VSOC – A Virtual Security Operating Center

Eric Falk*, Stefan Repcek*, Beltran Fiz*, Stefan Hommes*, Radu State*
Raimondas Sasnauskas†
*University of Luxembourg, SnT
6 Rue Richard Coudenhove-Kalergi, 1359 Luxembourg
Email: {eric.falk, beltran.fiz, stefan.hommes, radu.state}@uni.lu, repcek2@gmail.com
†SES Engineering
Château de Betzdorf, Rue Pierre Werner, L-6815 Betzdorf
Email: raimondas.sasnauskas@ses.com

Abstract—**Security in virtualised environments is becoming increasingly important for institutions, not only for a firm's own on-site servers and network but also for data and sites that are hosted in the cloud. Today, security is either handled globally by the cloud provider, or each customer needs to invest in its own security infrastructure. This paper proposes a Virtual Security Operation Center (VSOC) that allows to collect, analyse and visualize security related data from multiple sources. For instance, a user can forward log data from its firewalls, applications and routers in order to check for anomalies and other suspicious activities. The security analytics provided by the VSOC are comparable to those of commercial security incident and event management (SIEM) solutions, but are deployed as a cloud-based solution with the additional benefit of using big data processing tools to handle large volumes of data. This allows us to detect more complex attacks that cannot be detected with todays signature-based (i.e. rules) SIEM solutions.**

I. Introduction

Securing IT infrastructures is one of the most challenging aspects for cloud operators nowadays. While traditional networks usually had a limited number of servers and hosts that could be secured by respective security devices (e.g. firewall), the new trend of cloud computing has increased the number of virtualised hosts tremendously and thus makes it even more difficult to adapt to the required level of security. At the same time, large numbers of log files produced by the various entities need to be processed in order to detect malicious activities near real-time.

In order to assure security monitoring and mitigation, a *Security Operating Center* (SOC) is a dedicated entity that is used by cloud operators for this purpose. Moreover, it includes not only the technical means but also the required personal and administrative aspects that are needed to thwart attacks successfully. In order to monitor the ICT infrastructure itself, *Security Information and Event Management* (SIEM) solutions are deployed that include the following features: 1) Import and parsing of various log sources and formats, 2) Signature-based detection of events by using customised rule sets, 3) Creation of reports and visualisation of data for the security officers.

While traditional SIEM solutions are expensive and limited by its configuration capabilities, we propose a *Virtualised SOC* (VSOC) that is based on open source software and is operated in a cloud infrastructure. It allows the integration of various type of log sources, and the parsing of different log entries which are then forwarded and stored in a distributed streaming platform. This includes a messaging bus that streams data based on a publisher and subscriber model, and allows various modules to receive data for further processing. We developed our solution based on the Kappa [1] architecture paradigm, allowing real-time processing of events by using a customisable set of rules. Moreover, we provide the operator a platform for large scale data processing, allowing the development of new algorithms for anomaly detecting, for instance by using a machine-learning based approach to detect outliers by using unsupervised learning. We implemented our VSOC solution by using OpenStack [2] as a cloud environment, since the latter is highly customisable and used by many IT companies in operational mode. In order to minimise the effort for a cloud operator to set up a VSOC instance for a customer, we used orchestration tools to automate this process. This also allows for a multi-tenancy mode, which gives the security cloud provider the capability to monitor multiple customers by using multiple VSOC instances within a single OpenStack environment.

The underlying work is the result of a Proof-of-Concept (PoC) that is based on three years research in the area of high velocity security data analysis and more specifically IP flow records (Netflow). For many clients in the ICT industry, due to legal and market requirements (e.g. PCI DSS, ISO27000, NIST 800-66, NIST 800-53, FISMA), there is a large interest in a solution that provides an easy to use service and that is offering similar capabilities than commercial SIEM solutions. Our approach achieves this goal by being highly customisable, and by supporting Big Data workloads while relying on open source software. We successfully build and tested the VSOC as a prototype, which is ready for operational usage by cloud operators.

The paper is structured as follows. In section II, we discuss the related work for security monitoring in both research and industry. Section III defines the requirements for a virtualised SOC, and we describe the architecture of our solution in section IV. In section V, through the description of a deployed use case, we illustrate how VSOC can be used for an event processing task and a more general stream processing job. We conclude our work in section VI with some final remarks.

II. Related Work

The challenges of securing a today's IT infrastructure are manifold and are becoming even more complicated with new policies such as "bring your own device". The surfaces for cyber-attacks are becoming larger and the attacks themselves more sophisticated, often times targeting multiple edges of an infrastructure at the same time. Current *Intrusion Detection Systems* (IDS) and *Intrusion Prevention Systems* (IPS) are mostly non-collaborative, which makes them less and less adequate in front of new threats [3][4]. The demand for a mature SOC, capable of facing these challenges, is therefore heavily increased.

In their survey work about Big Data supported security monitoring in critical infrastructures, Aniello et al. [4] define three axis a modern full-fledged system has to cover: a) The appliance must be capable of handling large amounts of data from different sources; b) the depth of the data analysis, or the granularity of the data must be adaptable; c) and finally, the library of known attack scenarios must be extensible to stay up-to-date and to enable state-of-the-art detection mechanism. Several academic papers approached the subject of Big Data monitoring for security purposes. In the remainder of this section we provide a brief survey and classify the solutions according to the requirements listed above.

In [5][6] disjunct techniques of security monitoring are applied on a same massive datasets. In [5] the authors propose MapReduce [7] mechanics to detect Botnets in large amounts of Netflow data. In [6] Marchal et al. introduce *PhishStorm*, a system for real-time detection for phishing URLs in voluminous datasets. Another related work describes an architecture performing intrusion detection through the analysis of merged data sources [8]. The system named *Beehive* extracts information from DHCP servers, VPN connections, logs from Virus scans and from web proxies, and stores it in a commercial SIEM solution. A well defined set of features is extracted from the merged data, and is clustered prior to the proper anomaly detection. To handle large amounts of data, Beehive relies on data specific reduction methods. Previous work carried out in our group resulted in a Big Data architecture for large scale security monitoring [9]. It allows to render real-time decisions by combining insights from logged DNS requests, HTTP requests, accesses to honeypots and Netflow data. Our appliance, in contrast to Beehive, is build on top of open source distributed file storages which improves the scalability. Nevertheless, neither of the previous systems satisfies the requirements from [4]. Despite being scalable for the handling of growing amounts of data, the adjustment of other system aspects is cumbersome. For instance, extending the systems to monitor additional data sources or compute aggregations and apply anomaly detection methods beyond the initial scope can only be achieved with significant development efforts. Also, to support adaptable data granularity it must be possible to seamlessly add and remove monitored data sources. Likewise, to maintain an evolving dictionary of attack patterns knowledge must be extracted from historical data. In this work we introduce VSOC, a system designed to address the challenges described in this section.

With regards to commercial SIEM solutions, the most prominent representatives are IBM QRadar, HP ArcSight and Splunk Enterprise [10]. IBM QRadar includes event processing based on signatures, but adds other modules such as deep packet inspection and vulnerability scanners which makes it more specific to network security requirements. HP ArcSight can be considered as a classical SIEM solution with a focus on event correlation, and is highly customisable with regards to type of log sources, rules and reports. Moreover, it provides some work flow capabilities which include a ticketing system to support the security experts. Splunk is focussing on log management, and has a powerful search engine but only basic correlation capabilities. The pricing of the commercial SIEM solution is either based on the number of events per second (EPS), the number of log sources per day, or the size of data indexed per day. Since the number of events can grow rapidly in larger networks, our solution is less expensive when compared to commercial SIEM solutions since it has no license restrictions towards the number of processed events or log sources.

III. Requirements

In this section we define the requirements a modern Security Operating Center (SOC) needs to fulfill. We mainly iterate over the requirements defined in [4], outlined in the previous section, and we extend the former with regards to a virtualised cloud solution.

Multi-modal log sources — Current ICT infrastructures consists of a large variety of devices, for instance routers, firewalls, IDS appliances or software running in virtualised server environments. In case of an incident, a log entry is created with detailed information about the event. Our system needs to be able to import and parse various log sources and log files.

Big Data processing — Log sources can easily grow to a large amount of data that needs to be analysed for security purposes. Our architecture needs to handle such large amounts of data and needs to distribute it to various consumers. The granularity of the data and number of consumers needs to be adaptable for each data source.

Security monitoring — The detection of security incidents should be realised in two different ways: 1) A signature-based mode should be able to detect an incident based on a defined set of rules, which needs to be adapted to the respective customer and scenario. And 2), a machine learning based approach should allow the detection of more complex anomalies, for instance by creating a network profile over time in order to detect outliers by using unsupervised learning.

Virtualisation — Our system should be operated in a cloud environment, and might either be operated as a standalone application or integrated into an already existing cloud architecture (e.g. Microsoft Azure, OpenStack). This enables the adaptation to variations in the workload, by adding or removing resources (e.g. virtual machines (VMs)) automatically, and so ensure a reliable monitoring process. Moreover, it simplifies the integration of log sources that are based on applications that are already running in VMs or containers.

Multi-Tenancy — Our system is focussing on security service providers, that need to monitor multiple customers in one single VSOC environment. This requires the separation of data, a dedicated dashboard and reports for each customer.

IV. Architecture

In the following, the requirements enumerated in section III are projected on a Big Data architecture. The latter must support large scale batch operations and posses real-time processing capabilities. Both criteria are rather exclusive for big data architectures although proposals exist to achieve both. The two most prevalent are commonly known as the Lambda [11] and the Kappa [1] architectures.

The assumption behind the two paradigms is that models of strong consistency are generated by long running batch computations over the full dataset, while fast but less consistent models are gained by stream processing sliding windows of data. The Lambda architecture is composed by three layers: a) a batch layer running the long term computation on the full sized master dataset; b) a speed layer updating the output of the batch layer until the model of the next batch layer loop is available; c) and finally a query layer merging the results from the query and batch layer for user queries. The typical Lambda architecture as depicted in [11] relies on Hadoop [7][12] for the batch layer, Apache Storm [13] for the speed layer, and on ”off-the-shelf” *Relational Database Management Systems* (RDBMS) for the query layer.

The Kappa architecture, instead of storing the master dataset on the *Hadoop Distributed File System* (HDFS) persists the data on Apache Kafka [14], a distributed log aggregator. Kafka implements a messaging API, hence a stream processor can consume messages in their chronological order to perform computations. To generate models from the entire master dataset it suffices to spawn a second instance of the stream processing job, which requests all messages stored in Kafka. To assure that one strongly consistent model is permanently available, n replications of stream analytics process have to be synchronized accordingly.

For the VSOC project the Kappa Architecture paradigm is retained as it is more agile, especially in accordance with the Virtualisation and Multi-Tenancy requirements. VSOC relies on OpenStack [2] to start virtual containers, each taking a role from the set of different actors of the Kappa architecture. To guarantee Multi-Tenancy, several analytics pipelines must be deployable, and since the Kappa Architecture relinquishes a full-fledged HDFS installation, the former is leaner in a virtual environment than the Lambda Architecture. In the remainder of this section, the implementation details of the VSOC platform are described from two angles. The environment comprising the units building a Kappa architecture define what is called the *Tenant Space*, whereas the *Orchestration Space* designates the management layer on top of the virtual environments. Said otherwise the Orchestration Space provides the tools to coordinate multiple Tenant Spaces, either manually or automatically.

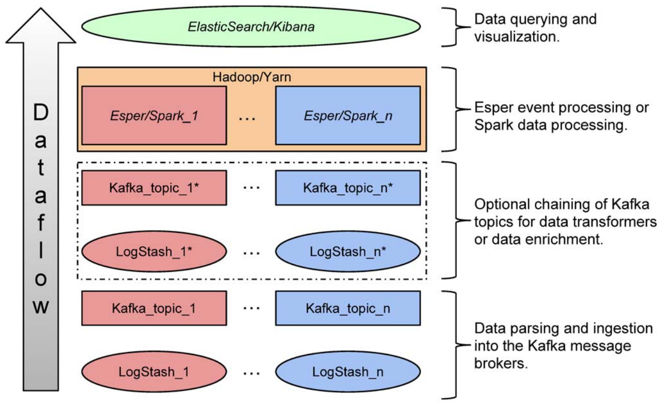
ElasticSearch/Kibana
Data querying and visualization.
Hadoop/Yarn
Esper/Spark_1
Esper/Spark_n
Esper event processing or Spark data processing.
Kafka_topic_1*
Kafka_topic_n*
LogStash_1*
LogStash_n*
Optional chaining of Kafka topics for data transformers or data enrichment.
Kafka_topic_1
Kafka_topic_n
LogStash_1
LogStash_n
Data parsing and ingestion into the Kafka message brokers.
Dataflow

Fig. 1. VSOC Architecture: Tenant Space

A. Tenant Space

All VSOC units deployed in the user space are deployed on demand or dynamically as OpenStack virtual machines. For an optimal utilization of bare-metal resources single VMs feature minimalistic Linux operating systems. Figure 1 shows a schematic representation of a VSOC Tenant Space. To cover a maximum amount of potential use cases the VSOC platform provides two modes: a *default mode* and an *expert mode*.

The data integration units called *collectors* are based on LogStash [15], a well known data processing utility. In default mode users can resort to a variety of ”out-of-the-box” LogStash data parsers, while in expert mode custom made parsers can be installed as long as they are compliant to the LogStash format. The utility transforms raw data into events. Depending on the task's requirements raw data can be written to Kafka before being parsed, or should different formats of the same data be required, it is possible to chain LogStash parsers and Kafka topics. The sequence of the stream transformation units in the data pipelines is completely configurable.

On the opposite of Kafka based data transition are the *event processors*. Again two modes are available. The default mode relies on Esper [16], a framework for *Complex Event Processing* (CEP) [17] and event correlation. Esper allows users to define queries to be applied on the stream of incoming events. The queries are expressed with a declarative CEP query language (an example is given in section V). A predefined set of security monitoring relevant queries are packaged in VSOC. In addition to the possibility to define custom Esper queries, in expert mode more advanced analysis methods, even proper stream processing on raw data can be applied, by providing a custom Spark Streaming [18] application. Both types of event processors are distributed and run inside a YARN [19] container, YARN being an all purpose distributed resource manager included in the Hadoop package.

The output from the event processors is stored in a distributed indexing engine called ElasticSearch [20]. ElasticSearch is seamlessly horizontally scalable and provides a

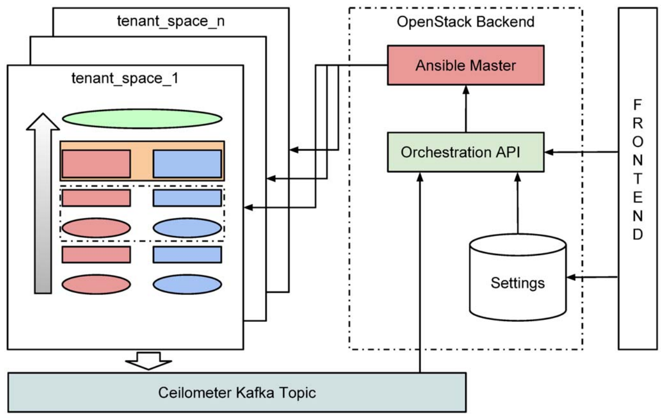

Fig. 2. VSOC Architecture: Orchestration Space

rich user interface for queries and graphical visualizations of the results from the processing units. To visualise the data stored in ElasticSearch, Kibana [21] is deployed as the default visualization utility, although further custom interfaces can be set up by implementing the REST API exposed by ElasticSearch.

B. *Orchestration Space*

In VSOC the process of coordination is managed by the custom made *Orchestration API*. The latter can be controlled via a web interface but also supports automated procedures. The web application sends requests to the Orchestration API of the OpenStack installation hosting the working environment of the tenant. Each request is performed by a specific tenant and has to be authenticated. Through the same web interface, a configuration file describing the types of resources to include within a tenant's initial stack is generated. The information contained in this file include details about stream processing tools (Spark, Esper), the data sources (logs, LogStash configuration, Kafka topics) and the cloud infrastructure (virtual machines, virtual networks, routers etc.). The configuration files also describe the set of rules and thresholds under which to scale the tenant's resources. For example, it can be defined that if the CPU utilization of an event processor unit exceeds the 90% mark it has to be replaced by a more powerful one. The system resources utilization information of each single VM is obtained through the *OpenStack Ceilometer*. The Ceilometer is an OpenStack module providing data collection services for OpenStack core units. The Ceilometer performance data is written to a dedicated Kafka cluster from where it is exploited by our Orchestration API (see figure 2) for decision making. For the automatic coordination of the *Tenant Space* the Orchestration API also takes into account tenant settings, which are limited to the terms of agreement with the VSOC operators.

The technical part of the orchestration is managed through Ansible [22], which is an open source tool for the configuration and management of large clusters of computing devices. Ansible uses pure SSH and implements an agent-less ”push” model to manage resources which means that all machines in the Tenant Space can be managed from one single node: the *Ansible Master*. Ansible can manage a cluster in two modes: via ad-hoc commands or via *playbooks*. Ad-hoc commands are simple one-line commands which perform tasks on the target machine. The initial configuration of the resources within a VSOC Tenant Space is described using playbooks which can be managed and modified through the user interface, whereas scaling events such as the automated deployment of more powerful resources is done through ad-hoc commands. For the launch of a Tenant Space, the configuration file build based on parameters from the web interface is converted to an Ansible playbook. The latter regroups information about the instances as for example the instance names, resources (CPU, RAM), purposes (data collector, data broker, or data processor), but also the configuration of the hosted applications themselves: the LogStash data parsing format, the Esper event processing policies, or the Spark Streaming application. The instantiation of the requested Tenant Space is then performed by the Ansible Master by running the playbook (see figure 2).

In the next chapter the functionality of event processing with VSOC is illustrated with a concrete use case. The resulting application has been deployed, and is automatically scaled in accordance with the descriptions provided above.

V. Event Processing

An installation of VSOC is running in our lab on a 7 machine cluster allocated as follows: 3 Dell PowerEdge R210 units with 8 2.66 Ghz CPUs and 24 GB RAM host the Orchestration Space applications, and 4 Dell PowerEdge R430 servers (32 2.39 Ghz CPUs and 96 GB RAM) are dedicated to Tenant Spaces.

As mentioned in section IV, the possibilities for data analytics in VSOC are twofold. By uploading a custom Spark Streaming application virtually every streaming analytics or machine learning task can be deployed in VSOC. Tenants can rely on a waste amount of Spark tools as for instance MLlib [23] for machine learning and GraphX [24] for extensive graph processing tasks. The second option in VSOC is the usage of the CEP framework Esper introducing the *Event Pattern Language* (EPL), a declarative language for the querying of event streams in real-time. In a Tenant Space a cluster of Esper processors is placed across a set of virtual machines with the help of the YARN resource manager. Esper jobs are suited best for simple stream processing (filtering, aggregating streams) and event-based analysis where the goal is to detect event correlations and create a reaction model based on it. The advantage of the Esper model over the Spark model is that no application has to be programmed and tested. EPL queries are specified at the time of configuration of a Tenant Space and remain alterable throughout the uptime of the Tenant Space. VSOC also includes a variety of predefined EPL queries for well-known security monitoring tasks.

In the upcoming subsection, we illustrate the functionality of VSOC with Esper event processing by describing how the monitoring tasks from [9] are reproduced on the more flexible VSOC platform.

A. Network Monitoring

For network monitoring, our group previously built [9] four data sources that were further observed: DNS requests, Netflow records from routers, HTTP requests and accesses to corporate honeypots. For that purpose the current platform from [9] employs heterogeneous data storages and therefore different techniques and tool-sets for the analysis. In order to monitor the network three scores were derived:

- Dom_{score} expresses the maliciousness of a DNS request. The data is stored in an Apache Cassandra Database [25].
- $Flow_{score}$ assessing the risk of communication using Netflow: stored as plain Cisco Netflow [26] files.
- Web_{score} to determine the maliciousness of HTTP traffic. HTTP packages are directly forwarded to the event processors.

The honeypot data is stored in a standard RDBMS. The heterogeneous nature of the data storage leads to complex monitoring applications and hinders scalability. The work flow for the score computations is shown in Figure 3.

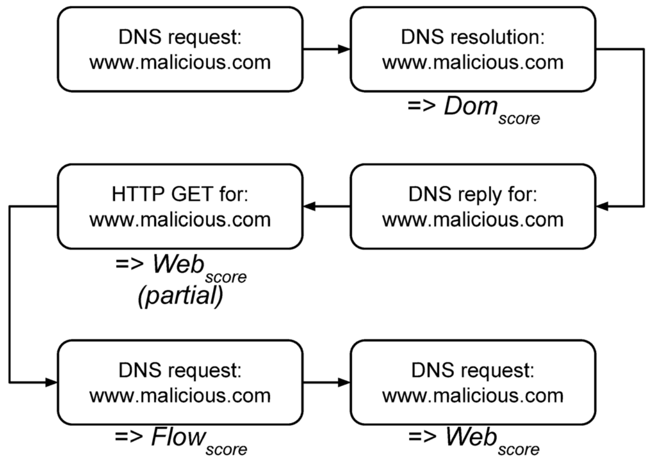
DNS request: www.malicious.com
DNS resolution: www.malicious.com
=> Dom score
HTTP GET for: www.malicious.com
DNS reply for: www.malicious.com
=> Web score (partial)
DNS request: www.malicious.com
DNS request: www.malicious.com
=> Flow score
=> Web score

Fig. 3. Information flow graph and score computation for the HTTP request to www.malicious.com (as described in previous work [9]).

On the VSOC platform the same monitoring mechanics are reproduced as follows. The data from each source is parsed by LogStash and written to a dedicated Kafka topics in parallel. It is to be noted that a Kafka topic can be partitioned over several virtual machines to adapt the throughput to the data volume. This information can be derived from the Ceilometer performance data enabling the automatic scaling of the data brokers. From the Kafka topics the data is read in chronological order by respective Esper event processors. The honeypot topic for instance, is consumed by all processors to match IPs which accessed the honeypot against IPs involved in Netflow, DNS and HTTP requests. Figure 4 shows the respective VSOC work-flow for this scenario.

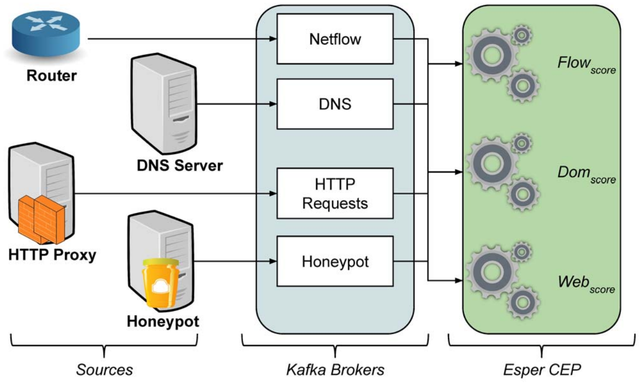

Fig. 4. Data sources write the information to a dedicated Kafka topic, from where the data can be consumed at will by respective Esper jobs. In the underlying use case, besides the respective data source, the honeypot data is required for the computation of each single score.

An example of matching IP addresses from HTTP requests with IP addresses from honeypot accesses with a corresponding Esper query is shown below:

Esper Query 1. *Be **HPEvents** a cached lookup table of IPs which accessed the honeypot in the past. HPEvents is maintained by the messages transiting through the honeypot Kafka topic. **HTTPEvents** are HTTP requests incoming in a pre-defined time frame.*

```
select HTTPEvents.*, HPEvents.*
from   HTTPEvents.win:time(1 min),
       HPEvents.win:time(60 min)
where  HTTPEvents.srcIP = HPEvents.ip or
       HTTPEvents.dstIP = HPEvents.ip
```

In the query above (query 1) it can be seen that source and destination IP addresses of HTTP requests from the past minute are matched against IP addresses accessing the honeypot over the past hour. This type of query is the initial step in the computations of the Dom_{score}, $Flow_{score}$ and Web_{score} values. The results of the score computations are stored in the ElasticSearch environment to maintain user dashboards.

VI. Conclusion

We have designed, implemented and tested a virtualised security operating center that is able to monitor a cloud infrastructure consisting of multiple tenants. Our idea is motivated by state of the art data processing models and allows the integration of various log source that are further processed by a distributed streaming platform. This increases the flexibility of our approach, allowing to have multiple consumers that analyse data in order to find security violations. The latter can be detected either by using customisable signatures that trigger an alarm, or by using Apache Spark that allows to detect more complex events based on machine learning techniques. The orchestration of the various components is fully automated and reduces the orchestration effort for the operator and tenant to a minimum. We have tested the VSOC by implementing it as a

Proof-of-Concept into an OpenStack based cloud environment. While the set up phase is supported by a graphical user interface, a dedicated dashboard is provided to the security operator and includes information about the current monitoring activities.

Acknowledgment

The project was supported by the luxembourgish National Research Fund (FNR) based on a Proof-of-Concept proposal.

References

[1] J. Kreps, *I Heart Logs: Event Data, Stream Processing, and Data Integration*. O'Reilly Media, 2014. [Online]. Available: https://books.google.lu/books?id=l7nvoQEACAAJ

[2] Openstack open source cloud computing software. https://www.openstack.org/.

[3] L. Aijaz, B. Aslam, and U. Khalid, "Security operations center - a need for an academic environment," in *2015 World Symposium on Computer Networks and Information Security (WSCNIS)*, 9 2015, pp. 1–7.

[4] L. Aniello, A. Bondavalli, A. Ceccarelli, C. Ciccotelli, M. Cinque, F. Frattini, A. Guzzo, A. Pecchia, A. Pugliese, L. Querzoni, and S. Russo, "Big data in critical infrastructures security monitoring: Challenges and opportunities," *CoRR*, vol. abs/1405.0325, 2014. [Online]. Available: http://arxiv.org/abs/1405.0325

[5] J. Francois, S. Wang, W. Bronzi, R. State, and T. Engel, "Botcloud: Detecting botnets using mapreduce," in *Proceedings of the 2011 IEEE International Workshop on Information Forensics and Security*, ser. WIFS '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 1–6.

[6] S. Marchal, J. François, R. State, and T. Engel, "Phishstorm: Detecting phishing with streaming analytics," *IEEE Transactions on Network and Service Management*, vol. 11, no. 4, pp. 458–471, 12 2014.

[7] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 1 2008.

[8] T.-F. Yen, A. Oprea, K. Onarlioglu, T. Leetham, W. Robertson, A. Juels, and E. Kirda, "Beehive: Large-scale log analysis for detecting suspicious activity in enterprise networks," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ser. ACSAC '13. New Orleans, Louisiana, USA: ACM, 2013, pp. 199–208. [Online]. Available: http://doi.acm.org/10.1145/2523649.2523670

[9] S. Marchal, X. Jiang, R. State, and T. Engel, "A big data architecture for large scale security monitoring," in *Proceedings of the 2014 IEEE International Congress on Big Data*, ser. BIGDATACONGRESS '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 56–63.

[10] Gartner, "Magic quadrant for siem - summary commentary and mq placement," Gartner, Inc., Tech. Rep., 2016.

[11] N. Marz and J. Warren, *Big Data: Principles and Best Practices of Scalable Real-time Data Systems*. Manning, 2015. [Online]. Available: https://books.google.lu/books?id=HW-kMQEACAAJ

[12] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, ser. MSST '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[13] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy, "Storm@twitter," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '14. Snowbird, Utah, USA: ACM, 2014, pp. 147–156.

[14] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece*, 2011.

[15] Logstash: Centralize, transform and stash your data. https://www.elastic.co/products/logstash.

[16] Esper: Complex event processing - cep. http://www.espertech.com/esper/.

[17] N. Leavitt, "Complex-event processing poised for growth," *Computer*, vol. 42, no. 4, pp. 17–20, 4 2009.

[18] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica, "Discretized streams: Fault-tolerant streaming computation at scale," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, ser. SOSP '13. Farminton, Pennsylvania: ACM, 2013, pp. 423–438.

[19] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth, B. Saha, C. Curino, O. O'Malley, S. Radia, B. Reed, and E. Baldeschwieler, "Apache hadoop yarn: Yet another resource negotiator," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13. Santa Clara, California: ACM, 2013, pp. 5:1–5:16.

[20] Elasticsearch: Restful, distributed search and analytics. https://www.elastic.co/products/elasticsearch.

[21] Kibana: Explore, visualize, discover data. https://www.elastic.co/products/kibana.

[22] D. Hall, *Ansible Configuration Management*. Packt Publishing, 2013.

[23] Mllib: Spark's scalable machine learning library. http://spark.apache.org/mllib/.

[24] Graphx: Spark's api for graphs and graph-parallel computation. http://spark.apache.org/graphx/.

[25] A. Lakshman and P. Malik, "Cassandra: A decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, pp. 35–40, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1773912.1773922

[26] Cisco netflow. http://www.cisco.com/c/en/us/products/ios-nx-os-software/ios-netflow/index.html.