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

Encryption agnostic classifiers of traffic originators and their application to intrusion detection

Daniele Canavese, Leonardo Regano, Cataldo Basile, Gabriele Ciravegna and Antonio Lioy

***Abstract*—Monitoring the security status of a network is a difficult task and it is even more difficult now, in the era of the encrypted communications. This paper presents an approach that leverages classical machine learning techniques to classify and identify the tools that originated traffic for security monitoring purposes. Our study proves that classifier can detect with excellent accuracy the category of tools that generated the analysed traffic (e.g. browsers vs. DoS tools), the actual tools (e.g. Firefox vs. Chrome vs. Edge) and the individual tool versions (e.g. Chrome 48 vs. Chrome 68). We also discuss how the information obtained by the classifiers is useful for early detection of DDoS attacks, potentially fraudulent copies of entire websites, and to identify sudden changes in users' behaviour, which might be the consequence of a malware infection or data exfiltration.**

***Index Terms*—Network traffic anomaly,intrusion detection,machine learning,DoS attacks,web crawling**

I. Introduction

Last years have seen an emerging trend: secure communication channels are more and more used to protect integrity and confidentiality of user communications and web traffic [11]. Indeed, in 2018, TLS-encrypted web pages accounted for more than 75% of time spent by users employing two of the most common web browsers: Google Chrome[1] and Mozilla Firefox[2]. This trend originates from the need of companies to protect user data and information from competitors and ISPs, but as a *side effect*, secure channels provide huge benefits for the users: MitM attacks are circumvented and user privacy is better guaranteed, as sensitive data are not sent in clear.

On the other hand, encryption severely hinders one of the pillars of the security, the monitoring ability. Indeed, Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) are unable to correctly detect the security relevant events that require sniffing the payload. At corporate level, the re-encryption is increasingly being adopted [30]. Secure channels are forcefully terminated at specific corporate gateways that inspect the traffic before re-encrypting it towards the final destinations. This solution is undoubtedly invasive, even if companies often justify this practice with the need to ensure the use of company-accepted cryptographic suites. At "political" level, lawful interception allows selected and authorized subjects to force the disclosure of the encrypted traffic, often in a way that is transparent to the users [14]. Lawful interception is objectively too invasive and should be used with care (at least, the law in different countries prescribes this), thus it cannot work for corporate security.

Our research aims at improving the monitoring abilities of the defenders without trading off user privacy and security. We have thus focused our research on *determining the tool that generates the traffic* independently of the protection applied to channels, which has practical relevance yet has not been consistently explored. We concentrated on machine learning classification systems. Indeed, machine learning has shown great effectiveness in several fields of the computer security, including the monitoring [10], [5].

Determining if machines are able to identify the tool that generated the traffic regardless of encrypted or cleartext content (e.g. with TCP connection information only) is, in our opinion, an interesting question *per se*, as well as knowing if tools (better, the peculiarities of their implementations) leak information that may allow machines to recognize them. Nonetheless, answering these questions has important implications and applications to network security. For instance, in a typical corporate network environment, users surf the web for personal or business purposes with browsers. It is very unlikely that non-tech-savvy users use command line browsers (e.g. curl or wget). Detecting the use of these tools is an anomaly and may be the evidence that a user computer has been compromised by malware connecting to its command and control site. Moreover, Distributed Denial of Service (DDoS) and web forgery attacks attacks could be detected by identifying the tool that generates the traffic. The former resorts to a myriad of instances of specific tools to generate a huge amount of traffic, the latter uses crawlers to create offline copies of entire web sites. Hence, even this non-exhaustive list of attacks that a monitoring system can detect with traffic originator information can clarify that gathering tool information may be useful for security purposes.

The main contribution of this paper is a study on the possibility for classical machine learning approaches to detect the software that generated the traffic and how to use the detected information for monitoring purposes. We have considered in our study three categories of software tools: web browsers, web crawlers, and network stress tools (i.e. tools for DDoS), as significant yet non-exhaustive representative of tools that generate traffic in the today's network. For each

This paper has been partially developed within the EIT Digital innovation activity Deep-Augur (18217). Computational resources were provided by HPC@POLITO, a project of Academic Computing within the Department of Control and Computer Engineering at the Politecnico di Torino (http://www.hpc.polito.it). *(Corresponding author: Daniele Canavese.)*

Daniele Canavese, Leonardo Regano, Cataldo Basile and Antonio Lioy are with the Dipartimento di Automatica e Informatica, Politecnico di Torino, 10129 Torino, Italy. (e-mail: first.last@polito.it)

Gabriele Ciravegna is with the Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Firenze, 53100 Siena, Italy. (e-mail: gabriele.ciravegna@unifi.it)

[1]https://transparencyreport.google.com/https/overview

[2]https://letsencrypt.org/stats/

category, we have considered the traffic generated by several tools (e.g. Chrome, Firefox, GrabSite, SlowLoris) and, for a selection of tools, we have considered multiple versions (e.g. Firefox 42.0, 62.0 and 68.0). We have evaluated the online detection of tools, versions, and categories, i.e. we have verified if the detection is possible as soon as packets are received, that is, before the whole connection is terminated. Therefore, our study covers both live anomaly detection and post-mortem analysis, the availability of capture data being the only requirements.

To the best of our knowledge, this is the first work that uses ML-based approaches not only to classify the content of the sniffed traffic or to recognize tool-specific application layer protocols (using time-series or network connection statistics), but it focuses on identifying and categorizing the entities that generated the traffic from the security point of view.

Our analysis has showed several interesting results, some are sketched here and detailed in the rest of the paper. The classifiers trained to determine the categories of tools that generate traffic showed a high accuracy, with an F-score and AUC of 96%. Furthermore, the classifiers were able to identify new browsers, i.e. browsers not considered in the training set, but they were not able to recognize unknown web crawlers, and network stress tools, which were confused with tools in other categories. We have deduced that browsers manifest peculiar fingerprints, and different browsers, either because of the human beings behind them or because of their intrinsic characteristics, have similar fingerprints. When training additional classifiers to detect the tools that generate the traffic, we noticed a good accuracy even if not as good as for categories (AUC = 93%). Classifiers trained to detect specific tool versions were less accurate but still effective (AUC = 91%). In most cases, when a classifier was wrong with a browser, it selected a wrong version of the same tool.

The performance of the classifiers depends on the numbers of packets examined, the more the packets the better the accuracy. However, after six packets the performance of the classifiers stabilizes. Hence, they could be used in monitoring systems for online threat detection. Moreover, this study presents additional research issues to investigate on the possibility to use this approach for monitoring purposes.

The rest of the paper is organized as follows. Section II illustrates the application scenarios and the high-level objectives of our research. Section III presents the categories of tools that generate traffic that we have collected as data set for our experiments as well as the detailed research objectives. Then, the Section IV presents the data set that is used for the training and testing of the classifiers described in Section V together with all the numerical results. Section VI sketches possible applications of the classifiers to a monitoring system. Section VII summarizes the findings of our research and discusses further works that may help answering the open questions. Finally, Section VIII presents the relevant works in the field and Section IX draws conclusions.

II. Motivating examples

The high-level objectives of our research is to investigate novel uses of ML techniques that can improve the state of the art of monitoring controls. The idea is to define classifiers that can be perform checks that are not available in current IDS/IPS tools or are not possible with traditional monitoring systems based on statistical data.

- As a first objective, we would like to improve the capability of early detection of DDoS attacks, because discovering DDoS too late strongly reduces the chances of successful reactions [24] [26] [9] [13].
- Another research objective is identifying the web crawlers (and telling them apart from web browsers in a trustworthy manner) to block the most aggressive or misbehaving ones, both to avoid stressing the site resources and to avoid that they clone an entire website for attack purposes (e.g. web forgery).
- We are interested in determining if is possible to support anti-malware systems to detect infections by identifying the anomalous use of web tools to access unusual resources. For instance, we are interested in identifying when tools that are expected to be used by tech-savvy people (e.g. command line tools) are used by non-expert employees, which may be an evidence of a malware that reaches its command and control sites.
- Finally, we would like to detect changes in the web navigation user habits. For instance, we are interested in detecting when a user changes the web browser or if he is using at the same time more than one tool.

We stress that we want to achieve our objectives without threatening the user privacy. However, as an additional objective, if re-encryption is used or at the endpoints, we would like to detect mismatches in the information extracted from the actual HTTP requests (e.g. tools declaring a fake User Agent).

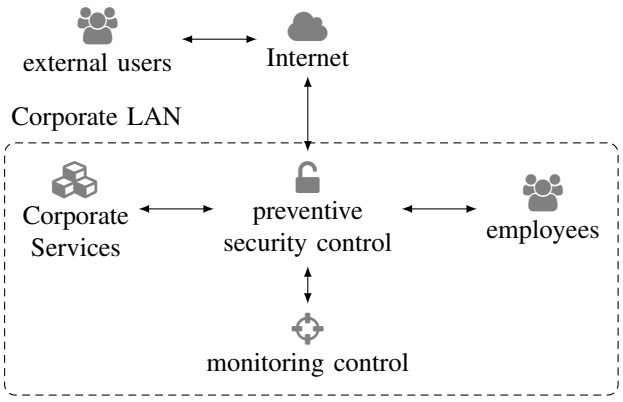


Fig. 1: Company network scenario.

We have identified two main sample application scenarios for the application to an IDS of the results that can be obtained with the our classifiers. The first one is a typical corporate scenario, where preventing and monitoring security controls are used to defend the corporate network perimeter. In this case, the security administrators configure a border firewall and couple it with a monitoring system, like an IDS/IPS (see Figure 1 for a possible graphical representation). In most cases, internal users are recognizable in an unequivocal way,

e.g. because of a static IP address of sniffed MAC address. Therefore, their behaviour can be monitored.

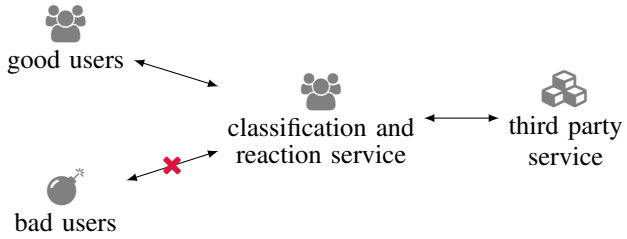


Fig. 2: SECaaS scenario.

The second scenario is the implementation of the SECaaS (SECurity as a Service) paradigm, where companies outsource the security of their IT infrastructure to a trusted third party, which provides such security services remotely, without requiring on-premises hardware [17]. In this scenario, the SECaaS provider[3] acts as a front-end of a third party service. It may enforce, in place of a web service owner, a set of defensive measures, like TLS-protected channels and Web Application Firewalls. Moreover, it sells monitoring services, i.e. it observes all the connections and looks for anomalies, optionally, it can react to unexpected events. In a SECaaS scenario, DPI is usually not feasible for both technical (i.e. encrypted connections) and lawful reasons (e.g. compliance to the European GDPR). Instead, we envision the use of advanced detection techniques, like the ones we propose in this paper, which can gather relevant monitoring information from encrypted communications, even if recognising individual users is harder.

Another application scenario is cloud computing as browser identification can be used to monitor anomalies when accessing to SaaS (Software as a Service) instances. Moreover, the DDoS early detection also applies to software networks, as it can protect both SDN (Software Defined Networks) controllers and NFV (Network Function Virtualization) components [24], [33].

III. Traffic of interest

The motivation of our work is discovering if ML approaches are able to identify the software that has originated the traffic from the data captured in the network, instead of classifying the data itself.

In this paper, we will refer to *tool* as a specific software application able to produce traffic. For instance, Chrome is an instance of web browser whereas GrabSite is an instance of web crawler. *Tool instances* (also referred to as *tool versions*) are specific releases of a tool. For instance, Chrome version 68.0.3440.84 is a tool instance as well as GrabSite version 2.1.16. We have only considered the TCP connections generated by selected tools instances belonging to three *tool categories*: 1) web browsers; 2) web crawlers; 3) network stress tools. Indeed, our goal is to determine if tool identification is feasible, not to exhaustively classify all the possible tools, categories, and instances.

A. Tool categories

The three categories of tools we have considered all have their peculiarities that make them interesting for detection purposes.

1) Web browsers: Web browsers are the tools that users employ to surf the Internet. Moreover, they can also be used to access music and video streaming services. While consider superfluous to further details browsers and their features, as the reader is certainly aware of them, we report that web browsing and video streaming together constitute the major part of Internet traffic[4].

2) Web crawlers: Web crawlers, known also as spiders or spiderbots, are web applications that automatically browse web pages by recursively following hyper-links. Crawlers are used by web search engines (e.g. Google Search, Yahoo Search, DuckDuckGo) to download web pages that are later analysed to extract relevant keywords and metadata useful to index (or spider) them.

Web crawlers may cause resource consumption on the target web servers. They download several pages in a short time frame, often in parallel, and can revisit pages repeatedly in order to keep web search engines indexes synchronized with the actual state of the pages [8]. To mitigate this problem, the *robot exclusion standard*[5]has been defined. Websites developers may expose a text file, named `robots.txt`, to instruct web crawler, with machine-readable indications, about the exclusion of unnecessary or sensitive pages. Moreover, web crawlers that are allowed to index pages, can be filtered with the `User-Agent` field [12].

Nonetheless, crawlers must voluntarily abide to the indications listed in the `robots.txt` file. A misbehaving crawler may cause significant slowdowns in web server, potentially preventing legitimate users to access the hosted web pages. Even if a Web Application Firewall (WAF)[6], can filter the HTTP Requests by the `User-Agent` or other fields (e.g. IP addresses), such checks may be bypassed. A crawler may spoof the `User-Agent` fields, e.g. by using a web browser one, or send its requests through a proxy having a non-blacklisted IP address.

Finally, web crawlers may be support tools for dangerous attacks. *Web scraping* consists in automatically downloading a whole website to offline extract valuable data, e.g. to harvest the email addresses needed for marketing campaigns or phishing attacks[7]. Also, a copy of a website can be brought on-line, in an attack known as *Website forgery*, which consists in duplicating a web site to deceive users of the legitimate website to reveal sensitive information, e.g. login credentials of an e-banking website [32].

[3]As an instance of SECaaS provider, see Cloudfare https://www.cloudflare.com.

[4]https://www.sandvine.com/hubfs/downloads/phenomena/2018-phenomena-report.pdf

[5]http://www.robotstxt.org/orig.html

[6]https://www.owasp.org/index.php/Web_Application_Firewall

[7]https://www.owasp.org/index.php/Phishing

3) Network stressing tools: A Denial of Service (DoS) is an attack aiming to halt the fruition of a network services by its intended users [22]. For example, a DoS attack may target a web server to prevent access from legitimate users (volume-based attacks).

A DoS attack can be performed by sending a great amount of messages to the target machine to consume server resources (e.g. bandwidth, CPU time, memory), and cause a slow-down of the server until it is unable to provide the intended services. To deploy such attacks, a great bandwidth would be needed. To get around this limitation, attackers typically execute Distributed DoS attacks (DDoS). A great deal of machines launch a coordinated attack against a single target. The owners of computers involved in DDoS attacks may not be aware of it, for example if their machines have been infected by a malware that enables attackers to take control of them [34]. For example, the Low Orbit Ion Cannon (LOIC)[8] attack tool operates by flooding the target with TCP or UDP packets, providing also an *"hivemind"* feature to coordinate remote machines in executing a DDoS attack.

Another set of DoS attack tools targets a specific protocol, by exploiting some of its legitimate features (protocol attacks). For example, HTTP Slow DoS attacks aim at exhausting web servers resources by establishing a high number of low bandwidth connections with the victim [6]. Indeed, the attacker does not flood the victim, he slowly sends legitimate HTTP requests on connections that are forcefully maintained open for long periods through the HTTP Keep-Alive mechanism [9]). In this work, we considered the HTTP Slow DoS attacks worth to investigate, as they are difficult to recognize with existing methods without resorting to DPI.

Finally, another type of DoS attack exploits vulnerabilities of the application or daemon providing the service (application attacks). Typically, the attacker sends to the server non-standard messages that have been crafted to target a specific vulnerability. Depending on the vulnerability the attacker can also force the service to stop or the server to crash. As an example, we report a vulnerability of the Apache HTTP Server which has been used to execute DoS attacks[10]. The *mod_md* Apache Module, which implements automatic SSL certificate provisioning[11], can be forced by a specifically crafted HTTP request, to dereference a NULL pointer[12], causing a segmentation fault that halts the web server execution.

B. Tools and tool instances

We have selected different tools that we consider representative of each of the three tool categories (see Table I. In particular we have considered four of the most used browsers at the moment of this writing, namely, Chrome, Firefox, Edge, and Opera. We have considered more than one version of Chrome (48.0.2564.109 and 68.0.3440.84) and Firefox (42.0, 62.0, and 68.0) to perform additional tests of our detection abilities.

Moreover, we have considered five tools as representatives of the web crawlers. Three of them, namely Wget[13], Wpull[14] and Curl[15], are not web crawlers per se. Rather, they are command line browsers that are used into more sophisticated scripts and tool as web crawler engines. On the other hand, GrabSite[16] and HTTrack[17] are two open source programs that have been designed and engineered to backup entire web sites. Both tools are GUI-based, thus being usable also by non-experienced users, and are equipped with advanced features to optimize web crawling and disk usage. For instance, they avoid duplication of content shared among pages and endless loops, and may selectively ignore pages based on user-defined regular expressions. HTTrack is also able to update existing offline website copies, only downloading new or modified pages.

Finally, we have considered five tools in the category of the network stress tools, namely GoldenEye, HULK, RudyJS, SlowHTTPTest, and SlowLoris. For example, Slowloris[18] is a tool designed to perform HTTP Slow DoS attacks. HTTP Unbearable Load King (HULK)[19] and its evolution GoldenEye [20] operate in a similar way. They only differ on the method they use to keep the connections with the server opened, the former leverages the HTTP 1.0 Keep-alive header and the latter the HTTP 1.1 Cache-Control options[21]. RUDY (R-U-Dead Yet?)[22] is another tool that attacks the HTTP protocol. This tool analyzes the target web page to identify forms that can be filled by the user, and then performs a huge HTTP POST request to fill such form, sending it as slowly as possible, also in this case with the objective of keeping the connection with the web server open as long as possible. Finally, SlowHttpTest[23] tool is another tool that implements various techniques for HTTP DoS attacks, including the ones implemented by SlowLoris and RUDY.

Almost all of these tools are available both for Microsoft Windows and for Linux (maybe different versions), but Microsoft Edge that is only available for Windows Operating systems. Therefore, we have considered traffic generated by the same tools from both the OSes, as explained in Section IV. https://www.overleaf.com/8172285512rskjnrqvbfcg

C. Computing traffic statistics

Since we aim at detecting the tools that generate the traffic, also in encrypted channels, the payload cannot be used as it is not intelligible. Thus for our analyses we resorted to the flow statistics of the TCP connections. The statistics have been

[8] https://sourceforge.net/projects/loic/

[9] https://tools.ietf.org/id/draft-thomson-hybi-http-timeout-01.html#rfc.section.2

[10] http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-8011

[11] https://datatracker.ietf.org/doc/draft-ietf-acme-acme/

[12] http://cwe.mitre.org/data/definitions/476.html

[13] https://www.gnu.org/software/wget/

[14] https://github.com/ArchiveTeam/wpull

[15] https://curl.haxx.se/

[16] https://github.com/ArchiveTeam/grab-site

[17] https://www.httrack.com/

[18] https://github.com/gkbrk/slowloris

[19] https://github.com/grafov/hulk

[20] https://wroot.org/projects/goldeneye/

[21] https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html

[22] https://www.cloudflare.com/learning/ddos/ddos-attack-tools/r-u-dead-yet-rudy/

[23] https://tools.kali.org/stress-testing/slowhttptest

APPLICATION	CATEGORY	WINDOWS	LINUX
Chrome 48.0.2564.109	browser	✔	✔
Chrome 68.0.3440.84	browser	✔	✔
Firefox 42.0	browser	✔	✔
Firefox 62.0	browser	✔	✔
Firefox 68.0	browser	○	✔
Edge 42.17134	browser	✔	○
Opera 62.0.3331.66	browser	✔	○
GoldenEye 3.49.2	stress tool	✔	✔
HULK 1.0	stress tool	✔	✔
RudyJS 1.0.0	stress tool	✔	✔
SlowHTTPTest 1.6	stress tool	○	✔
SlowLoris 7.70	stress tool	✔	✔
Curl 7.55	web crawler	✔	✔
GrabSite 2.1.16	web crawler	○	✔
Httrack 3.49.2	web crawler	✔	✔
Wget 1.19	web crawler	✔	✔
Wpull 2.0.1	web crawler	✔	✔

TABLE I: Tools and versions we have selected for our experiments.

computed with the *TCP Statistic and Analysis Tool (Tstat)*[24], a traffic measurement tool developed by the *Telecommunication Networks Group*[25] of Politecnico di Torino. Tstat is able to compute a large number of traffic statistics, both on transport layer protocols (TCP and UDP) and on application layer protocols (HTTP, RTP, RTCP, XMPP). Tstat works both for *TCP Complete* and *TCP Nocomplete* flows. Flows that have been terminated by a packet with either the FIN/ACK or RST TCP flags set, or that may be deemed terminated since no packet belonging to the flow has been observed after a configurable timeout, fall into the first category, while other flows are deemed active and fall in the second category. Supporting *nocomplete* flows permits the use of our approach in cases where classification of live traffic is necessary, e.g. in IDS/IPS. Therefore, we concentrated on both types of flows, thus considered the statistics that are available for both TCP *complete* and *nocomplete* flows, which are contained in the *Core/Basic TCP Set*[26].

D. Research questions

The research objectives reported in Section II can be formulated in a more precise way in form of research questions (RQ):

RQ 1 *Is a system based on ML able to determine the category of the tool that generated the traffic?*

RQ 2 *Is a system based on ML able to recognize the tool that generated the traffic?*

RQ 3 *Is a system based on ML able to recognize an unknown version of a known tool and/or correctly categorize it?*

RQ 4 *Can a system based on ML be updated to recognize additional tools and/or tool versions?*

RQ 5 *Is a system based on ML able to perform the recognition implied by the previous research question on-line, i.e. before waiting that the connection is terminated?*

[24]http://tstat.polito.it/

[25]https://www.tlc-networks.polito.it/

[26]http://tstat.polito.it/measure.shtml#log_tcp_complete

Being able to positively answering the question RQ 1 is important as it can allow identifying network stress tools or web crawlers thus anticipating the impact of attacks of resource consumption.

Analogously, RQ 2 may determine if ML-based approaches allow identifying the tools that users are employing thus determining when users change their behaviour. Moreover, it can make attacks deducted by answers to RQ 1 even more precise.

RQ 3 asks if tools have a specific fingerprint and preserve it in different versions so that it is possible to identify the future versions of a tool without the need of performing the training of the classifiers. Note that this research question does not imply that we are interested in determining when a completely new tool, thus completely unknown to the classifiers, is producing the sniffed traffic, as we consider this task outside the scope of our work.

Answering RQ 3 could allow identifying important user behaviour related to the security, e.g. when they update browsers of when they revert to previous versions of tool for instance to avoid security restrictions.

In case new tools or versions of tools are of interest from the security point of view, it is important to know how easy is to maintain up to date the classifiers. An answer to RQ 4 would give the solution.

Finally, RQ 5 permits the understanding of the applicability of the classifiers to monitoring systems, that is, if they can be used live, in IDS, or they best fit post-mortem forensic analysis.

IV. DATA SET CREATION

The data set used to perform the tools' classification consists of the traffic statistics computed on the network captures we have collected. The data set preparation consisted in two phases: the acquisition of selected network captures and the computation of the statistics on these captures.

Note that, each TCP connection is a sample used to train or test our classifiers. The analysis of each flow in isolation allows our approach to be independent of the number of users simultaneously active at the moment of the detection. This characteristic is helpful also for detecting volumetric attacks, such as some denial-of-service attacks, since we can train a classifier in a small controlled environment and use it to detect larger scale DoS attacks.

A. Network captures acquisition

In order to have a data set with a significant level of variety, we decided to gather all our captures on two different machines with two different operating systems: Windows 10 and Debian/GNU Linux (with kernel 4.17.0, 4.18.0, and 4.19.0). All of our captures where performed using WireShark 2.6.4 by using `tshark`, its command line interface. Table I lists all the applications that we used to build our data set.

In order to have a more realistic scenario, we captured the browser traffic when we were manually surfing the Internet with Chrome, Firefox, Edge, and Opera. All the other captures where instead performed using ad-hoc Python 3 scripts, since

this is a more common scenario when performing DoS attacks or downloading an entire web site. For the command line tools we used the default parameters.

Being unable to find a testbench for the web sites to use for these kind of experiments, we have decided to randomly select to navigate web sites taken from the most popular ones[27]. We have not used any automatic approach. We avoided the use of browser automation frameworks (e.g. Selenium[28]) since they could introduce unwanted patterns in the navigation of the selected websites. The people involved in the captures was requested to manually navigate such sites following their normal behaviour. Moreover, to increase the variability of data acquisition, three different persons, after having navigated the most popular sites have been asked to to reach additional web sites of their choice. It should be noted that such websites embedded video streams (e.g. YouTube), thus the captures contain also multimedia content.

Once we gathered all the capture files, we cleaned them up by using the `tshark` filters in order to remove unnecessary background connections (such as UDP packets and connections not related to the tools in analysis).

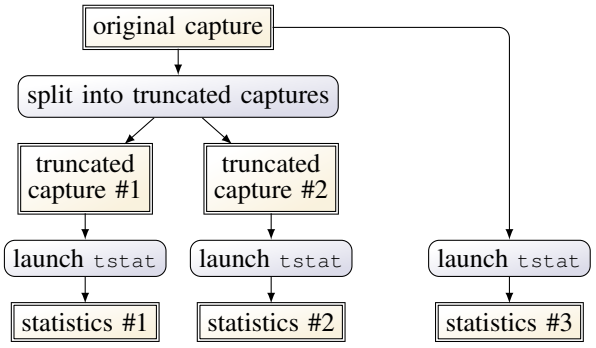


Fig. 3: Our network statistics extraction work-flow.

Since we want to verify whether our approach could work on-line, that is while a TCP connection is still open and not yet terminated, we created multiple truncated version of all our captures by following the work-flow depicted in Figure 3. In short, we used `tshark` to generate multiple truncated capture files for each one of our original capture files by putting a hard limit on the each packet arrival time, thus retaining only the received segments of each TCP connection[29]. Then, we launched `tstat` on each of the truncated captures files (and the original ones) to compute their network statistics.

`tstat` offers 44 statistics that are common to the two categories of TCP connections. However, since we are interested in performing a shallow packet inspection, we decided to ignore all the features that Tstat evaluates resorting to DPI techniques, thus reducing their number to 31. In addition to these values, we also added a new Boolean feature that reports if a TCP connection has been gracefully terminated or not. Table II summarizes the TCP flow statistics[31] that we decided to use as classification features and that we used to build our final data set from both the truncated and original capture files.

FEATURE	UNIT
# packets (both directions)	*packets*
# packets with payload (both directions)	*packets*
# retransmitted packets (both directions)	*packets*
# out of sequence packets (both directions)	*packets*
# packets with ACK set (both directions)	*packets*
# packets with ACK set and no payload (both directions)	*packets*
# packets with FIN set (both directions)	*packets*
# packets with RST set (both directions)[30]	*packets*
# packets with SYN set (both directions)	*packets*
# payload bytes excluding retransmissions (both directions)	*bytes*
# payload bytes including retransmissions (both directions)	*bytes*
# retransmitted bytes (both directions)	*bytes*
flow duration	*ms*
relative time of first payload packet (both directions)	*ms*
relative time of last payload packet (both directions)	*ms*
relative time of first ACK packet (both directions)	*ms*
TCP connection correctly terminated	*boolean*

TABLE II: TCP statistics used as classification features.

In order to increase the variability of our data set we decided to add connections generated by different versions of the same tools (e.g. Firefox 42.0 and 62.0) running under different OSes (i.e. Windows and Linux).

B. Data set statistics

By using the work-flow described in Section IV-A, we obtained 1219466 TCP flows of which 185547 are gracefully terminated and 1033919 are non-terminated. When building the data set we decided to discard all the connections only containing one packet since they carry too few information that can be used to train some machine learning algorithms.

Table III shows some average traffic statistics grouped by application. A quick look at the restricted number of statistics reported in Table III allows the formulation of simple yet important considerations.

- The browser connections are, in average, the longest ones (i.e. in average they last about half a minute, while all the other connections last at most five seconds) — this is probably due to the fact that browsers are usually human-driven[32], while all the other tools are automatized by means of scripts;
- Web crawlers request a lot of data (bytes) from target servers — this is coherent with their typical behavior, since they tend to download entire web sites indiscriminately;
- Network stress tools have a similar behavior to web crawlers thus the former cannot be easily distinguished from the latter by a quick visual inspection of the traffic statistics (thus the need for AI).

[27]https://en.wikipedia.org/wiki/List_of_most_popular_websites

[28]https://www.seleniumhq.org/

[29]We decided to use a timeout instead of truncating the TCP sequence number to keep potential out-of-order packets.

[30]Actually this can be only 0 or 1 since a proper TCP implementation will reset a connection after receiving a RST packet.

[31]Note that most table rows represent two statistics, one from the client to the server and another one in the opposite direction.

[32]This is respected in our data set since we collected the browser captures without using scripts or browser automation frameworks, even if they exists (e.g. Selenium at https://www.seleniumhq.org/).

TOOL INSTANCE	OS	COUNT	CLIENT SIDE		SERVER SIDE		DURATION
			PACKETS	BYTES	PACKETS	BYTES	$[ms]$
Chrome 48.0.2564.109	Linux	5,264	44.22	1,998.69	58.85	60,032.10	28,438.63
Chrome 48.0.2564.109	Windows	8,976	23.47	2,536.64	32.04	31,296.28	38,006.89
Chrome 68.0.3440.84	Linux	1,933	67.55	4,062.74	116.42	150,830.23	29,989.34
Chrome 68.0.3440.84	Windows	8,153	20.90	2,675.26	29.06	30,445.84	40,167.00
Edge 42.17134.1.0	Windows	28,913	23.14	2,024.09	21.69	21,782.88	12,693.36
Firefox 42.0	Linux	2,551	57.10	4,849.41	89.16	105,808.44	33,218.93
Firefox 42.0	Windows	6,142	29.57	2,462.07	43.92	50,346.47	21,035.63
Firefox 62.0	Linux	2,658	62.86	4,360.09	90.95	107,950.76	37,402.74
Firefox 62.0	Windows	8,323	44.66	3,039.47	66.16	77,539.21	32,559.49
Firefox 68.0	Linux	819	54.05	2,070.60	79.80	100,400.66	18,579.62
Opera 62.0.3331.66	Windows	8,953	25.25	1,913.91	47.86	53,401.82	40,414.49
Curl 7.55.1	Windows	1,604	31.20	649.63	53.38	65,814.02	431.33
Curl 7.61.0	Linux	1,434	67.34	416.75	90.77	119,766.43	806.24
GrabSite 2.1.16	Linux	3,666	368.80	3,928.34	582.88	1,802,525.31	15,423.33
Httrack 3.49.2	Linux	10,624	14.84	834.19	16.92	16,901.24	802.33
Httrack 3.49.2	Windows	2,975	22.07	1,637.59	36.44	44,863.53	8,644.35
Wget 1.11.4	Windows	2,849	92.61	1,024.64	184.09	249,312.02	2,076.60
Wget 1.19.5	Linux	2,234	176.12	3,549.99	326.44	430,592.15	3,624.39
Wpull 2.0.1	Linux	1,374	139.08	1,313.92	204.50	284,275.69	8,342.97
Wpull 2.0.1	Windows	1,112	86.31	1,146.56	226.41	310,694.03	8,824.09
GoldenEye 2.1	Linux	123,662	16.10	854.75	30.23	36,591.57	2,318.79
GoldenEye 2.1	Windows	664,133	12.49	789.77	19.36	21,910.12	1,240.32
HULK 1.0	Linux	47,130	7.93	782.25	6.66	4,885.02	778.93
HULK 1.0	Windows	246,465	5.29	533.41	4.18	2,233.53	6,890.59
RudyJS 1.0.0	Linux	1,452	13.13	731.00	12.01	3,646.56	17,576.82
RudyJS 1.0.0	Windows	2,116	10.10	701.99	10.30	3,133.69	14,530.37
SlowHTTPTest 1.6	Linux	10,999	8.64	1,406.35	6.82	3,492.74	11,969.76
SlowLoris 0.1.4	Linux	6,367	5.40	164.22	3.97	48.30	13,392.93
SlowLoris 0.1.5	Windows	6,585	5.16	165.01	3.81	47.43	13,881.58

TABLE III: Network statistics of examined captures per tool instance (average).

V. TRAINING AND RESULTS

We trained three different random forests for classifying the network traffic in the three categories listed in Section III (web browsers, web crawlers and network stress tools), another one for identifying the traffic generator tool and the last one for detecting the specific tool instance. Random forests are the implementation of the idea of ensemble learning, combining weak classifiers of the same type to produce a strong classifier. Random forest implement ensemble learning with decision trees [3], built in a random fashion, with each decision tree trained using only a random subset of features and training sets samples. They have been firstly introduced by Ho [16], and have been extended by Breiman [4] with the concept of *bootstrap aggregating*, also known as *bagging*, which is used to increase accuracy in case of noisy data and avoid overfitting. They are robust, versatile, quite resilient to overfitting and handle well noisy data, making them an excellent choice for building a network traffic classifier.

We performed the classifier training and testing using MATLAB 2019a (via the `TreeBagger` class) on our university HPC cluster named Legion[33].

To verify the research questions posed in Section III-D, we designed all our experiments in order to evaluate if our classifiers were able to:

- classify new flows generated by tools included in the training set;
- identify a completely unknown tool not included in the training set – in particular we wanted to assess the ability of our 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 – in particular we evaluated our random forests abilities to identify Firefox 68.0, given that our classifiers were trained using some connections generated by Firefox 42.0 and 62.0;
- check if the classifiers, after adding to the training set the information about a new version of a tool improves or maintains the detection abilities of the classifiers on the other tools.

[33]See http://hpc.polito.it, whose main information is reported in Table IV for additional information.

INTERCONNECTION	Infiniband EDR 100 Gib/s
SERVICE NETWORK	gigabit Ethernet 1 Gib/s
CPU	2x Intel Xeon Gold 6130 @ 2.10 GHz
NODES	14
CORES	448
RAM	5.376 TB DDR4 registered ECC
OS	CentOS 7.6.1810 with OpenHPC 1.3.8.1
SCHEDULER	SLURM 18.08.8

TABLE IV: PoliTO's Legion HPC cluster information.

A. Training classifiers

We performed the training of our classifiers in three consecutive steps. First, 1) we split the data set in a training and test sets; 2) we searched for the random forests' optimal

hyper-parameters; 3) we performed the actual random forests training.

1) Training and test sets: In a first phase, we trained the classifiers with all the connections generated by the tools reported in Table I but Firefox 68.0, Opera, GrabSite and SlowHTTPTest.

The training set has been built from the entire data set after having performed these steps:

1) we removed all the TCP connections that had been generated by using the Firefox 68.0, Opera, GrabSite and SlowHTTPTest;
2) we picked at random 80% of the remaining TCP flows in the original non-truncated capture files;
3) we added in the training set all the TCP flows selected during the previous phase and all their truncated versions.

After these steps, the training set consisted of 1075520 samples, while all the remaining ones become our test set (143946 samples). This method ensures that no potential information is shared between the two sets, since a TCP connection and its related truncate versions can only be used for the classifier training or during its a-posteriori analysis, but not for both.

We have then generated two different test sets:

- the *Known Tools test Set* (KTS), containing the test set samples related to the applications also present in the training set (this collection has 119321 observations);
- the *Unknown Tools test Set* (UTS), consisting of only the test set samples generated by Opera, GrabSite, SlowHTTPTest and Firefox 68.0.

2) Hyper-parameter optimization: Choosing the right hyper-parameters is an important aspect that can significantly increase the accuracy of a classifier. Their choice is non-trivial, since it can depends on a multitude of factors. Fortunately, a good selection of hyper-parameters can be obtained by treating this problem as an optimization one, where the variables are the hyper-parameters and the objective function is the maximization (or minimization) of a classification statistic. Instead of maximizing the classification accuracy, we decided to maximize the average 5-fold cross-validation [19] R_k statistic [15], which is the extension of the Matthews correlation coefficient to multi-class problems. Having a $k \times k$ confusion matrix C, this value can be computed as:

$$\begin{cases} R_k(C) & = \frac{N(C)}{D_1(C) \cdot D_2(C)} \\ N(C) & = \sum_l \sum_m \sum_n \left(C_{l,l} \cdot C_{m,n} - C_{l,m} \cdot C_{n,l} \right) \\ D_1(C) & = \sqrt{\sum_l (\sum_m (C_{l,m}) \cdot (\sum_{n|n \neq l} \sum_o C_{l,o}))} \\ D_2(C) & = \sqrt{\sum_l (\sum_m (C_{l,m}) \cdot (\sum_{n|n \neq l} \sum_o C_{o,l}))} \end{cases}$$

The maximum value of R_k is 1, while the minimum is between -1 and 0 and depends on the data set class distribution. 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 our case), differently from several metrics such as the classification accuracy. Second, it is somehow more informative than traditional performance measures, since it takes into account all the possible classifications and misclassifications in the confusion matrix. The F-score, for instance, ignores the true negatives, therefore optimizing this statistic can potentially lead to unbalanced classifiers.

We used a process of Bayesian optimization to compute the ideal hyper-parameters by maximizing the R_k coefficient [23]. We performed a hyper-parameter optimization with a conservative timeout of 24 hours (using 32 cores on the HPC), even if our optimization logs showed that the R_k statistic started to become stable after about a couple of hours.

In particular we chose to optimize the following hyper-parameters: 1) the number of trees; 2) the number of features for training each tree; 3) the fraction of the training set used for training each tree; 4) the maximum number of splits per tree; 5) the algorithm used to compute the splits (maximizing the Gini's diversity index or the deviance reduction [3]); 6) the minimum number of samples per leaf. Table V summarizes the results of this optimization phase for the three classifiers.

3) Random forest training: Once found the optimal hyper-parameters, we used them for the actual training of the random forests.

B. Category classification

We trained a random forest for classifying the traffic in the three categories listed in Section III: web crawlers, network stress tools and web browsers.

Training time on the HPC took about 13 minutes. Table VI reports the performance metrics we have computed on the training set, via a 10-fold cross-validation, with the known tools test set. The precision, sensitivity, specificity, AUC and F-score metrics were devised to validate only binary classifiers, so we extended them to a multi-class problem by using the macro-averaging technique [28]. That is, we computed four values for each statistic by treating our random forest as a collection of four binary classifiers (one for each class), and averaged together these values to obtain the final result.

Table VII reports the confusion matrix computed on the known tools test set and shows that we can classify a flow with excellent accuracy. The precision, sensitivity (also known as recall) and specificity have high values, meaning that our classifier has a very low false positive/negative rate. Since our initial data set is imbalanced, the classification accuracy is not the most suited metric to validate our classifier. For this reason, we included also the AUC (Area Under Curve) statistic, which is not dependent on the data set samples distribution.

Browsers, web crawlers and network stress tools produce very similar connections (all in all, they are TCP connections). Sometimes they make use of the same underlying network libraries. Despite these similarities, they are usually correctly classified by our random forest. Only the web crawlers show inaccurate results, as they are confused with browsers in about 10% of the cases. Nonetheless, this behaviour is not unexpected, as crawlers and browsers download data by sending proper HTTP requests. It is interesting to note, however, that network stress tools are confused more often with browsers than with web crawlers.

Figure 4 plots the AUC of our classifier depending on the number of packets in the examined flow on KTS.

The general trend is that the classifier performance tends to increase as the number of exchanged data increases. By

HYPER-PARAMETER	CATEGORY CLASSIFIER	TOOL CLASSIFIER	INSTANCE CLASSIFIER
number of trees	194	61	51
number of features for training each tree	22	10	14
training set percentage for training each tree	85	84	99
maximum number of splits per tree	126184	143560	97336
algorithm used to compute the splits	GDI	GDI	GDI
minimum number of samples per leaf	2	2	14

TABLE V: Optimal hyper-parameters for the three classifiers.

STATISTIC	CATEGORY CLASSIFIER			TOOL CLASSIFIER			INSTANCE CLASSIFIER		
	TRAINING	CV	KTS	TRAINING	CV	KTS	TRAINING	CV	KTS
accuracy [%]	99.57	98.96	98.73	96.98	96.00	95.47	95.05	94.29	93.21
precision [%]	96.65	93.86	92.30	85.21	81.17	77.86	78.46	74.30	67.79
sensitivity [%]	99.79	96.27	93.61	98.45	92.62	86.67	98.09	92.46	82.42
specificity [%]	99.85	98.69	98.34	99.66	99.52	99.45	99.62	99.56	99.48
AUC [%]	99.82	97.48	95.98	99.05	96.07	93.06	98.86	96.01	90.95
F-score [%]	98.18	95.03	92.93	89.86	85.27	81.25	85.86	81.25	73.45
R_k	0.97	0.93	0.92	0.94	0.92	0.91	0.91	0.89	0.87

TABLE VI: Performance metrics for our classifiers.

		PREDICTED		
		BROWSER	NET STRESSER	CRAWLER
ACTUAL	BROWSER	6994	289	119
	NET STRESSER	638	108759	161
	CRAWLER	205	100	2056

TABLE VII: Confusion matrix of the category classifier on KTS.

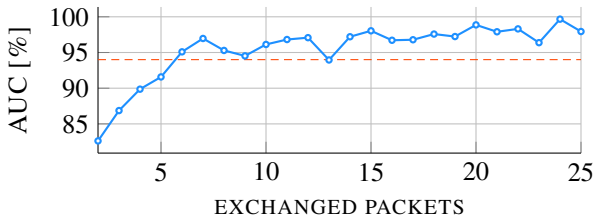


Fig. 4: AUC of the category classifier vs. exchanged packets.

only looking at the first two packets of a TCP connection, our random forest has an AUC of 82.61%, showing a mediocre performance. However, from the second packet onward, the accuracy increases significantly. Once a connection reaches six exchanged packets, the AUC stabilizes, being above 94% (indicated by the horizontal red line in the plots). That means that, for detection purposes, a sensor equipped with our classifiers should observe at least six packets to be able to identify the traffic originators in a trustworthy manner.

C. Tools classification

We also trained a second random forest to classify the TCP streams according to their generator tools (with the optimal hyper-parameter reported in Table V). This classifier identifies the eleven applications listed in Table III, namely Chrome, Curl, Edge, Firefox, GoldenEye, Httrack, HULK, RudyJS, SlowLoris, Wget and Wpull. The total training time of the random forest was about four minutes on our HPC.

Table VI reports performance statistics computed on the training set, with a 10-fold cross-validation approach and on KTS. The task of identifying the tool that produces a TCP stream is intuitively harder than only detecting the categories. This conjecture has been proved by our experiments, the overall performance of this classifier are lower with respect to the one presented in Section V-B. Nonetheless, results are satisfactory as we obtained an AUC of 93.06% on the KTS.

Tables VIII and IX respectively report the classifier confusion matrix computed on the KTS observations where the last column contains the accuracy per row and the confusion matrix where the rows and columns were grouped by category. All the applications can be identified with an accuracy greater than 76%. The network stress tools (GoldenEye, HULK, RudyJS and SlowLoris) can be detected with a very high accuracy, even if the classifier only analyzes a single flow at time to reach its decision. For detection purposes, evaluating the concurrent presence of multiple flows generated by the same network stress tool can allow performing an easily and early detection of the beginning of a DoS attack.

On the other hand, our random forest exhibits less precision when predicting browsers. This might be due to the variability introduced by the human intervention on the TCP stream features. A user can in fact close a connection before its termination (e.g. before a video has been completely downloaded) or keep it open, while all the other tools have a more deterministic approach, allowing a better classification.

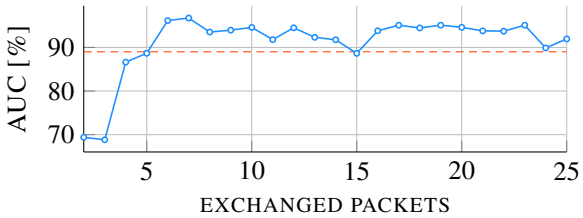


Fig. 5: AUC of the tool classifier vs. exchanged packets.

Also in this case, we report in Figure 5 the AUC trend when the number of exchanged packets increases. As for the classification of tool categories, the general trend is that the

ACTUAL \ PREDICTED	CHROME	EDGE	FIREFOX	GOLDENEYE	HULK	RUDY	SLOWLORIS	CURL	HTTRACK	WGET	WPULL	ACC. [%]
CHROME	1917	191	128	116	23	0	2	7	9	5	10	79.61
EDGE	97	2715	116	40	5	0	0	8	3	16	5	90.35
FIREFOX	244	76	1516	71	18	0	8	19	26	2	9	76.22
GOLDENEYE	736	169	310	75931	1132	66	1	16	106	21	374	96.28
HULK	274	34	103	389	28262	10	1	0	8	11	27	97.06
RUDY	1	1	4	3	2	267	0	0	3	2	4	93.03
SLOWLORIS	0	0	12	0	0	0	1253	6	19	0	0	97.13
CURL	9	14	8	7	1	1	0	270	5	7	4	82.82
HTTRACK	14	15	9	12	5	0	5	5	1229	0	0	94.98
WGET	11	8	22	25	3	2	0	20	4	367	11	77.59
WPULL	10	12	13	33	3	0	0	2	5	7	183	68.28

TABLE VIII: Confusion matrix of the tool classifier on KTS.

ACTUAL \ PREDICTED	BROWSER	NET STRESSER	CRAWLER
BROWSER	7000	283	119
NET STRESSER	1644	107317	597
CRAWLER	145	97	2119

TABLE IX: Confusion matrix of the tool classifier on the KTS, grouped by category.

classification performance increases as the number of packets increases. After about 6 exchange packets the AUC stabilizes with a value that is always above 89% (showed by the red line in the plot). With respect to the category classification, however, the performance is slightly lower, most likely due to the higher difficulty of the classification task.

D. Tool instances classification

Finally, we created a random forest to identify all the 16 tool instances in our training set. Table V reports the optimal hyper-parameters of our classifier, which took about three minutes and half to train on our HPC cluster.

Table VI reports performance statistics computed on the training set, using a 10-fold cross-validation approach and on the KTS. The tool instance random forest shows an adequate performance that is, however, less accurate than the other classifiers. This is further evident in the confusion matrices depicted in Table X and XI.

ACTUAL \ PREDICTED	BROWSER	NET STRESSER	CRAWLER
BROWSER	7000	229	173
NET STRESSER	3101	105750	707
CRAWLER	151	75	2135

TABLE X: Confusion matrix of the instance classifier on the KTS, grouped by category.

Identifying the tool and also its version number is a hard task since it depends on a variety of factors, such as changes in the code of the tool as it evolves over the time, but also the introduction of new network libraries. Nonetheless, most of the tools can be detected with a good level of accuracy (above 80%), but some are apparently stealthier than other. Web crawlers are the easiest ones to identify, indeed, all the predictions are correct in at least the 94%. On the other hand, browsers are the most difficult ones for the classifiers. However, by looking at the data, they are often confused with other browsers. For instance, Firefox 42.0 is most of the time misclassified as Firefox 62.0 and viceversa. Chrome is more easily confused with Edge than with Firefox and, surprisingly, with Goldeneye. We also noted that, in general, Goldeneye, is easily confused with browsers, as well as HULK. Both should have something in common, also with browsers, as Goldeneye is confused the most with HULK and viceversa.

We have considered the Wget version 1.11 that has been released 9 years before the version 1.19[34]. They do not seem to have a common fingerprint. The same consideration applied to Curl. Even if the version 7.61.0 has been released about a year after the 7.55.1[35], there have been several releases that may have hidden the fingerprints. On the other hand, for SlowLoris we have considered two consecutive versions (0.1.5 two years after 0.1.4[36]), they are correctly identified or confused between each other. Finally, the accuracy of the classifier is in general lower when instances of the same tools are confused with each other. Therefore, research is needed to evaluate classification for real world usage, i.e. with larger training sets containaing several instances of several tools.

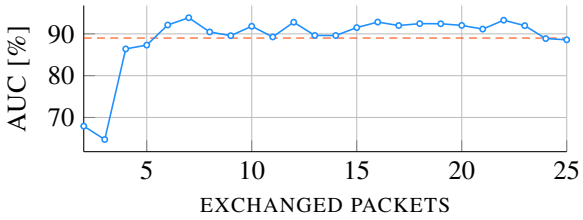


Fig. 6: AUC of the instance classifier vs. exchanged packets.

Figure 6 reports the plot of the macro-averaged AUC of the tool instance classifier vs the exchanged packets. As in the other two cases, six packets are usually enough to have a stable AUC, which never goes below 89%.

E. Classification of unknown tools

Classification of the tools were not used during the training phase (i.e. Firefox 68.0, Opera, GrabSite and SlowHTTPTest),

[34]https://en.wikipedia.org/wiki/Wget
[35]https://curl.haxx.se/docs/releases.html
[36]https://pypi.org/project/Slowloris/#history

ACTUAL \ PREDICTED	CH48	CH68	ED	FI42	FI62	GO	HU	RU	SL0.1.4	SL0.1.5	CU7.55.1	CU7.61.0	HT	WG1.11	WG1.19	WP	ACC. [%]
CH48	1022	134	125	25	46	45	8	0	0	4	4	0	10	6	6	10	70.73
CH68	110	639	70	32	44	43	10	0	0	2	2	0	5	0	1	5	66.36
ED	124	58	2608	71	47	32	5	1	0	2	2	13	8	9	18	7	86.79
FI42	67	57	64	519	120	26	5	0	0	7	8	3	14	0	4	8	57.54
FI62	47	52	24	151	744	27	6	0	0	6	13	0	12	0	2	3	68.45
GO	438	1070	360	183	431	74116	1427	207	0	14	26	6	204	68	31	281	93.98
HU	110	302	59	48	86	413	28004	29	0	5	1	1	29	10	9	13	96.17
RU	0	1	1	0	2	1	2	271	0	1	0	0	3	2	0	3	94.43
SL0.1.4	0	0	0	0	0	0	0	0	559	8	0	0	0	0	0	0	98.59
SL0.1.5	0	0	0	7	3	0	0	0	14	679	2	0	18	0	0	0	93.91
CU7.55.1	3	0	0	1	3	4	0	0	0	0	129	0	2	1	0	2	88.97
CU7.61.0	5	0	5	3	1	0	0	1	0	0	1	150	3	6	2	4	82.87
HT	4	15	18	3	8	6	5	0	0	5	4	0	1225	0	1	0	94.67
WG1.11	5	3	0	9	0	16	1	3	0	0	4	5	2	196	2	3	78.71
WG1.19	1	10	0	0	15	0	0	0	0	0	0	7	1	3	183	4	81.70
WP	7	7	15	2	8	24	4	3	0	3	0	0	12	7	2	174	64.93

TABLE XI: Confusion matrix of the tool instance classifier on KTS (where CH = Chrome, CU = Curl, ED = Edge, FI = Firefox, GO = Goldeneye, HT = HTTtrack, HU = HULK, RU = RUdyJS, SL = Slowloris, WG = Wget and WP = Wpull).

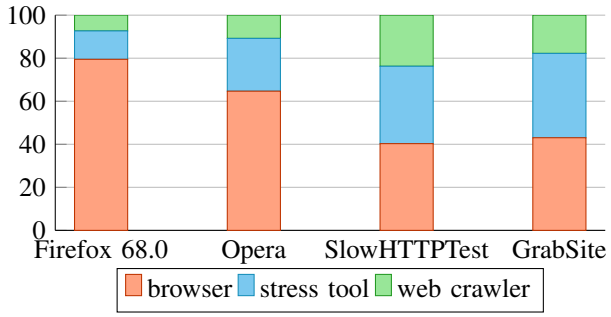


Fig. 7: Classification of the unknown tools.

however, showed mixed results. Figure 7 shows the classification results of the UTS.

On the one hand, we note that Firefox 68.0 and Opera samples have been identified as browser respectively with a fraction of 79% and 65%, which is a good level of accuracy for tools that were not considered during the training phase of our classifiers. This proves that the behaviour of browsers is understandable by a machine, which is thus able to recognize the same patterns in the traffic generated by other tools. A slightly better performance on Firefox than with Opera can be justified by the fact that the training set included two previous versions of the same browser, while the Opera browser was completely unknown for our classifiers.

On the other hand, Grabsite and SlowHTTPTest were not correctly classified as a web crawler and a network stress tool, the accuracy was 18% and 36%, respectively. Our interpretation of these results is that the network fingerprints of web crawlers and network stress tools appears to be very specific of the tool that generated the examined packets. For instance, an interesting example that seem to confirm the hypothesis of tool-instance specific fingerprints derives from the classification of SlowHTTPTest, which internally uses the the same basic approach of RudyJS and Slowloris. Nonetheless, even if we included several samples of these two DoS applications, our classifier had great difficulty in recognizing it as another instance of a network stress tool. This is probably related to the fact 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.

Firefox 62.0, present in the training set has been classified as a browser in 97% of the cases in the KTS, while Firefox 68.0, unkown to the classifier, is only correctly identified with an accuracy of 79%. We can then conclude that close versions of the same browser already show evident differences in the traffic they generate to be able characterize their fingerprint.

Finally, GrabSite completely fooled our classifier. Since it was recognized as a web crawler only in 18% of the cases.

F. Updating the training set with new products

We added Grabsite, SlowHTTPtest, and Opera in the training set and trained the classifiers without re-optimizing the hyper-parameters, to simulate quick-and-dirt updates of the classifiers (e.g. fast reaction in critical conditions). As expected, the new classifier was less accurate (with an AUC of 93.25%): Grabsite was correctly recognised as crawler in just 14% of the cases, Opera as browser in 66% of the cases, and SlowHTTPtest in just 36%. Discussing the causes for these poor results was uninteresting (e.g. not enough trees, too small trees), as we proved that there are cases where quick-and-dirt updates do not work. Nonetheless, after performing the full training process (as in Section V-A), the performance of the classifier were comparable to the performance of the classifier trained on KTS only (accuracy on the test set was 98.43% and the AUC was 95.70%) and Grabsite, SlowHTTPtest, and Opera were properly predicted with accuracy 87.98%, 86.18%, and 97.04%, respectively.

VI. Use of classifiers for monitoring

The next step of our work has been to evaluate how the results produced by our classifiers could be used for monitoring purposes. We present here a set of examples of detection rules, using classifiers outputs, for a standard IDS architecture inspired by the Zeek (formerly known as Bro) framework. The effectiveness of the detection abilities has been evaluated based on the classifier accuracy measurements. The classifier would generate three detection events:

- *category recognized*: when, after reading n_c packets, the classifier decrees that the `connection` has been generated by a tool in the `category_name`
  `event (category,connection,category_name)`
- *tool recognized*: when, after reading n_t packets, the classifier decrees that the `connection` has been generated by `tool`
  `event ( tool, connection , tool_name )`
- *instance recognized*: when, after reading n_i packets, the classifier decrees that the traffic of `connection` has been generated by `tool_instance`
  `event ( instance, connection , tool_instance )`

The parameters n_c, n_t, and n_i are considered a configuration parameter of the sensors generating the events. Based on these events we have first investigated the possibility to perform an error detection using the three generated events to improve the detection abilities (i.e. reduce false negatives and false positives).

We have considered three Boolean values (obtained by comparing predicted vs. real values) for the cross validation: 1) the inferred category is correct, 2) the inferred tool is correct, 3) the category of the inferred tool equals the inferred category 4) the inferred tool instance is correct, and 5) the category of the inferred tool instance equals the inferred category, and 6) the tool the inferred tool instance is a version of, equals the tool inferred by the tool classifier.

The Boolean values obtained by comparing guesses from different classifiers are important as they allows determining (when working on test cases or real data) if classifiers are in disagreement. Nonetheless, this check cannot recognise cases where all the classifier are wrong in a coherent way, e.g., no errors are visible if Grabsite is confused as (browser, Firefox, and Firefox 42.0) by the category, tool, and instance classifiers. By considering all the connections regardless of the number of exchanged packets, in about $C_2 = 95.2\%$ of the cases both tool and category were correct, and all the three classifiers were correct in $C_3 = 92.6\%$ of the cases. By only considering category and tool classifiers, in about the $I_2 = 1.8\%$ of the cases it was possible to infer that were errors in one of the two classifiers, and only $S_2 = 3\%$ were stealth to the cross check. By using excluding the cases where errors are detected, the tool classifier increases its performance from 95.47% to 96,97%, however, for 14.94% of the cases recorded as errors the classifier was correct (overall 0.44%).

By also considering the Boolean relations concerning the instance classifier, $I_3 = 4.7\%$ were recognizable errors in one of the three classifiers and $S_3 = 2.7\%$ were stealth errors. In this case, by ignoring errors, the performance of the instance classifier increases from 93.21% to 95.67% and 9.89% of the errors were indeed correct predictions of the instance classifier (overall 0.27%). If only connections longer than 10 packets are considered, the previous six percentage become $C_2 = 97.8\%$, $C_3 = 95.6\%$, $I_2 = 0.5\%$, $S_2 = 1.6\%$, $I_3 = 2.4\%$, and $S_3 = 1.6\%$. Moreover, by excluding the errors, the classification was correct in 97.36%.

Building on the classifiers, we can write a few correlation rules that tell that an alarm needs to be raised:

- if more than X ‘category recognised events’ in the last T seconds report a network stress tool,
- if more than X ‘tool recognized events’ in the last T seconds report a tool in the network stress tool category
- if more than X ‘instance recognized events’ in the last T seconds report an instance of a tool in the network stress tool category (same tool or different tools may have different thresholds)
- if more than X ‘category recognised events’ in the last T seconds report a web crawler
- if more than X ‘tool recognized events’ in the last T seconds report a web crawler from the same IP/IP range
- if the ‘category recognised events’ report non-browsers from IP in the list of IP addresses/subnets where you don’t expect to have connections made by non-browsers

VII. Analysis of results and research directions

We summarize here the main findings of our research and discuss how they allow answering the Research Questions we have asked in Section III-D.

Yes we can identify categories. (RQ 1). Our results prove that (supervised) classifiers trained with traffic generated by a set of tools can reliably determine the category of the tool that generated the new traffic, only if the traffic was generated with a tool already considered in the training set. On the other hand, determining the category of a tool that was not considered in the training set is challenging, as classifiers only worked reasonably good for browsers whereas they were completely wrong for network stress tools and web crawlers.

Yes we can identify tools, but less accurately (RQ 2).. Our results proved that if (supervised) classifiers are trained with traffic generated with a set of tools, they can reliably assign the sniffed traffic to the originating tools, provided that these tools were in the training set. However, these classifiers are not as accurate as the ones that just determine the category. Determining other versions of the tools considered in the training seems challenging, yet not impossible, at least for browsers. However, more extensive studies would be needed to confirm these hypotheses. Also in this case, we envision improvements from the use of semi-supervised methods. Indeed, recognising tools that are not in the training set can improve the accuracy and effectiveness of decisions of monitoring systems.

Identify tool instances is more difficult (RQ 3). Our results proved that supervised ML-based classifiers trained with traffic generated with a set of tools can identify the version of the tool that produced the traffic with reasonable accuracy. It is very worth investigating how the accuracy of classifiers changes if several instances of the same tool

are considered (e.g. from 3 to 20/30 versions of firefox and Chrome). Indeed, it would be interesting to check the evolution of the software fingerprint as well as to improve the performance of a correlator that cross-checks these values to make more precise decisions.

Early DoS detection is feasible. We have noted that network stress tools that are considered in the training set manifest a clear fingerprint that the classifiers are able to detect. Therefore, we conclude that it is possible to determine if a site is under a volume-based or protocol DoS attacks. This detection can be performed very early and the confidence of this detection can be even bigger if data from concurrent classification of more flows are aggregated. Nonetheless, more effort is needed to understand if a similar approach can work for another class of DoS attacks that is not covered by our work: application attacks.

Early but not too early (RQ 4). Our data proved that a reasonably accurate prediction requites at least 5 packets and AUC stability is reached after 10 packets. An IDS/IPS should not make decisions based on classifications made on few packets. However, classifiers only work on a single flow. An IDS/IPS considering the global security status of the network in the last time frame can achieve much better correlations and decisions. Designing an IDS correlator that properly leverages this kind of classifiers is a challenging (industrial) research that can impact on monitoring.

A browser is always a browser or a human being is always a human being (RQ 2,RQ 3)? Browsers have been recognized also when their data sets were not considered in the training set. A possible explanation is that the way browsers interact with web sites is standard, e.g. prefetching pages, loading complex pages with multiple requests. Nonetheless, our data do not allow us to establish if these peculiarities are of the tools and their communication libraries or they are generated by the fact that they are directly driven by humans. Further research is needed to better motivate this evidence and to separate human patterns from tools patterns, which may be useful for monitoring.

New web crawlers and network stress tools, new fingerprint, new training (RQ 4). Unsurprisingly, our result showed that our classifiers, based on supervised methods, have troubles in classifying unknown tools (which are not browsers). Unsupervised or semi-supervised methods should be tested in these cases.

New tool version almost always implies new training (RQ 5). Our results showed that tools may have a fingerprint and this fingerprint may occasionally be preserved in the next versions. Also in this case a disclaimer applies as this result was more evident for browsers where the "human effect" cannot be isolated. It would have been desirable to have specific tool fingerprints, which propagate through versions. However, to have accurate classifiers we are condemned to train supervised classifiers with all the tools of interest to make them effective for monitoring purposes. Further studies would be needed to better characterize how this fingerprint propagates in close version (e.g. 62.0 vs. 63.0) for longer periods of time (e.g. in the last 10 years). And research would check if unsupervised and semi-supervised methods can see patterns in the data that allow classifiers to properly label unknown tools and versions.

Quick updates do not work. As largely expected, if new tools are added into the training set, classifiers trained reusing previously computed hyper-parameters have poor performance. Only when a full training is performed, the classifiers perform well.

Malware may be detected if it uses known tools and libraries. Our analysis proved, that tools and libraries can be easily recognised. Therefore, training classifiers with connections generated by malware and by libraries the malware uses, can allow detecting when connections are generated by these really unwanted tools. In this case, data sets with malware communications can be used in future research to prove this hypothesis, which resembles what already done in literature for detecting skype and attacks (see Section VIII).

User habits only for known tools. In corporate scenario, where a reliable association exists between IP addresses, being able to recognise tool instances can help administrators in determining when old browser with known vulnerabilities are used. An interesting improvement may consider adding semantic information to determine anomalous users' behaviour, like the average level of competence of people working hours, and the knowledge (and history) of tools used.

Detect (browser) liars. Given the reasonably good accuracy of our classifiers, whenever the decrypted payload is accessible to a monitoring system or a security control (e.g. re-encryption or at the endpoint servers or cleartext communications), a comparison is possible, e.g. to determine potential liars and attacks aiming at bypassing local HTTP filters. An analysis could determine all the cases where this comparison is beneficial and how such comparisons can impact the security and its performance in terms of false positives and negatives.

Threats to validity. We have identified a set of threats to validity of our results. First, we have only covered a limited number of tool categories. It is not guaranteed, but plausible, that similarly trained classifiers can be as accurate as the ones presented here. The same consideration applies to classifiers for tools and tool instances. Selected different tools and different versions of the tools could lead to different accuracy results. Also, considering very large sets of tools and considering several instances of the same tools can change the accuracy results. However, the fact that the methodology that we have adopted follows the best practice ensures that Our classifiers are based on random forests. Other classifiers may work better. Nonetheless, we have selected them based on previous analysis on effectiveness of classical ML approaches and performed an internal validation against neural networks. Then, all our classifiers have been built using Matlab. Python-based approaches could reach slightly better results. Nonetheless, better classifiers may only confirm the correctness of this approach and reduce the impact of current limitations. Furthermore, we did not isolate, by design, the effect of the tools 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 in performing a more careful analysis. Finally, the considerations about the application of the classifiers to

monitoring (as in Section VI) miss an additional evaluation in a real context. However, the absence of labels in real traffic data and the limited coverage of tools the classifiers is currently able to identify compared to the real traffic (together with the network security policy of our institution) would have made this validation completely significant.

VIII. Related works

This section surveys related traffic classification techniques and its applications to encrypted communications. Traffic classification used to rely on port number. Since their accuracy decreased with the advent of dynamic ports, new classifiers relied on Deep Packet Inspection (DPI) techniques. The advent of encrypted traffic, however, reduced the accuracy of these techniques too. As a result, methods based on traffic flow statistics and time series analysis stand out as they only use unencrypted headers.

Reported works despite belonging to this latter category, are very different from each other. They differ for the goal of the classification, the type of feature employed, and for the Machine Learning (ML) method adopted. First, the goal of the classification strongly depends on the problem to solve, which may be the identification of a specific application generating the data flow (Skype, Torrent, Chrome, Facebook), the traffic type (browsing, video chat, downloading), the protocol type (UDP, TCP, FTP), the destination website (Google, YouTube), and the type of ongoing attack (Crawling, R2L, U2R, DOS, Probe). Second, the type of features employed in the classification may vary a lot. Following the classification given by Rezaei et al. [27], there are three types of input features commonly used: Time Series + Header, Statistical Features, Payload + Header. Payload analysis is still used when an end-to-end approach is employed, in which the classifier itself - a neural network generally - extracts its own representation of raw data. At last, feature selection strongly affects the model selection as well as the required computational complexity. As reported by Rezaei, in case of statistical features, classical ML algorithms and MLP reach the best results without overfitting, while Payload data require more complicated models as CNN or CNN+LSTM. Time series, instead, have medium complexity and all previous algorithms have been used in literature. In the following, some of the most important works are briefly reported whereas Table XII reports a comparison of their main characteristics.

In an early work on applying machine learning techniques to application protocol classification in encrypted traffic, Bar-Yanai et al. in [2] proposed a novel algorithm which combines K-means and K-NN to classify traffic data (HTTP, SMTP, POP, Skype, eDonkey, BitTorrent, Encrypted BitTorrrent, RTP and ICQ). Authors claim their method is accurate and fast enough to be used in a real-time environment. Nonetheless, the flow classifier needs 100 packets, if available, or at least 15 packets.

Naseer et al. investigates the suitability of deep leaning (DL) approach for anomaly detection [25]. They trained different DL architectures, namely Auto Encoders, Convolutional Neural Network (CNN), Long Short-Term Memory on the NLSKDD dataset, which includes four attack typologies: DoS, U2R, R2L, and Probe. Their comparison against classical ML techniques showed that DL techniques improve of a few percentage points classical ML accuracy at the cost of at least three order of magnitude higher training time. This work confirmed that, despite employing GPU testbed, DL techniques are not suitable for an online environment yet.

Wang et al. introduced the end-to-end learning in the context of traffic classification [31]. They employed a 1D CNN on raw data of the SCX VPN-nonVPN dataset to classify 12 classes of traffic types. Data have been pre-processed into two different ways: either the first 784 bytes were taken from the whole packet or just from the 7th layer. Their study showed that considering all layers rather than only the 7th leads to higher classification accuracy. Nevertheless, authors did not report any results in terms of training or evaluating time nor they did a full comparison against traditional techniques.

Chen et al. in [7] used deep neural network to classify IP traffic. They employed a classic 2D CNN after having projected 6 time-series features into a multi-channel image. They introduced of a novel Sequence-to-Image technique (Seq2Img) based on Reproducing Kernel Hilbert Space (RKHS) embeddings and used it to convert 6 features (both static and dynamic size, inter-arrival time and direction of the packets) through the RKHS embedding into a 6-channel image used by the CNN to determine traffic class. They identified either the APP (Skype, Facebook, Instagram, etc.) or the protocol (FTP, HTTP, SSH, etc.) used to generated the data. They claim the proposed framework may be used online, as only 10 packets are needed, but the work lacks training and evaluating time.

The work of Balla et al. aimed at performing real-time detection of web-crawler [1]. As in this paper, authors employed decision tree to understand whether the ongoing session was started by a human or a crawler. In our paper, we have also classified the specific crawler tool producing the traffic. Hence, we can argue that this paper generalize this previous work.

The most similar work in literature has been presented by Vargas et al. [29]. Authors created a Bayesian Network model capable of correctly classify different types of attack such as worms and brute force attacks (DoS, DDoS) while using Time Series. Reported accuracy, True negative rate, True positive rate and False positive rate are impressively high even though no comparison is made with any other methods. Besides, only terminated flows are considered, thus, as reported by authors, their method cannot be employed online.

Miller et al. identified specific web pages within websites and deduced some of its content with an accuracy of 87% [21]. After having clusterized the data set based on ingoing and outgoing packet dimensions information, a Hidden Markov Model classifer concluded the job. Authors reported that attacks based on their method could expose ”Personal details including medical conditions, financial and legal affairs and sexual orientation”. Authors also proposed a possible mitigation, which decreases web page classification accuracy of 27% with a little increase in traffic volume. A similar work but in a completely different context was proposed by Koch and Rodosek [18] who attempted to detect potentially dangerous command sequences within SSH traffic. They proposed a novel IDS that detects attacks based on the probabilities of certain

PAPER	YEAR	TRAFFIC CLASSIFICATION	ATTACK CLASSIFICATION	ML METHOD	FEATURE TYPE
This work	2019	✔ browser identification	✔ DOS, crawling	random forest	statistical features
Bar-Yanai et al. [2]	2010	✔ protocol identification	○	K-means, k-NN	time series
Naseer et al. [25]	2018	○	✔ DOS, U2R, R2L, probe	AEs, CNN, LSTM	statistical features
Wang et al. [31]	2017	✔ ISCX VPN-nonVPN	○	1D CNN	payload inspection
Balla et al. [1]	2011	○	✔ crawling	decision trees	statistical features
Vargas-Munoz et al. [29]	2018	○	✔ DoS, DDoS, worms	Bayesian network	time series
Lopez-Martin et al. [20]	2017	✔ RED IRIS	○	RNN, CNN, RNN	time series
Miller et al. [21]	2014	✔ single web pages	○	logistic regression, HMM	bag of Gaussian
Koch and Rodosek [18]	2010	○	✔ R2L, U2R over SSH	clustering	time series
Chen et al. [7]	2017	✔ protocol or app identification	○	RKHS, CNN	time series

TABLE XII: Overview of machine learning methods for traffic classification.

command sequences. Each command is classified according to the nearest cluster of command features learned by the system. Authors claimed that, besides being only a prototype, the system can be used in a general context as it is based on flow evaluation only. However, the ever-changing SSH scenario, where new commands are introduced yearly, strongly limits the usability of this approach.

IX. Conclusions

In this paper we have presented our approach for the classification of the tools that generate user traffic. Based on machine learning, out approach categorizes the tools by only analyzing the traffic statistics computed on the network and transport ISO/OSI layers. Our set of random forest classifiers is not only able to identify the category of the tools (browser, network stress tool and web crawler), but also the specific tool used to generate such traffic (e.g. Chrome vs. Firefox). The classifiers we have trained have two major advantages when compared to existing works. First, they can be used to categorize live traffic, which is paramount to use them in IDS/IPS scenarios. Then, being able to cope equally well with both clear and encrypted traffic, they do not need to resort to Deep Packet Inspection to correctly work.

In the near future, we plan to integrate our approach into existing IDS tools, such as Zeek and Suricata. In addition, we will investigate the use of deep learning techniques to further increase the accuracy of our methodology, especially when detecting the specific versions of the tool used to generate the analyzed traffic. Furthermore, we will try to extend our approach also to UDP traffic. This will let us to support for example the increasingly widespread QUIC protocol. Also, this will extend the applicability of our DoS identification approach to DRDoS (Distributed Reflective DoS) attacks, since these are mainly based on exploiting UDP-based services (e.g. DNS). In addition, we plan to explore the performance of other machine learning approaches on this task, in particular semi-supervisioned algorithms and deep learning techniques.

References

[1] Andoena Balla, Athena Stassopoulou, and Marios D. Dikaiakos. Real-time web crawler detection. In *2011 18th International Conference on Telecommunications, ICT 2011*, pages 428–432. IEEE, 2011.

[2] Roni Bar-Yanai, Michael Langberg, David Peleg, and Liam Roditty. Realtime classification for encrypted traffic. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 6049 LNCS, pages 373–385. Springer, Berlin, Heidelberg, 2010.

[3] L. Breiman. *Classification and Regression Trees*. CRC Press, 2017.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct 2001.

[5] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications Surveys & Tutorials*, 18(2):1153–1176, 2016.

[6] Enrico Cambiaso, Gianluca Papaleo, and Maurizio Aiello. Taxonomy of slow dos attacks to web applications. In Sabu M. Thampi, Albert Y. Zomaya, Thorsten Strufe, Jose M. Alcaraz Calero, and Tony Thomas, editors, *Recent Trends in Computer Networks and Distributed Systems Security*, pages 195–204, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[7] Z. Chen, K. He, J. Li, and Y. Geng. Seq2img: A sequence-to-image based approach towards ip traffic classification using convolutional neural networks. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 1271–1276, Dec 2017.

[8] Junghoo Cho and Hector Garcia-Molina. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.*, 28(4):390–426, December 2003.

[9] A. Compagno, M. Conti, P. Gasti, and G. Tsudik. Poseidon: Mitigating interest flooding ddos attacks in named data networking. In *38th Annual IEEE Conference on Local Computer Networks*, pages 630–638, Oct 2013.

[10] Sumeet Dua and Xian Du. *Data mining and machine learning in cybersecurity*. Auerbach Publications, 2016.

[11] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring https adoption on the web. In *26th USENIX Security Symposium*, pages 1323–1338, 2017.

[12] R. Fielding and J. Reschke. Hypertext transfer protocol (http/1.1): Semantics and content. RFC 7231, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7231.txt.

[13] Jérôme François, Issam Aib, and Raouf Boutaba. Firecol: A collaborative protection network for the detection of flooding ddos attacks. *IEEE/ACM Trans. Netw.*, 20(6):1828–1841, December 2012.

[14] Mathieu Gorge. Lawful interception–key concepts, actors, trends and best practice considerations. *Computer Fraud & Security*, 2007(9):10–14, 2007.

[15] Jan Gorodkin. Comparing two k-category assignments by a k-category correlation coefficient. *Computational biology and chemistry*, 28(5-6):367–374, 2004.

[16] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, volume 1, pages 278–282 vol.1, Aug 1995.

[17] Ryan Kok Leong Ko and Kim-Kwang Raymond Choo, editors. *The Cloud Security Ecosystem - Technical, Legal, Business and Management Issues*. Elsevier, 2015.

[18] Robert Koch and Gabi Dreo Rodosek. Command evaluation in encrypted remote sessions. In *Proceedings - 2010 4th International Conference on Network and System Security, NSS 2010*, 2010.

[19] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[20] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network Traffic Classifier with Convolutional and Recurrent Neural Networks for Internet of Things. *IEEE Access*, 5:18042–18050, 2017.

[21] Brad Miller, Ling Huang, A D Joseph, and J D Tygar. I know why you went to the clinic: Risks and realization of HTTPS traffic analysis. In *Lecture Notes in Computer Science (including subseries Lecture Notes*

in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 8555 LNCS, pages 143–163, 2014.

[22] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[23] Jonas Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

[24] S. M. Mousavi and M. St-Hilaire. Early detection of ddos attacks against sdn controllers. In *2015 International Conference on Computing, Networking and Communications (ICNC)*, pages 77–81, Feb 2015.

[25] Sheraz Naseer, Yasir Saleem, Shehzad Khalid, Muhammad Khawar Bashir, Jihun Han, Muhammad Munwar Iqbal, and Kijun Han. Enhanced network anomaly detection based on deep neural networks. *IEEE Access*, 6:48231–48246, 2018.

[26] S. Oshima, T. Nakashima, and T. Sueyoshi. Early dos/ddos detection method using short-term statistics. In *2010 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 168–173, Feb 2010.

[27] Shahbaz Rezaei and Xin Liu. Deep Learning for Encrypted Traffic Classification: An Overview, 2019.

[28] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009.

[29] M. J. Vargas-Munoz, R. Martinez-Pelaez, P. Velarde-Alvarado, E. Moreno-Garcia, D. L. Torres-Roman, and J. J. Ceballos-Mejia. Classification of network anomalies in flow level network traffic using Bayesian networks. In *2018 28th International Conference on Electronics, Communications and Computers, CONIELECOMP 2018*, volume 2018-Janua, pages 238–243, 2018.

[30] Louis Waked, Mohammad Mannan, and Amr Youssef. To intercept or not to intercept: Analyzing tls interception in network appliances. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, pages 399–412, New York, NY, USA, 2018. ACM.

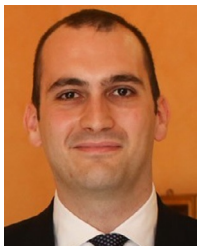
[31] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-To-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics: Security and Big Data, ISI 2017*, number July 2017, pages 43–48, 2017.

[32] Ibrahim Waziri. Website forgery: Understanding phishing attacks and nontechnical countermeasures. In *Proceedings of the 2015 IEEE 2Nd International Conference on Cyber Security and Cloud Computing (CSCloud)*, CSCLOUD '15, pages 445–450, Washington, DC, USA, 2015. IEEE Computer Society.

[33] Q. Yan, F. R. Yu, Q. Gong, and J. Li. Software-defined networking (sdn) and distributed denial of service (ddos) attacks in cloud computing environments: A survey, some research issues, and challenges. *IEEE Communications Surveys Tutorials*, 18(1):602–622, Firstquarter 2016.

[34] Saman Taghavi Zargar, James Joshi, and David Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

Daniele Canavese Daniele Canavese received a MSc degree in 2010 and a PhD 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 Leonardo Regano received a MSc degree in 2015 and a PhD in Computer Engineering in 2019 from Politecnico di Torino, where he is currently a research assistant. His current research interests focus on software security, applications of artificial intelligence and machine learning to cybersecurity, analysis of security policies, and assessment of software protection techniques.

Cataldo Basile Cataldo Basile received a MSc (summa cum laude) in 2001 and a PhD in Computer Engineering in 2005 from the Politecnico di Torino, where is currently 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 Gabriele Ciravegna is a PhD student in Smart Computing at the Università degli Studi di Firenze. In 2018 he received a MSc (summa cum laude) in Computer Engineering from the Politecnico di Torino. He is interested in machine learning and in its application in critical contexts as medical diagnosis. He is focusing on overcoming intrinsic limits of machine learning and neural networks, above all their understandability.

Antonio Lioy Antonio Lioy is full professor at Politecnico di Torino, where he leads the TORSEC research group active in information system security. His research interests include network security, public-key infrastructure (PKI), and policy-based system protection. Lioy received a MSc in Electronic Engineering (summa cum laude) and a PhD in Computer Engineering, both from Politecnico di Torino.