A data infrastructure for heterogeneous telemetry adaptation. Application to Netflow-based cryptojacking detection

Alejandro A. Moreno-Sancho
Cybersecurity Laboratory
Atos Research & Innovation
Málaga, Spain
0000-0001-8863-1875

Antonio Pastor
GCTIO Office
Telefónica I+D
Madrid, Spain
0000-0003-2849-9782

Ignacio D. Martinez-Casanueva
GCTIO Office
Telefónica I+D
Madrid, Spain
0000-0002-8573-127X

Daniel González-Sánchez
Dpto. de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
Madrid, Spain
0000-0002-7691-0030

Luis Bellido Triana
Dpto. de Ingeniería de Sistemas Telemáticos
Universidad Politécnica de Madrid
Madrid, Spain
0000-0001-9591-0928

***Abstract*—The increasing development of cryptocurrencies has brought cryptojacking as a new security threat in which attackers steal computing resources for cryptomining. The digitization of the supply chain is a potential major target for cryptojacking due to the large number of different infrastructures involved. These different infrastructures provide information sources that can be useful to detect cryptojacking, but with a wide variety of data formats and encodings. This paper describes the Semantic Data Aggregator (SDA), a normalization and aggregation system based on data modelling and low-latency processing of data streams that facilitates the integration of heterogeneous information sources. As a use case, the paper describes a Cryptomining Detection System (CDS) based on network traffic flows processed by a machine learning engine. The results show how the SDA is leveraged in this use case to obtain aggregated information that improves the performance of the CDS.**

***Index Terms*—Netflow, YANG, Data modelling, Data Normalization, Data Aggregation, Supply chain, Cryptojacking.**

I. INTRODUCTION

The advancement of cloud technologies and IoT has enabled the digitization of the supply chain, thus achieving countless benefits, however, this also leaves the supply chain open to potential attacks [1][2]. Projects such as FISHY [3], seek to develop solutions that bring cyber resilience to the supply chain to enable business continuity. With supply chain systems, we are faced with a considerable amount and variety of data. When IT is combined with OT (Operational Technology), and information is obtained from a wide variety of infrastructures (e.g., cloud, network, IoT...), the schema and even the encoding of the data can be very diverse. In order for the various modules or applications that conform a project to consume the monitored data, they must not only be adapted to the correct format and encoding, but also be anonymized, filtered or even derived and enriched if necessary. This task is both costly and not reusable, as each consumer needs the data in a specific schema and encoding format.

Moreover, supply chain systems may require a high computational capacity since they must process a very large amount of data, therefore making them the perfect target for cryptomining attacks, which have been enhanced as a result of cryptocurrency developments. Traditional Intrusion Detection Systems (IDS), which use Deep Packet Inspection (DPI) and signature-based methods, are not effective for detecting this kind of malware due to the complexity of infrastructure, the massive increase in traffic, and the heavy use of encrypted connections. But with the advent of ML (Machine Learning) mechanisms, IDS have become more efficient, especially when using flow statistics, which allows detection of threats with lower generated latency and even with encrypted traffic. Using ML for threat detection has its downsides: predictions are commonly not perfect, and its likelihood to have false positives and negatives is high. In addition, datasets are scarce, may contain poor quality samples (i.e., samples with a large number of erroneous measurements), and may have poor format specification. Some problems, such as traffic-related data scarcity, can be addressed by generating synthetic traffic, as reflected in [4], where Network Function Virtualization and Software Defined Networks are used for this purpose. Despite this solution, a pre-processing stage is still necessary, which must be particularly developed depending on the source and consumer.

This work describes the Semantic Data Aggregator (SDA), which through YANG data modelling, allows us to abstract from the encoding and data model differences between source and consumer. Such abstraction allows the generalization of the pre-processing stage, which will no longer need to be different for each source-to-consumer combination. On the other hand, a ML-based Cryptomining Detection System (CDS) is developed to act as a consumer of the SDA, since in projects such as FISHY [5], the impact that this type of attack can have on scenarios such as the connected vehicle becomes evident. Both the SDA and the CDS have already proven their usefulness in other projects such as PALANTIR [6], where they are used as components of the security-as-a-service solution.

The remainder of the document is organised as follows. Section II gives a brief description of the current state of the art of traffic monitoring, modelling, and threat detection technologies. Next, in section III, the architecture of the proposed system is detailed, with emphasis on individual elements of the architecture and how the system is deployed. This is followed by section IV with the evaluation of the system. And finally, section V, provides the conclusions and future directions of this research.

II. Background

A. Traffic Monitoring

1) SNMP: If a network component fails, the first step in getting it to repair itself automatically is for it to realize it is down. This is why traffic monitoring is so important. Later, traffic monitoring began to be used for more powerful tasks: knowing the network load, identifying needs, identifying problems, or even with security purposes.

Due to these monitoring and management needs, SNMP (Simple Network Management Protocol) was created. Any device that has a MIB (Management Information Base) can be managed and monitored, so it can support routers, switches, servers, and even printers. Metrics to pull from devices may include: CPU consumption, memory, temperature, or any variable that is present in the MIB of the device.

Netflow was created by Cisco with the purpose of monitoring network traffic. Netflow does not allow the collection of device information like SNMP; rather focused on network traffic flows. Netflow can only monitor IP traffic, which is not a problem considering that most of the traffic nowadays is IP. Netflow and SNMP are both different and complementary protocols, the choice of one or the other will depend on the objective to be achieved.

Netflow was originally implemented as a feature of Cisco routers. There are currently several versions (1, 5, 7, 8 and 9). Although an informative document exists [7], it is not really standarized.

2) Netflow version 9 (Netflowv9): Netflow version 9 is becoming one of the most widely used protocols for flow monitoring in security-related aspects. There are a large number of articles that refer to its use for threat detection systems using Machine Learning [8][9].

The purpose of Netflow is to monitor network flows, unidirectional sequences of packets that share the same source and destination IP, the same source and destination port, and the same protocol. The Netflow architecture consists of three main components:

- **Flow Exporter:** Responsible for aggregating the packets belonging to the same flows and sending them to the collectors.
- **Flow Collector:** It is in charge of receiving the flows sent by the exporter.
- **Analysis Application:** The Collector will send the information so it can be processed, stored, or analyzed by other applications for any purpose.

B. Modelling Languages

A modelling language refers to a set of terms and rules that allow the information structure and behaviour of a system to be represented in written form. Protocols such as the aforementioned SNMP have associated modelling languages, in this case, SMI (Structure of Management Information), based on ASN.1 (Abstract Syntax Notation One). With the emergence of NETCONF, there was a need for a language that defines the data models for the management of network infrastructures, so YANG [10] was developed.

Although its initial development was for NETCONF protocol and XML encoding, YANG now supports independent transport protocols such as RESTCONF or gNMI, and data encoding formats like XML, JSON, or PROTOBUF. Since the SDA long-term intention is to be able to handle data coming from heterogeneous sources, this is a considerable advantage, as it would not matter which protocol is used, or in which encoding the data is transmitted, as long as there is a YANG model describing the data.

The benefits of YANG apply both to the system that has been implemented and to the data itself. YANG allows to add semantic descriptions and metadata that make the data modelled more readable and understandable, so the benefit is not only at the machine level but also at the human level. Furthermore, it is quite common to obtain erroneous metrics and to have to perform a data pre-processing stage prior to the training of a ML algorithm. However, the normalization of the data received with respect to the YANG model used, will reduce the reception of erroneous metrics, since it is possible to accurately describe the structure and each field of the data, describing name, type of value, range for numerical values, and rules for text values. This flexibility in describing a field will allow any data received to be normalized without the need to implement specific pre-processing stages.

In regards to Netflow, there are YANG models developed by Cisco [11]. The models are for managing Netflow exporters and not for modelling Netflow monitoring data itself. This is why a model described in a previous work [12] is used. This work develops YANG models of the data provided by Netflow v9.

C. Threat Detection

One of the main motives for cyber attacks is to steal money. Ransomware have been one of the most famous methods since 2005. An analysis of the first ransomware attacks carried out in [13] indicates that some of the factors that made the attacks initially not very effective include the low reachability, a weak encryption, or the payment method.

Due to the development of more advanced accessibility and encryption techniques, ransomware attacks have improved considerably. Moreover, the creation of cryptocurrencies brought advantages such as privacy and security in transactions, leading to a better payment method. However, ransomware attacks have some disadvantages: the victim may not pay the ransom, and it can be simply protected against by making frequent, secured backups. On the other hand, the solution is not so simple when it comes to cryptomining attacks, which are on the rise. Studies such as Symantec show that 8 million attacks could be detected in just 3 months [14].

Cryptojacking is a type of malware that aims to infect a machine to use its computational resources for the purpose of benefiting the attacker by mining cryptocurrency. This is clearly an advantage compared to ransomware, where success depends on many factors such as the victim's willingness to pay the ransom. In cryptojacking, the more machines that are infected the better, since payment is guaranteed.

Traditional signature-based intrusion detection is insufficient to prevent cryptojacking due to the continuous and rapid development of attacks. In addition, IDS are only able to analyse the traffic passing through them. Using anomaly-based systems can also be challenging as cryptomining protocols hardly generate any traffic, as evaluated in [15]. Therefore, a proper monitoring scheme accompanied by a good detection system, can allow all traffic occurring inside the network to be analysed, and detect activity such as one machine trying to infect another.

Among all the detection techniques, the most common are to use device metrics or traffic related metrics. When using metrics from the device, it is common to use hardware metrics as the temperature, hardware consumption, CPU utilisation among others [16][17]. However, using these metrics is generally insufficient, as some of the cryptomining protocols have functionalities that make detection based on these parameters difficult. Some may set CPU usage limits, while others act for a short period and then move on to the next victim, making them difficult to detect by antivirus. Most algorithms even use operations that are common in benign applications such as compression and decompression. A solution to some of these inconveniences, along with an increase in the accuracy of the models, can be seen in [18] through the use of performance counters. They allow the status of components from the hardware level to the application level to be obtained, and are an effective solution to reduce the number of false positives [19][20].

On the other hand, there are detection solutions by means of the traffic produced. In this case systems have already been developed that satisfactorily classify flows based on features obtained from the traffic. For example in [21] it is used *libpcap* to capture traffic and an application based in *flowtbag* to transform it into flows. In [15] a passive network eavesdropper is used to capture cryptocurrency-related traffic along with non-malicious traffic from YouTube, Skype and office network traffic.

Taking advantage of the benefits of both methods, there are also studies [22][23] that use both metrics to implement detection systems.

This study focuses on the use of traffic metrics. The traffic metrics used are based on Netflow flows, which have already been analyzed for cryptocurrency purposes. In [24] The behaviour of Ethereum is analysed on the basis of Netflow related flows. Netflow/IPFIX is proposed to replace DPI-based detection systems in [25]. In [26] Pastor *et al.* propose a solution based in traffic flows for encrypted connections, making a comparative analysis between multiple ML algorithms and features extracted from different applications (i.e., Netflow and Tstat).

Although the main focus of this paper is the creation of a YANG model-based normalization and aggregation system for Netflow data, the development of cryptomining detection models is a very interesting proof of concept for such a system.

III. System design and implementation

A. Scenario, architecture and tools

A scheme of the complete scenario can be seen in Figure 1, where it can be seen on the left the simulated network that is being monitored, in the centre it can be seen the SDA, and finally on the right the CDS (Cryptomining Detection System), which will act as a consumer of the SDA. A brief description of each of the components of the scheme is given below, however, each of the components will be described in depth in the following sections.

OpenStack has been used to setup 11 machines, 10 (crypto1 to crypto10) of them will act as infected, the remaining one (Exporter) will be used to receive the traffic reflected from these machines. Each of the infected machines will send a copy of any traffic passing through its interface, whether it is cryptomining traffic or not, to the Exporter machine. The machines generate video and audio streaming, cloud storage, web surfing and other type of traffic, connecting to servers on the internet and within the dedicated network. For the generation of cryptomining traffic, different cryptomining Monero clients are used, these are connected to mining pools on the internet. Both cryptomining and normal traffic are formed by encrypted and non-encrypted connections, in order to accurately simulate all possible kinds of traffic in any device. The Exporter will use the Softflowd tool to transform received network traffic into valid Netflowv9 flows. Within the SDA, the Goflow2 tool will be in charge of collecting these Netflowv9 flows. From here, an input driver and an output driver, together with other applications developed with the Apache Flink framework, will make the relevant transformations until the final data is written in an output topic, which will be consumed by the CDS.

As shown in Figure 1, Apache Kafka acts as a message bus, providing interconnection between the different components and applications that will process the network lows. In the following subsections there is a detailed description of each of the applications developed with Flink, in addition to the

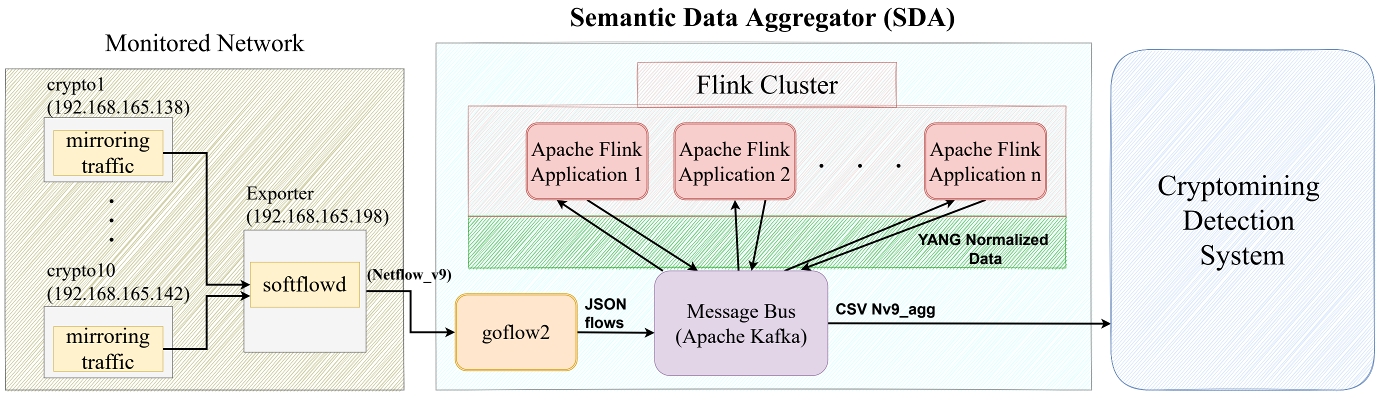
Semantic Data Aggregator (SDA)
Monitored Network
crypto1 (192.168.165.138)
mirroring traffic
crypto10 (192.168.165.142)
mirroring traffic
Exporter (192.168.165.198)
softflowd
(Netflow_v9)
goflow2
JSON flows
Flink Cluster
Apache Flink Application 1
Apache Flink Application 2
Apache Flink Application n
YANG Normalized Data
Message Bus (Apache Kafka)
CSV Nv9_agg
Cryptomining Detection System

Fig. 1: Diagram of the complete scenario: Monitored Network, SDA and CDS.

CDS, and the tools used for the Netflow related exporter and collector.

B. Netflow Exporter

The traffic from the 10 machines is mirrored to the eleventh one. Softflowd is used to obtain the Netflowv9 export packets from the mirrored traffic received. Then Softflowd is configured to push the export packets into the machine and port where the Netflow collector is listening.

Softflowd has been chosen over other exporters due to its ease of installation and configuration, as well as its flexibility, since it allows configuring parameters that facilitate the debugging of applications, such as creating Netflowv9 flows from PCAP files, or selecting the Netflow version to be generated.

C. Netflow Collector

The collector is the first component of the SDA. It receives Netflowv9 traffic and generates a number of JSON values equal to the number of flows inside the existing export packets.

For the collector we are using Goflow2, an open-source application programmed in Go that can be used for different versions of Netflow and also for IPFIX and sFlow protocols. It is developed for systems with a decent amount of samples and to scale horizontally, as is our case, since the monitored networks can send a considerable amount of traffic.

Through this project an open-source contribution have also been made to the Goflow2 solution, so that the output has been enriched, adding timestamps related to the flow with greater granularity than the already provided. Therefore, a greater consistency has been achieved with respect to the Netflow standard [27], by adding these improvements also in Netflow version 5 and IPFIX.

D. Input Driver

Java and Maven are used for the development of the applications, because Java and Scala are the languages in which the core of Flink is implemented. Moreover, the Java library YANG-TOOLS, which is an OpenDayLight compatible project, allows us to use YANG in JAVA. Using Maven and YANG-TOOLS we can generate Java classes from a YANG file, and instantiate them in the Flink application to serialize, deserialize and perform the relevant transformations.

In Figure 2, the input driver logic can be seen. First, the message received in JSON format from a Kafka topic, which is a NetFlow record previously published by the GoFlow2 collector, is mapped into the Java class generated from the YANG file. Then the data, which now follows the schema modelled in the YANG file, is serialized to be sent to another Kafka topic where the following application is subscribed.

The development of any application will have a similar methodology, the only difference will be that instead of a mapping between two schemas, it will be done any other data transformation.

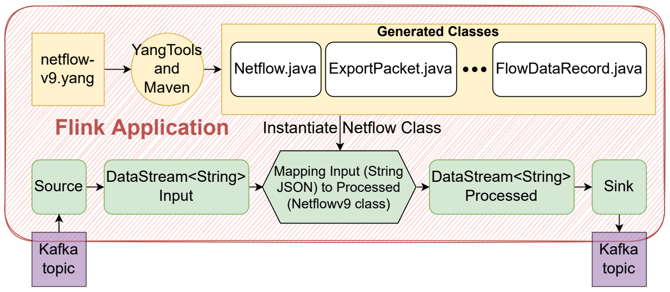

Fig. 2: Flink input-driver application architecture diagram.

A future proposal is to have the ability to achieve this one-to-one mapping without the need to develop new drivers, but to indicate to a generic driver how the mapping between YANG models should be done.

E. Aggregation

Aggregation is one of the most important parts of the system. If we did not implement any aggregation application, the benefits of the system would only limit to the normalization of the data. Although normalizing the data brings great advantages already mentioned, such as the ease of avoiding invalid samples thus eliminating a possible pre-processing step or the context information provided by standardising the data received from the different sources into models, aggregation brings other very valuable advantages for consumers. Through an aggregation stage, data can be improved so that the consumer can use this enriched information to achieve better results in its objective. From the point of view of the CDS as a consumer, training the model with one feature or another will cause significant variations in the outcome of the models.

In [26], parameters that are not found in Netflowv9 or obtained through Goflow2 are used. Therefore, to obtain these parameters it is necessary to implement a flow aggregation application. The application will use the information of each flow to obtain these statistics. This implies the implementation of a new YANG model that models the related flows of Netflowv9 along with the aggregated parameters. The YANG language facilitates this by means of the "augment" statement. These allow us greater modularity, since for the implementation of the aggregated model we will import the base model, and use the keyword "augment" to extend the fields of the base model.

To perform the aggregations with the Flink DataStream API we will use the "map" transformation, which allows us to take one element and produce another.

Some of the Netflow related variables, such as bytes or packets, contain the string "in" or "out", which indicates the direction of the flow. However, the variables "bytes-out" and "pkts-out" always appear with a value of 0 when obtained by Goflow2. After studying this behaviour, it is observed that Softflowd always separates a communication into unidirectional flows. That is, the communication between hosts A and B is separated into A→B and B→A flows, and is not considered as a single bidirectional flow. This behaviour prevents the correct functioning of the developed aggregation application and does not take advantage of Netflow's capabilities. Therefore, it was decided to implement another aggregation application that allows transforming unidirectional flows into bidirectional flows by means of windows. The Flink API provides us with functions that allow the separation of flows into different partitions via the *keyby* function. The flows will be separated by means of a key formed by the addresses and ports of origin and destination ordered together with the start and end time of the flow. They must be ordered so that two unidirectional flows must go to the same partition.

Subsequently a window is created, each partition will have its own window. For these windows Flink implements time-dependent triggers and triggers dependent on the number of events in the window. A trigger is what will lead the window to be processed. Using a time trigger can be a solution. When the time ends and the window is processed, the number of elements in the window is checked. If there is one element, it is sent without further processing and the flow is assumed to be unidirectional, and thus cannot be aggregated. If there are two events, they are added in a bidirectional flow. However, the operation is not optimal, because if the rate of arrival of events together with the time of the window are high, the amount of partitions created may be too large. To solve this efficiency problem, we implement a trigger itself, which allows to create windows depending on the time and the amount of elements in the window. In this way, if two events arrive at the window, they are processed directly without waiting any timer, and if an event arrives and passes a certain time, it is also processed. In this way we manage to considerably increase the rate of packets that we can aggregate. A schematic of how this aggregator works can be seen in the Figure 3.

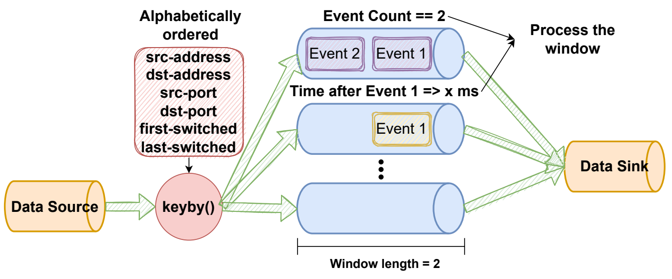

Fig. 3: Unidirectional to Bidirectional Aggregation Application.

F. Output Driver

The output driver will always be the end of the pipeline, working to adapt the normalized and aggregated events to the consumer. YANG-TOOLS library gives us plenty of ease, due to the fact that it already supports the serialization of the events to different schemas (e.g., JSON or XML).

The consumer will be a Cryptomining Detection System that will use Machine Learning. These type of systems normally need an array format to make the predictions, at least with well-known libraries such as Scikit-Learn. Therefore, for the ease of the subsequent transformation into array-like structures, the output will be in Comma Separated Values (CSV). YANG-TOOLS does not support CSV serialization, therefore, a specific function is developed to be able to serialize to CSV for the consumer.

Emphasize again that the purpose of the SDA is to facilitate the ingestion of data from heterogeneous sources by consumers. Therefore, the final goal is to lay the foundations for generalizing this output driver as was done with the input driver case.

G. Consumer

The events processed by the system can be consumed to be stored, to continue processing the data, for anomaly detection, threat classification, among others. In this case, a consumer is used to detect cryptomining traffic. For this purpose, a Python application has been developed. Making use of Scikit-Learnlibraries, a classification model is trained. The results shown in [26] indicate that using *RandomForestClassifier* is a good choice for the traffic we are using. The application reads events from a Kafka topic in a specific CSV format and writes results in another topic if the traffic belongs to cryptomining protocols.

H. Deployment

Both Docker and Kubernetes are open-source applications that allow the orchestration of containerised applications. In fact, Kubernetes can work with Docker, the difference between them is that Docker is designed to package applications on a single node, while Kubernetes is more flexible and makes it easier to handle the scalability of applications.

For the development of the system and the initial deployment Docker was used. Services were launched by using Docker Compose, where different parameters can be specified, such as the image to use or environment variables for the configuration of them. Thus, a single Flink cluster is deployed, consisting of a JobManager and one or more Task Managers (in charge of executing the applications), sharing all the same resources of a JVM (Java Virtual Machine). This results in concurrency problems with YANG-TOOLS library, because the YANG schemas seem to be loaded into memory. To solve this, two solutions are proposed and developed:

1) Flink Application Mode with Kubernetes: Using Application Mode deployment each application runs in a different JVM. With the previous mode we could create several slots in a TaskManager, providing scalability and a failover system. Now this capabilities are provided with Kubernetes, which also provides some degree of abstraction of the Flink clusters, facilitating system orchestration. A Kubernetes operator for Flink developed by Spotify (*flink-on-k8s-operator*) is used, facilitating the deployment and lifecycle management of Flink applications.

2) Apache Flink Statefun: For this solution, a special image of Flink called *Flink StateFun* has been used. This image let us develop applications that communicate via HTTP/gRPC. Therefore, any programming language that allows to deploy a server of this type can be used. So instead of using YANG-TOOLS we can use libraries for other languages, such as *pyang* for Python or *ygot* for Go. In addition, this allows physical separation between the Flink cluster and the different applications or functions, so they can be scaled independently. Figure 4 illustrates the described scheme.

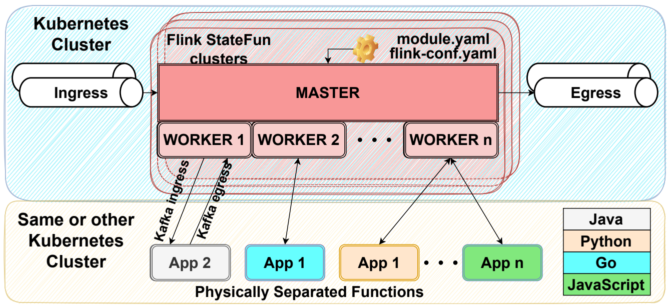

Fig. 4: Scheme of multiple functions implemented in different programming languages using Apache Flink StateFun cluster.

The file *flink-conf.yaml* allows to configure Flink and Statefun properties. The file *module.yaml* is used to specify the location of the HTTP/gRCP endpoints, the source (ingress) and the consumer (egress).

The *flink-on-k8s-operator* does not support this Flink cluster. Therefore, Helm is used to package the applications, adding flexibility and ease of deployment.

Although the first solution is chosen, the second solution is also implemented to add flexibility to the development of the system.

IV. Evaluation

In this section, multiple tests will be carried out to determine the efficiency of the system implemented for both the SDA and the CDS.

A. SDA Evaluation

These tests focus on measuring the efficiency of the SDA for latency and scalability purposes. To measure the latency, the use of functions such as *System.nanoTime()* is not feasible due to the complexity of the applications (use of partitions and windows, topic reading and writing latencies, etc).

Some mechanisms are implemented in the Flink API such as *LatencyMarkers*, but as discussed in [28] they do not reflect the real latency as they bypass user functions where most of the latency is expected. Also, there are components in the system that do not work with the Flink API, such as Goflow2. Due to this, two additional Flink applications have been developed. The two applications are placed at the input and output of the related application or driver to be measured. They take timestamps both when they send and receive the event, together with a hash of specific fields that will allow later, through a script, to relate both timestamps and calculate the latency. This process adds the latency relative to reading and writing from Kafka topics, but it allows a fairer comparison between the SDA components and a more accurate measurement than the above-mentioned methods.

A test with 1250 samples was performed. Figure 5 shows the results of latency obtained for the four applications. In the first figure, it can be seen that for the input-driver the latency increases for each event that arrives, indicating that we have overloaded the driver, introducing a number of events per second higher than the driver's processing capacity. Since Kafka command console has been used to write the events into the Kafka topic, the event rate has not been controlled, so the latency results do not allow a fair comparison between the different applications. However, it is clear that the input-driver has been overloaded, while the aggregation application and the output driver seem to have maintained a stable latency despite having a higher rate, which was measured later. These are clear indicators that in a real system with a high event rate, we may need more parallelization for the input driver to avoid overloading the system.

To avoid the disparity in the input rate and to be able to make a fair comparison in terms of latency measurement, another application is implemented to allow us to manipulate the rate of events per second that are written to the topic. In addition, a very low rate is chosen (one event per second) to ensure that none of the applications are overloaded. In Figure 5, the figure below reveals that the aggregator and output driver have very low latency, followed by the input driver, which is now not overloaded. Aggregation of individual flows to bidirectional (i.e., *netflow2bidi*) seems to perform worst.

This is due to the event rate, since the minimum time for the application to process a window is when at least two samples arrive (one second).

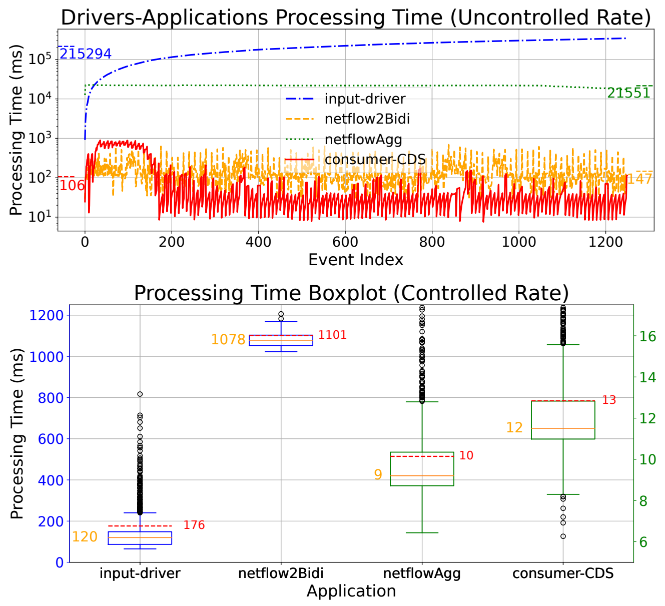

Fig. 5: Latency for uncontrolled and controlled event rate. In the boxplot, red dashed marks are the mean while the orange ones are the median. The left y-axis represents the values for input-driver and netflow2Bidi while the right one represents the values for the remaining drivers. Both axes in milliseconds.

As seen in Figure 5, it is clear that the system may need to be scaled up at some point. Since the deployment of the system is done through Kubernetes, a Helm chart is developed that allows to launch the complete pipeline. Therefore, multiple pipelines (i.e., Goflow2 and all the Flink applications) are deployed with this Helm chart. In the Kubernetes cluster, a Prometheus service is installed so that we can collect metrics related to the system's memory and CPU. The experiment consists of launching seven complete pipelines every five minutes, so that we can see the variations in memory and CPU during each deployment. Collected metrics are plotted in Figure 6.

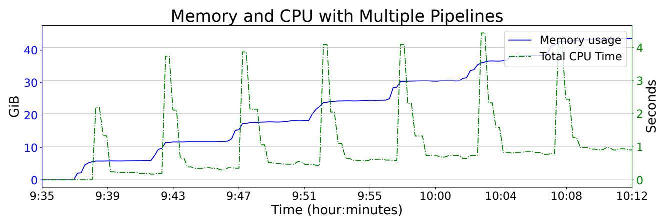

Fig. 6: Memory and total CPU time during the deployment of seven full Netflowv9 pipelines. Left y-axis represents values for memory usage while right y-axis represents values for total CPU time.

The figure shows the sum of memory and CPU consumed by each pipeline. We can observe seven peaks of CPU usage at the times when each pipeline is deployed. After that, the CPU increase is minimal and behaves in a linear fashion. In terms of memory, the increase is similar with each deployment, indicating also a linear behaviour. As a result, it is shown that the system can scale correctly without any CPU or memory issues.

B. CDS Evaluation

In [26], the authors make a comparison between the aggregated features obtained from Netflowv9 and the ones obtained with TSTAT. Tests are also performed with the combination of both features. The best results are obtained with TSTAT. However, no tests are performed to see whether this aggregation of Netflowv9 features has a significant impact on the prediction results compared to not using no aggregated Netflowv9 features.

Features	F1	Precision	Recall	AUC_ROC
netflow	0.9843	0.9842	0.9844	0.9922
netflow_inbound	0.9845	0.9805	0.9885	0.9942
netflow_inoutbound	0.9874	0.9862	0.9885	0.9943

netflow	**netflow_inbound**	**netflow_inoutbound**
$\begin{bmatrix} 16055097 & 70 \\ 69 & 4355 \end{bmatrix}$	$\begin{bmatrix} 16055080 & 87 \\ 51 & 4373 \end{bmatrix}$	$\begin{bmatrix} 16055106 & 61 \\ 51 & 4373 \end{bmatrix}$

TABLE I: Comparison of results using Netflow, Netflow and aggregated, and data into consideration the bidirectionality of the flows. The confusion matrix is shown below.

Therefore, tests are performed with Netflowv9 features, and compared with the features aggregated by the SDA. First by taking into account the Netflow aggregator application. The application for unidirectional to bidirectional flows will also be added later to observe this difference as well.

In [26], hyper-parameter optimisation is already being used by separating the dataset into training, validation and test. In this case, it is not necessary to separate a set for validation, as hyper-parameters will not be optimised, since the aim is not to find the best model, but to demonstrate the improvement introduced by the aggregations performed with the SDA prior to the CDS. Therefore, the dataset will be composed by training and testing (16 million samples each) with the same distribution between them.

Table I shows the results obtained in the form of different classification metrics for a Random Forest model trained with the aforementioned datasets.

For the case of Netflowv9 with the aggregated features, it can be seen that the results improve for the F1, recall and AUC_ROC metrics, but not for accuracy, since it can be observed in the confusion matrix that the false positives have increased although the false negatives have decreased considerably. In the case where we also use the aggregator from unidirectional to bidirectional flows, we obtain an improvement for all metrics, thus reducing the number of false negatives and false positives.

As a result, the efficiency of the SDA for improving the performance of a specific detection system such as the one tested is demonstrated, in addition to the benefits of SDA mentioned above.

V. Conclusions and Future Work

A. Conclusions

This paper presents an data normalization and aggregation system based on YANG models.

Netflow version 9 is used to demonstrate the methodology of implementing a complete pipeline within the system. The impact of a system that allow normalizing and aggregating information in a simple way becomes evident, especially nowadays when the use of Artificial Intelligence is on the rise and used to solve a large number of problems. The effectiveness and benefits of using aggregated variables for cryptomining detection results are also demonstrated. In addition, tests are performed on the system to demonstrate the robustness and scalability that it allows, which will allow great flexibility for the integration of heterogeneous sources.

B. Future Work

For the analysis of threat detection, it would be very interesting to combine several sources. It has already been proven in other studies that the combination of traffic with HPC can increase the effectiveness of cryptomining detection systems.

On the other hand, both this mentioned combination and the integration of a source is currently a methodology. This work lays the foundations for the creation of a tool capable of integrating any source automatically and easily. The final goal is to facilitate the adaptation of any source, regardless of the data encoding format and the transport protocol used, to any consumer, simply by modelling source and consumer with YANG, also allowing the combination and aggregation of heterogeneous sources.

Acknowledgment

The research leading to these results received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement no. 952644 (FISHY) and no. 883335 (PALANTIR). The paper reflects only the author view. The Commission is not responsible for any use that may be made of the information it contains.

References

[1] D. Zhang, M. He, X. Wang, and M. Tehranipoor, "Dynamically obfuscated scan for protecting ips against scan-based attacks throughout supply chain," in *2017 IEEE 35th VLSI Test Symposium (VTS)*, pp. 1–6, 2017.

[2] W. J. Heinbockel, E. R. Laderman, and g. J. Serrao, "Supply chain attacks and resiliency mitigations," in *Guidance for System Security Engineers*, 2017.

[3] H. H2020, "Fishy, a coordinated framework for cyber resilient supply chain systems."

[4] A. Pastor, A. Mozo, D. R. Lopez, J. Folgueira, and A. Kapodistria, "The mouseworld, a security traffic analysis lab based on nfv/sdn," in *Proceedings of the 13th International Conference on Availability, Reliability and Security*, ARES 2018, (New York, NY, USA), Association for Computing Machinery, 2018.

[5] FISHY, "D6.2 it-1 fishy release validated," in *A coordinated framework for cyber resilient supply chain systems over complex ICT infrastructures*, 2022.

[6] H. H2020, "Palantir, practical autonomous cyberhealth for resilient smes & microenterprises."

[7] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, RFC Editor, 10 2004.

[8] D. Geneiatakis, R. Kozik, M. Pawlicki, and M. Chora, "Cost-sensitive distributed machine learning for netflow-based botnet activity detection," *Security and Communication Networks*, 2018.

[9] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[10] M. Bjorklund, "Yang - a data modeling language for the network configuration protocol (netconf)," RFC 6020, RFC Editor, 10 2010.

[11] YangModels, "yang vendor cisco xr 622," Sep 2020. [Online; posted 2017].

[12] D. Gonzlez-Snchez, I. D. Martinez-Casanueva, A. Pastor, L. B. Triana, C. P. M. Zamarro, A. A. M. Sancho, D. F. Cambronero, and D. Lopez, "Model-driven network monitoring using netflow applied to threat detection," in *2022 IEEE 8th International Conference on Network Softwarization (NetSoft)*, pp. 450–455, 2022.

[13] P. O'Kane, S. Sezer, and D. Carlin, "Evolution of ransomware," *IET Networks*, vol. 7, no. 5, pp. 321–327, 2018.

[14] C. Symantec, "Internet security threat report 2019," 2019.

[15] M. Caprolu, S. Raponi, G. Oligeri, and R. Di Pietro, "Cryptomining makes noise: Detecting cryptojacking via machine learning," *Computer Communications*, vol. 171, pp. 126–139, 2021.

[16] A. D. Yulianto, P. Sukarno, A. A. Warrdana, and M. A. Makky, "Mitigation of cryptojacking attacks using taint analysis," in *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, pp. 234–238, 2019.

[17] D. Tanana, "Behavior-based detection of cryptojacking malware," in *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology (USBEREIT)*, pp. 0543–0545, 2020.

[18] G. Mani, V. Pasumarti, B. Bhargava, F. T. Vora, J. MacDonald, J. King, and J. Kobes, "Decrypto pro: Deep learning based cryptomining malware detection using performance counters," in *2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*, pp. 109–118, 2020.

[19] R. Tahir, S. Durrani, F. Ahmed, H. Saeed, F. Zaffar, and S. Ilyas, "The browsers strike back: Countering cryptojacking and parasitic miners on the web," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 703–711, 2019.

[20] I. Petrov, L. Invernizzi, and E. Bursztein, "Coinpolice:detecting hidden cryptojacking attacks with neural networks," 2020.

[21] N. Helio N. Cunha, L. Martin Andreoni, F. Natalia C., and M. Diogo M. F., "Minecap: super incremental learning for detecting and blocking cryptocurrency mining on software-defined networking," *Annals of Telecommunications*, 2020.

[22] R. Ning, C. Wang, C. Xin, J. Li, L. Zhu, and H. Wu, "Capjack: Capture in-browser crypto-jacking by deep capsule network through behavioral analysis," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 1873–1881, 2019.

[23] C. Kelton, A. Balasubramanian, R. Raghavendra, and M. Srivatsa, "Browser-based deep behavioral detection of web cryptomining with coinspy," in *Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2020*, pp. 1–12, 2020.

[24] Z. Li, J. Hou, H. Wang, C. Wang, C. Kang, and P. Fu, "Ethereum behavior analysis with netflow data," in *2019 20th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 1–6, 2019.

[25] J. Z. i. Muoz, J. Surez-Varela, and P. Barlet-Ros, "Detecting cryptocurrency miners with netflow/ipfix network measurements," in *2019 IEEE International Symposium on Measurements & Networking (M&N)*, pp. 1–6, 2019.

[26] A. Pastor, A. Mozo, S. Vakaruk, D. Canavese, D. R. Lpez, L. Regano, S. Gmez-Canaval, and A. Lioy, "Detection of encrypted cryptomining malware connections with machine and deep learning," *IEEE Access*, vol. 8, pp. 158036–158055, 2020.

[27] Cisco, "Netflow version 9 flow-record format," May 2011.

[28] J. Qin and N. Kruber, "Getting into low-latency gears with apache flink - part one." [Available; 18-May-2022].