DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

A Real-time Log Correlation System for Security Information and Event Management

Clémence Dubuc

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Authors

Clémence Dubuc <dubuc@kth.se>
Information and Network Engineering
KTH Royal Institute of Technology

Examiner

Prof. Panos Papadimitratos
Division of Software and Computer Systems (SCS)
School of Electrical Engineering and Computer Science (EECS)
KTH Royal Institute of Technology

Supervisors

Mohammad Khodaei
Division of Software and Computer Systems (SCS)
School of Electrical Engineering and Computer Science (EECS)
KTH Royal Institute of Technology

Abstract

The correlation of several events in a period of time is a necessity for a threat detection platform. In the case of multi-step attacks (attacks characterized by a sequence of executed commands), it allows detecting the different steps one by one and correlating them to raise an alert. It also allows detecting abnormal behaviors on the IT system, for example, multiple suspicious actions performed by the same account. The correlation of security events increases the security of the system and reduces the number of false positives. The correlation of the events is made thanks to pre-existing correlation rules. The goal of this thesis is to evaluate the feasibility of using a correlation engine based on Apache Spark. There is a necessity of changing the actual correlation system because it is not scalable, it cannot handle all the incoming data and it cannot perform some types of correlation like aggregating the events by attributes or counting the cardinality. The novelty is the improvement of the performance and the correlation capacities of the system. Two systems are proposed for correlating events in this project. The first one is based on Apache Spark Structured Streaming and analyzed the flow of security logs in real-time. As the results are not satisfactory, a second system is implemented. It uses a more traditional approach by storing the logs into an Elastic Search cluster and does correlation queries on it. In the end, the two systems are able to correlate the logs of the platform. Nevertheless, the system based on Apache Spark uses too many resources by correlation rule and it is too expensive to launch hundreds of correlation queries at the same time. For those reasons, the system based on Elastic Search is preferred and is implemented in the workflow.

Keywords

Correlation, SIEM, Security Logs, Apache Spark, Elastic Search

Sammanfattning

Korrelation av flera händelser under en viss tidsperiod är en nödvändighet för plattformen för hotdetektering. När det gäller attacker i flera steg (attacker som kännetecknas av en sekvens av utförda kommandon) gör det möjligt att upptäcka de olika stegen ett efter ett och korrelera dem för att utlösa en varning. Den gör det också möjligt att upptäcka onormala beteenden i IT-systemet, t.ex. flera misstänkta åtgärder som utförs av samma konto. Korrelationen av säkerhetshändelser ökar systemets säkerhet och minskar antalet falska positiva upptäckter. Korrelationen av händelserna görs tack vare redan existerande korrelationsregler. Målet med den här avhandlingen är att utvärdera genomförbarheten av en korrelationsmotor baserad på Apache Spark. Det är nödvändigt att ändra det nuvarande korrelationssystemet eftersom det inte är skalbart, det kan inte hantera alla inkommande data och det kan inte utföra vissa typer av korrelation, t.ex. aggregering av händelserna efter attribut eller beräkning av kardinaliteten. Det nya är att förbättra systemets prestanda och korrelationskapacitet. I detta projekt föreslås två system för korrelering av händelser. Det första bygger på Apache Spark Structured Streaming och analyserade flödet av säkerhetsloggar i realtid. Eftersom resultaten inte var tillfredsställande har ett andra system införts. Det använder ett mer traditionellt tillvägagångssätt genom att lagra loggarna i ett Elastic Search-kluster och göra korrelationsförfrågningar på dem. I slutändan kan de två systemen korrelera plattformens loggar. Det system som bygger på Apache Spark använder dock för många resurser per korrelationsregel och det är för dyrt att starta hundratals korrelationsförfrågningar samtidigt. Av dessa skäl föredras systemet baserat på Elastic Search och det implementeras i arbetsflödet.

Nyckelord

Korrelation, SIEM, Säkerhetsloggar, Apache Spark, Elastic Search

Acknowledgements

I would like to express my very great appreciation to Mohammad Khodaei for all the help and the advice he gave me during my thesis. I also would like to thank Panos Papadimitratos for accepting my master thesis subject and agreeing being my examiner.

I am grateful to all of those with whom I have had the pleasure to work during this project. A special thanks to the entire team I work with during this project. Each of them helped me during my master thesis and took time to teach me with benevolence. I learned a lot thank to all of you.

Finally, I would like to thank my parents, my siblings and my friends for their love and support in all the projects I undertake.

Acronyms

API	Application Programming Interface
CEP	Complex Event Processing
CTI	Cyber Threat Intelligence
DAG	Directed Acyclic Graph
DDoS	Distributed Denial of Service
DoS	Denial of Service
EPL	Event Processing Language
eps	events per seconds
ETL	Extract Transform and Load
HTTPS	HyperText Transfer Protocol Secure
IDS	Intrusion Detection System
JSON	JavaScript Object Notation
RDD	Resilient Distributed Dataset
SIEM	Security Information and Event Management
SQL	Structured Query Language
SRE	Site Reliability Engineering
SSL	Secure Sockets Layer
STIX	Structured Threat Information Expression
TCP	Transport Control Protocol
TDR	Threat, Detection and Research
TLS	Transport Layer Security
VPN	Virtual Private Network
2FA	Two-Factor Authentication

Contents

Chapter 1

Introduction

1.1 The Platform

The platform is a cybersecurity platform as a service. The goal of this solution is to automate cyber defense. There are two main modules: One based on the Threat Intelligence knowledge constantly updated by analysts. The other (on which the thesis is based on) is a Security Information and Event Management (SIEM) application that can detect intrusions and raise alerts from the security events received. To perform this detection, analysts and customers can create what are called detection rules. They are based on dedicated Cyber Threat Intelligence (CTI) indicator feeds generated. The user forwards events to the platform thought different log collectors: syslog over Transport Layer Security (TLS), HyperText Transfer Protocol Secure (HTTPS) or cloud hosting polling. Those events can be anything that happen on the IT system of a company. They are analyzed by the platform to detect potential attacks.

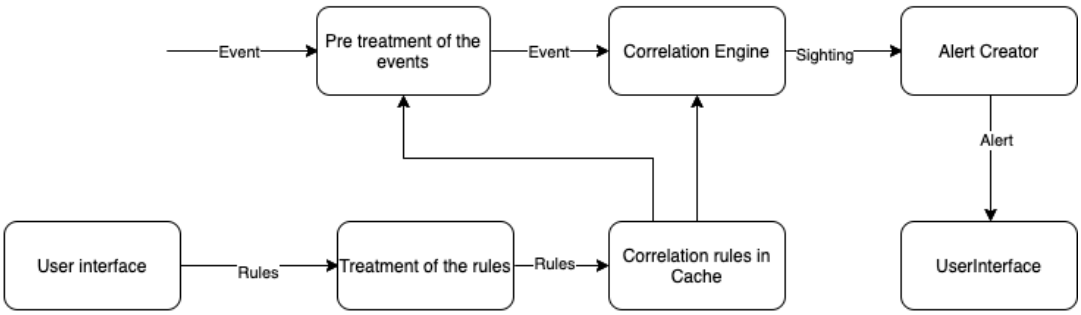

Figure 1.1.1: Components of the platform and their linkage.

The workflow is the sequence of micro-services that is used to detect attacks. Its goal

is to analyze events and raise alerts. It is illustrated in the figure 1.1.1. It is divided into multiple components. The rules written by the customers are stored in the cache after treatment and are used by the different components including the correlation engine. Events are translated in a more readable way and then they are going through different micro services to be analysed. The block pre-treatment analyses the events individually to find potential traces of an attack. The Correlation Engine correlates them to detect patterns of events and then send a *sighting* to create an alert.

1.2 Correlation Engine

The listing 1.1 shows an example of a correlation rule.

```
(A FOLLOWEDBY B) WITHIN 30 SECONDS
```

Listing 1.1: An example of a correlation rule with the operator *followedby*.

A and B are attributes of the events (IP address, event id, username and other attributes that could define an event). We want to know if the attribute A followed by the attribute B has been seen on the IT system within 30 seconds. The correlation engine stores the events with the attribute A and keeps them in memory for 30 seconds. If an events with the attribute B is seen, it sends an alert. If not, the old events are deleted. The complexity of this algorithm is at worst O(m^2) where m is the number of logs arrived during the period of time. In the worst case each log has to be compared to all the previous logs. It makes the complexity high and the algorithm inefficient. The major problem with this system is that it is not scalable and it cannot handle the number of logs the platform received.

The aggregation operator is a necessity the system. For example, the repetitive connection on the same URL can be an attack if they come from the same computer but it can just be a normal connection from many users if it comes from multiple computers. It is important to be able to do the difference between a real threat and a normal activity. The actual correlation system creates bugs on the platform and stops working when too many events are involved in the correlation. This leads to a loss of data and potential undetected attacks by the system. The actual correlation system is based on a correlation language called STIX Patterning [21] but this language is limited and does not allow writing complex queries. The correlation language is changed in this project for Sigma [27]. This choice is motivated by different reasons:

- Sigma is a more common correlation language than STIX Patterning. It is easier for the users to find existing correlation rules.
- Sigma is more powerful in term of capacities of detection: group by, cardinality.

1.3 Problem Statement

Correlating multiple events over a period of time is a necessity for a threat detection platform. It allows detecting attacks more precisely: for example, a single executed command could appear inoffensive while the execution of a certain sequence of commands could precede an intrusion. This is what we call in this thesis a multi-step attack. Correlation also helps to reduce false positives by specifying the signs of an attack. The current correlation system has some performance limitation that could lead to a flaw in threat detection.

Another problem is the scalability of the current correlation system. The platform is bound to grow in the next few years and the correlation has to handle an increasingly large volume of data. This need is not only caused by the increasing number of customers but also by the increasing number of security logs. Indeed, The IT security tools for monitoring systems become more and more elaborate and collect more and more information. Scalability is crucial for analyzing all those data and the current system does not handle as many logs as it should.

The final issue is real-time analysis of events. Detecting the alerts as quickly as possible increases the chances of avoiding damage on IT systems. However, correlating a current event with the previous ones is not an easy task because comparing all of them one by one is costly in terms of computations.

1.4 Objectives and Methodologies

The goal of this thesis is to develop a scalable correlation engine. One of the objectives is to be able to create hundreds of real-time correlation rules that work in real time on 15,000 events per seconds (eps). The correlation engine has to detect a threat in less than one minute. The other objective is to add new correlation operators to be able to create new correlation rules.

The project includes multiples objectives:

- Implemented a new correlation system on the existing system of the platform.
- Deal with performances and memory issues with thousands of events per second.
- Implemented a correlation language with new correlation operators and that is easily understandable by the user so they can write their own rules. This language is a link between the user and the correlation system. It receives the correlation rules and transforms them into something readable by the correlation engine.

1.5 Benefits

This project will benefit to the platform because it will improve its correlation engine and from a broader perspective it will benefit all its customers. Indeed, it will help them protect themselves by improving the monitoring of their IT system. A well-monitored system allows attacks and abuses to be detected in a short time and countermeasures to be taken more quickly.

1.6 Novelty

The novelty is the improvement of the performance and the correlation capacities of the correlation engine. The principal challenge comes from the adaptation of the new system to the actual workflow and the creation of a new correlation language. The system developed in this thesis is scalable and correlates thousands of events in a very short time because it uses distributed computing technologies which reduce the processing time. Another constraint is the necessity of real time. The platform analyzes all the events in real time and the new correlation scheme is able to correlate and raise an alert within the allowed limited interval of time.

1.7 Limitations

The first limitation of the project is the industrial secret between the security companies. The competitors do not show to the world how they are doing temporal detection. There is no open source code or explanation. Another limitation is that the project must be profitable for the company. This constraint limits the number of resources the system can use.

1.8 Terminologies

1.8.1 Security Information and Event Management (SIEM)

A SIEM is a tool for managing security events and information within a company at the same time. It allows a company to centralise all security information in one tool. Data collected from antivirus software, firewalls, servers, anti-theft protection and operating systems of all kinds will be analysed in a single tool. SIEMs allow [25] :

- Collection
- Aggregation
- Normalization
- Correlation
- Reporting
- Archiving
- Event replay

The detection capacities of SIEMs are based on their existing detection rules and their detection engine. Those rules evolve with the knowledge based on the existing group of attackers and the potential attacks. They can be based on the detection on one or multiple malicious logs. The first SIEMs were created when the IT environment were closed and the volume of data far less important. Those SIEMs used database-based storage cannot cope with today's scalability requirements. The platform is a SIEM next generation [20]. It means it is built on a big data platform that can handle the massive volumes of data produced by enterprises. The specificity of the system as a SIEM is the real-time treatment of the events.

The main advantage of a SIEM compared to an Intrusion Detection System (IDS) is the centralization of the logs. Indeed, an IDS is placed on a strategic position in the network for analyzing the logs on the machine while a SIEM centralized all the logs of the system and then analyzes and correlates. A SIEM is more powerful in term of threat detection because it can detect an attack coming from different machine on the network.

	search based correlation	real-time correlation
Pros	• possibility to do research after the event • the rules can be set periodically	• threats are detected in real-time
Cons	• no real time monitoring • require big databases	• difficulties to correlate a current log with the previous ones in real time.

Table 1.8.1: Pros and cons of the different types of correlation.

1.8.2 Event Correlation

This section clarifies the term correlation and defines what it means in this project. Event correlation consists of taking all the security information collected by the system and analyzing the data to identify relationships. For example, a large number of login attempts are noticed on an employee account and start executing suspicious commands. Through event correlation, an intrusion detection system can send an alert indicating that an attack is in progress.

We want to do a type of correlation named Knowledge-Based Correlation. It requires some pre-existing knowledge on the attacks. With this knowledge, analysts create detection rules. Those rules could include the detection of multiple events that are characteristic of a certain cyber attack. This is what it is called correlation in this project.

There are two ways to correlate logs:

- **Search based searches**: The events are stored in a database and the correlation is done by search-queries ran periodically. In a scheduled rule with search based SIEM two criteria are set up:
  - The frequency (run query every x minutes)
  - The maximum lookup period
- **Real-time correlation**: The events are analysed in real time and are not necessarily stored. This type of correlation is harder to implement because it is difficult to correlate current events with something that happened before.

This project presents those two type of correlation. The first part of the work is based on real-time correlation with Apache Spark. The second part presents a solution based on search based correlation but it makes it look as if it is real-time correlation. The next session presents scientific works about correlation and the technologies that are used.

Chapter 2

State-of-the-art Overview

2.1 Prior Work

The main challenge in intrusion detection and pattern recognition is to manage to detect attacks whereas the events which composed the attack do not reach the correlation system at the same time. This section presents the prior scientific works that has been published in this area.

2.1.1 Threat Detection with Machine Learning

There are different ways to detect multi-step attack scenarios. In their article [24], Ahmadian Ramakia et al. propose an algorithm to detect those scenarios. It begins by aggregating alerts according to their similarity and generating rules with machine learning. The system is able to detect the next step of the attack with more than 95% accuracy. Machine learning with big-data tools is a very efficient way to deals with cyber attacks. Even if the final goal of our project is similar, we use a very different method. It is not based on the prediction of an algorithm but on the knowledge of the attacks and the detection rules made by analysts.

The SCARFF project of Fabrizio Carcilloa et al. [5] proposes to use Spark [3], Kafka [2] and Cassandra [1] to detect credit card fraud. Spark's role is to aggregate historical data, to return the estimated fraud risk and to store the data in Cassandra. The authors explain that the advantage of Apache Spark is the scalabily. Indeed, if more resources are needed, one can increase the cluster size. This article uses Apache Kafka and Apache Spark. Those technologies are common in big data projects. The

Apache Spark system has three missions: the aggregation of historical transactions, the online classification of the transactions returning the estimated fraud risk and the cold storage of transactions in a distributed database. The functions of detection fraud and cyber threat are made by the machine learning module of a Apache Spark. It has the same purposes in the article of Hafsal and Jemili [12] which developed a fast, real-time intrusion detection system based on Apache Spark Structured Streaming with Microsoft Azure.

Spark Structured Streaming is used and not Spark Streaming because Casas et al. [6] showed that it was not sufficient for intrusion detection. The system is very close to the system we want to implement. It manages to process 55,175 records per second using only two worker-nodes cluster. The algorithm used is called *Decision Tree classifier*. It is a supervised Machine-learning algorithm. In the article, the authors show the efficiency of the Machine Learning algorithm with Apache Spark.

Those project are efficient to detect threat and fraud on network because the machine learning with Apache Spark allows analyzing and classifying a large amount of data in a short time. We chose to do knowledge based correlation in this project because we believe that it is necessity to combine both of this approaches to have a good detection capacity. In our project, we want to use the big data tools because it allows treating the data faster thanks to their distributed architecture but to avoid the false positive or the false negative, the detection rules implemented are created by cyber analysts. The rules look at sequences of events or repetition of events.

2.1.2 Threat Detection with CEP

Husák et al. [13] explains the necessity to aggregate data. This type of detection is different from the algorithm or machine learning approach. The idea of aggregate and filtering events for pattern detection is called CEP. CEP is a streamed process which relies on different techniques including event-pattern detection, event-filtering, event aggregation and transformation and this is what we implement in our system. Apache Spark [3] and Esper [9] are used in the work of Husák and Kašpar [14] to create a new framework for real-time correlation. This framework is directly comparable with ours. It contains components that filter, aggregate, and correlate the alerts. The Spark analytical engine receives a stream of alerts from Kafka [2], process them, and return them to Kafka. Esper also receives streamed events and process them. The main

features of Esper is its powerful query language. Indeed Event Processing Language (EPL) allows SQL-like queries over a stream of data. Each query in Esper requires the approximately the same available resources and they are running 10 queries in parallel. This number is too low for us, we want to be able to run hundreds of queries in parallel which represents hundreds of rules. In our project we do not use Esper but only Apache Spark which also has module for SQL-like queries named Spark SQL.

The article of Avanov and Taafe [16] explores the functionalities of Spark Structured Streaming. It shows that Spark can handle a lot of data and it is possible to handle multiple queries but with a change of velocity and volume sizes. However, only three queries in parallel are tried in the article. One of the objectives is to find a way to make it works with hundreds of queries in parallel on a large amount of data.

The article of Ficco and Romano [10] proposes a model to use CEP in IDS. It explains that the system is working well however one of the major drawbacks is the necessity of knowledge and the impossibility to detect unknown attacks with CEP. We are well aware of this limitation but most of the systems targeted by cyber criminal groups are threatened by already known and referenced attacks. We consider the CEP solution as a first barrier against the cyber threat and it could come as a complement of a machine learning solution.

The work of Armbrust et al. [4] shows a use case of Apache Spark Structured Streaming. It is used for an information security platform. The figure 2.1.1 is taken from the article of Armbrust et al. [4]. It shows the architecture of the platform. The name is not mentioned because it is not an open-source solution. It is explained that one of the key challenge is to create a effective environment to query fresh data. Our project is very similar to this work however the main challenge is to adapted Apache Spark Structured Streaming to the existing workflow. Indeed, the detection of attacks on single event stays as before because it has great performances. The correlation system should complement this workflow.

Apache Spark is not the only big data technologies with which CEP could be implemented. The article of Kotenko et al. [19] presents a SIEM solution based on Elastic Search. It is used as a search and analytical system. In their project Elastic Search [7] is used to realise most of the analysis of the data. The solution developed in this thesis is based on Elastic Search only for the correlation component. The detection on single events is made by the actual workflow. Elastic Search does not analyze the

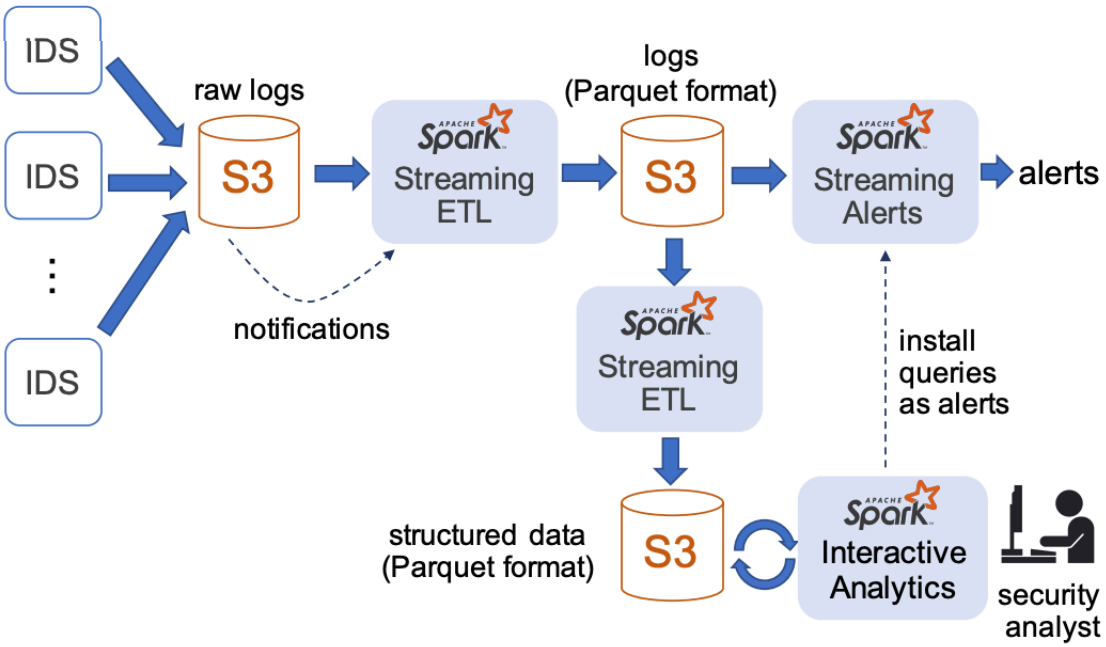

Figure 2.1.1: Information security platform use case. Using Structured Streaming and Spark SQL, a team of analysts can query both streaming and historical data and easily install queries for new attack patterns as streaming alerts. This figure is taken from [4].

events in real time and is based on the search based correlation. This type of correlation is more common and most of the SIEM solutions are based on it.

Chapter 3

System Model

3.1 Architecture of the System

The platform presented in this thesis is a SIEM that collects logs from companies and analyzes them in order to detect all kind of attacks. This thesis focuses on how the system analyzes the logs of customers and how it can be improve. A simplified architecture is represented by figure 3.1.1. The different components of this system are linked by Kafka topics [15] that contain the events treated by the previous component. Kafka topics have name and store the events. They can be compared to folder in a file system. A Kafka cluster possesses multiple Kafka topics that can contains different information.

A micro service is a component of the platform which takes part in the workflow by realizing a task. All micro services form the workflow. This thesis is focused on the correlation system. The logs treatment is not explained in details.

Correlation Worker

The correlation worker is the component that links multiple events to check if they are parts of the same attacks. This thesis focuses on how this component can be improve.

The figure 3.1.1 represents a simplify version of the workflow with the correlation system at the center. The correlation system receives logs of the customers' IT system. Those logs have already been treated by other micro services. Its goal is to correlate

events that can be link together. This link is created by correlation rules that are created and written by analysts or by the customers.

The inputs of the system are the logs from previous micro services. These logs have only been treated individually. The output of the system is a sighting. This sighting is used to create the alert and send the information needed to the customers so they can protect themselves.

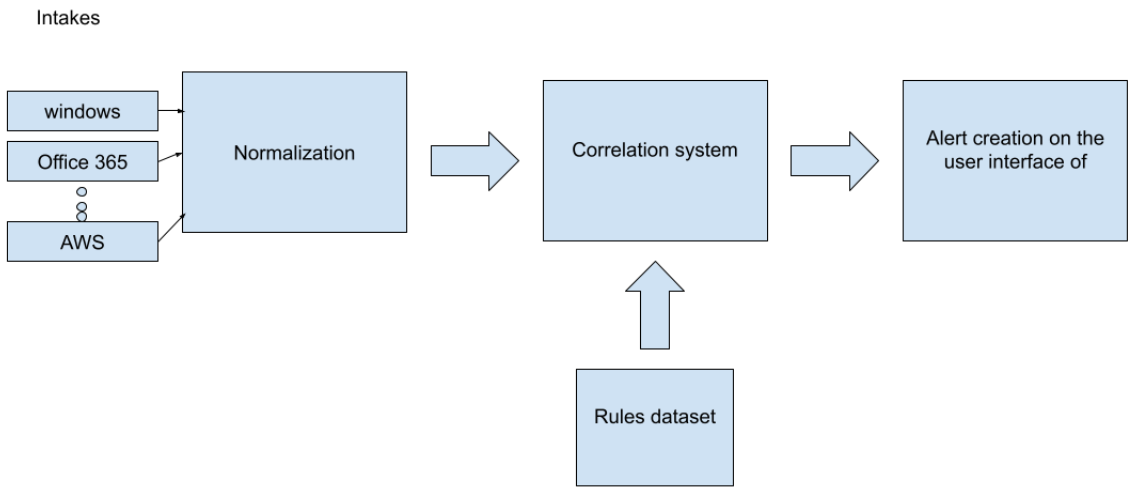

Figure 3.1.1: Architecture of the system model.

A correlation is performed by the actual correlation worker in the workflow. It is coded in python. The events that take part of a correlation rule are stored in python lists. Each rules has its own list. When a new event arrives, the system checks if an alert should be raised according to the previous events in the lists. This system works well in theory but its main inconvenient is that it takes a lot of RAM. When a rule matches a lot of events the list becomes huge and the system shuts down.

Another drawback of the actual correlation system comes from the correlation language. The workflow uses the STIX Patterning [21]. It is an efficient language when it comes to treat events individually because it is easy to write and understand. However, it is not possible to group events by attributes. The impossibility to group by attributes generates false positives and diminishes the efficiency of the system. The force brute is a good example to explain the need for correlation. This attack is defined by multiple tries to connect to a system made by the same computer. To detect it, one solution is to count the failed login attempts and raise an alert if there is more than 15 attempts in less than 5 minutes for example. If there are no group by the attribute

computer name, the 15 failed login attempts could come from different computer and could be unintentional.

3.2 Functional Requirements

The system has to be able to find a correlation between events in a very short period (seconds). A cyber attack can be very destructive and the faster the threat is detected the faster a countermeasure is taken. The goal is to achieve one minute between the receipt of the event and the triggering of the alert. The correlation component is the key component to detect multiple step attacks. It has to be fast because the countermeasure to mitigate the attacks has to be taken fast. The correlation engine has to be available all the time. The system handles thousands of events every seconds. If the system goes down for a period of time, the data arriving during this period of time cannot be lost and have to be analysed later. The project is expected to grow and the correlation system has to be scalable and adaptable. Spark is a scalable tool because one can add memory and processors if it is needed. However, adding resources can be very expensive. A scale effect is necessary for the system. The number of data and the number of queries must not be linearly dependent of the number of resources, a logarithmic dependency is expected for the system.

3.3 Security Requirements

The data the system analyses come from the customers. They can be critical and cannot leak after an attack or an intrusion. The confidentiality of the data is a top priority. The system has to be secure and cannot have breach. The encryption and authentication through the workflow is ensured with TLS protocol perform by Apache Kafka [17].

The integrity of the system is also vital. The logs received by the system should not be compromised by the adversaries. In that case, it could reduce the efficiency of the system. Some attacks may not be detected by the correlation system which have as consequence of reducing the defences of our customers. The messages go through the workflow with Apache Kafka. These technologies which are discussed later ensure the integrity of the messages among other things. It is achieved by the utilization of asymmetric keys and certificates between the services.

An adversary must not have the possibility to send false information into the system. If it happens the performances are reduced because the system treats useless information. For this reason we need authentication, authorization and access control.

The system has to be available all the time. Denial of Service (DoS) (or Distributed Denial of Service (DDoS)) attacks could jeopardize our system and its availability. The workflow is scaled to analyze a certain amount of logs per second. If a large amount of customers send an important volume of data, the system could become unavailable because it could not treat all those logs. To avoid this scenario, a quota is defined per clients by system.

3.4 Adversarial Model

Many attacks could jeopardize the correlation engine and its performances from a security point-of-view. Every system is vulnerable to different threats. The goal of this adversary model is to identify and mitigate the vulnerabilities to dissuade the adversaries to attack the system by making it difficult to threaten. The problem at hand is how to keep the correlation system operating always in a secure state. In other words, the aim is to secure the correlation system to the full extent so that it mitigates all possible attacks and thwarts the threats. It is targeted not to allow an imposter to stage an attack using the vulnerabilities and weaknesses within the system. The system is designed in a way that staging an attack and jeopardizing the system from the security aspects get difficult.

The scope of this thesis is to design and implement a correlation scheme to detect potential attacks or intrusion. The system treats logs received from customers, the adversary could be a compromised or a malicious client. We do not consider here the benign faults, for example, delaying or loss, which can occur either under normal operational conditions or due to equipment failure. The main focus is on the behavior of the adversaries to handle illegitimate and destructive activities.

Multiple vulnerabilities are identified, they are listed here and the next paragraphs detail them.

- DoS attacks.

- Compromising events integrity
- Interrupting events transmission.
- Other adversarial models.

3.4.1 Denial-of-Service (DoS) Attacks

The correlation scheme developed in this work is scaled for a platform that received 20,000 events per seconds. The number of events the system treats and analyzes depends on the number of correlation rules. Indeed, only the logs taking part into a correlation rule go through the system. We consider that around 3% of the events are treated by the correlation scheme. Its means that the system is designed to handle around 600 events per seconds. If an adversary manages to send a large volume of logs to the system, it could become unavailable and unable to detect malicious logs.

3.4.2 Compromising Events Integrity

The integrity of the logs received ensures a good threat detection. An adversary could reduce the efficiency of the system by compromising events integrity. The listing 3.1 shows an example of an attack and the detection rule associated to it.

```
A FOLLOWEDBY B GROUPBY username WITHIN 30 SECONDS
```

Listing 3.1: Example of a detection rule.

For the threat to be detected with the detection rule described in the listing 3.1, the usernames of the events A and B have to be identical. If an adversary manages to compromise the integrity of the message and change the username of one of them during the transmission from the customer to the platform, the threat cannot be detected and the system is fooled.

3.4.3 Interrupting Events Transmission

The correlation engine detects threat for our customers because it receives logs that gives it the information about what happens in their IT system. If an adversary manages to interrupt the transmission between the platform and the customer, the system becomes unable to analyze and correlate the events which make it useless to detect the threats.

3.4.4 Other Adversarial Models

The platform could be jeopardized by other threats than the ones detailed in the previous paragraphs. However for the shortness of this thesis and because there are considered out of the scope, we only discuss in this work the vulnerabilities that could deceive the correlation system or reduce its efficiency.

Chapter 4

Methods and Technologies

In this chapter, the tools used for this thesis are described. Those technologies are Apache Kafka, Elastic Search, Apache Spark and more specifically its module named Structured Streaming.

4.1 Apache Spark

4.1.1 Presentation of the Technology

Apache Spark [3] is a framework for distributed in-memory computing which allows Extract Transform and Load (ETL), analytic with a rich and complete library, Machine Learning and also graph processing on large volumes of data, with different formats in batch or pseudo real time. Spark processing can be developed with several different languages: Scala, Python, Java, SQL and the R language. Spark processes data in memory which means they are not stored in database when treated by Spark. Spark is deployed in a cluster with multiple nodes which executes multiple tasks in parallel.

The following section presents the functionalities of Apache Spark used in this project.

Apache Spark Core

The Apache Spark Application Programming Interface (API) is very friendly to developers, hiding much of the complexity of a distributed processing engine behind

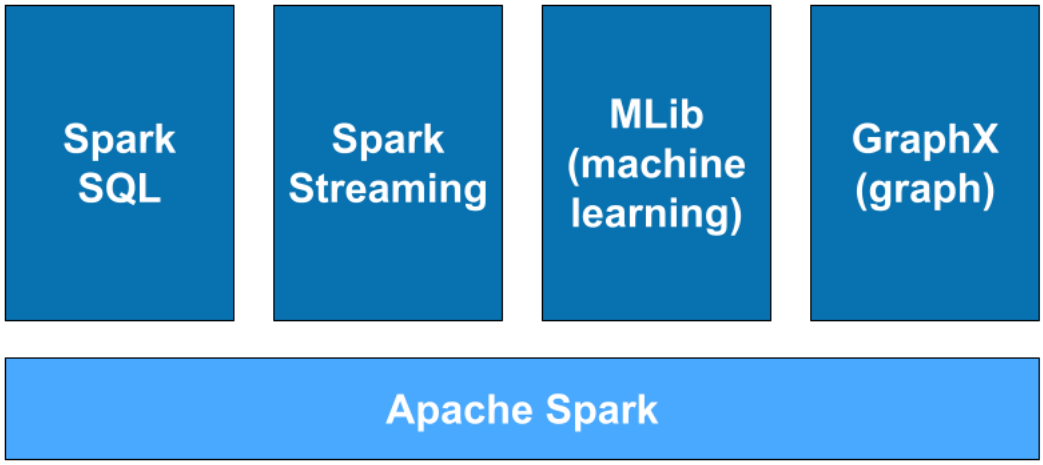

Figure 4.1.1: Components of the Spark Framework. The figure is taken from Spark's website [3].

simple method calls. The Resilient Distributed Dataset (RDD) API is implemented on Spark Core. RDD is a programming abstraction that represents an immutable collection of objects that can be split across a computing cluster. RDDs can be created from simple text files, SQL databases, NoSQL stores (such as Cassandra and MongoDB), Amazon S3 buckets, and much more besides. Spark runs in a distributed fashion by combining a driver core process that splits a Spark application into tasks and distributes them among many executor processes which execute them. These executors can be scaled up and down as required for the application's needs.

Spark SQL

It is likely the interface most commonly used by developers when creating applications. Spark SQL is focused on the processing of structured data, using a dataframe approach borrowed from R and Python. Apache Spark uses a query optimizer called Catalyst that examines data and queries in order to produce an efficient query plan for data locality and computation that will perform the required calculations across the cluster.

Spark Streaming

Spark Streaming extended the Apache Spark concept of batch processing into streaming by breaking the stream down into a continuous series of micro batches, which could then be manipulated using the Apache Spark API. In this way, code in

batch and streaming operations can share mostly the same code, running on the same framework, thus reducing both developer and operator overhead. It is not possible to use Spark SQL with Spark Streaming, this is why Spark Structured Streaming was developed.

Spark Structured Streaming

Structured Streaming allows the user to execute SQL queries on streamed data. The key idea in Structured Streaming is to treat a live data stream as a table that is being continuously appended. This leads to a new stream processing model very similar to a batch processing model. The user expresses his streaming computation as standard batch-like query as on a static table, and Spark runs it as an incremental query on the unbounded input table. The language is easy (type SQL). The idea is to create queries written in Structured Query Language (SQL). If the return table is empty, there is no alert raised. Otherwise, an insight must be sent. It can read data from Kafka and the output can also be a Kafka sink. Spark Structured Streaming is able to run multiple queries successfully in parallel on data with changing velocity and volume sizes.

4.1.2 Spark Functionalities

Spark is faster than its competitors because of its cheaper shuffle steps. Spark limits its reads/writes operations on disk and stores the data of the intermediate steps in memory.

Spark transformations are called lazy, which means that when you execute Spark transformation functions, the framework does not execute them immediately but keeps a record of the called function. All these operations build a Directed Acyclic Graph (DAG). It is the set of operations of a Spark processing is executed when an action type function is invoked in the program. In lazy evaluation, data is not loaded from the source until it is needed. This avoids costly and unnecessary code execution and improves performance during execution.

4.1.3 Spark Application

Spark Applications consist of a driver process and a set of executor processes. The driver is responsible for analyzing, distributing and scheduling work across the executors. The executors are responsible for executing the work assigned by the driver

and reporting the state of the computation to the driver. One of the aspects of this thesis is to create Spark applications that apply detection rules on the event stream.

4.2 Managed Spark: OVHcloud

This section explains the advantages of using a managed Spark instead of implementing our own Spark cluster. The main advantages of using a managed Spark are to leave the implementation of Spark aside and focus on the performances and the possibilities. As the work of this thesis is to evaluate the feasibility of a correlation engine based on Apache Spark, it has been decided to use a managed Spark at the beginning and if the results are conclusive, a Spark cluster could be implemented internally.

OVHcloud is the cloud provider of the platform In this thesis, the data processing product of OVHcloud is used. The philosophy of this product is the job as a service which means that for each job, a certain amount of resources (RAM and cores) are used and the user does not need to handle the cluster implementation and just has to decide its size.

4.3 Apache Kafka

This section presents the Apache Kafka technology, named Kafka in the following. It links the micro-services and is used with Apache Spark in this project.

The Kafka software is an open source stream processing platform [15]. Its main function is the centralization of data flows. The primary function of Kafka is to optimize the transmission and processing of data streams that are directly exchanged between the data recipient and the source. Kafka acts as a messaging instance between the sender and the receiver and offers solutions to solve the problems usually associated with this type of connection. Kafka is executed as a cluster which centralize the data. The data are distributed in different topic. Each topic corresponds to one queue. In each topic the data are distributed in different partitions.

The system uses Kafka as a queue between the different micro-services of the workflow. The logs in the Kafka topics are stored in JavaScript Object Notation (JSON) format. The Kafka of the workflow is only available from the internal network of the platform.

In the thesis, the correlation engine based on Apache Spark receives data from a Kafka topic and sends the results in another Kafka topic.

Kafka with Apache Spark

The data in the Kafka topics can be read by Spark. As a managed Spark is used, the Kafka has to have an opening on the Internet. The first faced challenge had been to find a way to do that in a secure way without create a new Kafka cluster. For this purpose, a Kafka-proxy has been installed [18]. It allows the user to read the Kafka topics inside the internal network. In the future, this technology will be changed because it is not stable enough for production. Indeed, it does not support the volume of data handle by the platform. One solution is to use another Kafka cluster open on the Internet and send it the useful events of our internal cluster. In this case if the cluster opened on the Internet is compromised the attacker cannot trace back to the internal Kafka. Another solution could be to install a VPN between the internal network of the platform and the OVHCloud.

4.4 Elastic Search

The following section presents Elastic Search and some of its query languages. This technology is used in the second part of this thesis as another alternative to create a correlation system. Elastic Search is a search and analysis engine [7] where users can store and query their data. Elastic Search is deployed on a cluster. Each node corresponds to a running instance of Elastic Search, and can be added to or removed from the cluster even when the cluster is running. Its distributed structure makes possible to do real-time search on the stored data.

The events sent by the customers and treated by the first part of the workflow are stored in the Elastic Search. The basic way to do queries with Elastic Search is to use *Query DSL* [23]. As it will be seen later, this language it not sufficient when looking for succession of events. A new language called *EQL Search* [8] is in beta version on the Elastic Search and can detect sequence of events.

4.4.1 Query DSL

Query DSL interests us because it can realise aggregation on the data that are stored in the Elastic Search. It can also practice filters on the data. The operator AND, OR, NOT have equivalent in query DSL. The idea to correlate events is to practice filter on a period of time and some criteria and then aggregate on different fields. It can also compute the cardinality.

4.4.2 EQL Search

EQL Search is a query language designed for threat hunting. We want to use this language to detect sequence of events. The listing 4.1 gives an example of a detection rules in STIX Patterning. The same rules is written in EQL search in the listing 4.2.

```
([EVENTA] FOLLOWEDBY [EVENTB]) WITHIN X SECONDS
```

Listing 4.1: Example of a rule written in STIX Patterning.

```
{"filter":
    {"range" :
        {"@timestamp" :
            {"gte" : "-nowX", "lte" : "now"}
        }
    },
"query":
    """sequence
        [EVENTA]
        [EVENTB]
    """
}
```

Listing 4.2: Example of a rule written in EQL search.

EQL Search can also group the sequence of events on a common field for example a username. The detection of sequence of events is primordial in threat detection. A known attack is analysed by the security team and they create a detection rule to find track of this attack. The detection of just one event lead most of the time to false positive while a sequence of events is more accurate. EQL search which is still in beta version is one of the only tool to propose this type of detection.

Table 4.5.1: Correlation operators.

Operators	definition
followedby	detect sequences of events
repeat	detect repetition of events
within	define the period of correlation
groupby	group the events by the chosen attribute

4.5 Detection Languages

The detection language used by the system is a thorny issue. This language needs to be written by the customers and users of the system but it has to be complete enough to includes all possible use-cases. The language has to handle the correlation operators of the table 4.5.1. In this project, two correlation languages are evoked: STIX Patterning [21] and Sigma [27].

4.5.1 STIX Patterning

The STIX Patterning is the detection language used at the beginning of this work. This language has many advantages and allows detecting many type of attacks (this is why it was choosen as a detection language) but it is more adapted to detection on single events. The correlation with STIX is rather limited: It contains the operators *followedby*, *repeat* and *within* but not *groupby*.

4.5.2 Sigma

Sigma is a generic and open signature format that allows describing relevant log events. The writing of correlation with sigma is changing [26] and we wants to use it as a detection language.

4.5.3 Examples

The two following rules detect the same behavior. The sigma one has a *groupby* operator that does not exist in the STIX Patterning detection language. For the need of the work, it is supposed in some part of the thesis that the STIX patterning language has a *groupby* operator. The Sigma language is more complicated to write and understand

but it enables the analysts to add some descriptions and details about the rules.

STIX Patterning

```
[x-event:id=4625] REPEAT 100 WITHIN 3600 SECONDS
```

Listing 4.3: Example of a rule written in STIX Patterning.

Sigma Correlation

```
title: chain correlation
name: failed_login
description: Detect failed logins
author: Clémence Dubuc
date: 2021/02/15
logsource:
  product: windows
  service: sysmon
detection:
    selection:
        EventID: 4625
    condition : selection
falsepositives:
  - unknown
level: medium
---
action: correlation
name: many_failed_logins
type: event_count
rule: failed_login
group-by:
- ComputerName
timespan: 1h
condition:
    gte: 100
```

Listing 4.4: Example of a rule written in Sigma.

4.5.4 Comparison between the Correlation Languages

	STIX Patterning	Sigma
ease of understanding and writting	High	Medium
Used in the system	Yes	No
Used in threat detection	Occasionally	Frequently
Operator *within*	Yes	Yes
Operator *followedby*	Yes	Yes
Operator *repeat*	Yes	Yes
Operator *groupby*	No	Yes

Table 4.5.2: Table of comparison between the STIX Patterning and Sigma

The table 4.5.2 compares the two correlation languages. With all those elements, the correlation language chosen for the new system is Sigma.

Chapter 5

Implementation

5.1 Implementation of the Apache Spark System

The Apache Spark system receives events from the system, analyzes them and return alerts if necessary. This chapter focuses on the Spark applications created to implement the detection rules.

5.1.1 Input

The logs are stored in a Kafka before being treated by Spark Structured Streaming. The data format includes *value* and *timestamp*. *Value* stores the log in JSON format. It includes all the information about the events. *Timestamp* gives at which time the logs is entered into the Kafka topic. It is useful for the tests and for debug but in the real implementation, the timestamp in the JSON value will be used because it represents the time the event occurs. In this JSON object, we are interesting in the *obexs* which is the attribute we are looking for. An *obex* is the diminutive of observable expression. It denotes an attribute like listing 5.1 but in an optimized way. A hash function is used to do the transformation. The hash function computes a digital fingerprint (a string in our case) to quickly identify the original data.

```
[ipv4-addr:value='127.0.0.1']
```

Listing 5.1: Example of an observation expression (obex).

The JSON object also contains *idcommunity* which is the attribute that identifies the customer and *definition* which is the attribute that contains all the information for the

triggering of the alert.

The following code listing 5.2 imports the data, it is explained in the next paragraphs.

```
df_event = spark
 .readStream
 .format("kafka")
 .option("kafka.bootstrap.servers", "kafka.test:9090")
 .option("subscribe", "sic.solved_events")
 .option("kafka.security.protocol", "SSL")
 .option("kafka.ssl.truststore.location", "client.truststore.jks")
 .option("kafka.ssl.truststore.password", "password")
 .option("kafka.ssl.keystore.location", "client.keystore.jks")
 .option("kafka.ssl.keystore.password", "password")
 .option("startingOffsets", "latest")
 .option("failOnDataLoss","false")
 .load()
 .selectExpr("CAST(value AS STRING)")
 .select(from_json(col("value"), schema).alias("df"))
 .select(col("df.obexs").alias("obexs"),
     element_at(col("timestamp"),
     col("df.definition.id").alias("id_community"),
     col("df.definition").alias("definition"))
```

Listing 5.2: Data import code.

SSL Protocol

As we use the Spark on OVHcloud we need to send them our data thought the Kafka. To do that, we use a proxy Kafka. The Spark talks to our proxy Kafka to access logs. It is done with an SSL protocol. The certificate and the key are stored in the files *client.truststore.jks* and *client.keystore.jks*.

StartingOffsets

The function *startingOffsets* tells the Spark when it has to start reading the Kafka. There are two modes *latest* and *earliest*.

- **Latest**: Spark starts reading from the end of the queue.
- **Earliest**: Spark starts reading from the beginning of the queue.

5.1.2 Output

Once the log is treated by the Spark. The output is sent in another Kafka topic. The information send in the topic are the matched rules and the definition of the log. For the test, the timestamp is also sent.

The listing 5.3 returns the results, it is explained in the next paragraphs:

```
return_results = join
        .writeStream
        .option("checkpointLocation", "query1/checkpointlocation")
        .outputMode("append")
        .queryName("query")
        .format("kafka")
        .option("kafka.bootstrap.servers", "kafka.ftest:9090")
        .option("topic", "cdu-test-correlation")
        .option("kafka.security.protocol", "SSL")
        .option("kafka.ssl.truststore.location", "client.truststore.jks")
        .option("kafka.ssl.truststore.password", "password")
        .option("kafka.ssl.keystore.location", "client.keystore.jks")
        .option("kafka.ssl.keystore.password", "password")
        .start()
```

Listing 5.3: Code returning the results of the query.

Output Modes

There are three output modes for the Kafka output : complete, append and update

- **Complete**: At each micro-batch, this output mode releases all the previous data, it cannot be use in our system.
- **Append**: At each micro-batch, this output mode releases the new logs.
- **Update**: A mix between append and complete. It releases new logs and the logs which have been updated. (example : new logs in a aggregation window.)

CheckpointLocation

The checkpointLocation function is used in case of query failure. Spark stores all the data that are needed for recovery in a specific location. If the query fails the

system can restart without losses by retrieve the needed data. It makes our code fault-tolerant.

5.1.3 Logic

Functions

This session describes the different functions that are used in the SQL request. All those functions are used to implement the logic of the correlation rules and the way the events are processed according to the correlation rules.

- **Element_at** : Have a certain item of a list (is used when the elements of a column are the list type)
- **Filter** : Filter a column on some conditions.
- **GroupBy** : Group the the element by attribute.
- **Join** : Join two dataframes on certain conditions.
- **Select** : Select some columns of the dataframe.
- **Size** : Have the size of a list. (Is used when the elements of a column are of the list type)
- **window** : Group the element of a column by window of time. The following code groups the logs by windows of 30 minutes every 15 minutes.

```
window("timestamp", "30 minutes", "15 minutes")
```

- **WithColumn** : Add a column to a table.
- **WithWatermark** : Handle the late logs. The logs are deleted from the memory after a certain among of time. If it arrives before this limit, it is put in the good window, if it arrives after it will not appear in the data.

Repeat Logic

The *repeat* rules count the number of *obexs* 'x' for a community on a period of time. If the number of *obexs* 'x' becomes higher than 'y', an alert is created. As explained in the beginning of the section, an *obex* or *observable expression* is an attribute of a log. It can be anything from the IP address to the username. The *repeat* rule is written in the listing 5.4 in STIX Patterning language.

```
[obex = x] REPEAT y GROUPBY community WITHIN t SECONDS
```

Listing 5.4: Correlation rule with the operators *groupby* and *repeat* in STIX Patterning.

The *repeat* is written python rule in the listing 5.5. Apache Spark on OVHcloud can be used with both of those two languages. The complete code can be found in the appendices of the thesis. The SQL query logic is developed in the following code. The *df_events* is the stream table of the logs. We suppose that the logs are already extracted from the Kafka. The extraction is developed in the Kafka section.

```
rule = "[obexs = x] repeat y GROUPBY idcommunity WITHIN t MINUTES"

query = df_event
    .withWatermark("timestamp", "2*t seconds")
    .filter(array_contains(col("obexs"),"x"))
    .groupBy(window(col("timestamp"), "2*t seconds", "t seconds"),
                  col("id_community"))
    .agg(collect_list(col("definition")).alias("list_def"),
        collect_list(col("timestamp")).alias("time"))
    .select(size("list_def").alias("count"),
        element_at("list_def",1).alias("def"),
        element_at("time",1).alias("time1"),
        element_at("time",size("list_def")).alias("time2"))
    .filter("count > y-1")
    .withColumn("rule", lit(rule))
    .select(to_json(struct(col("def"),
        col("count"),
        col("time1"),
        col("time2"),
        col("rule"))).alias("value"))
```

Listing 5.5: Code of a correlation rule with the operators *groupby* and *repeat* with Apache Spark.

This rule filters the events with the *obex* 'x', groups them by window of time and by community, counts them and returns a result if the count is higher than 'y'. In the return data, there are the definition of the events, the count, the time the first events is received, the time the last event is received and the match rule.

Followed by Logic

The *followedby* rules look if an *obex* 'a' is followed by an *obex* 'b' in a period of time t. If it is the case, an alert is raised. The *followedby* rule is written in the listing 5.6 in STIX Patterning language.

```
[obex= a] FOLLOWEDBY [obex = b] WITHIN t SECONDS
```

Listing 5.6: Correlation rule with the operators *followedby* in STIX Patterning.

The code of the rule is pretended in the listing 5.7 and explained in the next paragraph.

```
rule = "[obexs = a] FOLLOWEDBY [obexs = b] WITHIN t SECONDS"

query1 = df_event #stream of events
    .withWatermark("timestamp","t")
    .filter("obexs == 'a'")

query2 = df_event #stream of events
    .withWatermark("timestamp","t")
    .filter("obexs == 'b'")

join = query1.as("a")
    .join(broadcast(query2).as("b"),
      expr("a.id_community = b.id_community
        AND a.timestamp < b.timestamp
        AND b.timestamp < a.timestamp + interval t seconds"))
    .withColumn("rule", lit(rule))
    .select(to_json(struct(col("a.definition"),
      col("a.timestamp"),
      col("b.timestamp"),
      col("rule"))).as("value"))
```

Listing 5.7: Code of a correlation rule with the operator *followedby* with Apache Spark: method 1.

This rule filters the events with the *obexs* 'a' or 'b'. The two tables are joined on certain conditions. If those conditions are met, the code returns a result. The out put contains the definition of the event containing the *obex* 'a', the times of the events and the name of the rule.

The *followedby* logic has a performance problems. Because of the *join* operator, the

processing time was around 2s per batch whereas in the *repeat* logic, the *aggregation* operator makes the processing time shorter (around 0.5s per batch). Another way to do the *followedby* rule has been developed. The code is presented in the listing 5.8 and explained in the next paragraph.

```
rule = "[obex='a'] FOLLOWEDBY [obex='b'] WITHIN t SECONDS"

aggregation = df_event
    .withWatermark("timestamp", "2*t minutes")
    .withColumn("realobexs", when(array_contains(col("obexs"),"a"),
        "obexsa")
    .when(array_contains(col("obexs"),"b"),"obexsb")
    .otherwise("unknown"))
    .filter("realobexs == 'obexsa' OR realobexs == 'obexsb'")
    .withColumn("amount", lit(1))
    .groupBy(window(col("timestamp"),"2*t", "t"),
        col("id_community"))
    .agg(expr("sum(if(realobexs = 'obexsa',
            amount, null))").alias("A"),
        expr("sum(if(realobexs = 'obexsb',
            amount, null))").alias("B")),
        expr("first(if(realobexs = 'obexsa',
            definition, null),true)").alias("defA"),
        expr("first(if(realobexs = 'obexsa',
            timestamp, null),true)").alias("timeA"),
        expr("first(if(realobexs = 'obexsb',
            timestamp,null),true)").alias("timeB"))
    .filter("A > 0 AND B > 0 AND timeA < timeB")
    .withColumn("rule", lit(rule))
    .select(to_json(struct(col("defA"),col("A"), col("B"),
        col("timeA"), col("timeB"), col("rule"))).alias("value"));
```

Listing 5.8: Code of a correlation rule with the operator *followedby* with Apache Spark: method 2.

This rule creates a new attributes in the dataframe when *obex* 'a' or *obex* 'b' are seen. The rule filters on those attributes. It allows avoiding the *join* operation. The events are then grouped by windows of time and community. After that the rule counts the number of A and B and researches the first A and the first B. If there is more than one A and one B and if A has appears before B an result is returned. This result contains the definition of A, the count of A, the count of B, the time of the first A, the time of the

Rule #	Rule
1	([process:name='wscript.exe'] FOLLOWEDBY ([process:name='rundll32.exe'] OR ([process:name='wmiprvse.exe'] FOLLOWEDBY [process:name='rundll32.exe'])) FOLLOWEDBY [process:name='msiexec.exe']) WITHIN 60 SECONDS
2	[x-action:id = 4625] REPEATS 200 TIMES WITHIN 60 SECONDS
3	([process:name='cmd.exe' AND process:command_line LIKE '%WMIC.exe%shadowcopy%delete'] FOLLOWEDBY [process:name='cmd.exe' AND process:command_line LIKE '%vssadmin.exe%Shadows%/all%/quiet'] FOLLOWEDBY [process:name='cmd.exe' AND process:command_line LIKE '%bcdedit%/set%default%recoveryenabled%No%']) WITHIN 30 SECONDS
4	([process:command_line LIKE '%net%view%/all'] FOLLOWEDBY [process:command_line LIKE '%net%view%/all%/domain'] FOLLOWEDBY [process:command_line LIKE '%nltest%/domain_trusts%/all_trusts']) WITHIN 30 SECONDS
5	(([x-action:id=4625 AND x-action:properties[*].WorkstationName=$WORKSTATIONNAME] REPEAT 10 TIMES) FOLLOWEDBY [x-action:id=4624 AND x-action:properties[*].WorkstationName=$WORKSTATIONNAME]) WITHIN 300 SECONDS
6	([A = 'b'] FOLLOWEDBY [A != 'b']) REPEAT 1 GROUPEDBY C WITHIN 300 SECONDS
7	[A = 'b' OR A = 'c'] REPEAT 5 WITHIN 1 HOUR
8	([A = 'a'] AND [B = 'b'] AND [C = 'c'] AND [D = 'd'] AND [E = 'e'] AND [F = 'f'] AND [G = 'g'] AND [H = 'h']) REPEAT 1 GROUPEDBY N WITHIN 30 SECONDS

Table 5.2.1: Examples of correlation rules.

first B and the name of the rule.

5.2 Detection Rules

In the previous section, the different logics to create a detection rule has been presented. This section shows some detection rules.

Those rules are written by the Threat, Detection and Research (TDR) team. They correspond to a need of detection of some type of attacks. The detection language used to write those rules is the STIX Patterning [21]. Examples of those correlation rules are presented in the table 5.2.1.

Rule #	Rule
1	([A='a'] FOLLOWEDBY ([B='b'] OR ([X='x'] FOLLOWEDBY [B='b'])) FOLLOWEDBY [C='c']) WITHIN 60 SECONDS (A.2)
2	[A='a'] REPEATS 200 TIMES WITHIN 60 SECONDS (A.4)
3	([A='a'] FOLLOWEDBY [B='b'] FOLLOWEDBY [C='c']) WITHIN 30 SECONDS (A.6)
4	(([A='a'] FOLLOWEDBY [B='b'] FOLLOWEDBY [C='c']) WITHIN 30 SECONDS (A.8)
5	(([A='a'] REPEAT 10 TIMES) FOLLOWEDBY [B='b']) WITHIN 300 SECONDS (A.10)
6	([A = 'b'] FOLLOWEDBY [A != 'b']) REPEAT 1 GROUPEDBY C WITHIN 300 SECONDS (A.12)
7	[A = 'b' OR A = 'c'] REPEAT 5 WITHIN 1 HOUR (A.14)
8	([A = 'a'] AND [B = 'b'] AND [C = 'c'] AND [D = 'd'] AND [E = 'e'] AND [F = 'f'] AND [G = 'g'] AND [H = 'h']) REPEAT 1 GROUPEDBY N WITHIN 30 SECONDS (A.16)

Table 5.2.2: Simplified correlation rules.

As each observable expressions are transformed into an hash in the system [21], only the temporal logic of the rules is considered. The transformation into hash in done by another micro-service already implemented in the system. The table 5.2.2 represented the simplified correlation rules.

The codes of the detection rules of the table 5.2.2 are in the appendix.

5.3 Implementation of the Elastic Search System

5.4 Approach

Elastic Search is a search engine and analytic engine. The correlation rules are written in Sigma and translated in Elastic Search query by the system.

The approach with Elastic Search is different from the approach with Spark. The events are stored in the Elastic Search and each time an event which can take part of a correlation rule is received an Elastic Search query is launched. The idea behind that is to create an impression of real-time as if the events where analysed as a stream of data.

This approach has another advantage: it avoids making useless aggregation. If the rule contains a *groupby* on an attribute and is associated with an event, the *groupby* is realized by a filter with the attributes value corresponding to the events instead of an aggregation.

5.4.1 Time Filter

To avoid the research on all the database, the query is limited to the time of the correlation rules. Each correlation rules has an associated time in which a certain sequence of events has to happen for an alert to be raised. When an event that could take part of this rule is seen, a query is launched only on the last five hours (if the correlation time is five hours) . This approach reduces the search time and avoids time-consuming aggregations.

Example

The listing 5.9 is an example of correlation rule written in Sigma.

```
action: correlation
name: many_failed_logins
type: event_count
rule: failed_login
timespan: 1h
condition:
    gte: 100
```

Listing 5.9: Correlation rule with the time filter in Sigma.

The course of action of this rule is:

1. A failed login event is received.
2. A filter on the events of the last hour is applied.
3. A query counts the number of failed login events and if this number exceeded 100, a alert is raised.

5.4.2 Groupby Filter

Some correlation rules involve *groupby* operations. An important number of failed authentications is a sign of attacks only if it comes from the same person. The easier way to create this kind of rules is to aggregate the events by computer name, IP address or user name. For example to count the number of logs sends by each user during the last hour, the logs can be aggregate by username and count. Elastic Search makes this kind of operation but it is expensive in term of time and resources. The search engine has to group all the events according to one or more attributes.

The approach we chose makes possible this type of rule in a less expensive manner. As the queries are launched with an event, the *groupby* operations can be done by filtering: If the rule groups by a certain attribute, to applied the rule on the event, it is enough to filter by the value of the attribute in the event associated to the rule.

Example

The listing 5.10 is an example of correlation rule written in Sigma.

```
action: correlation
name: many_failed_logins
type: event_count
rule: failed_login
group-by:
    - user.name
timespan: 1h
condition:
    gte: 100
```

Listing 5.10: Correlation rule with the groupby filter in Sigma.

The course of action of this rule is:

1. A failed login event with *user.name* = *Joe* is received.
2. A filter on the events of the last hour and on the events with *user.name* = *Joe* is applied.
3. A query counts the number of failed login events *user.name* = *Joe* with and if this number exceeded 100, an alert is raised.

5.5 Detection Rules

The Sigma correlation language defines three main types of correlation:

- **Event count**: Count the number of occurrence of an event. It is the equivalent of the operator *repeat* in STIX patterning.
- **Value count** : Count the number of different value a field can take.
- **Temporal proximity**: Discover patterns. It is the equivalent of the operator *followedby* in STIX patterning.

The Elastic Search cluster is already implemented in the system with all the logs of the customers. The first tests realized on this cluster showed that the queries take less than a second to return a result. This is promising but as the new workflow is not yet implemented the real performances of the Elastic Search correlation engine is not known. The correlation system based on Elastic Search requires to change the workflow: The events are analyzed individually and stored in Elastic Search in a specific index used only for correlation. This index is queried each time an event that takes part of a correlation rule arrives in the system.

One of the implementation made during the second part of this thesis was to create a translator between the Sigma language and Elastic Search. This translator is coded in Python. It receives as input the events and the rules in Sigma in a python dictionary and it returns the corresponding Elastic Search query. The translation between the two languages is developed in the next sections.

5.5.1 Event Count

Event count is the equivalent in Sigma of the operator *repeat* in STIX Patterning.

The *event count* rule is written in Sigma in the listing 5.11. The rule is explained in the next paragraph.

```
action: correlation
name: many_failed_logins
type: event_count
rule: failed_login
group-by:
    - user.name
```

```
timespan: 1h
condition:
    gte: 100
```

Listing 5.11: Event Count correlation rule written in Sigma.

This query counts by username the number of events corresponding to a failed login and returns a result only if the count is higher than 100. The query is launched when an event corresponding to a failed login is received. This process allows creating a real-time rules and to avoid expensive aggregation.

The *failed_login* corresponds to *event.id* = 4625 in windows events.

We suppose the event received has the following attributes:

- event.id = 4625
- user.name = Joe

The rule translated in DSL query gives the listing 5.12.

```
{
  "query": {
    "bool": {
      "must": [
      {"range": {
      "@timestamp": {
        "gte": "now-1h",
        "lt": "now"
      }}},
      {"term": {"event.id": "4625"}},
      {"term": {"user.name": "Joe"}}
      ]
  }}}
```

Listing 5.12: Event Count correlation rule translated in DSL query.

This query returns the number of corresponding events and the events themselves. If this number is higher than 100 an alert is raised.

5.5.2 Value Count

Value count computes the cardinality of an attribute. It means the number of different value the attribute takes during a period of time. A *value count* rule is written in Sigma in the listing 5.13. The rule is explained in the next paragraph.

```
action: correlation
type: value_count
rule: failed_login
field: user.name
group-by:
    - computer.name
timespan: 1d
condition:
    gte: 100
```

Listing 5.13: Value Count correlation rule written in Sigma.

This query counts the number of different user.name uses by each computer.name on the failed login attempt.

We suppose the event received has the following attributes:

- event.id = 4625
- user.name = Joe
- computer.name = system

The rule translated in DSL query gives the listing 5.14.

```
{
  "query": {
    "bool": {
      "should": [ }
      ],
      "must": [
      {"range": {
      "@timestamp": {
        "gte": "now-1d",
        "lt": "now"
```

```
        }
    }},
    {"term": {"action.id": "4625"}},
    {"term": {"computer.name": "system"}}]
    },
    "aggs": {
        {"user_name": {
                "cardinality": {
                "field":"user.name"
              }
            }
}}}
```

Listing 5.14: Value Count correlation rule translated in DSL query.

This query returns the number of different user.name detected in the last day with computer.name = system and event.id = 4625. If this number is higher than 100, an alert is raised.

5.5.3 Temporal Proximity

Temporal proximity detects sequences of events during a period of correlation. It is the equivalent in Sigma of the operator *followedby* in STIX Patterning. A *temporal proximity* rule is written in Sigma in the listing 5.15 and is explained in the next paragraph.

```
action: correlation
type: temporal
rule:
    - recon_cmd_a
    - recon_cmd_b
    - recon_cmd_c
group-by:
    - user.name
timespan: 5m
ordered: true
```

Listing 5.15: Temporal proximity correlation rule written in Sigma.

This query checks the sequences of events in the last five minutes.

We suppose an event corresponding to *recon_cmd_b* is received. The following query in Elastic Search is launched. The rule on Elastic Search is translated in the listing 5.16. It is written in EQL search.

```
{"filter": {
    "range" : {
      "@timestamp" : {
        "gte" : "now-5m",
        "lte" : "now"
      }
    }
  },
  "query": """
  sequence by user.name
      [ any where action.id == "524" ]
      [ any where action.id == "506" ]
      [ any where action.id == "703" ]
  """
}
```

Listing 5.16: Temporal proximity correlation rule translated in EQL Search.

The query returns the number of sequence corresponding to this sequence of attributes. If a sequence is detected, an alert is raised.

Chapter 6

Qualitative Analysis

This section describes how the potential attacks on the correlation system described in the adversarial model are mitigated. An adversary could threaten the system by DoS (or DDoS). The workflow is scaled to analyzed a certain amount of data. If multiple customers are malicious or hacked and begin to forward a very large volume of logs, the system could become unavailable because it could not treat all of them. The mitigation of this attack is out of the scope of the thesis but we rely on other functionalities to avoid it. The correlation system retrieves logs from a Kafka topic. It only treats the number of events it could handle. In case, the correlation system is too slow, the events are stored in the Kafka until they are analyzed by the system. Nevertheless, a first filter is made by the platform at the beginning of the workflow to be sure that the incoming logs are legitimate.

An adversary could also threaten the good detection of the attacks by blocking the transmission of the events of our customers. The blocking could be complete or partial. If the adversary blocks completely the transmission between the client and the platform, the team dedicated to the customers in the company realizes there is a problem in the transmission, notifies the client and advises it. However if the blocking is only partial, the flow of data do not change a lot and it could be missed. In that case, the protocols used to forward events from customers to the platform gives a security over packets loss and packets interruption because they are using Transport Control Protocol (TCP).

The falsification of events is also a weakness for the correlation scheme. Indeed, if the logs received do not contain the correct information about what happens in the

IT system of the customers, it could not detect the potential intrusions or attacks. To mitigate this vulnerability the system uses the Secure Sockets Layer (SSL) protocol with Apache Kafka between the components of the workflow. It ensures the authentication and the authorisation. The integrity is ensured by certificates and asymmetric keys. Those mechanisms prevent the falsification of logs inside the workflow. To be sure that the events are not modified during the transmission from the customer to the platform, the logs are sent by secure protocol such as HTTPS, syslog over TLS, cloud hosting polling or Virtual Private Network (VPN). For the shortness of this thesis and because it is out of the scope, all the mechanisms quoted in this paragraph are not detailed in this work.

The platform could be threaten by other attacks that do not directly concern the correlation system. For those vulnerabilities, we rely on other mechanisms not detailed in this work. For availability of the servers, we rely on the security mechanisms of the cloud. Indeed, the platform is hosted on the public cloud of OVHcloud [22]. They ensure the security of the infrastructure and its availability. For the authentication of the users, an authentication is done with Two-Factor Authentication (2FA) and in some cases, certificates. Many other protocols and mechanisms are in place to protect the platform and could not be detailed here because it is out of the scope of the thesis.

Chapter 7

Quantitative Analysis

7.1 Apache Spark

This section presents the results of the work on Apache Spark.

7.1.1 Performance Evaluation

In this session, the tests to evaluate the performance of Apache Spark on our system are presented. For the performance evaluation, Apache Spark is working on OVHcloud. The criteria Spark has to complete are the following:

- Possibility of handling hundreds of streamed rules in the same time.
- Possibility of treating the stream events in a very short time.

The time between the receipt of the events and the triggering of the alert has to be less than a minute. During those tests, only the availability, the efficiency and the scalability of the system are evaluated.

The performed tests are:

- Correlation time
- Number of events
- One rule per job
- Multiple rules per job

parameter	value
number of events	5 eps
number of rules	1
driver memory	4G
executors memory	4G
number of executors	2
number driver cores	1
number executors cores	2

Table 7.1.1: Parameters of the job for the test to evaluate the influence of the correlation time.

values of t (min)	0.5	1	15	60	180

Table 7.1.2: Different values of the correlation time during the test to evaluate its influence.

A job is an instance created by OVHcloud to run the detection rule. A certain number of resources is allocated to each job.

Correlation Time

The goal of this test is to understand the influence of the correlation time on the performance of the system. The parameters of the jobs are presented in the table 7.1.1.

The values of the correlation time t taken during this test are in the table 7.1.2.

The results of the test are given in the figure 7.1.1.

Number of Events

The goal of this test is to understand the influence of the number of events in the Kafka topic on the performance of the system. The parameters of the job are given in the table 7.1.3.

The values of the number of events E taken during this test are in the table 7.1.4.

The results of the test are given in the figure 7.1.2.

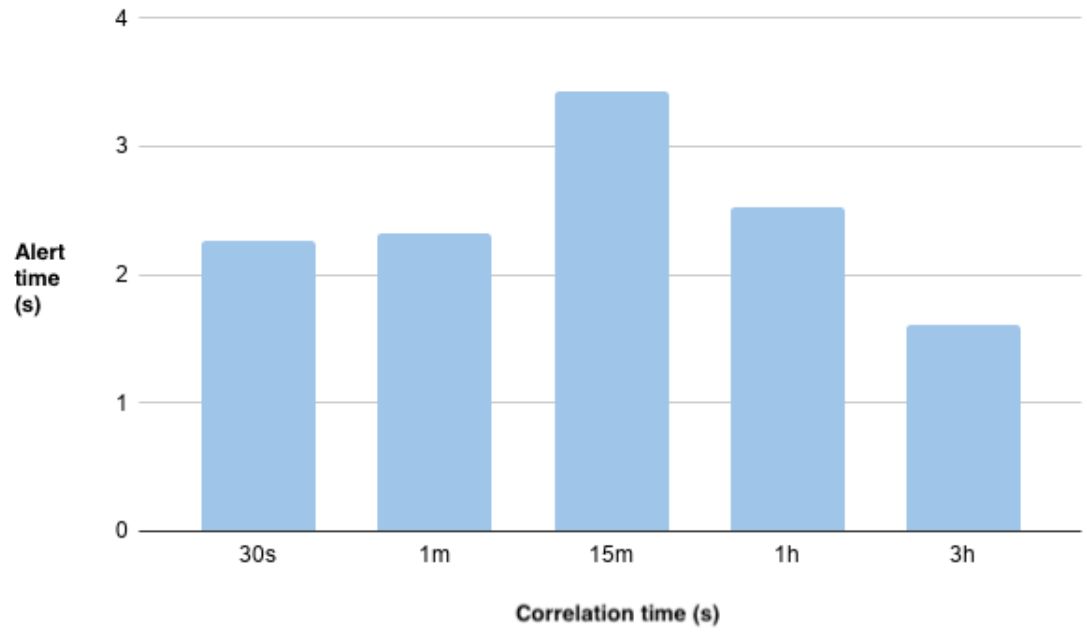

Figure 7.1.1: Alert time as a function of correlation time.

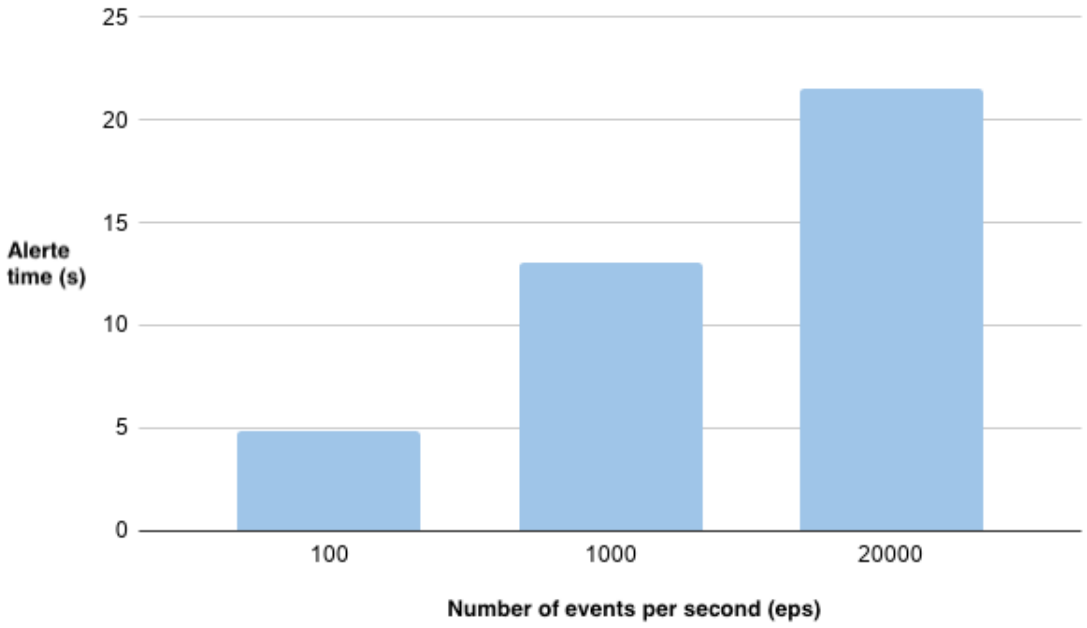

Figure 7.1.2: Alert time as a function of the number of events.

Test: One Rule per Job

In the following tests, the number of events and the correlation time are fixed. Eight rules are launched at the same time. Each rule is launched in a distinct job. The resources allocated to each job are :

parameter	value
correlation time	30s
number of rules	1
driver memory	4G
executors memory	4G
number of executors	2 or 4
number driver cores	1
number executors cores	2

Table 7.1.3: Parameters of the job for the test to evaluate the influence of the number of events.

values of E (eps)	10	100	20000

Table 7.1.4: Different values of the number of events per second during the test to evaluate its influence.

- 1 driver with 2G of RAM and 2 cores.
- 1 executor with 2G and 4 cores.

The jobs are supposed to be independent from each other. When a job is launched, no effect is expected to be seen on the other jobs.

During the first tests, each job launched slows down the previous ones. This behavior is not supposed to happen. OVHcloud team has been contacted about this issue. They changed the infrastructure replacing old server by new ones. After this modification the behavior of the application was as expected.

The rules handle 600 eps and run during an entire day without slowing down or failing. The batch duration is around 1 second. The availability and the efficiency are good.

This result is a good start but it means that each time a rule is added, the resources needed for running this rule are added. There is no scale effect and for having thousands of rules, a lot of resources are necessary. The following tests may help with these issues.

parameter	value
number of events	600 eps
correlation time	around 1 minute
number of alert per rule	one every 20s

Table 7.1.5: Parameters for the test.

test number	number of rules	RAM	number of executors	batch duration
1	1	3G	1	0.8s
2	2	3G	1	6s
3	3	3G	1	10s
4	4	3G	1	failed
5	20	5G	8	failed

Table 7.1.6: Resources per job for the test.

Test: Multiple Rules per Job

In this second test, instead of launching one rule per job, all the rules are launched in the same job. The based parameters are presented in the table 7.1.5.

Each job has one driver with four cores. Each executor has four cores. The number of executors and the RAM changes depending on the result of the previous test. The RAM is the same for the driver and for the executors. The number of resources per job is summarized in the table 7.1.6.

Two observations have been made during those tests:

1. The batch duration increases with the number of rules. It can be explained by the fact that there is less processing resources for each rule. It can be resolved by increasing the number of executors.
2. The jobs fail when there is not enough memory. An *OutOfMemory* error has been found in the executors' logs. It can be resolved by adding RAM.

The goal is to increase the number of rules. Another job is launched with more resources and more detection rules. The resources for this job are presented in the last line of the table 7.1.6.

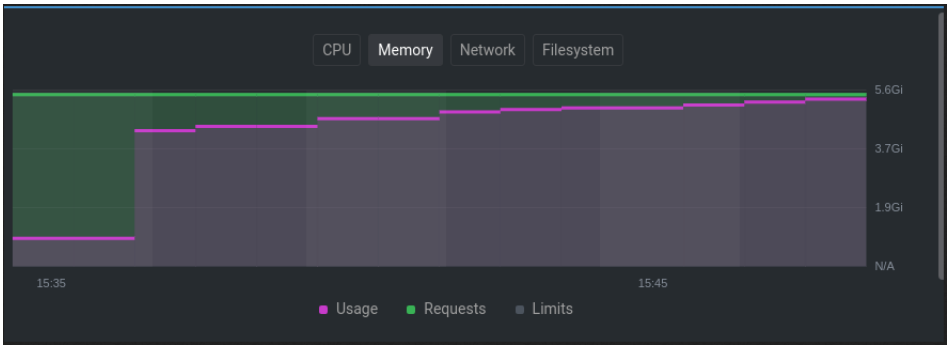

Figure 7.1.3: Memory use for the test with 20 rules launched at the same time in the same job.

When the tests runs, an error appears. The memory use is too high. Some RAM is added and but the error still appears. The figure 7.1.3 shows the memory use of the job. The usage is quickly too close of the limit of RAM.

7.1.2 Correlation Operators

One of the goals of the new correlation system is to manage to group the events by attributes. This part is a success because Apache Spark Structured Streaming can count the number of events per windows of time and per attributes.

7.1.3 Results

Correlation Time t

The tests show that the parameter correlation time t has not a big influence on the performance of the system. The correlation needed is on a short period of time (seconds or hours but not weeks). The RAM of the system has to be adapted because the system needs to store the amount of data received during t but if the RAM is well adjusted the correlation time has no influence on the efficiency of the system. The alert time is defined as the time between the last events that supposed to raise an alert and the triggering of the alert. The figure 7.1.1 shows the alert time as a function of the correlation time. It shows no big influence of t on the performance.

Number of Events E

It has been decided that the correlation system has to handle 600 eps. The entire platform receives 20,000 eps but the actual correlation engine receives only 100 eps

because of a tunnel system which filters events that are not suspicious. Indeed if an event is not part of a potential attack it is not send to the following micro service, so the number of events in the workflow diminishes between the input and the output. The correlation engine receives only the events which take part in an attacks. The figure 7.1.2 shows that for a fix number of resources the alert time increases with the number of events but this increase is logarithmic and the number of events sent in the system during the tests is far above the number of events expected in the real system. The parameter E has an influence on the performance but it can be handle with more resources, a scale effect intervenes and avoids the necessity of too many resources.

7.1.4 Multiple Rules

The main objective of this experimentation with Apache Spark is the evaluate the feasibility of an correlation engine based this technologies. The goals of this correlation system is to analyze and detect correlation on hundreds of eps based on hundreds of correlation rules. The previous section shows the parameter E has not a real influence unlike the number of rules launched.

The results of the tests show that it is possible to launch multiple jobs at the same time. If one rule corresponding to one job, the correlation system is working well and the attacks are detected. The problem comes from the number of resources. This approach obliges to ask more resources each time a new rule is launched which is not scalable.

The search of a scale effect leads to a seconds tests with multiple rules running in the same job. The tests with this technique are not conclusive. The multiple rules on the same resources created a latency which increases the alert time and the rules failed more frequently because of an overuse of the RAM 7.1.3. As the rules are in the same job it is not possible to relaunch them individually when one is failing.

7.2 Elastic Search

This section presents the results of the work on Elastic Search.

Elastic Search is used in the platform to store the events of our customers. As Elastic Search is installed in the system it is much easier to change it to make it works for the

correlation.

7.2.1 Performance Evaluation

The performance evaluation of Elastic Search is realized on the cluster already existing in the workflow on the data of our customers. The queries are made one at the time. It generates uncertainties on the performance of the final system because the conditions are not the exactly the same but it gives us a good idea of how to improve the performance. Multiple queries are launched manually in the system.

The course of action of those queries is:

1. Filtering on a large period of time.
2. Filtering on one or multiple attributes.
3. Aggregate by time windows.
4. Aggregate on one or multiples attributes.
5. Filtering on the number of events in the group of aggregation (bucket selector).

To test the impact of each parameters and find a way to improve the performance multiple queries are launched on a fixed number of events (around a billion). The parameters that are changing during the tests are:

- Number of aggregations.
- Number of filters.
- Number of buckets selectors.

We measure with those tests the influence of those parameters the time the query made to return a result.

7.2.2 Results

The table 7.2.1 shows the results of the tests.

Multiple observations are made thanks to those tests:

- Filters at the beginning of the queries reduce the processing time.
- Avoid aggregation reduces the processing time.

Test #	number of aggregations	number of filters	number of bucket selectors	number of events after filtering	time of the queries
1	2	2	0	more than 10k	500ms
2	2	2	1	more than 10k	700ms
3	2	3	1	5480	300ms
4	3	2	0	more than 10k	6000ms

Table 7.2.1: Performance evaluation of the Elastic Search queries.

- Avoid bucket selector reduces the processing time.

With those observations the queries are adapted to optimize the processing time:

- To avoid the aggregation, the query is associated to an event and the *groupby* is realized by filtering on the attributes of the event and on the aggregation time of the rules.
- To avoid bucket selectors, the process of filtering on the number of events in the aggregation group is made by a python algorithm.

The listing 7.1 is an example of a correlation rule.

```
[event.id = 5145] REPEAT 10 GROUPBY username WITHIN 30 SECONDS
```

Listing 7.1: An example of a correlation rule

Before the optimizations, the query is launched periodically and the course of action of the rule in Elastic Search is:

1. Filter *event.id = 5145*.
2. Filter on the last hour.
3. Aggregate by time windows of 1m (marge error).
4. Aggregate by *username*.
5. Select the groups with more than 10 events with bucket selector.

6. An alert is raised by the python algorithm if the Elastic Search query return a result.

After the optimizations, the query is launched when an event taking part of the rule is received. Suppose that an event is received with *event.id = 5145* and *username = Joe*, the course of action of the rule becomes:

1. Filter on the last minute.
2. Filter *event.id = 5145*.
3. Filter *username = Joe*.
4. An alert is raised by the python algorithm if the Elastic Search query return more than 10 results for this query.

The results of the tests are conclusive. It is possible to query the Elastic Search cluster and it takes less than a second to have the results. The optimization allows reducing the processing time. However, those tests were not realized in real-life condition. The number of events implied on correlation rules is around 200 eps which could potentially lead to 200 queries on the Elastic Search cluster per seconds.

7.3 Comparison among the Models

The three correlation systems have different ways of functioning. The correlation system used before the beginning of this work is coded by the developers. It uses a python lib to store each event that could take part of a correlation rule. The number of events is too high and the system is not sized to support them. This weakness in the system makes the behaviour of detection rules unpredictable: when a rule is enabled, the analysts has to check the behaviour of the workflow to see if it poses a problem. Another drawback with this system is the impossibility to treat the events depending on their attributes. For example it is not possible to count the number of failed logins by username during a certain period of time. It is provoking false positive and decreasing the correlation capacities. All these reasons make necessary to find a different system.

The system developed with Apache Spark Structured Streaming solves the issue of treating the events by attributes and one of the major advantages of Apache Spark is the In Memory computation which makes the system independent of storage systems. The

Correlation system	**Actual correlation system**	**Apache Spark system**	**Elastic Search system**
Technology used	Python lib	Apache Spark Structured Streaming	Elastic Search
Type of correlation	Real-time correlation	Real-time correlation	Search based correlation
GroupBy Operator	No	Yes	Yes
Advantages	Develop in intern	In memory computation and streaming computation	Stable storage solution and optimization of the run time of the rules
Drawbacks	Unpredictable behavior of the rules	Too many resources needed to launch hundreds of correlation rules	Based on an Elastic Search cluster and no real-time treatment of the events

Table 7.3.1: Comparison among the Implemented Correlation Systems.

main drawback of the implementation with Apache Spark is the amount of resources needed to run thousands of queries at the same time. As it is real-time computation the rules are running all the time in parallel. The number of rules in the system cannot be limited because it would reduce the correlation capacities. This element renders the system inadequate for us.

The approach with Elastic Search eliminates the problem of the number of rules running in parallel continuously because it launches the rules only if it is necessary. The performance of this correlation system is still unknown because the development is not finished but the first results are encouraging.

Chapter 8

Conclusion and Future Work

The Apache Spark correlation system has to meet four requirements: efficiency, availability, scalability and security. In the tests, only the efficiency, the availability and the scalability have been evaluated. Efficiency and availability can be easily implemented by adding the necessary resources. With more resources, the system is faster and do not crash. The scalability proposed by Apache Spark is not sufficient for us. Each query needs a certain number of resources depending on the correlation time t, on the number of events E and the structure of the query (the number of filters or the number of aggregations). The ideal way to execute multiple queries is to run each query in separate jobs. The implementation works well and can correlate the way we need but the number of necessary resources explodes with the number of queries. It is a major problem because the platform needs to be able to propose hundreds of correlation rules and each rule corresponds to one query.

The tests show that there is no scale effect when the queries are run in the same job. It is possible to do it that way, but the resources are not really shared by the queries and each query needs to have the number of resources necessary for its execution. Sharing a job also induces high concurrency which slows down the execution of the queries. It is not recommended because the monitoring of the queries is more difficult and when one query fails it cannot be restarted without stopping all the others.

All of this means that there is no scale effect with Apache Spark Structured Streaming. To add a detection rule, it is necessary to add the number of resources the rule needs. It is a problem for us because the goal is to implement hundreds of correlation rules at the same time. The resources needed in that case are huge.

Spark Structured Streaming is not the correlation solution for the platform. It can correlate a lot of events per seconds but with few queries or huge numbers of resources. We need to execute hundreds of queries in parallel and we cannot afford to increase the number of resources each time a rule is added. Apache Spark with Structured Streaming is not adapted for this use case.

The approach with Elastic Search is still in the development phase and has not been deployed in production but the results are promising. Elastic Search allows realizing all the operators needed for the system, the response is fast and multiple queries can be launched at the same time.

In term of security, the Elastic Search cluster is deployed on the network and has no opening outside the internal network which increase the security of the system. The only person allowed to have access to the data are members of the development team and the access has to be made from the local network or via a VPN. In terms of availability, the Elastics Search cluster is maintained by anSite Reliability Engineering (SRE) team. As it is already used and deployed they have the necessary experience. Plus, the events from our client are already stored in this system so there are no huge structural changes. The only thing is that there will be more requests on the Elastic Search which could lead to an overuse of Elastic Search and a decrease of the performances. However the size of the cluster could be adapted.

The main objectives of this thesis are to improve the correlation capacities, to make it scalable and able to handle hundreds of correlation rules. Both approaches reach the first goal. They make possible the operator *groupby* in the detection rules which is an improvement compared to the previous correlation system. The objective of scalability was more ambitious. It is defined in this system by the possibility of treating thousands events per seconds with thousands of detection rules. The Apache Spark system does not meet this objective. It is possible with the system to treat thousands of events per seconds but the analysis with multiple detection rules at the same time is expensive and uses too many resources. However the objectives in terms of realization are met: with infinite resources the system is working. Concerning the Elastic Search system the implementation is in progress and the results on the performances in production are unknown. For now, the system is promising and brings many improvement compare to the actual correlation system. The tests realized with the actual Elastic Search cluster meet the requirements.

Much work could be done to continue this work, the implementation of the Elastic Search system has to be finished and pushed in production. Depending on the performances of the system multiple work could be done. The correlation capacities could be improved by the addition of new possibilities of data treatment. In the continuity of the work on Apache Spark, the same approach could be explored with Apache Flink [11]. Both technologies have almost the same functionality. Apache Spark is more famous and its community is bigger but Flink features a library for CEP to detect patterns in data streams that could be interesting for the project.

Finally, the detection capacities of SIEMs technologies are evolving fast and correlation avoids having too many false positives in the alerts raised. The main difficulties in those system is to manage to correlate events in real-time to be able to alert the customer of the threat.The systems developed in this project give some tracks for analyzing the logs and correlate them how ever the solutions has their drawbacks and a lot of work has to be made in this area.

Bibliography

[1] *Apache Cassandra's website*. Mar. 2021. URL: https://cassandra.apache.org/.

[2] *Apache Kafka's website*. Mar. 2021. URL: https://kafka.apache.org/.

[3] *Apache Spark's website*. Mar. 2021. URL: https://spark.apache.org/.

[4] Armbrust, Michael, Das, Tathagata, Torres, Joseph, Yavuz, Burak, Zhu, Shixiong, Xin, Reynold, Ghodsi, Ali, Stoica, Ion, and Zaharia, Matei. "Structured streaming: A declarative API for real-time applications in apache spark". In: *Proceedings of the 2018 International Conference on Management of Data*. Houston TX USA, June 2018, pp. 601–613.

[5] Carcillo, Fabrizio, Dal Pozzolo, Andrea, Le Borgne, Yann-Aël, Caelen, Olivier, Mazzer, Yannis, and Bontempi, Gianluca. "Scarff: a scalable framework for streaming credit card fraud detection with spark". In: *Information fusion* 41 (May 2018), pp. 182–194.

[6] Casas, Pedro, Soro, Francesca, Vanerio, Juan, Settanni, Giuseppe, and D'Alconzo, Alessandro. "Network security and anomaly detection with Big-DAMA, a big data analytics framework". In: *2017 IEEE 6th international conference on cloud networking (CloudNet)*. Prague, Czech Republic, Sept. 2017, pp. 1–7.

[7] *Elastic Search's website*. Mar. 2021. URL: https://www.elastic.co/fr/.

[8] *EQL Search*. Mar. 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.

[9] *Esper's website*. Mar. 2021. URL: https://www.espertech.com/esper/.

[10] Ficco, Massimo and Romano, Luigi. “A generic intrusion detection and diagnoser system based on complex event processing”. In: *2011 First International Conference on Data Compression, Communications and Processing*. IEEE. Palinuro, Italy, June 2011, pp. 275–284.

[11] *Flink’s website*. Mar. 2021. URL: https://flink.apache.org/.

[12] Hafsa, Mounir and Jemili, Farah. “Comparative study between big data analysis techniques in intrusion detection”. In: *Big Data and Cognitive Computing* 3.1–13 (Dec. 2018), p. 1.

[13] Husák, Martin, Čermák, Milan, Laštovička, Martin, and Vykopal, Jan. “Exchanging security events: Which and how many alerts can we aggregate?” In: *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Lisbon, Portugal, May 2017, pp. 604–607.

[14] Husák, Martin and Kašpar, Jaroslav. “AIDA framework: Real-time correlation and prediction of intrusion detection alerts”. In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. Canterbury CA United Kingdom, Aug. 2019, pp. 1–8.

[15] *Ionos website: Apache Kafka*. Mar. 2021. URL: https://www.ionos.fr/digitalguide/serveur/know-how/apache-kafka/.

[16] Ivanov, Todor and Taaffe, Jason. “Exploratory analysis of spark structured streaming”. In: *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*. Berlin, Germany, Apr. 2018, pp. 141–146.

[17] *Kafka Documentation*. Mar. 2021. URL: http://kafka.apache.org/documentation.html#security.

[18] *Kafka-Proxy Documentation*. Mar. 2021. URL: https://github.com/grepplabs/kafka-proxy/tree/master.

[19] Kotenko, Igor, Kuleshov, Artem, and Ushakov, Igor. “Aggregation of elastic stack instruments for collecting, storing and processing of security information and events”. In: *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. San Francisco, CA, USA, Aug. 2017, pp. 1–8.

[20] *Next generation SIEM*. Mar. 2021. URL: https://www.securonix.com/what-is-next-generation-siem/.

[21] Oasis. *STIX™ Version 2.0. Part 5: STIX Patterning*. Mar. 2021. URL: http://docs.oasis-open.org/cti/stix/v2.0/stix-v2.0-part5-stix-patterning.html.

[22] *OVHcloud's website*. Mar. 2021. URL: https://www.ovh.com/fr/.

[23] *Query DSL*. Mar. 2021. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html.

[24] Ramaki, Ali Ahmadian, Amini, Morteza, and Atani, Reza Ebrahimi. "RTECA: Real time episode correlation algorithm for multi-step attack scenarios detection". In: *computers & security* 49 (Mar. 2015), pp. 206–219.

[25] *Security Information and Event Management*. Mar. 2021. URL: https://fr.wikipedia.org/wiki/Security_information_management_system.

[26] *Sigma Correlations*. Mar. 2021. URL: https://onedrive.live.com/view.aspx?resid=3454E59DF98D7D65!7485&ithint=file%5C%2cdocx&authkey=!ADb97TgRX9Fr4xQ.

[27] *Sigma github*. Mar. 2021. URL: https://github.com/Neo23x0/sigma.

Appendix - Contents

Appendix A

Correlation Rules

A.1 Rule 1

This rule describes a spark application. The *schema* defines the JSON format of the data in the Kafka. It helps the code to read into the dataframe.

The rule written in STIX Patterning is in the listing A.1 :

```
([ipv4-addr:value='10.0.0.1'] FOLLOWEDBY ([ipv4-addr:value='10.0.0.3'] OR
   ([ipv4-addr:value='10.0.0.2'] FOLLOWEDBY [ipv4-addr:value='10.0.0.3']))
    FOLLOWEDBY [ipv4-addr:value='10.0.0.4']) WITHIN 60 SECONDS
```

Listing A.1: Rule 1 in STIX Patterning.

The listing A.2 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
   , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *

#SparkSession configuration
spark = SparkSession
    .builder
    .appName("tdr_1_test")
    .config("spark.sql.shuffle.partitions", 4)
    .config("spark.authenticate.secret","true")
    .getOrCreate()

schema = StructType()
```

```
.add("definition", StructType()
  .add("id",StringType())
  .add("type",StringType())
  .add("spec_version", StringType())
  .add("objects", ArrayType(StructType()
.add("created", StringType())
.add("created_by_ref", StringType())
.add("description", StringType())
.add("first_observed",TimestampType())
.add("id",StringType())
.add("identity_class", StringType())
.add("last_observed",TimestampType())
.add("modified", StringType())
.add("name", StringType())
.add("number_observed", LongType())
.add("objects", StructType()
  .add("0", StructType()
    .add("action_ref", StringType())
    .add("type", StringType())
    .add("value", StringType()))
  .add("1", StructType()
    .add("action_ref", StringType())
    .add("dst_byte_count", StringType())
    .add("dst_packets", StringType())
    .add("dst_port", StringType())
    .add("dst_ref", StringType())
    .add("end", StringType())
    .add("extensions", StructType()
.add("http-request-ext", StructType()
  .add("request_header", StructType()
    .add("User-Agent",  StringType()))
  .add("request_method", StringType())
  .add("request_value", StringType())
  .add("request_version", StringType())
  .add("x_request_referrer", StringType())
  .add("x_url_ref", StringType()))
.add("x-http-response", StructType()
  .add("x_response_bytes", LongType())
  .add("x_response_status_code", LongType()))
.add("x-log", StructType()
  .add("hostname", StringType())))
```

```
    .add("protocol", ArrayType(StringType()))
    .add("src_byte_count", StringType())
    .add("src_packets", StringType())
    .add("src_port", StringType())
    .add("src_ref", StringType())
    .add("start", StringType())
    .add("type", StringType())
    .add("x_latency_ms",StringType())))
.add("type", StringType())
.add("x_event_type", StringType())
.add("x_row_event", StringType())
.add("x_sic_entity_by_ref", StringType())
.add("x_sic_event_type", StringType())
.add("x_sic_ingestion_timestamp", StringType())
.add("x_sic_site_id", StringType())
.add("x_sic_tags", ArrayType(StructType()
  .add("object_id", StringType())
  .add("object_path", StringType())
  .add("tag_id", StringType())
  .add("value", StringType()))))))
  .add("obexs",ArrayType(StringType()))

#kafka configuration
df_event1_1 = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "kafka.test:9090")
    .option("subscribe", "sic.solved_events")
    .option("kafka.security.protocol", "SSL")
    .option("kafka.ssl.truststore.location", "client.truststore.jks")
    .option("kafka.ssl.truststore.password", "password")
    .option("kafka.ssl.keystore.location", "client.keystore.jks")
    .option("kafka.ssl.keystore.password", "password")
    .option("startingOffsets", "latest")
    .option("failOnDataLoss","false")
    .load()
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json(col("value"), schema).alias("df"),col("timestamp"))
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),
    col("df.definition").alias("definition"))
```

```
ruleaxbc = "([ipv4-addr:value='10.0.0.1'] FOLLOWEDBY ([ipv4-addr:value
    ='10.0.0.3'] OR ([ipv4-addr:value='10.0.0.2'] FOLLOWEDBY [ipv4-addr:
    value='10.0.0.3'])) FOLLOWEDBY [ipv4-addr:value='10.0.0.4']) WITHIN 60
     SECONDS"


aggregation1 = df_event1_1.withWatermark("timestamp", "2 minutes")
  .withColumn("realobexs", when(array_contains(col("obexs"),"31
    cccf724c5304567639fb3414c70e08cef0ab744b2986d7258d725dd47a4e08"), "
    obexsa")
     .when(array_contains(col("obexs"),"
    ecd30eb7cb53e487d9c24f767a79bb89fb4b1e8ec77a799bf208c7f693bc6d4c"),"
    obexsx")
     .when(array_contains(col("obexs"),"5
    b06a5cd1404e9d79973eee9b68c4106f35b4e46c6800dd7359ca2f84ce91d0b"),"
    obexsb")
     .when(array_contains(col("obexs"),"726
    ba4b41d3a84075b9de46a679648f307e9f35e6463e2595fec06fa3cc293a2"),"obexsc
    ")
     .otherwise("unknown"))
  .filter("realobexs == 'obexsa' OR realobexs == 'obexsx' OR realobexs == '
    obexsb' OR realobexs == 'obexsc'")
  .withColumn("amount", lit(1))
  .groupBy(window(df_event1_1.timestamp,"120 seconds", "60 seconds"),
    df_event1_1.id_community)
  .agg(expr("sum(if(realobexs = 'obexsa', amount, null))").alias("A"), expr
    ("sum(if(realobexs = 'obexsx', amount, null))").alias("X"),
  expr("sum(if(realobexs = 'obexsb', amount, null))").alias("B"), expr("sum
    (if(realobexs = 'obexsc', amount, null))").alias("C"),
     expr("first(if(realobexs = 'obexsa', definition, null),true)").alias("
    defA"),
     expr("first(if(realobexs = 'obexsa', timestamp, null),true)").alias("
    timeA"),
     expr("first(if(realobexs = 'obexsx', timestamp, null), true)").alias("
    timeX"),
     expr("first(if(realobexs = 'obexsb', timestamp,null),true)").alias("
    timeB"), expr("first(if(realobexs = 'obexsc', timestamp, null),true)")\
     .alias("timeC"))
  .filter("A > 0 AND B > 0 AND C > 0 AND timeA < timeB AND timeB < timeC")
```

```
queryaxbc = aggregation1.withColumn("rule", lit(ruleaxbc))
  .select(to_json(struct(col("defA"),col("A"), col("X"), col("B"), col("C")
   ,col("timeA"),col("timeX"),col("timeB"),col("timeC"), col("rule"),col("
   window"))).alias("value"))
  .writeStream
  .option("checkpointLocation", "/tmp/checkpoint/query1")
  .outputMode("update")
  .queryName("query1")
  .format("kafka")
  .option("kafka.bootstrap.servers", "kafka.test:9090")
  .option("topic", "cdu-test-correlation")
  .option("kafka.security.protocol", "SSL")
  .option("kafka.ssl.truststore.location", "client.truststore.jks")
  .option("kafka.ssl.truststore.password", "password")
  .option("kafka.ssl.keystore.location", "client.keystore.jks")
  .option("kafka.ssl.keystore.password", "password")
  .start()
```

Listing A.2: Rule 1 in Apache Spark.

A.2 Rule 2

The rule written in STIX Patterning is in the listing A.3 :

```
[ipv4-addr:value='192.168.192.168'] REPEAT 200 WITHIN 60 SECONDS
```

Listing A.3: Rule 2 in STIX Patterning.

The listing A.4 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
import pyspark.sql.functions as F

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_2_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
    .getOrCreate()
```

```
#kafka configuration
df_event = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(F.from_json(F.col("value"), schema).alias("df"),
    F.col("timestamp"))\
    .select(F.col("df.obexs").alias("obexs"),F.col("timestamp"),F.
   element_at(F.col("df.definition.objects.id"),2).alias("id_community"),
    F.col("df.definition").alias("definition"))


rule = "[ipv4-addr:value='192.168.192.168'] repeat 200 WITHIN 60 SECONDS"


query = df_event.withWatermark("timestamp", "120 seconds")\
  .filter(F.array_contains(F.col("obexs"),"
   e1385a743b5c5565ff1a676c9099e8ab88d2314c4cbe2747795818bb70665ee4"))\
  .groupBy(F.window(F.col("timestamp"), "120 seconds", "60 seconds"),F.col(
   "id_community"))\
  .agg(F.collect_list(F.col("definition")).alias("list_def"),F.collect_list
   (F.col("timestamp")).alias("time"))\
  .select(F.size("list_def").alias("count"),F.element_at("list_def",1).
   alias("def"),F.element_at("time",1).alias("time1"),
  F.element_at("time",F.size("list_def")).alias("time2"))\
  .filter("count > 199")\
  .withColumn("rule", F.lit(rule))\
  .select(F.to_json(F.struct(F.col("def"),F.col("count"),F.col("time1"),F.
   col("time2"),F.col("rule"))).alias("value"))\
```

```
  .writeStream\
  .option("checkpointLocation", "/tmp/query2/checkpoint")\
  .outputMode("update")\
  .queryName("Query2")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.4: Rule 2 in Apache Spark.

A.3 Rule 3

The rule written in STIX Patterning is in the listing A.5 :

```
([ipv4-addr:value='192.168.0.1'] FOLLOWEDBY [ipv4-addr:value='192.168.0.2']
    FOLLOWEDBY [ipv4-addr:value='192.168.0.3']) WITHIN 30 SECONDS
```

Listing A.5: Rule 3 in STIX Patterning.

The listing A.6 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *


#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_3_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
    .getOrCreate()


#kafka configuration
```

```
df_event1_1 = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json("value", schema).alias("df"),"timestamp")\
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),
    col("df.definition").alias("definition"))


rule = "([ipv4-addr:value='192.168.0.1'] FOLLOWEDBY [ipv4-addr:value
   ='192.168.0.2'] FOLLOWEDBY [ipv4-addr:value='192.168.0.3'])
WITHIN 30 SECONDS"


aggregation1 = df_event1_1.withWatermark("timestamp", "1 minutes")\
  .withColumn("realobexs", when(array_contains(col("obexs"),"8338
   a069e0626af51f840351047647f40f738b3972d21b70f76ff448671d9fb0"),
  "obexsa")\
    .when(array_contains(col("obexs"),"33
   dfb5cd2166c0eacf2cef2fd4c5e86b0ac99f7440bec2f746073cdf5be82669"),"
   obexsb")\
    .when(array_contains(col("obexs"),"
   e41cd6f0e023df350f42b7fd029e23181a5a89df4e532a659c6be06b5ed3b6dc"),"
   obexsc")\
    .otherwise("unknown"))\
  .filter("realobexs == 'obexsa' OR realobexs == 'obexsb' OR realobexs == '
   obexsc'")\
  .withColumn("amount", lit(1))\
  .groupBy(window(col("timestamp"),"60 seconds", "30 seconds"),col("
   id_community"))\
```

```
  .agg(expr("sum(if(realobexs = 'obexsa', amount, null))").alias("A"),
  expr("sum(if(realobexs = 'obexsb', amount, null))").alias("B"),
  expr("sum(if(realobexs = 'obexsc', amount, null))").alias("C"),\
    expr("first(if(realobexs = 'obexsa', definition, null),true)").alias("
   defA"),expr("first(if(realobexs = 'obexsa', timestamp,
    null),true)").alias("timeA"),expr("first(if(realobexs = 'obexsb',
   timestamp,null),true)").alias("timeB"),
    expr("first(if(realobexs = 'obexsc', timestamp, null),true)").alias("
   timeC"))\
  .filter("A > 0 AND B > 0 AND C > 0 AND timeA < timeB AND timeB < timeC")


queryaxbc = aggregation1.withColumn("rule", lit(rule))\
  .select(to_json(struct(col("defA"),col("A"),col("B"),col("C"),col("timeA"
   ), col("timeB"),col("timeC"),col("rule"),
  col("window"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "query3/checkpointlocation")\
  .outputMode("update")\
  .queryName("query3")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.6: Rule 3 in Apache Spark.

A.4 Rule 4

The rule written in STIX Patterning is in the listing A.7 :

```
([ipv4-addr:value='1.2.3.4'] FOLLOWEDBY [ipv4-addr:value='5.6.7.8']
   FOLLOWEDBY [ipv4-addr:value='2.4.6.8']) WITHIN 30 SECONDS
```

Listing A.7: Rule 4 in STIX Patterning.

The listing A.8 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_4_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
    .getOrCreate()


#kafka configuration
df_event = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json(col("value"), schema).alias("df"),
    col("timestamp"))\
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),
    col("df.definition").alias("definition"))


rule = "([ipv4-addr:value='1.2.3.4'] FOLLOWEDBY [ipv4-addr:value='5.6.7.8']
    FOLLOWEDBY [ipv4-addr:value='2.4.6.8'])
WITHIN 30 SECONDS"
```

```
aggregation1 = df_event.withWatermark("timestamp", "1 minutes")\
  .withColumn("realobexs", when(array_contains(col("obexs"),"1
   a8e3160fec94f0980f380f1f1bc95be6c39d3c08f2b966f7d6a8e2ff1b64513"),
  "obexsa")\
    .when(array_contains(col("obexs"),"679
   d2b368d25a0d50c0a57082517cfa9a4ce3eef13e505c68d065466e63cce95"),"obexsb
   ")\
    .when(array_contains(col("obexs"),"
   eeac514412453757fd4faab01f69d24bfc200e97e3da5824e89ad7ab11ea43a5"),"
   obexsc")\
    .otherwise("unknown"))\
  .filter("realobexs == 'obexsa' OR realobexs == 'obexsb' OR realobexs == '
   obexsc'")\
  .withColumn("amount", lit(1))\
  .groupBy(window(col("timestamp"),"60 seconds", "30 seconds"),col("
   id_community"))\
  .agg(expr("sum(if(realobexs = 'obexsa', amount, null))").alias("A"), expr
   ("sum(if(realobexs = 'obexsb', amount, null))").alias("B"),
  expr("sum(if(realobexs = 'obexsc', amount, null))").alias("C"),\
    expr("first(if(realobexs = 'obexsa', definition, null),true)").alias("
   defA"),
    expr("first(if(realobexs = 'obexsa', timestamp, null),true)").alias("
   timeA"),
    expr("first(if(realobexs = 'obexsb', timestamp,null),true)").alias("
   timeB"),
    expr("first(if(realobexs = 'obexsc', timestamp, null),true)").alias("
   timeC"))\
  .filter("A > 0 AND B > 0 AND C > 0 AND timeA < timeB AND timeB < timeC")

queryaxbc = aggregation1.withColumn("rule", lit(rule))\
  .select(to_json(struct(col("defA"),col("A"), col("B"), col("C"),col("
   timeA"),col("timeB"),col("timeC"),
  col("rule"),col("window"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "query4/checkpointlocation")\
  .outputMode("update")\
  .queryName("query4")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
```

```
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.8: Rule 4 in Apache Spark.

A.5 Rule 5

The rule written in STIX Patterning is in the listing A.9 :

```
(([ipv4-addr:value='10.10.10.10'] REPEAT 10 TIMES) FOLLOWEDBY [ipv4-addr:
    value='11.11.11.11']) WITHIN 300 SECONDS
```

Listing A.9: Rule 5 in STIX Patterning.

The listing A.10 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_5_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
    .getOrCreate()


#kafka configuration
df_event = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
```

```
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json(col("value"), schema).alias("df"),col("timestamp"))\
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),
    col("df.definition").alias("definition"))


rule = "(([ipv4-addr:value='10.10.10.10'] REPEAT 10 TIMES) FOLLOWEDBY [ipv4
   -addr:value='11.11.11.11']) WITHIN 300 SECONDS"

query1 = df_event.withWatermark("timestamp", "600 seconds")\
  .withColumn("realobexs", when(array_contains(col("obexs"),"
   e4f1a13b05c84ddf8daca9747a1dea24dcb06512c9e814ec24415b56c8c5d5c7"),
  "obexs200")\
  .when(array_contains(col("obexs"),"1
   f7f206ac7d9e3160b2409260c48a92da3a5a6794fcab9ca2fc9f29cd04688db"),"
   obexs1")\
  .otherwise("unknown"))\
  .filter("realobexs == 'obexs200' OR realobexs == 'obexs1'")\
  .withColumn("amount", lit(1))\
  .groupBy(window(col("timestamp"), "600 seconds", "300 seconds"),col("
   id_community"))\
  .agg(expr("sum(if(realobexs = 'obexs200', amount, null))").alias("A"),
  expr("sum(if(realobexs = 'obexs1', amount, null))").alias("B"),
  expr("last(if(realobexs = 'obexs200', timestamp, null),true)").alias("
   timeA"),
  expr("first(if(realobexs = 'obexs1', timestamp, null),true)").alias("
   timeB"),
  expr("first(if(realobexs = 'obexs1', definition, null),true)").alias("
   defB"))\
  .filter("A > 9 AND B > 0 AND timeA < timeB")\
  .withColumn("rule", lit(rule))\
  .select(to_json(struct(col("defB"), col("timeA"),col("timeB"),col("A"),
   col("B"),col("id_community"),col("rule"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "/tmp/query5/checkpoint")\
  .outputMode("update")\
```

```
  .queryName("Query5")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.10: Rule 5 in Apache Spark.

A.6 Rule 6

The rule written in STIX Patterning is in the listing A.11 :

```
([ipv4-addr:value='1.1.1.1'] FOLLOWEDBY [ipv4-addr:value!='1.1.1.1'])
    GROUPBY username WITHIN 300 SECONDS
```

Listing A.11: Rule 6 in STIX Patterning.

The listing A.12 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_6_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
    .getOrCreate()


#kafka configuration
df_event = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
```

```
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json(col("value"), schema).alias("df"),
    col("timestamp"))\
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),\
    col("df.definition").alias("definition"),element_at(col("df.definition.
   objects.name"),2).alias("username"))


rule = "([ipv4-addr:value='1.1.1.1'] FOLLOWEDBY [ipv4-addr:value
   !='1.1.1.1']) GROUPBY username WITHIN 300 SECONDS"


aggregation1 = df_event.withWatermark("timestamp", "600 seconds")\
  .withColumn("realobexs", when(array_contains(col("obexs"),"597
   be2e0d7f89ab12c1719dd8c8b4fcd97464354eb698e6157b02093efd1d4ff"),
  "obexsa")\
    .when(array_contains(col("obexs"),"3605
   c52ba3325c98c116ba4945721e41499bd0d1b3c4824c185f6edabc3b195c"),"obexsb"
   )\
    .otherwise("unknown"))\
  .filter("realobexs == 'obexsa' OR realobexs == 'obexsb'")\
  .withColumn("amount", lit(1))\
  .groupBy(window(col("timestamp"),"600 seconds", "300 seconds"),col("
   id_community"),col("username"))\
  .agg(expr("sum(if(realobexs = 'obexsa', amount, null))").alias("A"),
  expr("sum(if(realobexs = 'obexsb', amount, null))").alias("B"),\
    expr("first(if(realobexs = 'obexsa', definition, null),true)").alias("
   defA"),
    expr("first(if(realobexs = 'obexsa', timestamp, null),true)").alias("
   timeA"),
    expr("first(if(realobexs = 'obexsb', timestamp,null),true)").alias("
```

```
    timeB"))\
  .filter("A > 0 AND B > 0 AND timeA < timeB")

queryaxbc = aggregation1.withColumn("rule", lit(rule))\
  .select(to_json(struct(col("defA"),col("A"), col("B"),col("timeA"),col("
    timeB"), col("rule"),col("window"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "query6/checkpointlocation")\
  .outputMode("update")\
  .queryName("query6")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.12: Rule 6 in Apache Spark.

A.7 Rule 7

The rule written in STIX Patterning is in the listing A.13 :

```
[ipv4-addr:value='10.10.0.1' OR ipv4-addr:value='10.10.0.2'] REPEAT 5
    WITHIN 1 HOUR
```

Listing A.13: Rule 7 in STIX Patterning.

The listing A.14 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
    , StructField, ArrayType, TimestampType
import pyspark.sql.functions as F

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_7_test")\
  .config("spark.sql.shuffle.partitions", 4)\
```

```
  .config("spark.authenticate.secret","true")\
    .getOrCreate()

#kafka configuration
df_event = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(F.from_json(F.col("value"), schema).alias("df"),
    F.col("timestamp"))\
    .select(F.col("df.obexs").alias("obexs"),F.col("timestamp"),F.
   element_at(F.col("df.definition.objects.id"),2).alias("id_community"),
    F.col("df.definition").alias("definition"))


rule = "[ipv4-addr:value='10.10.0.1' OR ipv4-addr:value='10.10.0.2'] REPEAT
    5 WITHIN 1 HOUR"


query = df_event.withWatermark("timestamp", "2 hours")\
  .filter(F.array_contains(F.col("obexs"),"6
   a381d5a7f1e36d3f36deddf921430c1441fcb19b0c38e80be0c982aa5edd96a"))\
  .groupBy(F.window(F.col("timestamp"), "2 hours", "60 minutes"),F.col("
   id_community"))\
  .agg(F.collect_list(F.col("definition")).alias("list_def"),F.collect_list
   (F.col("timestamp")).alias("time"))\
  .select(F.size("list_def").alias("count"),F.element_at("list_def",1).
   alias("def"),F.element_at("time",1).alias("time1"),
  F.element_at("time",F.size("list_def")).alias("time2"))\
  .filter("count > 4")\
  .withColumn("rule", F.lit(rule))\
```

```
  .select(F.to_json(F.struct(F.col("def"),F.col("count"),F.col("time1"),F.
   col("time2"),F.col("rule"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "/tmp/query7/checkpoint")\
  .outputMode("update")\
  .queryName("Query7")\
  .format("kafka")\
  .option("kafka.bootstrap.servers", "kafka.test:9090")\
  .option("topic", "cdu-test-correlation")\
  .option("kafka.security.protocol", "SSL")\
  .option("kafka.ssl.truststore.location", "client.truststore.jks")\
  .option("kafka.ssl.truststore.password", "password")\
  .option("kafka.ssl.keystore.location", "client.keystore.jks")\
  .option("kafka.ssl.keystore.password", "password")\
  .start()
```

Listing A.14: Rule 7 in Apache Spark.

A.8 Rule 8

The rule written in STIX Patterning is in the listing A.15 :

```
([ipv4-addr:value='1.0.0.0'] AND [ipv4-addr:value='2.0.0.0'] AND [ipv4-addr
   :value='3.0.0.0'] AND [ipv4-addr:value='4.0.0.0'] AND [ipv4-addr:value
   ='5.0.0.0'] AND [ipv4-addr:value='6.0.0.0'] AND  [ipv4-addr:value
   ='7.0.0.0'] AND [ipv4-addr:value='8.0.0.0']) GROUPBY username WITHIN 30
    SECONDS
```

Listing A.15: Rule 8 in STIX Patterning.

The listing A.16 shows the spark application for this rule.

```
from pyspark.sql import SparkSession
from pyspark.sql.types import LongType, StructType, StringType, IntegerType
   , StructField, ArrayType, TimestampType
from pyspark.sql.functions import *

#SparkSession configuration
spark = SparkSession \
    .builder \
    .appName("tdr_8_test")\
  .config("spark.sql.shuffle.partitions", 4)\
  .config("spark.authenticate.secret","true")\
```

```
    .getOrCreate()

#kafka configuration
df_event1_1 = spark\
    .readStream\
    .format("kafka")\
    .option("kafka.bootstrap.servers", "kafka.test:9090")\
    .option("subscribe", "sic.solved_events")\
    .option("kafka.security.protocol", "SSL")\
    .option("kafka.ssl.truststore.location", "client.truststore.jks")\
    .option("kafka.ssl.truststore.password", "password")\
    .option("kafka.ssl.keystore.location", "client.keystore.jks")\
    .option("kafka.ssl.keystore.password", "password")\
    .option("startingOffsets", "latest")\
    .option("failOnDataLoss","false")\
    .load()\
    .selectExpr("CAST(value AS STRING)","CAST(timestamp AS TIMESTAMP)").
   select(from_json(col("value"), schema).alias("df"),col("timestamp"))\
    .select(col("df.obexs").alias("obexs"),col("timestamp"),element_at(col(
   "df.definition.objects.id"),2).alias("id_community"),
    col("df.definition").alias("definition"),element_at(col("df.definition.
   objects.name"),2).alias("username"))


rule = "([ipv4-addr:value='1.0.0.0'] AND [ipv4-addr:value='2.0.0.0'] AND [
   ipv4-addr:value='3.0.0.0'] AND [ipv4-addr:value='4.0.0.0'] AND [ipv4-
   addr:value='5.0.0.0'] AND [ipv4-addr:value='6.0.0.0'] AND [ipv4-addr:
   value='7.0.0.0']AND [ipv4-addr:value='8.0.0.0']) GROUPBY username
   WITHIN 30 SECONDS"


aggregation1 = df_event1_1.withWatermark("timestamp", "1 minutes")\
  .withColumn("realobexs", when(array_contains(col("obexs"),"9
   c9e8719aee33312f75d84e9f7883c71c92f219a1bf39a3e17bd3f028b429cc5"),
  "obexsa")\
    .when(array_contains(col("obexs"),"0651
   d3d05feaee22e0944fd3c790684e91612660949322c91d997f01b0fa92b9"),"obexsb"
   )\
    .when(array_contains(col("obexs"),"19
   d1a43653010b20838ca3f7c696d4f6ad493f7119f259b048575a6d5f225517"),"
   obexsc")\
```

```
                        .when(array_contains(col("obexs"),"94380
    a0426952c247b6de0c31af01d10e0a84d6168140f74932282f89167557d"),"obexsd")
    \
                        .when(array_contains(col("obexs"),"0917930461
    fec508ae05e7540a44f6db3a537e9f212088742bb1e1994dad181b"),"obexse")\
     .when(array_contains(col("obexs"),"
    da9d7e21f3de6975fc75eaeea659fdfda61a38d0634192177ce7bcdb4121de76"),"
    obexsf")\
                        .when(array_contains(col("obexs"),"
    f1b7fcb0c8d78889279c4b8008a647fb75d19449ab9a4b1de7835319e05e6f33"),"
    obexsg")\
                        .when(array_contains(col("obexs"),"25
    a0008fc25a90cc2030d9578a19a9b290abed191f86946d2baf4c3fecf8f5c0"),"
    obexsh")\
     .otherwise("unknown"))\
  .filter("realobexs == 'obexsa' OR realobexs == 'obexsb' OR realobexs == '
    obexsc' OR realobexs == 'obexsd' OR realobexs == 'obexse' OR
  realobexs == 'obexsf' OR realobexs == 'obexsg' OR realobexs == 'obexsh'")
    \
  .withColumn("amount", lit(1))\
  .groupBy(window(col("timestamp"),"60 seconds", "30 seconds"),col("
    id_community"),col("username"))\
  .agg(expr("sum(if(realobexs = 'obexsa', amount, null))").alias("A"), expr
    ("sum(if(realobexs = 'obexsb', amount, null))").alias("B"),
  expr("sum(if(realobexs = 'obexsc', amount, null))").alias("C"),\
     expr("sum(if(realobexs = 'obexsd', amount, null))").alias("D"), expr("
    sum(if(realobexs = 'obexse', amount, null))").alias("E"),
     expr("sum(if(realobexs = 'obexsf', amount, null))").alias("F"),\
     expr("sum(if(realobexs = 'obexsg', amount, null))").alias("G"), expr("
    sum(if(realobexs = 'obexsh', amount, null))").alias("H"),\
     expr("first(if(realobexs = 'obexsa', definition, null),true)").alias("
    defA"))\
  .filter("A > 0 AND B > 0 AND C > 0 AND D > 0 AND E > 0 AND F > 0 AND G >
    0 AND H > 0")


queryaxbc = aggregation1.withColumn("rule", lit(rule))\
  .select(to_json(struct(col("defA"),col("A"), col("B"), col("C"), col("D")
    ,col("E"), col("F"), col("G"), col("H"),
  col("rule"),col("window"))).alias("value"))\
  .writeStream\
  .option("checkpointLocation", "query8/checkpointlocation")\
```

```
.outputMode("update")\
.queryName("query8")\
.format("kafka")\
.option("kafka.bootstrap.servers", "kafka.test:9090")\
.option("topic", "cdu-test-correlation")\
.option("kafka.security.protocol", "SSL")\
.option("kafka.ssl.truststore.location", "client.truststore.jks")\
.option("kafka.ssl.truststore.password", "password")\
.option("kafka.ssl.keystore.location", "client.keystore.jks")\
.option("kafka.ssl.keystore.password", "password")\
.start()
```

Listing A.16: Rule 8 in Apache Spark.