Detection of encrypted cryptomining malware connections with machine and deep learning

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 18, 2020, accepted August 4, 2020, date of publication August 26, 2020, date of current version September 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019658

Detection of Encrypted Cryptomining Malware Connections With Machine and Deep Learning

ANTONIO PASTOR[1], ALBERTO MOZO[2], STANISLAV VAKARUK[2], DANIELE CANAVESE[3], DIEGO R. LÓPEZ[1], LEONARDO REGANO[3], SANDRA GÓMEZ-CANAVAL[2], AND ANTONIO LIOY[3], (Member, IEEE)

[1]Telefónica I+D, 28010 Madrid, Spain
[2]Departamento de Sistemas Informáticos, Universidad Politécnica de Madrid, 28031 Madrid, Spain
[3]Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Turin, Italy

Corresponding author: Alberto Mozo (a.mozo@upm.es)

This work was supported in part by the European Union's Horizon 2020 Research and Innovation Programme under Grant 833685 (SPIDER), and in part by the European Institute of Innovation and Technology (EIT) Digital's Innovation Activity under Grant 18217-19 (Deep-Augur).

ABSTRACT Nowadays, malware has become an epidemic problem. Among the attacks exploiting the computer resources of victims, one that has become usual is related to the massive amounts of computational resources needed for digital currency cryptomining. Cybercriminals steal computer resources from victims, associating these resources to the crypto-currency mining pools they benefit from. This research work focuses on offering a solution for detecting such abusive cryptomining activity, just by means of passive network monitoring. To this end, we identify a new set of highly relevant network flow features to be used jointly with a rich set of machine and deep-learning models for real-time cryptomining flow detection. We deployed a complex and realistic cryptomining scenario for training and testing machine and deep learning models, in which clients interact with real servers across the Internet and use encrypted connections. A complete set of experiments were carried out to demonstrate that, using a combination of these highly informative features with complex machine learning models, cryptomining attacks can be detected on the wire with telco-grade precision and accuracy, even if the traffic is encrypted.

INDEX TERMS Cryptomining detection, malware detection, cryptojacking detection, cryptocurrency mining, netflow measurements, encrypted traffic classification, machine learning, deep learning.

I. INTRODUCTION

Malware is an epidemic problem based on a simple concept: exploit computer resources of a victim. Throughout recent history, malware evolution offers more resilience and versatility to achieve multiple goals: denial of services (DoS), sensitive data theft, anonymity for illegal activities between others. But in general, the main motivation is economical. Malware families have developed a wide variety of techniques to profit, from simple blackmail with DoS menace, to sophisticate bank trojans, with the expectation to reach in the end some fiduciary money. In this unstoppable evolution [1] cybercriminals search new models for quick profits. Digital currencies fit very well in this strategy.

Bitcoin [2] was the first decentralized digital currency, based on blockchain. Anyone with enough computational capacity can participate and make profit, doing cryptographic calculations to contribute to the blockchain. As a result of this calculations, a variable reward is provided, using the very same cryptocurrency. This is commonly named as cryptomining. Unfortunately, standalone cryptomining is no longer profitable and the solution is to associate multiples computers (or bots) using a new type of service, called mining pools. These arrangements offer a percentage of the mining rewards proportional to the computational resources offered by their participants using a specific protocol (see Section III). Nowadays, multiple cryptocurrencies have been born and died [3], offering multiple profitable ecosystems for mining. Precisely, as part of this ecosystem proliferation, some cryptocurrencies have generated an opposition to specialized hardware (ASIC) devices, altering the algorithms [4], [5] to make them useless, and opening the opportunity again to profit with normal computer mining.

Cybercriminals can populate their cryptocurrency wallets following two main approaches. First, by exploiting the ransomware malware family to encrypt a victim's data

The associate editor coordinating the review of this manuscript and approving it for publication was Jihwan P. Choi.

and receive a payment using untraceable cryptocurrency in order to let the victim to recover the encrypted information. The second alternative consists of using botnets or executing illegal processes in browsers (cryptojacking) to add surreptitiously the computer resources of the victims, without their consent, to an ASIC-resistant mining pool and spend computational resources in the benefit of the criminal. The first one has the inherent risk to attract attention from authorities and depends in the victim willingness to pay, but also gives punctual big benefits. On the contrary, the second approach can operate for long periods of time generating constantly increasing benefits [6].

The focus of our research is on the second approach, offering a solution to industry for detecting cryptomining activity over the network. Mining traffic is often encrypted to avoid detection, and botnets can use private proxy IPs to aggregate and masquerade the mining pools.

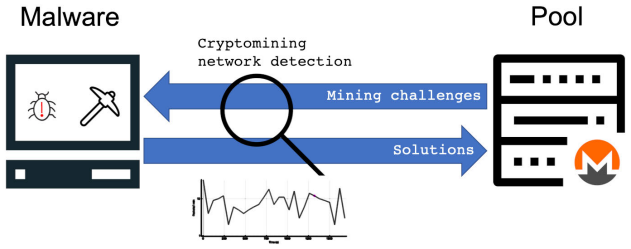


FIGURE 1. Profiling network behaviour to detect cryptomining activity.

It is widely known that nowadays, more and more traffic is being encrypted (> 80%) as encryption is been adopted by many network protocols (e.g. TLS, SSH, QUIC, VPNs). Additionally, governments concerned about privacy issues are imposing limitations to telecom providers for accessing user data or inspecting packet payload. In this context, we present a novel approach for detecting cryptomining activity over the network (Fig. 1) even if it is encrypted, based on profiling the network behaviour of the mining clients by means of machine and deep learning techniques, as opposed to classic deep packet and payload inspection techniques or mining pool domain names identification. We propose to apply supervised machine and deep learning techniques that will exclude the usage of input features such as IP addresses, port numbers or application level information, which may contain personal and sensitive user data. In addition, encrypted traffic will be utilised extensively during training and testing phases of machine learning models as this type of traffic is going to be the prevalent in the near future.

We selected Monero (XMR) as the cryptomining protocol for this study because of several reasons. First, Monero is by far the favorite cryptocurrency among cybercriminals in the underground market [6]. Second the public commitment by Monero developers to exclude ASIC from the mining ecosystem, and the low amount of resources required (such as the light mode in RandomX algorithm [5] that only requires 256 MiB of shared memory), has increased its adoption by malware families. Finally, the Monero preference for CPU mining makes more realistic the experimentation over standards clients with respect to GPUs or ASIC devices.

We demonstrate in a controlled but realistic environment that a careful feature selection is crucial for obtaining good detection performance with machine and deep learning models in traffic scenarios where cryptomining flows are present. We compare machine learning performance using two different sets of statistical features obtained from network traffic flows. The first set is extracted using the Tstat tool, a well known passive sniffer able to provide per flow statistics at both the network and transport levels, and the second is derived from the IETF standard NetFlow/IPFIX metrics, which are widely used in the industry. To the best of our knowledge, our research work is the first proposal to utilise features obtained with Tstat as input to supervised machine learning algorithms. We demonstrate that the first set provides more information than the second, which translates into the fact that, independently of the machine learning model utilised, predictions are more stable, accurate and precise. Furthermore, we show that complex models (e.g. Random Forest and Fully connected deep neural networks) are able to detect cryptomining traffic activity with significantly greater precision, accuracy and stability than simpler models (e.g. Decision trees and Logistic Regression), even if the traffic is encrypted. Finally, we conclude that complex machine learning models, as those proposed in this article, using as input a set of exhaustive features obtained with Tstat, identify cryptomining traffic with the accuracy, precision and stability required by industrial deployments.

The rest of the paper is structured as follows: Section II presents the main contributions of this research work. Regarding that in order to detect cryptomining traffic promptly and with accuracy, the network is the best place to identify this type of attack, Section III introduces the reader to the details of cryptomining protocols that will be used later in our experiments. Section IV summarises a few number of previous works proposed in this topic and the problem setting is depicted in Section V. The set of machine learning models utilised in our experiments (Fully Connected Neural Networks, Random Forest, C4.5 and CART decision trees and Logistic Regression) are detailed in Sections VI. Section VII describes the approach and method that we adopted to create our data set and train our models, while the experimental results are explained in Section VIII. We conclude in Section IX.

II. CONTRIBUTION

The detection of cryptomining malware over the network is still in its infancy and, to the best of our knowledge, there is only one research work [7] that addresses this problem but assuming a rather unrealistic scenario in which i) encrypted traffic is not considered, ii) only a reduced set of traditional machine learning models were proposed and iii) a set of slightly uninformative features were used as input. Hence, when the experiments were reproduced in our controlled

environment, the obtained performance was very far from the expected by the telecom industry.

In order to surpass the limitations of the current proposals, our experimental work presents the following novelties:

- A complex and realistic cryptomining scenario that was set up for training and testing machine learning models for cryptomining flow detection. This scenario is composed of typical Internet applications (e.g. web browsing, multimedia, file access via shared folders) and cryptomining clients interacting with real mining server pools spread across the Internet. This controlled setup allows to generate real network traffic that can be automatically collected and tagged with 100% of precision even if the traffic is encrypted. The labelled traffic will be used for training and testing machine learning models for cryptomining flow detection that fulfill the strict requirements of the telecom industry.
  A distinguishing characteristic of our experimental setup from existing works is the usage of encrypted connections in addition to non-encrypted ones. Nowadays, more and more applications are encrypting their communications and the identification of their contents poses an additional challenge to machine learning classifiers and in particular to cryptomining detectors.
- A proposal for a new set of highly informative features to be used as input to real-time cryptomining flow detection. These features were derived from a set of forensic statistics produced by Tstat, a tool conceived for producing statistical data per connection from network packet traces. We modified the Tstat tool to derive a set of real-time flow features to be input to ML classifiers for cryptomining detection.
  In our experiments we demonstrated that when network traffic is encrypted, the usage of the proposed new set of features significantly increased ML performance with respect to using standard features.
- A complete set of experiments that demonstrate that crytomining attacks can be detected on the wire with telecom-grade precision and accuracy even if the traffic is encrypted, using the derived Tstat features jointly with complex machine learning models.

III. CRYPTOCURRENCIES AND CRYPTOMINING PROTOCOLS

Cryptocurrencies can be defined as digital media of exchange, based on the use of cryptographic primitives to both regulate the emission of new cryptocurrency units and to verify transactions. The first and most widespread cryptocurrency, Bitcoin (BTC), was introduced in the seminal paper by Satoshi Nakamoto [2]. Cryptocurrencies are completely decentralized, with no control authority, a characteristic shared with P2P file sharing networks. The validation of transactions is based on a distributed ledger, implemented with a blockchain (an immutable and shared data structure [8]). When a cryptocurrency is spent, the related transaction is added to the blockchain. Cryptocurrencies users (called miners) can then validate the new block, i.e. a set of transactions,

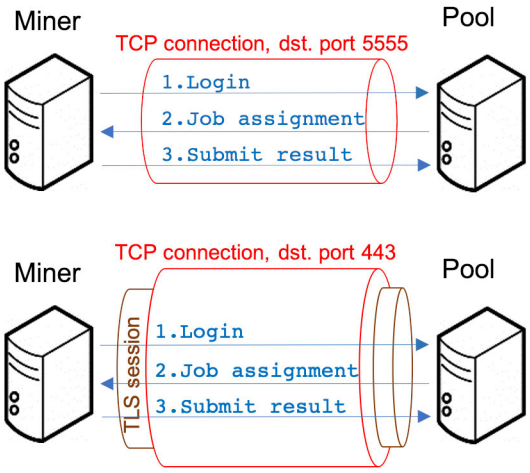


FIGURE 2. Message flows involved in stratum protocol. Upper diagram runs over a TCP connection, botton diagram over a TLS session.

with a computationally intensive cryptographic process, commonly known as mining, preventing unfair use of the cryptocurrencies (e.g. double spending). Miners are encouraged to participate in the validation of transactions with rewards (i.e. new cryptocurrency units). However, the high level of complexity of the validation process makes obtaining such rewards feasible only for specialized hardware (ASIC) owners. Mining pools try to overcome this limitation coordinating the work and resources of different miners, who share their computational power over a network. The most widespread protocol used to distribute the mining process among the mining pool participants is called Stratum [9], and have been first used in the Slushpool mining pool [10].

The Stratum protocol, shown in Fig. 2, specifies a bidirectional communication channel between the mining pool server and its participants, with messages encoded following the JSON-RPC 2.0 specification [11]. Following, we provide a brief description of the typical Stratum protocol flow. First, the client can initiate (or resume) a session with the mining pool server, calling the subscribe RPC method on the server, specifying in the request the mining software that the client will use to solve the challenges that the server will subsequently send to the client. It should be noted that a mining pool user may employ multiple devices (workers in the Stratum terminology) in the mining process. Thus, the client needs also to authorize the server to send challenges to each worker, by calling the authorize RPC method on the server for each worker, sending its username and password to the server (step 1 in Fig. 2). Then, the server can send challenges to workers, calling on them the submit RPC method (step 2 in Fig. 2). If a worker finds a solution for a mining challenge, he can send it to the server by calling on it the submit RPC method (step 3 in Fig. 2). After verifying the correctness of the solution, the server can send new challenges to the worker that has solved the challenge, and also to other workers that were trying to solve it, again calling on them the submit RPC

method. Stratum messages may be sent in an encrypted form, for example embedding them in TLS frames [12].

IV. RELATED WORK

In spite of being a hot topic in nowadays industrial scenarios, the detection of cryptomining activities has been poorly investigated in the current scientific literature. Most of the current state-of-the-art works focuses primarily on detecting a cryptomining or cryptojacking software module running on a victim, by analyzing the behavior of the target node. Even fewer works tries to detect Stratum or other cryptomining protocols.

A. IDENTIFICATION OF CRYPTOMINING TRAFFIC

Very few research works are focused on detecting cryptmining activities by looking at the network traffic. The closest work to our research was proposed by Muñoz *et al.* in [7]. In this work, the authors evaluated four different machine learning techniques (SVMs, Naïve Bayes, CARTs and C4.5 decision trees) to identify non-encrypted cryptomining Stratum flows applying a non-DPI (Deep Packet Inspection) approach. They concluded that it is possible to identify non-encrypted Stratum connections that are used for mining five different cryptocurrencies (Bitcoin, Bitcoin-Cash, DogeCoin, LiteCoin and Monero). Additionally, the authors showed that some machine learning models (e.g. CART decision trees) were able to distinguish with accuracy these cryptomining flows from normal traffic and also to classify the cryptocurrency that was being mined. The main differences of our research with respect to this work are that (a) we propose more complex models (random forests and fully connected neural networks) for detecting cryptomining activity, which significantly increase stability, precision and accuracy of predictions, (b) conversely to Muñoz *et al.*'s approach, encrypted traffic is also considered in our research in order to evaluate machine learning performance in realistic scenarios, and (c) we use a more informative set of 51 features derived with the Tstat tool that produces significantly better results than the approach followed by Muñoz *et al.* that utilised only 8 features derived from NetFlow/IPFIX metrics. Comparing the results of the two proposals in the realistic scenario we set up, it can be observed that the usage of simpler machine learning algorithms with Netflow features, as in [7], generates instability and poor performance in machine learning predictions. On the other hand, when complex machine learning models are used jointly with more informative features, as the ones proposed in our work, high performing, robust and stable predictions are obtained.

Swedan *et al.* [13] proposed a set of more traditional approaches for detecting and, eventually, blocking cryptomining connections. The authors proposed an architecture named MDPS (Mining Detection and Prevention System) for dropping non-encrypted cryptomining flows produced by a browser (e.g. CoinHive, Crypto-Loot). Their approach is based on deploying proxies that performs a DPI inspection on connections in order to deny any suspicious flow, what it is rather ineffective when miners encrypt their communications.

B. IDENTIFICATION OF CRYPTOMINING SOFTWARE MODULES

Most of the work for detecting cryptomining processes make use of some kind of monitoring agent installed on a host (usually inside a browser) which is continuously monitoring the system resources and reacting if some odd behavior is detected.

In the work of Konoth *et al.* [14], the authors proposed an approach able to detect in-browser cryptomining activities by statically analyzing the WebAssembly code found in web pages. Cryptomining is based on computing multiple hash values and these functions have peculiar patterns (e.g. long sequences of XOR instructions or use of various cryptographic primitive functions) that can be found by analyzing the assembly opcodes. The authors proposed MineSweeper, a component that analyzes all the WebAssembly instructions loaded by a web page and reacts accordingly if the pattern analysis reveals that a potential cryptominer is loaded. The analysis of the code is not performed by machine-learning techniques, but the authors manually investigated the most common in-browser miners and hardcoded their patterns in their tool.

Carlin *et al.* [15] used instead a dynamic approach to detect in-browser cryptominers. Their approach makes use of traces (list of instructions sent to the CPU) collected by launching a browser (FireFox) with a debugger (OllyDbg). The opcodes of the instructions in these traces are then counted and this vector of integers is then used as the input features for a random forest classifier. Their machine-learning model was trained to be able to distinguish: cryptomining scripts, deactivated cryptomining scripts (files without the start mining function call), canonical scripts (not cryptomining related files) and canonical injected scripts (a weaponized version of the canonical scripts but with some cryptomining scripts injected). Their method was able to score a nearly 100% accuracy on each class.

Liu *et al.* [16] modified the kernel of a Chrome browser to include their miner detection component named BMDetector. This component analyzes the heap memory and the stack traces of the browser and sends them to a LSTM (Long-Short Term Memory) recurrent neural network that performs the actual classification in two classes: mining and non-mining scripts.

Other machine-learning models were also considered by Kharraz *et al.* [17] to identify cryptojacking websites by analyzing the browser behavior. Their monitoring agent, Outguard, continuously checks the scripts loaded and executed by a browser, computes simple numerical features and sends them to a trained machine-learning model to detect cryptomining scripts. Their features includes how many parallel tasks are used by a script, if WebAssembly is requested or how many WebSockets are created. They tested both random forests and SVMs (Support Vector Machines)

and experimentally found that the best results were obtained by the SVMs.

V. PROBLEM SETTING

Victims usually do not perceive as a problem this type of attacks, mainly because there is no direct economical loss or ransom to pay. The identification of traffic generated when cryptocurrencies are mined (in the so-called cryptomining process) can be useful both for companies and end users. First, companies may be interested into detecting stratum cryptomining messages exchanged on their corporate network, since they can point out that one or more employees are using company resources (e.g. company owned devices, electrical power) to mine cryptocurrencies, thus obtaining an illicit financial gain to the detriment of the company finances. A second scenario regards the worrying menace of cryptojacking, which endangers both end users and companies. Cybercriminals are understanding the economic potential of cryptomining and therefore, they spread malware that, after having successfully infected the target device, hides a cryptomining process running on the victim device. Cryptomining, especially when not limited, may be extremely detrimental for the end-user device performances, battery duration in the case of mobile devices, and can lead to a narrower lifespan of such devices. Furthermore, the identification of cryptojacking-related traffic can be used as an Indicator of Compromission (IoC) of the monitored device, even if the responsible malware has not yet been included into anti-malware applications databases.

Therefore, detecting and blocking cryptomining activity based on network traffic analysis allows to disable the former impacts in IT resources and discourage the usage of illegal cryptomining communications. Firstly, because no communication is possible and the mining process stops to consume resources. Secondly, because if the computational transaction cannot be ordered or collected using the network, the rewards are not generated.

In order to detect with accuracy and precision different types of cryptomining activity over the network, we propose a controlled network environment using real clients and servers to generate cryptomining related traffic competing with normal traffic (web surfing, video and audio streaming, cloud storage, file transfer, email and P2P among others) over a dedicated network and sharing Internet connectivity through a residential broadband access. Normal traffic clients interact both with servers located in the dedicated network and in the Internet. Cryptomining clients connect to real mining pools in the Internet. Being by far the favorite cryptocurrency between cybercriminals, Monero was chosen as the cryptocurrency of our experiments. Two different Monero's clients were selected to provide implementation variety of the Stratum protocol, which is by far the most widespread protocol used to distribute the mining process among mining pools. CPU and memory in client virtual machines were assigned in the range of normal user computers, without any support of specialised hardware such as GPU or ASIC, and the same constraints were applied to server virtual machine configurations. We configured all clients and servers with capacity to establish plain or encrypted connections. In particular, cryptomining clients were connected to mining pools that support both types of connections over the network.

Finally, we provided the capacity to capture and copy all the network traffic crossing the controlled network environment by means of port mirroring functionality in a physical switch. In this way, all the captured traffic can be labelled and stored for training supervised machine learning models or directly used for testing purposes. Collected packets are grouped into flows using the well-known 5-tuple: IP destination, IP source, port destination, port source and transport protocol. During the training phase of a machine learning model, each time a new packet arrives, a set of flow statistics (features) are generated and stored jointly with the corresponding label in a dataset for training or testing purposes. In production environments, the features are extracted from the flow and input to a previously trained machine learning model to predict the flow type (e.g. cryptomining or normal flow) in the form of a probability value.

VI. MACHINE LEARNING MODELS

Aiming to find the best performing models, we considered different supervised machine and deep learning classifiers in order to take advantage of the specific characteristics of each of them. Regarding that we do not assume either temporal or spatial relation among the features to be input to models, we decided to use deep learning architectures based on Fully Connected Neural Networks (FCNNs). In addition, we also considered several other well-known machine learning algorithms such as Logistic Regression models [18], [19], Classification And Regression decision Trees (CART [20] and C4.5 [21]) and Random Forests [22].

The Logistic Regression model is essentially a classification algorithm which allows modelling binary outcomes that have only one of two possible values. Several works propose the application of this technique to protocol based classification over Internet traffic ([23], [24]).

The Classification And Regression Tree (CART) [20] and the C4.5 algorithm [21] are decision tree models able to classify data into different classes based on a tree of rules inferred from both discrete and continuous features. These models allow to classify a input feature vector giving the probability to belong to a particular class based on the probability of the resultant leaf of the tree.

Random Forest (RF) model [22] combines several Decision Trees (specifically, CART trees) to produce a more accurate classification using majority vote process through bootstrap aggregating (bagging) and random selection features for each tree. The RF Model is widely used due to its non-linear classification capabilities, efficiency and robustness ([25], [26]).

The Fully Connected Neural Network (FCNN) model [27], [28], also known as MultiLayer Perceptron, is a type of feed forward neural network that organises their nodes in layers

with weighted connections feeding forward from one layer to next. Input vectors are propagated from the input layer through the hidden layers towards the output layer in order to map the input into output vectors. The FCNN model has been successfully used in a wide range of problems and domains for recognizing complex patterns from a big amount of data. Particularly, in [29] it was used to detect DDoS attacks over network traffic.

VII. OUR APPROACH

Our goal is to distinguish cryptomining traffic from regular Internet traffic, that is a variety of flows that consists of web connections, e-mail transmissions, streaming videos, data transfers from and to cloud services and so on. In order to achieve this goal we used the workflow depicted in Figure 3 to generate a suitable data set and to train a set of machine and deep learning models.

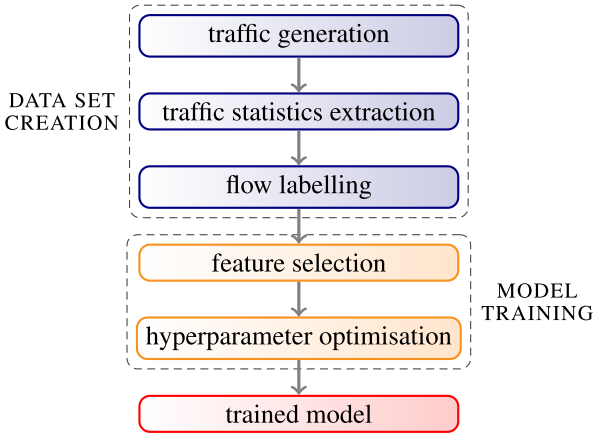


FIGURE 3. Our training workflow.

The first step in any machine learning based process is to produce a high quality data set. We used Mouseworld [30], a controlled NFV-based infrastructure, to set up and deploy on it the cryptomining attack scenario that would generate the traffic of our experiments. We installed a series of virtual machines with some miners connected to the Internet to produce the encrypted and non-encrypted Stratum flows and used a variety of other methods to gather the regular traffic (e.g. HTTP/HTTPS flows, multimedia streams). The TCP flows were captured and analyzed by a modified version of Tstat, a traffic analysis tool, in order to extract in real-time a set of features derived from several connection statistics that serve as the input features for our machine learning models. Then, we added an ad-hoc code to a special Mouseworld internal component, named Tagger, to automatically assign a binary label to each TCP flow instance (0: normal traffic connection, 1: cryptomining connection). After being labelled, we used these samples to train and test a variety of machine learning models. Before the actual model training, however, we perform a feature selection phase in order to increase the accuracy of our models and an hyperparameter optimisation step to search for the optimal model configuration. In order to validate the quality of our models we used a similar approach to the one depicted in Figure 3, by using Mouseworld as our network infrastructure in order to generate as many times as needed a variety of new and realistic traffic samples to be classified by our ML models. It worth noting that ML model training was done in a separate environment where specialised GPU cards were available for accelerating the training process of deep neural networks.

The following paragraphs describes in more detail how we built out data set and how we trained our models, while Section VIII contains the results of our experiments.

A. DATA SET CREATION

As mentioned before, the basis for our network infrastructure is the Mouseworld laboratory, a controlled environment set up in Telefonica R&D premises for running experiments that allow deploying complex network scenarios in a controlled way and generate realistic labelled data sets for training supervised ML components and validate supervised and unsupervised solutions. The Mouseworld Lab provides a way to launch clients and servers, collect the traffic generated by them even if they interact with clients and servers outside the Mouseworld in the Internet, and finally add labels to this traffic without operator intervention. This environment is deployed on an NFV-enabled architecture, under the management of an orchestrator (NFVO), extending an ETSI NFV MANO stack as necessary.

Figure 4 shows a detailed picture of the Mouseworld Lab that is composed of four modules interacting in a pipeline: Launcher, Data Collector, Feature extractor and Tagger.

The Launcher coordinates the Mouseworld environment and runs experiments that generate real network traffic that cross not only the Mouseworld internal network but also the Internet. This component allows to produce three basic types of regular traffic: web, video and file hosting (e.g. Dropbox, OwnCloud) by running real clients installed in a batch of Linux virtual machines. In addition, the Launcher can also manage ad-hoc virtual machines, for instance the ones we set up in our research work containing Monero miners, and it can also run sessions to generate network traffic of a collection of complementary Internet protocols using Ixia Breakingpoint, a commercial tool that allows to generate complex patterns of synthetic traffic.

The Data collector module gathers all the packets generated by a single experiment. The Feature Extraction module groups the collected packets into flows based on the classic five-tuple of source and destination IP-address/port number and transport protocol and calls an external module to obtain the set of features of each flow. In our experiment, we utilised a modified version of the Tstat tool in order to to derive from flow statistics a set of standard Netflow features jointly with our proposal for a set of highly informative features. The modified Tstat not only obtains flow features in forensic mode at the end of the connection as it is done in the original Tstat, but also in real-time at different instants of time, which allows us to train and test machine learning components

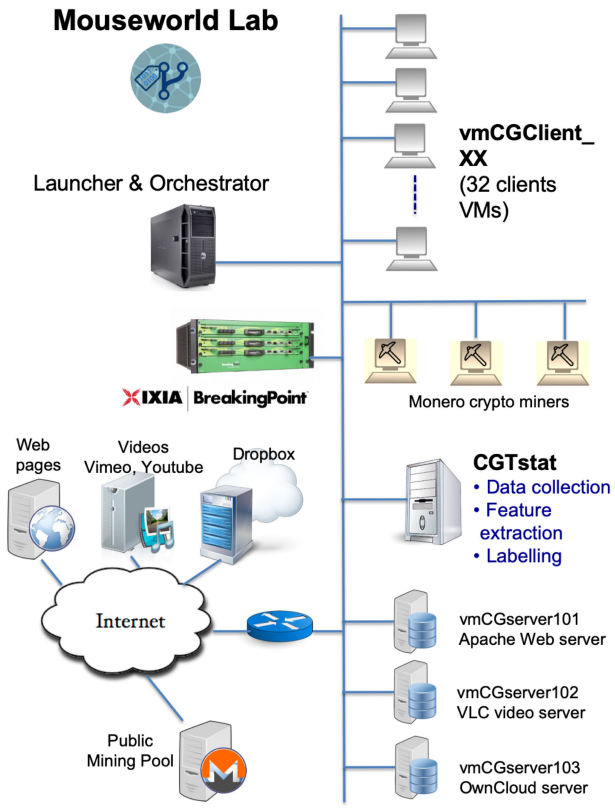


FIGURE 4. The Mouseworld Lab in Telefonica R&D premises.

that can identify cryptomining flows even when only a few packets of the flow have been transmitted.

Last, the Tagger adds automatically and without human intervention labels to each flow using external and log information output by the Launcher during the execution of each experiment. As flow tagging is highly dependent on the machine learning task, this component is simply a wrapper of the ad-hoc tagger that must be developed for each type of scenario. In our experiments we coded this component using the IP addresses and ports of the miners (clients) and cryptomining server pools in order to differentiate cryptomining flows from the rest containing normal traffic. Note that labels are only collected for training and testing the machine learning models, which do not need labels to generate predictions when they are running in a production environment.

The deployed infrastructure and configuration in Mouseworld are described below. We deployed 30 VMs for the generation of regular traffic (i.e. web, video and shared-folder flows). Web and video requests are generated using a Chrome headless browser and shared folder (i.e. file hosting) requests are produced using specific clients for Dropbox and OwnCloud. Using a configuration file in which the frequency of each type of request is specified, web, video and shared-folder requests are randomly generated from clients. These requests are randomly send to internal Mouseworld servers and to external servers located in the Internet. It is worth noting that both encrypted (e.g. HTTPS requests) and non-encrypted (e.g. HTTP requests) traffic requests were generated for all the services. In addition, the IXIA BreakingPoint tool was also configured to generate and inject synthetic patterns of various Internet network services (web, multimedia, shared-folder, email and P2P). The traffic generated by BreakingPoint was also configured in order to be composed of encrypted and non-encrypted flows.

In addition we created three cryptomining Linux virtual machines. We installed both xmr-stak [31] and xmrig [32] in all the three virtual machines and configured them to use both encrypted and non-encrypted Stratum connections. We used all these clients to mine the Monero [33] (XMR) cryptocurrency, commonly used for illegal purposes [6]. The cryptomining clients use default settings to connect with public mining pools using non encrypted TCP and encrypted TLS connections. Each miner was forcibly disconnected and reconnected each hour in order to simulate a fresh mining pool connection.

We designed four experiments combining the Launcher and BreakingPoint tools with different cryptomining clients and protocols. Each experiment was run for one hour with an average packet rate of approximately 1000 packets per second generating data sets with 8 millions of TCP flows of which 4 thousands are related to Stratum. In particular:

- in *experiment 1* the BreakingPoint tool was run jointly with two xmr-stak clients using TLS encrypted connections and one xmrig client establishing only non-encrypted TCP connections;
- in *experiment 2* the real clients running in Mouseworld's virtual machines were launched to interact with servers located in the Internet and in the Mouseworld internal network; in addition one xmr-stak client using TLS encrypted connections and two xmrig clients using non-encrypted TCP connections were deployed;
- in *experiment 3* a combination of real clients with two xmr-stak clients using TLS encrypted connections and one xmrig client establishing only non-encrypted TCP connections was launched;
- finally, in *experiment 4* the BreakingPoint tool was run jointly with one xmr-stak client using TLS encrypted connections and two xmrig clients using non-encrypted flows.

The four obtained data sets were split in two separate subsets for training and testing purposes. Specifically, data sets from experiments 1 and 4 were joined in DS1 (training) data set and the other two data sets collected in experiments 2 and 3 were combined into DS2 (testing) data set. In this way, DS1 and DS2 can be considered to be of the same nature as they contain similar percentages of encrypted and non-encrypted traffic, types of internet services and cryptomining protocol flows. It is worth noting that regarding the small amount of traffic that cryptomining protocols generate when compared with normal traffic, an unbalance on the number of cryptomining flows against the normal traffic (1:5000) will

appear in all data sets, which poses an extra challenge for training ML classifiers.

The DS1 dataset (16 million of examples) was shuffled and split in two sets, training (85%) and validation (15%) for the hyperparameter optimisation. DS2 dataset (16 million of examples) was kept only for final testing purposes.

B. MODEL TRAINING

For each of our data sets, the first step towards a proper training and testing of a ML model is a feature selection phase. Traditionally, deep packet inspection techniques for traffic classification techniques acquire features from inspection of port numbers or by interpretation of packet payloads contents. However, these techniques tend to be increasingly limited due to the pervasive use of encryption methods to encapsulate packet contents (including TCP or UDP port numbers) and the governments increasingly introducing privacy regulations in order to constraint the access to packet payload data. To overcome these restrictions, more recent machine and deep learning techniques use flow descriptions as input to machine learning models. These descriptions are composed of a set of features, which are statistical data obtained from externally observable traffic attributes such as per-flow duration and volume, inter-packet arrival time, packet size and byte profiles [34].

Choosing informative, discriminating features is a crucial step for effective machine learning algorithms in different tasks such as classification. In particular, the performance of machine learning classifiers depends not only on the differences among models and their specific configuration, but also on the selection of input features. Nowadays, there is no strong consensus in research and industrial communities on a reliable set of features that can perform well with machine learning in all scenarios. Nevertheless, proposals based on time-based statistical features such as sizes and inter-packet times of the packets of a flow, or entropy of byte distribution in packet headers or payload, are gaining momentum ([35], [36]).

In the last years, network traffic analysis is facing emerging challenges related to the incursion of new, more sophisticated and unknown traffic. In particular, it is not easy to identify an adequate set of features that can help to obtain stable and highly performing machine learning models for new scenarios such as the cryptomining flow detection. For that reason we decided to compare two different sets of features with the aim to determine their impact in machine learning performance and to demonstrate that using more informative features, a significant increase in machine learning performance can be obtained. Both sets of features are not specifically tailored for criptomining attacks as they reflect parameters of a network flow and therefore they could be used in other network traffic scenarios (e.g. network traffic flow classification). As our goal was to detect cryptomining attacks on the network, flow (e.g. TCP connections) identification using flow features seems to be the adequate approach.

The first set of 51 features is proposed as a novel contribution of our work and was derived from a subset of the Tstat tool statistics.The set of these features is detailed in Table 1, where CS represents client-to-server and SC server-to-client traffic. The Tstat tool [37] allows the extraction of a large number of classification statistics from the transport and application layers and is not tailored for any specific type of network traffic or application.

Considering the initial assumption of not inspecting packet payloads, we will not use either IP addresses or port numbers or features derived from statistics of the application layer data. In addition, and in order to derive the set of features, we selected Tstat statistics using the following criteria: First, we did not consider Tstat statistics showing a constant value for all connections in different instants of time as they do not contain any information (e.g. c_rst_cnt, s_rst_cnt, c_isint, s_isint). Second, we applied a linear correlation analysis on data in order to identify and remove highly correlated variables. In this case, we followed a rather conservative approach when discarding a Tstat statistic using this criteria as the correlation analysis only showed linear correlations among variables and neural networks tend to discover interesting non-linear relationships among variables. Therefore, we did not remove any Tstat statistic using this criteria in order to give an opportunity to all machine learning models and in particular to neural network models to discover non-linear relationships among features. Finally, we applied our previous expertise in networks and protocols to discard Tstat statistics that could not be beneficial for the application of machine learning. In particular, statistics such as net_dup (number of network packet duplicates), reordering (number of packet reordering observed), rtx_RTO (Number of retransmitted segments due to timeout expiration) and rtx_FR (Number of retransmitted segments due to Fast Retransmit, three dup-ack) among others were removed as they represent statistics of network traffic congestion situations appearing in a specific period of time that may not appear at other times.

In addition, as a double-checking mechanism, we obtained feature importance values after training models with Random Forests models and we observed values ranging from 0.12 to 0.01 for nearly all features with the exception of s_tm_opt (Timestamp option sent by the server) that revealed a very small value of 0.0000184. This result emphasizes that there is not any subset of features that are more important than others, and on the contrary, all the selected Tstat derived features are contributing to the high performance of the complex models. It is worth noting that when deep neural networks are used, feature importance cannot be obtained so easily and therefore, the recommended approach is to provide all the input features to the neural network and let the optimization algorithm find by itself what features are the most informative. Regarding that interpretability and explainability of neural network decisions and their relationship with the input features is nowadays a topic under research, we decided to keep the set of initial 51 Tstat derived features as the input to all machine

TABLE 1. Tstat selected features.

CS	SC	Name	Metric	Description
3	17	packets	-	Total number of packets observed from the client/server
5	19	ACK sent	-	Number of segments with the ACK field set to 1
6	20	PURE ACK sent	-	Number of segments with ACK field set to 1 and no data
7	21	unique bytes	bytes	Number of bytes sent in the payload
8	22	data pkts	-	Number of segments with payload
9	23	data bytes	bytes	Number of bytes transmitted in the payload, including re-transmissions
31	-	Completion time	ms	Flow duration since first packet to last packet
32	-	C first payload	ms	Client first segment with payload since the first flow segment
33	-	S first payload	ms	Server first segment with payload since the first flow segment
34	-	C last payload	ms	Client last segment with payload since the first flow segment
35	-	S last payload	ms	Server last segment with payload since the first flow segment
36	-	C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
37	-	S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
45	52	Average rtt	ms	Average RTT computed measuring the time elapsed between the data segment and the corresponding ACK
46	53	rtt min	ms	Minimum RTT observed during connection lifetime
47	54	rtt max	ms	Maximum RTT observed during connection lifetime
48	55	Stdev rtt	ms	Standard deviation of the RTT
49	56	rtt count	-	Number of valid RTT observation
50	57	ttl_min	-	Minimum Time To Live
51	58	ttl_max	-	Maximum Time To Live
65	88	RFC1323 ws	0/1	Window scale option sent
66	89	RFC1323 ts	0/1	Timestamp option sent
67	90	window scale	-	Scaling values negotiated [scale factor]
68	91	SACK req	0/1	SACK option set
70	93	MSS	bytes	MSS declared
71	94	max seg size bytes	-	Maximum segment size observed
72	95	min seg size	bytes	Minimum segment size observed
73	96	win max	bytes	Maximum receiver window announced (already scale by the window scale factor)
74	97	win min	bytes	Maximum receiver windows announced (already scale by the window scale factor)

learning models in order to evaluate them under the same conditions.

The second set of features will be derived from the IETF standard NetFlow/IPFIX metrics. This set of eight features was recently proposed in [7] for detecting cryptomining activity using machine learning. These derived features are:

- inbound and outbound packets / second;
- inbound and outbound bits / second;
- inbound and outbound bits / packet;
- bits-inbound / bits-outbound ratio;
- packets-inbound / packets-outbound ratio.

Therefore, we evaluated our machine learning models with three sets of features:

- in *scenario A* the NetFlow/IPFIX metrics were utilised;
- in *scenario B* only the features derived from Tstat statistics were used;
- finally in *scenario C* both the Tstat derived and NetFlow/IPFIX features were jointly used.

After the feature selection phase we perform an hyperparameter optimisation stage in order to find the optimal hyperparameters to achieve a highly accurate model. Optimising hyperparameters is considered to be one of the trickiest parts of building machine learning models as it is virtually impossible to obtain the optimal parameters while building a model by simply guessing and testing several combinations of these values. On the other side, trying all the combinations of values for a set of hyperparameters is not scalable as the number of combinations to be tested grows exponentially with the number of hyperparameters and the ranges of the values to be tested in each of them. There exist several heuristics that can help to find these optimal hyperparameters, being random search one of the most popular and efficient approach. As its name suggests the evaluated combinations of hyperparameters are chosen at random from a hyperparameter multidimensional grid.

The basic strategy for random search consists in evaluating the validation score for each hyperparameter value combination and recording the results along with the hyperparameter combination. At the end of searching, the hyperparameters that yielded the highest validation score are chosen and the model is trained on all the training data set. Finally, predictions are made with this model on the test data set. Each time random search is run, the hyperparameter space is used as input and the algorithm produces a random combination of hyperparameter values to try. There are no requirements for random search other than that the next values are selected at random among the available ones in the hyperparameter space.

In our experiments we used a Random Search algorithm (RS) to find optimal combinations of hyperparameters values for training each model. After having trained each model a significant number of times and with a sufficient number of hyperparameter combinations, we selected the models with the highest F1-scores on validation. At least several dozens of different combinations were used for each machine learning model. Random search was stopped after observing that search results were stabilised (i.e. a better model did not

appear after several random search iterations) Table 2 shows the ranges and the hyperparameters that we optimised for each model.

Before running the random search we made some preliminary evaluations with different deep learning models based on Fully Connected neural network (FCNN) architectures. We run these preliminary evaluations using up to 10 hidden layers in FCNNs using forward bypass connections and batch normalisation in order to avoid vanishing gradients. It was observed that using a large number of layers (e.g. 8 to 10) did not produce better values than when we trained FCNNs with few layers (2 to 4). Therefore, during the random search we reduce the search range for FCNN layers from 1 to 5. Therefore, we trained and tested shallow (1-2 hidden layers) and deep (3-5 hidden layers) Fully Connected neural networks.

TABLE 2. Hyperparameter values.

Hyperparameter	Model	Values
L2 regularization	LR	$(10^{-3}, 100)$
	FCNN	$(10^{-6}, 10)$
Max depth	CART	(1, 200) or None
	RF	
# Trees	RF	(1, 200)
Pruning confidence	C4.5	$(10^{-2}, 1.0)$
# Min. instances per tree	C4.5	(2, 200)
Dropout	FCNN	$(10^{-6}, 0.7)$
Learning rate	FCNN	$(10^{-5}, 1.0)$
Class weight	FCNN	(1, 100)
# Layers	FCNN	(1, 5)
# Units per layer	FCNN	(1, 1000)

Note that *Class weight* hyperparameter was defined to manage the imbalanced nature of the datasets in the context of the cryptomining use case. As previously commented, the ratio of normal connections with respect to cryptomining ones is approximately 5,000, which negatively impacts to the final results of the training process. Therefore, different values of this parameter were tried during training in FCNN models to modulate the behaviour of the optimization algorithm. Random Forest and Logistic regression were trained using Scikit-learn library implementations, in which *class-weigth* parameter tries to cope with this problem. In addition, FCNN models used Rectified Linear Units (ReLU) as activations functions and therefore, no hyperparameter search was made on this parameter.

VIII. EXPERIMENTAL RESULTS

We carried out a set of experiments in order to evaluate the performance of several machine and deep learning models detecting cryptomining flows and differentiating them from normal traffic connections. Two complementary goals were defined at the beginning of the experiments. Firstly, to demonstrate that complex machine learning models (e.g. deep neural networks) perform better than simpler models and secondly, to show that the set of Tstat features (Table 1) we are proposing increases significantly the performance of these models when compared with previous feature proposals. As representatives of simpler models we selected the ML models proposed in [7] (SVM, Naive Bayes and CART and C4.5 trees) with the aim to compare their results with complex deep learning models in a more realistic scenario as the one deployed in our experiments. In addition, Random Forest was selected as it is widely accepted as one of the best performing ML techniques and Logistic Regression as the representative of a extremely simple method.

In the next paragraphs we describe the results that we obtained by using the data set creation and model training procedures detailed in Section VII.

A. MACHINE LEARNING SETUP

Our target problem aims at predicting cryptomining flows and differentiating them from other network flows. Therefore, this problem is modelled as a classifier where the expected output is a binary value: 1 corresponds to a cryptomining connection identified by the classifier whilst 0 is assigned to the rest of connections.

In order to be able to classify with accuracy these two types of connections, we trained and tested machine an deep learning classifiers based on the ones previously introduced in Section VI. In particular, we evaluated Fully Connected neural networks (FCNN) and compared against traditional machine learning techniques such as Logistic Regression (LR), CART and C4.5 decision trees (CART, C4.5) and Random Forest (RF). It is worth noting that in preliminary experiments we included SVM and Naive Bayes classifiers in the set of machine learning techniques to be evaluated. However, we observed that both techniques underperformed when compared with CART and C4.5 trees. Therefore, Naive Bayes and SVM were excluded from the experimental comparison we made with the rest of the models. In addition, SVM training times took two days using only a small percentage of the training dataset (1/100 of 16 millions of examples). As it can be seen in Table 6, the rest of the models took in the worst case no more than 40 minutes to train the model using the whole dataset (16 millions of examples).

We used the training procedure detailed in Section VII-B and for each model, the following steps were applied: 1) perform a feature selection phase; 2) run a random search on the hyperparameter space and train the model using the obtained hyperparameter combinations; 3) rank the best performing models using the F1-score on the validation set; 4) test the best models on the testing (DS2) dataset.

In order to compare and rank the obtained results in validation and testing, we compute a set of quality metrics widely used in classification problems ([38], [39]): accuracy, macro and micro versions of precision, recall and F1-score and confusion matrices. Additionally, ROC and P-R Areas under Curve (AUC) are calculated and their corresponding curves are plotted. These metrics allow us to measure the quality with which our models can identify and detect cryptomining flows among a heterogeneous set of Internet connections.

Traditional machine learning models (LR, CART and RF) were trained using the available Python code versions in the well-known scikit-learn library. Not having available a C4.5 implementation in the scikit-learn library, we utilised the J48 version of C4.5 that was included in the Weka tool [40]. Deep learning models (FCNN) were implemented in Python using Keras framework [41] over Tensorflow [42] to take advantage of GPU acceleration. All neural networks were trained using the Adam optimiser [27], minimising cross-entropy loss. In this case, the training process consisted of a sequence of training iterations. Each training iteration comprised 5 epochs and at the end of each iteration F1-score was computed on the validation set and used as early stopping criterion (i.e. after 10 iterations with no enhancement in the validation metric the training process stopped). In any case, the training stopped after 500 iterations. In addition note that when training FCNNs, a small amount of the training data (15%) was set aside and utilised for early stopping the number of epochs during the training process and therefore avoiding overfitting and local minimum stagnation.

B. RESULTS AND COMPARATIVE ANALYSIS OF MODELS

Using the random search procedure described previously and DS1 dataset, we trained and validated 15 model combinations of LR, CART, C4.5 and RF and 74 of FCNN for each scenario A, B and C. Having neural network models a greater number of hyperparameters, more hyperparameter combinations were required to find out a significant number of models with a decent performance.

The obtained results for scenarios A, B and C are shown in Tables 3, 4 and 5 respectively. In these tables, each column shows the testing results of the best representative for each model (FCNN, RF, CART, C4.5 and LR) on the testing dataset (DS2). The best representative for each model was selected among all model combinations using their ranked results in the validation dataset. Each model's representative is identified with a summary of its hyperparameters, including architectural details, that are described using a compact notation as in the following example:

```
#1/75 FCNN
[722, 847, 710]
0.001500, 0.000002
0.000012, 2;
F1 (macro) 0.9886)
```

The applied nomenclature is as follows:

- (1st line) #*X*/*Y*modelName: *X* and *Y* indicate respectively the model position in the test ranking and the total number of model combinations that were trained. Recall that models shown in tables are always the best models on validation. Therefore, a figure *X* close to 1 indicates that a model obtaining good performance in the validation phase, performs consistently when tested with a different DS2 dataset. Hence, we can be confident about the performance of a model combination in real-time production scenarios if the model was chosen among the top models on validation.
- FCNN models add the following values:
  - (2nd line) A list of values describing the neural network architecture: $[X_0, X_1, \ldots, X_i, \ldots, X_n]$ where n is the number of layers and X_i is the number of units in the layer i.
  - (3rd and 4th lines) Dropout, L2 regularization value, Adam learning rate and the value for the *Class weight* (i.e. the weight of the *cryptomining flow* class with respect to the *normal traffic flow* class as defined in Table 2).
- Random Forest models include:
  - The number of trees
  - The Maximum *depth limit* for trees.
- Logistic Regression models show the applied regularization coefficient.
- CART models indicate the maximum depth for trees.
- C4.5 models include the confidence value (C) and the minimum number of instances in the two most popular branches (M).
- (last line) All models show the F1-score (macro) value obtained in validation.

Tables 3, 4 and 5 show for each model the quality metrics computed on the testing dataset. F1-score, Precision and Recall macro values are presented jointly with Accuracy and RoC and P-R Areas Under Curve (AUC ROC and AUC P-R). Last row shows the Confusion Matrix in which the values of cyptomining connections are disaggregated in two figures for a better analysis. *SSL* value represents the number of cryptomining encrypted flows and *NoSSL* represents the non-encrypted ones.

From a global perspective, it can be observed that: a) complex models -deep neural networks (FCNN), Random Forest and C4.5- obtain much better results than simpler strategies - CART and LR- in all scenarios; and b) the best results were obtained in Scenario B, which proposes to utilise Tstat features as input to machine learning models. Furthermore, Scenario C, which utilised a combination of Tstat and NetFlow features, did not obtain any observable advantage with respect to use only Tstat features (Scenario B).

Random Forest (RF) and C4.5 are the only models that obtain decent results even in the context of Scenario A. Using a less expressive set of features (NetFlow features), RF obtained a satisfactory F1-score value of 0.9724 that is however worse than the one obtained in Scenarios B and C (0.9964 and 0.9957). RF superior performance when compared with simpler approaches as CART decision trees is explained by the usage of boosting techniques that make it more resistant to problems such as noise [22]. It can be observed that 77 flows of normal traffic were classified as cryptomining (false positives) in Scenario A but only 31 and 25 in Scenarios B and C. However 393 false negatives (cryptomining flows that are not detected) were produced in Scenario A, which is more than 10 times the number of false

TABLE 3. Scenario A: Overall results with the testing dataset (DS2 dataset) of the best models chosen based on the validation dataset using Netflow feature selection method.

	Models				
Metrics on Testing	**#2/74 FCNN** [903, 951] 0.000905, 0.000040 0.000036, 43 F1 (macro) 0.6506 on validation	**#2/16 Random Forest** 53 trees 77 depth limit F1 (macro) 0.9823 on validation	**#1/23 C4.5** 0.4316 C 22 M F1 (macro) 0.9755 on validation	**#6/15 CART** 132 depth limit F1 (macro) 0.9739 on validation	**#6/16 Logistic Regression** 6.65216 reg. F1 (macro) 0.4999 on validation
F1 (macro)	0.6387	0.9724	0.9729	0.8970	0.4999
Prec (macro)	0.9448	0.9906	0.9875	0.9607	0.4999
Recall (macro)	0.5822	0.9556	0.9590	0.8488	0.5000
Accuracy	0.9998	1.0000	1.0000	0.9999	0.9997
AUC ROC	0.9826	0.9962	0.9798	0.8488	0.9411
AUC P-R	0.1388	0.9816	0.8948	0.6746	0.0023
Confusion matrix SSL/NoSSL 3007/1417	[16055077 90] [3003/694 4/723]	[16055090 77] [347/46 2660/1371]	[16055090 104] [314/48 2693/1369]	[16054904 263] [1218/120 1789/1297]	[16055167 0] [3007/1417 0/0]

TABLE 4. Scenario B: Overall results with the testing dataset (DS2 dataset) of the best models chosen based on the validation dataset using Tstat feature selection method.

	Models				
Metrics on Testing	**#2/75 FCNN** [722, 847, 710] 0.001500, 0.000002 0.000012, 2 F1 (macro) 0.9873 on validation	**#2/24 Random Forest** 25 trees No depth limit F1 (macro) 0.9967 on validation	**#4/16 C4.5** 0.2193 C 73 M F1 (macro) 0.9950 on validation	**#6/15 CART** No depth limit F1 (macro) 0.9975 on validation	**#1/16 Logistic Regression** 10.5626 reg. F1 (macro) 0.9784 on validation
F1 (macro)	0.9902	0.9964	0.9948	0.6767	0.9805
Prec (macro)	0.9977	0.9965	0.9933	0.6076	0.9881
Recall (macro)	0.9829	0.9963	0.9964	0.9964	0.9732
Accuracy	1.0000	1.0000	1.0000	0.9990	1.0000
AUC ROC	0.9958	0.9997	0.9999	0.9978	0.9998
AUC P-R	0.9673	0.9958	0.9852	0.6043	0.9677
Confusion matrix SSL/NoSSL 3007/1417	[16055147 20] [118/33 2889/1384]	[16055136 31] [24/9 2983/1408]	[16055107 60] [25/7 2982/1410]	[16039135 16032] [18/9 2989/1408]	[16055065 102] [190/47 2817/1370]

TABLE 5. Scenario C: Overall results with the testing dataset (DS2 dataset) of the best models chosen based on the validation dataset using Netflow and Tstat feature selection method.

	Models				
Metrics on Testing	**#1/75 FCNN** [722, 847, 710] 0.001500, 0.000002 0.000012, 2 F1 (macro) 0.9886 on validation	**#2/24 Random Forest** 53 trees 77 depth limit F1 (macro) 0.9962 on validation	**#1/16 C4.5** 0.0236 C 136 M F1 (macro) 0.9929 on validation	**#3/16 CART** 132 depth limit F1 (macro) 0.9963 on validation	**#2/16 Logistic Regression** 10.8882 reg. F1 (macro) 0.9789 on validation
F1 (macro)	0.9910	0.9957	0.9921	0.6762	0.9789
Prec (macro)	0.9998	0.9972	0.9891	0.6073	0.9837
Recall (macro)	0.9826	0.9943	0.9953	0.9953	0.9741
Accuracy	1.0000	1.0000	1.0000	0.9990	1.0000
AUC ROC	0.9968	0.9997	0.9999	0.9966	0.9998
AUC P-R	0.9666	0.9963	0.9943	0.6027	0.9675
Confusion matrix SSL/NoSSL 3007/1417	[16055165 2] [120/34 2887/1383]	[16055142 25] [38/12 2969/1405]	[16055069 98] [18/24 2989/1393]	[16039112 16055] [24/13 2983/1404]	[16055026 141] [182/47 2825/1370]

negatives observed in Scenarios B and C (33 and 50). This means that using only Netflow features, RF fails to detect 10 times the number of cryptomining flows not detected when using Tstat features (Scenarios B and C). Regarding that one of the main goals of this work is to provide machine learning methods for detecting with accuracy as many (malware) cryptomining flows as possible, this result demonstrates that ML models using only Netflow features (Scenario A) seem to fail in their purpose when compared with ML models using Tstat features (Scenarios B and C). The best results with RF were obtained with a small of estimators (between 25 and 100 trees) and with modest tree depth limits (between 40 and 200 with one case of no depth limit). The best C4.5 model obtained nearly identical F1-score and confusion matrix values and therefore, we can conclude that their performance is roughly the same in the three scenarios.

FCNN deep models performed in a similar range than RF in Scenarios B and C, obtaining F1-score (macro) values of 0.9902 and 0.9910 respectively and approximately equal figures for false positives and negatives. From an architectural point of view, only three hidden layers with approximately 700-800 neurons in each layer were needed in order to obtain the best performance. However, FCNN produced significantly bad results when using NetFlow features (Scenario A). The best configuration only achieved a F1-score value of 0.6387 reflecting the appearance of a big number of false positives of the cryptomining class (3697) with respect to the true positives detected for this class (727). In addition, note that the number of false positives was significantly smaller in the case of RF (393). We conjecture that the existing class unbalance jointly with the moderate quality of Netflow features did not allowed the NN optimization algorithm to reduce in similar proportions the number of false negatives and positives and to increase the true positives of the cryptomining class.

In all three scenarios, CART models showed greater F1-score values in training (0.9739, 0.9975 and 0.9963) than in testing (0.8970, 0.6767 and 0.6762), which clearly indicates that these models tend to overfit training datasets. In addition, a extremely high number of false negatives (1338) and false positives (16032 and 16055) were observed respectively in scenario A and in scenarios B and C. Therefore, we would discourage CART usage for detecting cryptomining flows in production environments due to the lack of generalization and the high number of false positives and negatives that these models exhibited in all scenarios. It is interesting to note that these models seem to generalise slightly better when only Netflow features are used as in Scenario A (0.8970 F1-score in testing). We conjecture that Netflow features utilised in Scenario A provide less information during training than Tstat ones and therefore, the resultant overfitting is not so severe than in Scenarios B and C.

Logistic Regression models could only be correctly trained with Tstat features (Scenarios B and C), obtaining decent F1-score values of 0.9805 and 0.9789 respectively. However, in Scenario A, LR models were not able to identify any cryptomining flow as all flows were classified as normal traffic. Confusion matrices of Scenarios B and C show a greater number of false positives and negatives than RF and FCNN. This fact could preclude the usage of these simple models in cybersecurity scenarios in which the reduction of false positives and negatives is essential. It is worth noting that using a significant regularization, this model generalised much better than CART models when Tstat features were used (Scenarios B and C).

Regarding that the ratio of encrypted (SSL) versus non-encrypted (NoSSL) cryptomining connections in the testing dataset is approximately 2:1 - figures can be found in *Confusion Matrix* cells in the left bottom corner in Tables 3, 4 and 5 -, the proportion of false positives with SSL network traffic in Scenario A is more than two times the original ratio: this ratio is approximately 7:1 in RF, 6.5:1 in C4.5, 10:1 in CART and 4:1 in FCNN. For Logistic Regression this value is irrelevant because the model failed to classify anything as all traffic was predicted as class 0 (Normal traffic). When Tstat features were used in scenarios B and C, the SSL/NoSSL ratio of false positives in cryptomining flows was 2:1 for Random Forest and CART models, 3:1 for C4.5, approximately 3:1 for FCNN model and 4:1 for Logistic Regression model, which clearly indicates that Tstat features helped to consistently identify cryptomining flows even when they were encrypted.

Figures 6 and 7 clearly show that RF C4.5, and FCNN models exhibit very good separability values for the decision function in ROC and P-R curves when Tstat features are used, as their areas under the ROC/P-R curves are both above 0.99 for RF and 0.98 for FCNN and C4,5. Note that P-R curves tend to be rather sensitive to unbalanced distributions as the ones we are considering in our experiments. In this context, it can be observed in the P-R figures that CART trees do not achieve a good separability behaviour with small AUC values of 0.6 and curve shapes very close to the diagonal. It is worth noting that when Netflow features are used (figure 5) only RF and C4.5 showed decent separability results and conversely, FCNN generated a highly unstable behaviour as shown in a P-R curve that abruptly went to zero both on the left and on the right due to precision reaching values very close to zero. This instability was caused because (a) the number of true positives (for the cryptomining class) went to zero when the decision threshold was greater or equal than 0.6 (and therefore, precision was also zero), and (b) the number of false positives abruptly moved from 86 to 205,048 when decision threshold was less than 0.5, being the number of true positives around 700 (and therefore, precision was approximately zero).

At the light of these results we can conclude that (a) the set of Netflow features used in Scenario A produced less accurate models than when Tstat features were selected as input (Scenarios B and C), and (b) RF, C4.5 and FCNN models consistently obtained better results than simpler models such as CART and Logistic Regression. In particular, the best results were obtained using Tstat features and RF, C4.5 and FCNN models. These models achieved a F1-macro value of

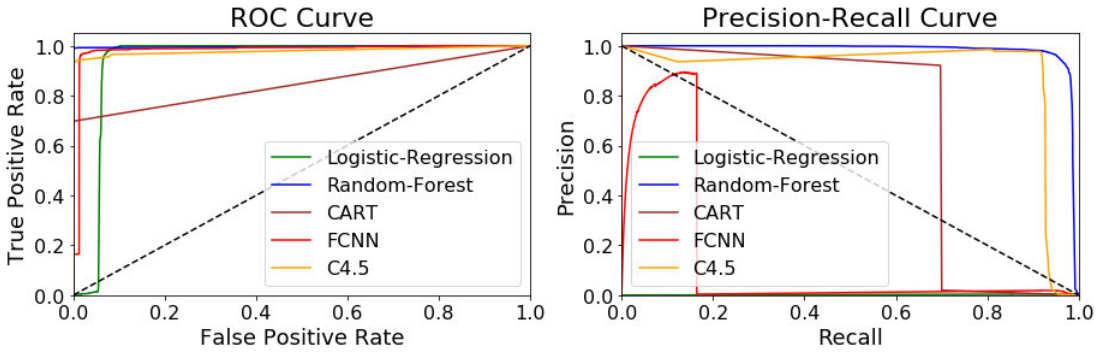


FIGURE 5. ROC and P-R curves for the testing dataset of the best models chosen based on the validation dataset. Features extraction method "NetFlow".

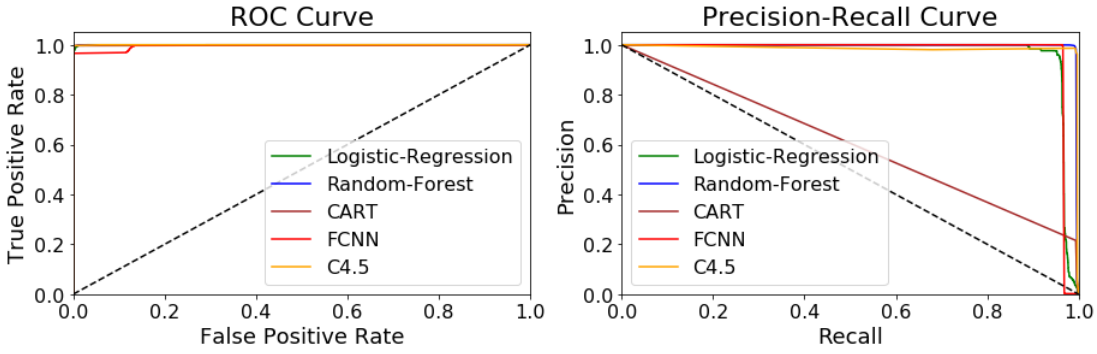


FIGURE 6. ROC and P-R curves for the testing dataset of the best models chosen based on the validation dataset. Features extraction method "Tstat".

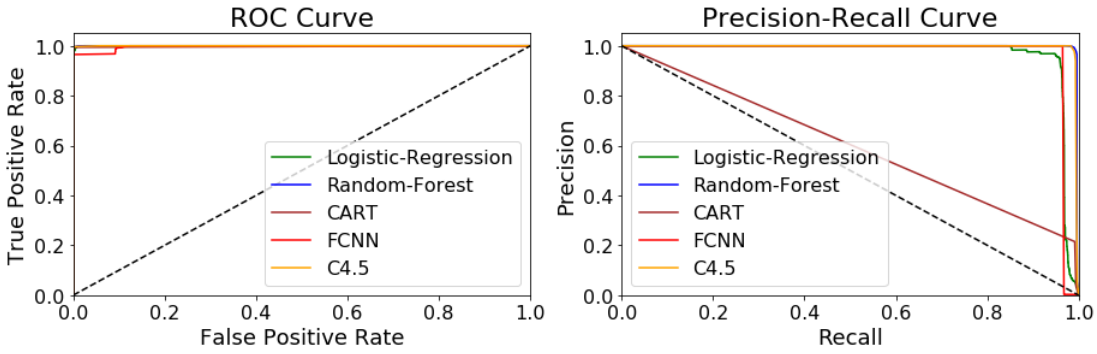


FIGURE 7. ROC and P-R curves for the testing dataset of the best models chosen based on the validation dataset. Features extraction method "Combined".

approximately 0.99 reflecting that only a very small number of false positives and negatives were produced - less than 4% of the total number of cryptomining flows -. Conversely, using only Netflow features the number of false negatives tripled the true positives which highlighted a lack of precision when identifying cryptomining flows. Additionally, the ratio of SSL/Non_SSL cryptomining flows appearing as false negatives was greater than in the whole set of cryptomining flows, which reflects the inability of these features for the precise identification of encrypted (SSL) cryptomining flows. Future scenarios, in which more and more cryptomining flows are expected to be encrypted, will not suggest the solely utilisation of this limited set of features.

Finally, it is worth noting that the high accuracy of the best models was not achieved at the expense of long training sessions. It is the combination of highly informative features (i.e. Tstat features) with complex machine learning models that achieves high values of accuracy, precision and recall. Regarding that the same training processes were applied for the three feature sets, it can be observed that the performance

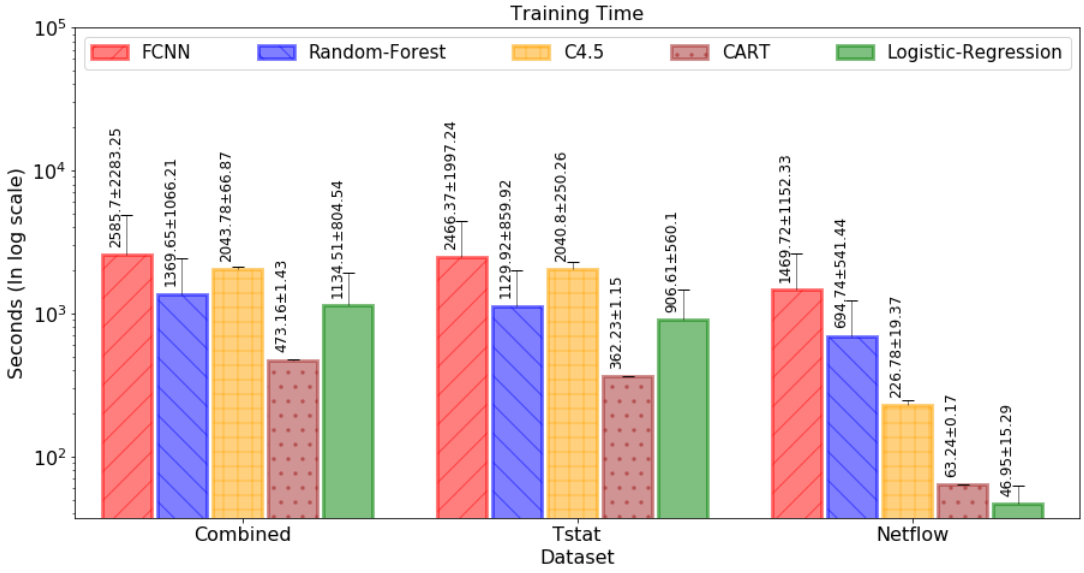


FIGURE 8. Models training time computed with 10 random search executions on an PC workstation.

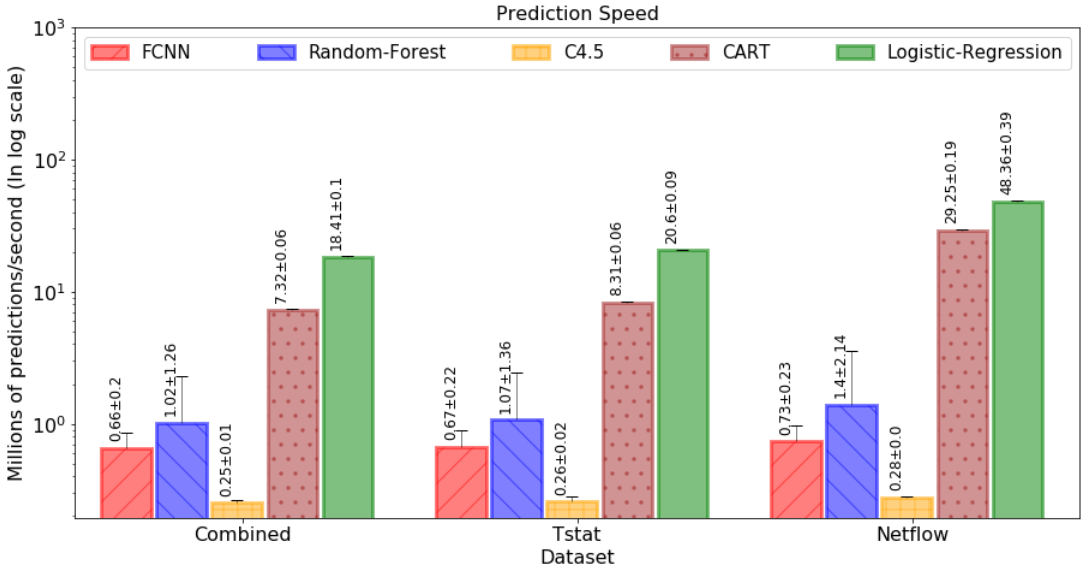


FIGURE 9. Models prediction speed computed with 10 random search executions on an PC workstation.

decreased significantly when we used less informative features (Netflow feature set). Figures 8 and 9 present training times and prediction speeds of each model (mean and standard deviation are plotted in the figures). A modest PC workstation with an Intel i5-9400F at 2.90 GHz with 6 threads cpu, 64 Gbytes of RAM and equipped with a GTX1080 GPU was used for training and testing. It can be observed that the training times for all models are always below 40 minutes for 16 millions of examples and tend to be proportional to the number of input features. Similarly, predictions are inversely proportional to the model complexity and in the range of millions per second using the same hardware. Note that C4.5 was run using the J48 version available in the Weka tool and therefore, its prediction performance was significantly worse than the rest of the models that were coded directly in Python.

C. RESULTS SEGREGATED BY NUMBER OF PACKETS AND CONNECTION DURATION

With the aim to study how stable the predictions of a model are along the whole life of flows, we present the previous testing results of the best performing configurations in validation for each model and scenario segregated by ranges of transmitted packets and flow duration. In Figure 10 we

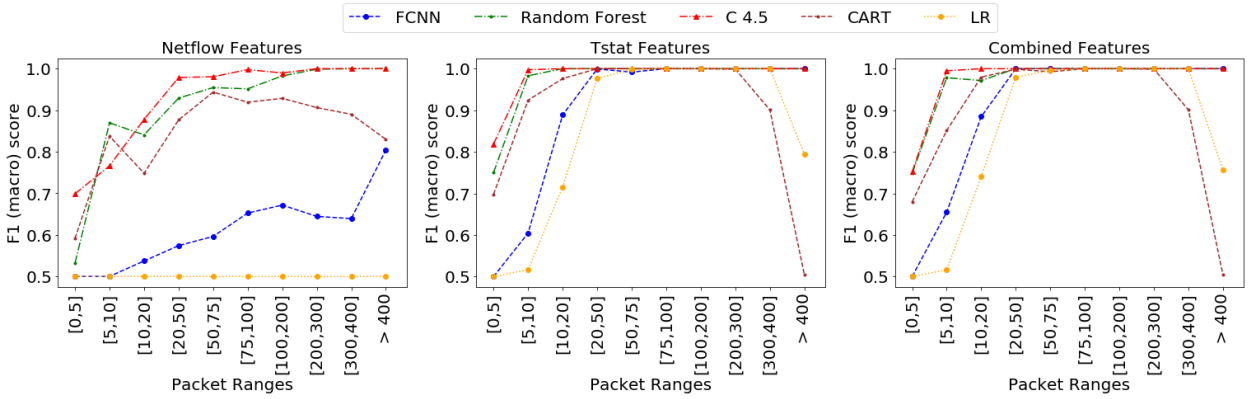


FIGURE 10. F1 (macro) score on testing segregated by packet ranges.

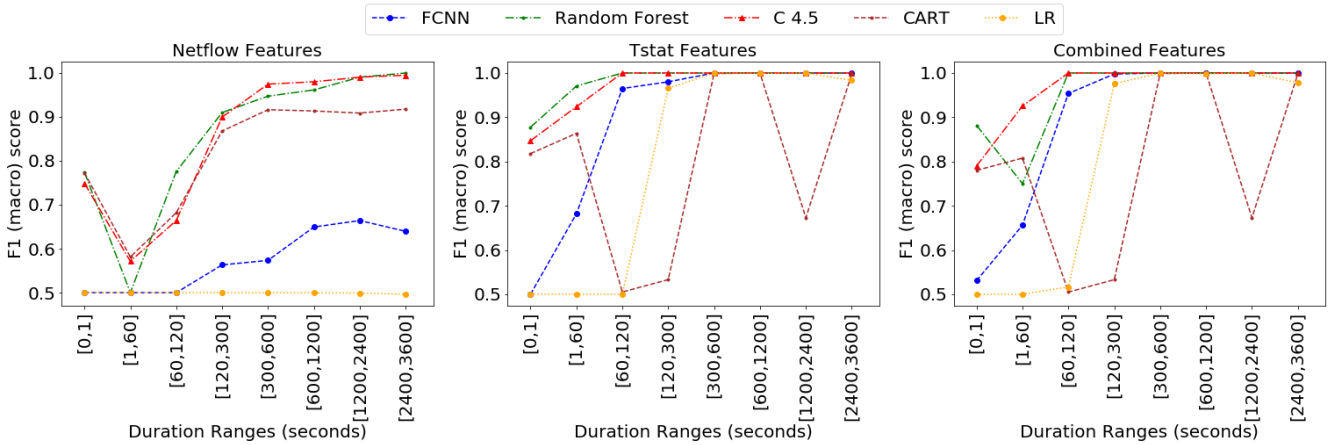


FIGURE 11. F1 (macro) score on testing segregated by connection duration ranges.

TABLE 6. Number of network flows (total number and percentage over the total) and number of cryptomining flows segregated by packet ranges.

Packets Ranges	Flows	Percentage of Flows	Crypto Flows
[0,5]	762,740	4.74	60
[5,10]	688,572	4.28	60
[10,20]	1,010,592	6.29	120
[20,50]	1,612,818	10.04	360
[50,75]	745,114	4.63	300
[75,100]	595,347	3.70	300
[100,200]	1,508,626	9.39	1,200
[200,300]	896,383	5.58	1,200
[300,400]	675,591	4.20	753
> 400	7,563,808	47.09	71

TABLE 7. Number of network flows (total number and percentage over the total) and number of cryptomining flows segregated by connection duration in seconds.

Connection Duration Ranges (Seconds)	Flows	Percentage of Flows	Crypto Flows
[0,1]	5,013,517	31.21	124
[1,60]	9,347,001	58.20	27
[60,120]	284,120	1.76	59
[120,300]	279,859	1.74	216
[300,600]	272,795	1.69	352
[600,1200]	373,141	2.32	654
[1200,2400]	389,923	2.42	1,360
[2400,3600]	99,071	0.61	1,468

present the testing metrics computed for the networks flows contained in the testing dataset, grouping flows by ranges of transmitted packets. In this way, we can determine how many flow packets are needed to be transmitted in order to be able to predict with a specific precision whether the flow is normal traffic or a cryptomining connection. Additionally, we show metrics for the networks flows in the testing dataset grouping flows by periods of time measured in seconds in Figure 11.

These metrics allow us to estimate how many seconds are needed to identify with a certain precision whether a flow is a cryptomining or a normal connection.

Figure 10 shows the macro F1-Score values obtained in testing for the best performing models' configurations in validation for scenarios A, B and C. The results are segregated by intervals of packets that represent the sum of all packets sent from a client to a server and from a server to a client during a connection. Note that the establishment of a TCP connection involves the transmission of 3 packets and the set up of an encrypted new SSL connection involves the transmission of at least 4 additional packets (without resumption). Therefore, it is practically impossible to identify whether a SSL connection carries a normal or a cryptomining flow with less than 7 packets. Moreover, for a non-encrypted TCP connection we need at least 4 packets to identify it as the first three packets are common to all (encrypted and non encrypted) TCP connections. Recall that, as a basic assumption of our analysis, we did not use as input features either IP addresses or ports. Table 6 shows the number of flows included in each range of packets as absolute value and percentage of total jointly with the number of cryptomining flows falling in this interval.

Analyzing the obtained results using Netflow features, it can be observed that neither model achieves good results in any interval with the exception of RF and C4.5 that need at least 200 packets to be transmitted in order to obtain a F1-score greater than 0.99. This result of RF and C4.5 models is not very useful in practice considering that the transmission rate of cryptomining flows is extremely low if compared with normal traffic rates and therefore, it would take approximately 30-45 minutes from its creation to identify a cryptocurrency flow with high precision (F1-score of 0.99). In addition, CART model exhibits a significant lack of stability because F1-score starts decreasing in flows with more than 300 packets. However, F1-score values in FCNN, C4.5 and RF models show a monotonically increasing behaviour and never destabilise as CART model does. Logistic Regression could not be trained using these features and therefore, an F1-score of 0.5 was obtained consistently in all intervals. Conversely, by using Tstat features FCNN, C4.5 and RF obtain an F1-score of 1 (100% of accuracy, precision and recall) after a small number of packets (10 for RF and C4.5 and 75 for FCNN). CART and LR also achieved an F1-score of 1 with few packets but as in the former scenario, they destabilise (decreasing their F1-score) when the number of packets increased (200 packets in CART and 400 packets in LR). Results using combined Netflow and Tstat features in Scenario C show similar results than in Scenario B with a slight enhancement in the case of FCNN that achieved an F1-score of 1 with only 20 packets.

Figure 11 shows the F1-score (macro) results per model and segregated by time intervals. In each time interval we include flows that have been alive since their establishment a number of seconds falling in such range. Therefore, the same flow can be sampled at different instants of time with its features appearing several times in the same dataset and in different time intervals. For example, a TCP connection with a total duration of 200 seconds, will appear as different flow occurrences in the intervals [0,1], [1,60], [60,120] and [120,300]. Table 7 shows the total number and percentage of flows jointly with the number of cryptomining flows appearing in each interval.

Reported results in Figure 11 are similar to the ones obtained when segregating results by packet ranges. When using Netflow features, only the RF and C4.5 models behave nicely but achieving an F1-score of .999 only when the flow duration is greater than 2400 seconds. Note that in the set of flows that last for 1200 seconds or more, the percentage of cryptomining connections is much greater than in the rest of intervals including flows with a shorter existence (Table 7). This significant difference is explained by the fact that cryptomining protocols tend to set up connections for long periods of time which are not so frequent in normal traffic flows. Therefore, the classification with a greater precision of long-lasting flows is easier than in the rest of time intervals. Using Tstat features, RF and C4.5 obtain perfect classification (F1-score of 1) after 60 seconds and FCNN needs some extra time (300 seconds). CART and Logistic Regression exhibit the previously commented destabilisation when the flows are long-lasting. When both set of features are combined in Scenario C no significant differences appeared when compared with models using only Tstat features.

At the light of the obtained results when flows are segregated by packet or time intervals, we can conclude that (a) the best results (F1-score of 1) are achieved using Tstat features and in the case of RF and C4.5 models as soon as the flow has transmitted 10 packets or after having passed only 60 seconds from the flow creation and (b) RF, C4.5 and FCNN models do not destabilise even when only Netflow features are used (i.e., in these models, F1-score increases monotonically with the number of packets transmitted in the flow. However, CART and Logistic Regression models exhibited an unstable behavior when the number of packets in flows reached 200 packets in CART and 400 in LR, or when flows are alive more than 1200 seconds in CART and 2400 in LR, which might preclude their usage in real production environment.

IX. CONCLUSION AND FUTURE WORK

We designed, trained and tested a set of machine and deep learning models for detecting cryptomining activity. We selected several complex models such as Deep Neural Networks, Random Forest and C4.5 in order to compare their performance with known research. As a novelty with respect to other proposals and in order to evaluate machine learning performance in realistic scenarios, encrypted and non-encrypted flows of normal and cryptomining traffic were considered in our experiments. As a main contribution, we proposed a new set of highly informative features derived from the Tstat tool statistics to test whether using highly informative features can increase machine learning

performance of complex models. We used the Tstat tool to derive this set of 51 features per flow and a second set of 8 features was extracted from IETF standart NetFlow/IPFIX metrics.

We set up a controlled network environment in the Mouseworld Lab at Telefonica R&D premises, using real clients and servers to generate realistic experiments with cryptomining traffic competing with normal traffic (web surfing, video and audio streaming, cloud storage, file transfer, email and P2P among others) over a dedicated network with Internet connectivity. Cryptomining clients were connected to real mining pools in the Internet and Monero was chosen as the cryptocurrency of our experiments as it is by far the favorite cryptocurrency among cybercriminals. Monero clients run the Stratum protocol, which is the most widespread protocol used to distribute the mining process. Both encrypted and plain connections were established not only for the Stratum protocol but also for the rest of traffic in order to generate realistic traffic traces. The traffic generated by the experiments was utilised for two different tasks. Firstly, traffic was captured, labelled and stored for training supervised machine learning models. Later, traffic was input in real time to machine learning models for testing purposes. Three scenarios were configured, the first using the Netflow derived features, the second only with Tstat derived features and the last joined both sets of features.

The best results were obtained using the set of features derived from Tstat and it is worth noting that joining the two sets of features (Tstat + Netflow) we did not obtain any observable advantage with respect to using only Tstat features. Conversely, the set of Netflow derived features, which represent the current state of the technique, produced instability and bad performance in predictions and therefore, we suggest that industry should adopt more exhaustive features when using machine and deep learning in order to obtain the expected performance and stability. Furthermore, when Random Forest, C4.5 and Deep Neural Networks were used in conjunction with Tstat features, the detection of encrypted cryptomining connections was done with a similar performance than with plain (non-encrypted) connections. On the other hand, when simpler machine learning models or Netflow features were used, the detection of encrypted cryptomining flows suffered from a lack of precision and accuracy that was proportionally greater than the obtained with non-encrypted cryptomining flows.

In the light of the obtained results it can be concluded that our proposal of using sufficiently exhaustive features as input to complex machine learning models, such as Random Forest, C4.5 or Fully Connected Deep Neural networks, allows to deploy precise, accurate and stable mechanisms for detecting cryptomining activity as required by industry.

Future work will extend current research with new experiments using different cryptomining protocols and cryptocurrencies, which could imply the design of more complex models to learn new patterns of cryptomining activity.

REFERENCES

[1] J. Lewis, *The Economic Impact of Cybercrime—No Slowing Down*. Santa Clara, CA, USA: McAfee, 2018.

[2] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. [Online]. Available: https://git.dhimmel.com/bitcoin-whitepaper/.

[3] Deadcoins. *Deadcoins Curated List of Cryptocurrencies and ICOs*. Accessed: Feb. 23, 2020. [Online]. Available: https://deadcoins.com

[4] L. Ren and S. Devadas, "Bandwidth hard functions for ASIC resistance," in *Theory of Cryptography*, (Lecture Notes in Computer Science), vol. 10677. Cham, Switzerland: Springer, 2017, pp. 466–492.

[5] GithubRepository, Tevador. (2019). *RandomX: Experimental Proof of Work Algorithm Based on Random Code Execution*. [Online]. Available: https://github.com/tevador/RandomX

[6] S. Pastrana and G. Suarez-Tangil, "A first look at the crypto-mining malware ecosystem: A decade of unrestricted wealth," in *Proc. Internet Meas. Conf.*, Oct. 2019, pp. 73–86.

[7] J. Z. I. Munoz, J. Suarez-Varela, and P. Barlet-Ros, "Detecting cryptocurrency miners with NetFlow/IPFIX network measurements," in *Proc. IEEE Int. Symp. Meas. Netw. (M&N)*, Jul. 2019, pp. 1–6.

[8] S. Haber and W. S. Stornetta, "How to time-stamp a digital document," *J. Cryptol.*, vol. 3, no. 2, pp. 99–111, Jan. 1991.

[9] Braiins Systems. *Stratum V2 | The Next Generation Protocol for Pooled Mining*. Accessed: Jan. 23, 2020. [Online]. Available: https://stratumprotocol.org/

[10] Braiins Systems. *Stratum Mining Protocol*. Accessed: Jan. 23, 2020. [Online]. Available: https://slushpool.com/help/topic/stratum-protocol/

[11] JSON-RPC Working Group. *JSON-RPC 2.0 Specification*. Accessed: Feb. 23, 2020. [Online]. Available: https://www.jsonrpc.org/specification

[12] R. Recabarren and B. Carbunar, "Hardening stratum, the bitcoin pool mining protocol," *Proc. Privacy Enhancing Technol.*, vol. 2017, no. 3, pp. 57–74, Jul. 2017.

[13] A. Swedan, A. N. Khuffash, O. Othman, and A. Awad, "Detection and prevention of malicious cryptocurrency mining on Internet-connected devices," in *Proc. 2nd Int. Conf. Future Netw. Distrib. Syst. (ICFNDS)*, 2018, pp. 23:1–23:10.

[14] R. K. Konoth, E. Vineti, V. Moonsamy, M. Lindorfer, C. Kruegel, H. Bos, and G. Vigna, "MineSweeper: An in-depth look into drive-by cryptocurrency mining and its defense," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1714–1730.

[15] D. Carlin, P. OrKane, S. Sezer, and J. Burgess, "Detecting cryptomining using dynamic analysis," in *Proc. 16th Annu. Conf. Privacy, Secur. Trust (PST)*, Aug. 2018, pp. 1–6.

[16] J. Liu, Z. Zhao, X. Cui, Z. Wang, and Q. Liu, "A novel approach for detecting browser-based silent miner," in *Proc. IEEE 3rd Int. Conf. Data Sci. Cyberspace (DSC)*, Jun. 2018, pp. 490–497.

[17] 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 *Proc. World Wide Web Conf. (WWW)*, 2019, pp. 840–852.

[18] J. Berkson, "Application of the logistic function to bio-assay," *J. Amer. Stat. Assoc.*, vol. 39, no. 227, pp. 357–365, Sep. 1944.

[19] J. Cramer, "The origins of logistic regression," Tinbergen Inst. Discuss. Papers 02-119/4, 2002.

[20] D. Steinberg, "CART: Classification and regression trees," in *The Top Ten Algorithms Data Mining*, X. Wu and V. Kumar, Eds. London, U.K.: Chapman & Hall, 2009, ch, 10.

[21] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Amsterdam, The Netherlands: Elsevier, 2014.

[22] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[23] T. En-Najjary, G. Urvoy-Keller, M. Pietrzyk, and J.-L. Costeux, "Application-based feature selection for Internet traffic classification," in *Proc. 22nd Int. Teletraffic Congr. (ITC)*, Sep. 2010, pp. 1–8.

[24] T. En-Najjary and G. Urvoy-Keller, "A first look at traffic classification in enterprise networks," in *Proc. 6th Int. Wireless Commun. Mobile Comput. Conf. ZZZ (IWCMC)*, 2010, pp. 764–768.

[25] Y. Wang and S.-Z. Yu, "Machine learned real-time traffic classifiers," in *Proc. 2nd Int. Symp. Intell. Inf. Technol. Appl.*, vol. 3, Dec. 2008, pp. 449–454.

[26] J. Li, S. Zhang, Y. Xuan, and Y. Sun, "Identifying skype traffic by random forest," in *Proc. Int. Conf. Wireless Commun., Netw. Mobile Comput.*, Sep. 2007, pp. 2841–2844.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980
[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.
[29] A. Saied, R. E. Overill, and T. Radzik, "Detection of known and unknown DDoS attacks using artificial neural networks," *Neurocomputing*, vol. 172, pp. 385–393, Jan. 2016.
[30] A. Pastor, A. Mozo, D. R. Lopez, J. Folgueira, and A. Kapodistria, "The mouseworld, a security traffic analysis lab based on NFV/SDN," in *Proc. 13th Int. Conf. Availability, Rel. Secur. (ARES)*, 2018, pp. 57:1–57:6.
[31] fireice-uk. *FIREICE-UK/XMR-STAK: Free Monero RandomX Miner and Unified CryptoNight Miner*. Accessed: Dec. 28, 2019. [Online]. Available: https://github.com/fireice-uk/xmr-stak/
[32] Xmrig Team. *XMRIG/XMRIG: RandomX, CryptoNight, AstroBWT and Argon2 CPU/GPU Miner*. Accessed: Jan. 23, 2020. [Online]. Available: https://github.com/xmrig/xmrig
[33] The Monero Project. *Home | Monero-Secure, Private, Untraceable*. Accessed: Feb. 23, 2020. [Online]. Available: https://www.getmonero.org/
[34] T. T. T. Nguyen and G. Armitage, "A survey of techniques for Internet traffic classification using machine learning," *IEEE Commun. Surveys Tuts.*, vol. 10, no. 4, pp. 56–76, 4th Quart., 2008.
[35] A. Dainotti, A. Pescape, and K. Claffy, "Issues and future directions in traffic classification," *IEEE Netw.*, vol. 26, no. 1, pp. 35–40, Jan. 2012.
[36] A. Habibi Lashkari, G. Draper Gil, M. S. I. Mamun, and A. A. Ghorbani, "Characterization of tor traffic using time based features," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 253–262.
[37] A. Finamore, M. Mellia, M. Meo, M. M. Munafo, P. D. Torino, and D. Rossi, "Experiences of Internet traffic monitoring with tstat," *IEEE Netw.*, vol. 25, no. 3, pp. 8–14, May 2011.
[38] G. Shobha and S. Rangaswamy, "Machine learning," in *Computational Analysis and Understanding of Natural Languages: Principles, Methods and Applications* (Handbook of Statistics), vol. 38, V. Gudivada and C. Rao, Eds. Amsterdam, The Netherlands: Elsevier, 2018, ch. 8, pp. 197–228.
[39] A. Mozo, I. Segall, U. Margolin, and S. Gomez-Canaval, "Scalable prediction of service-level events in datacenter infrastructure using deep neural networks," *IEEE Access*, vol. 7, pp. 179779–179798, 2019.
[40] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: An update," *ACM SIGKDD Explor. Newslett.*, vol. 11, no. 1, pp. 10–18, 2009.
[41] F. Chollet. (2015). *Keras*. [Online]. Available: Available: https://keras.io
[42] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, and M. Kudlur, "Tensorflow: A system for large-scale machine learning," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 265–283.

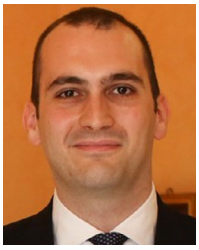
ANTONIO PASTOR received the M.Sc. degree in industrial engineering from the University Carlos III of Madrid (UC3M), Spain, in 1999. Since then, he has been with Telefónica I+D (Research and Development), Spain, where he works on the engineering of different worldwide Telefónica networks. From 2006 to 2011, he worked as an Expert in network security for Telefónica. Since 2012, as a part of the Telefónica Global CTIO, he has been leading innovation activities in network security based on network virtualization, SDN, and artificial intelligence, where he has produced several patents and scientific articles. Additionally, he holds several certifications from ISACA and GIAC in this area.

ALBERTO MOZO received the M.Sc. and Ph.D. degrees in computer science from the Universidad Politécnica de Madrid. He is currently a Professor with the Universidad Politécnica de Madrid. His current research interests include network protocols and machine learning. He has been involved in the technical leadership of several research projects funded by the European Commission related to the application of machine learning to cybersecurity, network, and cloud management scenarios.

STANISLAV VAKARUK received the B.Eng. degree in computer science and the M.Sc. degree in artificial intelligence from the Universidad Politécnica de Madrid. He is currently pursuing the Ph.D. degree in artificial intelligence. He is also a Research Personnel in training. His research interests include unconventional computing models, parallel and distributed algorithms, and machine learning.

DANIELE CANAVESE received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, in 2010 and 2016, respectively. He is currently a Research Assistant with the Politecnico di Torino. His research interests include security management via machine learning and inferential frameworks, software protection systems, public key cryptography, and models for network analysis.

DIEGO R. LÓPEZ joined Telefónica I+D, in 2011, as a Senior Technology Expert. He is currently in charge of the technology exploration activities within the GCTIO Unit. Before joining Telefónica, he spent some years in the academic sector, dedicated to research on network services, and was appointed as a member of the High-Level Expert Group on Scientific Data Infrastructures by the European Commission. His current research interest includes applied research applicable to network infrastructures, with a special emphasis on virtualization, data-enhanced management, new architectures, and security. He chairs the ETSI ISGs on Network Function Virtualization and Permissioned Distributed Ledgers.



LEONARDO REGANO received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, in 2015 and 2019, respectively. He is currently a Research Assistant with the Politecnico di Torino. His current research interests include software security, applications of artificial intelligence and machine learning to cybersecurity, analysis of security policies, and assessment of software protection techniques.

SANDRA GÓMEZ-CANAVAL received the M.Sc. degree in software engineering and the Ph.D. degree in computer science and artificial intelligence from the Universidad Politécnica de Madrid. She is currently an Associate Professor with the Universidad Politécnica de Madrid. Her research interests include unconventional computing models, parallel and distributed algorithms, and machine learning. Her research results have been published in specialized journals and international conferences in these areas.

ANTONIO LIOY (Member, IEEE) received the M.Sc. degree *(summa cum laude)* in electronic engineering and the Ph.D. degree in computer engineering from the Politecnico di Torino. He is currently a Full Professor with the Politecnico di Torino, where he leads the TORSEC Cybersecurity Research Group. His research interests include network security, public-key infrastructure (PKI), electronic identity, and policy-based system protection.