Architectural Tactics for Big Data Cybersecurity Analytic Systems: A Review

Article *in* Journal of Systems and Software · February 2018

DOI: 10.1016/j.jss.2019.01.051

Faheem Ullah
University of Adelaide
22 PUBLICATIONS **187** CITATIONS
SEE PROFILE

Muhammad Ali Babar
University of Adelaide
434 PUBLICATIONS **9,432** CITATIONS
SEE PROFILE

Ullah, Faheem, and Muhammad Ali Babar. "Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review." *Journal of Systems and Software* 151 (2019): 81-118.

Architectural Tactics for Big Data Cybersecurity Analytics Systems: A Review

Faheem Ullah[a, b], Muhammad Ali Babar[a, b, ++]

[a]Cyber Security Adelaide, University of Adelaide, Australia
[b]CREST- the Centre for Research on Engineering Software Technologies, Australia

Abstract

Context: **B**ig **D**ata **C**ybersecurity **A**nalytics (BDCA) systems encompass a breed of systems that leverage big data technologies for analyzing security events data to protect organizational networks, computers, and data from cyber attacks. Given the growing importance of BDCA systems, it is important to systematically discover and document architectural aspects of such systems for building a suitable design space. *Objective:* We aimed at identifying the most frequently reported quality attributes and architectural tactics for BDCA systems reported in the literature. We also intended to highlight the research gaps in this area. *Method:* We used Systematic Literature Review (SLR) method for reviewing 74 papers. *Result:* Our findings are twofold: (i) identification of 12 most frequently reported quality attributes and the justification for their significance for BDCA systems; and (ii) identification and codification of 17 architectural tactics for addressing the identified quality attributes. The identified tactics include six performance tactics, four accuracy tactics, two scalability tactics, three reliability tactics, and one security and usability tactic each. *Conclusion:* Our study reveals that in the context of BDCA (a) performance, accuracy and scalability are the most important quality concerns (b) data analytics is the most critical architectural component (c) despite the significance of interoperability, modifiability, adaptability, generality, stealthiness, and privacy assurance, these quality attributes lack explicit architectural support (d) empirical investigation is required to evaluate the impact of the codified tactics and explore the quality trade-offs and dependencies among the tactics and (e) the reported tactics need to be modelled using a standardized modelling language such as UML.

Keywords: Big Data, Cybersecurity, Quality Attribute, Architectural Tactic

1. Introduction

Cybersecurity is aimed at protecting Information and Communication Technology (ICT) infrastructures (i.e., computer hardware, software, networks and data) from unauthorized access and/or disruptions [1]. Cybersecurity solutions are increasingly leveraging big data technologies for (i) collecting security events data (ii) performing deep security analytics and (iii) providing a consolidated view of security information [2][3]. A large-scale industrial survey [4] reports that an increasing number of organizations have been realizing the importance and value of big data technologies for protecting their ICT infrastructures against potential cyber attacks. In our research, a cybersecurity solution that incorporates big data technologies is called **B**ig **D**ata **C**ybersecurity **A**nalytics (BDCA) system, which is defined as "*A system that leverages big data technologies for collecting, storing, and analyzing large volume of security event data to protect organizational networks, computers, and data from unauthorized access, damage, or attack*".

Given that BDCA systems sit at the intersection of cybersecurity, big data technologies, and software engineering, it is a challenging undertaking to design and evaluate highly scalable BDCA systems that can effectively and efficiently manage and analyze a massive amount of security event data [5, 6]. Designers of BDCA systems are expected to consider several unique aspects of BDCA systems such as (i) privacy assurance (ii) authenticity and integrity of security events data (iii) lack of datasets for evaluating security systems (iv) asymmetrical cost of misclassification of security events (v) detecting and correcting adversarial learning and (vi) attack time scale [2][11]. Like any other large-scale software-intensive system, the achievement of common domain specific functional and non-functional (i.e., quality attributes) goals should be a key concern while designing BDCA systems. There is a lack of consensus on which are the most critical quality attributes of BDCA systems.

[+] Corresponding author.

Furthremore, there is a paucity of systematic efforts aimed at building guidelines for designing BDCA systems [3, 12].

For such design guidelines, the Software Engineering (SE) community has emphasized the importance of organising design knowledge (i.e., quality attributes, tactics, and patterns) as design spaces [13, 14], which communicate experienced-based design knowledge to software designers. A design space highlights the significance of certain quality attributes (e.g., scalability and reliability) and codify architectural tactics (i.e., *a reusable design strategy that helps achieve a particular quality attribute* [15]), which can be used for the systematic design of a software system [14]. Recently, there are increasing number of efforts to systematically reviewing and understanding the architectural aspects of software-intensive systems for building and leveraging architectural knowledge (i.e., design spaces) in different domains such as cyber-foraging [16], energy efficiency in cloud [17], ambient assisted living systems [18], and self-adaptive systems [19].

Motivated by such efforts and increasing need and importance of security design knowledge, we decided to carry out a systematic study of the state-of-the-art of BDCA systems from an architectural perspective. Our study aimed to identify and codify the most important quality attributes and architectural tactics for BDCA systems. The outcome of our study is expected to communicate experience-based design knowledge to software architects for systematically and efficiently designing BDCA systems. To this end, we have systematically selected and rigorously reviewed 74 relevant papers reporting BDCA systems. Our analysis and synthesis of the data extracted from the 74 reviewed papers have enabled us to make the following key contributions:

1. *Identification of the 12 most important quality attributes for BDCA systems;*
2. *A catalogue of 17 architectural tactics for addressing quality concerns in BDCA systems;*
3. *Identification of potential research areas to advance the state-of-the-art on architecting BDCA systems.*

We assert that this review will interest to researchers and practitioners from at least three communities, software engineering, cybersecurity, and big data technologies, that are expected to be interested in gaining a better understanding of the architectural knowledge (i.e., quality attributes and design choices) used for designing BDCA systems and the identification of the areas where further exploration and experimentation are required.

1.1. Big Data Cybersecurity Analytics System

Given that BDCA systems are a new breed of software-intensive systems, it would be pertinent to discuss some of the characteristics of such systems and their big data technologies needs before we delve into the details of the reported study. BDCA systems are characterized by (1) *Monitoring diverse assets:* BDCA systems monitor diverse set of activities on an organization's ICT assets such as computing machines, data storage systems, end-user applications, and computing networks [7] (2) *Security event data integration*: BDCA systems *integrate and coorelates security event data from* multiple security systems (e.g., IDS, anti-virus, and firewall) to gain a holistic understanding of the cyberspace surrounding organizational ICT assets [2, 3] (3) *Handling large volume of security event data*: The exponential growth in cyber activities over the years has made it challenging for organization to store and analyse large amounts of security event data. For instance, merely 1Gbps continuous network traffic can make an Intrusion Detection System (IDS) obsolete [8]. In addition to network traffic, organizations produce a huge amount of log data that is even difficult to store than being analyzed for security awareness. For example, Cloud Security Alliance reported in 2013 that an enterprise as large as HP generates around one trillion security events per day [3]. Hence, BDCA system leverages big data storage and processing technologies (e.g., Hadoop and MapReduce) to handle the storage and processing of such large volume of data (4) *Enabling real-time and deep analytics*: The traditional tools (e.g., a relational database) often become a bottleneck in performing real-time and holistic security analytics. A study conducted at Zions Bancorporation revealed that their traditional SIEM system takes around 20-60 minutes to query a month of security event data while the same task takes about one minute when using Hadoop technology [9]. A BDCA system enables (near) real-time and deep analytics of security event data to reveal hidden patterns and detect low and slow attacks (5) *Analyzing heterogenous data:* The security event data collected from various sources is often in different formats. In addition to structured data, a BDCA system operates on large volumes of semi-structured and unstructured data (e.g., text in logs files) to extract valuable security insight from the data [10].

1.2. Paper Organization

The rest of this paper is organized as follows. Section 2 presents the research methodology used for this research. Section 3 reports the demographic information about the reviewed papers. Section 4 presents the identified critical quality attributes, which is followed by Section 5 that presents the codified architectural tactics. Section 6 presents the lessons learnt, areas for future research, and provides recommendations for software architects. Section 7 reports the related work. Section 8 discusses the limitations of this work. Finally, Section 9 concludes the paper.

2. Research methodology

We used Systematic Literature Review (SLR) [20] method for conducting this review. An SLR is aimed at systematically identifying and selecting relevant papers to be reviewed on a particular topic and rigorously analysing the extracted data to answer a set of research questions [20]. The main components of our review protocol were: (i) research questions; (ii) search procedures; (iii) inclusion and exclusion criteria; (iv) papers selection; and (v) data extraction and data synthesis.

2.1. Research questions

The objective of this study was to identify and codify the most important quality attributes (RQ1) and architectural tactics (RQ2) for achieving those quality attributes in BDCA systems. Table 1 presents the research questions and their respective motivators for this SLR.

Table 1. Research questions for this SLR.

Research question	**Motivation**
RQ1: Which are the most important quality attributes for BDCA systems?	To find out the most frequently emphasized quality attributes and the motivation behind their importance for a BDCA system.
RQ2: What are the architectural tactics for addressing quality concerns in BDCA systems?	To gain a detailed understanding of the various architectural tactics employed for achieving quality in BDCA systems.

2.2. Search strategy

According to the guidelines provided in [20] and [21], we defined a search strategy to retrieve as many relevant peer-reviewed papers as possible. Our search technique is described as follows.

2.2.1. Search method

We used automatic search for retrieving the potentially relevant papers from six digital libraries: ACM Digital Library, IEEE Xplore, Scopus, ScienceDirect, SpringerLink, and Wiley Online Library. These libraries were searched using the search terms introduced in Section 2.2.2. In order to ensure the identification and selection of as many relevant papers as possible, we used snowballing [22] for complementing the automatic search.

2.2.2. Search terms

We designed a search string (shown in Figure 1) according to the guidelines provided in [20], which consisted of two parts – security and big data processing frameworks. We initially included architecture-centric parts (i.e., quality attribute and architectural tactic), however, the search string returned only 14 papers. One possible reason (supported by Section 3.2) for the low number of papers can be that BDCA systems are primarily published outside Software Engineering (SE) venues, thereby, the papers do not explicitly use architecture-centric terms (e.g., quality attributes and architectural tactics). We ensured through our inclusion criteria (i.e., *I2*) that only the papers reporting architectural solutions were selected. For the two parts (i.e., security and big data processing frameworks), we incorporated the synonyms and the terms related to the basic two terms. We finalised the search string after ensuring that the pilot searches using the search terms were returning the known papers related to our review. The search terms in the string were matched only with the title, abstract, and keywords of the papers in the searched digital databases (except SpringerLink which does not allow to restrict the search to specific parts of a paper). We found that several of the retrieved papers were focussed on improving security of big data and associated technologies (instead of leveraging big data and associated technologies for security, which is the focus of this SLR). We filtered out those papers during the inclusion and exclusion phase.

TITLE-ABS-KEY *((("security"* ***OR*** *"intrusion"* ***OR*** *"IDS"* ***OR*** *"SIEM"* ***OR*** *"anomaly detection"* ***OR*** *"outlier detection"* ***OR*** *"fraud detection"* ***OR*** *"network monitoring"* ***OR*** *"forensic"* ***OR*** *"threat"* ***OR*** *"data leakage"* ***OR*** *"data breach"* ***OR*** *"data theft"* ***OR*** *"data exfiltration"* ***OR*** *"attack"* ***OR*** *"data stealing")* ***AND*** *("hadoop"* ***OR*** *"HDFS"* ***OR*** *"mapreduce" OR "spark"* ***OR*** *"flink"* ***OR*** *"storm"* ***OR*** *"samza"* ***OR*** *"mesos")))*

Figure 1. Search string for this SLR.

2.2.3. Data sources

Table 2 shows the searched digital libraries. The IEEE Xplore and ScienceDirect do not support the execution of a query with more than 15 terms. We had to split our query into two parts to make it run on IEEE Xplore and ScienceDirect. Apart from SpringerLink, we ran our query on other databases to match the terms only in title, abstract, and keywords. We have already mentioned that SpringerLink does not support searches in the specific parts of a paper [23]. The limitation of SpringerLink forced us to either limit the search only to the title of a paper or apply the string to the whole text of each of the potentially relevant papers. Whilst the former retrieved a very low number of papers, the later retrieved quite a large number of potentially relevant papers (8483 in total). To solve this issue, we followed the strategy used in [23]. According to this strategy, we examined only the first 1000 papers out of 8483 returned papers. This strategy does not pose a significant threat of missing papers as our data sources include Scopus, which encompasses a large number of papers indexed on SpringerLink. We did not use Google scholar due to its low precision of the results and the tendency of returning irrelevant papers.

Table 2. Database sources.

Source	**URL**
IEEE Xplore	http://ieeexplore.ieee.org
Scopus	https://www.scopus.com/
ScienceDirect	http://www.sciencedirect.com
ACM	http://portal.acm.org
SpringerLink	https://link.springer.com/
Wiley	http://onlinelibrary.wiley.com/

2.3. Inclusion and exclusion criteria

Table 3 shows the inclusion (I) and exclusion (E) criteria applied for selecting the papers. We targeted papers that highlighted the importance of one or more quality attributes and leverage, propose, or evaluate the incorporation of architectural solutions (architectural tactics) for achieving certain quality attributes (e.g., performance, reliability, accuracy, and scalability) in BDCA systems. Our search string returned the papers that were not architecture-based (e.g., algorithmic solutions or solutions based on controlling some operating parameters), we excluded such papers. We also excluded the papers that addressed the security of the big data itself or its associated technologies (e.g., security of Hadoop). However, we made sure that any paper that leveraged big data technologies for the security of big data itself or its technologies should not be discarded from this SLR.

Table 3. Inclusion and exclusion criteria.

Inclusion criteria
I1: A study that is leveraging big data and its corresponding tools and technologies for cybersecurity
I2: A study that is architecture-based, which means the study should provide architectural solution (i.e., components and architectural model) to the BDCA problem
I3: A study that emphasizes the importance of one or more quality attributes for BDCA
I4: A study that evaluates the proposed BDCA solution(s)
Exclusion criteria
E1: A study that is not peer-reviewed (e.g., position papers, panel discussion, editorials, and keynotes)
E2: A study written in a language other than English

2.4. Study selection

The papers selected from each digital database at each stage are shown in Figure 2. The different phases of the selection process are briefly described in the following.

- *Automatic search:* We ran our designed search string on six digital databases and retrieved a total of 4634 papers as a result of running automatic searches.
- *Title-based selection:* We read the title of the papers to quickly decide whether or not each of the retrieved papers was relevant to our SLR. In cases, where we were not able to decide about the relevance of a paper only based on its title, the paper was transferred to the next round of selection. Title-based selection reduces the pool of papers from 4634 to 748.
- *Duplication removal:* Scopus indexes papers available in several databases such as IEEE Xplore and ACM, therefore, it was expected that there will be duplicate papers in the pool of 748 selected papers. In this phase, we removed the duplicate papers, which reduced the number of our papers to a total of 516.
- *Abstract-based selection:* We read the abstract of each of the 516 papers to ensure that the papers were related to our SLR. During this stage, we discarded 348 papers that left us with 168 papers.
- *Full-text selection:* We completely read each of the 168 selected papers. A total of 69 papers were selected based on reading the whole text of the 168 papers.
- *Snowballing:* We used snowballing method [22] to explore the references of the 69 selected papers. We found 26 potentially related papers. Based on applying inclusion and exclusion criteria, we selected 5 papers from the 26 papers, which brought us to the final total of 74 papers for this SLR.

We did not restrict the selection based on the publication date of a paper. The reason for this was that the incorporation of big data technologies for cybersecurity is a comparatively new research field and as such our search process did not return the papers published before 2009. Appendix A enlists the papers selected for this SLR. We used the terms paper and study interchangeably in this paper. Each paper has a unique identifier (S#). For example, the paper "A Cloud Computing Based Network Monitoring and Threat Detection System for Critical Infrastructures" is identified as S10. The system name included in Appendix A refers to the name of a BDCA system that was described in a paper. Some authors did not use an explicit name for their respective systems. For such systems, we followed the strategy adopted in [16] to name the system based on the unique features of each of the systems for discussing different aspects of those systems.

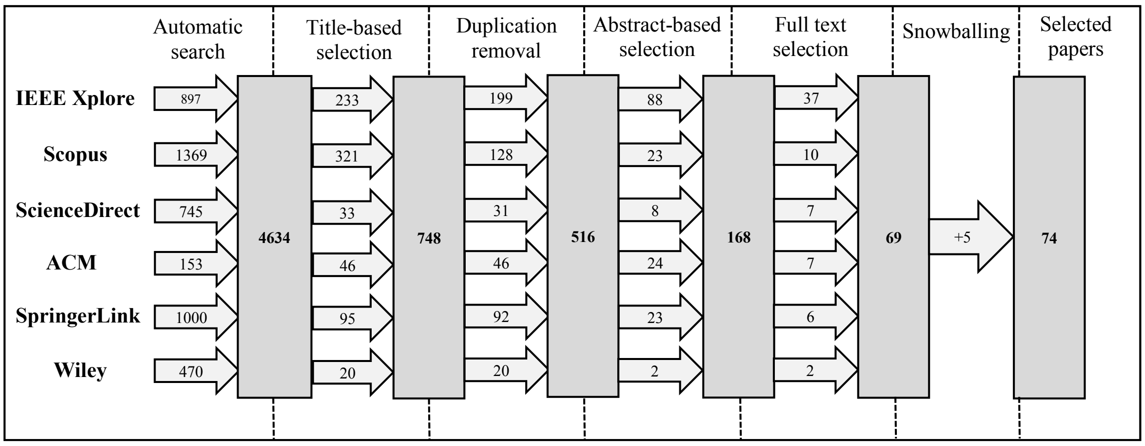


Figure 2. Phases of study selection process.

2.5. Data extraction and synthesis

This section describes the process of data extraction from the selected papers and the analysis of the extracted data to answer the research questions of this SLR.

2.5.1. Data extraction

We extracted the required data from the selected studies using a data extraction form that included the data items envisaged necessary to answer this review's research questions. The data extraction form is shown in Table 4. The data items D5 (publication venue) and D6 (citation count) were included as the indicators of the quality of the selected papers. The extracted data were recorded in an MS Excel spreadsheet for analysis.

2.5.2. Data synthesis

We extracted three types of data – demographic data, quality attributes, and architectural tactics. We analyzed the demographic data using descriptive statistics. The results of our analysis of the demographic data have been presented in Section 3. We analyzed the data items D11 (quality attributes), D12 (rationale for quality attributes) and D13 (architectural tactics) using thematic analysis method [24], which is a widely used qualitative data analysis method that identifies themes extracted from multiple studies, interpret the themes, and draw conclusions. Our data analysis process is depicted in Figure 3. We explain our process step-by-step with the help of an example for one quality attribute (i.e., performance (Section 4.1)) and an associated tactic (i.e., Feature Selection and

Table 4. Data extraction form.

#	Data item	Description	Research Questions
D1	Authors	Authors of the paper	
D2	Year	Publication year of the paper	Demographic data
D3	Title	Title of the paper	
D4	Publication type	The publication type of paper (e.g., conference)	Demographic data
D5	Publication venue	The venue where paper is published	Demographic data
D6	Citation count (google scholar)	The number of times paper is cited according to google scholar	
D7	Application domain	The type of BDCA system (e.g., Intrusion detection system)	Demographic data
D8	Data source(s)	The type of data the system is using for security analytics (e.g., network data)	Demographic data
D9	Big data processing framework	The processing framework system is leveraging (e.g., Hadoop)	Demographic data
D10	Proposed technique	A short summary of the security analysis technique proposed in the paper	
D11	Quality attribute(s)	The quality attribute(s) emphasized in the paper	RQ1
D12	Rationale for quality attribute(s)	The motivation for highlighting the quality attribute(s)	RQ1
D13	Architectural tactic(s)	The architectural tactic(s) proposed in the paper	RQ2

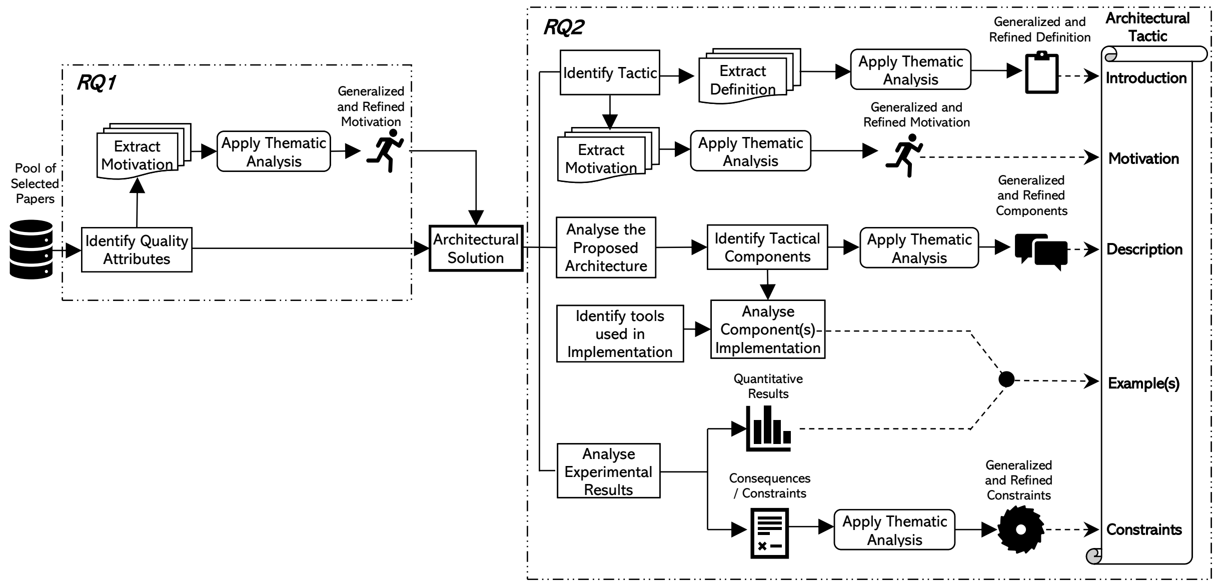

RQ1
Pool of Selected Papers
Identify Quality Attributes
Extract Motivation
Apply Thematic Analysis
Generalized and Refined Motivation
Architectural Solution
RQ2
Identify Tactic
Extract Definition
Apply Thematic Analysis
Generalized and Refined Definition
Extract Motivation
Apply Thematic Analysis
Generalized and Refined Motivation
Analyse the Proposed Architecture
Identify Tactical Components
Apply Thematic Analysis
Generalized and Refined Components
Identify tools used in Implementation
Analyse Component(s) Implementation
Quantitative Results
Analyse Experimental Results
Consequences / Constraints
Apply Thematic Analysis
Generalized and Refined Constraints
Architectural Tactic
Introduction
Motivation
Description
Example(s)
Constraints

Figure 3. An Overview of our Data Analysis Process.

Extraction (Section 5.1.3)). The example illustrates the working from the raw data extracted from the primary studies to the reporting of quality attribute and codification of the associated tactic.

Quality attribute (RQ1): For answering RQ1, we identified the explicitly mentioned quality attributes in the selected papers as shown in Figure 3. For example, the performance quality attribute is identified through explicit indication of the significance of response time, throughput, latency, or performance in the papers. After identifying the quality attribute, we extracted the reported motivators for a particular quality attribute in the reviewed system. We then applied the thematic analysis methods to the extracted motivators to become familiar with the data, generate initial codes, search the relevant themes, and name the themes. For instance, the motivators extracted from [S1], [S7], [S9], [S26], [S48] were themed as 'large size of data' as these papers argue that without high performance, a BDCA system will be overwhelmed with the massive amount of security event data. The application of the thematic analysis generalized and refined the extracted motivators as shown in Figure 3.

Architectural tactic (RQ2): For answering RQ2, we analyzed the reported solutions to identify and record a brief definition of the tactics that were employed for achieving the desired quality attributes. The brief definitions were subjected to thematic analysis that enabled us to generalize, refine, and name the tactics. For example, the data extracted from [S9], [S10], [S12], [S13], [S14], [S15], [S31], [S32], [S33], [S55], [S57] mention that the selection and extraction of features from the collected data improves performance by discarding irrelevant features, which was themed as 'Feature Selection and Extraction tactic'. Once the tactic had been identified, we analyzed each tactic from multiple perspectives.

First, we identified and extracted the motivators for choosing the specific tactic for achieving the desired quality attribute. We then applied the thematic analysis method on the motivators extracted from multiple papers to get generalized and refined motivators for the tactic. For example, the papers [S9], [S10], [S14], [S32], [S33], [S55] urge that several features within the collected security event data are irrelevant with respect to attack detection, hence, selection and extraction of pertinent features can help improve performance. We themed this motivator as 'irrelevancy of features' that is one of the motivators for using Feature Selection and Extraction tactic. Next, we investigated the proposed architecture to identify the components related to each of the identified tactics and their interactions with other components in the reported architectures. This was followed by the application of thematic analysis to popularize the components that best capture each tactic's details and their interactions. In case of Feature Selection and Extraction tactic, the identified components include Feature Selection and Feature Extraction that interact with data storage and data processing components respectively.

We analyzed the implementation aspects of the identified components and the tools used for implementation. This aspect of the tactics is reported in the example subsection. We analyzed the experimental results related to each tactic as reported in the reviewed papers. The quantitative results were reported along with examples while the consequences of each tactics, as identified through experimental evaluation or elsewise, were extracted and subjected to thematic analysis as shown in Figure 3. For instance, several papers [S10], [S12], [S14] report that discarding important features can be detrimental to accurately detecting attacks, which is themed as 'discarding important features' and is reported as one of the constraints for the Feature Selection and Extraction tactic. After codifying all the tactics, the dependences among the tactics were identified based on our analysis and reflections.

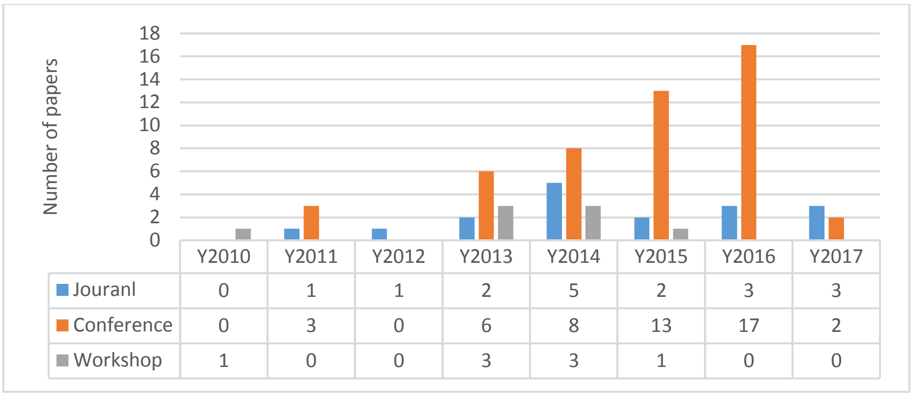
Our data synthesis process revealed that the architectural aspects of BDCA systems are primarily discussed around 12 main themes (i.e., quality attributes), which are shown in Table 6 along with the papers belonging to each theme. As shown in Figure 3, the thematic analysis has been applied at multiple levels of abstraction (e.g., quality attributes and architectural tactics), which enabled the analysis process to reveal a list of sub-themes (i.e., architectural tactics). These sub-themes, shown in Figure 8, are associated with the main themes (i.e., quality attributes). The themes and sub-themes have been elaborated in Section 4 and Section 5 respectively.

3. Demographic information

This section reports the distribution of the reviewed papers along the years and types of publications, big data processing frameworks used, application domains, and data sources leveraged by the reviewed BDCA systems.

3.1. Chronological view

Figure 4 shows that the reviewed papers were published between 2010 and 2017. Our SLR covers the papers published before *8th May 2017* when the search process for selecting the potentially relevant papers was completed. Figure 4 shows a regular upward growth in the number of papers on security analytics leveraging big data technologies. This is in accordance with the ideas surfaced in the Cloud Security Alliance Workshop [25] where it was anticipated that the role of big data analytics in security will continue increasing. This upward trend can be interpreted from two perspectives. First - the threat landscape is changing from cyber attacks launched by individuals to cyber attacks launched by organized groups. Second - the traditional tools and technologies are unable to deal with the high volume, large size, and heterogeneous nature of security events data. Therefore, the adoption of big data technologies is receiving an increasing attention from the cyber security researchers.



	Y2010	Y2011	Y2012	Y2013	Y2014	Y2015	Y2016	Y2017
🟦 Jouranl	0	1	1	2	5	2	3	3
🟧 Conference	0	3	0	6	8	13	17	2
⬜ Workshop	1	0	0	3	3	1	0	0

Figure 4. Number of selected papers per year and their distribution over venues' types.

3.2. Publication venues and types

Figure 4 indicates that most of the papers (49 papers (66.2%)) have been published in conferences; there were 17 (22.9%) journal papers and 8 (10.8%) workshops papers. The 74 reviewed papers were published in 59 venues, in which BigData Congress, Journal of Supercomputing, and Security and Privacy in Big Data workshop are the leading venues for publishing work on security analytics. These three venues have published three papers each. Five venues have published two papers each. These venues include TrustCom/BigDataSE/ISPA, Conference on Systems, Man, and Cybernetics, Network Operations and Management Symposium, Special Interest Group on Data Communication (SIGCOMM), and Conference on Parallel and Distributed Systems. The rest of the 55 papers were published in 55 different venues. The selected papers have been primarily published in three research areas – Big Data (21 paper), Network Communications (17 papers), and Cybersecurity (15 papers). It is interesting to note that only four papers have been published in the Software Engineering related venue. These findings show that researchers with different research backgrounds are interested in BDCA.

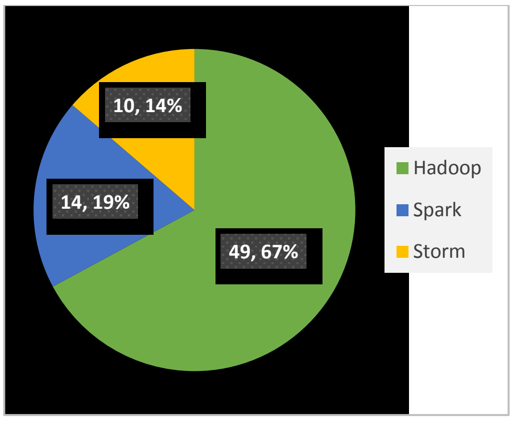
3.3. Big data processing frameworks

We have classified the reviewed papers based on the big data processing framework employed in the BDCA systems. The processing framework provides the guidelines for processing big data. In the reviewed papers, there are three frameworks used – Hadoop[1] (49 papers), Spark[2] (14 papers), and Storm[3] (10 papers). These frameworks are being used by well-known organizations such as Yahoo, Google, IBM, Facebook, and Amazon [26]. Figure 5a shows this information. We could not classify one of the papers into either of the three categories based on the available information. Figure 5b shows the patterns of the use of these frameworks adopted along the years. It is interesting to note that in the first four years (i.e., from 2010 to 2013), only Hadoop has been incorporated whilst for the last four years (i.e., from 2014 to 2017), a steady upward trend can be observed for adopting Spark and Storm. One possible reason for this trend can be the migration from batch processing to stream processing due to the rapidly changing time-sensitive requirements of BDCA systems. The low numbers shown in 2017 is due to the reason that the papers published only before 8th May 2017 are included in 2017.

[1] http://hadoop.apache.org/

[2] https://spark.apache.org

[3] http://storm.apache.org/



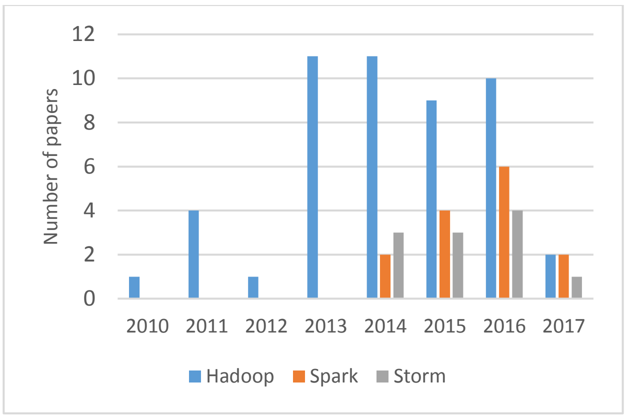


Figure 5. a) No. and percentage of papers per category b) Yearly distribution of papers over type of framework.

3.4. Application domains

We have categorized the included papers based on the application domain. For such categorization, we have analyzed the data item D7 in the data extraction form (Table 4). The domain-based categorization is of a potential value for researchers and practitioners interested in the domain-oriented prospects of BDCA systems. The papers on BDCA systems are primarily of two types – generic and specific. The generic category reports BDCA systems (IDS and Alert Correlation) for detecting a variety of attacks. The specific category includes the papers that report BDCA systems designed for detecting a particular type of attack such as Denial of Service (DoS). There are several cases where an IDS is evaluated with a particular attack (e.g., [S4] and [S12]), however, we have included such systems in the IDS category only. As shown in Table 5 that the 74 reviewed papers have been categorized into nine groups. The majority of the reviewed papers belong to IDS category (i.e., 34 papers) followed by unclear (11 papers) and alert correlation (i.e., 8 papers). The unclear category includes the papers that cannot be explicitly (i.e., without authors' interpretation) categorized into either of the remaining 8 groups.

Table 5. Distribution of application domains of the reviewed papers

S. No	Application domain	# of papers	Papers
1	Intrusion Detection System	34	S1, S3, S4, S5, S8, S9, S12, S14, S15, S18, S21, S26, S27, S29, S31, S32, S34, S35, S41, S44, S46, S48, S51, S54, S55, S57, S59, S63, S64, S66, S67, S72, S73
2	Alert Correlation	8	S6, S7, S11, S16, S19, S38, S39, S65
3	DoS Detection	7	S2, S20, S28, S33, S43, S61, S62
4	Botnet Detection	4	S22, S23, S36, S69
5	Forensic Analysis	2	S30, S53
6	APT Detection	2	S37, S47
7	Malware Detection	4	S43, S45, S60, S68
8	Phishing Detection	2	S17, S50
9	Unclear	11	S10, S13, S25, S40, S49, S52, S56, S58, S70, S71, S74

3.5. Data sources

A BDCA system can collect data from a number of data sources, which have been classified into three categories in this study – (1) *Network,* (2) *Host,* and (3) *Hybrid*. Figure 6 shows the number and the percentage of papers belonging to each category. The systems belonging to the first category (i.e., network) collect data from network infrastructure of an organization. Within the network infrastructure, a BDCA system can leverage different kinds of data such as Netflow data, packet data, honeypot data, IDS log data, and firewall logs [27]. The systems belonging to the second category (i.e., host) collect data from an organization's host machines. These data sources include but not limited to operating system logs, system call logs, web server logs, email logs, and windows event logs. The systems belonging to the third category (i.e., hybrid) collect data from both network and host machines. Figure 6 shows that most of the systems (i.e., 55) in our review rely on the data collected from the network, followed by hybrid (i.e., 13), and only 6 systems leverage host data for security analytics. One possible reason for such a focus on the network-based BDCA systems could be that unlike host-based systems that protect only a specific host within an organization, the network-based systems take into consideration the security of an entire organization's network. The pros and cons of network-based and host-based security systems have already been well explored. Interested readers can refer to [28-30].

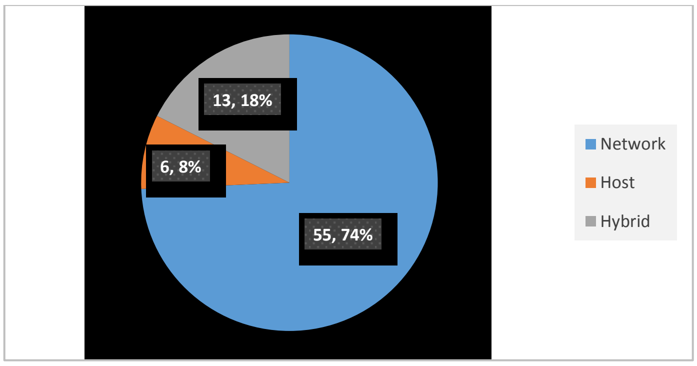


Figure 6. Number and percentage of papers distributed over type of data source.

4. RQ1: Quality attributes

This section reports the results based on analysis of the data about the critical quality attributes reported for BDCA and their respective rationale. The analysis is meant to answer RQ1, "*Which are the most important quality attributes for BDCA systems?"*. We answer this question from two perspectives: (1) We present the statistical analysis of the quality attributes identified in the reviewed papers to understand the depth of the emphasis placed on the identified quality attributes and (2) We report the identified motivation to justify that why these particular quality attributes have been highly emphasized for BDCA.

4.1. Statistics of the quality attributes

This section reports the results from analysis of the data for the item D11 of the data extraction form (Table 4). The item D11 records the information about the quality attributes emphasized in the reviewed papers. Table 6 presents the number, percentage, and identifiers of papers emphasizing particular quality attributes. These are the quality attributes that have been emphasized, highlighted, or considered important for BDCA systems. If a paper reports a quality attribute important that does not mean that that paper also focuses on achieving that particular quality attribute. For example, one paper [S15] highlights interoperability as a critical quality attribute for BDCA systems, however, it does not report any strategy for achieving interoperability. Multiple attributes may be emphasized in one paper such as [S2] reports performance, accuracy, scalability, and reliability quality attributes for a BDCA system. Table 6 shows that performance (67 papers), accuracy (43 papers), scalability (40 papers), and reliability (23 papers) have been reported as important quality attributes for BDCA systems by several papers.

The limited emphasis (i.e., only 4 papers) on security as a quality attribute warrants to distinguish between two scenarios – (i) a software system being itself secure (security as a quality attribute [31]) and (ii) a software system (e.g., BDCA system) providing security to an organization's cyber space (security as a functional attribute [31]). All of the 74 reviewed papers consider security of the cyberspace of an organization important. However, only 4 papers explicitly consider and address the security of BDCA system. Apart from the reported 12 quality attributes, the reviewed papers also mentioned some other quality attributes, which were found in less than 3 papers are not shown in Table 6. These quality attributes include design simplicity [S6], algorithm expressiveness [S6], modularity [S10], uniform programmability [S17], and reusability [S74].

4.2. Quality attributes definition for BDCA systems

This section reports our analysis of the data item D12. The objective is to clarify the definition, use, and emphasis on the 12 quality attributes presented in Table 6. Such clarification will help answer the question that why these particular quality attributes are more important for a BDCA system. For each quality attribute, we provide a definition and motivation for their achievement in a BDCA system.

Table 6. Papers emphasizing quality attributes

Quality attribute	# of papers	% of papers	Paper identifiers
Performance	67	90.5	S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17, S18, S19, S20, S21, S23, S24, S26, S27, S29, S30, S31, S32, S33, S34, S35, S36, S37,

			S38, S39, S40, S41, S42, S43, S44, S45, S46, S47, S48, S49, S50, S51, S52, S53, S54, S55, S56, S57, S58, S60, S61, S62, S63, S64, S65, S66, S67, S70, S71, S73, S74
Accuracy	43	58.1	S1, S2, S5, S8, S9, S10, S12, S14, S15, S18, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31, S36, S37, S40, S41, S42, S43, S46, S47, S49, S55, S57, S58, S59, S60, S61, S62, S64, S66, S67, S70, S73
Scalability	40	54.0	S2, S3, S5, S6, S7, S8, S9, S10, S12, S15, S16, S17, S18, S19, S21, S22, S24, S25, S29, S30, S31, S38, S42, S45, S47, S48, S49, S51, S52, S53, S54, S56, S57, S61, S65, S66, S68, S70, S73, S74
Reliability	23	31.0	S2, S6, S8, S10, S13, S17, S18, S19, S20, S30, S31, S43, S48, S49, S52, S54, S55, S59, S61, S62, S65, S66, S71
Usability	18	24.3	S10, S17, S19, S24, S29, S30, S40, S41, S51, S52, S53, S56, S65, S66, S68, S72, S74
Interoperability	15	20.2	S10, S15, S16, S17, S19, S35, S39, S40, S42, S46, S47, S50, S51, S54, S72
Adaptability	11	14.8	S1, S7, S10, S15, S20, S28, S28, S39, S42, S61, S74
Modifiability	7	9.4	S7, S10, S12, S29, S40, S44, S56
Generality	7	9.4	S7, S10, S12, S19, S22, S29, S51
Privacy assurance	7	9.4	S15, S19, S25, S30, S36, S42, S49
Security	4	5.4	S10, S49, S53, S73
Stealthiness	3	4.05	S9, S21, S51

Performance is a measure of how quickly a system responds to an event [31]. Performance is characterized by different properties such as response time, throughput, and latency. Response time is a highly critical attribute of BDCA systems [S1], [S17], [S20], which are expected to respond in real-time. It can be difficult for BDCA systems to achieve the desired level of real-time response as the security event data is usually huge [S1], [S7], [S9], [S26], [S48] and is generated at a very high speed [S3], [S46]. The collected data need to be pre-processed to ensure data quality. The data dimensionality needs to be reduced [S44] and the data should be converted from binary format to text format [S16]. The current threat landscape requires a widespread correlation and contextualization of security event data to detect sophisticated attacks designed to execute over a period of time. Such threats need complex computation that can make real-time analytics a challenging task [S19], [S26].

Accuracy is a measure to which a system provides correct results with the needed degree of precision [32]. To make it specific to security analytics, it is the ratio of the total number of correctly detected attacks to the total number of attacks [S31]. As evident from Table 6, accuracy is a highly critical quality of BDCA systems. A failure in detecting an attack can lead to catastrophic outcomes [33]. An accurate BDCA system is expected to detect and resist attacks without thwarting legitimate access requests (i.e., false positives) [S1], [S46].

Scalability is a measure of how easily a system can grow to handle more user requests, transactions, servers, or other extensions [31]. Table 6 reveals that around 54% of the reviewed papers consider scalability as an important quality attribute for BDCA systems, which are expected to be highly scalable because the volume and velocity of security event data are quite unpredictable [34][S7]. BDCA systems analyse constantly growing size of security event data to detect and thwart sophisticated cyber attacks (e.g., APTs) [S17], [S31], [S74]. For example, the Operation Shady RAT attack started in 2006 and was detected in 2011 when it had affected around 72 different organizations worldwide. Organizations are forced to monitor more and more sources (e.g., database access and users' activities) to deal with sophisticated attacks [S56]. Some of the reported challenges in achieving scalability include: (a) inefficient communication among a large number of processors in a distributed setup [S57]; (b) unfair load-balance among the computing nodes [S57] and (c) the choice of centralized data storage [S50].

Reliability is a measure of how long a system runs before experiencing a failure [31]. There are 23 reviewed papers (Table 6) that highlight the importance of reliability for BDCA systems, which may be vulnerable to a number of failures such as a data processing node can crash because of large data size [S1], [S18]. Reliable data collection is also quite crucial for security analytics [S33]. By reliable data collection, we mean that the data collector should cope with the speed of data so that all security event data (e.g., NetFlow data [35]) can be captured. It is quite possible that a single NetFlow may contain information significant for detecting an attack and letting such a NetFlow go uncaptured and unprocessed means letting the attack go undetected. It is worth mentioning that the traditional data collectors (e.g., Wireshark) are lagging behind in efficiently collecting data under peak conditions (e.g., DoS attack) [S63]. The inherent design limitations of software and hardware also need some attention while designing a BDCA system as it may lead to several types of malfunctioning [S71].

Usability is a measure of how easy it is for people to learn, remember, and use a system [31]. Considering the spontaneous nature of cyber attacks where delaying the response by a few seconds can be consequential, it is important that a BDCA system is user-friendly [S10]. A BDCA system should provide user-friendly functionality to visualize security incidents (i.e., threat alerts) so that the required mitigation action can be taken [S17], [S29], [S41], [S58], [S66]. Such a system should also incorporate some mechanism that enables a security administrator to focus on the most dangerous alerts [S41] out of hundreds of alerts.

Interoperability is a measure of how easily a system can interconnect and exchange data with other systems or components [31]. Table 6 shows that around 20.2% of the reviewed papers consider interoperability as an important quality attribute for BDCA systems. Unlike other data-intensive systems, the interoperability requirement is more critical for such systems, which need to be integrated together to enhance the overall security spectrum of an organization [S14], [S17]. For example, a collaborative IDS enables multiple IDSs to collaborate with each other for detecting sophisticated attacks [36]. A BDCA system connects to a variety of sources (e.g., network devices and host machines) for data collection for which limitation on interoperability can be problematic. Motivated by its data-intensive nature, a BDCA system needs to interoperate with a variety of databases such as Oracle, MySQL, and MsSQL [S10], [S16], [S47]. Moreover, a BDCA system should also be able to integrate with specialized hardware (e.g., GPU) for achieving fast data processing capability [S35].

Adaptability is the measure of how easily a system adapts itself to different specified environments using only its own functionality [32]. It is expected that a BDCA system automatically adjusts itself to various kinds of changes as apparent from our finding shown in Table 6. A BDCA system incorporating multiple data sources deals with different data formats (such as formatted text and binary data) so the system should be able to automatically adjust itself with data format without affecting the overall performance of a system [S1], [S39], [S74]. Moreover, a system should automatically change the security policies according to the requirements [S10]. For example, more in-depth monitoring of insiders during working hours as compared to non-working hours. Similarly, the network of an organization frequently experiences changes (e.g., change in network topology), therefore, a BDCA system should adjust itself to the changes in a network environment [S15].

Modifiability is a measure of how easy it is to maintain, change, enhance, and restructure a system [31]. Modifiability relates to manual modification by a user while adaptability is the automatic adjustment by a system itself without any involvement of a user. Several changes are beyond the control of a system itself and require an input from a user. In the context of security analytics, a number of different modifications are required from time to time. A signature-based BDCA system needs to be updated with the latest and emerging attack patterns [S7]. Similarly, with the advancement of algorithms and computational models, a system needs to be modified to employ the advanced algorithms and models [S10], [S12], [S56]. A BDCA system should be flexible enough to easily incorporate the upcoming tools and technologies [S17], [S40].

Generality is a measure of the range of attacks covered by a security system [S7]. Table 5 indicates that different BDCA systems target the detection and prevention of different types of attacks. It is also mentioned in the application domains (i.e., Section 3.4) that some of the BDCA systems (e.g., IDS, alert or correlators) are generic and cover a wide variety of attacks. BDCA systems are designed by considering a trade-off between generality and specificity; the latter types of systems are more accurate [37] but less flexible to deal with different types of attacks. Several of the reviewed papers ([S7], [S12], [S19], [S22]) assert that a BDCA system has to be generic enough to counter the variety, complexity, and sophistication of different types of attacks. If an organization's security experts believe that their organization is vulnerable only to a particular type of attack (e.g., botnet attack), then a specific security solution should be preferred over a generic solution.

Privacy assurance is the measure of the ability of a system to carry out its business according to defined privacy policies [38]. Several of the reviewed papers ([S15], [S19], [S25], [S30], [S36], [S42], [S49]) highlight the importance of privacy assurance for security analytics. In the context of BDCA, privacy assurance is quite critical because a number of BDCA systems capture, store, and process packet payload, which contains personal data of users. For example, one of the reviewed papers i.e., [S17] employs a technique called content inspection [39], which in addition to the header information, also capture, store, and process payload of network packets. According to the European data laws and regulations [40], it is not allowed to store and process the packet payload without the relevant users' consent.

Security is the measure of how well a system protects itself and its data from unauthorized access [31]. A BDCA system must also itself be secured to ensure the security of an organizational IT infrastructure. Otherwise such systems can be hacked for modifying the monitoring rules or the data transitioning between two services (e.g., data collection and data processing). Such unauthorized modifications can negatively affect a security system's abilities to safeguard organizational IT infrastructure; such have been highlighted in [S10], [S49], [S53], [S73].

Stealthiness is the measure of a system's ability to function without being detected by an attacker [37]. If an attacker can figure out the whereabouts of a security system, he/she is very likely to first focus on disabling the security system. It is important that a BDCA system operates in a stealthy mode, whereby an attacker does not even know that a security system is in place for monitoring an attacker's activities. In our review, only three papers (i.e., [S9], [S21], [S51]) highlight the significance of stealthiness for a BDCA system.

5. Architectural tactics

This section reports the architectural tactics for BDCA systems discovered from the reviewed papers. This part of our study is meant to answer RQ2, "*What are the architectural tactics for addressing quality concerns in BDCA systems?*" We have identified the key components of a BDCA system and arranged them in way to form a so-called reference architecture so that the details of each of the codified tactics can be reported and understood in the context of a BDCA system's architecture. We also present the template used for codifying the tactics.

Reference Architecture

A Software Reference Architecture (SRA) captures the essence of architectures of similar systems (i.e., systems belonging to the same class or domain) [41]. A SRA helps increase the overall understanding of a particular class of systems (e.g., BDCA systems) and provides guidelines for developing a concrete architecture. The elements (e.g., tools, modules, and tasks) shown in the BDCA SRA (Figure 7) have been identified from the concrete architectures of the BDCA systems reported in the reviewed papers. It is to be noted that the various elements employed in a BDCA system architecture are not limited to what is shown in Figure 7. This SRA is reported as a diagrammatic guide to help understand the key role of each tactic in a typical architecture of a BDCA system. The representation for various elements (e.g., green rectangle for data collection module) is separately shown in the legend in Figure 7. We follow the same representation for reporting each of the tactics, so a reader can relate each tactic to the SRA, whose elements are explained below.

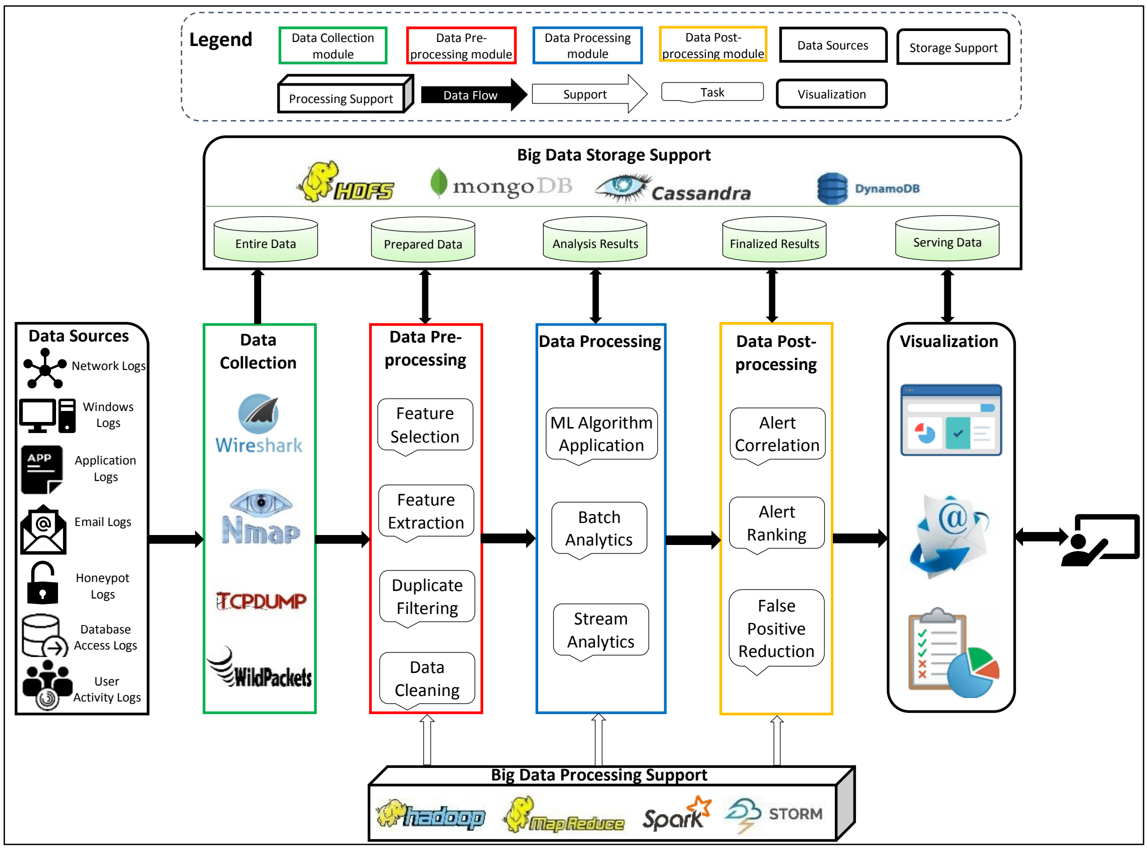


Figure 7. Reference Architecture for BDCA Systems.

Data Sources include the sources that generate rich data, which can be collected and analyzed for detecting and preventing cyber attacks. There are several options [42] (e.g., windows logs, NetFlow data, and email logs) that generate valuable data for security analytics. The choice of source(s) varies from enterprise to enterprise.

Data Collection interfaces a system with the external data sources. This module uses various tools [42] (e.g., Wireshark and Gulp) to collect data from the specified data sources and stores it in a Data Storage in its entirety.

Data Pre-processing prepares data for subsequent modules, especially data processing. The pre-processing can include several tasks such as feature selection and extraction, removal of duplicates and bad records, data validation and standardization. The data is received from the Data Storage and prepared data is stored back in the Data Storage as shown in Figure 7.

Data Processing module leverages big data technologies for extracting valuable insight about cyber attack. It should be noted that big data technologies (e.g., Hadoop, Spark, or MongoDB) are not limited to what is shown in Figure 7 rather we direct a reader to [43] for the details.

Data Post-processing module improves the results generated by the Data Processing module by applying various techniques such as correlating and ranking alerts. Like Data Processing, this module communicates with the Data Storage and is supported by the big data technologies for realization of the associated tasks.

Visualization module leverages various tools (e.g., dashboard, text or graphic report, and email notifier) to communicate the finalized results (e.g., security alerts) to security experts.

Big Data Storage Support manages the distribution and back and forth storage of security data after being processed in each module. The module not only manages data storage, but also maintains various procedures followed by different modules to access and transform data. The module uses different data storage tools (e.g., HDFS, MongoDB, Cassandra) to support the required data storage.

Big Data Processing Support manages the distribution of the data processing among computing nodes. The module uses big data framework (e.g., Hadoop, Spark, or Storm) to distribute the data processing. As shown in Figure 7, the module supports data pre-processing, data processing, and data post-processing modules to distribute the processing among the computing nodes.

Tactics Template

The tactics have been codified using the following template.

- *Introduction:* brief explanation of how tactic achieves the desired quality attribute;
- *Motivation:* rationale behind why the tactic needs to be incorporated in an architecture design;
- *Description:* detailed explanation on how the various components of a system interact to achieve the desired quality attribute and the architecture diagram highlighting the components related to a tactic;
- *Constraints:* necessary conditions for incorporating the tactic in the existing architecture of a system;
- *Example:* one or more systems from the reviewed papers demonstrating the application of the tactic;
- *Dependencies:* whether or not a tactic depends upon other tactic(s) for its incorporation in a system;
- *Variation (optional):* slightly modified form of the original tactic

Unlike [15], the codification of our tactics includes the details about constraints, examples, and dependencies. This is because our codified tactics are identified and extracted from the literature (similar to [16] and [17]), which enabled us to extract and report such details. We assert that the inclusion of constraints, examples, and dependencies in a tactic's description are quite useful in understanding the impact and limitations of each tactic. Given we use the same diagrammatic style for reporting all the tactics, we have to modify the diagrams in a way that can best explain the specific tactic instead of explaining the entire architecture of a BDCA system. For example, the component responsible for feature selection and extraction is not explicitly shown in tactics other than the Feature Selection and Extraction tactic (Section 5.1.3). It does not mean that this component does not necessarily exist in the diagrams for other tactics. Similarly, the rationale behind showing the data collection as a distributed process (in Data Ingestion Monitoring tactic (Section 5.4.1) and Secure Data Transmission tactic (Section 5.5.1)) and as a centralized process is to suit the tactic and explain its key points. Furthermore, we use the term component instead of module (as used in SRA that partitions the system into implementation units [15]) during the description of tactics. This is because we want to illustrate the runtime interactions among the architectural elements for accomplishing various tasks.

Codified Tactics

An architectural tactic is a design strategy that influences the achievement of a quality attribute [15]. Figure 8 shows the codified tactics associated with various quality attributes. We have identified and codified six tactics for performance, four for accuracy, two for scalability, three for reliability, and one for security and usability each.

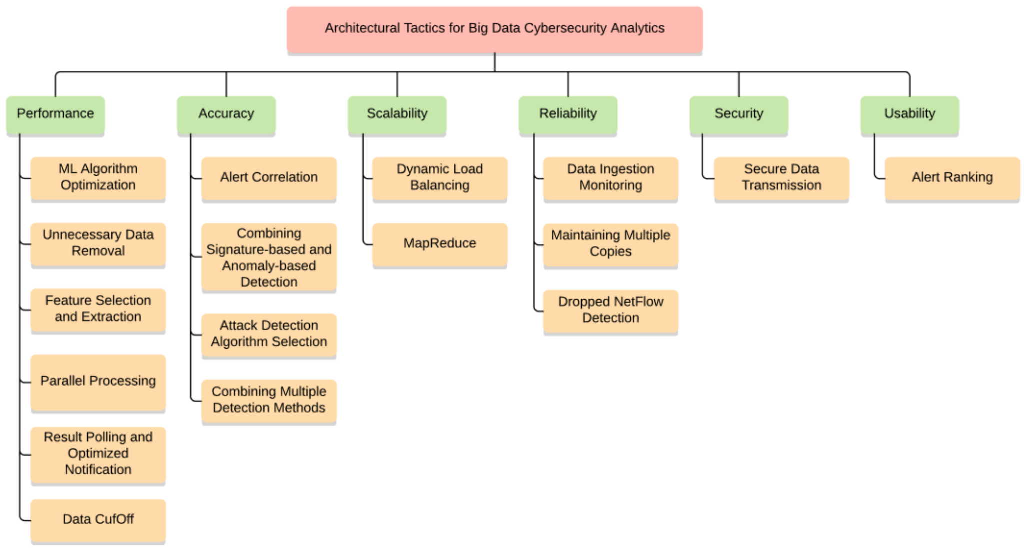

Architectural Tactics for Big Data Cybersecurity Analytics
Performance
Accuracy
Scalability
Reliability
Security
Usability
ML Algorithm Optimization
Unnecessary Data Removal
Feature Selection and Extraction
Parallel Processing
Result Polling and Optimized Notification
Data CufOff
Alert Correlation
Combining Signature-based and Anomaly-based Detection
Attack Detection Algorithm Selection
Combining Multiple Detection Methods
Dynamic Load Balancing
MapReduce
Data Ingestion Monitoring
Maintaining Multiple Copies
Dropped NetFlow Detection
Secure Data Transmission
Alert Ranking

Figure 8. Architectural tactics for BDCA Systems classified based on the relevant quality attributes

5.1. Performance

This section reports the architectural tactics related to Performance quality attribute.

5.1.1. ML algorithm optimization,

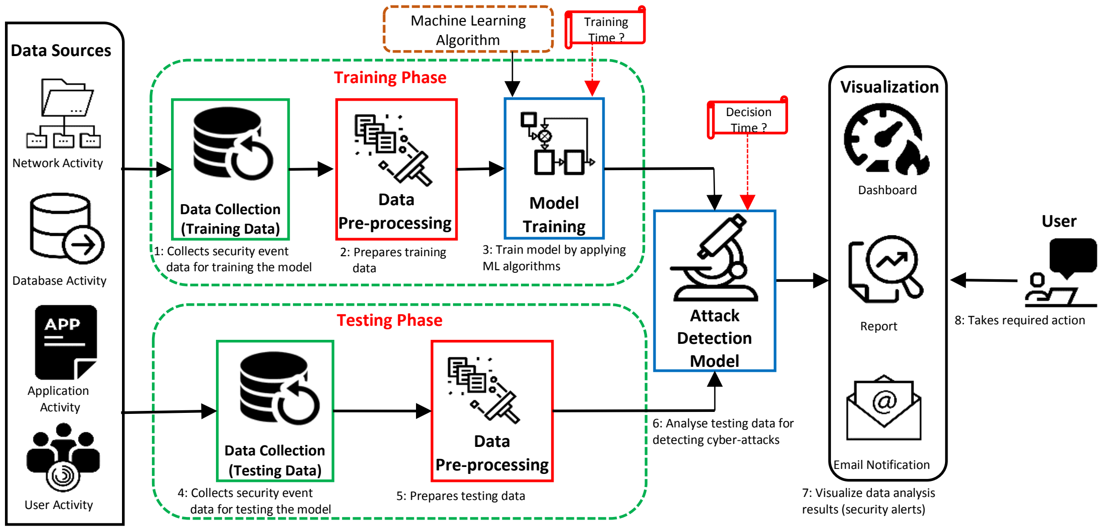
Introduction. The ML Algorithm Optimization tactic has been found in all the papers as all of the BDCA systems leverage Machine Learning (ML) algorithm for analyzing the security event data. This tactic guides the selection of a suitable algorithm for achieving computational efficiency for improving a system's performance.

Motivation. The two most important factors related to the performance of BDCA are input data type and ML algorithm [44]. The ML algorithms range from supervised learning (e.g., Logistic Regression, Support Vector Machine, Naïve Bayes, Random Forest, and Decision Trees) to unsupervised learning algorithms (e.g., K-means and Neural Networks) [44]. An ML algorithm can be selected based on several factors such as time complexity, incremental update capability, offline/online mode, and generalization capacity of the algorithm, and the potential impact of the algorithm on the detection rate (accuracy) of a system.

Description. The main components of a ML Algorithm Optimization tactic are shown in Figure 9[4]. The *data collection* component collects security event data for training a BDCA system. The training data can be collected from sources within an enterprise as depicted in Figure 9 or an already available dataset such as KDDcup99. The *data preparation* component prepares the data for training the model by applying various filters. The selected *ML algorithm* is applied to the prepared training data to train an attack detection model. The time taken by the algorithm to train a model (i.e., training time) varies from algorithm to algorithm. Once the model is trained, it is tested to investigate whether the model can detect cyber attacks. For testing the model, data is collected from an enterprise as shown in step 4. The testing data is filtered through the *data preparation* component and fed into the *attack detection model*, which analyzes the data for detecting attacks. The time taken by an *attack detection model*

[4] Source of icons in the diagrams: https://thenounproject.com/

to decide whether a particular stream of data pertains to an attack (i.e., decision time) depends upon the employed algorithm. The result of the data analysis is displayed to the user through *visualization* component.



Data Sources
Network Activity
Database Activity
APP
Application Activity
User Activity
Machine Learning Algorithm
Training Time ?
Training Phase
Data Collection (Training Data)
1: Collects security event data for training the model
Data Pre-processing
2: Prepares training data
Model Training
3: Train model by applying ML algorithms
Decision Time ?
Attack Detection Model
Testing Phase
Data Collection (Testing Data)
4: Collects security event data for testing the model
Data Pre-processing
5: Prepares testing data
6: Analyse testing data for detecting cyber-attacks
Visualization
Dashboard
Report
Email Notification
7: Visualize data analysis results (security alerts)
User
8: Takes required action

Figure 9. ML algorithm optimization tactic

Constraint. Several things need to be considered while selecting and using an optimized algorithm.

- An ML algorithm's performance may not be consistent in different domains [45]. Therefore, an ML algorithm may perform well for one type of security analytics (e.g., detecting DoS attack) but may not perform well in another type of security analytics (e.g., detecting brute force attack).
- The selection of an algorithm is challenging in the sense that in addition to performance, it affects other system's qualities such as accuracy, complexity, and understandability of the final result. For example, Cheng et al. [46] compare SVM with Extreme Learning Machine (ELM) in terms of accuracy and performance. It is observed that SVM generates more accurate results but is computationally expensive. On the other hand, ELM generates less accurate results but is more light-weight. A reasonable trade-off should be established among various system's qualities while selecting an algorithm.
- The selection of algorithm also depends upon the working mode (online or offline) of BDCA system. An algorithms having a time complexity less than O(n^3) are considered acceptable for online mode while algorithms with a complexity of O(n^3) and above are slower and suits only offline analysis mode [44].

Example. As mentioned all of our included systems leverage ML algorithms. We report the findings from a couple of papers to exhibit the role of an optimized algorithm in improving the performance of BDCA system.

- Spark-based IDS Framework [S9]: This BDCA system compares the training time and decision time of five ML algorithms namely Logistic regression, Support vector machine, Random forest, Gradient boosted decision trees, and Naïve Bayes. With KDDCup99 dataset, Naïve Bayes shows the best training time (i.e., 79.5 sec) and SVM shows the worst training time (i.e., 479.12 sec). On the other hand, with respect to decision time, SVM shows the best decision time (i.e., 10 sec) and Gradient boosted decision tree shows the worst decision time (i.e., 22.2 sec).
- Ultra-High-Speed IDS [S14]: Here, the BDCA system is tested with six ML algorithms to investigate the training time and decision time of each algorithm. The algorithms are Naïve Bayes, SVM, Conjunctive rule, Random Forest, J48, and RepTree. It is found that both in terms of training time and decision time RepTree is the most efficient followed by J48.
- Cloud-based Threat Detector [S10]: The system has been implemented with two ML algorithms – k-mean and Naïve Bayes, to explore the training time taken by both algorithms. It is observed that with 500 GB, k-means takes around 60 secs while Naïve Bayes takes around 92 secs to train a model.

Dependencies. ML Algorithm Optimization tactic requires Unnecessary Data Removal tactic (Section 5.1.2) and Feature Selection and Extraction tactic (Section 5.1.3) to help bring collected data into a refined form. After the application of these tactics, ML Algorithm Optimization tactic can be efficiently applied to the refined data to quickly train a system and detect attacks. ML Algorithm Optimization tactic needs to be incorporated along with

Attack Detection Algorithm Selection tactic (Section 5.2.3) as these two tactics establish the trade-off between the effects of ML algorithm on performance and accuracy.

5.1.2. Unnecessary data removal

Introduction. Unnecessary Data Removal tactic has been found in four of the reviewed papers ([S52], [S58], [S66], [S67]). This tactic removes the unnecessary data from the dataset of security event data that is supposed to be processed by the data processing component of a system to detect cyber attacks. The subset of security event data that does not contribute to the detection process is termed as unnecessary data. The removal of such data from the dataset reduces the size of the dataset, which decreases the processing time.

Motivation. Security-critical data is collected from a variety of sources within an enterprise to detect cyber attacks. However, not all of the collected data contributes to the detection process. For example, a network sniffer captures zero-byte data that is not useful in detecting cyber attacks as zero-byte flows are primarily used for handshaking in a TCP/IP connection [47]. Therefore, such useless data should be removed from the rest of the data captured by a network sniffer.

Description. Figure 10 shows the various components of Unnecessary Data Removal tactic. *Data collector* component collects security event data from various sources within an enterprise. The *data storage* component stores the collected data. Before forwarding data for analysis, data is intercepted by the *data cleaning* component, which filters the data and removes unnecessary data from the dataset. After the unnecessary data is removed, the rest of the data is forwarded to the *data processing* component that analyzes the data to detect cyber attacks. Finally, the analysis results are visualized through the *visualization* component.

Constraints. This tactic requires that the data cleaning functionality is not computationally expensive, otherwise, it would cancel the benefit of quick response acquired through data size reduction. The rules for filtering out unnecessary data need to be designed carefully to ensure that data that is critical for accurately detecting cyber attacks does not get filtered out. These data filtration rules vary from situation to situation as data that is of significant value in one situation may not be significant in another situation [S67].

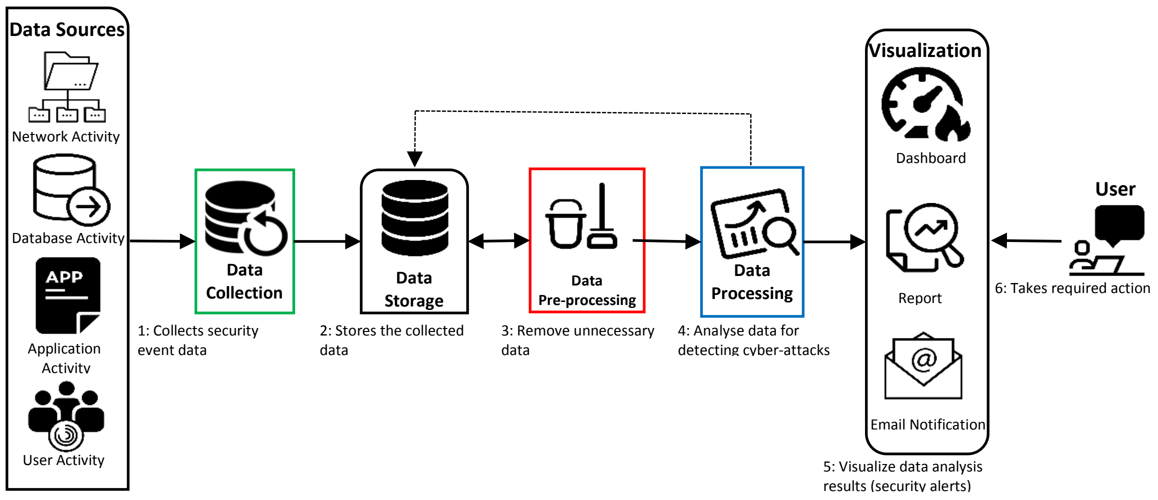


Figure 10. Unnecessary data removal tactic.

Example. The following two systems incorporate Unnecessary Data Removal tactic.

- Dynamic Time Threshold [S58] analyzes security event data collected from various sources that include web access log, network log, host log, and network behaviour log. This system incorporates Unnecessary Data Removal tactic to remove log instance with URL suffix JPG, GIF, JPEG, WAV, CSS and JS. These log records automatically get stored in the web access log when a user requests a web page related to JS, CSS, or image data. However, these log records do not contribute to the threat analysis process, therefore, these are considered unnecessary and are removed.
- Batch and Stream Analyzer [S66] collects and analyze NetFlow data for detecting cyber attacks. This system employs Unnecessary Data Removal tactic for removal of zero-byte flows captured in the NetFlow data. As mentioned, zero-byte flows are used for handshaking in TCP/IP connection and have no relevance to threat detection, therefore, such data is removed from the rest of the Netflow data.

Dependencies. Unnecessary Data Removal tactic requires Parallel Processing tactic (Section 5.1.4) to speed up the process of removing unnecessary data from the raw data collected from different sources.

Variations: Removal of duplicates. This tactic as described removes data that does not contribute to attack detection process. However, the collected data also contains a lot of duplicate records, whose analysis puts an extra burden on the computational process without any valuable contribution to the detection process. Therefore, specific representative instances of such records should be included, and duplicates should be removed before forwarding data to the analysis component. The removal of duplicates not only supports fast processing but also gives a clearer view of the activities directed towards or within a network [S6]. Compression Model for IDS [S21] implements this tactic on training data for IDS. The system uses affinity propagation [48] to remove duplicates from the training data and so extract a small dataset from a large-scale dataset. It was found that the incorporation of removal of duplicate tactics along with horizontal compression enhances the efficiency by 184 times with less than 1% negative effect on the attack detection accuracy. Multistage Alert Correlator [S6] incorporates this tactic for removing duplicate alerts in an alert correlation system designed to correlate individual alerts to determine attack patterns that can be used for predicting future attacks.

5.1.3. Feature selection and extraction tactic

Introduction. Feature Selection and Extraction tactic has been found in [S9], [S10], [S12], [S13], [S14], [S15], [S31], [S32], [S33], [S55], [S57]. This tactic selects and extracts the most relevant features from the network traffic data that can be analyzed for detecting cyber attacks. The incorporation of this tactic helps reduce storage volume, increase data processing speed, and reduce data complexity. This tactic not only improves the performance of a system but also contributes to the detection accuracy improvement.

Motivation. According to a Cisco report, global IP traffic was 1.2 Zettabytes in 2016, which is expected to reach 3.3 Zettabytes in 2021 [49]. Each IP traffic record contains more than 40 features (such as source IP address, destination IP address, source port, and destination port). For instance, the traffic records contained in Knowledge Discovery in Databases cup 1999 (KDDcup99) dataset [50] consist of around 41 features while traffic records in Centre for Applied Internet Data Analysis (CAIDA) dataset [51] has around 50 features. Many enterprises do not have the high computational capability for such a large size of data (using all available features). It is important to carefully select and extract specific features from the captured network traffic for analysis.

Description. Figure 11 shows the major components of Feature Selection and Extraction tactic. The *data collection* component collects network traffic data from various sources (e.g., switch, router, or firewall). Each record in the captured data contains more than 40 features from which the specific features for each IP traffic record using the feature selection component. The *feature extraction* component extracts the specific features. The extracted feature dataset is passed onto the data processing component that helps detect cyber attacks, that lead to the generation of alerts presented to users through the *visualization* component. Once such attack alerts come under notice, a user or enterprise can take necessary steps to prevent or mitigate the effects of the attack.

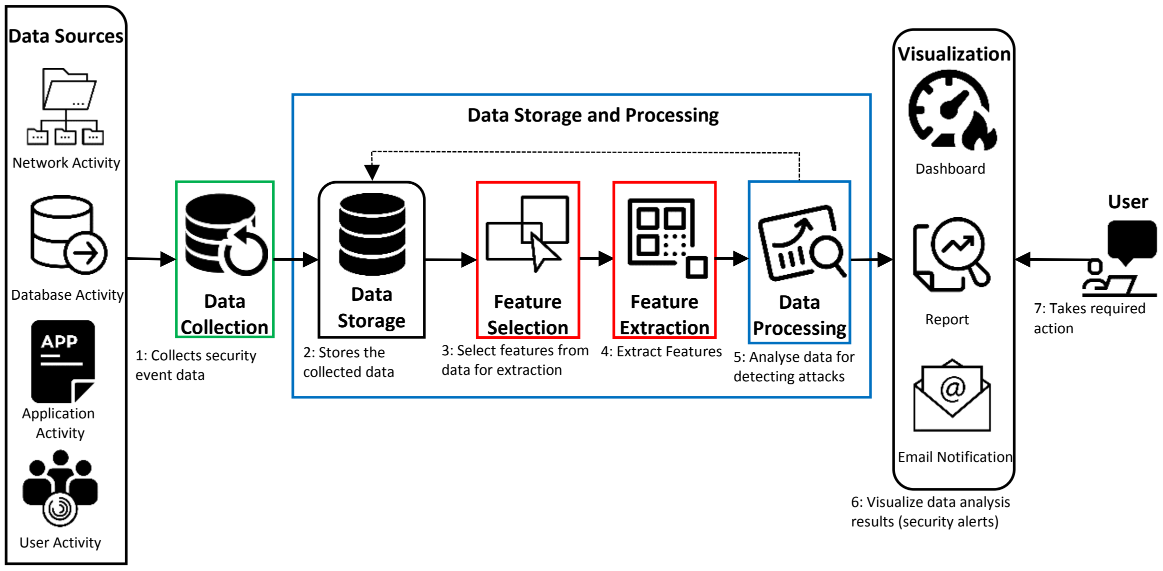


Figure 11. Feature selection and extraction tactic

Constraints. The Feature Selection and Extraction tactic facilitates real-time security analytics. There must be a careful approach to ensure that any feature that contributes to the detection process should not be missed, otherwise, a system's accuracy would be severely affected. This condition becomes more relevant in cases where the network traffic is constantly evolving due to emerging technologies and new attacks. The feature selection and extraction operations should not be computationally heavy, otherwise, it would cancel the benefit of the reduced response time achieved through the reduction in data size.

Example. The following two implementations demonstrate the Feature Selection and Extraction tactic.

- Cloud-based Threat Detector [S10] collects and analyze various types of data (e.g., system logs, firewall logs, and router logs) to detect cyber attacks such as DDoS attack and port scanning attack. This system leverages Naïve Bayes algorithm to select five features out of 50 features in CAIDA dataset. These features include source IP address, destination IP address, port, packet length, and protocol. After selecting the features, parallel processing framework (i.e., MapReduce) is used to extract the features from the records. The incorporation of Feature Selection and Extraction tactic reduces the size of dataset from 200 GB to 50 GB. It was found that incorporation of this tactic does not have any negative impact on the detection accuracy of a system.
- Quasi Real-Time IDS [S12] leverages Information Gain Ranking Algorithm to select eight features from network traffic records in CAIDA dataset. After selecting the features, parallel processing framework (i.e., MapReduce) is used to extract the features from the records. A distinguishing characteristic of this system is that it also allows user to select features manually at runtime. This flexibility is helpful in situations where the dynamics of network traffic changes frequently. Such manual selection of features is achieved through the incorporation of Tshark [52] and Apache Hive[5]. Tshark enables a user to select the features and Hive provide the table for storing the features.

Dependencies. Feature Selection and Extraction tactic requires Parallel Processing tactic (Section 5.1.4) to speed up the process of feature selection and extraction. It is worth noting that Unnecessary Data Removal tactic (Section 5.1.2) and Feature Selection and Extraction tactic are not interchangeable. Both of these tactics play separate roles in improving the performance of the BDCA system.

5.1.4. Parallel processing

Introduction. Parallel Processing tactic can be found in all (i.e., 74) of the reviewed studies. This tactic distributes the processing of a large amount of security event data among different nodes of a computing cluster. The nodes process the data in parallel fashion, which significantly improves the response time of a system.

Motivation. There are a number of sources within an enterprise that generate security event data. These sources include but not limited to network devices (e.g., switches and routers), database activities, application data, and user activities. Security event data is generated at a very high speed. For example, an enterprise as large as HP used to generate around one trillion security events per day in 2013, which was expected to grow further in the future years [3]. A standalone computer that processes such a large size of security event data in a sequential manner will take a lot of time to detect an attack, which is not tolerable in such security-critical situations.

Description. Figure 12 shows the main components of Parallel Processing tactic. The numbers in the figure show the sequence of operations. The *data collecton* component collects security event data from different sources depending on the type of security analytics and security requirements of an enterprise. The *data collectorn* forwards the collected data to *data storage* component, which stores the data. Data can be stored in several ways such as Hadoop Distributed File System (HDFS), HBase, and Relational Database Management System (RDBMS). In order to enable parallel processing, the stored data needs to be partitioned into fixed-size blocks (e.g., 64MB or 128MB). After partitioning, data is processed in the *data processing* component through several nodes working in parallel according to the guidelines of a distributed framework such as Hadoop or Spark. The result of analysis is shared with users through the *visualization* component.

[5] https://hive.apache.org/

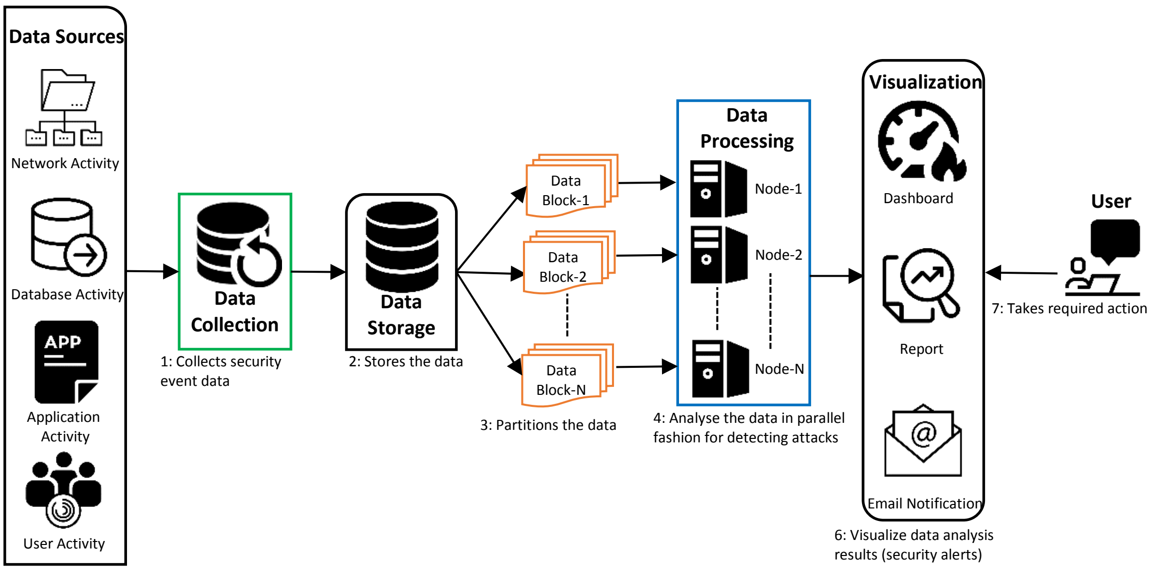


Figure 12. Parallel processing tactic

Constraints. Parallel Processing tactic assumes that a BDCA system incorporating this tactic is already integrated with a cluster of nodes capable of processing data in a parallel fashion. Another important factor that needs to be taken care of is the breaking of a logical record across two blocks during the partitioning of data into blocks. It is important to keep enough information about the file data type so that record can be reconstructed.

Example. Honeypot-based Phishing Detection [S17] demonstrates the applicability of this tactic for improving the response time of a phishing attack detection system. The authors compared the response time of a sequentially implemented system with a parallel-implemented system, each processing 268 GB of security event data. It was found that sequentially implemented system took 180 minutes to process all data. On the other hand, parallel implementation of a system with Hadoop and Spark frameworks took 21 minutes and 14 minutes respectively with a cluster of 9 nodes. The authors also demonstrated that the more the number of nodes in a parallel processing scenario, the faster will be the response time of a system. For example, the response time of Hadoop was recorded as 57, 36, and 21 minutes with 3, 5, and 9 nodes respectively.

Dependencies. Parallel Processing tactic depends upon the Dynamic Load Balancing tactic (Section 5.3.1) and Data Ingestion Monitoring tactic (Section 5.4.1) for balancing the load among the nodes and controlling the flow of data into the nodes respectively.

5.1.5. Result polling and optimized notification

Introduction. Result Polling and Optimized Notification tactic is found in Count Me In [S56]. This tactic helps optimize the delay caused due to a predefined time interval for feeding the results from the mapper nodes into the reducer nodes inside a parallel processing BDCA system. This tactic ensures that as soon as values inside the mapper nodes change to a sufficient degree (set by an admin), the mapper node notifies the reducer node and accordingly forward the updated results to the reducer node.

Motivation. MapReduce is a parallel processing framework that is widely adopted in a distributed setup [53]. This framework consists of two phases – (1) Map and (2) Reduce. The Map phase maps the features derived from the network traffic to a key and a value through multiple mapper nodes. For example, failed connections between a source ‘s’ and a destination ‘d’ can be presented in the form of a key-value pair (s, d). In the Reduce phase, the key-value pairs generated by mapper nodes are fed into multiple reducer nodes for generating results. The generated results are evaluated against a predicate threshold through a trigger and if the results exceed a specific limit, then an alert is generated that signal towards a possible cyber attack.

The key-value pairs from mapper nodes are fed into reducer nodes after a predefined time interval (e.g., 5-minute aggregation or 1-hour aggregation). Once reducer generates the results, the trigger executes the predicate threshold. This predefined time interval introduces a delay. For example, an attack may be launched at t=5 sec of the predefined time interval and the trigger will be executed at t=5 min. In this 5 min, it is quite possible that a significant damage might already have been caused.

Description. The incorporation of Result Polling and Optimized Notification tactic in a BDCA system is demonstrated in Figure 13. *Data collection* component collects security event data from multiple sources. The collected data is stored by the *data storage* component. Next, *parallel data processing* component reads the stored data. The *parallel data processing* component consists of mappers and reducers. The number of mappers and reducers depends upon the number of nodes in an underlying cluster. The mapper sub-components (e.g., Mapper Node-1 and Mapper Node-2) read the data in parallel and produce intermediate key-value pairs shown as result in Figure 13. When all the map jobs are completed, the key-value pairs (shown as values in the figure) are passed on to the reducers, which merge the values to produce the final results. Typically, the reducers have to wait until all map jobs are completed, which causes a significant delay in responding to cyber attacks.

To address this issue, the mechanism of the poll and notify is introduced. According to this mechanism, the mappers monitor the changes in incoming security event data. As soon as the change in the security event data crosses a predefined threshold, the mappers notify the reducers to get ready for taking the required data without waiting for completion of the predefined time interval. In addition to the optimized notification from mappers, the reducers can poll the mappers for results depending upon the processing capacity of the reducer(s). After receiving the intermediate key-value pairs, the reducers process the intermediate results to generate the final results. Immediately after the reducers produce the final results, the trigger is executed to check for possible cyber attacks.

Constraints. In a distributed setup, it is important that the threshold set for change in intermediate values at mapper nodes is applied to the average of all mapper nodes. For example, if a change in values at a single mapper node crosses the defined threshold, but the rest of the mapper nodes are not experiencing sufficient change in values, intermediate results should not be passed to the reducer nodes. This is due to the reason that by-passing the defined time interval and forwarding unchanged intermediate results from mappers to reducers will put extra processing burden on reducers without any significant gain.

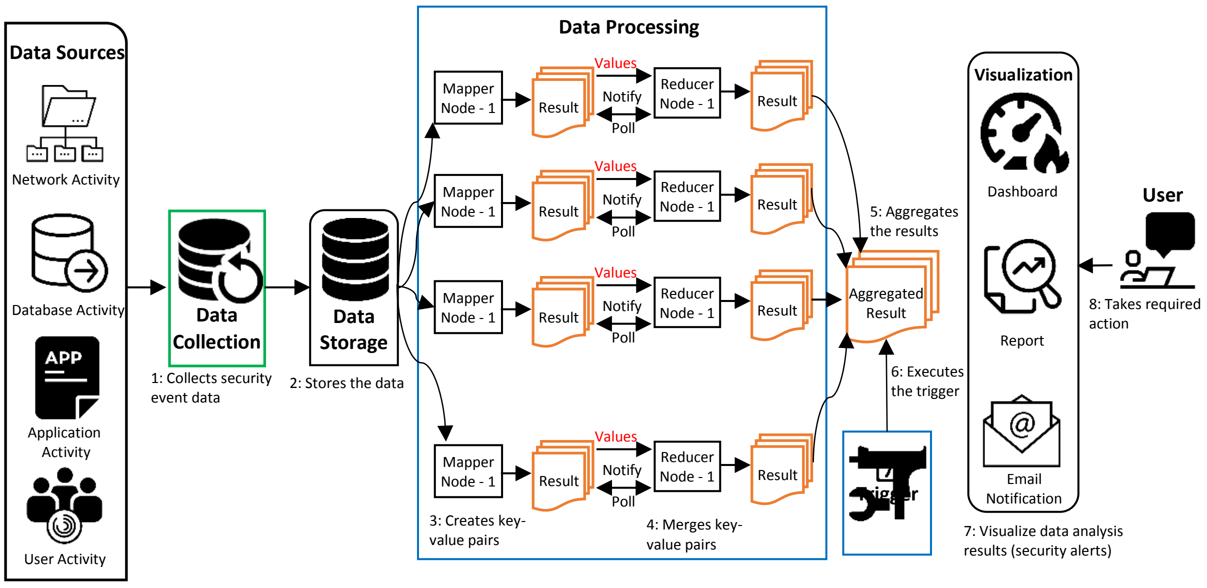


Figure. 13. Result polling tactic and optimized notification tactic.

Example. Count Me In [S56] experimentally examines the communication overhead caused by the incorporation of this tactic. The system monitors an uplink traffic averaging 1 Gb/sec during day-time hours. It is observed that out of the total messages communicated among the nodes of a system, only 0.40% relates to Result Polling and Optimized Notification tactic. In total out of 69, 810 notifications from mapper node to reducer nodes, the reducer nodes ignore 27, 704 notifications to limit simultaneous outstanding key updates.

Dependencies. Result Polling and Optimized Notification tactic requires Parallel Processing tactic (Section 5.1.4) to enable the reducer nodes to share the extra load generated due to result polling among the nodes. This tactic also depends upon the MapReduce tactic (Section 5.3.2) for providing the programming framework to design mappers and reducers.

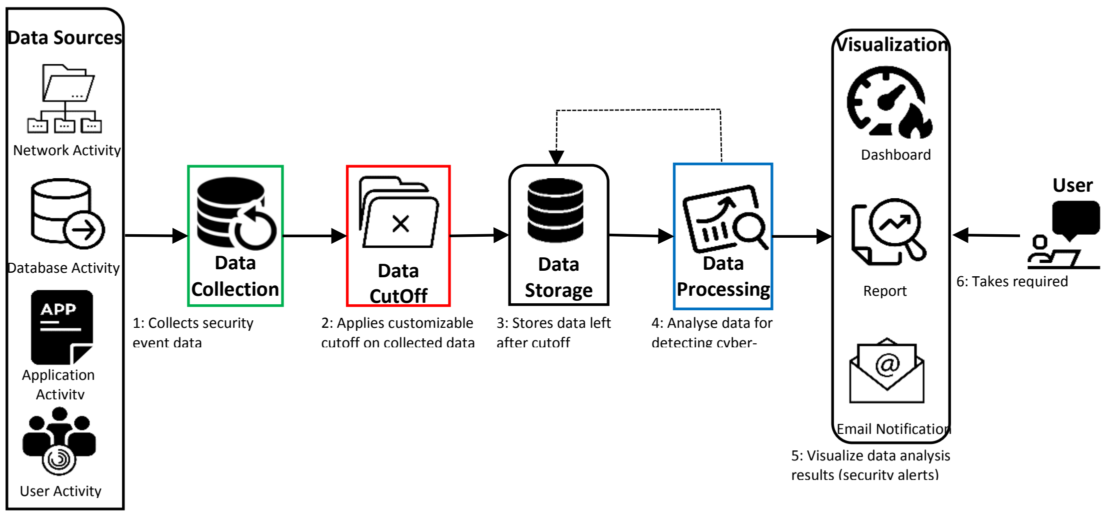
Variations: User-guided poll and notify. Result Polling and Optimized Notification tactic requires either the mappers to notify the reducers about updates or requires reducers to poll the mappers for updates. However, in some systems, the task of poll and notify can be relegated to a user. A user can initiate the process of sending intermediate key-value pairs from mappers to reducers at any point of time irrespective of the predefined time interval. For example, SEAS-MR [S38] has both the options – periodic aggregation and user-guided aggregation of intermediate results produced by mappers. The user-guided poll and notify can also be used for long-term analysis where a user can specify the start time and the end time for an interval. In this system, periodic aggregation is implemented with MapReduce as it is better for performance while user-guide aggregation is implemented with Pig[6] script as it facilitates user interactivity.

5.1.6. Data Cut-off

Introduction. Data Cut-off tactic has been identified from Forensic Analyzer [S30] and VALKYRIE [S45]. This tactic applies a customizable cut-off limit on each network connection or process to select and store data pertaining to a specific portion of the connection or process. For example, selecting and storing only first 15 KB of network traffic data for a connection or data pertaining to first 100 sec of the execution time of a process. Such a cut-off reduces size of the dataset for security analysis, which helps improve the overall performance of the system.

Motivation. Due to the ever-increasing volume of security relevant data (e.g., network traffic, system logs, and application activity), it is infeasible to collect, store, and analyze the data in its entirety. For example, Lawrence Berkeley National Laboratory (LBNL), a security research lab containing around 10,000 hosts, experiences around 1.5 TB of network traffic per day. In majority of the cases, only a small subset of the security event data turns out to be relevant for security analysis [54]. In case of network connections, more connections are short with few large connections accounting for bulk of total volume [55, 56]. Thus, by selecting and storing the first N bytes (cut-off) for each large connection, which can be stored in its entirety. Such connections' beginning portions contain the relevant information such as protocol handshakes, authentication logs, and data item names.

Description. Figure 14 shows the main elements of Data Cut-off tactic with the numbers to indicate the sequence of the operations. The *data collection* component collects security event data from one or several available sources and passes the collected data to *data cut-Off* component. Examples of security events include network security event (e.g., [source IP, destination IP, port, protocol]) and process event (e.g., [file name information, privilege level, parent process ID, timestamp]). The *data cut-Off* component enforces a cut-off by discarding security events that appear after a network connection or process has reached its predefined limit. Any security event that appears after the predefined limit does not contribute significantly to the attack detection process, therefore, analyzing such security events put an extra burden on data processing resources without any significant gain. The security event data left after cut-off is stored by the *data storage* component. The stored data is read by *data processing* component to analyze it for detecting cyber attacks. Finally, the result of analysis is displayed to a user through *visualization* component. A user takes the required action upon arrival of any outstanding alerts.



[6] https://pig.apache.org/

Figure. 14. Data cut-off tactic

Constraints. Data Cut-off tactic poses a risk of evading an attack from detection. If an attacker is smart enough to initiate an attack after the cut-off limit, a BDCA system will not record the malicious activity, and so the attack will go undetected. There are several delaying tactics usually employed by attackers with the intent to evade such tactics that record only the initial portion of an interaction. Several compensation techniques can be employed to reduce the risk of such an evasion. These techniques include: (1) selecting cut-off limit according to the type of data sources as delaying an attack to later stages is harder for some services as compared to others (2) the cut-off limit can be increased for data sources where there is a higher risk of an attack (3) the application of cut-off can be randomized instead of always being applied at the start of an interaction (e.g., network connection or a process) so that an attacker cannot predict at which point the cut-off will come into play. In addition, the functionality required for implementing Data Cut-Off tactic should not be computationally expensive, otherwise, it will cancel the benefit that is expected through the incorporation of this tactic.

Example. Data Cut-Off tactic has been implemented in Forensic Analyzer [S30] and VALKYRIE [S45].

- Forensic Analyzer [S30] is a cloud-based network security forensic analysis system that is evaluated by detecting phishing attacks based on analysis of captured network traffic. The data collected in a span of six months is around 20 TB in size. Data Cut-Off tactic was applied to the collected data with a cut-off limit of 15 KB. This means to only select the first 15 KB of each network connection. The incorporation of this tactic reduced the size of data from 20 TB to 1 TB. The effect of data cut-off on detection accuracy of a system has not been reported.
- VALKYRIE [S45] is a BDCA system that detects malware attacks based on an analysis of kernel-level telemetry data. The system implements Data Cut-Off tactic to reduce the size of data by selecting the data pertaining to only first 100 seconds of the execution time of a process. Despite the incorporation of Data Cut-Off tactic, the system achieves detection accuracy of around 97-99%.

Dependencies. Data Cut-Off tactic requires Parallel Processing tactic (Section 5.1.4) to speed up the process of discarding security event data that appears after the predefined cut-off limit.

5.2. Accuracy

This section reports the architectural tactics that are related to Accuracy quality attribute.

5.2.1. Alert Correlation

Introduction. Alert Correlation tactic have been found in [S6], [S7], [S11], [S16], [S19], [S38], [S39], [S41] and [S65]. This tactic analyzes the individual alerts produced by security system(s), discards the irrelevant alerts, and groups together the relevant alerts based on a logical relationship between them to provide a global and condensed view of the security situation of an organization's infrastructure (i.e., network and hosts). The incorporation of this tactic improves the accuracy of a BDCA system by reducing the number of false positives and detecting highly sophisticated and complex attacks.

Motivation. Organizations employ different security technologies for better detection coverage of their networks and hosts. For example, a typical organization may deploy firewall, anti-virus, and IDS for accepting/dropping network traffic, scanning malware based on predefined signature, and detecting known attack patterns or abnormal behaviours respectively. Unfortunately, these security tools generate a large number of alerts. For instance, an IDS deployed in a real-world network generates around 9 million alerts per day [57]. Investigating and responding to these many alerts is quite challenging especially when 99% of them are false positives [58]. Furthermore, these security tools monitor the security in isolation without considering the context and logical relationship between alerts generated by other security tools. This isolated approach is not able to detect attacks that operate in slow mode over a period of time where some alerts are precursors to more complex and dangerous attacks. To address these issue, a high-level management is required that correlate the alerts by taking context and their logical relationship into consideration before reporting them to users.

Description. Figure 15 shows the main components of implementing Alert Correlation tactic. The *data collection* component collects security event data from different sources. The collected data is stored in the *data storage* and copied to the *data pre-processor* component for pre-processing the raw data. The pre-processed data is ingested into the *alert analysis* component, which analyzes the data for detecting attacks. It is worth mentioning that Alert analysis component analyzes the data in an isolated fashion (without considering any contextual information) either using misuse-based analysis or anomaly-based or both. The generated alerts are transferred to the *alert verification* component, which uses different techniques [59] to determine whether an alert is a false positive. The alerts identified as false positives are discarded at this stage. The clean and synthesized alerts are forwarded to the *alert correlation* component. The alerts are correlated (i.e., logically linked) using different techniques [59] such as scenario-based correlation, rule-based correlation, statistical correlation, and temporal correlation. The Alert correlation component coordinates with *data storage* for taking the required contextual information about alerts. The results of correlation are released through the *visualization* component. Finally, either an automated response is generated, or a security administrator analyzes the threat and responds accordingly.

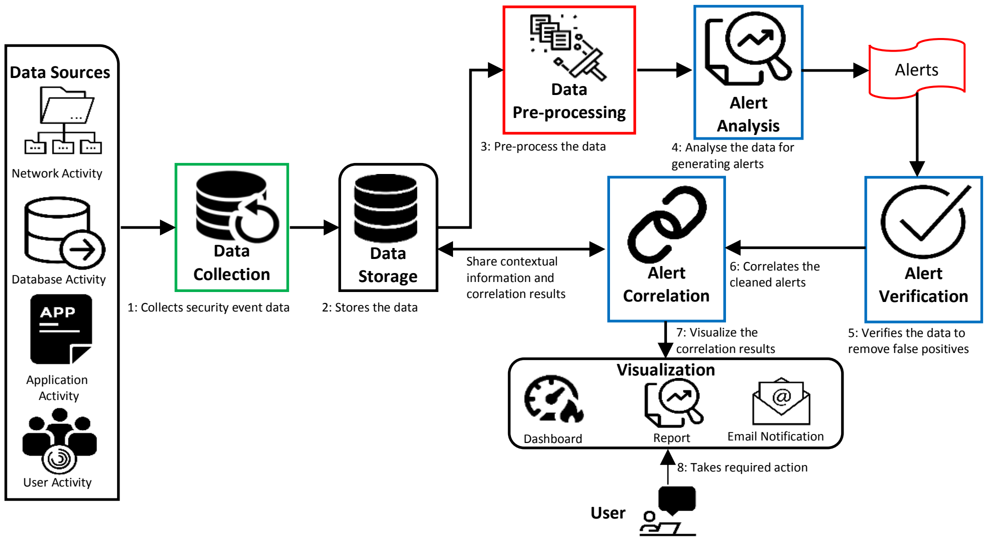


Figure 15. Alert correlation tactic.

Constraints. The in-depth analysis employed for alert correlation improves the accuracy but increases the response time due to the inclusion of an extra complex computation stage to a system. The tactic also requires mechanisms for acquisition of domain knowledge and adaptation to changes in the networks and host infrastructure [59].

Examples. This review has identified nine systems that implement Alert Correlation tactic. These systems collect individual alerts from different security technologies (e.g., IDS or Firewall) for detecting attacks. Each of them applies different correlation techniques, some of which have been described below.

- GSLAC [S7]: The system employs causal-based technique for correlating alerts. Each alert is treated as a vector with multiple attributes (i.e., destination IP, source IP, timestamp and so on). The alerts are presented in the form of a graph where each alert has a prerequisite alert and a consequence. A security analyst analyzes the graph for identifying complex attack scenarios.
- Hunting attacks in the dark [S41]: This system correlates alerts based on their source and destination IP addresses. The similarity between IP addresses for different alerts is measured using intersection cardinalities and if the similarity score is above a predefined threshold, it means alerts belong to same IPs that signals towards a potential attack.
- Multistage alert correlator [S6]: This system employs a model of prerequisites and consequences proposed in [60] for determining the relationship among individual alerts. The alerts are correlated based on similarities among their source IP, destination IP, start time, and end time. A graph is generated that shows each alert with its prerequisite and consequence. If a prerequisite of an alert is present in the graph as a consequence of a previous alert, the two alerts are closely related and analyzed for picturizing the complex attack scenario.

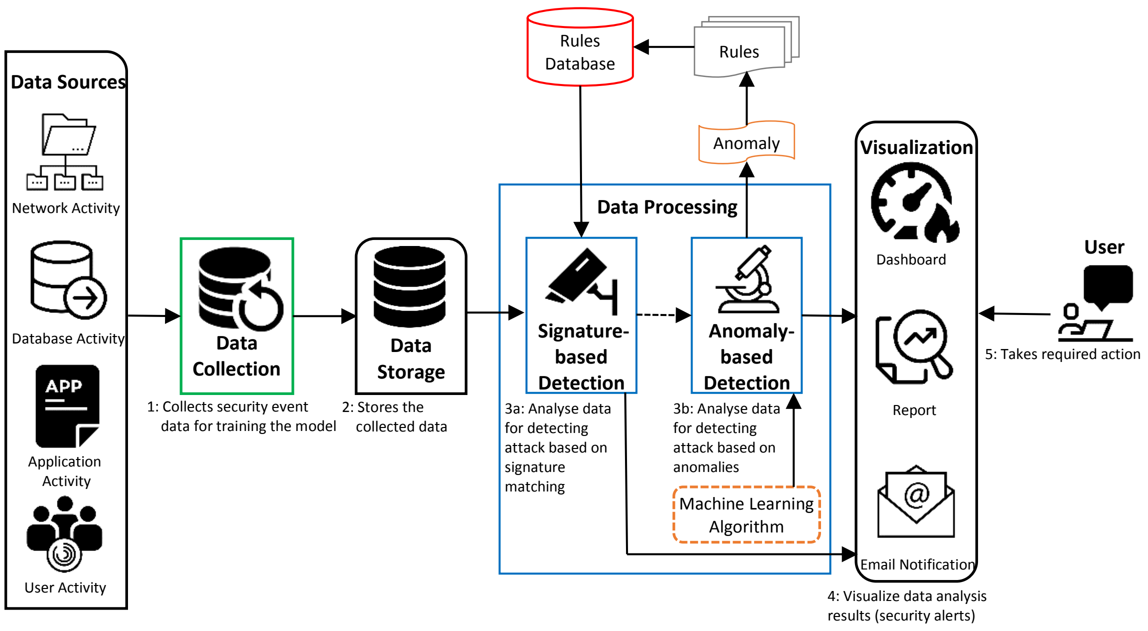
Dependencies. Alert Correlation tactic will correlate alerts of any quality; however, effective correlation requires alerts to be of a good quality that is dependent upon the tactics employed in the data processing component such as Attack Algorithm Selection tactic (Section 5.2.3) and Combining Signature-based and Anomaly-based Detection (Section 5.2.2).

5.2.2. Combining signature-based and anomaly-based detection

Introduction. Genetic Algo-based Distributed Denial of Service (DDoD) Detection [S61], Hybrid Intrusion Detection [S72], and Snort + PHAD + NETAD [61] employ this tactic. The Combining Signature-based and Anomaly-based Detection tactic enables a BDCA system to analyze the collected security event data for two objectives: (1) find a match with the already available attack patterns or signatures and (2) find a deviation from the learned normal behaviour i.e., an anomaly. In either case, finding a match or a deviation, a system generates an alert signalling towards a possible cyber attack. The combination of misuse and anomaly significantly improves the detection accuracy and reduces the false positive rate.

Motivation. Based on detection principle, BDCA systems are of two types – Signature-based (often called misuse-based) and Anomaly-based. Signature-based systems detect attacks based on predefined attack patterns. These patterns are designed based on already reported attacks. If an ongoing activity matches with attack patterns, the activity is termed as malicious. These types of systems are very effective in detecting known attacks but are unable to detect unknown attacks [62]. Anomaly-based systems learn the normal behaviour of an organization's infrastructure and any activity that deviates from this behaviour is termed an attack. This class of systems can detect unknown attacks, however, it generates a large number of false positive alarms [63]. Considering the limitations of both type of systems, it is important to come up with a solution that can minimize these limitations.

Description. The main components of Combining Signature-based and Anomaly-based Detection tactic are shown in Figure 16. The *data collection* component collects security relevant data from different sources. The collected data is stored by the *data storage* component. Next, data is fed into the *signature-based detection* component that analyzes the data to identify attack patterns. For such analysis, this component leverages the pre-designed rules from the *rules database* that define attack patterns. If a match is identified, an alert is directly generated through a *visualization* component. If *signature-based detection* component does not detect any attack pattern in the data, the data is forwarded to the *anomaly-based detection* component for detecting unknown attacks that cannot be detected by the *signature-based detection* component. The *anomaly-based detection* component analyzes the data using ML algorithms to detect deviations from the normal behaviour. When an anomaly (deviation) is detected, an alert is generated through the *visualization* component. At the same time, the *anomaly* is defined in the form of a *rule* or attack pattern and added to the *rules database*. This way the *rules database* is constantly updated to enable the *signature-based detection* component to detect a variety of attacks.



Rules Database
Rules
Anomaly
Data Sources
Network Activity
Database Activity
APP
Application Activity
User Activity
Data Collection
1: Collects security event data for training the model
Data Storage
2: Stores the collected data
Data Processing
Signature-based Detection
3a: Analyse data for detecting attack based on signature matching
Anomaly-based Detection
3b: Analyse data for detecting attack based on anomalies
Machine Learning Algorithm
Visualization
Dashboard
Report
Email Notification
4: Visualize data analysis results (security alerts)
User
5: Takes required action

Figure 16. Combining signature-based and anomaly-based detection tactic

Constraints. Combining Signature-based and Anomaly-based Detection tactic may impact the overall performance of a system by introducing additional data analytic requirements. Furthermore, combining signature-based and anomaly-based detections into a single BDCA system and getting them interoperate in an efficient and successful way can be more complex and challenging.

Example. The following two systems demonstrates the implementation of Combining Signature-based and Anomaly-based Detection tactic.

- Hybrid Intrusion Detection [S72]: This system combines signature-based IDS (Snort) with an anomaly-based detector implemented using Hadoop and Hive. The hybrid system is compared with the stand-alone signature-based system in terms of accuracy. Both the systems have been evaluated with several attacks such as ICMP, Smurf, SYN flood, UPD, and Port scanning attack. It is found that a hybrid system achieves higher detection rate for all attacks compared to a stand-alone system. For example, for port scanning attack, a stand-alone system achieves a detection rate of around 40% while a hybrid system shows a detection rate of around 50%.
- Snort + PHAD + NETAD [61]: This system combines signature-based IDS (Snort) with two anomaly detectors among which one (PHAD) detect anomalies based on packet header analysis and other (NETAD) detects anomalies based on network traffic analysis.. The hybrid system is evaluated using IDEVAL dataset [64] to explore its detection rate in comparison to the stand-alone signature-based system. It is observed that the stand-alone system is able to detect only 27 attacks while the hybrid system (i.e., Snort+PHAD+NETAD) detects 146 attacks from the dataset.

Dependencies. Combining Signature-based and Anomaly-based Detection tactic requires Parallel Processing tactic (Section 5.1.4) to reduce the additional analysis time required due to two-phase analysis (i.e., signature-based and anomaly-based). Additionally, this tactic also depends upon Attack Detection Algorithm Selection tactic (Section 5.2.3) to efficiently select an effective algorithm for detecting anomalies in the security event data.

5.2.3. Attack detection algorithm selection tactic

Introduction. We have already stated that all of the reviewed systems leverage some type of ML algorithm to analyze the security event data for detecting cyber attacks, therefore, this tactic can be found in almost all of the reviewed systems. Unlike ML Algorithm Optimization tactic (Section 5.1.1) that emphasizes the role of ML algorithm on performance, the objective of Attack Detection Algorithm Selection tactic is to highlight the role of suitable algorithms in improving the accuracy of a BDCA system and provide some guidelines for selecting ML algorithms that are most appropriate for accurate detection of attacks.

Motivation. BDCA systems are expected to accurately detect cyber attacks without generating security alerts for legitimate activities (i.e., false positives). In addition to several other factors such as data source and data quality, the employed ML algorithms play a significant role in accurately detecting cyber attacks. BDCA systems use ML algorithms to classify the security event data either as legitimate (corresponding to normal activity) or malicious (corresponding to attacks). There exists a variety of ML algorithms that can be leveraged to detect attacks. These algorithms include but not limited to Logistic Regression, Naïve Bayes, Support Vector Machine, Random Forest, and Gradient Boosted Decision Tree, K-means, and Artificial Neural Networks. For details on ML algorithms used in cybersecurity analytics, readers should refer to [44]. The big challenge here is to determine which of the many available algorithms will yield the most accurate results.

Description. The major components of Attack Detection Algorithm Selection tactic are shown in Figure 17. The *data collection* component collects security event data for training the BDCA system for detecting cyber attacks. The training data can be collected from sources within an enterprise where a system is supposed to be deployed as depicted in Figure 18 or an already available popular dataset such as KDDCup99 can be used as training dataset. How much data should be sufficient to train the model varies from one case to another. For example, the famous DARPA 1998 dataset contains data collected from network and operating system in a span of nine weeks. The first seven-weeks data were assigned as training data and the last two weeks as testing data. After collecting the training data, the *data preparation* component prepares the data for training the model by applying filters and feature extraction techniques. Next, the prepared training data start training the attack detection model. Once the model is trained, it is tested to investigate whether the model can detect cyber attacks. For testing the model, the data is collected from an enterprise as shown in step 4. The test data is prepared for feeding into the attack detection model. The prepared test data is forwarded to the *attack detection model*, which analyzes the data based on the rules learned during the training phase. Here, the incoming test data instances are classified as either legitimate or malicious. The analysis results are displayed to a user through the *visualization* component. In case of malicious or attack situation, a user can take immediate actions that may include blocking certain ports or cutting off the affected components from the network to stop further damage.

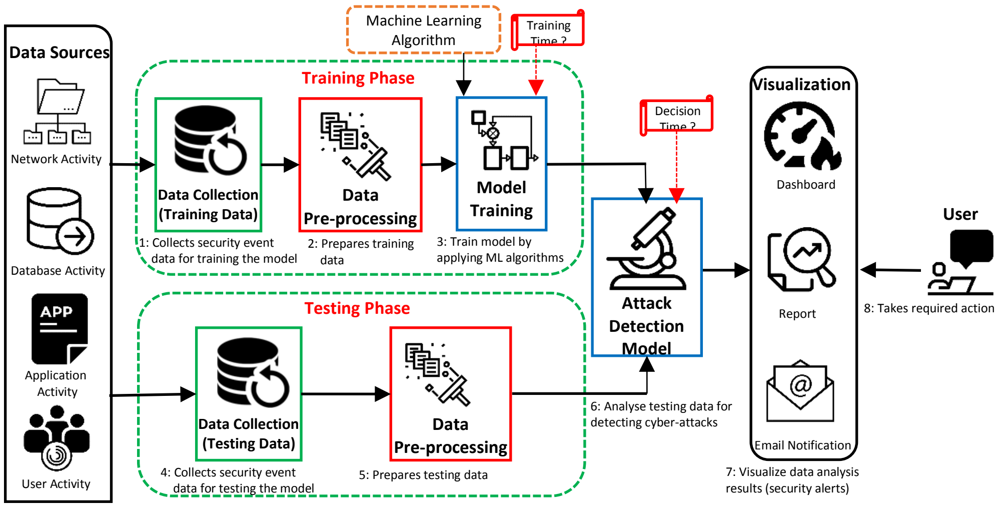

Data Sources
Network Activity
Database Activity
APP
Application Activity
User Activity
Machine Learning Algorithm
Training Time ?
Training Phase
Data Collection (Training Data)
Data Pre-processing
Model Training
1: Collects security event data for training the model
2: Prepares training data
3: Train model by applying ML algorithms
Decision Time ?
Attack Detection Model
Testing Phase
Data Collection (Testing Data)
Data Pre-processing
4: Collects security event data for testing the model
5: Prepares testing data
6: Analyse testing data for detecting cyber-attacks
Visualization
Dashboard
Report
Email Notification
7: Visualize data analysis results (security alerts)
User
8: Takes required action

Figure 17. Attack detection algorithm selection tactic

Constraints. While selecting an attack detection algorithm, software architect should keep an eye on several things due to the following reasons

- The selection of ML algorithm is tricky because an algorithm may work well for detecting one type of attack but may not work well for detecting other types of attacks
- Similarly, algorithm selection is challenging in the sense that in addition to accuracy, it affects other system's qualities such as response time, complexity, and understandability of the final result. For example, Cheng et al. [46] compare SVM with Extreme Learning Machine (ELM) in terms of accuracy and performance. It has been revealed that SVM generates more accurate results but is computationally expensive. On the other hand, ELM generates less accurate results, but is more light-weight. A reasonable trade-off should be established among various system's qualities while selecting an algorithm.
- There is no such standard available that compares the accuracy of all ML algorithms on the same dataset. Different research explorations use different datasets (e.g., KDD99 and DARPA 1999) or different subsets from the same dataset, which makes it harder to fully rely on these findings for selecting an algorithm for a specific BDCA system.
- The applicability of ML algorithms closely relates to the quality of the data, whose quality ought to be carefully assessed for selecting a suitable attack detection algorithm.
- The selection of algorithm also depends upon the working mode of a BDCA system [44]. An algorithm that best fits for offline analysis may not be suitable for online analysis.

Example. Whist all of the reviewed systems leverage ML algorithms, we report only those systems that explore and compare the effects of multiple algorithms on the attack detection rate of a system. The objective is to exhibit the importance of algorithm selection for the detection rate of a BDCA system.

- Streaming-based Threat Detection [S1]: This system explores the applicability of K-means clustering algorithm [65] and Fuzzy C-means clustering algorithm [66] for accurately detecting cyber attacks. The system is evaluated to detect DDoS and flooding attacks based on analysing 260 GB of network traffic data gathered at Chicago Equinix data centre [51]. Experimental results show that detection rate for K-means is 91.8% with 1.8% false positive while for Fuzzy C-means the detection rate is 86.5% with 2.7% false positive.
- Improved K-means IDS [S27]: K-means algorithm is improved by specifying criteria for selecting initial centre of cluster which is random in traditional K-means algorithm. Both K-means and improved K-means are implemented in IDS for comparing detection accuracy. It has been found that improved K-means accurately detects 89% of the attacks with 2.4% false positive in KDDCup99 dataset. On the other hand, K-means shows an accuracy of 86% with 1.0% false positive rate.
- Spark-based IDS Framework [S9]: This framework compares five ML algorithms: Logistic regression, Support vector machine, Random forest, Naïve Bayes, and Gradient boosted decision tree. Two datasets, KDDCup99 and NSL-KDD, are used to evaluate the accuracy of the system with each of the algorithms. With KDDCup99, Logistic regression shows the best accuracy (i.e., 91%) and SVM shows the worst

(i.e., 78%). With NSL-KDD, the best accuracy is achieved with Random Forest (i.e., 82.3%) while SVM delivered the worst accuracy rate of 37.8%.

Dependencies. Attack Detection Algorithm Selection tactic depends upon Unnecessary Data Removal tactic (Section 5.1.2) and Feature Selection and Extraction tactic (Section 5.1.3) to help bring the collected data into a refined form. After the application of Unnecessary Data Removal and Feature Selection and Extraction tactic, attack detection algorithm can be efficiently applied to the refined data to accurately detect cyber attacks. Attack Detection Algorithm Selection tactic needs to be incorporated alongside ML Algorithm Optimization tactic (Section 5.1.1) as these two tactics establish the trade-off between the effects of the ML algorithm on accuracy and performance.

5.2.4. Combining multiple detection methods

Introduction. Combining Multiple Detection Methods tactic has been used in Multiple Detection approaches [S46]. This tactic applies and integrates the results of multiple security analytic methods on security event data. The incorporation of multiple security analytic methods in a BDCA system reduces false positive alarm rate and improves the attack detection accuracy.

Motivation. Most of the BDCA systems employ a single detection principle for detecting attacks. Examples of detection principles include traffic volume to a server crosses a predefined threshold, a large number of port access in a certain period of time, and IP address that generate traffic above a certain threshold. The disadvantage of using a single detection approach is that a system usually generates a large number of false positives [S46], which can be reduced by using multiple detection approaches for the same security threat data. Each detector accommodates a different detection principle. For example, SYN flood attack can be detected by monitoring or analysing the number of network flows in network traffic data while a DoS attack can be detected by monitoring the volume of network traffic between source and destination IPs.

Description. The main components of Combining Multiple Detection Methods tactic are shown in Figure 18. Security event data is collected from different sources. It is worth noting that the sources from where security event data can be collected is not limited to what is shown in Figure 18. The choice of data sources varies from organization to organization depending upon their specific security requirements. After collection, data is stored in a *data storage*. Then the data is forwarded to the *data processing* module where various attack detection methods are applied to analyze the data. The number and choices of the attack detection approaches depend upon several factors, which include the processing capability of an organization, the data sources, security requirements, and an organization's security expertise. For example, a highly security-sensitive organization (e.g., National Security Agency) with high budget and computational power may incorporate more number of attack detection approaches to secure their data and infrastructure from cyber attacks. The attack detection approaches are applied to the whole dataset in a parallel fashion. The *visualization* component immediately reports any outstanding anomalies to users or administrators, who are expected to respond to security alerts.

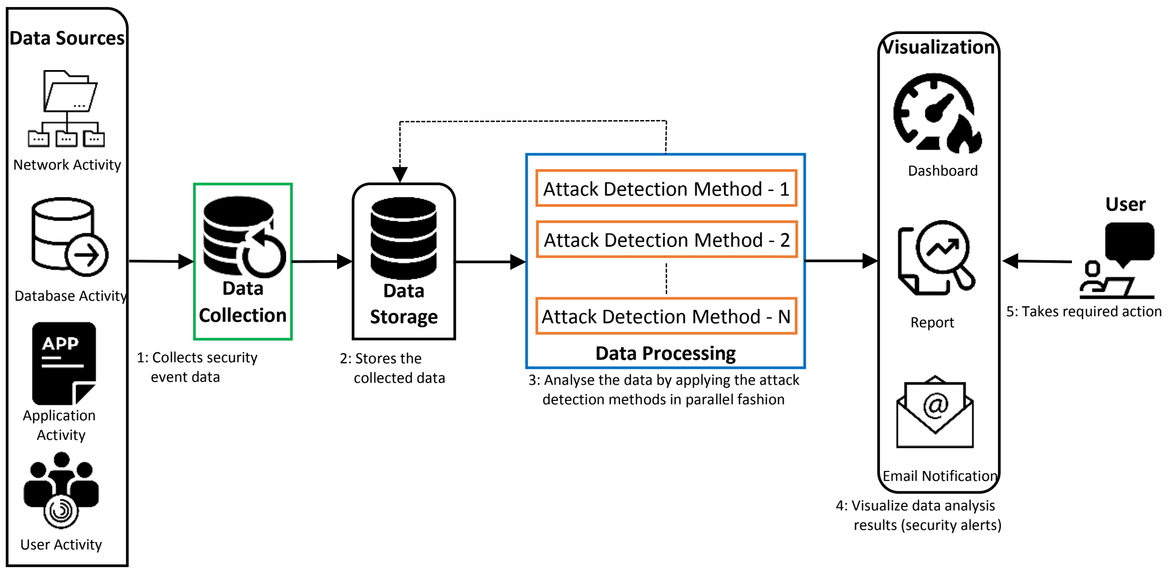

Data Sources
Network Activity
Database Activity
Application Activity
User Activity
APP
Data Collection
1: Collects security event data
Data Storage
2: Stores the collected data
Attack Detection Method - 1
Attack Detection Method - 2
Attack Detection Method - N
Data Processing
3: Analyse the data by applying the attack detection methods in parallel fashion
Visualization
Dashboard
Report
Email Notification
4: Visualize data analysis results (security alerts)
User
5: Takes required action

Figure 18. Combining multiple detection methods tactic

Constraints. Combining Multiple Detection Methods tactic requires more computational power as compared to a BDCA system that incorporates a single detection approach. If a system uses multiple detection approaches, it would need more computation power; this requirements can affect the overall performance of the system.

Example. Combining Multiple Detection Methods tactic has been implemented in Multiple Detection approaches [S46]. This system integrates three detection approaches for analyzing network traffic data. The first approach detects attacks based on finding anomalies with respect to the number of network flows. The second approach detects attacks based on finding anomalies with respect to the number of network packets. The third approach correlates the traffic volume with the IP flows to detect attacks. The system is tested with two types of attacks – DoS attack and SYN flooding attack. Under a DoS attack, the victim network experiences network flows that are large in size and small in number. On the other hand, under an SYN flooding attack, the victim network experiences network flows that are small in size but large in number. A BDCA system detects DoS attack through the second approach that detect anomalies based on number of network packets. On the other hand, SYN flooding attack is detected through the first approach that detects attacks based on number of network flows as the number of network flow during the attack situation crosses the pre-defined threshold.

Dependencies. Combining Multiple Detection Methods tactic depends upon Parallel Processing tactic (Section 5.1.4) that can provide the processing speed required for applying multiple attack detection methods on security event data. Without parallel processing, the system's response would be too slow.

5.3. Scalability

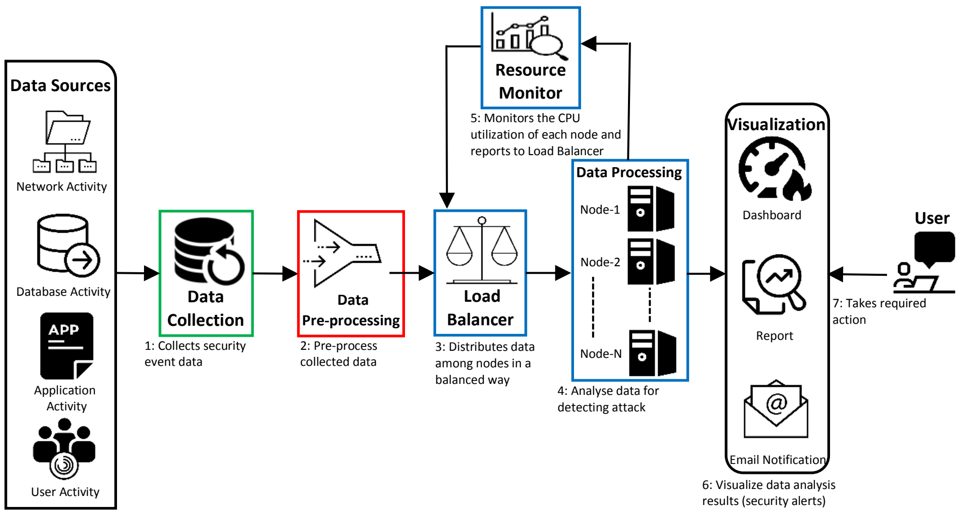
This section reports the architectural tactics that are related to Scalability quality attribute.

5.3.1. Dynamic load balancing

Introduction. Dynamic Load Balancing tactic can be found in GSLAC [S7] and Cloud Bursting [S34]. This tactic is used to balance the processing load among analysis nodes by dividing the security event data among the nodes. Having processing capacity available in a cluster, Dynamic Load Balancing tactic makes a system scale well without adding any further hardware resources.

Motivation. A BDCA system leverages a cluster of computing nodes for storage and analysis of security event data. The size of the cluster is different for different systems (e.g., 10 nodes in [S7] and 5 in [S34]). A system distributes the security event data among the nodes to speed up the process. When the speed of data input increases (for instance from 100 MB/sec in weekdays to 150 MB/sec over weekends), it is important that the system distributes the increased load in a balanced way to avoid a situation where one node is under extreme load (i.e., 100% CPU utilization) and another node is under-loaded (i.e., 30% CPU utilization).

Description. Figure 19 shows the main components of Dynamic Load Balancing tactic. The *data collection* component collects data from different sources. The captured data is sent to the *data filtration* component, which removes the data that does not contribute to the attack detection process. The filtered data is forwarded to a *load balancer*, which distributes the data among different nodes to balance the workload among the nodes in the *data processing* component. Data can be distributed based on different criteria. For example, network traffic can be distributed based on header information (e.g., IP address or TCP ports) or payload information (e.g., signatures). The balanced distribution of data is not fully reliable due to the heterogeneous nature of the security event data [67, 68]. For example, if network traffic is distributed based on IP range, one range may contain a greater number of packets with large size, which may exhaust one node. On the other hand, a node handling another IP range with few packets of small size might be sitting idle. Therefore, the *resource monitor* component is introduced that constantly monitors the CPU utilization of nodes and reports to the *load balancer*. When the difference between the CPU utilization of the nodes crosses a predefined threshold, the *load balancer* rebalances the load among the nodes by moving processing load from overloaded nodes to less loaded nodes. After successful analysis of the data in the *data processing* component, the results are shared with users through the *visualization* component.



Data Sources
Network Activity
Database Activity
APP
Application Activity
User Activity
Data Collection
1: Collects security event data
Data Pre-processing
2: Pre-process collected data
Resource Monitor
5: Monitors the CPU utilization of each node and reports to Load Balancer
Load Balancer
3: Distributes data among nodes in a balanced way
Data Processing
Node-1
Node-2
Node-N
4: Analyse data for detecting attack
Visualization
Dashboard
Report
Email Notification
6: Visualize data analysis results (security alerts)
User
7: Takes required action

Figure 19. Dynamic load balancing tactic

Constraints. This tactic assumes that a cluster has the processing capacity; otherwise, there is no chance of balancing the load if all nodes are already utilizing 100% of their CPU power. Furthermore, this tactic requires an efficient mechanism for selecting a target node and deciding the amount of data to be moved from the overloaded node to the target (under-loaded) node.

Examples. The following systems implement Dynamic Load Balancing tactic.

- GSLAC [S7]: During runtime, the resource monitor monitors the workload of the nodes and periodically updates the latency list that shows the CPU utilization of the nodes. When the difference between CPU utilization among the nodes crosses the threshold, the resource monitor alerts the load balancer. The load balancer uses dynamic hot spots migration [69] to rebalance the workload among the nodes.
- Cloud Bursting [S34]: This system leverages Dynamic Load Balancing tactic in a slightly different way. The resource monitor monitors the workload of the local cluster of nodes and keeps providing status information to a load balancer. Upon arrival of a new stream of security event data, the load balancer decides whether to launch the data processing job locally or burst it to other clusters on a cloud.

Dependencies. In principle, Dynamic Load Balancing tactic does not require any other tactic for its implementation. However, in a BDCA system, it will work in coordination with tactics like Unnecessary data removal (Section 5.1.2), Feature selection and extraction (Section 5.1.3), Data Cut-Off (Section 5.1.6), and Parallel Processing tactic (Section 5.1.4).

5.3.2. MapReduce

Introduction. This tactic has been found in all of the reviewed systems that use Hadoop platform for storing and analyzing security event data. MapReduce tactic provides a programming framework for scaling software applications across a cluster of multiple nodes. Although MapReduce is a well-documented programming framework, this tactic captures the contribution of MapReduce relevant to the enhancement of scalability. The tactic abstracts various issues of scalability that include complex parallelization, synchronization, and communication mechanisms. In addition to security, MapReduce is also a widely accepted framework in other big data analytics domains such as bioinformatics [70, 71], astronomy [72, 73], and healthcare [74].

Motivation. In order to scale-out a BDCA system for handling a huge amount of data, a simple solution is to add more hardware resource to a system. The additional hardware can be added to the same physical machine (vertical scaling) or it can be added as a separate physical machine (horizontal scaling). It is quite challenging to efficiently handle complex parallelization, synchronization, communications, and resource utilization with the addition of hardware resources to an existing system. Therefore, a framework is required to handle these outstanding issues.

Description. The main components of MapReduce tactic are shown in Figure 20. The *data collection* component collects the security event data from one or multiple sources. The collected data is forwarded to *data filtration* component for removing data that does not contribute to attack detection. The filtered data is partitioned by the *master nodes* to store it in the HDFS files of *data nodes*. The *mapper* inside each data node reads its assigned

HDFS block of the data for processing. It is important to note that the number of mappers does not depend upon the number of data nodes, rather, it depends upon the number of input data blocks. The mappers process data simultaneously in the form of key-value pairs (key, value) to generate intermediate results (key, values list). The intermediate results are sorted, collated, and passed to the *reducers*, which merge and aggregate the intermediate results to generate the final output. The number of mappers and reducers running simultaneously depends upon the capacity, workload, and users' recommendations. If a system experiences an increased data input, additional hardware resources (data nodes) can be easily added to a cluster to handle the increased workload. For more details on the architecture of Hadoop and MapReduce, readers can consult [75, 76].

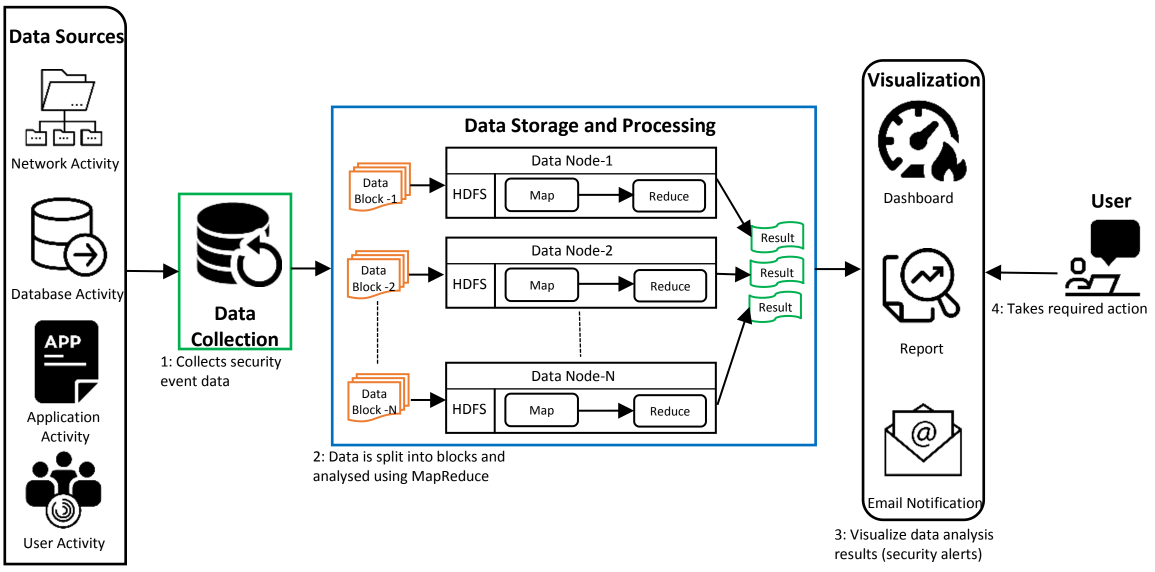


Figure 20. MapReduce tactic.

Constraints. This tactic assumes that additional hardware is available for horizontal scaling of a system. Furthermore, the introduction of overhead introduced due to disk read/write operations is a pressing issue with MapReduce, which can significantly prolong the response time of a BDCA system.

Example. MapReduce tactic is used by a number of the reviewed systems, however, we describe a few of them to illustrate its contribution in achieving scalability.

- Traffic Measurement and Analysis with Hadoop [S70]: This system uses heuristic algorithm [77] that enables mappers to read packet records from HDFS based on timestamp bit of a packet header. In order to evaluate scalability, 1 TB of security event data is analyzed by a cluster with varying number of nodes from 5 to 30. It is observed that system performance improves in proportion to the resource allocation. For example, the analysis completion time decreases from 71 minutes to almost 36 minutes as the number of nodes increases from five nodes to 10 nodes.
- IDS-MRCPSO [S31]: This system uses Particle Swarm Optimization clustering algorithm [78] with MapReduce. To investigate scalability, the system is implemented with different number of nodes. It is found that the speedup time increases linearly in the start from 2 to 8 nodes, but it starts to diverge from linear while moving from 8 to 16 nodes. This diversion is attributed to Hadoop framework i.e., starting MapReduce jobs and storing intermediate results.
- Extreme Learning Machine [S24]: This BDCA system combines the linear scaling capability of Extreme Learning Machine algorithm [46] and MapReduce. The system has been implemented with 15, 20, 25, and 30 nodes to evaluate its scalability. The system shows linear scaling from 15 to 25 nodes but diverges from linear scaling after 25 nodes. The authors argue that such diversion is experienced due to the increased communication between the nodes.

Dependencies. As such this tactic does not require any other tactic for its implementation.

5.4. Reliability

This section reports the architectural tactics that are related to Reliability quality attribute.

5.4.1. Data ingestion monitoring tactic

Introduction. Data Ingestion Monitoring tactic is found in Streaming-based Threat Detection System [S1], Malicious IP Detection [S18], and GPGPU-based IDS [S35]. This tactic monitors the flow of data from data collector into the computing servers in real-time data streaming systems. If the speed of data influx becomes too fast to be handled by a computing server, this tactic automatically blocks the flow of data into that server.

Motivation. Security system are expected to receive and handle a large amount of security event data. However, sometimes the security event data might be generated and collected at a speed that is beyond the capacity of a computing cluster. Such situation can lead the server to crash. Therefore, a tactic is required to monitor the speed of data generation and control the flow of data into the computing servers.

Description. The main components of Data Ingestion Monitoring tactic are shown in Figure 21. The *data collector* component collects data from various sources using different tools such as Wireshark for collecting network traffic data. The data collected by various nodes of the *data collection* component is moved to the *distributed data storage and analysis* component through *data ingestion monitor*. The role of *data ingestion monitor* is to keep the real-time data streaming into the *distributed data storage and analysis* cluster. The *data ingestion monitor* monitors the speed of the flow of the data becomes higher than the capacity of the computing server, this tactic will block the flow of the data into the computing cluster. The *data ingestion monitor* adjusts the flow of the data between *data collection* component and *distributed data storage and analysis* component. The stream of security event data fed into the *distributed storage and analysis* component is analyzed for detecting cyber attacks.

Constraints. Data Ingestion Monitoring Tactic requires expertise for installation of data ingestion monitor, which is a challenging task, especially in highly distributed setup. Furthermore, the tactic also requires more investment for setting up a monitoring server. The tactic is best fit for BDCA systems that are deployed in enterprises, which generate a large volume of security event data at a high speed and stream it directly into the analytic component for real-time processing.

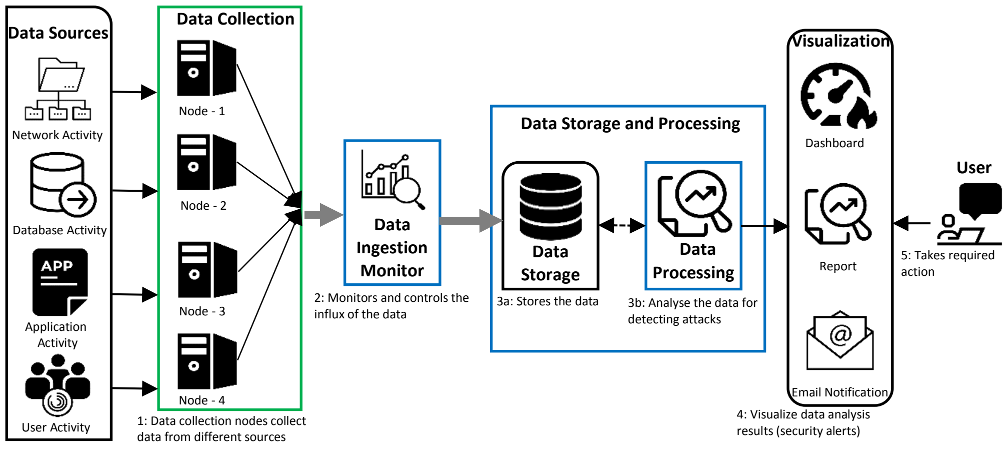


Figure 21. Data ingestion monitoring tactic.

Example. Data Ingestion Monitoring tactic is incorporated in the following systems.

- Malicious IP Detection [S18]: This system demonstrates how Data Ingestion Monitoring tactic prevents computing server from crashing. Flume server [79] is used as data ingestion monitor to monitor and control the influx of data into the computing cluster. The authors have evaluated the system with four data influx data rates (i.e., 0.4 million records/min, 0.7 million records/min, 0.8 million records/min, 0.85 million records/min). It has been reported that the computing server has idle capacity up to 0.8 million records/min, however, as the speed is increased to the 0.85 million records/min, the computing server reaches its maximum limit. Increasing the data influx rate beyond the 0.85 million records/min would crash a computing server. Therefore, at 0.85 million records/min, the data ingestion monitor controls the speed from increasing any further.
- Streaming-based Threat Detection System [S1] and GPGPU-based IDS [S35]: Both these systems implement Data Ingestion Monitoring tactic through Flume service, however, these studies have not evaluated the impact of the tactic on the reliability of the system.

Dependencies. Data Ingestion Monitoring tactic does not depend upon any other tactic; however, it can better be consolidated when implemented together with Secure Data Transmission tactic (Section 5.5.1), which will help get the data to be monitored in its original form.

5.4.2. Maintaining multiple copies

Introduction. Maintaining Multiple copies tactic can be found in any system that leverages Hadoop framework, however, only seven [S2], [S6], [S10], [S13], [S17], [S31], [S71] of the reviewed papers explicitly demonstrate how this tactic can make a system more reliable. This tactic enables Hadoop's HDFS to maintain multiple copies of each data block. These copies are placed on different nodes in a Hadoop infrastructure. In case of a node failure, the computation is diverted to another node that hosts the copy of the data block [80].

Motivation. During data processing, a Hadoop node may go down or fail due to several reasons such as RAM crash, power shutdown, and hard-disk failure. In such a case, data on the same node will no longer be accessible and a user has to wait until the issue is addressed. Therefore, a mechanism is required that can enable a BDCA system to have access to the data even if a node fails.

Description. The main components of Maintaining Multiple Copies tactic are shown in Figure 22 with the numbers showing the sequence of operations. The *data collector* component collects security event data from different sources, which is stored in files. The files are divided into blocks of same sizes. The default block size is 64 MB; however, the block size can be customized. For the sake of understanding, only one file split into four blocks is shown in Figure 22. After dividing files into blocks, the blocks are stored on different nodes (i.e., machines). By default, three copies of each data block are created. However, the number of replicas created can be configured. Each data block is stored on three different machines. For example, Block 1 is stored on Node 1, Node 4, and Node 6. Then the *data processing* component reads the data blocks for security analysis. If a data block is not accessible at any instance (for instance block 1 on Node 1), the same data block will be accessed from another node (for instance block 1 from Node 4). Once the data is analyzed in a reliable manner, the results are shared with the user through the *visualization* component.

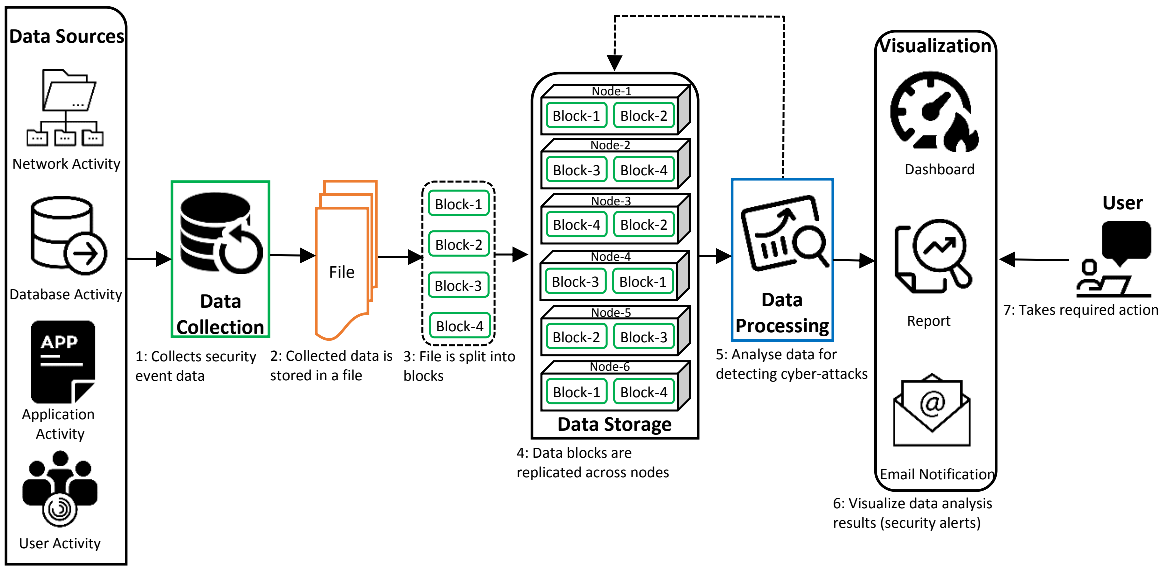


Figure 22. Maintaining multiple copies tactic.

Constraints. With the incorporation of Maintaining Multiple Copies tactic, the cost of storing the same data increases. This tactic may also introduce some data inconsistencies.

Example. Following are a couple of examples that illustrate the implementation of this tactic.

- Honeypot-based Phishing Detection [S17]: The impact of replication factor (i.e., number of copies) on the performance of a BDCA system has been investigated with one, two, and three replicas. It is observed that a system's response time is best with three replicas (21 min), followed by two replicas (22 min). With one replica, a system can have the lowest response time (25 min) and no reliability in case of a node failure. The difference in response time is due to the number of non-local accesses incurred in each case, which is 4 with three replicas, 7 with two replicas, and 113 with one replica.

- Reliable Traffic Analysis [S71]: This system has been tested with two node failure scenarios – one where node executing map task fails and another where node executing reduce task fails. The node running map task is forced to reboot, while the node running reduce task is shutdown. The corresponding map tasks and reduce tasks are migrated to other nodes having the replicas of the data. It has been observed that the system remained available, however, it takes little more time in completing the tasks. For example, for flow count of 3.2 million, the completion time without nodes failure is 220.2 seconds, while with node failure the completion time is 380.2 seconds.

Dependencies Maintaining Multiple Copies tactic does not require any other tactic for its implementation, however, it can better be consolidated when implemented together with Secure Data Transmission tactic (Section 5.5.1), which will help get the data to be replicated in its original form.

5.4.3. Dropped NetFlow detection

Introduction. Dropped NetFlow Detection tactic has been found in Hybrid Stream and Batch Analyzer [S54]. NetFlow is a sequence of packets that typically share seven values, which include destination IP, source IP, destination port, source port, IP protocol, type of service, and ingress interface [35]. This tactic helps BDCA systems, which rely on collecting and analyzing NetFlow data for detecting attacks, to detect if the data collector has missed some NetFlows. The tactic monitors the NetFlow Sequence Numbers [81] and if they are found out of order, a warning message specific to that stream is logged. This tactic is important for detecting faults that can have severe consequences for reliable data collection.

Motivation. For BDCA system relying on NetFlow data, it is important to collect and analyze each NetFlow. Missing a single NetFlow may result in missing the detection of an attack (e.g., a single flow record may summarize malicious transfer of a large amount of data). In the existing network dynamics, it is quite possible that some NetFlows are missed due to several reasons that include (a) network dropping packets; (b) router not able to handle the volume of traffic; and (c) data collector of a BDCA system not able to handle the volume of traffic. Hence, it is quite crucial to monitor and detect whether a BDCA system is analyzing all NetFlow or there are some NetFlows going missing. In the latter case, a security administration needs to trace the source causing loss and fix the issue.

Description. The main components of the Dropped NetFlow Detection tactic are shown in Figure 23 along with the number indicating the sequence of the operations. The network traffic is passing through the router shown in the figure. A *NetFlow collector* is connected to the router, which collects the NetFlows and stores them in the *NetFlow storage*. During the NetFlow collection process, *NetFlow sequence monitor* component is monitoring the sequence numbers embedded in the NetFlows. If the sequence numbers are found out of order at any stage, the *NetFlow sequence monitor* generates a *warning message* indicating the missing flow in the particular stream of NetFlow. The warning message is logged alongside the specific stream in the *NetFlow storage* to specify that the stream of NetFlow has some flows missing that might be crucial in relation to detecting an attack. At the same time, a warning is displayed to a security administrator through the *visualization* component. Then a security administrator may take the required actions for fixing the issue due to which some NetFlows may get dropped.

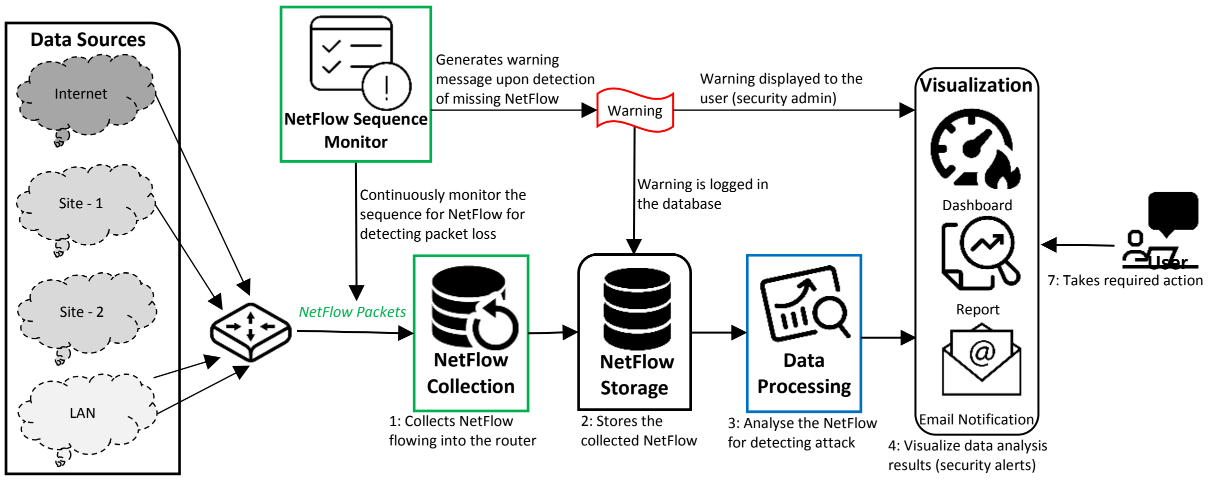

Data Sources
Internet
Site - 1
Site - 2
LAN
NetFlow Sequence Monitor
Generates warning message upon detection of missing NetFlow
Warning
Warning displayed to the user (security admin)
Continuously monitor the sequence for NetFlow for detecting packet loss
Warning is logged in the database
NetFlow Packets
NetFlow Collection
NetFlow Storage
Data Processing
Visualization
Dashboard
Report
Email Notification
User
7: Takes required action
1: Collects NetFlow flowing into the router
2: Stores the collected NetFlow
3: Analyse the NetFlow for detecting attack
4: Visualize data analysis results (security alerts)

Figure 23. Dropped NetFlow Detection Tactic.

Constraints. The NetFlow sequence monitoring should cope with the speed at which NetFlow is collected so that additional delay for monitoring can be avoided. This tactic is best fit for BDCA systems that monitor highly busy networks where there is a chance of packet loss. It is worth noting that this tactic only detects the loss of NetFlows. For mitigation, i.e., identifying the source of loss and fixing, it requires a separate mechanism.

Example. Hybrid Stream and Batch Analyzer [S54] is the example systems that have implemented Dropped NetFlow Detection tactic. The system has multiple NetFlow collectors that run on the probe nodes. Each NetFlow collector has two associated AMPQ queues [82] – one for the data stream and another for log information. The system experiences loss of NetFlows due to link aggregation [83]. In order to detect the missing NetFlows, the sequence numbers of the NetFlows are monitored and when they are found out of sequence, a warning message specific to the stream is placed in the log information queue.

Dependencies. This tactic can be implemented independent of any other tactics.

5.5. Security

This section reports the architectural tactic that is related to Security quality attribute.

5.5.1. Secure data transmission

Introduction. Secure Data Transmission tactic has been reported in Cloud-based Threat Detector [S10] and PKI-based DIDS [S73]. This tactic ensures secure transmission of security event data from data collection nodes to the data processing nodes.

Motivation. A BDCA system protects a critical infrastructure from cyber attacks, however, it also requires some security measures. In most cases, the data collection and analysis operations reside on separate nodes (physical machines). The nodes hosting data collection operation are placed in different regions within an organization [84]. The data collected by these nodes is transmitted to the nodes (cloud cluster) hosting the data analysis operation. It is quite possible that an attacker intercepts this data transmission (e.g., with a sniffing tool) to tamper with or spoof the data [85]. The consequences of such interception can be catastrophic if an attacker manages to tamper with the data critical for detecting an attack. Therefore, it is important to guarantee secure transmission of data from data collection nodes to the cloud cluster responsible for data analysis.

Description. Figure 24 shows the main elements of the Secure Data Transmission tactic. The nodes for collecting security event data are placed in different regions for collecting different types of data. Some collect network traffic and others collect database access information. Security measures are applied to the collected data to ensure its security while in transition from one component to another component. Some systems prefer to encrypt the collected data and then transfer the data. Other systems use Public Key Infrastructure (PKI) to ensure secure exchange of data and verification of the party sending the data. As data is received by the *data storage and analysis* component in a secure way, the data analytic operations are applied to analyze the data for detecting attacks. The results of the analysis are presented to users through the *visualization* component.

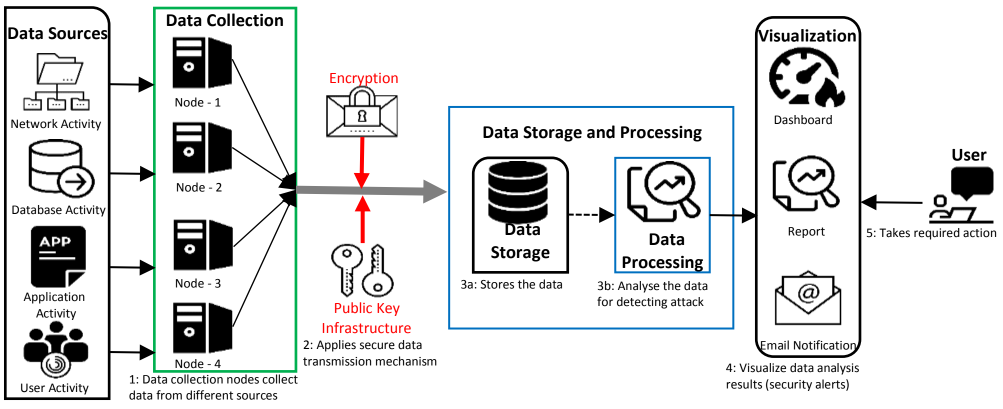


Figure 24. Secure data transmission tactic.

Constraints. The introduction of security measures in communication among different components will slow down the data transmission process and eventually the response to an attack. It is also worth noting that Secure Data Transmission tactic protects data only in motion state, which means that additional measures will be required to secure the data in use and in rest state both at the collection nodes and the storage and analysis nodes. For instance, the injection of false data in one of the data collection nodes can have disastrous consequences in terms of attack detection.

Example. The two systems that have implemented Secure Data Transmission tactic incorporate some kind of security measure for securing the data during transition.

- Cloud-based Threat Detector [S10]: In this system, all communications between various modules take place in an encrypted form, which is achieved by adopting HTTPS protocol. In order to ensure the authenticity of access, session tokens are checked and verified.
- PKI-based DIDS [S73]: This system deploys Public Key Infrastructure [86] between data collection nodes and data storage and analysis nodes. According to this security measure, digital certificates are used to encrypt data to be transmitted from data collection to data storage and analysis. It also enables the data collection and data storage component to verify the authenticity of data collection nodes. This way, even if an attacker intercepts the transmission, it will be very hard to tamper with the confidentiality or integrity of the data.

Dependencies. This tactic can be implemented independent of other tactics.

5.6. Usability

This section reports the architectural tactic that is related to Usability quality attribute.

5.6.1. Alert ranking

Introduction. Alert Ranking tactic has been reported in Hunting attacks in dark [S41]. This tactic ranks the alerts generated by a BDCA system based on the dangerousness of alert. This ranking improves the usability of a system by enabling users (e.g., security administrator) to respond to alerts on a priority basis.

Motivation. A BDCA system can generate a large number of alerts [87, 88]. In around 90% of the cases, the alerts are either false positives or of low importance [89]. It is also well understood that the more dangerous an alert is the more challenging it is to respond and mitigate its effects [S41]. If a security administrator responds to alerts in a sequence in which alerts are generated, it is quite likely that serious alerts requiring an immediate response will be responded quite late. On the other hand, if a security administrator assesses the seriousness of alerts, it will be quite time-consuming. A delayed response may be quite detrimental as it will be challenging to mitigate the effects of an attack. Therefore, it is important to incorporate a mechanism in a BDCA system that can help a security administrator to prioritize the responses to more serious alerts.

Description. The main components of Alert Ranking tactic are shown in Figure 25 with the numbers indicating the sequence of the operations. The *data collection* component collects security event data from different sources, which is pre-processed by the *data pre-processing* component. The pre-processed security event data is forwarded to the *data processing* component, which analyzes the data for detecting cyber attacks. The results of the analysis (i.e., alerts) are forwarded to the *alert ranking* component, which ranks the alerts based on a predefined criterion to assess the impact of an alert on the overall organization's infrastructure. The criterion for ranking alerts depends upon an organization. For example, the ranking criteria for an organization vulnerable to DoS attacks will be different to an organization vulnerable to brute force attacks. The ranked list of simple and easy-to-interpret alerts is shared with security administrators through the *visualization* component, which eases the job of a security administrator to first respond to the alerts on top of the rank list.

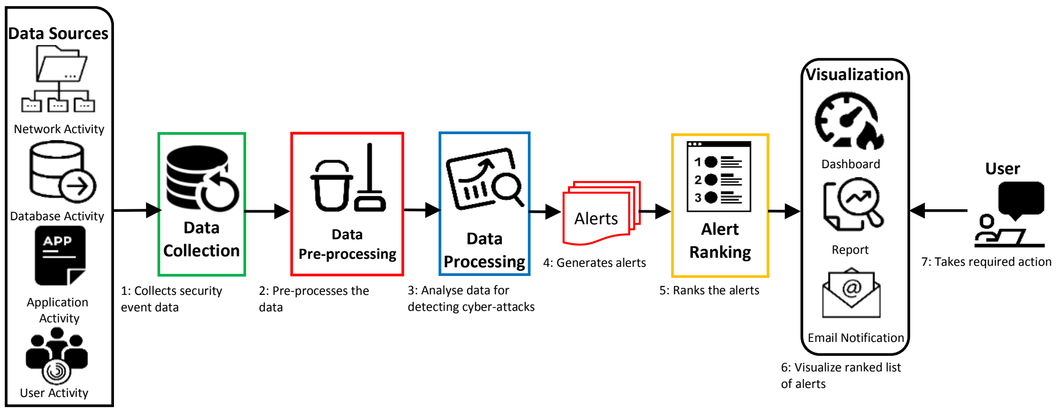


Figure 25. Alert ranking tactic.

Constraints. Alert Ranking tactic requires that the functionality responsible for ranking the alerts should not be computationally expensive, otherwise it will cancel the benefit of quick response acquired through prioritized response to dangerous alerts. Moreover, all ranking algorithms are not equally accurate [90]. Therefore, it is important to carefully select the ranking algorithm (criteria) that can accurately rank alerts based on the specific security requirements of an organization.

Example. Hunting attacks in the dark [S41] is an example system that has implemented Alert Ranking tactic. This system ranks an alert based on the amount of traffic that belongs to alert. The underlying consideration for this criterion is that alerts indicating a sudden increase in the network traffic (e.g., flooding-based attacks) are the most serious. The system uses the number of packets and the number of bytes belonging to each alert to calculate Dangerousness Index (DI) for each alert using the formula (i.e., DI(alert) = Number of packets + Number of bytes). The bigger the DI of an alert is, the more dangerous is the alert. Based on the DI scores, alerts are ranked and presented to security administrators.

Dependencies. Alert Ranking tactic does not entirely depend on any other tactic, however, this tactic is usually implemented together with tactics that can support the data processing component such as Parallel Processing tactic (Section 5.1.4) and Attack Detection Algorithm Selection tactic (Section 5.2.3).

6. Discussion

Section 3, 4, and 5 have presented the findings from our review. In this section, we draw some lessons from the findings and reflect upon the potential usefulness and implications of the findings of this review for practitioners.

6.1. Lessons Learned

6.1.1. Mapping of tactics to the modules of BDCA system

Figure 26 shows the mapping of the tactics reported in Section 5 onto the major modules of a BDCA system: *data collection, data storage, data pre-processing, data processing, data post-processing and visualization*. The mapping is expected to help a reader to understand which modules(s) or stage(s) are highly critical from an architectural perspective. It is clear that around 47% (i.e., 8 out of 17 tactics) of the architectural effort is focussed on the *data processing* module. The *data processing* module mainly relates to accuracy (four tactics) and performance (three tactics), which are considered as the two highly emphasized quality attributes (*Lesson 3*). We recommend that a significant amount of time and energy be invested in designing this module. A flawed design decision can lead to detrimental consequences. For example, the selection of an inefficient ML algorithm (Section 5.2.1) can negatively impact a BDCA system's several qualities attributes such as accuracy and performance ([S1], [S9], [S10], [S14], [S27]). The mapping also shows that the *visualization* module's design has received the least amount of attention in the reviewed systems. Given the importance of efficient visualization in big data systems [91], there is an urgent need of advanced research aimed at realizing the potential of *visualization* for BDCA systems.

Lesson 1: From an architectural perspective, **Data Analysis** is the most critical module of a BDCA system

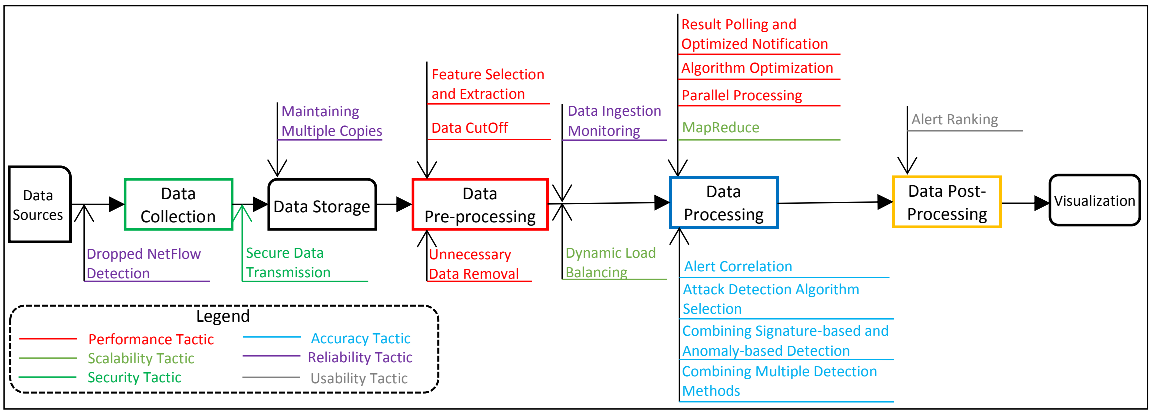


Figure 26. Mapping of tactics to the modules of BDCA system.

6.1.2. Mapping of quality attributes to the modules of a BDCA system

The mapping (shown in Figure 26) provides a relationship between the quality attributes and different modules. This type of visual information is expected to help a reader to understand which modules are important for achieving certain quality attributes. Figure 26 shows that the accuracy related tactics (i.e., Section 5.2) are completely focussed on the *data processing* module. The tactics associated with the remaining quality attributes are largely scattered throughout a system. For example, performance related tactics (Section 5.1) are incorporated in the *data pre-processing* and *data processing* modules. The reliability related tactics (Section 5.4) are associated with multiple modules such as *data collection*, *data storage*, and *data processing*. This analysis identifies the areas of knowledge/expertise that an organization aspires to have for its desired quality goals. For instance, a BDCA system having accuracy as the primary quality goal requires expertise in data analytics. An organization intending to develop a BDCA system that can satisfy multiple quality goals should have a diverse set of resources and expertise ranging from data collection to data analytics and visualization.

Lesson 2: Apart from **accuracy**, the architectural support for achieving quality goals is largely distributed among the modules of a BDCA system

6.1.3. First class quality attributes

Given that the priority of quality attributes of a system varies from domain to domain [15], it is important to know the priorities of quality attributes for BDCA systems from our findings (*RQ1*). Our findings indicate that a wide range of quality concerns have been reported in the reviewed papers – ranging from *performance* to *stealthiness* (Table 6). We observe a significantly highly number of papers focus on three quality attributes: *performance* (90.5%), *accuracy* (58.1%), and *scalability* (54%). Our finding related to architectural tactics (*RQ2*) further strengthen this observation, where 12 out of 17 tactics (i.e., 70.5%) aim to achieve these three quality attributes. Ian and colleagues' paper [14] also emphasize *performance* and *scalability* as key quality attributes for a big data analytics. A possible reason for not including accuracy as a key quality attribute could be the application domains (i.e., aeronautics and healthcare) considered in [14], where the cost of misclassification may not be as detrimental as in cybersecurity [11]. Given it is challenging to achieve several quality attributes for a system, a good approach is to first focus on the most significant ones (e.g., *performance*, *accuracy*, and *scalability*).

Lesson 3: **Performance**, **accuracy**, and **scalability** are the three most important quality attributes in the context of BDCA.

6.1.4. Under-addressed quality attributes

Section 4 reports our analysis of the importance of specific quality attributes for BDCA systems. Apart from performance, accuracy, scalability, reliability, security, and usability, our analysis has revealed that quality attributes such as interoperability, adaptability, modifiability, generality, stealthiness and privacy assurance are also important (Table 6) for BDCA systems. There is a little focus on architectural support for achieving these quality attributes in the reviewed papers. It is worth mentioning that some of the reviewed papers briefly mention the need of achieving these qualities. For example, the papers [S19] and [S42] advocate the use of open source tools for achieving interoperability. The papers [S10] and [S29] recommend higher modularity for achieving interoperability and modifiability. For generality, the papers [S12] and [S23] encourage the incorporation of diverse data sources and frequently updating attack patterns. However, the reported recommendations are quite generic and abstract. Therefore, we assert that there is a need for more research efforts aimed at providing architectural support for interoperability, modifiability, adaptability, generality, stealthiness and privacy assurance in BDCA systems. Such an architectural support is expected to devise tactics, patterns, and design principles for addressing quality concerns related to the aforementioned quality attributes.

Lesson 4: It is crucial to investigate architectural support for achieving **interoperability**, **modifiability**, **adaptability**, **generality**, **stealthiness**, and **privacy assurance** in BDCA systems

6.1.5. Tactics evaluation

It is important to assess the system-wide impact of the elicited architectural tactics for BDCA systems because (i) the tactics have been elicited from specific implementations, (e.g., BDCA system detecting DoS attack) using different datasets (e.g., KDD, DARPA, and CAIDA), therefore, their incorporation and potential impact cannot be generalized to all types of BDCA systems (ii) Due to the lack of sufficient empirical evidence, a software architect may employ a tangled implementation of a tactic(s), which can lead to severe negative impacts [92] (iii) a concrete catalogue of qualitative and quantitative evidence will build software architect's trust in the tactics and will highlight the cautions considerable during the incorporation of various tactics. The primary studies from which the tactics have been extracted do include an evaluation phase, however, it does not report the evaluation of each of the identified tactics. For example, the impact of Combining Multiple Detection methods tactic (Section 5.2.4), Alert Correlation tactic (Section 5.2.1), Dynamic Load Balancing tactic (Section 5.3.1), Dropped NetFlow Detection tactic (Section 5.4.3), Secure Data Transmission tactic (Section 5.5.1), and Alert Ranking tactic (Section 5.6.1) have not been evaluated. We believe that there can be expert based assessment (e.g., [93]) or experimentation (e.g., [94], [95]) to qualitatively and quantitatively evaluate the potential impact of the identified tactics.

Lesson 5: It is essential to (qualitatively and quantitively) **investigate the system-wide impact** of the codified tactics

6.1.6. Quality trade-offs among tactics

It is well known that architectural tactics may positively support one set of quality attributes but may have negative impact on another set of quality attributes [15]. For example, Alert Correlation tactic (Section 5.2.1) helps achieve *accuracy* but can have negative impact on *performance;* Maintaining Multiple Copies tactic (Section 5.4.2) improves *reliability* but increases the overall *storage cost*. Moreover, an architectural tactic can have a positive impact on multiple quality attributes unlike its association with one quality attribute. For instance, Parallel Processing tactic (Section 5.1.4) improves both *performance* and *reliability*. Making and understanding the trade-offs centric design decisions can be quite challenging task [15]. We present a two-dimensional matrix (Figure 27) to highlight the relationship between the reported advantages/disadvantages of the quality attributes and the codified tactics. The matrix can be used by a software architect in two ways. First, a software architect can identify which benefits are of key interest in their situation and accordingly select the set of tactics. For example, a BDCA system aims to detect highly sophisticated attacks should incorporate Alert Correlation (Section 5.2.1) and Combining Multiple Detection Methods (Section 5.2.4) tactics. Second, the matrix helps in assessing the impact of tactics at the design time. For instance, the matrix illustrates that Secure Data Transmission tactic (Section 5.5.1) improve data integrity and data confidentiality, however, slows down the overall response time. This information can stimulate a software architect to deliberate what is more significant (i.e., data security or quick response) for a particular scenario.

Lesson 6: It is important to analyse the **quality trade-offs** among the codified tactics at the design time before selecting/dropping a tactic

Quality Advantages / Architectural Tactics	ML Algorithm Optimization	Unnecessary Data Removal	Feature Selection and Extraction	Parallel Processing	Result Polling and Optimized Notification	Data CutOff	Alert Correlation	Combining Signature-based and Anomaly-based	Attack Detection Algorithm Selection	Combining Multiple Detection Methods	Dynamic Load Balancing	MapReduce	Data Ingestion Monitoring	Maintaining Multiple Copies	Dropped NetFlow Detection	Secure Data Transmission	Alert Ranking
Reduce computational complexity	✓		✓						✓								
Removes unnecessary data		✓															
Remove redundant records		✓															
Dimensionality reduction			✓														
Reduce data complexity			✓														
Distributes processing among nodes				✓								✓					
Reduce processing delays					✓												
Applies customizable data cutoff						✓											
Reduce false positives							✓	✓		✓							
Enable detection of highly sophisticated attacks							✓			✓							
Enable detection of zero-day attacks							✓	✓									
Ensures multi-dimensional data analysis										✓							
Balances load among the nodes											✓	✓					
Improve communication among nodes												✓					
Improves synchronization among tasks												✓					
Controls influx of data													✓				
Handles node failures													✓				
Improves data integrity														✓	✓	✓	
Detect data collection faults															✓		
Improves data confidentiality																✓	
Facilitates quick response																	✓
Quality Disadvantages																	
Risk of discarding relevant data		✓				✓											
Risk of discarding critical features			✓														
Potential of damage to logical records				✓													
Increase in computational cost				✓													
Risk of evading attack detection						✓											
Increase in computational time							✓	✓		✓		✓					
Requires domain knowledge							✓	✓									
Introduce interoperability challenges																	
Increase storage cost														✓			
Introduce data inconsistencies														✓			
Slows down overall response time																✓	✓
Requires more resource and expertise													✓				

Figure 27. Matrix linking Quality Advantages/Disadvantages to Architectural Tactics

6.1.7. Dependencies among tactics

The reported tactics can be used into different stages of a system (Figure 26). It is important to understand how different tactics work together by exploring the inter-dependencies and collaborations among the reported tactics. As mentioned in Section 5, architectural tactics may depend on other tactics for achieving their respective objectives. For example, Feature Selection and Extraction tactic (Section 5.1.3) depends upon Parallel Processing tactic (Section 5.1.4) for speeding up the process of feature selection and extraction. A combination of tactics may complement each other in a way that allows an architect to drop one tactic in favour of another tactic to resolve a conflict between two quality attributes. For example, Combining Multiple Detection Methods tactic (Section 5.2.4) and Combining Signature-based and Anomaly-based Detection tactic (Section 5.2.2) complement each other to achieve *accuracy* quality attribute. However, the use of both tactics will increase *response time*. Whilst the dependencies among tactics have been reported to a certain level in Section 5, this review has enabled us to assert there is an important need of empirical research to identify and understand the deeper dependencies among the identified architectural tactics.

Lesson 7: Research is essential to investigate the (level of) **dependencies** among the codified tactics

6.1.8. Modelling the tactics

The architectural tactics, as reported in this study, are not modelled using a standard modelling language such as Unified Modelling Language (UML). However, we assert that it would be useful to define the semantics of the codified tactics (similar to [96]) using a standard modelling language. Our assertion is motivated by two reasons – (I) Modelling of a tactic is expected to abstract the explicit design details to enable a software designer to focus on the significant tasks of a tactic in a designed solution (II) Modelling is expected to ensure that a tactic's constraints are respected and to efficiently trace the tactics in the source code. A tactic's modelling is expected to capture the structural and behavioural details of the tactics. For example, the model for Features Selection and Extraction tactic (Section 5.1.3) should illustrate the roles of feature selection and feature extraction components, the details of their interaction with other components, and the lifeline of their interactions. It is important to model the dependencies and relationships among tactics. Such modelling is meant to reveal the type of dependencies among tactics. For example, a software architect needs explicit guidance on whether the Parallel Processing tactic (Section 5.1.4) is mandatory for Feature Selection and Extraction tactic (Section 5.1.3) or it is only suggestive. One such option for modelling the relations among tactics is Features Modelling [97], which is a technique for modelling the commonalities and dependencies among patterns and tactics.

Lesson 8: **Formal modelling** of the codified tactics is required to facilitate the software architect in **better understanding** the role of tactics and respect the associated constraints

6.1.9. Demographic Lessons

Our demographic view (Section 2) has revealed some points that should be of interest and value to researchers and practitioners working in the area of BDCA specifically and cybersecurity in general. The popularity and adoption of Spark (as compared to Hadoop) is quite evident from Figure 5b. One possible reason for this, as reported in several of our reviewed papers (e.g., [S16], [S17], and [S29]), is the processing speed with respect to which Spark outclasses Hadoop by at least 10 times. Section 3.2 reveals that only four papers out of 74 included the papers that were published in software engineering related venues; this situation indicates a lack of focus on this important domain of research and development. This paucity is not limited to just BDCA, instead circles the focus of software engineering for any domain of big data analytics as highlighted by Madhavji and colleagues [98]. This study has revealed several research areas (reported in Section 6.1.4, 6.1.5, 6.1.6, 6.1.8) that requires the attention of SE research community. Our finding (Figure 4) also strengthens the already proposed view [3] that the traditional cybersecurity solutions will not be able to sustain for long, hence, the adoption of big data technologies is inevitable as evidenced by its increased adoption over the years.

Lesson 9: In the context of BDCA (i) **Spark is getting more popular** as compared to Hadoop (iii) there is a **paucity of Software Engineering** focus (iii) **Role of big data technologies** is on rise.

6.2. Implications

Given the focus of our study (i.e., architectural tactics) and its intended application for practitioners, we make a few recommendations for practitioners about using the codified tactics.

6.2.1. Choosing among competing tactics

Our catalogue of tactics, a kind of design space, can be used by practitioners for supporting different types of quality attributes in BDCA systems. For example, there are six tactics available (as shown in Figure 7) that aim to achieve *performance* in different ways. Considering the quality trade-offs (Lesson 6), it is not a good design decision to randomly select any particular set of tactics [15]. The choice of which tactic(s) to choose or not to choose becomes a challenge for software designers. Section 5 reveals extensive correlation and dependencies among the codified tactics (i.e., including or excluding a tactic(s) impacts other tactics and subsequently quality attributes), which should be selected and used through a systematic analysis and design process [15]. Whilst there is no silver bullet for optimal choice of tactics, we suggest prioritizing the selection of tactics, which has minimum impact (as highlighted in the *consequence* section for each tactic) on other quality attributes. Future research should empirically investigate the system-wide impact of the identified tactics (i.e., Lesson 5). Such research can help practitioners to devise suitable strategies for choosing an optimal set of tactics for design decisions.

Recommendation 1: Follow a comprehensive **analysis and design process** to choose among competing tactics

6.2.2. Refining and contextualizing tactics

The identified tactics have been codified at different abstraction levels. For example, *Feature Selection and Extraction tactic* (Section 5.1.3) is at a higher level of abstraction, which offers flexibility in its implementation. For instance, a software architect can decide whether features should be locked at the design time or runtime and which technique should be used for feature selection. On the other hand, tactics such as *Result Polling and Optimized Notification tactic* (Section 5.1.6) lies at a lower abstraction level, where an architect has little (if none) flexibility during implementation. That means the abstraction level for several tactics (e.g., Alert Correlation, Combing Multiple Detection Methods, and Data Ingestion Monitoring) is quite high; hence, these tactics need to be refined according to specific functional and non-functional requirements. The use of the identified tactics should take context (e.g., resource and cast) into consideration. For example, the availability of sufficient computational resource can support Feature Selection and Extraction tactic (Section 5.1.3) in a way that features can be selected intelligently at runtime, which will add to accuracy without impacting the performance. Similarly, a software architect has several options [99] for employing Alert Correlation tactic (Section 5.2.2).

Recommendation 2: **Refine and contextualize** tactics according to the underlying requirements and resource

6.2.3. Tools Literacy

The reviewed papers reported a variety of big data tools used in different phases of BDCA systems – *data collection* (e.g., Wireshark and nfdump), *data pre-processing* (e.g., Fume and Kafka), and *data analysis* (e.g., ML algorithms and big data frameworks). The option of having such a large pool of tools makes it challenging as to which set of tools should be selected. Apart from a few papers (e.g., [S9], [S14], [S26]) that compare the performance of some ML algorithms, there is a scarcity of literature aimed at comparing the relative performance of various tools (e.g., big data frameworks). We assert the need for comparative studies that can give a consolidated view for the selection of big data tools for BDCA. We recommend that software architects of BDCA systems should take a broader view of big data tools from multiple angles (e.g., cost, team and expertise) before incorporating a particular tool in a BDCA system.

Recommendation 3: Take time to make **well-informed decisions** about **selection of tools** for BDCA.

7. Related work

The design and development of BDCA systems have been drawing an increasing amount of attention of researchers and practitioners. There have been a few literature reviews on security analytics from different perspectives. Whilst there have been no literature review of architectural aspects (i.e., quality attributes and architectural tactics) of BDCA systems; nor has there been any review conducted using Systematic Literature Review methodology. We briefly summarize some of the existing reviews on BDCA to position our work. Kaj et al. [100] report a literature survey that provides a taxonomic analysis of the techniques for cybersecurity analytics. They categorized the techniques into descriptive, diagnostic, visual, predictive, and prescriptive. The prescriptive techniques have the potential for further research. Lidong and Randy [101] reviewed the literature on the big data processing frameworks (e.g., Spark and Storm), data pre-processing, and machine learning for security analytics. However, their work does not address the architectural aspects.

Tariq and Uzma [102] reviewed the literature that sheds light on different types of cyber attacks (e.g., botnets and phishing), various sources of security event data, sample outputs of a security analytic system, and commercially available security analytic solutions. Jeong et al. [103] investigated various challenges and Hadoop based potential solutions for IDS. The reported challenges include emerging attack patterns, high implementation cost, and large volume, high velocity and heterogeneous nature of security event data. The authors argue that the use of Hadoop, MapReduce and NoSQL databases, and incorporation of open source software help address the aforementioned challenges. Rasim and Yadigar [104] reviewed the state-of-the-art big data technologies for security analytics. Their review highlights various security analytics challenges such as privacy, APT detection, cryptography, huge amount of dataset, data provenance, security visualization, and lack of skilled personnel.

The work that relates more closely to our work is that of Ricard et al. [7], which reports the challenges faced by IDS for analyzing a large amount of heterogeneous data. The identified challenges include complex ML, lack of evaluation datasets, and feature selection. They recommend the correlation of security events, the collection of data from diverse sources, and incorporation of big data technologies for efficient intrusion detection. To the best of our knowledge, our work is the first SLR focussed on the architecture and Architecturally Significant Requirements (quality attributes) of BDCA systems.

8. Limitations

Whilst we have designed and conducted this SLR by following the guidelines of [20], there are certain limitations that need to be considered while examining its findings.

Our SLR has the potential of missing some relevant papers. We identified and selected the relevant papers by searching the six digital repositories, which are considered the most relevant sources of literature on computer science and software engineering [20]. We did not impose any restrictions on the publication date of the papers. To ensure the reliability of the search string, we tested the search string through pilot searches and iteratively improved it until the searches returned the papers that we already knew to be relevant. We reviewed the references of the primary studies obtained via the automatic searches (i.e., snowballing) [22].

The selection of the papers was based on the inclusion and exclusion criteria (reported in Section 2.3), the selection process could have been influenced by the researchers' subjective judgement. To address this issue, the reasons for inclusion and exclusion of the papers were recorded and reviewed by the authors. We also tried to reduce the potential misinterpretations to a minimum by following a multi steps selection process.

There can also be the potential impact of a researcher's bias in extracting the data from the reviewed papers. We designed a data extraction form (Table 4) for ensuring consistency in data extraction. As mentioned in Section 2, we employed quantitative and qualitative analysis techniques for analysing the extracted data. There is a possibility of some bias in the interpretation of the extracted data. To mitigate the researchers' bias in data interpretation, wherever possible we explored the antecedent and subsequent publications of the authors of the reviewed papers, websites of the tools based on the papers, and blogs of the authors of the reviewed papers.

9. Conclusion

Motivated by the growing significance of big data analytics for cyber security, we decided to systematically gather and rigorously analyze and synthesize the literature on architectural tactics used for designing BDCA systems. Based on a Systematic Review of 74 relevant papers, we have identified and explained 12 quality attributes considered important for BDCS systems. We have also identified and codified 17 architectural tactics for supporting the required qualities (i.e., performance, accuracy, scalability, reliability, security, and usability) in BDCA systems.

This SLR promises several potential benefits both for researchers and practitioners. For researchers, our SLR has identified a number of areas for future research. The demographic findings are of the potential value for researchers to shape their future research directions. It is clear that the application of big data technologies in security analytics is gaining significant traction. Compared to Hadoop, Spark is becoming more popular. Given that important quality attributes such as interoperability, modifiability, adaptability, generality, stealthiness and privacy assurance lack architectural support, we assert that researchers need to explore various options for developing such architectural support. Other areas for the future research include empirical evaluation of the reported architectural tactics, trade-off and dependency analysis among the tactics, and a comparative analysis among big data processing frameworks (e.g., Hadoop, Spark, and Storm) when employed in BDCA systems.

For practitioners, the identification of the most relevant quality attributes and a catalogue of architectural tactics can serve as useful design space while designing BDCA systems. Given that the elicitation of non-functional requirements (e.g., privacy, adaptability, and scalability) is a challenging task, practitioners can benefit from the identified quality attributes supported by qualitative and quantitative reasoning to establish non-functional requirements for BDCA systems. This SLR is a first step towards guiding software architects and software engineers to efficiently architect BDCA systems. For concretizing the body of knowledge and materializing practitioners' trust through empirical evidence, we plan to establish an experimental setup of distributed computing complemented by big data technologies for rigorous empirical evaluation of the codified architectural tactics. The experimental design will aim to investigate the trade-offs and dependencies among architectural tactics. We intend to develop automated support for instantiating the identified architectural tactics. We also plan to investigate the refactoring of existing BDCA systems to support the incorporation of our tactics in a system. Any future effort on this topic should also include an effort to develop a body of knowledge on the strengths and weakness of big data technologies for cybersecurity analytics. Such a body of knowledge will help practitioners to select suitable technologies according to their specific requirements.

Appendix A. Primary studies selected for this SLR.

Table 7. List of included primary studies

ID	System name	Title	Author(s)	Publication venue	Year
S1	Streaming-based Threat Detection	A Streaming-Based Network Monitoring and Threat Detection System	Zhijiang Chen , Hanlin Zhang , William G. Hatcher , James Nguyen , Wei Yu	Software Engineering Research, Management and Applications (SERA)	2016
S2	IP Spoofing typed DDoS detection	A Hadoop based analysis and detection model for IP Spoofing typed DDoS attack	Jian Zhang, Pin Liu, Jianbiao He, Yawei Zhang	IEEE TrustCom/BigDataSE/ISPA	2016
S3	Stochastic Self-similarity	Detecting Anomaly Teletraffic Using Stochastic Self-Similarity Based on Hadoop	JongSuk R. Lee, Sang-Kug Ye and Hae-Duck J. Jeong	Conference on Network-based Information Systems	2013
S4	Combining IDS Datasets	Combining intrusion detection datasets using MapReduce	Mondher Essid, Farah Jemili	Conference on Systems, Man, and Cybernetics	2016
S5	K-means based Anomaly Detector	Anomaly Detection in Network Traffic using K-mean clustering	R. Kumari, Sheetanshu, M. K. Singh, R. Jha, N.K. Singh	Conference on Recent Advances in Information Technology	2016
S6	Multistage Alert Correlator	Distributed Multistage Alert Correlation Architecture based on Hadoop	James Rees	International Carnahan Conference on Security Technology	2015
S7	GSLAC	GSLAC: A General Scalable and Low-overhead Alert Correlation Method	Li Cheng, Yijie Wang, Xingkong Ma and Yongjun Wang	IEEE TrustCom/BigDataSE/ISPA	2016
S8	Website IDS using Hadoop	Hadoop-based System Design for Website Intrusion Detection and Analysis	Xiaoming Zhang, Guang Wang	Conference on Smart City/SocialCom/SustainCom	2015
S9	Spark-based IDS Framework	A Framework for Fast and Efficient Cyber Security Network Intrusion Detection using Apache Spark	Govind P Guptaa, Manish Kulariyaa	Conference on Advances in Computing & Communications	2016
S10	Cloud-based Threat Detection	A Cloud Computing Based Network Monitoring and Threat Detection System for Critical Infrastructures	Zhijiang Chen, Guobin Xu, Vivek Mahalingam, Linqiang Ge, James Nguyen, Wei Yu , Chao Lu	Big Data Research	2015
S11	Spatiotemporal Correlator	A Cloud Computing Based Architecture for Cyber Security Situation Awareness	Wei Yu, Guobin Xu, Zhijiang Chen, and Paul Moulema	Workshop on Security and Privacy in cloud computing	2013
S12	Quasi Real-time IDS	Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests	Kamaldeep Singh, Sharath Chandra Guntuku, Abhishek Thakur a, Chittaranjan Hota a	Journal of Information Sciences	2014
S13	Spark-based Network Data Analysis	Network Data Analysis Using Spark	K.V. Swetha, Shiju Sathyadevan, and P. Bilna	Software Engineering in Intelligent Systems	2015
S14	Ultra-High-Speed IDS	Real time intrusion detection system for ultra-high-speed big data environments	M. Mazhar Rathore1 · Awais Ahmad1 · Anand Paul1	Journal of supercomputing	2016
S15	METRIS	METIS: A Two-Tier Intrusion Detection System for Advanced Metering Infrastructures	Vincenzo Gulisano, Magnus Almgren, and Marina Papatriantafilou	Conference on Security and Privacy in Communication Systems	2014
S16	Big Data Security Monitoring	A Big Data Architecture for Large Scale Security Monitoring	Samuel Marchal, Xiuyan Jian, Radu State, Thomas Enge	International Congress on Big Data	2014
S17	Honeypot-based Phishing Detection	A Big Data architecture for security data and its application to phishing characterization	Pedro H. B. Las-Casas, Vinicius Santos Dias, Wagner Meira Jr. and Dorgival Guedes	Conference on BigDataSecurity-HPSC-IDS	2016
S18	Malicious IP Detection	A Real-time Anomalies Detection System based on Streaming Technology	Yutan Du, Jun Liu, Fang Liu, Luying Chen	Conference on Intelligent Human-Machine Systems and Cybernetics	2014
S19	AGILIS	Distributed Attack Detection Using Agilis	Leonardo Aniello, Roberto Baldoni, Gregory Chockler, Gennady Laventman, Giorgia Lodi, and Ymir Vigfusson	Journal of Collaborative Financial Infrastructure Protection	2012
S20	Detecting DDoS in Cloud Service	A Spark-Based DDoS Attack Detection Model in Cloud Services	Jian Zhang, Yawei Zhang, Pin Liu(B), and Jianbiao He	Conference on Information security practice and experience	2016
S21	Compression model for IDS	Efficient classification using parallel and scalable compressed model and its application on intrusion detection	Tieming Chen, Xu Zhang a, Shichao Jin, Okhee Kim	Journal of Expert Systems with Applications	2014
S22	BotFinder	An Automated Bot Detection System through Honeypots for Large-Scale	Fatih Haltaş, Abdulkadir Poşul, Erkam Uzun, Bakır Emre	Conference on Cyber Conflict	2014
S23	Botnets with Graph Theory	Big Data Behavioral Analytics Meet Graph Theory: On Effective Botnet Takedowns	Elias Bou-Harb, Mourad Debbabi, and Chadi Assi	IEEE Network	2017
S24	Extreme Learning Machine	Using Extreme Learning Machine for Intrusion Detection in a Big Data Environment	Junlong Xiang, Magnus Westerlund, Dušan Sovilj, Göran Pulkkis	Workshop on Artificial Intelligent and Security	2014

S25	Intersection-based Pattern Matching	Privacy-Preserving Scanning of Big Content for Sensitive Data Exposure with MapReduce	Fang Liu, Xiaokui Shu, Danfeng (Daphne) Yao and Ali R. Butt	Conference on Data and Application Security and Privacy	2014
S26	WEKA-based Anomaly Detector	Anomaly detection model based on Hadoop platform and Weka interface	Baojiang Cui, Shanshan He	Conference on Innovative Mobile and Internet Services in Ubiquitous Computing	2016
S27	Improved K-means IDS	An Intrusion Detection System Based on Hadoop	Zhiguo Shi, Jianwei An	UIC-ATC-ScalCom-CBDCom-IoP	2015
S28	Neural-Network based DDoS Detection using Hadoop	A Neural-Network Based DDoS Detection System Using Hadoop And HBase	Teng Zhao	International Symposium on Cyberspace safety and security	2015
S29	RADISH	Detecting Insider Threats Using RADISH: A System for Real-Time Anomaly Detection in Heterogeneous Data Streams	Brock B¨ose, Bhargav Avasarala, Srikanta Tirthapura, Yung-Yu Chung, and Donald Steiner	IEEE Systems	2017
S30	Forensic Analyzer	Cloud Computing-Based Forensic Analysis for Collaborative Network Security Management System	Zhen Chen, Fuye Han, Junwei Cao, Xin Jiang, and Shuo Chen	Journal of Tsinghua Science and Technology	2013
S31	IDS-MRCPSO	MapReduce Intrusion Detection System based on a Particle Swarm Optimization Clustering Algorithm	Ibrahim Aljarah and Simone A. Ludwig	Congress on Evolutionary Computation	2013
S32	Hive vs MySQL	Performance Evaluation of Big Data Technology on Designing Big Network Traffic Data Analysis System	Nattawat Khamphakdee, Nunnapus Benjamas and Saiyan Saiyod	Conference on Soft Computing and Intelligent Systems	2016
S33	SDN DDoS Detector	A DDoS Detection and Mitigation System Framework Based on Spark and SDN	Qiao Yan(B) and Wenyao Huang	Conference on Smart Computing and Communication	2017
S34	Cloud Bursting	High-performance network traffic analysis for continuous batch intrusion detection	Ricardo Morla · Pedro Gonçalves, Jorge G. Barbosa	Journal of supercomputing	2016
S35	GPGPU-based IDS	Design Consideration of Network Intrusion Detection System using Hadoop and GPGPU	Sanraj Rajendra Bandre, Jyoti N. Nandimath	Conference on Pervasive Computing	2015
S36	BotCloud	BotCloud: Detecting Botnets Using MapReduce	J´erˆome Franc¸ois, Shaonan Wang, Walter Bronzi, Radu State, Thomas Engel	Conference on Information Forensics and Security	2011
S37	Ctracer	Ctracer: Uncover C&C in Advanced Persistent Threats based on Scalable Framework for Enterprise Log Data	Kai-Fong Hong, Chien-Chih Chen, Yu-Ting Chiu, and Kuo-Sen Chou	Congress on Big Data	2015
S38	SEAS-MR	Scalable Security Event Aggregation for Situation Analysis	Jinoh Kim, Ilhwan Moon, Kyungil Lee, Sang C. Suh, Ikkyun Kim	Conference on Big Data Computing Service and Applications	2015
S39	APSIS	Toward a Big Data Architecture for Security Events Analytic	Laila Fetjah, Karim Benzidane, Hassan El Alloussi, Othman El Warrak, Said Jai-Andaloussi and Abderrahim Sekkaki	Conference on Cyber Security and Cloud Computing	2016
S40	Security Orchestrator	Enabling Trustworthy Spaces via Orchestrated Analytical Security	Joshua Howes, James Solderitsch, Ignatius Chen, Jonté Craighead	Workshop on Cyber Security and Information Intelligence Research	2013
S41	Hunting Attacks in the Dark	Hunting attacks in the dark: clustering and correlation analysis for unsupervised anomaly detection	Johan Mazel, Pedro Casas, Romain Fontugne, Kensuke Fukuda and Philippe Owezarski	Journal of Network Management	2015
S42	Multi-Tier Threat Intelligence	Towards a Big Data Architecture for Facilitating Cyber Threat Intelligence	Charles Wheelus, Elias Bou-Harb, Xingquan Zhu	Conference on New Technologies, Mobility, and security	2016
S43	DDoS Detection with HTTP Packet Pattern	A method of DDoS attack detection using HTTP packet pattern and rule engine in cloud computing environment	Junho Choi · Chang Choi, Byeongkyu Ko · Pankoo Kim	Journal of soft computing	2014
S44	Dynamic Rule Creation based Anomaly Detector	A Dynamic Rule Creation Based Anomaly Detection Method for Identifying Security Breaches in Log Records	Jakub Breier, Jana Branisˇova	Journal of Wireless Personal Communication	2017
S45	VALKYRIE	Valkyrie: Behavioural malware detection using global kernel-level telemetry data	Sven Krasser, Brett Meyer, Patrick Crenshaw	Workshop on Machine learning for signal processing	2015
S46	Multiple Detection Methods	Towards online anomaly detection by combining multiple detection methods and Storm	Ziyu Wang, Jiahai Yang,Hui Zhang, Chenxi Li, Shize Zhang and Hui Wang	Conference on Network Operations and Management Symposium	2016
S47	APT Detector	Study And Research of APT Detection Technology Based on Big Data Processing Architecture	Lin Shenwen, Li Yingbo, Du Xiongjie	Conference on Electronics Information and Emergency Communication	2015
S48	IDS Log Analyzer	Scalable Intrusion Detection Systems Log Analysis using Cloud Computing Infrastructure	Manish Kumar, Dr. M. Hanumanthappa	Conference on Computational Intelligence and Computing Research	2013

S49	Data Intensive Framework	Real-Time Handling of Network Monitoring Data Using a Data-Intensive Framework	Aryan Taheri Monfared, Tomasz Wiktor Wlodarczyk, Chunming Rong	Conference on Cloud Computing Technology and Science	2013
S50	PhishStorm	PhishStorm: Detecting Phishing With Streaming Analytics	Samuel Marchal, Jérôme François, Radu State, and Thomas Engel	IEEE Transactions on Network and Service Management	2014
S51	Lightweight Security Framework	Massive Distributed and Parallel Log Analysis For Organizational Security	Xiaokui Shu, John Smiy, Danfeng (Daphne) Yao, and Heshan Lin	Workshop on security and privacy in big data	2013
S53	IoT Monitor	Parallel Processing of Big Heterogeneous Data for Security Monitoring of IoT Networks	Igor Saenko, Igor Kotenko, Alexey Kushnerevich	Conference on Parallel, Distributed, and network-based processing	2017
S53	VAST	Native Actors: How to Scale Network Forensics	Matthias Vallentin, Dominik Charousset	SIGCOMM	2014
S54	Hybrid Stream and Batch Analyzer	Scalable Hybrid Stream and Hadoop Network Analysis System	Vernon K. C. Bumgardner, Victor W. Marek	Conference on Performance Engineering	2014
S55	Web IDS using Storm	Research of Recognition System of Web Intrusion Detection Based on Storm	Li Bo, Wang Jinzhen, Zhao Ping, Yan Zhongjiang, Yang Mao	Conference on Network, communication, and computing	2016
S56	Count Me In	Count Me In: Viable Distributed Summary Statistics for Securing High-Speed Networks	Johanna Amann, Seth Hall, and Robin Sommer	Journal on research in attacks, intrusions, and defences	2014
S57	PSO Clustering	Towards a Scalable Intrusion Detection System based on Parallel PSO Clustering Using MapReduce	Ibrahim Aljarah and Simone A. Ludwig	Conference on genetic and evolutionary computing	2013
S58	Dynamic Time Threshold	A Statistical Threat Detection Method Based on Dynamic Time Threshold	Jian-Wei Tian, Hong Qiao, Xi Li, Zhen Tian	Conference on Computer and Communications	2016
S59	Compound Session based Anomaly Detector	A Real-time Network Traffic Anomaly Detection System based on Storm	Gang He,Cheng Tan,Dechen Yu,Xiaochun Wu	Conference on Intelligent Human-Machine Systems and Cybernetics	2015
S60	System Call based Detector	A System Call Analysis Method with MapReduce for Malware Detection	Shun-Te Liu, Hui-ching Huang, Yi-Ming Chen	Conference on Parallel and distributed systems	2011
S61	Genetic Algo-based DDoS Detector	Distributed Denial of Services Attack Protection System with Genetic Algorithms on Hadoop Cluster Computing Framework	Masataka Mizukoshi	Congress on Evolutionary computation	2014
S62	Neural Network & Spark based DDoS Detector	Detection of DDoS attacks based on neural network using apche spark	Chang-Jung Hsieh, Ting-Yuan Chan	Conference on Applied System Innovation	2016
S63	Feature Extractor	Distributed Network Traffic Feature Extraction for a Real-time IDS	Ahmad M Karimi, Quamar Niyaz, Weiqing Sun, Ahmad Y Javaid, and Vijay K Devabhaktuni	Conference on Electro Information Technology	2016
S64	Hashdoop	Hashdoop: A MapReduce Framework for Network Anomaly Detection	Romain Fontugne, Johan Mazel, Kensuke Fukuda	Workshop on Security and Privacy in Big Data	2014
S65	ICAS	ICAS: An Inter-VM IDS Log Cloud Analysis System	Shun-Fa Yang , Wei-Yu Chen, Yao-Tsung Wang	Conf. on Cloud computing and intelligence systems	2011
S66	Batch & Stream Analyzer	Real-Time Network Anomaly Detection System Using Machine Learning	Shuai Zhao, Mayanka Chandrashekar, Yugyung Lee, Deep Medhi	Conference on the design of reliable communication networks	2015
S67	R-based Traffic Analyzer	Traffic Analysis using Hadoop Cloud	Aishwarya.K, Dr.Sharmila Sankar	Conference on Innovation in Information, Embedded, and communication systems	2015
S68	MALSPOT	MalSpot: Multi2 Malicious Network Behavior Patterns Analysis	Hing-Hao Mao1, Chung-Jung Wu1, Evangelos E. Papalexakis, Christos Faloutsos, Kuo-Chen Lee1, and Tien-Cheu	Conference on Knowledge discovery and data mining	2014
S69	P2P Bot Detector	Scalable P2P bot detection system based on network data stream	Shree Garg & Sateesh K. Peddoju & Anil K. Sarje	Journal of Peer-to-peer networking applications	2016
S70	Hadoop-based Traffic Monitor	Towards Scalable Internet Traffic Measurement and Analysis with Hadoop	Yeonhee Lee and Youngseok Lee	ACM SIGCOMM Computer Communication Review	2013
S71	Reliable Traffic Analyzer	An Internet Traffic Analysis Method with MapReduce	Youngseok Lee, Wonchul Kang, Hyeongu Son	Workshop on Network Operations and Management	2010
S72	Hybrid IDS	Design of a Hybrid Intrusion Detection System using Snort and Hadoop	Prathibha.P.G, Dileesh.E.D	Journal of Computer Applications	2013
S73	PKI-based IDS	Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework	Rafael Timoteo de Sousa Junior	Journal of Revista telecommunications	2011
S74	MATATABI	MATATABI: Multi-layer Threat Analysis Platform with Hadoop	Hajime Tazaki, Kazuya Okada, Yuji Sekiya, Youki Kadobayashi	Workshop on Building analysis datasets and gathering experience returns for security	2014

References

1. Dua, S. and X. Du, *Data mining and machine learning in cybersecurity*. 2016: Auerbach Publications.
2. Cardenas, A.A., P.K. Manadhata, and S.P. Rajan, *Big data analytics for security.* IEEE Security & Privacy, 2013. **11**(6): p. 74-76.
3. Cárdenas, A.A., P.K. Manadhata, and S. Rajan, *Big data analytics for security intelligence. Available at https://goo.gl/wxKqDV*. 2013: p. 1-22.
4. KuppingerCole, B.a., *Big Data and Information Security Report. Available at https://goo.gl/tffZVv*. 2016.
5. Cybenko, G. and C.E. Landwehr, *Security Analytics and Measurements.* IEEE Security & Privacy, 2012.
6. Shackleford, D., *Security Analytics Survey. Available at https://goo.gl/S88DTJ*. 2016.
7. Zuech, R., T.M. Khoshgoftaar, and R. Wald, *Intrusion detection and big heterogeneous data: a survey.* Journal of Big Data, 2015. **2**(1): p. 3.
8. Nassar, M., B. al Bouna, and Q.M. Malluhi. *Secure Outsourcing of Network Flow Data Analysis*. in *BigData Congress*. 2013.
9. Chickowski, E., *A case study in security big data analysis.* Dark Reading, 2012. **9**.
10. Shackleford, D., *Security Analytics Survey. Available at https://goo.gl/cEUZZK.* SANS Survey, 2016.
11. Verma, R., et al., *Security analytics: essential data analytics knowledge for cybersecurity professionals and students.* IEEE Security & Privacy, 2015. **13**(6): p. 60-65.
12. Otero, C.E. and A. Peter, *Research directions for engineering big data analytics software.* IEEE Intelligent Systems, 2015. **30**(1): p. 13-19.
13. Shaw, M., *The role of design spaces.* IEEE software 29.1, 2012: p. 46-50.
14. Gorton, I. and J. Klein, *Distribution, data, deployment: Software architecture convergence in big data systems.* IEEE Software, 2015. **32**(3): p. 78-85.
15. Bass, L., P.C. Clements, and R. Kazman, *Software architecture in practice.* Addison-Wesley, 2012. **3rd ed.**
16. Lewis, G. and P. Lago, *Architectural tactics for cyber-foraging: Results of a systematic literature review.* Journal of Systems and Software, 2015. **107**: p. 158-186.
17. Procaccianti, G., P. Lago, and S. Bevini, *A systematic literature review on energy efficiency in cloud software architectures.* Sustainable Computing: Informatics and Systems, 2015. **7**: p. 2-10.
18. Garcés, L., et al., *Quality attributes and quality models for ambient assisted living software systems: A systematic mapping.* Information and Software Technology, 2017. **82**: p. 121-138.
19. Mahdavi-Hezavehi, S., et al., *A systematic literature review on methods that handle multiple quality attributes in architecture-based self-adaptive systems.* Information and Software Technology, 2017.
20. Kitchenham, B. and S. Charters, *Guidelines for performing systematic literature reviews in software engineering*, in *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. 2007, sn.
21. Zhang, H., M.A. Babar, and P. Tell, *Identifying relevant studies in software engineering.* Information and Software Technology, 2011. **53**(6): p. 625-637.
22. Wohlin, C. *Guidelines for snowballing in systematic literature studies and a replication in software engineering*. in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 2014. ACM.
23. Maplesden, D., et al., *Performance analysis for object-oriented software: A systematic mapping.* IEEE Transactions on Software Engineering, 2015. **41**(7): p. 691-710.
24. Cruzes, D.S. and T. Dyba. *Recommended steps for thematic synthesis in software engineering*. in *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. 2011. IEEE.
25. Pratyusa, A.A.C. and S. Rajan, *Big Data Analytics for Security Intelligence.* CLOUD SECURITY ALLIANCE, 2013.
26. Tan, J., et al. *Improving reducetask data locality for sequential mapreduce jobs*. in *INFOCOM, 2013 Proceedings IEEE*. 2013. IEEE.
27. NetFort, *Flow Analysis Versus Packet Analysis. What Should You Choose? Available at https://goo.gl/2PtAEF.* White Paper, 2014.
28. Vasilomanolakis, E., et al., *Taxonomy and survey of collaborative intrusion detection.* ACM Computing Surveys (CSUR), 2015. **47**(4): p. 55.
29. Liao, H.-J., et al., *Intrusion detection system: A comprehensive review.* Journal of Network and Computer Applications, 2013. **36**(1): p. 16-24.
30. Mitchell, R. and I.-R. Chen, *A survey of intrusion detection techniques for cyber-physical systems.* ACM Computing Surveys (CSUR), 2014. **46**(4): p. 55.
31. Wiegers, K. and J. Beatty, *Software Requirements.* Microsoft Press, 2014. **Third edition**.
32. Losavio, F., et al., *Quality characteristics for software architecture.* Journal of Object Technology, 2003. **2**(2): p. 133-150.
33. ITRC, *Breach Report. Available at https://goo.gl/oBNsPv*. 2017.
34. Dreger, H., et al. *Operational experiences with high-volume network intrusion detection*. in *Proceedings of the 11th ACM conference on Computer and communications security*. 2004. ACM.
35. Cisco, *NetFlow. Available at https://goo.gl/ko3Dyw*. 2017.
36. Zhou, C.V., C. Leckie, and S. Karunasekera, *A survey of coordinated attacks and collaborative intrusion detection.* Computers & Security, 2010. **29**(1): p. 124-140.
37. Stevanovic, M. and J.M. Pedersen, *Machine learning for identifying botnet network traffic.* 2013.
38. Pearson, S. *Taking account of privacy when designing cloud computing services*. in *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*. 2009. IEEE Computer Society.

39. Ullah, F., et al., *Data Exfiltration: A Review of External Attack Vectors and Countermeasures.* Journal of Network and Computer Applications, 2017.
40. Commission, E., *Protection of Personal Data. Available at https://goo.gl/JSCxrq*. 2016.
41. Galster, M. and P. Avgeriou. *Empirically-grounded reference architectures: a proposal*. in *Proceedings of the joint ACM SIGSOFT conference--QoSA and ACM SIGSOFT symposium--ISARCS on Quality of software architectures--QoSA and architecting critical systems--ISARCS*. 2011. ACM.
42. Bhuyan, M.H., D.K. Bhattacharyya, and J.K. Kalita, *Network anomaly detection: methods, systems and tools.* IEEE communications surveys & tutorials, 2014: p. 303-336.
43. Chen, C.P. and C.-Y. Zhang, *Data-intensive applications, challenges, techniques and technologies: A survey on Big Data.* Information Sciences, 2014. **275**: p. 314-347.
44. Buczak, A.L. and E. Guven, *A survey of data mining and machine learning methods for cyber security intrusion detection.* IEEE Communications Surveys & Tutorials, 2016. **18**(2): p. 1153-1176.
45. Caruana, R. and A. Niculescu-Mizil. *An empirical comparison of supervised learning algorithms*. in *Proceedings of the 23rd international conference on Machine learning*. 2006. ACM.
46. Cheng, C., W.P. Tay, and G.-B. Huang. *Extreme learning machines for intrusion detection*. in *Neural networks (IJCNN), the 2012 international joint conference on*. 2012. IEEE.
47. Zhao, S., et al. *I-can-mama: Integrated campus network monitoring and management*. in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. 2014. IEEE.
48. Frey, B.J. and D. Dueck, *Clustering by passing messages between data points.* science, 2007. **315**(5814): p. 972-976.
49. Cisco, *Cisco Visual Networking Index: Forecast and Methodology, 2016–2021. Available at https://goo.gl/WYyTV5*. White Paper, 2017.
50. Bay, S.D., et al., *The UCI KDD archive of large data sets for data mining research and experimentation.* ACM SIGKDD Explorations Newsletter, 2000. **2**(2): p. 81-85.
51. CAIDA, *CAIDA Dataset. Available at https://goo.gl/Tg9p8R*. 2016.
52. Combs, G., *TShark—Dump and Analyze Network Traffic.* Wireshark, 2012.
53. Dean, J. and S. Ghemawat, *MapReduce: simplified data processing on large clusters.* Communications of the ACM, 2008. **51**(1): p. 107-113.
54. Maier, G., et al. *Enriching network security analysis with time travel*. in *ACM SIGCOMM Computer Communication Review*. 2008. ACM.
55. Park, K., G. Kim, and M. Crovella. *On the relationship between file sizes, transport protocols, and self-similar network traffic*. in *Network Protocols, 1996. Proceedings., 1996 International Conference on*. 1996. IEEE.
56. Paxson, V. and S. Floyd, *Wide area traffic: the failure of Poisson modeling.* IEEE/ACM Transactions on Networking (ToN), 1995. **3**(3): p. 226-244.
57. Qiao, L.-B., et al. *Mining of attack models in ids alerts from network backbone by a two-stage clustering method*. in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International*. 2012. IEEE.
58. Perdisci, R., G. Giacinto, and F. Roli, *Alarm clustering for intrusion detection systems in computer networks.* Engineering Applications of Artificial Intelligence, 2006. **19**(4): p. 429-438.
59. Sadoddin, R. and A. Ghorbani. *Alert correlation survey: framework and techniques*. in *Proceedings of the 2006 international conference on privacy, security and trust: bridge the gap between PST technologies and business services*. 2006. ACM.
60. Ning, P., Y. Cui, and D.S. Reeves. *Constructing attack scenarios through correlation of intrusion alerts*. in *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002. ACM.
61. Aydın, M.A., A.H. Zaim, and K.G. Ceylan, *A hybrid intrusion detection system design for computer network security.* Computers & Electrical Engineering, 2009. **35**(3): p. 517-526.
62. Kabiri, P. and A.A. Ghorbani, *Research on intrusion detection and response: A survey.* IJ Network Security, 2005. **1**(2): p. 84-102.
63. Nieves, J.F. and Y.C. Jiao, *Data clustering for anomaly detection in network intrusion detection.* Research Alliance in Math and Science, 2009: p. 1-12.
64. MIT, *DARPA intrusion detection evaluation data set. Available at https://goo.gl/jYBYNe*. 1998.
65. Kanungo, T., et al., *An efficient k-means clustering algorithm: Analysis and implementation.* IEEE transactions on pattern analysis and machine intelligence, 2002. **24**(7): p. 881-892.
66. Bezdek, J.C., R. Ehrlich, and W. Full, *FCM: The fuzzy c-means clustering algorithm.* Computers & Geosciences, 1984. **10**(2-3): p. 191-203.
67. Arlitt, M., D. Krishnamurthy, and J. Rolia, *Characterizing the scalability of a large web-based shopping system.* ACM Transactions on Internet Technology, 2001. **1**(1): p. 44-69.
68. Challenger, J.R., et al., *Efficiently serving dynamic data at highly accessed web sites.* IEEE/ACM transactions on Networking, 2004. **12**(2): p. 233-246.
69. Kalogeraki, V., P. Melliar-Smith, and L.E. Moser. *Dynamic migration algorithms for distributed object systems*. in *Distributed Computing Systems, 2001. 21st International Conference on*. 2001. IEEE.
70. Lewis, S., et al., *Hydra: a scalable proteomic search engine which utilizes the Hadoop distributed computing framework.* BMC bioinformatics, 2012. **13**(1): p. 324.
71. Pratt, B., et al., *MR-tandem: parallel X! tandem using hadoop MapReduce on amazon Web services.* Bioinformatics, 2011. **28**(1): p. 136-137.
72. Wiley, K., et al. *Astronomical image processing with hadoop*. in *Astronomical Data Analysis Software and Systems XX*. 2011.

73. Loebman, S., et al. *Analyzing massive astrophysical datasets: Can Pig/Hadoop or a relational DBMS help?* in *Cluster Computing and Workshops, 2009. CLUSTER'09. IEEE International Conference on*. 2009. IEEE.
74. Zhang, F., et al., *A task-level adaptive MapReduce framework for real-time streaming data in healthcare applications.* Future Generation Computer Systems, 2015. **43**: p. 149-160.
75. White, T., *Hadoop: The Definitive Guide. O'Reilly.* Scbastopol, California, 2009.
76. Lee, K.-H., et al., *Parallel data processing with MapReduce: a survey.* AcM sIGMoD Record, 2012. **40**(4): p. 11-20.
77. Lee, Y., W. Kang, and Y. Lee. *A hadoop-based packet trace processing tool*. in *International Workshop on Traffic Monitoring and Analysis*. 2011. Springer.
78. Aljarah, I. and S.A. Ludwig. *Parallel particle swarm optimization clustering algorithm based on mapreduce methodology*. in *Nature and biologically inspired computing (NaBIC), 2012 fourth world congress on*. 2012. IEEE.
79. Apache, *Flume. Available at http://flume.apache.org/. [Last Accessed: 1st Nov, 20127]*. 2017.
80. White, T., *Hadoop: The Definitive Guide. Availalbe at https://goo.gl/cRjiXL.* O'Reilly, 2015.
81. Cisco, *Cisco Systems NetFlow Services Export Version 9. Available at https://www.ietf.org/rfc/rfc3954.txt. [Last Accessed: 24 Oct, 2017].*
82. Vinoski, S., *Advanced message queuing protocol.* IEEE Internet Computing, 2006. **10**(6).
83. Wikipedia, *Link Aggregation. Available at https://goo.gl/ELB9mR.* 2016.
84. Bass, T., *Intrusion detection systems and multisensor data fusion.* Communications of the ACM, 2000. **43**(4): p. 99-105.
85. Muttik, I. and C. Barton, *Cloud security technologies.* Information security technical report, 2009. **14**(1): p. 1-6.
86. Myers, M., et al., *X. 509 Internet public key infrastructure online certificate status protocol-OCSP*. 1999.
87. Vaarandi, R. *Real-time classification of IDS alerts with data mining techniques*. in *Military Communications Conference, 2009. MILCOM 2009. IEEE*. 2009. IEEE.
88. Viinikka, J., et al., *Processing intrusion detection alert aggregates with time series modeling.* Information Fusion, 2009. **10**(4): p. 312-324.
89. Julisch, K., *Clustering intrusion detection alarms to support root cause analysis.* ACM transactions on information and system security (TISSEC), 2003. **6**(4): p. 443-471.
90. Allier, S., et al. *A framework to compare alert ranking algorithms*. in *Reverse Engineering (WCRE), 2012 19th Working Conference on*. 2012. IEEE.
91. Fisher, D., et al., *Interactions with big data analytics.* interactions, 2012. **19**(3): p. 50-59.
92. McNatt, W.B. and J.M. Bieman. *Coupling of design patterns: Common practices and their benefits*. in *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*. 2001. IEEE.
93. Khomh, F. and Y.-G. Gueheneuce. *Do design patterns impact software quality positively?* in *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*. 2008. IEEE.
94. Moghaddam, F.A., et al., *Empirical Validation of Cyber-Foraging Architectural Tactics for Surrogate Provisioning.* Journal of Systems and Software, 2017.
95. Procaccianti, G., H. Fernandez, and P. Lago, *Empirical evaluation of two best practices for energy-efficient software development.* Journal of Systems and Software, 2016. **117**: p. 185-198.
96. Kim, S., et al., *Quality-driven architecture development using architectural tactics.* Journal of Systems and Software, 2009. **82**(8): p. 1211-1231.
97. Czarnecki, K. and U.W. Eisenecker, *Generative programming: methods, tools, and applications*. Vol. 16. 2000: Addison Wesley Reading.
98. Madhavji, N.H., A. Miranskyy, and K. Kontogiannis. *Big picture of big data software engineering: with example research challenges*. in *Big Data Software Engineering (BIGDSE), 2015 IEEE/ACM 1st International Workshop on*. 2015. IEEE.
99. Elshoush, H.T. and I.M. Osman, *Alert correlation in collaborative intelligent intrusion detection systems—A survey.* Applied Soft Computing, 2011. **11**(7): p. 4349-4365.
100. Grahn, K., M. Westerlund, and G. Pulkkis, *Analytics for Network Security: A Survey and Taxonomy*, in *Information Fusion for Cyber-Security Analytics*. 2017, Springer. p. 175-193.
101. Wang, L. and R. Jones, *Big Data Analytics for Network Intrusion Detection: A Survey.* International Journal of Networks and Communications, 2017. **7**(1): p. 24-31.
102. Mahmood, T. and U. Afzal. *Security Analytics: Big Data Analytics for cybersecurity: A review of trends, techniques and tools*. in *Information assurance (ncia), 2013 2nd national conference on*. 2013. IEEE.
103. Jeong, H.-D.J., et al. *Anomaly teletraffic intrusion detection systems on hadoop-based platforms: A survey of some problems and solutions*. in *Network-Based Information Systems (NBiS), 2012 15th International Conference on*. 2012. IEEE.
104. Alguliyev, R. and Y. Imamverdiyev. *Big data: big promises for information security*. in *Application of Information and Communication Technologies (AICT), 2014 IEEE 8th International Conference on*. 2014. IEEE.