Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Encryption-agnostic classifiers of traffic originators and their application to anomaly detection[☆]

Daniele Canavese [a,*], Leonardo Regano [a], Cataldo Basile [a], Gabriele Ciravegna [b], Antonio Lioy [a]

[a] Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy
[b] Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Firenze, 53100 Siena, Italy

ARTICLE INFO

Keywords:
Network traffic anomaly
Intrusion detection
Machine learning
DoS attacks
Web crawling

ABSTRACT

This paper presents an approach that leverages classical machine learning techniques to identify the tools from the packets sniffed, both for clear-text and encrypted traffic. This research aims to overcome the limitations to security monitoring systems posed by the widespread adoption of encrypted communications. By training three distinct classifiers, this paper shows that it is possible to detect, with excellent accuracy, the category of tools that generated the analyzed traffic (e.g., browsers vs. network stress tools), the actual tools (e.g., Firefox vs. Chrome vs. Edge), and the individual tool versions (e.g., Chrome 48 vs. Chrome 68). The paper provides hints that the classifiers are helpful for early detection of Distributed Denial of Service (DDoS) attacks, duplication of entire websites, and identification of sudden changes in users' behavior, which might be the consequence of malware infection or data exfiltration.

1. Introduction

In recent years, secure communication channels have been increasingly employed to protect user communications and web traffic [1]. Secure channels provide enormous benefits for users. For instance, they prevent Man-in-the-Middle (MitM) attacks, and enable better protection of user-sensitive data. However, encryption severely hinders monitoring ability. Intrusion Detection and Prevention Systems (IDS/IPS) cannot detect security-relevant events that rely on sniffed payloads. Companies adopt re-encryption [2] to cope with this problem, while states resort to lawful interception [3]. Still, improving monitoring with encrypted communication remains one of the most significant issues in the security field.

This paper presents an approach towards improving defenders' monitoring without having to choose between privacy and security. The approach uses Machine Learning (ML) classification systems to determine the tool that generates the traffic independently of the protection applied to the communication channels (e.g., encryption).

Determining the tool that generated the traffic has important implications and applications to network security. For instance, distributed Denial of Service attacks use many network stress tools to generate a massive traffic volume. Web forgery exploits web crawlers to create offline copies of websites. Therefore, identifying the activities of these tools may help to prevent attacks. Moreover, detecting non-tech-savvy corporate users who surf the web using command line browsers may unveil compromised computers, e.g., malware connecting to command and control sites.

[☆] This paper is for regular issues of CAEE. Reviews processed and approved for publication by the co-Editor-in-Chief Huimin Lu.
[*] Corresponding author.
E-mail addresses: daniele.canavese@polito.it (D. Canavese), leonardo.regano@polito.it (L. Regano), cataldo.basile@polito.it (C. Basile), gabriele.ciravegna@unifi.it (G. Ciravegna), lioy@polito.it (A. Lioy).

https://doi.org/10.1016/j.compeleceng.2021.107621
Received 8 October 2020; Received in revised form 20 October 2021; Accepted 14 November 2021
Available online 6 December 2021

The main contribution of this paper is a set of classical ML classifiers that categorize with high accuracy the tools that generate the network traffic. More in detail, the paper contributes the design and training of:

- classifiers to identify three *categories of software tools*, i.e., web browsers, web crawlers, and network stress tools, as significant yet non-exhaustive representatives of traffic generation tools;
- classifiers to distinguish the traffic generated by several specific *tools*, e.g., Chrome, Firefox, GrabSite, and SlowLoris
- classifiers to identify the *tool versions*, e.g., Firefox 42, 62, and 68.

Moreover, the paper proves that the information generated by the classifiers is useful for monitoring purposes, both for live anomaly detection and post-mortem analysis. Finally, this work also identifies additional research issues for a broader application of this approach to intrusion detection.

To the best of our knowledge, this is the first work that uses ML approaches to identify and categorize the entities that generate the traffic. Past works, instead, focused on classifying the application layer protocol or the content in the sniffed traffic.

The complete source code used to train and test the classifiers, the data sets, and all the trained models are freely available online.[1]

The rest of the paper is organized as follows. Section 2 presents the relevant works in the field. Section 3 illustrates the application scenarios and the high-level objectives of the research. It also describes the tools and the categories considered, the traffic gathered for building the data set, and how the ML classifiers have been designed and trained. Section 4 presents the performance of the trained classifiers, summarizes the research findings, and discusses further works that may help answer the open questions. Finally, Section 5 draws the conclusions.

2. Related works

Traffic classification was initially based upon port numbers. Then, the employment of dynamic ports made this approach less effective and forced the adoption of DPI techniques. In turn, the advent of encrypted traffic reduced DPI accuracy. Thus, recently ML-based methods stood out since they only need unencrypted headers or information from encrypted data.

Many works are available in the literature. They differ for the classification problem, the type of feature employed, and the ML method adopted. First, the classification goal strongly depends on the problem at hand, which may be either protocol type classification [4,5], application fingerprinting [6], or identifying ongoing attacks [7,8]. Second, the features employed in this task vary considerably. Rezaei et al. [9] group the input features commonly used by classifiers in three categories: Payload + Header, Time Series + Header, and Statistical Features. End-to-end approaches, where the classifier itself extracts its representation of raw data, typically employ payload analysis. Time Series + Header and Statistical features are more general as they apply to both encrypted and unencrypted data. Finally, feature selection also affects model selection and the related computational power. Rezaei [9] reported that classical ML algorithms and shallow neural networks generally perform better with statistical features [10–12]. Instead, employing payload data requires more complicated models, like Convolutional Neural Networks, to achieve good classification results [4,13]. Time series have medium complexity and have been successfully used for diverse applications [5,6,8]. The remainder of this section reports some of the most important works, according to the classification goal.

Wang et al. introduced the end-to-end learning paradigm in the context of protocol classification [4]. They employed a 1D CNN on raw data of the SCX VPN-nonVPN data set. The authors did not report any results in training or evaluating time or made a complete comparison against traditional ML techniques. In the same context, Lopez-Martin et al. applied deep learning (DL) techniques [5], with time-series features, to traffic captures of the RedIRIS network. They proposed a few classifiers based on either CNN or Recurrent Neural Network (RNN) or a combination of the two. While achieving good results, the authors do not perform any comparison against classical ML algorithms.

Regarding app fingerprinting, Taylor et al. proposed AppScanner [10], a framework able to classify traffic generated by 100 different applications. They employed either Support Vector Classifier (SVC) or random forests from each traffic flow. Chen et al. showed the suitability of deep neural networks also for app fingerprinting [6]. They tested their pipeline to classify the application, on a first data set, or the protocol, on a second data set used to generate the traffic data. They employed a classic 2D CNN after having projected six time-series features into a multi-channel image.

The work of Balla et al. performed real-time detection of web crawlers [14]. As in this paper, the authors employed a decision tree to understand whether a human or a crawler started the ongoing session. The classifiers described in this paper can identify the specific crawler tool producing the traffic. Hence, the work presented in this paper can be considered a generalization of the work above.

Vargas et al. created a Bayesian Network model to classify attack types such as worms and DoS/DDoS attacks with time series [8]. The reported accuracy is impressively high. However, since the model takes into account only terminated flows, this method is not applicable online.

Naseer et al. investigated the suitability of DL for anomaly detection [7]. They trained different DL architectures, namely Auto Encoders (AE), CNN, Long Short-Term Memory (LSTM) on the NLS-KDD data set, including four attack typologies: DoS, U2R, R2L, and Probe. Their comparison against classical ML techniques showed that DL techniques improve the accuracy of a few percentage points at the cost of at least three orders of magnitude higher training time.

[1] https://github.com/daniele-canavese/fingerprinting/

Finally, the most similar work found in the literature is a paper by Wang et al. [13]. The authors created a data set comprehensive of both malware and real traffic data, used to train a CNN model. While authors claimed their framework has early-stage detection capability, the evaluation time of the overall pipeline is missing; thus, the employment of such a framework in a real-time detection scenario is dubious.

The research presented in this paper improves the app fingerprinting studies in the literature in two directions. First, it groups the tools that generate traffic into categories, which proved a relevant factor (see Section 4). Second, the paper also investigates the differences in the traffic generated by different versions of the same tools. Moreover, this paper frames the knowledge that the ML classifiers can extract into the context of intrusion detection; it evaluates their impact on the detection abilities and estimates how complex it is to update and manage monitoring controls using such classifiers constantly.

3. Proposed method

The high-level objective of this research is to investigate ML techniques that can improve the IDS/IPS monitoring abilities without threatening user privacy.

This paper proposes a method is to train encryption-agnostic network classifiers. The classifiers receive as input features extracted from a sequence of TCP packets, named TCP flow. The features are 31 network statistics computed on the IP and TCP headers of the packets in the flow. The classifiers perform three classification tasks that label the sequence with the following information:

1. the category of the tool that generated the traffic, among a fixed set of categories, i.e., web browser, web crawler, network stress tool;
2. the name of the tool;
3. the version of the tool;

The data set employed to train the classifiers comprises sequences of TCP packets sniffed from actual network traffic. The data set has been divided into training, development, and test sets according to best practices. The TCP flows in the training set have been labeled with the category, name, and version of the tool that produced them. Three ML models have been optimized and trained to execute each of the three classification tasks, random forests, extra-trees, and neural networks. The performance of the different models has been evaluated for each classification task.

The information of the classifiers can be useful for different monitoring purposes. The first objective is the early detection of DDoS attacks, as late discovery reduces the chances of successful reactions [15]. Another objective is blocking misbehaving web crawlers both to avoid exhaustion of site resources and web forgery. Detecting malware infections by observing anomalous uses of tools is very important. For instance, detecting inexpert employees using command-line tools and libraries may reveal malware connecting to its command and control site. Moreover, if clear-text data is available (e.g., obtained through re-encryption or at the endpoints), this approach could be potentially used to spot mismatches between detect tools and data extracted from the HTTP requests (e.g., User–Agent spoofing).

There are two main application scenarios for the classifiers presented here. The first is a corporate scenario, where security administrators configure a border firewall and couple it with a monitoring system (e.g., IDS/IPS) to protect the perimeter. Users are recognizable by their static IP address, and their behavior can be monitored. The second is a SECaaS (SECurity as a Service) scenario. The company outsources the security of the IT infrastructure to a trusted third party, which provides security services remotely, without requiring on-premises hardware [16]. In this scenario, Deep Packet Inspection (DPI) is usually not feasible for both technical (i.e., encrypted connections) and lawful reasons (e.g., General Data Protection Regulation compliance). Instead, ML-based techniques can gather relevant monitoring information from encrypted communications. Moreover, the classifiers could be used for cloud computing (identifying the browser may allow detecting access anomalies to “Software as a Service” instances), and also for the already mentioned detection of DDoS attacks in software networks [15,17].

The remainder of this section presents:

- the details about the tools considered, the categories used to label them, and the rationale for choosing specific versions of the tools (see Section 3.1);
- general information and statistics about the data set and its generation from network traffic captures (see Section 3.2);
- the ML models used, the training procedure, and the related optimizations (see Section 3.3).

3.1. Traffic to classify

This paper will refer to the term *tool* as a specific software application that produces traffic. For instance, Chrome is an instance of a web browser, and GrabSite is an instance of a web crawler. *Tool instances* (also referred to as *tool versions*) are specific releases of a tool. For instance, Chrome version 68 is a tool instance as well as GrabSite version 2.1.16. This research takes into account only TCP connections generated by selected tools instances belonging to three *tool categories*: 1. web browsers; 2. web crawlers; 3. network stress tools. This approach aims to determine if classifiers can successfully identify tool categories, tools, and tool instances. It does not aim at exhaustively classifying The purpose is to apply these classifiers to monitoring systems; thus, the research investigated whether they can be used online, in IDS/IPS, or best for post-mortem forensic analysis.

3.1.1. Tool categories

The three tool categories taken into account have peculiarities that make them interesting for detection purposes.

Web browsers. Web browsers are tools that users employ to surf the Internet, including music and video streaming services, which together constitute the predominant part of Internet traffic.[2]

Web crawlers. Web crawlers, also known as spiders or spiderbots, are web applications that automatically browse web pages by following hyperlinks. Crawlers download several pages in a short time frame. Hence, they may cause resource consumption on target web servers. They can be used to keep search engine indexes up to date but may also be support tools for dangerous attacks [18]. *Web scraping* consists in automatically downloading a whole website to offline extract valuable data (e.g., email addresses for a phishing campaign[3]). For instance, *Website forgery* duplicates a website to deceive users of the legitimate website into revealing sensitive information, e.g., e-banking login credentials [19]. The *robot exclusion standard*[4] or filters on the `User--Agent` field [20] may help with benign crawlers and agents, but they are ineffective against misbehaving crawlers.

Network stressing tools. Denial of Service (DoS) attacks aim to halt the fruition of services by its intended users [21]. *Volumetric DDoS attacks* involve many machines infected with malware that launch a coordinated attack against a single target to consume the resources (e.g., bandwidth, CPU time, memory) until it cannot provide the intended services. For example, the Low Orbit Ion Cannon (LOIC)[5] attack tool floods the target with TCP or UDP packets and provides a *"hivemind"* feature to coordinate remote machines. *DoS protocol attacks* exploit legitimate protocol features. For example, HTTP Slow DoS attacks aim at exhausting web server resources by establishing a high number of low-bandwidth connections maintained open for long periods through the HTTP Keep-Alive mechanism [22]. HTTP Slow DoS attacks are worth investigating, as they are difficult to recognize with existing methods without resorting to DPI. Finally, *DoS application attacks* exploit vulnerabilities of the application or daemon to stop the service or force the server to crash. For example, the *mod_md* Apache Module, which implements automatic SSL certificate provisioning [23], was vulnerable to these attacks. A specifically crafted HTTP request can force earlier versions of this module to dereference a NULL pointer, causing a segmentation fault that halts the webserver execution.[6]

3.1.2. Tools and tool instances

This research encompasses relevant tools for each of the three categories. It includes four most used browsers (at the moment of this writing), e.g., Chrome (v.48 and 68), Firefox (v.42, 62 and 68), Edge (v.42), and Opera (v.62).

Furthermore, it considers five web crawlers. Three, namely Wget[7] (v.1.19), Wpull[8] (v.2.0.1), and Curl[9] (v.7.55), are command-line browsers often used in more sophisticated scripts and tools such as web crawler engines. On the other hand, GrabSite[10] (v.2.1.16) and HTTrack[11] (v.3.49.2) are two open-source programs designed to back up entire websites.

Finally, this research takes into account five network stress tools: Slowloris[12] (v.7.70), HTTP Unbearable Load King (HULK)[13] (v.1.0) and its evolution GoldenEye[14] (v.3.49.2), RUDY (R-U-Dead Yet?)[15] (v.1.0.0), and SlowHttpTest[16] (v.1.6). These tools perform different forms of HTTP Slow DoS attacks.

All the tools listed in this section are available both for Microsoft Windows and Linux (in some cases, the exact versions were unavailable for different OSes). The only exceptions are Microsoft Edge and Opera (Windows only) and SlowHttpTest (Linux only). Section 3.2 explains the training and testing of the classifiers, using traffic from the same tools executed from different OSes.

3.2. Data set creation

The data set consists of the traffic statistics computed on a set of network captures. In this scenario, the TCP connections are the samples used to train and test the classifiers. Indeed, the analysis of each flow in isolation allows a classification independent of the number of flows simultaneously active during the detection.

[2] https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf
[3] https://www.owasp.org/index.php/Phishing
[4] http://www.robotstxt.org/orig.html
[5] https://sourceforge.net/projects/loic/
[6] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8011
[7] https://www.gnu.org/software/wget/
[8] https://github.com/ArchiveTeam/wpull
[9] https://curl.haxx.se/
[10] https://github.com/ArchiveTeam/grab-site
[11] https://www.httrack.com/
[12] https://github.com/gkbrk/slowloris
[13] https://github.com/grafov/hulk
[14] https://wroot.org/projects/goldeneye/
[15] https://www.imperva.com/learn/ddos/rudy-r-u-dead-yet/
[16] https://tools.kali.org/stress-testing/slowhttptest

Table 1
Average network statistics of examined captures per category.

	Browsers	Crawlers	Stress tools
Packets sent to the server	30.6	95.7	11.0
Packets received from the server	41.7	158.8	16.3
Bytes sent to the server $[B]$	2381.7	1568.1	738.4
Bytes received from the server $[B]$	46519.9	344685.0	17955.8
Duration $[s]$	25495.7	4589.7	2895.3

3.2.1. *Computing traffic statistics*

Since this work aims at detecting the tools that generate the traffic even when this is encrypted, the payload cannot be used by classifiers, as it might not be intelligible. Hence, classification used data from the statistics of the TCP connection flows. On the bright side, results become independent of the transport layer payload and allow the classification of both HTTP and HTTPS traffic in a truly agnostic manner.

The statistics considered for the classification have been computed with the *TCP Statistic and Analysis Tool (Tstat)*,[17] one of the most used traffic measurement tools. Even if Tstat can evaluate statistics on the transport-layer (TCP, UDP) and application-layer protocols (HTTP, RTP, RTCP, XMPP), this work employs only the subset related to the TCP header[18] to avoid the use of DPI. The linked GitHub repository reports all the statistics that this research has considered.

3.2.2. *Network captures acquisition*

The first task has been generating the traffic, captured using WireShark 2.6.4 with the `tshark` command-line interface. Browsers' traffic derives from manual Internet surfing with Chrome, Firefox, Edge, and Opera. Instead, ad-hoc Python 3 scripts generated the traffic in the other two categories, a more common scenario when performing DoS attacks or downloading an entire website. The command-line tools used the default parameters. Since there is no testbench for these kinds of experiments, the most popular websites have been navigated.[19] These websites embedded video streams (e.g., YouTube), thus the captures also contain multimedia. The people involved were requested to navigate such sites following their normal behavior. Moreover, three persons reached other websites of their choice. This research did not leverage browser automation frameworks (e.g., Selenium[20]) since they could introduce unwanted patterns in the navigation of the selected websites.

The next step has been using `tshark` to filter the unnecessary background connections (e.g., UDP packets and connections not related to the tools in analysis), followed by the generation of multiple truncated versions of all the captures. The latter are necessary to verify whether the classification could also work online, that is, while a TCP connection is still open. Connections containing only one packet are discarded since they did not carry useful information to train the classifiers. The truncated capture files have been obtained by only retaining the segments of each TCP connection received before a specific (hard) time constraint. The timeout has been preferred over the TCP sequence number to keep potential out-of-order packets.

The last task has been executing `tstat` on both truncated capture and original files to compute 31 network statistics (including the overall number of packets and details about packets with or without payloads or specific flag set, information about retransmissions, durations[21]), together with the additional Boolean feature that reports if a TCP connection has been gracefully terminated or not.

This research leverages different strategies to maximize the variability of the captures. Two machines with different operating systems, Windows 10 and Debian/GNU Linux (kernel 4.17.0, 4.18.0, and 4.19.0), generated the traffic. The machines were connected to the network in different locations (campus, residential facilities located in different cities), using both wired and wireless connections. Moreover, traffic was generated by different versions of the same tools (e.g., Firefox 42 and 62) whenever possible, running under different OSes.

3.2.3. *The data set*

The data set consisted of 1224985 TCP flows, 187547 gracefully terminated and 1037438 non-terminated. About 58% of the flows are encrypted with TLS (711492 connections), while the remaining flows (513493) are clear-text HTTP connections.

Table 1 shows the average traffic statistics grouped by tool category that allow the formulation of simple yet important considerations:

- the browser connections are, on average, the longest ones (i.e., half a minute vs. five seconds) — the most acceptable explanation is because browsers have been human-driven when generating the traffic in the captures;

[17] http://tstat.polito.it/
[18] http://tstat.polito.it/measure.shtml#log_tcp_complete
[19] https://en.wikipedia.org/wiki/List_of_most_popular_websites
[20] https://www.seleniumhq.org/
[21] The complete list of used statistics is available on the GitHub repository, and they correspond to the {3–14,17–28,31–37} statistics listed at http://tstat.polito.it/measure.shtml.

Table 2
Characteristics of the training and test platform.

OS	GNU/Linux Debian 5.8.14-1
Python	3.9.1
packages	scikit-learn 0.24.1, PyTorch 1.7.1, skorch 0.9.0, hyperopt 0.2.5
CPU	Intel i9-9820X @ 3.30 GHz
RAM	96 GiB (DDR4 @ 4 GHz)
video card	NVIDIA GeForce RTX 2080 Ti (11 GiB)

- web crawlers download a lot of data (bytes) from servers, coherently with their typical behavior, i.e., downloading entire websites indiscriminately;
- network stress tools have short connections and request very few bytes since their job is (usually) to keep multiple open connections and saturate servers.

3.3. Machine learning models and tools

Several machine learning models have been used to classify the network traffic in the three categories listed in Section 3.1 (web browsers, web crawlers, and network stress tools), identify the traffic generator tool, and detect the specific tool instance. The following three models have been trained for each of these classification tasks (category, tool, and tool instance):

- random forests [24], i.e., ensembles of binary decision trees that chooses the optimum split points;
- extra-trees (extremely randomized trees) [25], i.e., another ensemble of binary decision trees that, instead, chooses the split points at random;
- a fully connected neural network [26] where the hidden layers use the ReLU activation function, and the output layer uses the sigmoid to assign a class to the observations.

This section reports the most relevant information. The complete data are available in the GitHub repository.

The implementation, developed in Python 3, uses the `scikit-learn`[22] library to implement the random forests, extra-trees, SVMs, and kNN classifiers. Neural networks, developed using a custom implementation based on `PyTorch`[23] and `skorch`[24], have been trained and tested them using the GPU acceleration capabilities. Table 2 reports data about the platform used to train and test all the classifiers.

3.4. Training procedure

This section reports the training of the classifiers, executed to perform the following experiments:

- classify new flows generated by tools included in the training set;
- identify a completely unknown tool not included in the training set, to assess the ability of the classifiers to evaluate an unknown browser (Opera), new network stress tool (SlowHTTPTest), and web crawler (GrabSite);
- categorize a new tool version whose previous releases are already present in the training set — e.g., identify Firefox 68 with classifiers trained with data generated by Firefox 42 and 62;
- check if the classifiers improve or maintain the detection abilities after adding samples of a new version of one of the tools to the training set.

The training consists of three consecutive steps.

Training, development, and test sets. Data set have been split into a development set (for the hyper-parameter optimization) and two different test sets according to the following procedure:

1. the *Unknown Tools test Set* (UTS) consists of all the samples generated by Opera, GrabSite, SlowHTTPTest, and Firefox 68.0 (not used for training, 30144 samples);
2. the remaining observations formed the *training set* (955872 randomly selected samples, 80%), *development set* (for the hyper-parameter optimization, 119484 randomly selected samples, 10%), and the *Known Tools test Set* (KTS, the remaining 10%, 119485 samples).

[22] See https://scikit-learn.org/.
[23] See https://pytorch.org/
[24] See https://github.com/skorch-dev/skorch.

Table 3
Performance metrics of the category classifiers.

Statistic	Random forest	Extra trees	Neural network
Balanced accuracy [%]	94.041	80.373	90.308
F-score [%]	89.230	61.866	79.902
R_k	0.850	0.489	0.778
Classification time (one flow) [μs]	2.254	2.186	8.134
tstat + classification time (one flow) [μs]	16.317	15.902	21.461

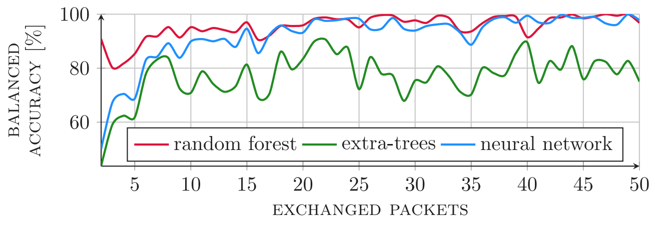

Fig. 1. Balanced accuracy vs. exchanged packets for the category classifiers.

Hyper-parameters optimization. Hyper-parameters optimization has been performed using `hyperopt` with a Bayesian optimization [27] to maximize the R_k statistic [28], extending the Matthews correlation coefficient to multi-class problems. The R_k statistic has two main advantages over most of the more traditional metrics. First, it works well with unbalanced data sets (as in this case). Second, it is more informative than traditional performance measures since it considers all the possible classifications and misclassifications in the confusion matrix and does not lead to unbalanced classifiers (as the F-score does). The optimization terminated after no increase in the best R_k metric for 30 consecutive iterations (a sign of stability to the near-optimum point).

Model training. The actual training of the models employed the optimal hyper-parameters. Since the data set contained an imbalanced number of classes, samples' classification weights were adjusted accordingly to avoid pruning any sample.

4. Results and discussions

This section reports the performance of the three trained classifiers. Furthermore, it discusses the analysis of the results, identifies relevant issues that remain unsolved, and highlights future research directions.

4.1. Category classification

Table 3 reports the performance metrics computed on the KTS [29] and their average classification time of the category classifiers that, as previously described, can classify the traffic into the three categories listed in Section 3.1: web crawlers, network stress tools, and web browsers.

The fastest classifier is the extra-trees. However, it is also the least accurate. The random forest model is the most precise and is nearly as fast as the extra-trees model. The neural network is the slowest one, most likely because there is an encoding/decoding phase to transform class labels into numbers.

Fig. 1 plots the balanced accuracy of the classifiers depending on the number of packets in the examined flow on KTS.

The general trend is that the classifiers' performance improves as the number of exchanged data increases. Once a connection reaches six exchanged packets, the balanced accuracy of the random forests stabilizes, being above 92%. Hence, sensors using these classifiers for detection purposes should observe at least six packets to identify the traffic originators in a trustworthy manner. The neural network has a similar performance as the random forest (albeit showing a slower convergence). The extra-trees, instead, are always less accurate than their counterparts.

4.2. Tools classification

The classifiers that categorize the TCP streams according to their generator tools correctly identified the eleven applications in the data set, namely Chrome, Curl, Edge, Firefox, GoldenEye, Httrack, HULK, RudyJS, SlowLoris, Wget, and Wpull. Table 4 reports the performance statistics computed on the KTS.

Identifying the tool that produces a TCP stream is intuitively harder than only detecting the categories. Experiments have proved this conjecture. The overall performance of these classifiers is lower than the one presented in Section 4.1. Also, the classification times are significantly higher. The best classifier (a random forest) obtained a balanced accuracy of 90% on the KTS. However, it is also the slowest one. On the other hand, the extra-trees are the less accurate but also the fastest ones.

Table 4
Performance metrics of the tool classifiers.

Statistic	Random forest	Extra trees	Neural network
Balanced accuracy [%]	90.366	66.701	80.332
F-score [%]	81.477	51.257	59.479
R_k	0.904	0.711	0.782
Classification time (one flow) [µs]	24.675	5.828	8.054
tstat + classification time (one flow) [µs]	37.992	19.518	22.001

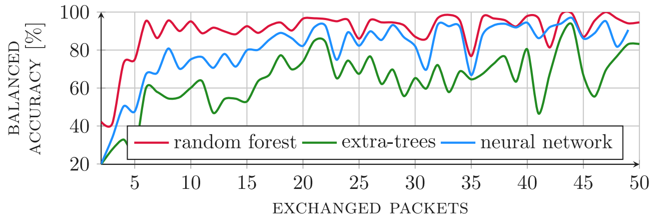

Fig. 2. Balanced accuracy vs. exchanged packets for the tool classifiers.

Table 5
Performance metrics of the tool instance classifiers.

statistic	Random forest	Extra trees	Neural network
Balanced accuracy [%]	87.183	62.533	76.767
F-score [%]	79.254	43.182	49.411
R_k	0.899	0.677	0.719
Classification time (one flow) [µs]	25.564	33.006	8.093
tstat + classification time (one flow) [µs]	39.214	46.911	22.023

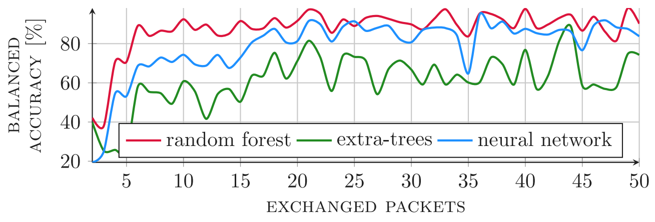

Fig. 3. Balanced accuracy vs. exchanged packets for the tool instance classifiers.

Fig. 2 reports the balanced accuracy trend when the number of exchanged packets increases. As for tool categories, classification performance increases as the number of packets increases. After about six exchanged packets, the balanced accuracy of the random forest stabilizes with a value that is usually above 90%. However, compared with the category classification, the performance is slightly lower, most likely due to the greater difficulty of the classification task.

4.3. Tool instances classification

Table 5 reports the performance statistics of the classifiers trained to identify the 16 tool instances on the known tool set.

As in the previous scenarios, the most accurate model is the random forest, achieving a balanced accuracy of about 87%. However, the classification time of the neural network is about 3–4 times faster than the tree-based models.

Fig. 3 reports the plot of the balanced accuracy of the tool instance classifier vs. the exchanged packets. As in the other two cases, six packets are usually enough to have a stable value of the balanced accuracy of the random forest, which usually floats around 90%.

4.4. Classification of unknown tools

The ability to classify the tools not included in the training set (i.e., Firefox 68.0, Opera, GrabSite, and SlowHTTPTest) showed mixed results. Fig. 4 shows the classification results of the UTS.

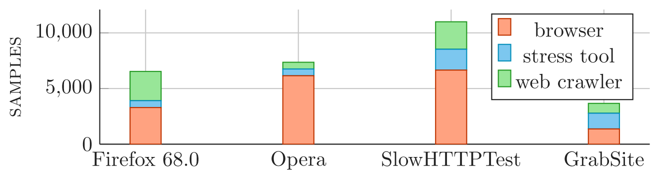

Fig. 4. Classification of the unknown tools by the random forest.

On the one hand, Opera has been identified as a browser in 69% of the cases. This constitutes a good level of accuracy for a tool not considered during the training. Firefox 68.0, however, was correctly identified as a browser only in about half of the observations. This may indicate that, in several cases, the browsers generate traffic with patterns that can be recognized using ML-based classifiers. On the other hand, the classifier correctly labeled Grabsite as a web crawler for 17% of the samples and SlowHTTPTest traffic as network stress in 24% of the cases. These results may indicate that the network fingerprints of web crawlers and network stress tools appear to be very specific to the tool that generated the examined packets. For instance, an interesting example that seems to confirm the hypothesis of tool-instance-specific fingerprints is the classification of SlowHTTPTest. Even if this tool internally uses the same basic approach as RudyJS and Slowloris, the results are contradictory. The data set includes several samples of these two DoS applications. Still, the classifier had great difficulty recognizing it as another instance of a network stress tool. A possible explanation is that RudyJS is written in JavaScript, Slowloris in Python, and SlowHTTPTest in C. The use of different technologies and libraries seems to alter enough the fingerprint making their identification very difficult.

4.5. Analysis of results and research directions

The main findings of this research are summarized here.

Category identification. The Category Classifiers can reliably determine the tool category that generated the traffic only if the tool traffic was in the training set. The categorization is challenging for a tool not considered in the training set as classifiers only worked reasonably well for browsers. They were almost entirely wrong for network stress tools and web crawlers.

Tool identification. The Tool Classifiers can reliably determine the tool that generated the traffic only if the tool traffic was in the training set. These classifiers are almost as accurate as the ones that determine the category. Determining traffic from versions of the tools not considered during the training seems challenging, yet not impossible, at least for browsers. However, more extensive studies would be needed to confirm these hypotheses also using semi-supervised methods.

Classifiers and IDS. Standard IDS architectures may employ the classifiers as sensors that generate three detection events when, after reading enough packets, they classify a connection as being generated by: 1. a tool category (e.g., browser), 2. a specific tool (e.g., Firefox), 3. a specific tool instance (e.g., Firefox 48). A correlator can be instructed with ad hoc rules to react based on these events' type, frequency, and cardinality.

Instance identification. The Instance Classifiers can reliably determine the version of the tool that generated the traffic only if the traffic generated with that version of the tool was in the training set. These classifiers are slightly less accurate than the category and tool classifiers. It is worth investigating how the accuracy of classifiers changes if several instances of the same tool are in the training set (e.g., 20 or 30 versions of Firefox and Chrome). Moreover, tracking the evolution of the fingerprint of tools may improve the performance of classifiers (as discussed later).

DoS detection. The network stress tools considered in the training set manifest a clear fingerprint that the classifiers can detect. IDS/IPS can use this information to determine if a site is under DoS attacks and react accordingly. Nonetheless, more effort is needed to extend this approach to application attacks.

Early detection. Data show that a reasonably accurate prediction requires at least six packets. Balanced accuracy usually becomes stable after ten packets. An IDS/IPS should not make decisions based on classifications made on few packets. An IDS/IPS can achieve better correlations and decisions when it uses data from the concurrent classification of more flows. Designing an IDS correlator that properly leverages these classifiers is challenging (industrial) research that can positively impact monitoring.

Browsers. Classifiers have correctly recognized browsers even when the training set has not comprised their traffic. A possible explanation is that the way browsers interact with websites is standard, e.g., prefetching pages, loading complex pages with multiple requests. Another interpretation is that these peculiarities depend on the fact that humans directly drive these tools. Further research is needed to separate human patterns from tool behaviors for monitoring purposes.

Unknown tools. Unsurprisingly, the classifiers, based on supervised methods, have trouble classifying unknown tools. Unsupervised or semi-supervised methods should be tested in these cases.

New versions of tools. The analysis showed that tools might have a fingerprint, which may occasionally be preserved in later versions. However, this result was more evident for browsers where the "human effect" cannot be isolated. Since, in general, there are no specific tool fingerprints that propagate through versions, supervised classifiers require re-training with traffic of the tools of interest to make them effective for monitoring purposes. Further studies would be needed to characterize how a fingerprint propagates in close versions (e.g., 62.0 vs. 63.0) for more extended periods (e.g., in the last ten years) and if unsupervised and semi-supervised methods perform better in this task.

Malware. Since classifiers accurately recognize tools and libraries, detecting traffic generated by malware applications should be possible. To increase the detection accuracy, however, the malicious traffic should be included in the training set of the classifiers.

Users' anomalies. In a corporate scenario, where more information about the users is available, recognizing tool instances can help detect several anomalies. Examples include detecting users performing operations that do not match their expertise level, people using tools they do not commonly employ, and differences between tools used during working and non-working hours. It may also serve to determine, without vulnerability scanners, when users employ old vulnerable browsers. Moreover, whenever the decrypted payload is also accessible, comparing the declared data (e.g., the User–Agent) against the detected one may help determine attacks aiming to bypass HTTP filters. Further analyses are needed to determine when a monitoring system can benefit from this information to improve performance (e.g., false positives and negatives).

Threats to validity. A set of threats to validity apply to the results presented in this paper. First, the research has only covered a limited number of tool categories, tools, and instances. It is plausible that similarly trained classifiers can be as accurate as the ones presented here. Considering huge sets of tools and several instances of the same tools can change the accuracy results. However, the methodology adopted follows the best practice. Hence, the approach presented here should also work with more labels. Then, implementing classifiers with other tools could reach slightly better results. Nonetheless, better classifiers may only confirm the correctness of this approach and reduce the impact of current limitations. Furthermore, the effect of the tools has not been isolated, by design, from the effect of the OS-specific and other shared libraries, which could lead to a better analysis. An analysis of the used libraries, at least for open source applications, could help perform a more careful analysis. Finally, the considerations about applying the classifiers to monitoring miss an additional evaluation in a real context.

5. Conclusions

This paper has presented an approach for the classification of the tools that generate network traffic. Based on machine learning, this approach categorizes the tools by only considering as features the traffic statistics computed on the IP and TCP headers. This research has highlighted that the best classifiers use random forests. They can identify the category of the tools and the actual tool that generated the traffic with high balanced accuracy (87% or better). These classifiers have two significant advantages when compared to already existing works. First, they can categorize live traffic, which is paramount to use in IDS/IPS scenarios. Second, they can cope equally well with clear and encrypted traffic, and they do not need to resort to DPI to work correctly.

A limitation of the approach described in this paper is that it cannot cope with the UDP traffic, whose support is steadily increasing, mainly due to the adoption of the QUIC protocol. Future research will tackle this problem by investigating what features of the UDP frames can be extracted and used to train some new machine-learning classifiers. In addition, since this approach leverages supervised techniques, it can still not accurately recognize completely unknown tools (i.e., whose traffic is not in the training set). Adopting a variety of unsupervised and semi-supervised algorithms will, hopefully, enable the classifiers to identify completely unseen tools better.

The adoption of sequential models (e.g., LSTM networks and transformers) is also worth investigating. Since these techniques keep track of how the distributions of the traffic statistics evolve, they will most likely lead to more accurate classifiers. Future research will also determine if the accuracy increases when classifiers are fed with more traffic statistics (e.g., RTT, TTL).

Finally, this approach will be tested against a variety of new attack types, with a particular focus on DRDoS (Distributed Reflective DoS) attacks and malware connections.

CRediT authorship contribution statement

Daniele Canavese: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing, Visualization. **Leonardo Regano:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing. **Cataldo Basile:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision. **Gabriele Ciravegna:** Conceptualization, Writing – original draft. **Antonio Lioy:** Writing – review & editing, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme, under projects PALANTIR (Grant Agreement No. 883335) and CyberSec4Europe (Grant Agreement no. 830929).

References

[1] Felt AP, Barnes R, King A, Palmer C, Bentzel C, Tabriz P. Measuring HTTPS adoption on the web. In: 26th USENIX secur. symp.. 2017, p. 1323–38.
[2] Waked L, Mannan M, Youssef A. To intercept or not to intercept: Analyzing TLS interception in network appliances. In: Proc. of the 2018 on Asia conf. on comput. and commun. secur.. ASIACCS '18, New York, NY, USA: ACM; 2018, p. 399–412.
[3] Gorge M. Lawful interception–key concepts, actors, trends and best practice considerations. Comput Fraud Secur 2007;2007(9):10–4.
[4] Wang W, Zhu M, Wang J, Zeng X, Yang Z. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In: 2017 IEEE int. conf. on intell. and secur. inform.. 2017, p. 43–8.
[5] Lopez-Martin M, Carro B, Sanchez-Esguevillas A, Lloret J. Network traffic classifier with convolutional and recurrent neural networks for internet of things. IEEE Access 2017;5:18042–50.
[6] Chen Z, He K, Li J, Geng Y. Seq2img: A sequence-to-image based approach towards IP traffic classification using convolutional neural networks. In: 2017 IEEE int. conf. on big data. 2017, p. 1271–6.
[7] Naseer S, Saleem Y, Khalid S, Bashir MK, Han J, Iqbal MM, et al. Enhanced network anomaly detection based on deep neural networks. IEEE Access 2018;6:48231–46.
[8] Vargas-Munoz MJ, Martinez-Pelaez R, Velarde-Alvarado P, Moreno-Garcia E, Torres-Roman DL, Ceballos-Mejia JJ. Classification of network anomalies in flow level network traffic using Bayesian networks. In: 2018 28th Int. conf. on electron., commun. and comput., Vol. 2018-Janua. 2018, p. 238–43.
[9] Rezaei S, Liu X. Deep learning for encrypted traffic classification: An overview. IEEE Commun Mag 2019;57(5):76–81.
[10] Taylor VF, Spolaor R, Conti M, Martinovic I. AppScanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In: 2016 IEEE Eur. symp. on secur. and privacy. 2016, p. 439–54.
[11] Brissaud P, Francçis J, Chrisment I, Cholez T, Bettan O. Transparent and service-agnostic monitoring of encrypted web traffic. IEEE Trans Netw Serv Manag 2019;16(3):842–56.
[12] Shen M, Liu Y, Chen S, Zhu L, Zhang Y. Webpage fingerprinting using only packet length information. In: 2019 IEEE int. conf. on commun.. 2019, p. 1–6.
[13] Wang W, Zhu M, Zeng X, Ye X, Sheng Y. Malware traffic classification using convolutional neural network for representation learning. In: Int. conf. on inf. networking. IEEE; 2017, p. 712–7.
[14] Balla A, Stassopoulou A, Dikaiakos MD. Real-time web crawler detection. In: 2011 18th Int. conf. on telecommun.. IEEE; 2011, p. 428–32.
[15] Mousavi SM, St-Hilaire M. Early detection of ddos attacks against SDN controllers. In: 2015 Int. conf. on comput., netw. and commun.. 2015, p. 77–81.
[16] Ko RKL, Choo KR, editors. The cloud security ecosystem - technical, legal, business and management issues. Elsevier; 2015.
[17] Yan Q, Yu FR, Gong Q, Li J. Software-defined networking (SDN) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. IEEE Commun Surv Tutor 2016;18(1):602–22.
[18] Cho J, Garcia-Molina H. Effective page refresh policies for web crawlers. ACM Trans Database Syst 2003;28(4):390–426.
[19] Waziri I. Website forgery: Understanding phishing attacks and nontechnical countermeasures. In: Proc. of the 2015 IEEE 2nd int. conf. on cyber secur. and cloud comput.. Washington, DC, USA: IEEE Computer Society; 2015, p. 445–50.
[20] Fielding R, Reschke J. Hypertext transfer protocol (HTTP/1.1): semantics and content. RFC 7231, RFC Editor; 2014, http://www.rfc-editor.org/rfc/rfc7231.txt.
[21] Mirkovic J, Dietrich S, Dittrich D, Reiher P. Internet denial of service: Attack and defense mechanisms (Radia Perlman Computer Networking and Security). Upper Saddle River, NJ, USA: Prentice Hall; 2004.
[22] Cambiaso E, Papaleo G, Aiello M. Taxonomy of slow DoS attacks to web applications. In: Recent trends in comput. netw. and distrib. syst. secur.. Springer; 2012, p. 195–204.
[23] Barnes R, Hoffman-Andrews J, McCarney D, Kasten J. Automatic certificate management environment (ACME). RFC 8555, RFC Editor; 2019.
[24] Breiman L. Random forests. Mach Learn 2001;45(1):5–32.
[25] Geurts P, Ernst D, Wehenkel L. Extremely randomized trees. Mach Learn 2006;63(1):3–42.
[26] Schmidhuber J. Deep learning in neural networks: An overview. Neural Netw 2015;61:85–117.
[27] Močkus J. On Bayesian methods for seeking the extremum. In: Optim. techn. IFIP tech. conf.. Springer; 1974, p. 400–4.
[28] Gorodkin J. Comparing two K-category assignments by a K-category correlation coefficient. Comput Biol Chem 2004;28(5–6):367–74.
[29] Sokolova M, Lapalme G. A systematic analysis of performance measures for classification tasks. Inf Proc Manag 2009;45(4):427–37.

Daniele Canavese received an M.Sc. degree in 2010 and a Ph.D. in Computer Engineering in 2016 from Politecnico di Torino, where he is currently a research assistant. His research interests are concerned with security management via machine learning and inferential frameworks, software protection systems, public-key cryptography, and models for network analysis.

Leonardo Regano received an M.Sc.degree in 2015 and a Ph.D. in Computer Engineering in 2019 from Politecnico di Torino, where he is currently a research assistant. His current research interests focus on software security, artificial intelligence and machine learning applications to cybersecurity, security policies analysis, and software protection techniques assessment.

Cataldo Basile received an M.Sc.in 2001 and a Ph.D. in Computer Engineering in 2005 from Politecnico di Torino, where he is currently an assistant professor. His research is concerned with software security, software attestation, policy-based security management, and general models for detection, resolution and reconciliation of security policy conflicts.

Gabriele Ciravegna is a Ph.D. student in Smart Computing at Università degli Studi di Firenze. In 2018 he received an M.Sc.in Computer Engineering from Politecnico di Torino. He is interested in machine learning and its application in critical contexts as medical diagnosis, focusing on overcoming intrinsic limits of machine learning and neural networks, especially their understandability.

Antonio Lioy is a full Professor of Cybersecurity. He received the M.Sc. in Electronic Engineering and the Ph.D. in Computer Engineering from Politecnico di Torino, where he currently leads the cybersecurity research group TORSEC. His research interests include electronic identity, PKI, trusted computing, and policy-based management of large IT systems.