CryingJackpot: Network Flows and Performance Counters against Cryptojacking

Gilberto Gomes[1,2] Luis Dias[1,2] Miguel Correia[2]
[1]CINAMIL, Academia Militar, Instituto Universitário Militar – Portugal
[2]INESC-ID, Instituto Superior Técnico, Universidade de Lisboa – Portugal
gomes.g@exercito.pt dias.lfxcm@exercito.pt miguel.p.correia@tecnico.ulisboa.pt

***Abstract*—Cryptojacking, the appropriation of users' computational resources without their knowledge or consent to obtain cryptocurrencies, is a widespread attack, relatively easy to implement and hard to detect. Either browser-based or binary, cryptojacking lacks robust and reliable detection solutions. This paper presents a hybrid approach to detect cryptojacking where no previous knowledge about the attacks or training data is needed. Our *Cryptojacking Intrusion Detection Approach*, CRYINGJACKPOT, extracts and combines flow and performance counter-based features, aggregating hosts with similar behavior by using unsupervised machine learning algorithms. We evaluate CRYINGJACKPOT experimentally with both an artificial and a hybrid dataset, achieving F1-scores up to 97%.**

***Index Terms*—intrusion detection, clustering, cryptojacking, network flows, performance counters, security analytics**

I. INTRODUCTION

Bitcoin's presentation in 2009 made headlines, being the starting point of what would be the popularization of crypto-mining and with this the dissemination of a new attack, *cryptojacking*. This recent threat is defined as the illicit appropriation of users' computational power without their knowledge or consent to mine cryptocurrencies. This attack can be performed through two major attack vectors: browser-based scripts and binary malware (executables) [1], [2].

Bitcoin created a new paradigm: an alternative to traditional electronic payment systems based on intermediaries (e.g., banks). Relying on such intermediaries is susceptible to problems like potentially misplacing trust, additional costs (money transfer fees), and operation delays [3]. Bitcoin, a digital representation of assets, alongside with *blockchain*, an append-only universally accessible ledger of transactions, came to replace those transaction intermediaries using a decentralized network of nodes [4]. For a new group of transactions, intensive computing power must be used to validate, add, and distribute a new block with those transactions through the blockchain network. The solution to a cryptographic problem must be found by computing a so-called Proof of Work (PoW). The *miner* that obtains the PoW and manages to append the block to the blockchain is rewarded with new coins [3], [5].

Nevertheless, malicious miners started to target users' computational resources for their own profit, a profit that is proportional to the number of victims and the resources used [1], [2], [6], [7]. Although recent, illicit cryptocurrency mining represents an immediate and long-term threat to both organizations and end-users. Security practices that allow the initial exploit could lead to physical damage, performance and productivity impact, increase in maintenance costs, and open security holes for more dangerous attacks. Also, it is important to notice the incoming revenue for the non-stop growing economy that is behind malicious activities. For example, numerous top-visited web sites, including YouTube [8], Showtime, and The Pirate Bay [9], have been criticized for permitting cryptojacking attacks against their users. At a different level, an European airport system was found to be running Monero (XMR) mining malware [10]. A recent survey found 65,000 domains conducting cryptojacking activities among the Alexa Top 1M domains [2].

Cryptojacking is one of several new threats that changed the prospect of network security. An effort is being made to develop more robust and efficient Intrusion Detection Systems (IDSs), capable of detecting attacks of several types. The most common intrusion detection approaches are: *Misuse / signature detection*, which uses previous knowledge of attacks to build signatures in a matching process that flags behaviors that share similarities with the existing signatures. It usually generates a low number of false positives, but it fails in detecting previously unseen attacks; *Anomaly detection*, where the idea is to identify patterns that deviate from normal behavior, appearing as an appealing solution because of its ability to detect previously unseen attacks. However, it often leads to *high false alarm rates* [16], [17].

Both approaches share a common disadvantage: they need prior knowledge, either of existing attacks, or of what normal behavior is. Both types of prior knowledge are hard to obtain.

IDSs can also be classified, in terms of data source, as (1) *host-based*, (2) *network-based*, or (3) *hybrid*. The first (1) take into account host characteristics for detection, the second (2) monitors network traffic, and the third (3) – CRYINGJACKPOT's approach – combines both data sources [18].

Machine Learning (ML) techniques, where statistical models are built with the knowledge extracted from data through a computational process, plays a core role in many IDSs [17]. These techniques can be divided into two major categories: supervised and unsupervised methods. In *supervised methods*, there is labeled training data that is used by an algorithm to build a model. One of the most common supervised methods is classification, whereas the model is used to label never seen data points and determine which class they belong to. Most of the ML methods employed in misuse detection systems are supervised [17]. *Creating an IDS based on these approaches is challenging*, mainly due to the availability of well-suited data

978-1-7281-8326-8/20/$31.00

TABLE I
CRYPTOJACKING DETECTION SOLUTIONS (ADAPTED FROM [11])

Paper	Features	Classifier
Seismic [12]	WASM opcode counts	Hand-crafted rules
RAPID [13]	Resource-related API calls and consumption	Support-Vector Machine
MiningHunter [13]	Fingerprints	Hand-crafted rules
Muñoz et al. [14]	Flow-based	Support-Vector Machine, C4.5, CART, Naïve Bayes
Outguard [15]	Signatures and behavioral features of aggressive miners	Support-Vector Machine
CoinPolice [11]	Throttling-independent behavior timeseries	Convolutional neural network
CRYINGJACKPOT (our paper)	Flow-based and performance counters	K-means, Agglomerative, DBSCAN, Ensemble

[16], [17], [19]–[23].

Unsupervised learning – the set of techniques we use – is compelling for intrusion detection because, by definition, it does not need labeled data as input, nor signatures, or training. Among its several advantages, this detection methodology allows detecting new attacks, which is an improvement over the conventional approaches previously described by finding patterns, structures, or knowledge in unlabelled data [16], [17]. Within unsupervised learning, *clustering algorithms* are a popular choice for anomaly detection, aiming to find malicious information within networks without prior knowledge. Consider that a *feature* is a measurable characteristic of the relevant phenomenon, e.g., the number of bytes received by an interface in port TCP/80. For a clustering algorithm, an entity (e.g., a machine or a user) is characterized by a vector of features, and these entities are aggregated in clusters based on their features similarity. The resulting groups or clusters can be analyzed and flagged as malicious or not using manual analysis, outlier detection, thresholds, or heuristics like considering small clusters suspicious [17], [19], [21]. These approaches have been investigated and implemented by several authors for network security applications [19]–[23].

Concerning the detection of cryptojacking, there is an increasing corpus of recent literature (Table I). These works have several limitations. Most of them address exclusively browser-based mining [11]–[13], [15], [24]–[26]. Even though this malicious approach represents the majority of all cryptojacking activity [27], the classical malware-related challenges, persistence and obfuscation, make of this an important threat that needs attention. Also, works that depend on well-known scripts, tags, or signatures [12], [28], are susceptible to the techniques that evade these detection mechanisms. The approaches that use the measurements of the host's resources consumption [13], [15], [29], cannot guarantee detection when there are hosts with legitimate heavy workloads.

We present CRYINGJACKPOT, a new hybrid IDS based on unsupervised learning for cryptojacking detection. Our approach performs *clustering ensemble* with different unsupervised ML algorithms, using both host-based and network-based features. Using the operating system performance counters that represent the system state or activity (host-based) and network flows (network-based), we aim to develop a robust and reliable approach that does not need previous knowledge or training data for the detection of browser-based as well as binary cryptojacking malware. Combining the dual nature of our features, our approach is able to detect cryptojacking under intensive loads and intensive network traffic, resistant to different throttling degrees (consumption percentages), code obfuscation, and the use of proxies.

We first tested CRYINGJACKPOT with a public dataset (CSE-CIC-IDS2018 [30]) and flow-based features with good results, even though this dataset includes many attacks with a more noticeable behavior. We also created a hybrid dataset to analyze network flows and performance counters simultaneously. CRYINGJACKPOT obtained values for an F1-score of 82% for the public dataset and an F1-Score of 97% for the hybrid dataset in several experiments designed to cover a large number of real-world scenarios.

II. BACKGROUND

This section starts by explaining cryptojacking attacks, provides some background on the use of clustering for intrusion detection, and explains how we chose the mechanism to pinpoint attacks (outlier detection).

A. Cryptojacking

To deal with some of the issues with blockchain technology, such as network consensus, operations' irreversibility, among others, the notion of PoW was introduced. This algorithm turns the task of adding new blocks to the chain highly expensive in terms of computing power [5]. More specifically, the task of adding a new block is based on the solution of a cryptographic problem, where the miner has to take a block of transactions and compute the hash value of that block and a nonce [5]. This task – designated *mining* – is done repeatedly until a specific number of prefixing zeros in the hash value is achieved. The first miner able to solve this problem is rewarded with new coins for the computational effort used. The new block is disseminated through the nodes of the network that add it to their local instance of the blockchain [3], [5].

Due to technological developments, not long after the appearance of Bitcoin, the practice of mining this cryptocurrency by end-user machines became infeasible. This paradigm would not change until 2014 with the appearance of a new cryptocurrency, *Monero (XMR)*. XMR uses a different PoW algorithm called RandomX, that makes the mining tasks equally efficient on the Central Processing Unit (CPU) and the Graphics Processing Unit (GPU), while making it less effective with Application-Specific Integrated Circuits (ASICs)[1]. Being ideal for mining by end-users, there was an increase in mining scripts development. Some web applications started using web mining scripts legitimately as a revenue alternative, e.g., for suppressing adds.

[1]Monero's PoW algorithm was updated from CryptoNight to RandomX in October of 2019, with the objective of being even more ASICs resistant [31].

However, others started running then illegitimately, without user knowledge or consent. These scripts usually use CPU computational power.

In 2017, *Coinhive*, a major browser service provider made a big entrance by providing an Application Programming Interface (API) so that developers could implement browser mining on their websites. Coinhive was using legitimate politics to establish dominance over the remaining similar services. This trend skyrocketed as Coinhive's scripts started to dominate, increasing browser-based mining, and in consequence, increasing cryptojacking [1]. Eventually Coinhive stopped its operation due to the problems it was raising.

Cryptojacking is in the attacker's perspective easier to deploy and entails fewer risks than other attacks, which calls the attention for this type of threat. Even though for browser-based mining the exploit behavior ceases whenever the victim closes the browser [1], crypto-mining malware share the persistence and obfuscation characteristics of traditional malware [7].

In a *mining pool* several individual miners come together to divide the computational effort and increase their change of satisfying the PoW before its competitors. The mining server has to be efficient when distributing tasks to optimize the profit, a scenario where latency is key.

Moreover, the *Stratum* protocol is a line-based protocol built over Transmission Control Protocol (TCP) specially designed for crypto-mining communication. It is used by most of the mining pools to communicate with miners or pool servers. Besides being easy to implement, extensible, and compatible with most platforms, the major advantage of Stratum against HTTP is that servers can drive the load by themselves and send broadcast messages to miners without any long-polling, load balancing or packet storms [2], [14]. A simple way to understand the need for a dedicated protocol arises from understanding the HTTP protocol's nature. The client asks the server for specific data, a request that the server can not predict. However, when mining, the information that will be exchanged is already known, by what the Stratum protocol came up to optimize these iterations.

B. Clustering and intrusion detection

Clustering is a set of techniques for finding patterns in high-dimensional unlabeled data [16]. The general idea is that similar objects are grouped by measuring the distance between them. A clustering algorithm takes as input a set of entities and returns a set of subsets (or clusters) of these entities. However, this process does not tell us anything about the nature of the individual subsets/clusters.

For intrusion or anomaly detection, many works leverage the idea that big clusters represent normal behavior and that the outliers (e.g., small clusters of entities or noise) may correspond to anomalous behavior [17]. Outliers are those points in a dataset that are highly unlikely to occur for a model of the data. Outliers may be considered to be those entities in clusters with a single entity [22], [32], or those that are further from all the other data points than a given a threshold [33], [34].

In this work, we pursue the idea of finding outliers as one entity clusters, for which the choice of the algorithms represents an important design step. Due to the lack of literature for the detection of cryptojacking with unsupervised methods, we decided to provide our system with redundancy by choosing three kinds of algorithms and the implementation of an ensemble technique influenced by previous works [23], [35], [36].

The clustering algorithms we chose can be classified as partition-based, hierarchical, and density-based. *K-means* ($O(nkdi)$), is a popular partition-based algorithm that distributes the given data points X into k clusters, where each data point is more similar to the centroid that defines its cluster than it is to other cluster's centroids. It is important for the implementation step to take into account two key factors: the number of clusters, and the distance metric, being the Euclidean distance generically the most used metric [17].

In *hierarchical clustering (HC)*, algorithms organize data into a hierarchical structure according to the proximity matrix. The results of HC are usually depicted by a binary tree or dendrogram. We chose *Agglomerative hierarchical clustering*, since divisive clustering is an unusual practice due to its computational effort [37]. The algorithm starts by assigning each object to a different cluster, then it computes pairwise distances of n data points and links them according to a linkage function (e.g., minimum distance). The results can be represented by a dendrogram where the root node represents the whole dataset (single cluster), and each leaf node is a data object. The complexity of the algorithm is $O(n^2)$, mainly due to the cost of computing all pairs of distances [23], [38].

DBSCAN groups data points into clusters that have a higher density than a threshold number (MinPts). It performs this task within a window of a specified size defined by the distance to the data point (eps). In other words, DBSCAN creates a new cluster from a data object by absorbing all objects in its neighborhood, given a user-specified density threshold [17], [39]. DBSCAN is a complex algorithm ($O(nlog(n))$), associated with high execution times. All clustering algorithms were set to use Euclidean distance. Moreover, motivated by the capabilities of each of these algorithms, we support our choice with the good results obtained in several works [21]–[23], [39]–[41].

III. The CryingJackpot Approach

The CryingJackpot approach aims to automatically identify outliers, i.e., entities – hosts – targeted by cryptojacking, by extracting, combining and processing a set of features. These features are obtained from *host-based performance counters* [42] and *network flows* [43], [44]. In relation to counters, in the paper we consider only counters provided by Microsoft Windows, but the approach is generic. The use of these host-based features is motivated by the nature of the attack and the associated resource consumption. Exploring resource consumption is an approach already used in previous works [13], [28], [29]. The latter, network flows, is motivated by their ability to allow detecting attacks immersed in large volumes of data [14], [21], [23], [45].

Figure 1 represents our approach that has four major steps: data collection (1), data processing (2), clustering (3), and evaluation (4), each of them presented in-depth in the following sections. The first step, concerns feature engineering and the

possible methods to obtain the data; the second step addresses all the processing done to the raw data (features extraction and normalization); the third, clustering, is the core our approach, involving parameter inference and the clustering algorithms. Finally, step four concerns the evaluation of the results.

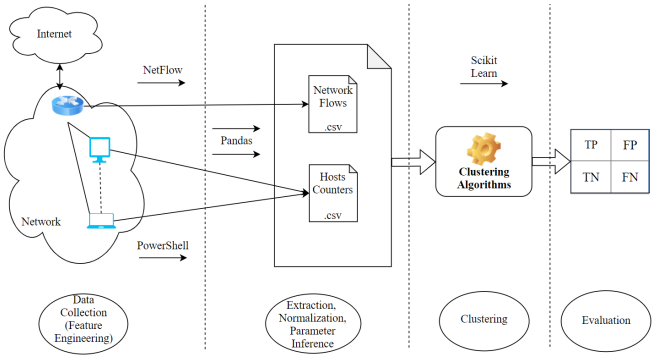

Fig. 1. CRYINGJACKPOT's general overview

A. Data collection

As previously mentioned, CRYINGJACKPOT obtains features from two different data sources, performance counters and network flows.

We lean towards *performance counters indicators* for Windows machines, to explore cryptojacking behavior as a CPU-heavy attack. For Windows machines, several performance counters are available for monitoring purposes [42], from which we chose the indicators that best reflected cryptojacking manifestation. We end up with up to 123 performance counters, a set of indicators that described *CPU, memory, network usage, and running processes within a host.* Some examples:

```
\Processor(*)\% Processor Time
\Memory\Available Bytes
\Network Interface(*)\Bytes Sent/sec
```

However, as they are, these indicators can not be employed as features. For a given time window and a given host, there will be several measurements regarding these metrics. Therefore, we had to combine them to generate a set of features that the clustering step could process. We studied the use of *average value, max value and the standard deviation.* We also combined these statistical metrics, but the best results were obtained by *using exclusively the average value for each performance counter.* From the initial set of 123 performance counters, we then created 123 features, each one as the average value of the respective indicator for a given time window. For example `CPUTime_avg` is a feature that, for a specific host and time window, gives the average value of the `\Processor(*)\% Processor Time` counter. Moreover, we studied the contribution of each one of the 123 initial features using heat maps, and by discarding redundant features, we *ended up with 61 performance counters-based features.*

Regarding *flow-based features* our choices were influenced by works that addressed the Stratum protocol and cryptojacking detection [14], [25], [28]. Some particularities of Stratum were fundamental for our choice of features (cf. Section II). As stated by Muñoz et al. [14], Stratum shows an asymmetry in connections, i.e., the server sends more data than the client/victim. This effect can be observed using features about inbound/outbound flows, e.g., inbound/outbound packet counts. Moreover, Stratum generates flows that take 5 to 30 minutes to finish. We decided to create features related with 80/HTTP and 443/HTTPS ports, as Stratum utilizes these ports.

CRYINGJACKPOT explores flows from both source and destination perspectives, where hosts are used as aggregation keys, either as source IP (SrcIP) or as destination IP (DstIP).

From a *source standpoint*, CRYINGJACKPOT starts by extracting *2 fixed features* that explore incoming data patterns: (1) `SrcBytesPktBwd`, number of bytes per packet received as SrcIP; (2) `SrcMeanBytesBwd`, average number of bytes received as SrcIP.

We then implemented a technique that greatly improved our results. We extracted a set of *11 features* from flows whose destination IPs had been, for a given time window, contacted only by one host/source IP, i.e, a rare destination. This decision aimed to explore connections made with mining pools' exclusive domains. Considering that these rare destination-related features are tagged with `_List`. For this reason, we used additional tags to better understand these features. For example, let's consider the `Src_PSH_Count_List` feature for further explanation. The initial tag indicates a feature extracted from flows were the network hosts represent the source IP. `PSH_Count` describes the feature as being the count of TCP PSH (push) flags during a given time window. This feature was obtained by observing the traffic between the victim and the malicious domain. The final tag is an indication that this feature was obtained from flows where the destination IP has a unique contact. Additionally, CRYINGJACKPOT also defines 5 port-related features, regarding HTTP/80 and HTTPS/443, using in this case, source ports as aggregation keys (Table II). Similar to the evaluation performed for performance counters, we also weighed our initial set of features, having *end up with 37 flow-based features.*

CRYINGJACKPOT is similar from a *destination standpoint*, but different port-related features where used (Table II).

B. Extraction, normalization, and parameter inference

CRYINGJACKPOT can process data on different predefined time windows, of duration $\mathcal{W} = \{w_1, w_2, ..., w_n\}$, e.g., 10 minutes, 1 hour, 1 day. This approach performs clustering for every window of duration $w_i \in \mathcal{W}$ for the periods of data available. For every time window of duration $w_i \in \mathcal{W}$, features are extracted from all flows and counters regarding the time window being analyzed. Following the rationale in Section III-A.

This step results in two *vectors of features* for each host, one containing the features obtained from the performance counters and the second with features from network flows.

For each time window $w_i \in \mathcal{W}$, features have to be *normalized* before the clustering step. Normalization is performed by scaling each feature to a common range, and it is important not to lose information during this process. Therefore, we chose the min-max scaling method described in Equation (1) and (2). This method transforms all features into the range [0,1], ensuring that features will contribute proportionally during the clustering step by using Euclidean distance as distance metric.

$$X_{standard} = (X - X_{min})/(X_{max} - X_{min}) \quad (1)$$

TABLE II
FEATURES USED IN CRYINGJACKPOT

Flow-based Features		Performance Counter-based Features		
# bytes per packet received as [SrcIP/DstIP]		cpu-% cpu time	process-[I/O] write bytes/sec	memory-pages faults/sec
average bytes received as [SrcIP/DstIP]		cpu-% user time	process-[I/O] data bytes/sec	memory-available bytes
# stablished sessions with a [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-% priviledge time	process-max utilization of the virtual space bytes	memory-committed bytes
# packets foward to [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-% idle time	process-max utilization of the paging file bytes	memory-commit limit
# packets received from [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-% C1 time	process-max utilization of the work set	memory-write copies/sec
flows' average duration with [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-% C2 time	network interface-total bytes/sec	memory-transition faults/sec
# PSH flags with [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-C1 transitions/sec	network interface-packets/sec	memory-cache faults/sec
# SYN flags with [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		cpu-C2 transitions/sec	network interface-packets received/sec	memory-pages input/sec
# FIN flags with [DstIP/SrcIP] having an unique contact as [SrcIP/DstIP]		process-% cpu time	network interface-packets sent/sec	memory-page reads/sec
# bytes per packet forward by [SrcIP/DstIP] to [DstIP/SrcIP] having an unique contact		process-% user time	network interface-bandwidth	memory-pages output/sec
# bytes per packet received by [SrcIP/DstIP] to [DstIP/SrcIP] having an unique contact		process-% priviledge time	network interface-bytes received/sec	memory-pool paged bytes
packets inbound/outbound ratio with [DstIP/SrcIP] having an unique contact		process-working set	network interface-unicast packets received/sec	memory-pool nonpaged bytes
bytes inbound/outbound ratio with [DstIP/SrcIP] having an unique contact		process- virtual bytes	network interface-non-unicast packets received/sec	memory-pages writes/sec
		process-page faults/sec	network interface-unknown packets received	memory-pool paged allocs
		process-page file bytes	network interface-bytes sent/sec	memory-pool nonpaged allocs
Src View —— Port-related —— Dst View		process-thread count	network interface-unicast packets sent/sec	memory-max use of cache's bytes
# of bytes per packet received in port x	# of bytes per packet sent to port x	process-pool paged bytes	network interface-packets não-unicast sent/sec	memory-pool paged resident bytes
# of packets received from port x	in / out packets sent to port x	process-pool nonpaged bytes	network interface-average rsc packet size for tcp	memory-total bytes in system's code
# of bytes received from port x	in / out bytes outbound sent to port x	process-[I/O] read operations/sec	network interface-rsc active connection for tcp	memory-total bytes in system's driver
# of packet sent to port x	# of packets received from port x	process-[I/O] write operations/sec	memory-pages/sec	memory-bytes in system's cache
# of bytes sent to port x	# of packets sent to port x	process-[I/O] read bytes/sec	memory-cache's bytes	memory-% bytes used
	# of bytes sent to port x			memory-kbytes available

$$X_{normalized} = X_{standard} \times (max - min) + min \quad (2)$$

Moving forward to the *clustering step*, a critical decision is about the clustering *algorithms' initialization parameters.* These parameters dictate the obtained results in what is known as a try and error process, because there is not a specific set of prior parameters that are known to generate the ideal results.

In relation to the parameters of *K-means* and *Agglomerative*, both assign a set of objects into K clusters, so K must be defined. The best K value might be found through brute force, a method that is unfeasible in practice, due to the computational effort required. For this, we used two algorithms, *elbow algorithm* and the *silhouette algorithm*, to find the value of K. The *elbow algorithm* is widely employed, and has obtained good results in several works [22], [23]. This method looks at the obtained score explained as a function of the number of clusters. The score is usually obtained using the sum of square errors (SSE) between the samples and the correspondent centroid. This method is based on the idea that one should choose a number of clusters so that adding another cluster does not improve the result in a significant way. This criterion has the downside that this turning point may be unclear [46]. Therefore, we also implemented the *silhouette algorithm*, which uses two factors, cohesion and separation. The silhouette coefficients are obtained by comparing the similarity between the sample and the respective cluster (cohesion), and the similarity with other clusters (separation). This coefficient is in the range $[-1, 1]$ with 1 meaning a high similarity between object and cluster [46].

Regarding *DBSCAN*, this requires that the density in a neighborhood of a given object must be high enough to have a cluster assigned [37]. We have to choose the number of samples in a neighborhood for a point to be considered as a core point ($min\_samples$) and the maximum distance between two samples for one to be considered as in the neighborhood of the other (eps) [47]. eps can be chosen as the most pronounced change for when the distances between each entity and its neighbors are calculated and sorted [48]. All three clustering algorithms can be set to use Euclidean distance.

C. Clustering and evaluation

After the features extraction and normalization, the vectors of features are provided as input to the clustering algorithms. The idea is to group machines with similar behavior based on the set of fixed features. CRYINGJACKPOT is able to execute this task by using the host-related features independently from the flow-based ones, with the main goal being the use of both sets simultaneously. This is an important design step because it provides our system of flexibility for further implementations. It can, in future applications, make use of alternative host-based indicators, such as alternative operating systems.

The result of the clustering step is a set of clusters with hosts for each time window of duration $w_i \in \mathcal{W}$. Each host appears in a cluster for each time window.

We use three algorithms – K-means, Agglomerative, and DBSCAN – based on three clustering strategies – partition, hierarchical, and density-based. Furthermore, our goal of developing a robust and reliable system pushed us to implement a *clustering ensemble*, were the results of the clustering algorithms are combined for the overall improvement of our approach results. This is done by voting the results of each algorithm, i.e., we follow the rule of the 2 out of 3. More specifically, a host is considered to be a true positive (Section V-A), if flagged as an outlier for at least two algorithms. The same applies for false positives.

Several methods can be used for the classification. If needed, a manual inspection can be performed by a security analyst, starting with the smallest clusters. However, to automate the identification of anomalies, we consider that *an outlier is an entity that is isolated in a cluster*. The disadvantage of this approach is that for several simultaneous victims, our detection capabilities shows worst results as victims share clusters.

IV. HYBRID DATASET

Due to our approach's nature, we were not able to find out a dataset that met our needs of having both network flows and performance counters combined with cryptojacking attacks. Therefore, we created a hybrid dataset, where real-world data was collected to then be used for our needs.

As stated by Sharafaldin et al. [49], there is a set of considerations to take into account when using or creating a dataset. We took those observations into account, as we needed to obtain a set of specific data to our detection tests. This task was conducted in two stages: *benign data collection (B-Profiles)*

and malicious data collection (M-Profiles). Regarding the B-profiles, an ideal dataset would consist of all the information associated to a real-world network without any filtering. This idea has several drawbacks: the access to the entire network is not always possible; there are privacy or anonymity-related issues, among others. An alternative is the artificial generation of traffic, for both profiles, a technique still to be perfected [49].

Our approach to creating the dataset is described as follows. We decided to obtain a small set of real-world data, with no filtering. We extracted all the information, regarding network traffic (packets) and performance counters, for a set of users whose fingerprints reflected the current real-world computational trends. This initial set of data was then used to replicate the remaining network structure (Section IV-B) employing an approach that aimed to ensure that the data approximated what would be a real-world dataset. The M-profile (Section IV-A) data was obtained with a similar approach, using real malware samples combined with users' real traffic. We considered having obtained an up to date dataset, not constrained by anonymity issues, where we have access to all the pertinent information we needed to obtain reliable results for our detection approach.

A. Malicious samples collection

Finding cryptojacking samples was one of the most important tasks as we wanted to guarantee that only active malicious samples were gathered, and that we had covered as many possible variants of the attack. We aimed to find browser-based scripts as well as binary malware.

Regarding *browser-based cryptojacking*, we used a source code search engine (*https://publicwww.com/*) to look for specific tags that identified embedded code from mining providers scripts. Despite the shutdown of *Coinhive*, that provided a script for Monero mining, in 2019, a huge amount of top domains during our search had these mining scripts in their source code, however, they were deactivated. Nevertheless, since the Coinhive shutdown, alternative providers emerged [50], e.g., *https://minero.cc/* and *https://www.coinimp.com/*.

TABLE III
BROWSER-BASED CRYPTOJACKING SAMPLES

Search Tag	Domain	Mining Script
minero.cc	tibumpiscinas.com.br	minero.cc/lib/minero.min.js
minero.cc	naftkala.com	minero.cc/lib/minero.min.js
hostingcloud	serieshdpormega.com	hostingcloud.racing/8vVW.js
hostingcloud	vstars.pl	hostingcloud.racing/BBlJ.js
webminepool	oslafo.com	webminepool.com/lib/base.js

From our search we selected 5 browser-based samples that we proved to be executing mining tasks in the background. Table III summarizes information about the samples and Figure 2 shows an extract of source code. These samples also work with different throttle levels, i.e., percentage of CPU time consumption, allowing us to explore different scenarios during detection. We were able to show that even attacks with reduced CPU consumption are detected.

We also developed our approach to be able to detect *binary cryptojacking samples.* To find these binaries, we searched through malware repositories such as *https://urlhaus.abuse.ch/* and *https://dasmalwerk.eu/*. We end up with 5 binary malware

```
<script src="https://www.hostingcloud.racing/Aync.js"></script>
<script>
      var _client = new
  Client.Anonymous('03f0ff94450082fe747c62c0d24b9c90d4d5f06d581643c1ebe2740356f36321',
  {
         throttle: 0 , c: 'w' , ads: 0
     });
      _client.start();
</script>
<div style="display:none">
    </div>
```

Fig. 2. Malware source code for *www.serieshdpormega.com*

samples for cryptojacking with different throttle values (Table IV).

TABLE IV
BINARY CRYPTOJACKING SAMPLES

Binary sample
down.gogominer.com/sex_Live1.5.0.1099.exe
14.141.175.107/cryptominerbros/wordpress/
14.141.175.107/cryptominerbros/wordpress/wp-content/Vh/
149.28.24.180/miner/bashd32
ivansupermining.info/bin/minbuild.exe

B. Creating the baseline extractions

With the necessary malicious samples we created the baseline extractions that later were used to create the hybrid dataset we needed. B-Profiles aim to encapsulate the entity behaviors of users in their normal tasks. This is one of the great challenges of artificial datasets, to achieve the complex behavior of a real-world users. For our dataset extractions we defined 3 types of host behavior profiles: (1) *Low-Profile (L)*; (2) *Mid-Profile (M)*; (3) *High-Profile (H)*. Profile (1) represents hosts with low use of system resources and light network traffic. Profile (2) represents host with an intermediary user with a workload focused on productivity (Microsoft Office, cloud working). The high-profile (3) aims to reproduce network and CPU demanding loads (virtual machines, Adobe Photoshop, web downloads, multiple web tabs). Our approach aimed to create a dataset with different network types, i.e., to enable cryptojacking detection for different workloads, and most importantly, to understand if high-profile workloads hide the attack behavior.

Having our strategy defined, we used 9 Windows 10 Enterprise with Intel(R) Pentium(R) CPU G3250 @3.20GHz, 3200 Mhz, 2 Core and 8,00 GB of RAM to run our extractions. These machines were divided by the three profiles, and each machine had Wireshark installed to obtain the flows, and a PowerShell script developed by us to fulfill our need for the performance counters.

Following a schedule so that the expectations for the profiles were met, we end up with one day of extractions regarding *benign activity*, with 9 *CSV* files three for each profile and similarly, *9 PCAP files* with the network traffic. For the PCAPs files we used *CICFlowMeter* available in *https://github.com/ahlashkari/CICFlowMeter* to obtain the *CSV files* with flows from the network packets.

Following the same approach, but now using the malicious samples previously mentioned, we used 10 machines. 5 of them were used to run the browser-based scripts under strict schedule and the other 5 for the binary samples than run permanently in the background. Also, here the machines were

divided by profiles. At the end of our daily extraction, we obtained equivalent files as in the benign extraction.

As the final step to create the dataset, we created 10 days of extractions, one day for each one of the victims we had created. We defined the number of machines that we wanted in the dataset for each day, a number ranging from 300 to 400. The process to create these machines from the baseline extractions was:

- defining a number of machines per profile;
- reproducing the amount of desired machines from the baseline extractions adding noise.

Regarding the noise method, the objective was to clone machines/hosts by changing the baseline values by a random percentage (Equation 3).

$$x' = x + random([-1, 1]) \times \frac{x}{random([3, \alpha])}, \alpha \in [15, 30] \quad (3)$$

The predefined values were chosen to set a threshold for which we did not change the type of profile for the replicas. Our dataset ended up with 10 days of data, with the first 6 days having predominantly low-profile hosts, a common behavior in real-world networks, and the other 4 days with predominantly high-profile hosts, the main reason being the evaluation of our approach to detect cryptojacking among heavy workloads. Additionally, for this 10 days, half of them have victims under browser-based cryptojacking, i.e., cryptojacking exists for specific time windows during the day. The other half has binary malware and cryptojacking is constantly running during the day.

V. Experimental evaluation

To develop and evaluate CryingJackpot, we used Python (v3) [51]. In addition, we used popular libraries such as Pandas [52] for data manipulation, and Scikit-learn [47] for data processing and the clustering algorithms. All the experiments were done in commodity hardware (4-Core Intel Core i7 2.6GHz with 16GB RAM). The focus of the experiments is the test against different real-world possible scenarios and performance evaluation.

A. Metrics

TABLE V
Confusion matrix-based indicator

Indicator	Description
True Positives (TP)	entities correctly classified as outliers
False Positives (FP)	entities wrongly classified as outliers
True Negatives (TN)	entities correctly classified as inliers
False Negatives (FN)	entities wrongly classified as inliers

We considered an outlier – a positive – to be a host isolated in a cluster, identified by an IP address, flagged by the CryingJackpot. Based in the confusion matrix of Table V, the following metrics were used in the evaluation:

- Precision (PREC) – the fraction of outliers that are real (i.e., true positives): $PREC = TP/(TP + FP)$
- Recall (REC) – the fraction of outliers that are correctly classified as such by the detector: $REC = TP/(TP + FN)$
- F-Score – a global detection score: $FScore = 2 \times (PREC \times REC)/(PREC + REC)$

To obtain the values for the indicators of Table V, we analyzed the clustering results concerning attack's windows only (persistent malware implies the consideration of all time windows).

B. Dataset characterization

Two datasets were used. The first dataset, developed by us, is described in Section IV. It is a hybrid dataset, where data was generated through baseline real-world data profiles. This dataset contains both network flows and performance counters data for Windows machines. Also, several types of cryptojacking attacks are present among the data.

The second, *CIC-IDS 2018* [30], that we designate as *artificial dataset*, was created to test and evaluate network IDSs. We used this dataset combined with the cryptojacking attacks that we created to test our approach under different circumstances (the original dataset contained no cryptojacking attacks). Regarding the original dataset, its authors developed a systematic approach in order to produce a diverse and comprehensive benchmark dataset. In their approach, they created user profiles with abstract representations of activity seen on typical networks. The benign behavior of each machine was generated using CIC-BenignGenerator [49], which is a tool to generate B-Profiles, i.e., profiles that translate the abstract behavior of a group of users, thanks to several ML and statistical analysis techniques. Regarding malicious behavior, this was generated using M-Profiles. These profiles aim to describe an attack scenario unambiguously, in such a way that humans might interpret these profiles and subsequently carry their attacks.

We used the first 6 days from this dataset for our experiments. However, first, we had to add the cryptojacking attacks to the already existing data. For this process, we used the malicious extractions described in Section IV, and for each one of the 6 days, we added the respective cryptojacking-related flows. One aspect that must be taken into account is that the cryptojacking samples we produced and introduced in this artificial dataset differed in nature from the original data since they were not obtained under the same circumstances.

TABLE VI
Summary of the attacks for the CIC-IDS-2018 dataset

Day	Attack	Duration
1	Brute Force to FTP and SSH	90 min each
	Cryptojacking Script	3, 5, 7, 10 min
2	DoS GoldenEye and Slowloris	40 min each
	Cryptojacking Binary	permanent
3	Brute Force to FTP and DoS Hulk	60 min + 35min
	Cryptojacking Script	3, 5, 7, 10 min
4	DoS LOIC-HTTP	60 min
	Cryptojacking Binary	permanent
5	DoS LOIC-UDP and HOIC	30 min + 60 min
	Cryptojacking Script	3, 5, 7, 10 min
6	Brute force Web/XSS and SQL inj.	60 min + 40 min
	Cryptojacking Binary	permanent

Unlike the *hybrid dataset*, this dataset comes with a set of attacks with flow-based data only, on which we add information related with cryptojacking. The attacks and their duration are

described in Table VI. One must notice that browser-based cryptojacking (scripts) are associated with different time windows, while for binary cryptojacking we assume a permanent attack window, i.e., the attack was active during the whole day.

In both datasets data was divided into 24 hour intervals, and our test were conducted using for the clustering algorithms, time windows of 10, 30, 60, and 120 minutes. Furthermore, clustering focuses on internal host only, as our goal is to detect machines for a given network subject to cryptojacking.

C. Results with the hybrid dataset

Our first tests consisted in combining flow-based features with counters-based ones, to identify cryptojacking by using the clustering algorithms already mentioned. We also performed the detection using both types of features independently. A first observation was that *the combination of host-based and flow-based features generated better results than using them separately*. These differences translate into far more consistent results for all implemented algorithms and improvements on the overall results (Figure 3). From using only flow-based features, we obtained a 39% increase for the best F-Score, and we overcome the lack of robustness when using only counters with a F-Score 4.5% increase, when combining both types of features.

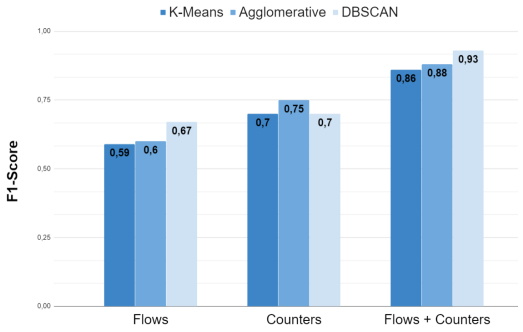

Fig. 3. Comparison of F1-Scores when using only flow-based features, performance counters-based features, and the combination of both (hybrid dataset).

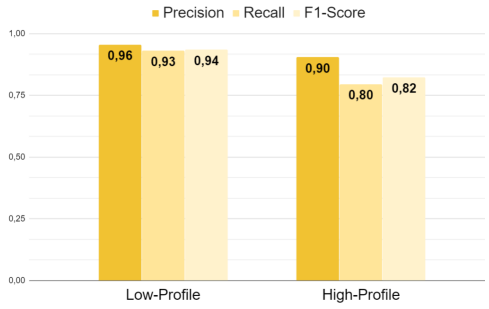

Fig. 4. Comparison of results between mostly low-profile days and mostly high-profile days (hybrid dataset).

Figure 4 presents results comparing 6 days with predominantly low-profile machines and 4 days with predominantly high high-profile machines (heavier workload). These results were obtained by computing the average of the individual metrics when dividing the days by the categories. The results indicate that *our detection approach is robust when detecting cryptojacking in machines with heavy workload*, which is a considerable challenge as the CPU consumption of the cryptojacking scripts or malware is masked by the machine workload. For example, day 16 had predominantly high-profile machines, with one low-profile victim, infected with a binary. We obtained an average F1-Score of 0.98.

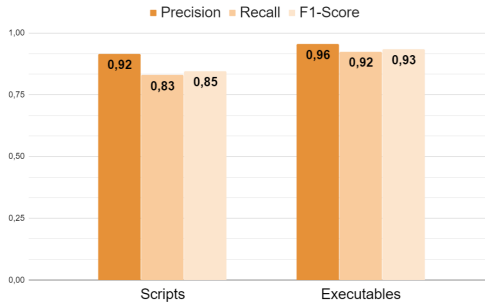

Fig. 5. Comparison of results between browser-based and binary cryptojacking (hybrid dataset).

A similar study was conducted by dividing the days according to the type of cryptojacking attack, browser-based (script) or binary (Figure 5). The overall *results are slightly worse when considering browser-based cryptojacking* because, as we discovered, for attack windows under 3 minutes, our detection capabilities were reduced. When studying the network packets and the system response, we saw that there is a delay while the connection with the mining pool is established, having less impact on the victim features since average values are used for some of them.

Figure 6 shows the results for the two best time windows when the silhouette method is used. The values displayed were obtained by computing the average of each metric for all days at a given time window. *The three implemented algorithms had similar behavior, with DBSCAN performing slightly better.* For K-means and Agglomerative results, silhouette contributed to better results than the most common elbow algorithm. However, the same did not apply for the artificial dataset (see Section V-D), which could be related to the use of a flow-based features without the host-based features.

D. Results with the artificial dataset

The artificial dataset has only flow-based features, but includes more attacks than cryptojacking. Therefore, here we evaluate the performance of CRYINGJACKPOT with that restriction and with other attacks.

The cryptojacking detection results for the two best windows are displayed in Figure 7. We observe that *our clustering ensemble approach reaches an "average" result*, an improvement for the somewhat inconsistent results obtained by the three implemented algorithms (e.g., low precision but high recall for Agglomerative). Moreover, the overall best performer detecting cryptojacking was K-means combined with the elbow method, with an average F1-Score for all time windows of 0.68.

Even though we expected that all cryptojacking attacks would be detected as they have an identifiable behavior, we obtained an average recall for the three algorithms of 0.84, so cryptojacking was not always detected (there were false negatives). This result leads us to conclude that in terms of network-based features,

based on flows, data regarding cryptojacking is fairly similar to the original data patterns of the artificial dataset.

TABLE VII
RECALL FOR CIC-IDS-18 ORIGINAL ATTACKS DETECTION

Time Window	**K-means**	**Agglomerative**	**DBSCAN**
10 min	0.03	0.40	0.11
30 min	0.09	0.54	0.32
60 min	0.24	0.62	0.32
120 min	0.46	0.79	0.63

Regarding the original attacks' detection – i.e., attacks other than cryptojacking –, we obtained low recall values (Table VII). This was to be expected as our scheme is targeted at detecting a specific attack, cryptojacking, and to use flow features in combination with host-based features.

VI. RELATED WORK

There are several papers related with cryptojacking detection with the majority focusing on detecting browser-based cryptojacking. The first papers used signature-matching as detection approach [12], [25], a technique that was employed in more recent works combined with ML algorithms [15], [24]. A different set of works explored the detection of resource consumption by the attack [13], [29], with several works using consumption indicators as features for ML methods [11], [26].

Using network flows for network security purposes, is a recent approach for cryptojacking detection. Muñoz et al. [14] presented a ML-based method able to detect cryptojacking using NetFlow/IPFIX measurements. Although it is the subject of less research, there are related works that explore these techniques for binary cryptojacking [7], [28].

As previously explained, we advance previous works by using unsupervised ML algorithms that combine flow-based and host-based features for both browser-based and binary cryptojacking. No previous work provides an equivalent mechanism.

VII. CONCLUSION

We present CRYINGJACKPOT, a hybrid approach for cryptojacking detection, based on unsupervised learning, that detects undefined attacks without signatures and clean training data. CRYINGJACKPOT is based on clustering, i.e., on aggregating hosts with similar traffic patterns, using for this end the combination of features from flow-based data and performance indicators for Windows OS machines. For the evaluation, two datasets were used. Firstly, we developed a reliable hybrid dataset with both flow-based and host-based data, where cryptojacking detection was performed with excellent results. We then used a public dataset to test our flow-based features capabilities by studying the detection of cryptojacking in the presence of other attacks, having achieved positive results. CRYINGJACKPOT has proven to be a reliable and flexible solution for cryptojacking detection with precision, recall, and F1-score between 0.92 and 1.

Acknowledgements This research was supported by national funds through Fundação para a Ciência e Tecnologia (FCT) with reference UIDB/50021/2020 (INESC-ID), by the Portuguese Army (CINAMIL), and by the European Commission under grant 830892 (SPARTA).

REFERENCES

[1] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark, "A first look at browser-based cryptojacking," in *Proceedings - 3rd IEEE European Symposium on Security and Privacy Workshops, EURO S and PW 2018*, Jul. 2018, pp. 58–66.

[2] H. L. J. Bijmans, T. M. Booij, and C. Doerr, "Inadvertently making cyber criminals rich: A comprehensive study of cryptojacking campaigns at internet scale," *Proceedings of the 28th USENIX Security Symposium*, pp. 1627–1644, 2019.

[3] M. E. Peck, "Blockchains: How they work and why they'll change the world," *IEEE Spectrum*, vol. 54, no. 10, pp. 26–35, Oct. 2017.

[4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system satoshi nakamoto," Tech. Rep., 2008.

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proceedings - 2017 IEEE 6th International Congress on Big Data, BigData Congress 2017*, Sep. 2017, pp. 557–564.

[6] Alliance, Cyber Threat, "The illicit cryptocurrency mining threat," Tech. Rep., 2018.

[7] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 73–86.

[8] D. Goodin, *Now even YouTube serves ads with CPU-draining cryptocurrency miners*, 2018. [Online]. Available: https://arstechnica.com/information-technology/2018/01/now-even-youtube-serves-ads-with-cpu-draining-cryptocurrency-miners (visited on 11/08/2019).

[9] The Guardian, *Ads don't work so websites are using your electricity to pay the bills*, 2017. [Online]. Available: https://www.theguardian.com/technology/2017/sep/27/pirate-bay-showtime-ads-websites-electricity-pay-bills-cryptocurrency-bitcoin (visited on 11/05/2019).

[10] S. Gatlan, *European airport systems infected with monero-mining malware*, 2019. [Online]. Available: https://www.bleepingcomputer.com/news/security/european-airport-systems-infected-with-monero-mining-malware (visited on 11/08/2019).

[11] I. Petrov, L. Invernizzi, and E. Bursztein, "Coinpolice: Detecting hidden cryptojacking attacks with neural networks," *arXiv preprint arXiv:2006.10861*, 2020.

[12] W. Wang, B. Ferrell, X. Xu, K. W. Hamlen, and S. Hao, "Seismic: Secure in-lined script monitors for interrupting cryptojacks," *European Symposium on Research in Computer Security*, vol. 11099 LNCS, pp. 122–142, 2018.

[13] J. D. Parra Rodriguez and J. Posegga, "RAPID: Resource and API-based detection against in-browser miners," *Proceedings of the 34th Annual Computer Security Applications Conference*, pp. 313–326, 2018.

[14] J. Z. I. Munoz, J. Suarez-Varela, and P. Barlet-Ros, "Detecting cryptocurrency miners with NetFlow/IPFIX network measurements," in *IEEE International Symposium on Measurements & Networking*, Jul. 2019.

[15] A. Kharraz, Z. Ma, P. Murley, C. Lever, J. Mason, A. Miller, N. Borisov, M. Antonakakis, and M. Bailey, "Outguard: Detecting in-browser covert cryptocurrency mining in the wild," in *The World Wide Web Conference*, 2019, pp. 840–852.

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

[17] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*. 2011, vol. 66, pp. 37–39.

[18] F. Sabahi and A. Movaghar, "Intrusion detection: A survey," *2008 Third International Conference on Systems and Networks Communications*, pp. 23–26, 2008.

[19] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the 28th Australasian Conference on Computer Science*, 2005, pp. 333–342.

[20] P. Casas, J. Mazel, and P. Owezarski, "Unsupervised network intrusion detection systems: Detecting the unknown without knowledge," *Computer Communications*, vol. 35, no. 7, pp. 772–783, 2012.

[21] L. Sacramento, I. Medeiros, J. Bota, and M. Correia, "Flowhacker: Detecting unknown network attacks in big traffic data using network flows," in *Proceedings of IEEE Trustcom*, 2018.

[22] L. Dias, H. Reia, R. Neves, and M. Correia, "Outgene: Detecting undefined network attacks with time stretching and genetic zooms," vol. 11928 LNCS, pp. 199–220, 2019.

[23] L. Dias, S. Valente, and M. Correia, "Go with the flow: Clustering dynamically-defined netflow features for network intrusion detection with DynIDS," 2020, in submission.

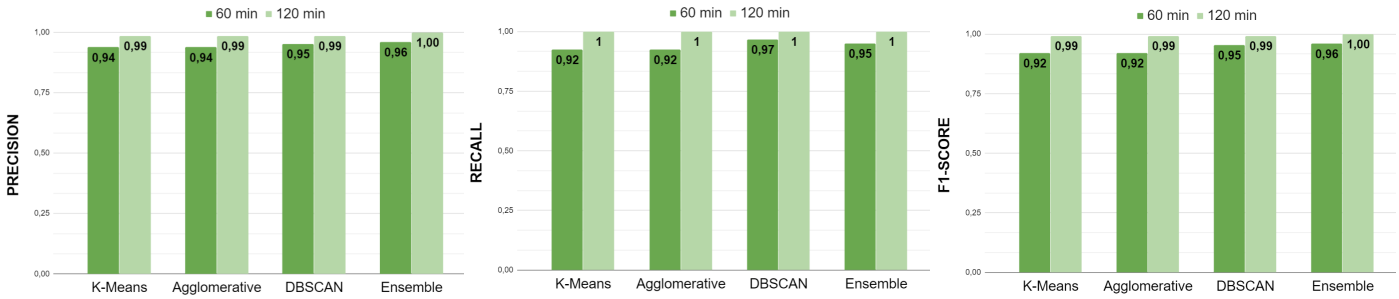

Fig. 6. Comparison of the overall performance for the two best time windows (hybrid dataset)

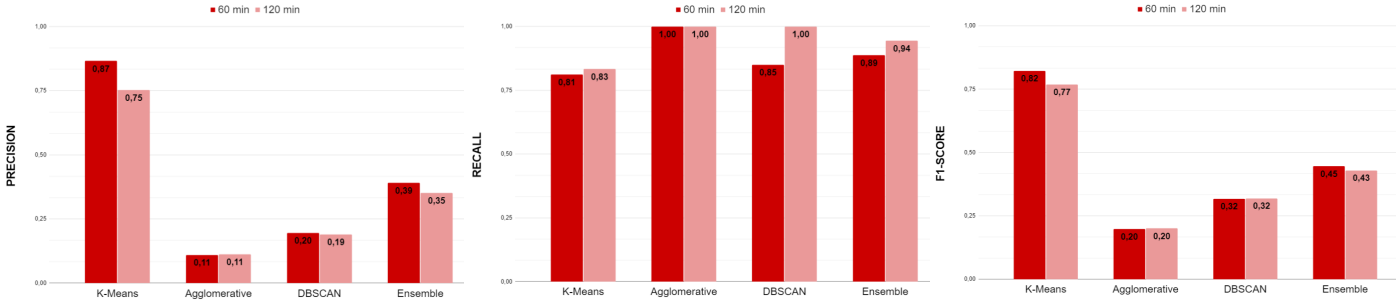

Fig. 7. Comparison of the overall performance for the two best time windows (artificial dataset)

[24] D. Carlin, P. O'Kane, S. Sezer, and J. Burgess, "Detecting cryptomining using dynamic analysis," in *2018 16th Annual Conference on Privacy, Security and Trust*, 2018, pp. 1–6.

[25] J. Rauchberger, S. Schrittwieser, T. Dam, R. Luh, D. Buhov, G. Pötzelsberger, and H. Kim, "The other side of the coin: A framework for detecting and analyzing web-based cryptocurrency mining campaigns," *Proceedings of the 13th International Conference on Availability, Reliability and Security*, 2018.

[26] 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, pp. 1873–1881.

[27] Symantec, "ISTR internet security threat report volume 23," Tech. Rep., 2018.

[28] D. Draghicescu, A. Caranica, A. Vulpe, and O. Fratu, "Crypto-mining application fingerprinting method," *2018 International Conference on Communications*, pp. 543–546, 2018.

[29] D. Tanana, "Behavior-based detection of cryptojacking malware," in *2020 Ural Symposium on Biomedical Engineering, Radioelectronics and Information Technology*, 2020, pp. 543–545.

[30] *A realistic cyber defense dataset (cse-cic-ids2018) - registry of open data on aws*. [Online]. Available: https://registry.opendata.aws/cse-cic-ids2018 (visited on 08/12/2020).

[31] ecurrencyhodler, *Was monero's pow change a success — Medium*, 2019. [Online]. Available: https://medium.com/@ecurrencyhodler/was-moneros-pow-change-a-success-81cfeaa08aae (visited on 05/04/2020).

[32] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Communications Surveys and Tutorials*, vol. 16, no. 1, pp. 303–336, Mar. 2014.

[33] T. Zoppi, A. Ceccarelli, and A. Bondavalli, "Evaluation of anomaly detection algorithms made easy with RELOAD," in *IEEE 30th International Symposium on Software Reliability Engineering*, 2019, pp. 446–455.

[34] T. Zoppi, A. Ceccarelli, L. Salani, and A. Bondavalli, "On the educated selection of unsupervised algorithms via attacks and anomaly classes," *Journal of Information Security and Applications*, vol. 52, p. 102 474, Jun. 2020.

[35] A. Strehl and J. Ghosh, "Cluster ensembles: A knowledge reuse framework for combining multiple partitions," *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 583–617, 2002.

[36] C. X. Zhang, J. S. Zhang, N. N. Ji, and G. Guo, "Learning ensemble classifiers via restricted boltzmann machines," *Pattern Recognition Letters*, vol. 36, no. 1, pp. 161–170, Jan. 2014.

[37] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, 2005.

[38] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, Jan. 2012.

[39] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, vol. 96, 1996, pp. 226–231.

[40] D. Beeferman and A. Berger, "Agglomerative clustering of a search engine query log," in *Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000, pp. 407–416.

[41] 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 ACM Annual Computer Security Applications Conference*, 2013.

[42] *Windows performance monitor - Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/perfctrs/about-performance-counters (visited on 08/08/2020).

[43] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, Oct. 2004.

[44] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information," STD 77, Sep. 2013.

[45] S. Zhao, M. Chandrashekar, Y. Lee, and D. Medhi, "Real-time network anomaly detection system using machine learning," in *11th International Conference on the Design of Reliable Communication Networks*, Jul. 2015, pp. 267–270.

[46] C. Yuan and H. Yang, "Research on k-value selection method of k-means clustering algorithm," *Multidisciplinary Scientific Journal*, vol. 2, no. 2, pp. 226–235, Jun. 2019.

[47] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[48] N. Rahmah and I. S. Sitanggang, "Determination of optimal epsilon (eps) value on DBSCAN algorithm to clustering data on peatland hotspots in Sumatra," in *IOP Conference Series: Earth and Environmental Science*, vol. 31, Feb. 2016.

[49] I. Sharafaldin, A. Gharib, A. H. Lashkari, and A. A. Ghorbani, "Towards a reliable intrusion detection benchmark dataset," *Software Networking*, vol. 2017, no. 1, pp. 177–200, 2017.

[50] S. Varlioglu, B. Gonen, M. Ozer, and M. Bastug, "Is cryptojacking dead after coinhive shutdown?" *Proceedings - 3rd International Conference on Information and Computer Technologies*, pp. 385–389, 2020.

[51] G. van Rossum, "Python tutorial release 3.8.1," Tech. Rep., 2020.

[52] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445, 2010, pp. 51–56.