Cloud-Based Protocol IDS using Apache Kafka and Spark Streaming

Leon Wirz
Sirindhorn International Institute of Technology, Thammasat Univerity, Pathum Thani, Thailand
leon.wir@dome.tu.ac.th

Rinrada Tanthanathewin
Sirindhorn International Institute of Technology, Thammasat Univerity, Pathum Thani, Thailand
rinrada.tanth.gal@gmail.com

Asipan Ketphet
Sirindhorn International Institute of Technology, Thammasat Univerity, Pathum Thani, Thailand
asipanketphet@gmail.com

***Abstract*—In the current era of technological advancements web applications are a major medium in the internet world. Many of these mentioned web applications adopt the usage of a dedicated back-end server where most of the technical processes happen. Typically the front-end sends requests to the back-end which can be done in multiple ways including, GraphQL, gRPC, REST, etc. In this particular paper, the focus is on REST as it is still considered a norm in most web applications. In REST, HTTP requests by the front-end which are mapped to GET, POST, PUT, and DELETE have been proven to be prone to attacks and exploits that can inflict extensive amounts of damage to a system. Therefore, this paper's main point will be the detection of such attacks and exploits, mainly being Automated Brute Forcing on web-based login, HTTP flood attacks, a), Cross-Site Scripting (XSS). Both Apache Kafka and Spark streaming are used as main pillars in the architecture for processing the user inputs in REST HTTP header fields, which are the main vulnerabilities that are being exploited in such attacks. As HTTP requests from the front end of a web application come in massive volumes it is crucial to process these HTTP requests in an efficient and resourceful way. Spark streaming allows both of these critical processing characteristics by using batch processing and MapReduce. Apache Kafka allows back-end engineers to send real-time request logs to a Kafka broker and furthermore bridges the streamed data to Spark streaming using the Spark Streaming & Kafka Integration library. In this paper algorithms that were developed in Scala are used to determine if an HTTP request contains any patterns that resemble any of the 4 mentioned attacks and both send an alert to the system security engineer and log the event in a Google Cloud Storage Bucket, as the developed IDS is implemented in Google Cloud Platform.**

Keywords—Intrusion Detection, Apache Kafka, Spark Streaming, SQL injection, Cross-site scripting, Brute Force Login, HTTP flood attacks

I. INTRODUCTION

An Intrusion detection system is a security system which main functionality is the monitoring and analysis of system events for the purpose of finding and providing real-time or near real-time alerts and warnings of attempts of unauthorized access to system resources. Protocol-based intrusion detection systems are specialized IDSs that are commonly installed on web servers and used to exclusively monitor and analyze HTTP or HTTPS requests in a stream-like format. There are three types of intrusion detections methods, which are as follows:

i) Signature/Heuristic-based detection, which is a simple and efficient method of detection that involves the matching against large collections of rules or known patterns of malicious data. Nevertheless, the ability of an IDS that uses this method to correctly detect an intrusion solely depends on the available collection of signatures. Hence, the system cannot detect unknown intrusions in which patterns are not contained in the collection of signatures.

ii) Anomaly-based detection, which involves actual observations is more dynamic and more intelligent compared to the latter. Using either statistical methods or Machine-learning are used to analyze events and categorize them into either being a legitimate request or an intrusion attempt. Statistical methodologies are applied to observe the behavior of events using univariate, multivariate, or time-series models to observe measurable metrics. The Machine-learning approach uses Classification ML models, which work by intaking events and activities and classifying them into being either intrusion attempts or not, as has been done in [3]. Additionally, more advanced ML models can classify events into more than two classes as in [11], where Siva Reddy and Saravanan have developed multiple Classification Models with more than 2 variations of outputs using one-hot encoding. Regardless of the mentioned advantages, compared to Signature/Heuristic-based detection, Anomaly-based detection's performance is not as favorable and has some effects on less powerful machines running real-time IDS.

iii) Distributed/Hybrid-based detection, which performs using a combination of Signature/Heuristic-based detection and Anomaly-based detection. This detection method has a more complex structure, but with it comes a better identification rate and response time.

A good intrusion detection system including protocol intrusion detection systems should give results in a real-time manner, must not take up a lot of resources, have a low false positive and false negative rate, have high to permanent uptime, and cover a wide area of intrusion methods that it can detect. But for protocol intrusion detection systems, it is difficult to achieve the characteristic of not using up a lot of resources because medium-sized to mainstream-sized web applications have a massive amount of HTTP requests that are sent at all times. Applying any kind of algorithm efficiently has always been a challenge, which this paper will be tackling. Furthermore, the problem of having permanent uptime on an on-premise IDS is not guaranteed, as it is an apparent problem that comes with every system that is on-premise due to power or internet outage, natural disasters, etc.

This paper's IDS is a cloud-based protocol IDS, which means that it is a protocol IDS that runs on a cloud environment, specifically Google Cloud Platform (GCP) [21]. This solves the problem of the lack of guarantee when talking about the uptime which should be high enough to a certain degree. Google Cloud Platform's service level agreement (SLA) dictates that the services that the system uses have a corresponding uptime being:

i. Compute Engine single instance SLA Monthly Uptime percentage of $\geq 99.5\%$

ii. Cloud Storage in regional location SLA Monthly Uptime percentage of $\geq 99.9\%$

iii. Dataproc SLA Monthly Uptime percentage of $\geq 99.5\%$

This allows the system to run almost permanently as the lowest SLA is 99.5% per month. The problem of huge amounts of streaming data, which needs to be processed in an efficient way, which comes with being a Protocol IDS is also being approached with the combination of using Apache Kafka [20] and Spark Streaming [19]. Kafka acts as the receiving end of a bridge that allows all the streaming data to pass through via four separate topics in a broker, being GET, POST, PUT, and DELETE, which correspond to their actual HTTP method name. These four topics, which are streaming data are then sent to separate Spark Streaming jobs that each detect their own Intrusions namely:

i. Automated Brute Forcing on web-based login

ii. HTTP flood attacks

iii. SQL Injections (SQLi)

iv. Cross-SiteSite Scripting (XSS)

Spark Streaming is a framework that works on Spark; therefore, it utilizes parallel computation and synchronization of multiple machines on one or more clusters. Dividing Streaming data into batches by time frames allows the usage of batch processing, furthermore the creation of a specialized Data structure called Resilient Distributed Dataset (RDD). Algorithms created in Spark typically run MapReduce Jobs which have been proven to give tremendous amounts of resource utilization and efficiency, which can be backed by Tun's, Nyaung's, and Phyu's experimentation results in [12], which show that 500,000 records as one batch can be processed in around 0.5 to 1.2 seconds using medium computation power, being an Intel Core i5 processor and 8GB memory.

Automated Brute Forcing [1] on web-based login is the act of using a program or script to forcefully send login requests, typically GET requests in HTTP to a web server. The frequency of requests sent varies from program to program and machine sent from. The Automated Brute Force Login detection algorithm alerts and logs requests if a certain request limit threshold is breached from a single IP address attempting to send requests at a high frequency. HTTP flood attacks are similarly detected using a different algorithm. Since HTTP flood attacks originate from all four HTTP request methods, being GET, POST, PUT, and DELETE, all requests are being processed in the HTTP flood attack detection algorithm. The frequency counting part also remains the same with the exception of the request frequency limit which is dependent on the implementation of IDS itself. As for SQL injections and Cross-Site Scripting detection, both have algorithms that function closely. A list of sub-strings for each attack is constructed which are extracted from datasets [16] [17], which are most likely to occur in an attack attempt.

The main objective of this paper is to show the capabilities of Apache Kafka and Spark streaming when used as a Protocol IDS to detect the aforementioned attacks while describing the architecture and methods used in detail with their corresponding results. Moreover, this paper's architecture can be used as a foundation for more specialized implementation in the Future.

This template, modified in MS Word 2007 and saved as a "Word 97-2003 Document" for the PC, provides authors with most of the formatting specifications needed for preparing electronic versions of their papers. All standard paper components have been specified for three reasons: (1) ease of use when formatting individual papers, (2) automatic compliance to electronic requirements that facilitate the concurrent or later production of electronic products, and (3) conformity of style throughout a conference proceedings. Margins, column widths, line spacing, and type styles are built-in; examples of the type styles are provided throughout this document and are identified in italic type, within parentheses, following the example. Some components, such as multi-leveled equations, graphics, and tables are not prescribed, although the various table text styles are provided. The formatter will need to create these components, incorporating the applicable criteria that follow.

A. *Selecting a Template (Heading 2)*

First, confirm that you have the correct template for your paper size. This template has been tailored for output on the A4 paper size. If you are using US letter-sized paper, please close this file and download the Microsoft Word, Letter file.

B. *Maintaining the Integrity of the Specifications*

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

II. Prepare Your Paper Before Styling

Before you begin to format your paper, first write and save the content as a separate text file. Complete all content and organizational editing before formatting. Please note sections A-D below for more information on proofreading, spelling and grammar.

Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

A. *Abbreviations and Acronyms*

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. *Units*

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as "3.5-inch disk drive".

- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.

- Do not mix complete spellings and abbreviations of units: "Wb/m2" or "webers per square meter", not "webers/m2". Spell out units when they appear in text: ". . . a few henries", not ". . . a few H".
- Use a zero before decimal points: "0.25", not ".25". Use "cm3", not "cc". (*bullet list*)

C. *Equations*

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled.

Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in:

$$a + b = \gamma \tag{1}$$

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use "(1)", not "Eq. (1)" or "equation (1)", except at the beginning of a sentence: "Equation (1) is . . ."

D. *Some Common Mistakes*

- The word "data" is plural, not singular.
- The subscript for the permeability of vacuum μ_0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter "o".
- In American English, commas, semicolons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an "inset", not an "insert". The word alternatively is preferred to the word "alternately" (unless you really mean something that alternates).
- Do not use the word "essentially" to mean "approximately" or "effectively".
- In your paper title, if the words "that uses" can accurately replace the word "using", capitalize the "u"; if not, keep using lower-cased.

Identify applicable funding agency here. If none, delete this text box.

- Be aware of the different meanings of the homophones "affect" and "effect", "complement" and "compliment", "discreet" and "discrete", "principal" and "principle".
- Do not confuse "imply" and "infer".
- The prefix "non" is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the "et" in the Latin abbreviation "et al.".
- The abbreviation "i.e." means "that is", and the abbreviation "e.g." means "for example".

An excellent style manual for science writers is [7].

III. Methodology

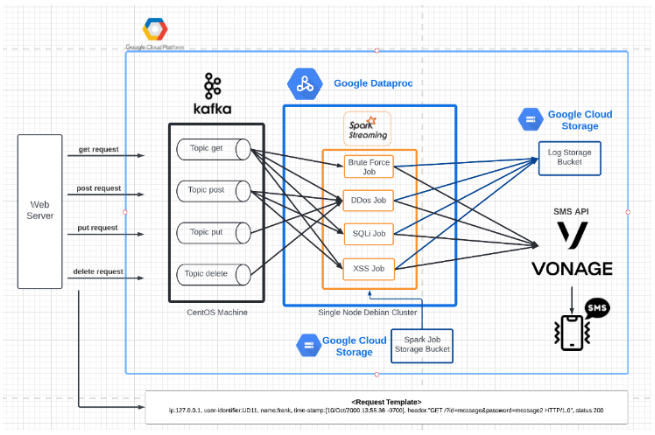

Fig.sd System diagram

Figure sd illustrates how the system works. Our system works entirely on the Google Cloud Platform, starting from query requests directly from the web server by Apache Kafka. Figure aa shows a request template that will be queried by Apache Kafka.

```
ip:127.0.0.1, user-identifier:UD11,
name:frank, time-stamp:[10/Oct/2000:13:55:36 -0700],
header:"GET /?id=message&password=message2
HTTP/1.0", status:200
```

Figure aa : Request Template

Apache Kafka then splits the request into four topics, GET, POST, PUT, and DELETE, depending on the header type of each request. The request string is then sent to Spark Streaming running on the Google Dataproc cluster. Google Dataproc allows users to create different cluster types and versions. This project uses a single node master on Debian 10, Hadoop 3.2, and Spark 3.1. A single node master provides a single node that acts as both a master and a worker.

Internal Spark Streaming is divided into four jobs: BruteForce job, DDOS job, SQL Injection(SQLi) job, and Cross-Site Scripting(XSS) job, which will be operated on the Debian cluster. The output of each Spark Streaming job will be stored as a log file in the bucket on Google Cloud Storage. In addition, we use the SMS API Vonage for SMS notifications.

1) BruteForce Algorithm

With a BruteForce algorithm, the system is attacked with the GET command, so it filters only the GET Kafka topic. Use the flatmap command to split the request stream into parts. Figure 1a shows the request string split by Flatmap.

```
flatmap(_.value().split(“, ”))
```

```
(ip:127.0.0.1)
(user-identifier:UD11)
(name:frank)
(time-stamp:[10/Oct/2000:13:55:36 -0700])
(header:"GET /?id=message&password=message2
HTTP/1.0")
(status:200)
```

Figure 1a : Request string after flatmap.

The algorithm keeps track of the number of IP addresses attempted in a particular time period. If the specified threshold is exceeded, it is considered a brute force attack and the algorithm saves the IP address and the number of attempts in a log file. Figure 1b shows examples of malicious IP addresses and the number of attempts for both BruteForce and DDOS attacks.

```
(ip:127.0.0.1, 15)
(ip:199.12.26.22, 16)
(ip:149.244.168.229, 22)
(ip:42.40.79.231, 65)
(ip:219.57.228.72, 101)
(ip:3.111.203.205, 11)
(ip:29.13.130.4, 69)
(ip:39.119.21.208, 42)
```

Figure 1b : Examples of BruteForce and DDOS attack’s saved IP addresses and the number of attempts.

2) DDos Algorithm

For the DDOS algorithm, the system is attacked with the GET, POST, PUT, and DELETE commands, so it uses every Kafka topic. Similar to the BruteForce algorithm, first, split the request stream into parts. Second, filter for malicious IP addresses that has a number of attempts that exceeds the limit threshold. Finally, save all malicious attempts into a log file.

3) SQLi Algorithm

With the SLI Injection algorithm, the system is attacked with the GET and POST commands, so it filters only those two Kafka topics. First, we have to rearrange the request stream into a form of request string that consists of IP address and header parts by using flatmap and map methods. Figure 3a shows examples of request strings that had rearranged by flatmap and map.

```
(ip:127.0.0.1, header:"GET /?id=admin" or
"1"="1&password=message2 HTTP/1.0")
(ip:127.0.0.1, header:"GET
/?id=message1&password=message2 HTTP/1.0")
(ip:29.13.130.4, header:"POST /?id=" or 1=1 –
&password=message2 HTTP/1.0")
(ip:149.244.168.229, header:"POST /?id= or 0=0 --
&password=message2 HTTP/1.0")
(ip:127.0.0.1, header:"GET
/?id=message5&password=message2 HTTP/1.0")
(ip:127.0.0.1, header:"GET
/?id=message6&password=message2 HTTP/1.0")
```

Figure 3a : Rearranged request string by flatmap and map methods.

The algorithm will filter the request strings that match SQLi request pattern and save it into a log file. Figure 3b shows examples of request string that match SQLi request pattern.

Figure 3b : Examples of request string match with SQLi

```
(ip:127.0.0.1, header:"GET /?id=admin" or
"1"="1&password=message2 HTTP/1.0", …. )
(ip:3.111.203.205, header:"GET /?id=' and
substring(password/text(),1,1)='7&password=message2
HTTP/1.0")
(ip:29.13.130.4, header:"POST /?id=" or 1=1 –
&password=message2 HTTP/1.0")
```

pattern.

4) XSS Algorithm

For the Cross-Site Scripting algorithm system is attacked with the GET and POST commands, so it filters only those two Kafka topics. Similar to the SQLialgorithm, first, rearrange request string into a specific pattern. Second, filter for malicious IP addresses that has request pattern match with SQLi pattern Finally, save all malicious attempts into a log file.

B. *Authors and Affiliations*

The template is designed for, but not limited to, six authors. A minimum of one author is required for all conference articles. Author names should be listed starting from left to right and then moving down to the next line. This is the author sequence that will be used in future citations and by indexing services. Names should not be listed in columns nor group by affiliation. Please keep your affiliations as succinct as possible (for example, do not differentiate among departments of the same organization).

1) *For papers with more than six authors:* Add author names horizontally, moving to a third row if needed for more than 8 authors.

2) *For papers with less than six authors:* To change the default, adjust the template as follows.

a) Selection: Highlight all author and affiliation lines.

b) Change number of columns: Select the Columns icon from the MS Word Standard toolbar and then select the correct number of columns from the selection palette.

c) Deletion: Delete the author and affiliation lines for the extra authors.

C. *Identify the Headings*

Headings, or heads, are organizational devices that guide the reader through your paper. There are two types: component heads and text heads.

Component heads identify the different components of your paper and are not topically subordinate to each other. Examples include Acknowledgments and References and, for these, the correct style to use is “Heading 5”. Use “figure caption” for your Figure captions, and “table head” for your table title. Run-in heads, such as “Abstract”, will require you to apply a style (in this case, italic) in addition to the style provided by the drop down menu to differentiate the head from the text.

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named “Heading 1”, “Heading 2”, “Heading 3”, and “Heading 4” are prescribed.

D. *Figures and Tables*

a) Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation “Fig. 1”, even at the beginning of a sentence.

TABLE I. TABLE TYPE STYLES

Table Head	**Table Column Head**		
	Table column subhead	***Subhead***	***Subhead***
copy	More table copy[a]		

[a] Sample of a Table footnote. (*Table footnote*)

Fig. 1. Example of a figure caption. (*figure caption*)

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity “Magnetization”, or “Magnetization, M”, not just “M”. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write “Magnetization (A/m)” or “Magnetization {A[m(1)]}”, not just “A/m”. Do not label axes with a ratio of quantities and units. For example, write “Temperature (K)”, not “Temperature/K”.

ACKNOWLEDGMENT *(Heading 5)*

The preferred spelling of the word “acknowledgment” in America is without an “e” after the “g”. Avoid the stilted expression “one of us (R. B. G.) thanks ...”. Instead, try “R. B. G. thanks...”. Put sponsor acknowledgments in the unnumbered footnote on the first page.

REFERENCES

[1] T. Kakarla, A. Mairaj and A. Y. Javaid, "A Real-World Password Cracking Demonstration Using Open Source Tools for Instructional Use," 2018 IEEE International Conference on Electro/Information Technology (EIT), 2018, pp. 0387-0391, doi: 10.1109/EIT.2018.8500257.

[2] C. Ping, "A second-order SQL injection detection method," 2017 IEEE 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2017, pp. 1792-1796, doi: 10.1109/ITNEC.2017.8285104.

[3] J. K. R, S. Balaji B, N. Pandey, P. Beriwal and A. Amarajan, "An Efficient SQL Injection Detection System Using Deep Learning," 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), 2021, pp. 442-445, doi: 10.1109/ICCIKE51210.2021.9410674.

[4] T. H. Hai and N. T. Khiem, "Architecture for IDS Log Processing using Spark Streaming," 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), 2020, pp. 1-5, doi: 10.1109/ICECCE49384.2020.9179188.

[5] L. Bošnjak, J. Sreš and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2018, pp. 1161-1166, doi: 10.23919/MIPRO.2018.8400211.

[6] P. N. Joshi, N. Ravishankar, M. B. Raju and N. C. Ravi, "Contemplating Security of Http From SQL Injection and Cross Script," 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2017, pp. 1-5, doi: 10.1109/ICCIC.2017.8524376.

[7] M. M. Najafabadi, T. M. Khoshgoftaar, C. Calvert and C. Kemp, "Detection of SSH Brute Force Attacks Using Aggregated Netflow Data," 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), 2015, pp. 283-288, doi: 10.1109/ICMLA.2015.20.

[8] Keonwoo Kim, "Distributed password cracking on GPU nodes," 2012 7th International Conference on Computing and Convergence Technology (ICCCT), 2012, pp. 647-650.

[9] H. S. Shreenidhi, S. Prabakar and P. A. Kumar, "Intrution detection system Using IoT device for safety and security," 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), 2021, pp. 340-344, doi: 10.1109/ICCIKE51210.2021.9410730.

[10] B. Debnath et al., "LogLens: A Real-Time Log Analysis System," 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS), 2018, pp. 1052-1062, doi: 10.1109/ICDCS.2018.00105.

[11] S. V. Siva reddy and S. Saravanan, "Performance Evaluation of Classification Algorithms in the Design of Apache Spark based Intrusion Detection System," 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020, pp. 443-447, doi: 10.1109/ICCES48766.2020.9138066.

[12] M. T. Tun, D. E. Nyaung and M. P. Phyu, "Performance Evaluation of Intrusion Detection Streaming Transactions Using Apache Kafka and Spark Streaming," 2019 International Conference on Advanced Information Technologies (ICAIT), 2019, pp. 25-30, doi: 10.1109/AITC.2019.8920960.

[13] S. Honda, Y. Unno, K. Maruhashi, M. Takenaka and S. Torii, "TOPASE: Detection of brute force attacks used disciplined IPs from IDS log," 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), 2015, pp. 1361-1364, doi: 10.1109/INM.2015.7140496.

[14] Elmasry, Wisam, Akbulut, Akhan and Zaim, Abdul Halim. "A Design of an Integrated Cloud-based Intrusion Detection System with Third Party Cloud Service" Open Computer Science, vol. 11, no. 1, 2021, pp. 365-379. https://doi.org/10.1515/comp-2020-0214

[15] Wenyi Xu (2017) Visor: Real-time Log Monitor [Source Code]. https://github.com/xuwenyihust/Visor

[16] payloadbox (2021) SQL Injection Payload List [Dataset]. https://github.com/payloadbox/sql-injection-payload-list

[17] payloadbox (2021) Cross Site Scripting (XSS) Vulnerability Payload List [Dataset]. https://github.com/payloadbox/xss-payload-list

[18] iryndin (2018) 10K-Most-Popular-Passwords [Dataset]. https://github.com/iryndin/10K-Most-Popular-Passwords

[19] https://spark.apache.org/docs/latest/

[20] https://kafka.apache.org/documentation/

[21] https://cloud.google.com/