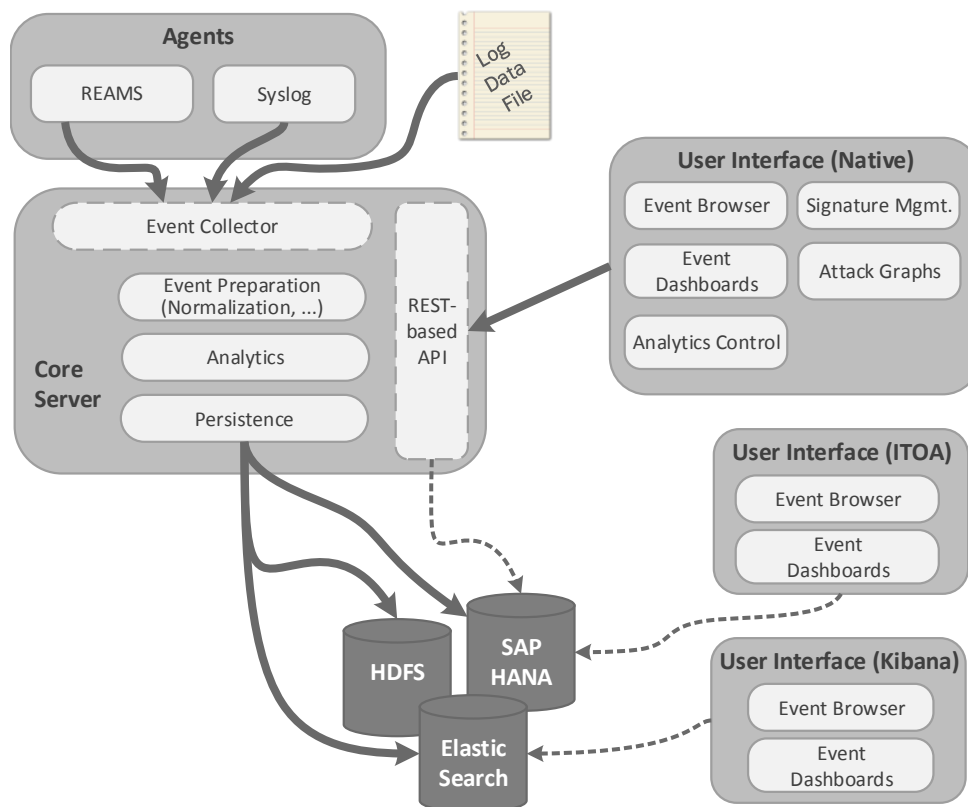


Figure 11.1: REAMS deployment running live in our lab

installable on every modern operating system that supports Java. However, we are also accessing the data managed by REAMS with other third-party user interfaces. Below is a detailed description of the REAMS components.

11.1.1 Agents

The agents provide the connection between the environment and the REAMS core system. Generally, an agent is installed on an event data source or acts as a forwarder for another agent. As the main task, an agent splits up a log file into the single events that are contained in the file. After the splitting, the agent forwards each event to the core server it is configured for. Theoretically, an agent can forward its events to multiple core servers if data redundancy is required.

In our lab environment, we are mainly using Syslog agents for all UNIX-based systems. As Syslog clients are installed on most of these systems in the standard OS distribution, there is no need to install further software to support the monitoring of a system. In contrast, all Windows-based systems do not provide an easy way to

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

forward events to an external system, so that a dedicated agent is required. As a result, we are using our custom REAMS agent to export events from Windows systems.

11.1.2 Core Server

The core server is the main component of REAMS and is responsible for all SIEM- and IDS-related tasks. It implements the architecture we have proposed in Section 5.3 and therefore takes over all event processing tasks in REAMS.

As an input, the core server retrieves events from the agents or directly from log files via its *event collector* module. Similarly to our architecture, the event processing is subdivided into *event preparation*, *analytics* and a separate *persistence* into several databases. The event preparation takes care of all the steps necessary to later efficiently work with the events, meaning the transformation of raw events to OLF events. The persistence moves the normalized events into a persistent storage, where it can be correlated and accessed in larger chunks. In the end, the analytics module realizes the analytical algorithms of REAMS, which covers single- and multi-step signature detection as well as various unsupervised outlier detection approaches, which are based on k-means [151], ocSVM [118], and k-modes. Based on the complexity of the algorithms, the processing is either conducted in stream- or in batch-mode.

```
Real-time Event Analytics and Monitoring System
REAMS> i
REAMS/Import> ?list
abbrev name      params
ld      load_data   (filename)
ldwc    load_data_wc (basepath, wildcard)
ldcsv   load_csv     (filename)
sr      set_start_rule (startrule)
litoa   load_itoa  (p1)
REAMS/Import> load_data events.csv
```

Figure 11.2: Command-line interface of REAMS

To make the retrieved events and analysis results, such as generated alerts, available to a security operator with a graphical user-interface (GUI), the core server also provides access to all its data via a REST-based API. An external user interface can then choose whether it retrieves its information either from this API or directly by accessing the persistent storage. For a quick interaction with the core server, there is also a command-line interface (CLI). This interface can be used to load event files, install additional modules, and trigger analytical algorithms. Thus, the core server can even run standalone without an agent or a GUI.

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

11.1.3 Graphical User Interfaces

The graphical user-interfaces (GUIs) are the primary way to interact with the REAMS core server for a security operator, because the command-line interface only supports the most basic tasks to be performed. Our goal is to make the interface choice as flexible as possible and therefore provide multiple ways of interacting with the server. The most powerful way of interaction is through the REST-interface, because it provides access to the data and also allows to execute functions on the server. Another choice for a GUI is to access all event data directly from the persistence layer of REAMS. In our deployment, there are HDFS, ElasticSearch, and SAP HANA as data stores.

11.1.3.1 Desktop Client Interface

The interface is a self-developed desktop application for REAMS and provides the most functionalities on top of the server. It connects directly to the core server over the REST-interface and therefore enables the starting of analytical functions and the management of signatures for the core server.

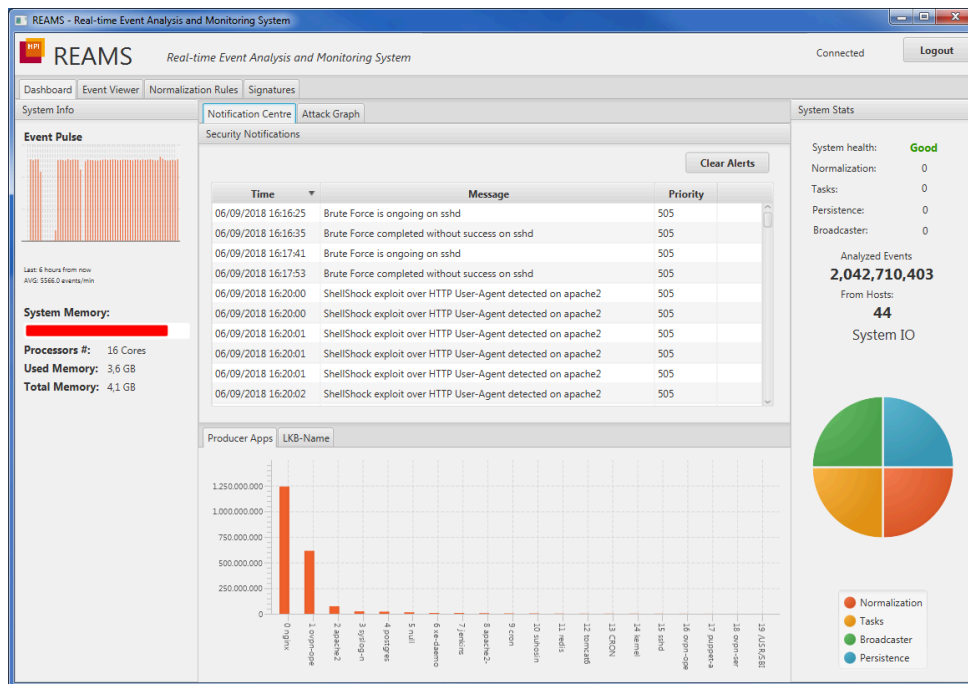


Figure 11.3: Dashboard of the REAMS desktop client

As the main functions, the application has a dashboard that summarizes useful information about the system and the currently monitored environment. This dashboard includes an alert overview, the applications and normalization rules of normalized events, the current system resource consumption and the number of all retrieved

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

events and monitored hosts. Figure 11.3 shows the main tab of the application with some previously raised alerts. A special feature of the dashboard is the presentation of alerts in an attack graph. This attack graph visualizes all critical detected attacks and their relations to each other. Using this visualization, a security operator can trace the path of an attacker.

In addition to the dashboard, an operator can see a list of all events in the system and can apply filters that only show a subset of the events, such as only those from a specific system. The filtering is achieved with an SQL-like query language. Figure 11.4 shows the event listing tab for a REAMS instance with some demo log data in it.

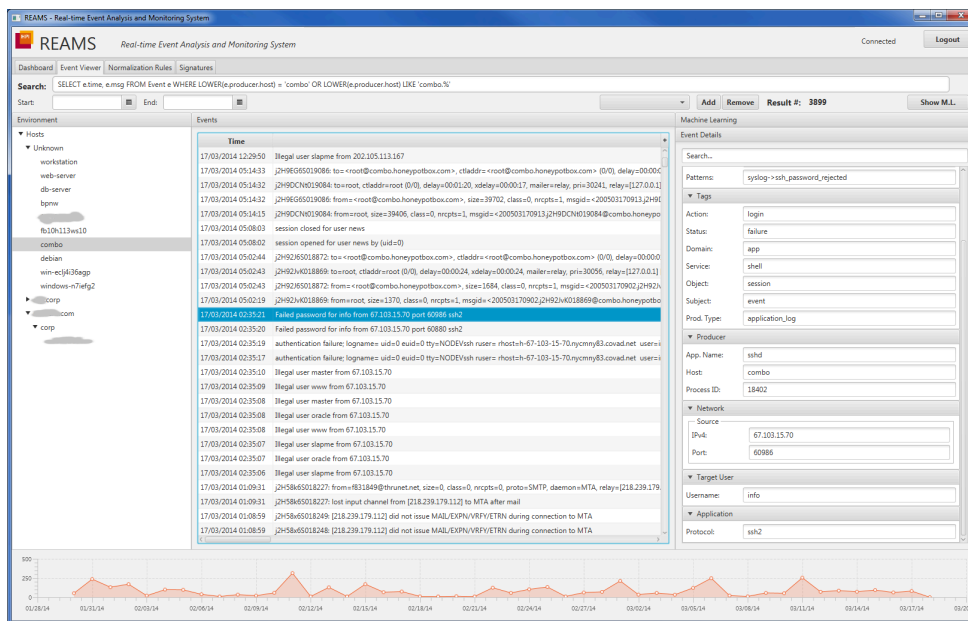


Figure 11.4: Event listing of the REAMS desktop client

At the top, a query for the events to be shown can be entered. On the left, an operator has an overview of all the systems that are covered by all event logs, the middle section lists the time and main message of each event, and the right panel comes with a detailed overview of all set OLF fields of the currently selected event. At the bottom is a histogram that allows an operator to identify time frames of unusually low or high event load. By right-clicking on the query bar, the operator can trigger all available analysis methods on the currently selected events. Figure 11.5 shows the context menu with all supported methods after a mouse click. Over this menu, the method is invoked on the core server by facilitating its REST-interface. The results of the analysis are presented in the GUI as follows.

- **Pattern Detection:** New potential alerts will appear in the dashboard and the

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

event listing.

- **K-Means:** An overview of the calculated clusters will be shown.
- **Anomaly Detection:** A listing of anomalous events will be shown in the event listing.

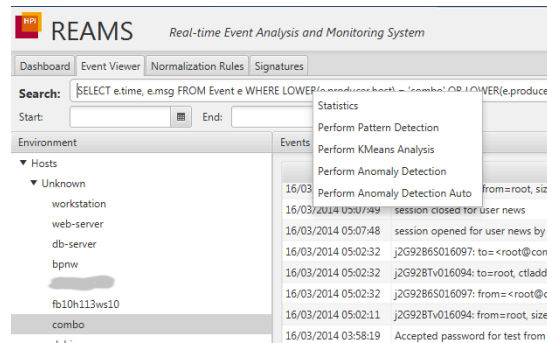


Figure 11.5: Context menu for available analysis methods

11.1.3.2 Web-Based Interface

As an alternative to the desktop client, we are also experimenting with other web-based user interfaces. They have the advantage that no additional software, such as a JVM, is required on the client side. Compared to the desktop version, they are not as powerful, since they do not interact with the core server directly. At the moment, we are supporting the two web-based GUIs Kibana and ITOA.

Kibana Kibana is an open source data visualization tool for the ElasticSearch database and is part of the Elastic Stack of ElasticSearch. In combination with Logstash, it is mainly focused on event monitoring and provides many types of metrics, diagrams, and graphs on top of event data. Although Kibana is tailored to the records produced by Logstash, it can visualize any kind of indexed data in ElasticSearch. This means that, by having our event data persisted in an ElasticSearch instance, Kibana is also able to visualize OLF events without further doing, assuming that the events are correctly indexed. Figure 11.6 shows the dashboard of Kibana with the data of a REAMS core server.

The advantage of Kibana, over the desktop-based client, is the flexibility of an operator to create new visualizations on demand. For example, the data can be represented in a bar chart, line chart, pie chart or in a simple listing. Furthermore, there is the possibility to visualize entities of a specific type in a graph. This could be used to

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

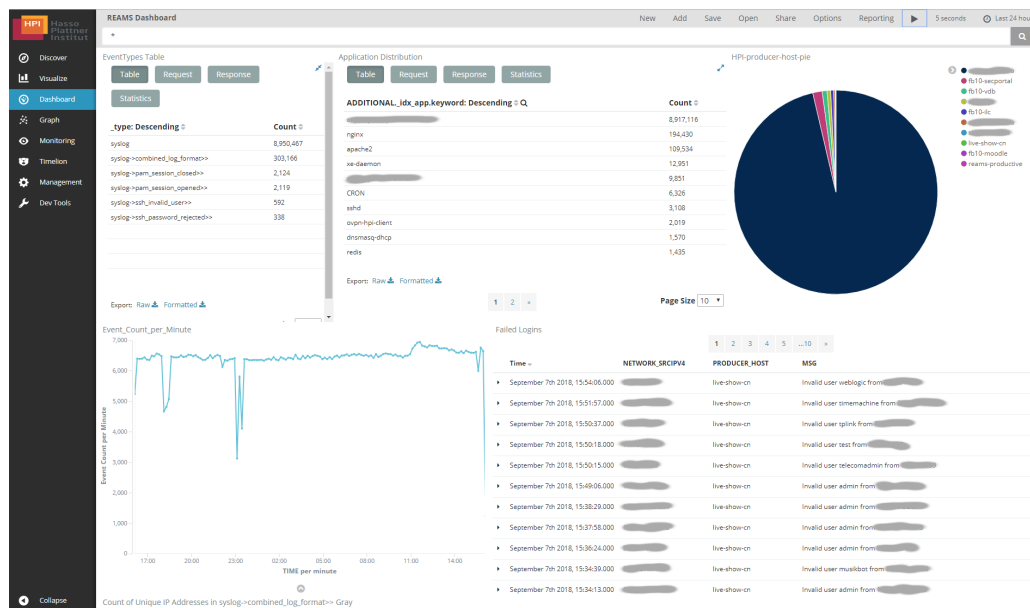


Figure 11.6: Dashboard of the Kibana-based REAMS GUI

visualize attack graphs similarly as the desktop client. A disadvantage of the Kibana interface is the reliance on Elasticsearch, because the persistence into Elasticsearch and the querying on top of its indexed data is not sufficient for high event loads. In our experiments, we could only insert around 20 k evts./s into an index. Also, more complex queries are taking longer to process than in a HANA database. However, this is expected, as Elasticsearch is more of a warm than hot storage.

SAP IT Operations Analytics (ITOA) SAP ITOA is a product for the monitoring of data centers¹ and is mostly running on top of SAP HANA. It mainly consists of an adapter that is collecting and persisting events from various data sources into SAP HANA and an application server that provides a web-based visualization of the event information to the data center operator. The web-based GUI mainly consists of a dashboard, an event listing, and various configuration options for the data collection and normalization. Similarly to the dashboard of Kibana, an operator can create diagrams and metrics based on the event data. In the event list, an operator can filter events by fixing the values of selected fields.

Due to the fact that ITOA is based on the HANA database and the core server writes to a HANA database by default, we can use ITOA for the visualization of OLF events without further modifications to REAMS. Also, ITOA is flexible with the data

¹SAP ITOA - <https://www.sap.com/products/it-operations-analytics.html>

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

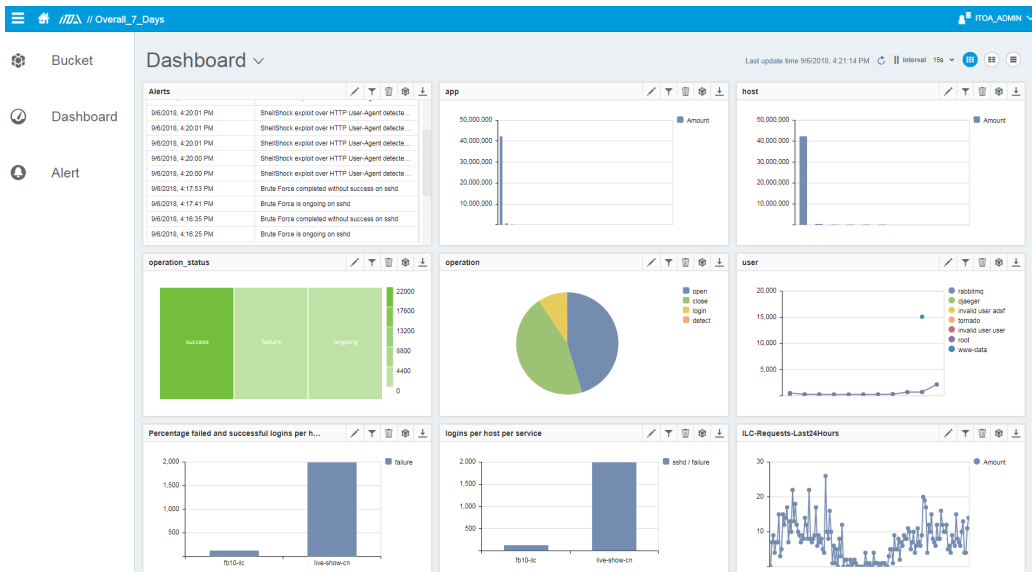


Figure 11.7: Dashboard of the ITOA-based REAMS GUI

model of events and can, therefore, visualize OLF events right away, not needing further configuration. Figure 11.7 shows a dashboard on REAMS events in ITOA. As opposed to Kibana, ITOA can act more fluently on queries, because it is based on the in-memory-based hot storage. Dashboards can be updated more frequently, and the event listing works more smoothly. However, as also ITOA is directly accessing the persistence layer of REAMS, it is not able to trigger operations, such as the loading of log files.

A special feature of ITOA that goes beyond the capabilities of Kibana is the alerting based on previously specified event patterns and actions. In regular time intervals, ITOA checks the event data against these event patterns and then executes the corresponding actions, such as generating an alert event or sending out a notification email to an operator.

11.2 Practical Use-Cases

Using the deployment strategy from above, we have used multiple REAMS instances in real-world scenarios. Foremost, we have set up an instance in our lab and are monitoring many internal servers with it. In addition, we are using the analytical capabilities of REAMS for analyzing the event data of two big international companies.

11.2.1 HPI Infrastructure

We have a setup of REAMS in our team’s lab environment that is monitoring 44 of our servers, as depicted in Figure 11.8. All of those 44 servers are equipped with a Syslog client that connects via SSL to a central Syslog server in our network. Further, from this Syslog server, the events are forwarded to three separate REAMS deployments. The primary deployment is called *REAMS_PROD* and uses an SAP HANA database for event persistence. The access to the instance is performed through the core server with the custom-developed desktop client. The second deployment, named *REAMS_ITOA*, also writes into an SAP HANA database with a slightly adjusted data schema in comparison to *REAMS_PROD*. It is accessed through the SAP ITOA user interface that operates on the database. *REAMS_ES* is the third deployment, which persists all events into an ElasticSearch instance. The view on top of the ElasticSearch data is provided by either the Kibana or Grafana web service.

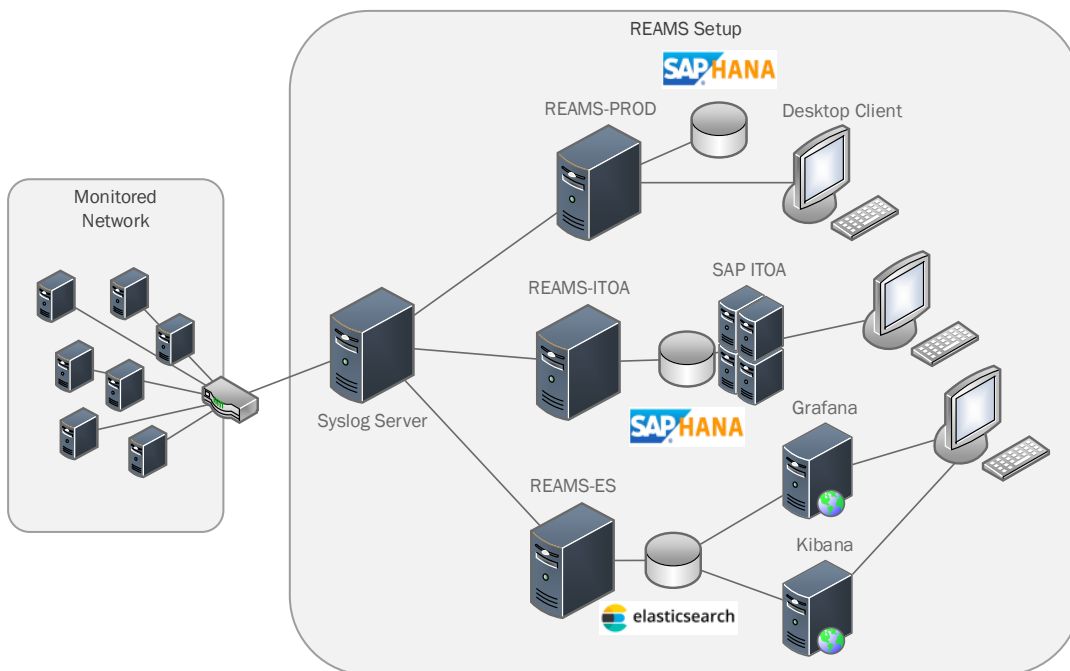


Figure 11.8: Setup of REAMS in our infrastructure

The *REAMS_PROD* instance is the most long-running instance of the three deployments. It has been running since September 2014 and analyzed over 2 billion events in our production environment. Foremost, we use it to identify problems in our infrastructure and services and are automatically detecting selected attacks during live operations. As one example, we are running the multi-step signature detection with two signatures, namely for login brute-force and ShellShock-based attacks. The

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

detected attacks are shown on the dashboard of our desktop client, where they are also visualized in the attack graph. Figure 11.9 shows an attack graph with the two attack types on three of our productive systems. We have started these attacks ourselves from two different servers. One server is located within our environment and another one is an Internet service that tests a domain for the ShellShock vulnerability.

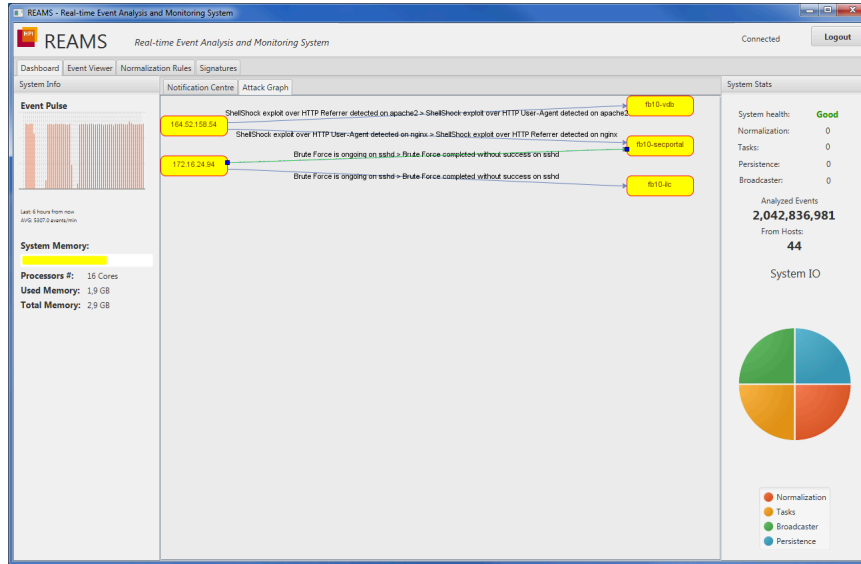


Figure 11.9: Detecting attacks in our environment

The advantage of the attack graph is a quick visual overview of critical attack activities in the environment and its involved systems. The coloring, being red or green, of the graph edges indicates whether the conducted attack has failed or was successful. In addition, if a new type of attack is started from an already compromised system, the attack graph will extend to a multi-stepped graph, so that the progress of the attacker in the environment becomes apparent.

Almost on a daily basis, there are login brute-force attacks on our SSH and web servers. In particular, we have identified numerous login brute-force attacks on our servers located in China, which are most likely conducted from an SSH-based botnet. Also, we can see regular failed brute-force attacks on admin panels of our web servers.

11.2.2 Event Data of Big International Companies

Even though the HPI deployment allows us to test REAMS in a real-world scenario, the monitored environment is not comparable to a large enterprise and can therefore not be considered as a Big Data challenge. However, we have projects with multiple big international companies, having more than 50 000 employees, in which we

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

analyze their log events. In contrast to our HPI deployment, these companies are producing many security-relevant events that are usable for attack analysis. Their stream of events is amounting to multiple billion events per day. With these properties, these events are very interesting for testing our approaches.

The composition of the events we are getting from the companies is very diverse. There are many different formats of various data sources. Even for similar types of events, such as proxy events, there are variations in the formats between vendors and products. As the most popular event types, there are firewall, proxy, DNS, DHCP, VPN, router, and domain controller logs.

Since REAMS is a prototypical solution, it is not directly placed into the production environment of the monitored enterprises. Many companies already have some solutions for collecting events at a central place and therefore cannot easily redirect their stream of events to our deployment. As an alternative to stream-processing, we are receiving exports of the raw event data from the existing event management systems, which we then import into our REAMS deployment. The rest of the event processing is similar to the already described event processing workflow. This means, we are normalizing the events into OLF objects and are persisting them either into a HANA database or are exporting them into a structured file, such as CSV or Parquet, where it can then be analyzed either by REAMS' analytics or by external software.

Based on the normalized event data, we could evaluate our analytical approaches for signature- and anomaly-based detection. We have experimented with attack signatures derived from CTI, such as Snort signatures and lists of IP indicators, as well as with machine learning approaches for the identification of malicious user behavior. As a result of our experiments with the provided enterprise data, our team was able to identify a number of potential security incidents, which could eventually be reported to the security operations centers (SOCs) of respective companies. These incidents included massive amounts of login attempts or the beaconing to command & control servers.

11.3 Conclusion

In this chapter, we have described the deployment and use of our prototypical Big Data SIEM REAMS in practice. This should show that our proposed event processing workflow is not just a theoretical construct but has a real implementation that handles large amounts of event data for security monitoring and can bring benefit to the operators of enterprise networks.

A deployment of REAMS consists of three components, the *agents* that collect the data from the sources, a *core server* that gathers the data from log servers or agents at a central place and manages the normalized events, and three user interfaces that visualize the data of the core server for a security operator. Concerning the user

CHAPTER 11. REAL-TIME EVENT ANALYSIS AND MONITORING SYSTEM (REAMS)

interfaces, we have shown that the core server is flexible enough to support even third-party interfaces, such as SAP ITOA or Kibana.

We have used REAMS in its current form within our infrastructure to monitor our servers for malfunctions and malicious activities. On top of that, we have used REAMS to normalize and analyze huge amounts of events from our project partners and could help them to identify further security incidents in their environment.

In the end, we have shown that REAMS is already mature enough to identify security incidents in practice and can even operate with a better performance than already existing SIEM solutions.

Chapter 12

Identity Leak Checker (ILC)

We have elaborated on the relevance of leaked identity data for enterprise and individual user security in Chapter 10. As credential misuse can be considered as one of the main attack vectors leading to data breaches [73], the knowledge about leaked identities in the form of CTI can establish a significant advantage over an attacker. With REAMS, we have already presented a prototypical SIEM that incorporates CTI into the security monitoring process of an enterprise. As a next step, we are presenting a platform called Identity Leak Checker (ILC) that provides such CTI on leaked identities. The platform implements the leak processing workflow from Section 10.2 and acts as a foundation for research on the topic of identity leaks and password security as well as an awareness and notification platform for victims of identity theft. In the following, we are introducing the deployment of the ILC and then follow with an overview of its current use cases.

12.1 Real-World Deployment

The Identity Leak Checker (ILC) is a platform that consists of three major components, the backend, a web interface, and a dedicated client.

12.1.1 Backend

The backend is the core of the platform. It is mainly responsible for processing leaks according to the leak processing workflow. Foremost, this processing includes the automated collection of smaller leaks, the normalization of leaks into our common data model for leaked identities, and the population of the database for the web service and native client. On top of the normalized data, the backend also creates multiple statistical measures, such as a list of top passwords, the number of leaked credentials per leak, and the email domains with the highest number of affected identities. The

CHAPTER 12. Identity Leak Checker (ILC)

statistical results are written into the database, too, so that they can be shown to the platform users.

The platform itself is composed of multiple Java program modules that can be executed independently by each other, depending on the implemented leak assessment strategy. Each module is responsible for a subset of the leak processing workflow.

12.1.2 Web Interface

The web interface is the primary end-user interface to the ILC. It is based on the data that was collected and processed in the backend before. The main functionality of the service is a query interface for leaked identities. A user can check whether their personal information is included in a known identity leak by entering their email address into a search mask. Subsequently, the user receives an email with the query result to his previously entered email address. The main page with the address input is shown in Figure 12.1. As a secondary functionality to the email address check, interested users can view an overview of statistics on passwords and leaks.

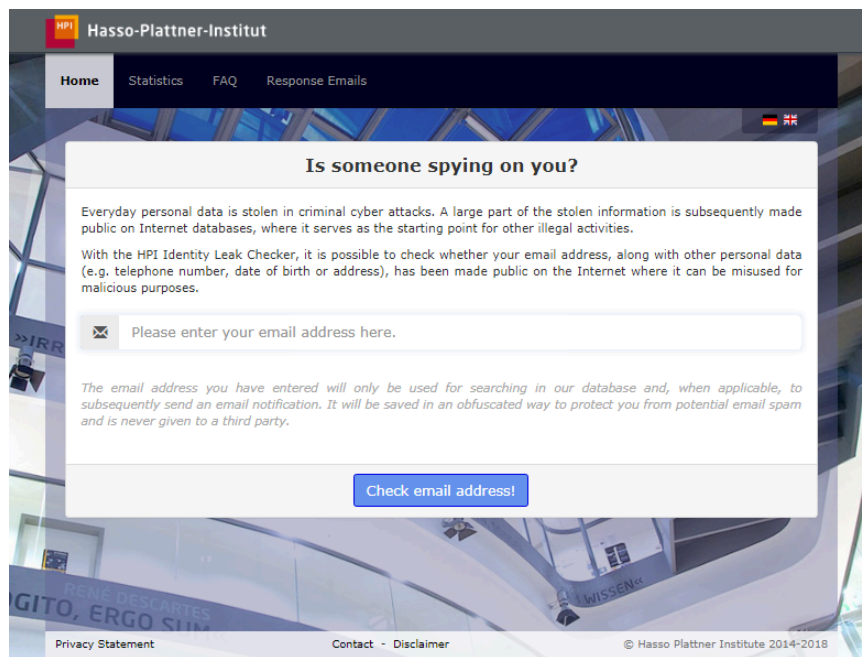


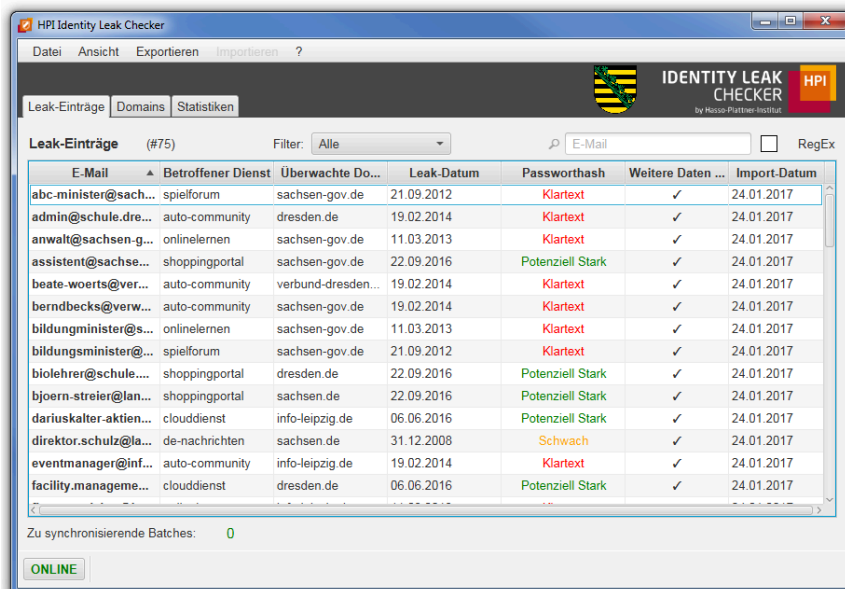
Figure 12.1: Main query interface of the ILC web service

The web application for the website is deployed on a Java application server which delivers its web pages through JSP. The web application itself is connected to a database that keeps reduced and obfuscated records of the leaked credentials, so that a breach of the database cannot lead to further misuse.

CHAPTER 12. Identity Leak Checker (ILC)

12.1.3 Client

The client is a more feature-rich interface to the ILC and is meant for companies and institutions that want to monitor their email addresses and domains for breaches. It runs locally on the user's computer as a Java FX application but retrieves its data over a dedicated server component on the central ILC server. The retrieved records of affected users are in turn stored in a local database on the user's computer or a server in control of the user, so that the information can be used as threat intelligence in a SIEM, such as REAMS. Figure 12.2 shows the GUI of the ILC with some demo data. This data was initially downloaded from the ILC server to the local database and then requested from the local database server for viewing.



The screenshot shows the HPI Identity Leak Checker application window. The title bar reads "HPI Identity Leak Checker". The menu bar includes "Datei", "Ansicht", "Exportieren", and "Importieren...?". The main interface has tabs for "Leak-Einträge", "Domains", and "Statistiken". Below the tabs, there is a filter dropdown set to "Alle" and a search field for "E-Mail" with a "RegEx" checkbox. The main area displays a table with the following columns: "E-Mail", "Betroffener Dienst", "Überwachte Do...", "Leak-Datum", "Passwortshash", "Weitere Daten ...", and "Import-Datum". The table contains 15 rows of data, including email addresses like "abc-minister@sach...", "admin@schule.dre...", and "anwalt@sachsen-g...". The password hashes are color-coded: "Klartext" (red), "Potenziell Stark" (green), and "Schwach" (orange). A status bar at the bottom shows "Zu synchronisierende Batches: 0" and an "ONLINE" button.

E-Mail	Betroffener Dienst	Überwachte Do...	Leak-Datum	Passwortshash	Weitere Daten ...	Import-Datum
abc-minister@sach...	spielforum	sachsen-gov.de	21.09.2012	Klartext	✓	24.01.2017
admin@schule.dre...	auto-community	dresden.de	19.02.2014	Klartext	✓	24.01.2017
anwalt@sachsen-g...	onlinelernen	sachsen-gov.de	11.03.2013	Klartext	✓	24.01.2017
assistent@sachse...	shoppingportal	sachsen-gov.de	22.09.2016	Potenziell Stark	✓	24.01.2017
beate-woerts@ver...	auto-community	verbund-dresden...	19.02.2014	Klartext	✓	24.01.2017
berndbecks@verw...	auto-community	sachsen-gov.de	19.02.2014	Klartext	✓	24.01.2017
bildungsminister@s...	onlinelernen	sachsen-gov.de	11.03.2013	Klartext	✓	24.01.2017
bildungsminister@...	spielforum	sachsen-gov.de	21.09.2012	Klartext	✓	24.01.2017
biolehrer@schule....	shoppingportal	dresden.de	22.09.2016	Potenziell Stark	✓	24.01.2017
bjoern-streier@lan...	shoppingportal	sachsen.de	22.09.2016	Potenziell Stark	✓	24.01.2017
dariuskalter-aktien...	clouddienst	info-leipzig.de	06.06.2016	Potenziell Stark	✓	24.01.2017
direktor.schulz@ja...	de-nachrichten	sachsen.de	31.12.2008	Schwach	✓	24.01.2017
eventmanager@inf...	auto-community	info-leipzig.de	19.02.2014	Klartext	✓	24.01.2017
facility.manageme...	clouddienst	dresden.de	06.06.2016	Potenziell Stark	✓	24.01.2017

Figure 12.2: Native client interface for the ILC with demo data

12.2 Practical Use-Cases

After we have described the deployment of the ILC with its three components, we now show how these components are helping victims of identity theft in practice. A productive setup of the ILC is running since early 2014 and is maintained and used daily since then.

12.2.1 Identity Leak Checker Service

The Identity Leak Checker (ILC) web service is the most visible part of the platform and has the goal to raise security awareness in public. The service is hosted on the Internet¹ under the project overview of the HPI security group. Since the start of the service, we have experienced a large amount of interest from users around the world.

Leak Statistics The most essential part of the ILC is the availability of as many leaks as possible, so that many victims can be informed about the theft of their data. While the amount of data was rather small at the start of the service, the numbers have almost exploded in the past two years. Initially, we have started the service with the infamous Adobe leak from October 2013 [152]. Only this leak alone affected as many as ≈ 150 M identities. In the two years following this leak, the rate at which new major leaks appeared, slightly increased. The highest amount of data was published at the end of 2016, because there were major breaches of Dropbox, LinkedIn, and MySpace. Today, we are experiencing larger leaks almost every week, which led to a steady increase in identities over the past two years. Figure 12.3 shows a graph with the number of identities available in the ILC from the start in early 2014. The maximum of leaked identities is at ≈ 5.9 G entries which are extracted from ≈ 800 leaks. Altogether, there are ≈ 2.35 G distinct email addresses in all leaks, meaning that already a significant fraction of the Internet users worldwide is affected.

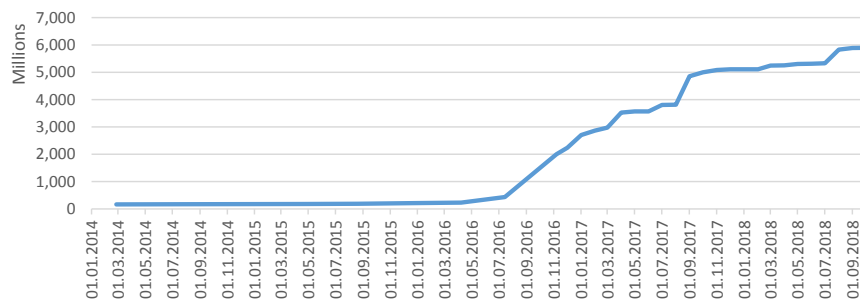


Figure 12.3: Number of identities available in the ILC service

While there are a few other services that are providing similar services as the ILC, such as *Have I Been Pwned*² or *BreachAlarm*³, our service is distinguishable from these services by the type of data available for querying. First of all, we are ensuring the exactness and quality of the data by manually parsing leaks. In comparison to other services, we are interpreting the leak as a whole and are not just extracting

¹HPI Identity Leak Checker (ILC) - <https://sec.hpi.de/ilc/>

²Have I Been Pwned - <https://haveibeenpwned.com>

³BreachAlarm - <https://breachalarm.com/>

CHAPTER 12. Identity Leak Checker (ILC)

email addresses. As a result, we can tell which kind of data has been leaked for an individual. Second, we are focusing on leaks that are affecting users in Europe, and particularly Germany, rather than on users in the USA.

User Statistics Due to the focus on the European and German language region, we have received much attention for our service. Since the start in April 2014, we have served ≈ 7.5 M email address queries, of which ≈ 1.4 M people are affected by at least one leak. Even though the cumulative average of affected users is at around 18.66 %, we are even achieving a positive result rate of ≈ 40 % on the current dataset in the ILC.

On an average day, the service receives ≈ 1000 requests. Interestingly, the number of requests is highly dependent on the reports of data breaches in the mainstream media. For example, during the report of a major leak in the German language region, we have received up to 250 k requests per day.

Press Attention Due to regular announcements referring to our service, the importance of the security awareness topic, and the focus on the German language region, our service has gained some media attention in Germany. The service itself and the results of our password analyzes have been mentioned in some major German news, such as *Spiegel*, *Stern*, *Süddeutsche Zeitung* for printed media and *ARD Tagesschau*, and *RBB Abendschau* in the television program. Each report on these channels has made many new users aware of our service, so that we are able to reach a considerable amount of people whenever a new leak appears.

The regular news reports have also made us one of the common contact points for information on leaks and password security. We have been contacted by investigating authorities in Germany for adding new datasets to the service [153], so that as many users as possible can be warned in a relatively short time.

12.2.2 Monitoring for Federal States and Companies

While the ILC focuses on individual users, we also want to provide a mechanism for an enterprise to check all users in their domain. Many company representatives have contacted us to support them in the lookup of all their email addresses. In theory, a security operator could issue a request for each company email address. However, while this would be feasible for smaller companies with a few employees, this becomes a tedious task for large companies with thousands of employees. The client is a solution for such use case, because a security operator can monitor all email addresses in the company's domain and is then able to see the request result in the GUI. He can also extract those from the local database, in the sense of threat intelligence.

CHAPTER 12. Identity Leak Checker (ILC)

At the moment, we are working together with the State of Saxony (Germany). We are monitoring all the email addresses that are in their scope of responsibility, covering more than 2000 email domains. The results in the client are available to the Computer Emergency Response Team (CERT) of the state, so that immediate threats to the state's IT environment can be detected and prevented early on. As an additional feature to the listing of affected email addresses, the client also visualizes a threat level for the state in the form of a pie diagram. The diagram shows the fraction of identities that have their cleartext password or a weak password hash leaked, making their accounts easily compromisable in case of password reuse.

12.3 Conclusion

In this chapter, we have presented a productive deployment of the ILC, a platform that informs individuals as well as enterprises about potential account compromises and identity theft. The platform is reachable via two interfaces, i.e., the web interface, meant for individuals to check their email address for data theft and to inform themselves about proper password use, and the client, meant for enterprise users to gain an overview of all their affected email addresses and to derive a threat level for their IT environment.

Over the course of multiple years, the ILC web service has been established as one of the main contact points for individuals, i.e., private consumers and company employees, with regards to account and password security. Our service has reached over 7 M users in Germany and still reaches around 1000 users on an average day. We see the web service as the most important component of the ILC, because it reaches people in different domains, in their private life or as an employee of a company, and directly raises awareness to potential account compromises. Also, by warning people individually, we can immediately reduce the attack surface in enterprises since employees are changing potentially weak credentials on their own. The client is another tool that helps to integrate the knowledge on account compromises directly into the security analysis process of a company. Together with the State of Saxony in Germany, we have shown that even large networks with thousands of users can be monitored for account compromises with a single tool. The results are available to the state's CERT, which informs affected individuals or institutions and coordinates protective measures. For now, we are also working on the expansion of the client to make the leak information readily available to companies and create ways to integrate into their existing security landscape, such as their SIEM systems.

Chapter 13

Conclusion

The security monitoring of computer networks has become a challenging task, as networks are growing in size and attacks are getting more complex and sophisticated. In the security community as well as in the SOCs of companies, log events are becoming the most valuable source for tracing malicious activities and identifying attackers in the own network. Already, in large networks, there are thousands of sensors and devices that are logging each activity and event that is occurring. However, in the past, all these events were lying dormant at the location where they have been produced. Only recently, software that is collecting these events at a central place has emerged, the so-called SIEM. Still, these systems are not able to keep up with the volume, heterogeneity, and velocity of the incoming events. In other words, the processing of events is apparently a typical Big Data problem that impedes an efficient security analysis in real-time.

In this thesis, we have addressed the problem of processing huge amounts of heterogeneous event data by applying the paradigms of Big Data processing. We are proposing an event processing workflow that normalizes, persists and analyzes log events in near real-time and incorporates external CTI for more efficient identification of known threats. As events from various network nodes and further external threat intelligence is now available within a single system, activities and knowledge can be correlated for the whole network, so that complex attacks, which are mostly spanning over multiple nodes in the network, can be detected. Our main contributions are summarized in the following points.

- An event processing workflow was proposed and implemented that relies on popular Big Data paradigms like multi-processing, immutability, in-memory storage, and distribution to handle the huge amounts of event data. By combining the paradigms, we can prepare events for a following real-time analysis with ≈ 280 k evts./s, which is sufficient for monitoring even large enterprise networks.

CHAPTER 13. CONCLUSION

- Multiple complex analysis approaches were proposed and implemented that are working solely on log events and across application and host boundaries. Our focus was the proposal of a single- and multi-step signature engine as well as to enable the execution of machine learning algorithms, which are originally known only from traditional IDS systems that are limited to a single machine or application.
- External intelligence, in the form of CTI and OSINT from existing security solutions and threat sharing platforms, has been integrated into the workflow. This intelligence is used for deriving signatures, so that an operator of the system can detect known attacks without further security knowledge.
- The ecosystem and economy of identity leaks have been researched and a processing workflow that collects, normalizes, and persists information on leaked digital identities has been proposed and implemented. The knowledge about these compromised accounts is an essential data source for CTI, as credential misuse is one of the main attack surfaces in enterprises. Altogether, we have assessed leak information on almost half of the entire Internet community.
- The approaches proposed in this thesis have been implemented and integrated into our prototypical SIEM platform, named REAMS. This SIEM is used to analyze the log event of our IT infrastructure and that of our project partners. In addition, we have created a leak warning service, called ILC, which informs individuals as well as domain owners about potentially compromised accounts.

Altogether, our approaches show that security monitoring of large enterprise networks in near real-time is indeed possible, even though we are dealing with a Big Data problem. We are representing all events in the same common event format, so that events from different systems and applications can be correlated and put into relation with each other. This enables us to trace the path of an attacker over multiple machines and over a longer time span, as it is typical for complex multi-step attacks. Further hints about potential attack indicators are brought in by external intelligence, such as information on compromised accounts of the ILC or as signatures from known open-source IDS systems. Although the proposed REAMS has solved many challenges that are apparent in existing SIEMs, there are still some challenges left that we face when using our system in productive environments.

One major point is the availability of event logs from all parts of the network. In fact, many companies are just starting with the collection of logs from all their systems, because there was no prior support in processing them. So far, the main focus is on logs of the most critical systems, such as servers or network infrastructure. However, to also follow the steps of an attacker on the final target, we also need logs from workstations and ideally even mobile devices. Only by having a complete

CHAPTER 13. CONCLUSION

overview of performed activities, one can comprehend what the attacker has done on the network.

Another challenge of event analysis is the ongoing demand for data privacy on the user side. Especially with the introduction of the General Data Protection Regulation (GDPR), enterprises are obliged to implement privacy-preserving measures for the collected log events, such as the stripping of the host part from IP addresses or the pseudonymization of usernames. Since the reduction of information leads to an intended obfuscation of an individuals' activities, some of the approaches of security monitoring have to be adapted to this new situation of information scarcity.

In addition to the presented limitations of the approach, we are considering the following questions for future work on the topic of advanced event analytics.

- For now, we have foremost focused on misuse detection algorithms and also showed that anomaly detection approaches are applicable. Both detection approaches are known from IDSs. As neural networks are currently on the rise and are promising to derive knowledge from large datasets, it would be interesting to also apply these to event data. However, as the algorithms are still supervised and there is only limited labeled data, either new labeled data has to be created or further research on unsupervised algorithms must be conducted.
- Many enterprises already have a distributed processing framework, such as Spark, deployed in their environment. Therefore, it makes sense to make use of these resources and do the analytical tasks of REAMS in a Spark environment. The challenging part is to integrate and orchestrate the tasks all within one environment.
- The signature detection is now mostly derived from existing CTI. As signatures are an image of a known attack, it is desirable to automatically derive signatures from findings of machine learning approaches.
- There are many threat sharing platforms on the market, but they are still limited in the complexity of the indicators they provide. Also, their data quality is questionable. Further research can be put into the improvement of data quality and the sharing of more signatures or generally attack patterns. One focus could be multi-step signatures, as there is limited support by the existing standards.

Bibliography

- [1] Viktor Mayer-Schönberger and Kenneth Cukier. *Big Data: A Revolution That Will Transform How We Live, Work and Think*. John Murray, Oct. 2013. ISBN: 978-1848547926.
- [2] MunichRe. *Cyber Risk Research: HSB's 2016 Survey*. May 2016. URL: <https://www.munichre.com/HSB/cyber-research-2016/index.html> (visited on 12/05/2018).
- [3] Information is Beautiful. *World's Biggest Data Breaches*. Information is Beautiful. Oct. 2015. URL: <http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/> (visited on 12/05/2018).
- [4] Seena Gressin. *The Equifax Data Breach: What to Do*. Federal Trade Commission. Sept. 2017. URL: <https://www.consumer.ftc.gov/blog/2017/09/equifax-data-breach-what-to-do> (visited on 12/05/2018).
- [5] David E. Sanger and Julie Hirschfeld Davis. *Hacking Linked to China Exposes Millions of U.S. Workers*. New York Times. June 2015. URL: <http://www.nytimes.com/2015/06/05/us/breach-in-a-federal-computer-system-exposes-personnel-data.html> (visited on 12/05/2018).
- [6] Matthew Goldstein, Nicole Perlroth, and David E. Sanger. *Hackers' Attack Cracked 10 Financial Firms in Major Assault*. New York Times. Oct. 2014. URL: <http://dealbook.nytimes.com/2014/10/03/hackers-attack-cracked-10-banks-in-major-assault/> (visited on 12/05/2018).
- [7] Richard Bejtlich. "Understanding the Advanced Persistent Threat". In: *Information Security* (Aug. 2010). URL: <http://searchsecurity.techtarget.com/magazineContent/Understanding-the-advanced-persistent-threat>.

BIBLIOGRAPHY

- [8] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. “Advanced Persistent Threats: Behind the Scenes”. In: *Proceedings of the 50th Annual Conference on Information Sciences and Systems (CISS'16)*. Princeton, NJ, USA: IEEE, Mar. 2016, pp. 181–186. DOI: 10.1109/CISS.2016.7460498.
- [9] Dan Mcwhorter. *APT1 - Exposing One of China's Cyber Espionage Units*. Tech. rep. Mandiant, Feb. 2013.
- [10] Claudio Guarnieri. *Digital Attack on German Parliament: Investigative Report on the Hack of the Left Party Infrastructure in Bundestag*. June 2015. URL: <https://netzpolitik.org/2015/digital-attack-on-german-parliament-investigative-report-on-the-hack-of-the-left-party-infrastructure-in-bundestag/> (visited on 12/05/2018).
- [11] Ponemon Institute. *2016 Cost of Data Breach Study: Global Analysis*. Tech. rep. Ponemon Institute, July 2018.
- [12] Jim Finkle and Ron Grover. *Sony hires Mandiant after cyber attack, FBI starts probe*. Reuters. Dec. 2014. URL: <http://www.reuters.com/article/us-sony-cybersecurity-mandiant-idUSKCN0JE0YA20141201> (visited on 12/05/2018).
- [13] Brian Krebs. *Discount Chain Fred's Inc. Probes Card Breach*. June 2015. URL: <http://krebsonsecurity.com/2015/06/discount-chain-freds-inc-probes-card-breach/> (visited on 12/05/2018).
- [14] Big Data Working Group. *Big Data Analytics for Security Intelligence*. Tech. rep. Cloud Security Alliance, Sept. 2013.
- [15] Kelly M. Kavanagh and Toby Bussa. *Magic Quadrant for Security Information and Event Management*. Tech. rep. Gartner, Dec. 2017.
- [16] Judith Hurwitz, Alan Nugent, Fern Halper, and Marcia Kaufman. *Big Data for Dummies*. Wiley, 2013. ISBN: 978-1-118-50422-2.
- [17] Bernard Marr. *Big Data: Using Smart Big Data Analytics and Metrics to Make Better Decisions and Improve Performance*. Wiley, 2015. ISBN: 978-1-118-96583-2.
- [18] Alvaro A. Cardenas, Pratyusa K. Manadhata, and Sreeranga P. Rajan. “Big Data Analytics for Security”. In: *IEEE Security & Privacy* 11.6 (2013), pp. 74–76. ISSN: 1540-7993. DOI: 10.1109/MSP.2013.138.
- [19] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 978-0201727890.

BIBLIOGRAPHY

- [20] David C. Luckham. *Event Processing for Business: Organizing the Real-Time Enterprise*. John Wiley & Sons, Inc, 2013. ISBN: 978-0470534854.
- [21] Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Pushing the Limits in Event Normalisation to Improve Attack Detection in IDS/SIEM Systems”. In: *Proceedings of the First International Conference on Advanced Cloud and Big Data (CBD’13)*. Nanjing, China: IEEE, Dec. 2013, pp. 69–76. ISBN: 978-1-4799-3260-3. DOI: 10.1109/CBD.2013.27.
- [22] Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Event Normalization Through Dynamic Log Format Detection”. In: *ZTE Communications* 12.3 (2014), pp. 62–66.
- [23] Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “A New Approach to Building a Multi-Tier Direct Access Knowledge Base For IDS/SIEM Systems”. In: *Proceedings of the 11th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC’13)*. Chengdu, China: IEEE, Dec. 2013, pp. 118–123. ISBN: 978-1-4799-3380-8. DOI: 10.1109/DASC.2013.48.
- [24] David Jaeger, Andrey Sapegin, Martin Ussath, Feng Cheng, and Christoph Meinel. “Parallel and Distributed Normalization of Security Events for Instant Attack Analysis”. In: *Proceedings of the 34th IEEE International Performance Computing and Communications Conference (IPCCC’15)*. Nanjing, China: IEEE, Dec. 2015. DOI: 10.1109/PCCC.2015.7410270.
- [25] David Jaeger, Feng Cheng, and Christoph Meinel. “Accelerating Event-Based Attack Detection with a Distributed In-Memory Platform”. In: *Proceedings of the 16th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC’18)*. Aug. 2018, pp. 634–643. DOI: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00114.
- [26] Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, and Christoph Meinel. “Hierarchical Object Log Format for Normalisation of Security Events”. In: *Proceedings of the 9th International Conference on Information Assurance and Security (IAS’13)*. Yasmine Hammamet, Tunisia: IEEE, Dec. 2013, pp. 25–30. ISBN: 978-1-4799-2989-4. DOI: 10.1109/ISIAS.2013.6947748.
- [27] Andrey Sapegin, David Jaeger, Amir Azodi, Marian Gawron, Feng Cheng, and Christoph Meinel. “Normalisation of Log Messages for Intrusion Detection”. In: *Journal of Information Assurance and Security* 9.3 (Sept. 2014), pp. 167–176. ISSN: 1554-1010.

BIBLIOGRAPHY

- [28] David Jaeger, Amir Azodi, Feng Cheng, and Christoph Meinel. “Normalizing Security Events with a Hierarchical Knowledge Base”. In: *Proceedings of the 9th International Conference on Information Security Theory and Practice (WISTP’15)*. Ed. by Raja Akram and Sushil Jajodia. Vol. 9311. Lecture Notes in Computer Science 1. Springer International Publishing, 2015, pp. 238–248. ISBN: 978-3-319-24017-6. DOI: 10.1007/978-3-319-24018-3.
- [29] Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Runtime Updatable and Dynamic Event Processing using Embedded ECMAScript Engines”. In: *Proceedings of the 4th International Conference on IT Convergence and Security (ICITCS’14)*. Beijing, China: IEEE, Oct. 2014. ISBN: 978-1-47996-542-7. DOI: 10.1109/ICITCS.2014.7021808.
- [30] David Jaeger, Feng Cheng, and Christoph Meinel. “Enriching Normalized Security Event Logs for Deeper Security Analytics”. 2018.
- [31] Jerry Shenk. *Ninth Log Management Survey Report*. Tech. rep. SANS Institute, Oct. 2014.
- [32] Amir Azodi, David Jaeger, Feng Cheng, and Christoph Meinel. “Passive Network Monitoring using REAMS”. In: *Proceedings of the International Conference on Information Science and Applications (ICISA’15)*. Ed. by Kuinam J. Kim. Vol. 339. Lecture Notes in Electrical Engineering. Pattaya, Thailand: Springer Berlin Heidelberg, Feb. 2015, pp. 205–215. ISBN: 978-3-662-46577-6. DOI: 10.1007/978-3-662-46578-3_24.
- [33] 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)*. Vol. New York, NY, USA. IEEE Computer Society, Nov. 2015, pp. 390–398. DOI: 10.1109/CSCloud.2015.26.
- [34] David Jaeger, Hendrik Graupner, Andrey Sapegin, Feng Cheng, and Christoph Meinel. “Gathering and Analyzing Identity Leaks for Security Awareness”. In: *Proceedings of the 7th International Conference on Passwords (PASSWORDS’14)*. Vol. 9393. Lecture Notes in Computer Science (LNCS). Trondheim, Norway: Springer International Publishing, Dec. 2014, pp. 102–115. ISBN: 978-3-319-24191-3. DOI: 10.1007/978-3-319-24192-0_7.
- [35] Hendrik Graupner, David Jaeger, Feng Cheng, and Christoph Meinel. “Automated Parsing and Interpretation of Identity Leaks”. In: *Proceedings of the 13th Computing Frontiers Conference 2016 (CF’16)*. Como, Italy: ACM, May 2016, pp. 127–134. ISBN: 978-1-4503-4128-8. DOI: 10.1145/2903150.2903156.

BIBLIOGRAPHY

- [36] David Jaeger, Hendrik Graupner, Chris Pelchen, Feng Cheng, and Christoph Meinel. “Fast Automated Processing and Evaluation of Identity Leaks”. In: *International Journal of Parallel Programming (IJPP)* 44.2 (Dec. 2016), pp. 441–470. ISSN: 1573-7640. DOI: 10.1007/s10766-016-0478-6.
- [37] David Jaeger, Chris Pelchen, Hendrik Graupner, Feng Cheng, and Christoph Meinel. “Analysis of Publicly Leaked Credentials and the Long Story of Password (Re-)use”. In: *Proceedings of the 11th International Conference on Passwords (PASSWORDS’16)*. Lecture Notes in Computer Science. Springer, 2016.
- [38] McKinsey. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. Tech. rep. McKinsey Global Institute, June 2011.
- [39] John Gantz and David Reinsel. *Extracting Value from Chaos*. Tech. rep. IDC iView, June 2011.
- [40] Han Hu, Yonggang Wen, Tat-Seng Chua, and Xuelong Li. “Toward Scalable Systems for Big Data Analytics: A Technology Tutorial”. In: *IEEE Access* 2 (June 2014), pp. 652–687. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2014.2332453.
- [41] Daniel C. Weeks and Tom Gianos. “Netflix: Petabytes Scale Analytics Infrastructure in the Cloud”. In: *QCon*. 2016. URL: <https://www.infoq.com/presentations/netflix-big-data-infrastructure> (visited on 12/05/2018).
- [42] Pamela Vagata and Kevin Wilfong. *Scaling the Facebook Data Warehouse to 300 PB*. Facebook. Apr. 2014. URL: <https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/> (visited on 12/05/2018).
- [43] Randall Munroe. *What if?: Google’s Datacenters on Punch Cards*. Oct. 2013. URL: <http://what-if.xkcd.com/63/> (visited on 12/05/2018).
- [44] Container Eyes. *In 2013 the Amount of Data Generated Worldwide Will Reach Four Zettabytes*. June 2013. URL: <https://vsatglobalseriesblog.wordpress.com/2013/06/21/in-2013-the-amount-of-data-generated-worldwide-will-reach-four-zettabytes/> (visited on 12/05/2018).
- [45] Åse Dragland. “Big Data, For Better or Worse: 90% of World’s Data Generated Over Last Two Years”. In: *ScienceDaily* (May 2013). URL: <https://www.sciencedaily.com/releases/2013/05/130522085217.htm> (visited on 12/05/2018).

BIBLIOGRAPHY

- [46] Ralph Jacobson. *2.5 quintillion bytes of data created every day. How does CPG & Retail manage it?* IBM. Apr. 2013. URL: <https://www.ibm.com/blogs/insights-on-business/consumer-products/2-5-quintillion-bytes-of-data-created-every-day-how-does-cpg-retail-manage-it/> (visited on 12/05/2018).
- [47] Jeff Desjardins. *What Happens in an Internet Minute in 2018?* Visual Capitalist. May 2018. URL: <http://www.visualcapitalist.com/internet-minute-2018/> (visited on 11/15/2018).
- [48] IBM. *The Four V's of Big Data*. 2014. URL: <http://www.ibmbigdatahub.com/infographic/four-vs-big-data> (visited on 12/05/2018).
- [49] David P. Rodgers. "Improvements in Multiprocessor System Design". In: *Proceedings of the 12th annual International Symposium on Computer Architecture (ISCA'85)*. ACM, 1985, pp. 225–231. ISBN: 0-8186-0634-7. DOI: 10.1145/327070.327215.
- [50] Hasso Plattner. *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases*. 2nd. Springer-Verlag Berlin Heidelberg, 2014. ISBN: 978-3-642-55269-4. DOI: 10.1007/978-3-642-55270-0.
- [51] Tyler Akidau. *The World Beyond Batch: Streaming 101. A High-Level Tour of Modern Data-Processing Concepts*. Blog entry. 2015. URL: <https://www.oreilly.com/ideas/the-world-beyond-batch-streaming-101> (visited on 12/05/2018).
- [52] Seth Gilbert and Nancy Lynch. "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services". In: *SIGACT News* 33.2 (June 2002), pp. 51–59. ISSN: 0163-5700. DOI: 10.1145/564585.564601.
- [53] Nathan Marz. *How to Beat the CAP Theorem*. Oct. 2011. URL: <http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html> (visited on 12/05/2018).
- [54] Sean T. Allen, Matthew Jankowski, and Peter Pathirana. *Storm Applied: Strategies for Real-Time Event Processing*. Manning Publications, 2015. ISBN: 978-1617291890.
- [55] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. 5th. Addison-Wesley, 2011. ISBN: 978-0-13-214301-1.

BIBLIOGRAPHY

- [56] Nathan Marz and James Warren. *Big Data: Principles and Best Practices of Scalable Real-Time Data Systems*. Ed. by Renae Gregoire and Jennifer Stout. Manning Publications, 2015. ISBN: 9781617290343.
- [57] Jay Kreps. *Questioning the Lambda Architecture*. Website. July 2014. URL: <https://www.oreilly.com/ideas/questioning-the-lambda-architecture> (visited on 12/05/2018).
- [58] William H. Inmon. *Building the Data Warehouse*. 4th. Wiley Publishing Inc., 2005. ISBN: 978-0-7645-9944-6.
- [59] Adam Jacobs. “The Pathologies of Big Data”. In: *Communications of the ACM* 52.8 (Aug. 2009), pp. 36–44. ISSN: 0001-0782. DOI: 10 . 1145 / 1536616 . 1536632.
- [60] Divyakant Agrawal, Philip Bernstein, Elisa Bertino, et al. *Challenges and Opportunities with Big Data*. Tech. rep. The Community Research Association, 2012.
- [61] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. “The Rise of Big Data on Cloud Computing: Review and Open Research Issues”. In: *Information Systems* 47 (2015), pp. 98–115. ISSN: 0306-4379. DOI: 10 . 1016 / j . is . 2014 . 07 . 006.
- [62] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, et al. “The Stratosphere platform for Big Data Analytics”. In: *The VLDB Journal* 23.6 (2014), pp. 939–964. ISSN: 0949-877X. DOI: 10 . 1007 / s00778 - 014 - 0357 - y.
- [63] Martin Fowler. *The LMAX Architecture*. July 2011. URL: <http://martinfowler.com/articles/lmax.html> (visited on 12/05/2018).
- [64] Martin Thompson, Dave Farley, Michael Barker, Patricia Gee, and Andrew Stewart. *Disruptor: High Performance Alternative to Bounded Queues for Exchanging Data Between Concurrent Threads*. Tech. rep. LMAX, May 2011.
- [65] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Proceedings of the 6th Symposium on Operating Systems Design and Implementation (OSDI'14)*. Dec. 2004. DOI: 10 . 1145 / 1327452 . 1327492.
- [66] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. “The Google File System”. In: *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*. ACM, Oct. 2003. DOI: 10 . 1145 / 945445 . 945450.

BIBLIOGRAPHY

- [67] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, et al. “Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing”. In: *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation (NSDI’12)*. USENIX Association, Apr. 2012. ISBN: 978-931971-92-8.
- [68] Michael G. Noll. *Understanding the Internal Message Buffers of Storm*. 2013. URL: <http://www.michael-noll.com/blog/2013/06/21/understanding-storm-internal-message-buffers/> (visited on 12/05/2018).
- [69] Karthik Ramasamy. *Flying faster with Twitter Heron*. 2015. URL: https://blog.twitter.com/engineering/en_us/a/2015/flying-faster-with-twitter-heron.html (visited on 12/05/2018).
- [70] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al. “Bigtable: A Distributed Storage System for Structured Data”. In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI’06)*. 2006, pp. 205–218. DOI: 0.1145/1365815.1365816.
- [71] US Office of Management and Budget (OMB). *Annual Report to Congress: Federal Information Security Modernization Act of 2014, Fiscal Year 2016*. Tech. rep. US Office of Management and Budget, Mar. 2017.
- [72] Symantec Corporation. *Internet Security Threat Report*. 2016.
- [73] Verizon. *2017 Data Breach Investigations Report*. Apr. 2017.
- [74] Kiran Bandla. *APTnotes*. URL: <https://github.com/aptnotes/data> (visited on 12/05/2018).
- [75] Nicole Pelroth. “Governments Turn to Commercial Spyware to Intimidate Dissidents”. In: *New York Times* (May 2016). URL: <https://www.nytimes.com/2016/05/30/technology/governments-turn-to-commercial-spyware-to-intimidate-dissidents.html> (visited on 12/05/2018).
- [76] Amrit T. Williams and Mark Nicolett. *Improve IT Security With Vulnerability Management*. Tech. rep. Gartner, 2005. URL: <https://www.gartner.com/doc/480703/improve-it-security-vulnerability-management> (visited on 12/05/2018).
- [77] Julia Allen, Alan Christie, William Fithen, John McHugh, Jed Pickel, and Ed Stoner. *State of the Practice of Intrusion Detection Technologies*. Tech. rep. Carnegie Mellon Software Engineering Institute, Jan. 2000.

BIBLIOGRAPHY

- [78] Sandeep Bhatt, Pratyusa K. Manadhata, and Loai Zomlot. “The Operational Role of Security Information and Event Management Systems”. In: *IEEE Security & Privacy* 12 (Oct. 2014), pp. 35–41. DOI: 10.1109/MSP.2014.103.
- [79] Bryan Glick. *Information Security is a Big Data Issue*. 2013. URL: <http://www.computerweekly.com/feature/Information-security-is-a-big-data-issue> (visited on 12/05/2018).
- [80] Dave Shackelford. *Who’s Using Cyberthreat Intelligence and How?* Tech. rep. SANS Institute, Feb. 2015.
- [81] Splunk Inc. *Splunk Enterprise*. URL: <https://www.splunk.com/> (visited on 12/05/2018).
- [82] Micro Focus. *ArcSight Enterprise Security Manager (ESM)*. URL: <https://software.microfocus.com/en-us/products/siem-security-information-event-management/overview> (visited on 12/05/2018).
- [83] Cloud Native Computing Foundation (CNCF). *FluentD - Open Source Data Collector*. URL: <https://www.fluentd.org/> (visited on 12/05/2018).
- [84] IBM. *IBM QRadar - The Intelligent SIEM*. URL: <https://www.ibm.com/security/security-intelligence/qradar> (visited on 12/05/2018).
- [85] Tilmann Rabl, Sergio Gómez-Villamor, Mohammad Sadoghi, Victor Muntés-Mulero, Hans-Arno Jacobsen, and Serge Mankovskii. “Solving Big Data Challenges for Enterprise Application Performance Management”. In: *Proceedings of the VLDB Endowment*. 2012. DOI: 10.14778/2367502.2367512.
- [86] Karen Kent and Murugiah Souppaya. “Guide to Computer Security Log Management”. In: *NIST Special Publication* (Sept. 2006).
- [87] Chris Lonvick. *The BSD Syslog Protocol*. RFC 3164 (Informational). RFC. Obsoleted by RFC 5424. Fremont, CA, USA: RFC Editor, Aug. 2001. DOI: 10.17487/RFC3164. URL: <https://www.rfc-editor.org/rfc/rfc3164.txt>.
- [88] *Logstash*. URL: <https://www.elastic.co/de/products/logstash> (visited on 12/05/2018).
- [89] GrayLog Inc. *GrayLog*. URL: <https://www.graylog.org/> (visited on 12/05/2018).

BIBLIOGRAPHY

- [90] The Apache Software Foundation. *Common Log Format*. Web Site. URL: <https://httpd.apache.org/docs/trunk/logs.html> (visited on 12/05/2018).
- [91] Rainer Gerhards. *The Syslog Protocol*. RFC 5424 (Proposed Standard). Internet Engineering Task Force, Mar. 2009. URL: <http://www.ietf.org/rfc/rfc5424.txt>.
- [92] The MITRE Corporation. *CEE Overview*. Aug. 2012. URL: <https://cee.mitre.org/language/1.0-beta1/overview.html> (visited on 12/05/2018).
- [93] Hewlett-Packard. *Implementing ArcSight CEF*. 20. Hewlett-Packard. June 2013.
- [94] Eric Hutchins, Michael Clopperty, and Rohan Amin. “Intelligence-Driven Computer Network Defense Informed by Analysis of Adversary Campaigns and Intrusion Kill Chains”. In: *Proceedings of the 6th International Conference on Information Warfare and Security*. Lockheed Martin. 2011, pp. 80–106.
- [95] Jeffrey E. F. Friedl. *Mastering Regular Expressions*. Ed. by Andy Oram. 3rd. O’Reilly Media, Aug. 2006.
- [96] Lucas Sparvieri. *SAP HANA Text Analysis*. SAP. Jan. 2014. URL: <https://blogs.sap.com/2013/01/03/sap-hana-text-analysis/> (visited on 12/05/2018).
- [97] Satoru Kobayashi, Kensuke Fukuda, and Hiroshi Esaki. “Towards an NLP-based log template generation algorithm for system log analysis”. In: *Proceedings of The Ninth International Conference on Future Internet Technologies*. 2014, p. 11. DOI: 10.1145/2619287.2619290.
- [98] Flowerfire. *SawMaill - Universal Log File Analysis and Reporting*. URL: <https://www.sawmill.net/> (visited on 12/05/2018).
- [99] AlientVault. *AlienVault Unfied Security Management*. URL: <https://www.alienvault.com/> (visited on 12/05/2018).
- [100] Eric Knight. “Investigating Digital Fingerprints: Advanced Log Analysis”. In: *Network Security* 2010.10 (Oct. 2010), pp. 17–20. DOI: 10.1016/S1353-4858(10)70127-6.
- [101] Donal Casey. “Turning Log Files Into a Security Asset”. In: *Network Security* 2008.2 (Feb. 2008), pp. 4–7. DOI: 10.1016/S1353-4858(08)70016-3.

BIBLIOGRAPHY

- [102] Robert D. Steele. “Open Source Intelligence: What Is it? Why Is It Important to the Military?” In: *Proceedings of the 6th International Conference on Open Source Intelligence*. 1997.
- [103] Edgar Frank Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* 13.6 (June 1970), pp. 377–387. DOI: 10.1145/362384.362685.
- [104] Philip Hazel. *PCRE - Perl Compatible Regular Expressions*. URL: <http://www.pcre.org/> (visited on 12/05/2018).
- [105] Oracle. *Trail: The Reflection API*. URL: <http://docs.oracle.com/javase/tutorial/reflect/index.html> (visited on 12/05/2018).
- [106] Cisco. *Snort - Network Intrusion Detection & Prevention System*. URL: <https://www.snort.org> (visited on 12/05/2018).
- [107] Niall Gallagher. *CQEngine - Collection Query Engine*. URL: <https://github.com/npgall/cqengine> (visited on 12/05/2018).
- [108] The HoneyNet Project. *HoneyNet Challenges: Scan of the Month 34*. Web Site. 2005. URL: <http://old.honeynet.org/scans/scan34/> (visited on 12/05/2018).
- [109] Cisco. *Snort Snapshot Ruleset*. URL: <https://www.snort.org/downloads/#rule-downloads> (visited on 12/05/2018).
- [110] Proofpoint. *ETOpen - Emerging Threats Open Ruleset*. URL: <https://etadmin.proofpoint.com/etpro/open> (visited on 12/05/2018).
- [111] Eishay Smith. *JVM Serializers Performance*. July 2016. URL: <https://github.com/eishay/jvm-serializers/wiki> (visited on 12/05/2018).
- [112] Matt Welsh and David Culler. “Jaguar: Enabling Efficient Communication and I/O in Java”. In: *Concurrency: Practice and Experience* 12.7 (2000), pp. 519–538. DOI: 10.1002/1096-9128(200005)12:7<519::AID-CPE497>3.0.CO;2-M.
- [113] Mark Reinhold. *JSR 51: New I/O APIs for the Java Platform*. Sun Microsystems, Inc. May 2002. URL: <https://www.jcp.org/en/jsr/detail?id=51> (visited on 12/05/2018).
- [114] Alan Bateman. *JSR 203: More New I/O APIs for the Java Platform ("NIO.2")*. Oracle. July 2011. URL: <https://jcp.org/en/jsr/detail?id=203> (visited on 12/05/2018).
- [115] John Appleby. “Best Practices for SAP HANA Data Loads”. In: (Apr. 2013). URL: <https://blogs.sap.com/2013/04/08/best-practices-for-sap-hana-data-loads/> (visited on 12/05/2018).

BIBLIOGRAPHY

- [116] Andrey Sapegin, Marian Gawron, David Jaeger, Feng Cheng, and Christoph Meinel. “High-Speed Security Analytics Powered by In-memory Machine Learning Engine”. In: *Proceedings of the 14th IEEE International Symposium on Parallel and Distributed Computing (ISPDC’15)*. Limassol, Cyprus: IEEE, June 2015, pp. 74–81. ISBN: 978-1-4673-7147-6. DOI: 10 . 1109 / ISPDC . 2015 . 16.
- [117] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. “Pushing the Limits of Cyber Threat Intelligence: Extending STIX to Support Complex Patterns”. In: *Proceedings of the 13th International Conference on Information Technology: New Generations (ITNG’16)*. Ed. by Shahram Latifi. Vol. 448. Advances in Intelligent Systems and Computing. Las Vegas, Nevada, USA: Springer International Publishing Switzerland, Apr. 2016, pp. 213–225. ISBN: 978-3-319-32466-1. DOI: 10 . 1007 / 978 - 3 - 319 - 32467 - 8_20.
- [118] Andrey Sapegin, David Jaeger, Feng Cheng, and Christoph Meinel. “Towards a System for Complex Analysis of Security Events in Large-Scale Networks”. In: *Computers & Security*. Vol. 67. Elsevier, June 2017, pp. 16–34. DOI: 10 . 1016 / j . cose . 2017 . 02 . 001.
- [119] Martin Ussath, David Jaeger, Feng Cheng, and Christoph Meinel. “Identifying Suspicious User Behavior with Neural Networks”. In: *Proceedings of the 4th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud’17)*. New York, NY, USA: IEEE, June 2017. DOI: 10 . 1109 / CSCloud . 2017 . 10.
- [120] Alissa Torres. *Maturing and Specializing: Incident Response Capabilities Needed*. Tech. rep. SANS Institute, Aug. 2015.
- [121] Sean Barnum. *Standardizing Cyber Threat Intelligence Information with the Structured Threat Information eXpression (STIX)*. Tech. rep. MITRE Corporation, 2012.
- [122] Dave Shackelford. *The SANS State of Cyber Threat Intelligence Survey: CTI Important and Maturing*. Tech. rep. SANS Institute, 2016.
- [123] David Bianco. *The Pyramid of Pain*. Mar. 2013. URL: <http://detect-respond.blogspot.de/2013/03/the-pyramid-of-pain.html> (visited on 12/05/2018).
- [124] ESET. *En Route with Sednit: Approaching the Target*. Tech. rep. Oct. 2016. URL: <https://www.welivesecurity.com/wp-content/uploads/2016/10/eset-sednit-part1.pdf>.
- [125] Cisco Systems. *Snort Users Manual*. 2.9.11. 2017.

BIBLIOGRAPHY

- [126] Michael Meier. *Intrusion Detection effektiv!: Modellierung und Analyse von Angriffsmustern*. Springer, 2007, pp. 1–209. ISBN: 978-3-540-48251-2.
- [127] National Vulnerability Database. *CVE-2014-6271 Detail*. Sept. 2014. URL: <https://nvd.nist.gov/vuln/detail/CVE-2014-6271> (visited on 12/05/2018).
- [128] Michael Meier. “A Model for the Semantics of Attack Signatures in Misuse Detection Systems”. English. In: *Information Security*. Ed. by Kan Zhang and Yuliang Zheng. Vol. 3225. Lecture Notes in Computer Science. Springer, 2004, pp. 158–169. ISBN: 978-3-540-23208-7. DOI: 10.1007/978-3-540-30144-8_14.
- [129] Ulf Lindqvist and Phillip A. Porras. “Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST)”. In: *Proceedings of the 1999 IEEE Symposium on Security and Privacy*. May 1999, pp. 146–161. DOI: 10.1109/SECPRI.1999.766911.
- [130] Steven T. Eckmann, Giovanni Vigna, and Richard A. Kemmerer. “STATL: An attack language for state-based intrusion detection”. In: *Journal of Computer Security* 10.1 (2002), pp. 71–103.
- [131] Frédéric Cuppens and Rodolphe Ortalo. “LAMBDA: A Language to Model a Database for Detection of Attacks”. English. In: *Recent Advances in Intrusion Detection*. Ed. by Hervé Debar, Ludovic Mé, and S. Felix Wu. Vol. 1907. Lecture Notes in Computer Science. Springer, 2000, pp. 197–216. ISBN: 978-3-540-41085-0. DOI: 10.1007/3-540-39945-3_13.
- [132] Michael Meier, Niels Bischof, and Thomas Holz. “SHEDEL - A Simple Hierarchical Event Description Language for Specifying Attack Signatures”. English. In: *Security in the Information Society*. Ed. by M.Adeeb Ghonaimy, MahmoudT. El-Hadidi, and HebaK. Aslan. Vol. 86. IFIP Advances in Information and Communication Technology. Springer, 2002, pp. 559–571. ISBN: 978-1-4757-1026-7. DOI: 10.1007/978-0-387-35586-3_44.
- [133] Michael Meier, Sebastian Schmerl, and Hartmut Koenig. “Improving the Efficiency of Misuse Detection”. In: *Proceedings of the 2nd International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA’05)*. Ed. by Klaus Julisch and Christopher Kruegel. Vol. 3548. LNCS. Springer, July 2005, pp. 188–205. DOI: 10.1007/11506881_12.
- [134] Ulrich Flegel and Michael Meier. “Modeling and Describing Misuse Scenarios Using Signature-Nets and Event Description Language”. In: *it - Information Technology* 2 (Mar. 2012), pp. 71–81. DOI: 10.1524/itit.2012.0666.

BIBLIOGRAPHY

- [135] Martin Ussath, Feng Cheng, and Christoph Meinel. “Event Attribute Tainting: A New Approach for Attack Tracing and Event Correlation”. In: *Proceedings of the Network Operations and Management Symposium (NOMS)*. Istanbul, Turkey: IEEE, Apr. 2016, pp. 509–515. ISBN: 978-1-5090-0223-8. DOI: 10.1109/NOMS.2016.7502851.
- [136] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT Press, 2012. ISBN: 9780262018258.
- [137] *The UCI KDD Archive: KDD Cup 1999 Data*. URL: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html> (visited on 12/05/2018).
- [138] Andrey Sapegin, Aragats Amirkhanyan, Marian Gawron, Feng Cheng, and Christoph Meinel. “Poisson-Based Anomaly Detection for Identifying Malicious User Behaviour”. In: *Mobile, Secure, and Programmable Networking (MSPN’15)*. Vol. 9395. Lecture Notes in Computer Science. 2015, pp. 134–150. ISBN: 978-3-319-25743-3. DOI: 10.1007/978-3-319-25744-0_12.
- [139] BreachAlarm. *All Data Breach Sources*. URL: <https://breachalarm.com/all-sources> (visited on 12/05/2018).
- [140] Vindu Goel and Nicole Pelroth. “Yahoo Says 1 Billion User Accounts Were Hacked”. In: *New York Times* (Dec. 2016). URL: <https://www.nytimes.com/2016/12/14/technology/yahoo-hack.html> (visited on 12/05/2018).
- [141] Keen. *The Breached Database Directory*. URL: <https://vigilante.pw> (visited on 12/05/2018).
- [142] High-Tech Bridge. *300,000 Compromised Accounts Available on Pastebin: Just the Tip of Cybercrime Iceberg*. Web site. Feb. 2014. URL: https://www.htbridge.com/news/300_000_compromised_accounts_available_on_pastebin.html (visited on 12/05/2018).
- [143] *Data Breach QuickView Report: 2016 Data Breach Trends - Year In Review*. Tech. rep. Risk Based Security, 2017.
- [144] Jens Heyens, Kai Greshake, and Eric Petryka. *MongoDB databases at risk – Several thousand MongoDBs without access control on the Internet*. Tech. rep. Universität des Saarlandes, Jan. 2015.
- [145] Yakov Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. RFC 4180 (Informational). Internet Engineering Task Force, Oct. 2005. URL: <http://www.ietf.org/rfc/rfc4180.txt>.

BIBLIOGRAPHY

- [146] K. Moriarty, B. Kaliski, and A. Rusch. *PKCS #5: Password-Based Cryptography Specification Version 2.1*. RFC 8018. Jan. 2017.
- [147] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and Xiaofeng Wang. “The Tangled Web of Password Reuse”. In: *21st Annual Network and Distributed System Security Symposium (NDSS’14)*. Feb. 2014. DOI: 10.14722/ndss.2014.23357.
- [148] Roman V. Yampolskiy. “Analyzing User Password Selection Behavior for Reduction of Password Space”. In: *Proceedings of the 2006 40th Annual IEEE International Carnahan Conferences Security Technology*. 2006, pp. 109–115. DOI: 10.1109/CCST.2006.313438.
- [149] Feng Cheng, Amir Azodi, David Jaeger, and Christoph Meinel. “Multi-Core Supported High Performance Security Analytics”. In: *Proceedings of the 11th International Conference on Dependable, Autonomic and Secure Computing (DASC’13)*. Chengdu, China: IEEE, Dec. 2013, pp. 621–626. ISBN: 978-1-4799-3380-8. DOI: 10.1109/DASC.2013.136.
- [150] Feng Cheng, Amir Azodi, David Jaeger, and Christoph Meinel. “Security Event Correlation supported by Multi-Core Architecture”. In: *Proceedings of the 3rd IEEE International Conference on IT Convergence and Security (IC-ITCS’13)*. Macau, China: IEEE, Dec. 2013, pp. 408–412. DOI: 10.1109/ICITCS.2013.6717881.
- [151] Andrey Sapegin, Marian Gawron, David Jaeger, Feng Cheng, and Christoph Meinel. “Evaluation of In-Memory Storage Engine for Machine Learning Analysis of Security Events”. In: *Concurrency and Computation: Practice and Experience* 29.2 (Mar. 2016). DOI: 10.1002/cpe.3800.
- [152] Brian Krebs. *Adobe To Announce Source Code, Customer Data Breach*. Web Site. Oct. 2013. URL: <http://krebsonsecurity.com/2013/10/adobe-to-announce-source-code-customer-data-breach/> (visited on 12/05/2018).
- [153] BKA. *Hacker-Sammlung gefunden: 500 Mio. E-Mail-Adressen und Passwörter betroffen*. Website. July 2017. URL: https://www.bka.de/SharedDocs/Kurzmeldungen/DE/Kurzmeldungen/170705_HackerSammlung.html (visited on 12/05/2018).