Towards applications' fingerprinting through the usage of NetFlow/IPFIX technology

Mario R. Vuolo
Faculty of Electrical Engineering,
Mathematics and Computer Science
University of Twente
Email: m.r.vuolo@student.utwente.nl

***Abstract*—Flow monitoring has become an increasingly prevalent method for monitoring traffic in enterprises mainly due to its performance and scalability. We present a system that detects anomalous outbound HTTP communications, which exploits the advantages of NetFlow/IPFIX technology to passively extract fingerprints for each application running on a host. The aim of our work is to identify the most discriminative features within an IPFIX system to identify both the application types and detect fingerprints from anomalous communications. We evaluate our prototype with real-world data from an international organisation and a dataset of traffic generated from malware and show that it can detect malicious traffic with an accuracy of 98.6% and a recall of 91.6% for 246 monitored host machines. We compare our solution with DECANTeR [6], the current state-of-the-art application fingerprint approach, which detects anomalous outbound HTTP traffic independently from their payload without using malicious data during the training phase. The results show how our approach is a good alternative, in terms of detection rate and resources required in detecting malicious traffic. This capability is further demonstrated in an analysis of the dataset composed of malicious traffic, where our system detected malicious traffic in 99,06% of the cases.**

***Keywords*—Network Security, Anomaly Detection, Data Exfiltration, IPFIX, Passive Application Fingerprinting.**

I. Introduction

Data is one of the biggest and most valuable assets for organisations. For this reason, in the last years *sensitive data* has been a key motivator for most of cyber-attacks performed against enterprises. In 2015, the Italian security company *Hacking Team* was itself hacked, causing the disclosure of around 400GB of internal files including zero-day exploits, a list of its clients and private emails. In 2021, *Facebook* warned its customers that in 2019 a cyber-attack had compromised around 533 million accounts causing the loss of millions of phone numbers, full names, locations and biographical information [1]. These are two examples of companies among many others that have been breached in the last years. Additionally, data exfiltration has been shown to be a serious, expensive and increasing problem for companies by the latest Verizon Data Breach Investigations Report [2] and the IBM Security Report [3].

In response to these issues, automatic or semi-automatic methods have been proposed to mitigate the risks to incur data exfiltration. Different techniques exist to detect malicious communication by exploiting malware characteristics. In particular, the so-called *signature-based*, *behavioural-based* and *anomaly-based techniques*. In signature-based approaches, tools rely on datasets of known malware to identify unique patterns and characteristics of specific threats, called signatures. These solutions detect malware by identifying unique patterns that match the signatures of known malware. This type of technique works because many malware maintain portions of code recycled from old attacks. Differently, behavioural-based tools aim to understand the characteristics and patterns of what is considered "normal" in a given environment and then identify deviations or changes from this baseline. On the other hand, anomaly-based tools are built on the premise that malicious activities differ significantly from normal behaviours, and by identifying those deviations, potential threats can be detected. In particular, malicious traffic is analysed and clustered into families based on a notion of structural similarity between the malicious HTTP traffic they generate. In this way, it is possible to train classifiers to spot generic command and control (C&C) communication channels [12].

These techniques are the first steps many companies go through to prevent most threats. However, malware commonly employ evasion methods that exploit the weaknesses of these techniques to remain undetected. Firstly, these techniques are strongly influenced by the characteristics of the malware used for the generation of the signatures and classifiers. Thus, unknown or new malware are not detected by these signatures and behavior based tools. Secondly, variants of known malware can exploit obfuscation techniques to hide their behaviour and evade detection techniques. For these reasons, researchers have proposed anomaly-based approaches to overcome these limitations. These solutions are capable without any knowledge of malware to define a baseline of benign network data, which detect potentially a wide range of novel attacks. Unfortunately, most existing approaches do not provide a good detection rate in high speed network enterprise scenarios as they are easy to evade and require a lot of computational and storage costs.

Since many companies rely on NetFlow/IPFIX technology to monitor their network due to its performance, scalability, cost and less privacy concerns, we wanted to investigate whether we could use the advantages of this technology to perform *passive application fingerprinting* to identify anomalous outgoing HTTP connections. Passive application fingerprinting is a method that observes the network traffic generated by the applications to gather information about their behaviour and characteristics. By analysing this network traffic, the technique can identify the applications installed on each machine and detect any anomalies or the presence of new applications, including potential malware infections. Most monitoring occurs within the organization, granting us the advantage of reading unencrypted traffic like HTTP through MITM (Man-in-the-Middle) proxies. Further elaboration on our threat model is presented in Section III.

The main contributions of this paper can be summarised as follows:

- we designed and fully implemented a system capable of detecting anomalous HTTP flows, based on a passive application fingerprinting technique [6], using NetFlow/IPFIX technology. The approach can automatically model different HTTP-based applications of a host from their network traffic and detect anomalous traffic connections. While there is literature about OS fingerprinting within a scheme of IPFIX monitoring [7], to the best of our knowledge, none has been proposed towards the implementation of application fingerprinting.
- we evaluated the obtained solution by comparing the results obtained with DECANTeR. Our results involved two different datasets containing traffic generated by data exfiltration malware and samples of live network traffic.

The remainder of the paper is structured as follows. Section II describes the technologies and concepts adopted in our work and motivates our choices. Section III elaborate the threat model considered. Section IV gives an overview of the proposed system. Section V motivates the choice for the features that were extracted to generate application fingerprint and details each module of our solution. Section VI describes the datasets we used to evaluate the proposed solution and presents the evaluation performed on these datasets. Section VII goes through benefits and limitations of our solution. Section VIII compares our work with related literature. Finally, section IX presents the conclusions and possible directions for future research.

II. Background and Motivations

As mentioned in the previous section, our work focus on NetFlow/IPFIX technology to perform application fingerprinting to define users' 'normal' traffic behaviour to detect data exfiltration through HTTP protocol. This section explains the motivations of our choices.

A. NetFlow/IPFIX

NetFlow and IP Flow Information eXport (IPFIX) are scalable passive network monitoring approaches suitable for high-speed networks, where packets are aggregated into flows and exported for storage and analysis. A *flow* is defined as "a set of IP packets passing an observation point in the network during a certain time interval, such that all packets belonging to a particular flow have a set of common properties" [27]. In addition to their suitability for high-speed networks, these approaches have other advantages: 1) they are widely deployed, as integrated in high-end packet forwarding devices; 2) they can be efficiently stored and compressed, allowing flow collections for long periods of time (e.g., years); 3) they are usually less privacy-sensitive, since traditionally only packet headers are considered; and 4) they traditionally do not rely on Deep Packet Inspection (DPI), so these solutions can be applied to encrypted communications and standard protocols, such as HTTPS [26]. In our research we setup *flow exporters* using a tool called nProbe [28] that receives packets leaving and entering the international organisation network, aggregate them into flows and export the flow data to a *flow collector* for storage. In our case, IPFIX was used as a flow export protocol as it is nowadays considered much more accessible and open than its predecessor (i.e., NetFlow protocols). Each stored flow record contains different types of information extracted from the IP header, such as source and destination IP addresses, TCP and HTTP protocols. Examples of the type of information that we analysed in this research will be discussed in Section V.

B. Passive application fingerprinting

Just like a fingerprint's unique pattern serves to identify an individual, each application has unique characteristics in its communications that can be used to identify it on a network. Passive application fingerprinting only listens to the packets on the network. When a packet is received, it extracts specific values of IP, TCP or application headers in order to describe an application. Thus, by creating a baseline reference set of benign application fingerprints, passive fingerprinting can also be used to detect unknown applications and anomalous behaviours at regular intervals [6]. For instance, when a new software is installed, such as a malware, its traffic may be flagged as anomalous. In our system we aim to generate a set of fingerprints for each application from live traffic. This, however, has two main challenges: 1) flow records regarding a specific application may be insufficient to properly model the characteristics of the application due to their low frequency or changes of messages transmitted over time; 2) new applications can be installed or old ones can be updated by users, so new fingerprints need to be added to or modified from the set of fingerprints of the host. We address both these challenges in this paper in section V. In conclusion, even though there is literature in OS fingerprinting within a scheme of IPFIX monitoring [7], none has been proposed towards the implementation of a passive application fingerprinting.

C. HTTP

HTTP is the most popular protocol via which malware exfiltrate data and communicate with command and control (C&C) servers [20][9]. This protocol is almost always allowed, even by very restrictive firewalls or proxy, as it is mostly used by every user application. This guarantees to an attacker a persistent communication channel, where data can be exfiltrated from the organisation's network. Moreover, some characteristics of the HTTP(S) protocol (e.g., higher bandwidth usage, pattern irregularity and extensive use in enterprises) make it a much effective protocol to exfiltrate large volumes of data in a shorter amount of time without being noticed. For these reasons our research focus on HTTP protocol, addressing the challenges mentioned above.

III. Threat Model

We consider a network monitor that extract all network information from servers and workstations in an enterprise network. The malware under consideration utilises HTTP or HTTPS as its communication protocols. These are protocols commonly used by malicious software either to communicate with their C&C server or to exfiltrate data [21][8]. One of the primary reasons for this preference is that HTTP and

HTTPS traffic is generally allowed through enterprise firewalls, and malware can easily conceal data exfiltration among large volumes of benign HTTP and HTTPs traffic. Our solution focuses on HTTP but can be extended to HTTPS in enterprise scenarios where Man-In-The-Middle (MITM) proxies are deployed to inspect encrypted traffic. This is common practice for enterprise networks that inspect employees' encrypted traffic to gain network visibility and detect security threats [24][25]. In fact, many commercial solutions and open-source solutions are widely available depending on the use-case and protocol that should be analysed [23][22]. These solutions intercept, decrypt, inspect, then re-encrypt and forward on traffic between applications or browsers and external web servers.

We assume that access to the network monitors analysing HTTP(S) traffic is restricted and the probes cannot be compromised by an attacker. The attacker can just infect the monitored hosts with any number of malware. We assume attackers can obfuscate (i.e., encode, compress or encrypt) the content of the communication before transmitting it over the network. This is a common hypothesis, because attackers are used to exploiting different evasion techniques [30][32] to avoid detection mechanisms. Finally, our approach assumes there is a security operator that analyses the alerts produced by our solution, acting on those alerts.

IV. System Overview

We recall that our goal is to perform passive application fingerprinting on IPFIX/HTTP data to detect malicious applications. To do so, we present an overview of our approach for detecting data exfiltration by passively modelling benign traffic to identify anomalous behaviours in application network activities. The final output is a list of connections classified as anomalous because these connections were initiated by new installed applications, inconsistent application network communications or malicious traffic. We motivate the design choices that lead to our proposed framework.

The main goal of our approach is to overcome challenges posed by existing malware detection solutions through a new approach with the following characteristics:

- we work with *flow records* to efficiently deal with high volumes of network traffic. This design choice achieves better results both in terms of computational performance and storage cost. This is in contrast with solutions which require the analyses of huge amount of information that is often derived from data payload or host-based logs [18][19].
- We focus only on individual hosts and the behaviour of the applications running on those hosts. This approach enables us to detect suspicious activities conducted by applications installed on monitored hosts, considering their application type (i.e., browser or background applications), and past behaviour. By distinguishing between browser and background applications, we can treat flows identified as either type of application differently based on the expected behavioral patterns of their communication and the level of user interaction.
- we propose a set of features that is tailored to identify anomalous communications with respect to the different application type. For this purpose, we specifically extract features that are discriminatory for background applications from those commonly seen in browsers.

Figure 1 represents the five main phases of our framework: 1) flow export and collection; 2) bidirectional correlation; 3) feature extraction; 4) training; and 5) detection.
The *first phase* involves both the extraction and collection of flows of HTTP connections (e.g., GET and POST requests). Thanks to the computational and storage advantages given by NetFlow/IPFIX technology mentioned in section II-A, we analyse applications traffic in both directions. Thus, we analyse not only *outgoing traffic*, as would be reasonably to think when working with data exfiltration (e.g., [15][6][17]), but also *incoming traffic*. In this way, we can detect suspicious unidirectional flows that carry traffic in just one direction (i.e., client to server or server to client). For example, an application that periodically poll a remote server without answer.

The *second phase* involves the correlation between outgoing and incoming traffic. In order to have in one flow both information regarding the whole connection (i.e, obtain bidirectional flows) we had to separately match the unidirectional flows exported by nProbe corresponding to the same connection. We adopted this solution to overcome the fact that the Network Interface Card (NIC) in our flow probe was not able to reliably state which packets came in and which ones went out the network. Therefore, we correlated the outgoing flows initiated in the enterprise to the incoming flows merging the ones that belonged to the same connection (if any). The information used to do this correlation was the 5-tuple: protocol, source IP, source port, destination IP and destination port.

The *third phase* extracts features that are relevant for defining both the application type originating the requests and detect possible data exfiltration. These features are computed for each internal host every time flows are exported. Details, examples and motivations on the chosen features will be discussed in Section V.

The *fourth phase* is the training phase, where the system analyses the flows of the hosts in a non-infected state or through the supervision of a security operator. In this way, we can generate an initial set of benign applications' fingerprints running on the host machine. It can be seen as a setup phase, which runs for a specific amount of time, required before starting with the testing of flows. In this phase we have two sub-phases, called *browser identification* and *fingerprint generator*. In the browser identification case, our system classifies the observed flows as either browsers or background applications. Identifying a flow as generated by a background application signifies its association with tasks such as updates, data synchronisation or system maintenance, whereas flows originating from browsers are tipically more dynamic due to user interaction. For this reason, depending on the application type the flow belongs to, the next module generates for each flow the set fingerprints using a specific set of features. The output of the training phase is a set of fingerprints for each monitored host. There are two ways to obtain a non-infected state: analyse a new or formatted host or using a Threat Intelligence to help define which flows are legitimate. Details on how we implemented both browser identification and fingerprint generator will be discussed in Section V as we explain the key features used in these modules.

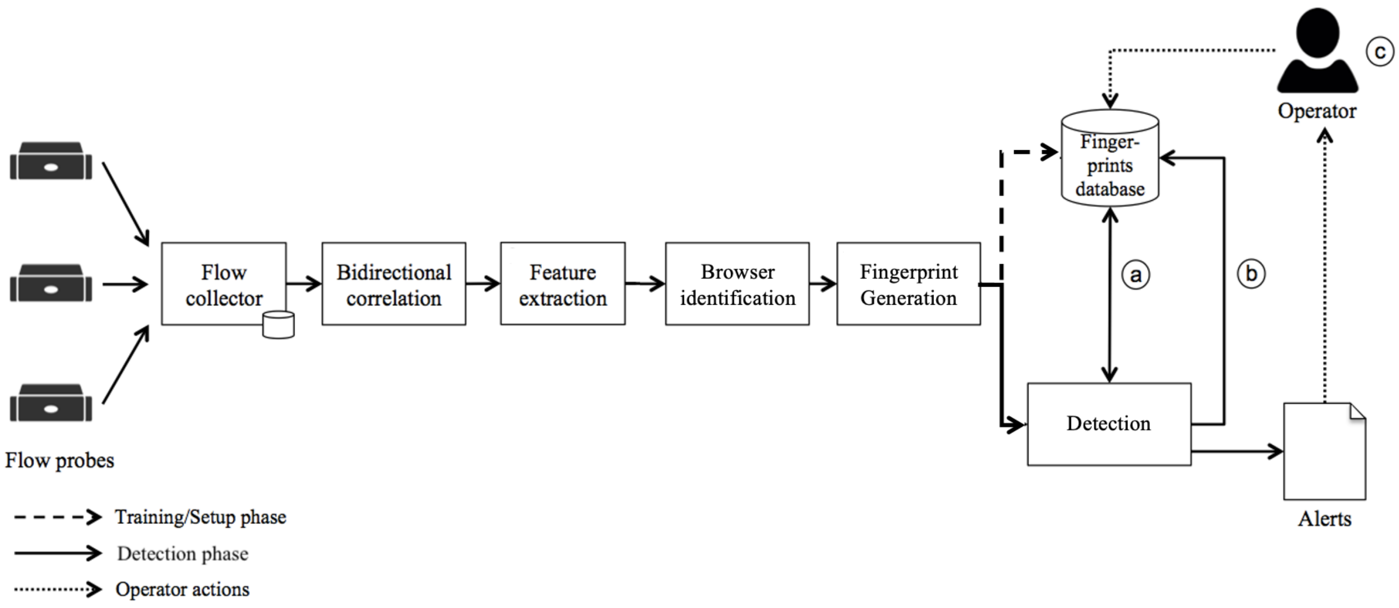

Fig. 1: Framework overview. a) check of the application fingerprints generated in the training phase with the current analysed flow; b) new fingerprints can be added to or updated from the set of fingerprints in the fingerprint database; c) a security operator analyses the alerts produced by the system acting on them.

The *fifth phase* is the last phase of our framework: the detection phase. The detection of the traffic of a host begins as soon as the training phase ends. This phase has three modules: browser identification, fingerprint generator and detection. While the first two follow the behaviour already described for the training phase, the *detection* module checks if there are any anomalous connections. This phase keeps on testing each flow that is passively extracted from the network. In this way it detects whether generated fingerprints result anomalous with respect to already seen applications. In case traffic is generated from new applications, heuristics are in place to automatically confirm new applications as legitimate or raise an alert. The outcome of this phase is a list of alerts that can be immediately seen by a security operator, which can act on the results flagging them as true or false alerts and updating the fingerprint database. Details on the heuristics, the detection module and the operator role in our system will be discussed in Section V.

V. System details

In this section, we present and motivate a set of discriminative features that we extract for each flow, as we discuss the details of each module of our solution.

A. Browser identification Module

The goal of the browser identification module is to distinguish between flows generated from *browsers* and *background applications* in both training and testing phases. Knowing the application type of a flow helped us determine the legitimacy of the request in terms of the behaviour that we would have expected to see from it. In particular, background applications usually follow common patterns, while browsers' traffic is more dynamic as it directly depends on user actions. Beyond that, dividing the problem in two different ones helped us reducing the complexity we face as we were defining fingerprints features to two different kinds of behaviours. In this module we were mainly interested in classifying flow as browsers or background applications according to the extracted flow values. In this module there is no attempt of understanding the legitimacy of behaviour of the application generating the flows. Their behaviour is analysed in the detection module discussed in Section V-C. In that module, any new or anomalous flows are alerted as potentially malicious, where "anomalous" being defined in relation to the identified application type.

As we collected the data using the setup described in IV, the browser identification module is used in both training and detection phase. The only difference between training and detection phase, regarding this module, is the type of data processed. In fact, in the training phase we handle only non-infected data, while in the detection phase we may see both legitimate and non-legitimate application flows.

We used one of the HTTP header fields extracted by nProbe, called User-Agent, to differentiate applications. We used this header as a way to aggregate flows deriving possibly from the same application. The User-Agent is a request-header field that contains information about the software agent, which acts on behalf of the user and originates the request. It is often used by software as a sort of identifier, thereby making it unique per application. In training phase this assumption holds because the training is assumed to be trusted, therefore no malicious traffic is expected to be aggregated. In detection phase this decision has some consequences regarding evasion techniques as seen in the works of [33] [34], which we discuss Section VII. It has been proven that this string is one of the most identifying metrics (together with information on plugins and fonts installed) to generate unique fingerprints as these are commonly user-specific [29].

TABLE I: Browser identification features.

#	Description	Type
1	Length of the User-Agent header field	Numeric
2	Entropy of the User-Agent header field	Numeric
3	Flag whether the HTTP referer header exists	Boolean
4	Length of the request body	Numeric
5	Longest packet (bytes) of the flow	Numeric
6	Length of the path in the URI	Numeric
7	Entropy of the path in the URI	Numeric
8	Number of alphanumeric values in the path and query of the URI	Numeric
9	Depth of the hierarchical path in the URI	Numeric

We implemented a binary classifier using random forests as it is a strong modelling technique and much more robust than a single decision tree. They aggregate many decision trees to limit overfitting as well as error due to bias and therefore yield useful results. In respect to other models considered lazy, such as the K-Nearest Neighbour, random forests work well with large datasets and is faster in testing. This was a requirement considering the scenario we described in Section III. Further analysis and comparisons between classification models are shown in Appendix A.

The set of features for our classifier are depicted in Table I. These features were derived from a user dataset used for validation. We employed feature importance techniques to assess the relevance of each feature in predicting the target variable, which in this case is the application type (i.e., browser or background application) in network flows. By analysing the importance scores of each feature, we identified the most informative and discriminative features for inclusion in our classifier. We note that the features related to incoming traffic were excluded from the set as not discriminant in determining the type of application generating the flow. Features 1 and 2 were chosen as the User-Agent field is often unique for each application, except for minor changes in version numbers due to software updates. We would expect browsers to have longer User-Agent strings linked to a higher entropy value than the ones in background applications. Feature 3 distinguishes a browser from a background application as it indicates the address of the webpage that linked to the resource being requested. Features 4 and 5 determine the communication pattern of the request as background applications are usually small with a steady ratio between packets and bytes transmitted. The last four features show the complexity of the location of a certain resource (i.e., URI) on the Internet. In this case, resources required by background applications are not usually identified by human-readable URIs, as usually are the URIs of websites.

B. Fingerprint Generation Module

The goal of the fingerprint generation module is to create a fingerprint of each flow coming from the browser identification module by extracting a set of features that changes according to the type of application identified in the previous module. The features we extracted on this module came from the analysis of the available features extracted in the previous modules, using the knowledge from data exfiltration malware and legitimate user traffic. In particular we applied feature importance techniques to determine the relevance of certain features for certain type of network flows generated by either browsers and background applications (an example of an output it is show in Appendix A). In the case of background applications, we extracted the following features:

- *Outgoing data:* the information representing the length of the body of each packet. In this way, we want to define how much data was transmitted in the payload. We have observed that many background applications transmit a similar amount of data.
- *MIME type:* the format of the resource requested by a host application. We have seen that most background applications request periodically resources with constant MIME type (e.g., 'application/vnd.ms-cab-compressed', 'application/rss+xml', etc.).
- *Request method:* the string of the HTTP request method field. Depending on the application function, an application may retrieve information from a host and upload information to another server.
- *Status code:* the return codes issued by a server in response to the client's request made to the server. As for the other extracted features, this one help distinguish between legitimate and anomalous connections.
- *URL path structure:* the location to one specific network resource requested by a host machine. Unlike browsers human-readable URIs, background applications often request resources located in fixed or encoded directories. In this case the information collected is the number of alphanumeric values and the entropy of the path of the URL.

We considered other type of features, such as the longest packet of the flow or the User-Agent, but we decided to use those features that could capture a clear snapshot of the normal application connection. In this case, 'normal' is defined as the behaviour that is consistent with the most common behaviour of the application. Moreover, we avoid using features that could possibly be spoofed by an attacker and favoured features that require a correct configuration of a server. In this way we made harder the job of the attacker as he has to spoof the User-Agent to pass the application type classification, configure a server that return the correct set of headers values and generate a specific request packet with a fixed payload. These assumptions slow down possible data exfiltration.

Fingerprints generated by this module are stored in a database for future comparison and reference. This database serves as a repository of known application fingerprint, stored during the training phase. The fingerprint can be updated automatically by the framework or populated by the operator (details on this are mention in Section V-B). This is a solution used also by DECANTeR as it facilitates comparison with newly generated fingerprints during the detection phase.

The way we have defined our objectives, we wanted to obtain a system able to identify flows of a specific class amongst all the others. For this reason, we focused on *one-class classification*. This term was coined by Moya and Hush [35], and many systems have been published about the advantages in the field of anomaly detection, outliers detection and novelty detection [36]. In our case we used a probabilistic outlier detection model, the python library LSAnomaly[1], based on the work of Quinn and Sugiyama [37] on a novel, probabilistic and nonparametric least-squares approach. The main advantage we obtained from using this approach was the capacity to recognise unknown applications requests. In addition, this kind of classifier can easily be updated, works very well with a limited set of training data, and is much faster to train and test on large datasets than many alternative methods, such as one-class support vector machine (SVM).

In our research we could not distinguish different browsers applications by using only the features we extracted in a single HTTP connection. The reasons are mainly because browsers contents are unpredictable and dynamic as it directly depends on both web sites and specific user actions. Beyond that the features extracted by default by nProbe resulted not enough to properly define different browsers. Details on how we cope with these issues will be explained in Section V-C, while we will discuss the difficulties we found in Section VII.

C. Detection Module

The goal of the detection module is to determine whether new generated fingerprints are similar to any of the application fingerprints generated during the training phase. The comparison is done by executing specific similarity functions using most of the features used on both *Browser identification* and *Fingerprinting Generation* modules. In case a tested fingerprint doesn't match any of the fingerprints in the database, the module treats the flow as a new application running on the host.

In the detection module we used different heuristics for two main reasons:

- *Updating the database fingerprints.* An anomaly detection system should provide an updating mechanism in case new fingerprints are generated related to new applications installed on the host. Depending on how similar the features were compared to trained fingerprints and the expected behaviour for the type of application (i.e., background application or browser) we would define the fingerprint as an update or as a complete different application. We employed the Euclidean distance to quantify the absolute difference between two numeric features, whereas we utilized the longest contiguous matching subsequence for string features.
- *Add new application fingerprints.* Differently than an update, the detection module would still trigger an alert, but it would still try to classify as a legitimate or malicious applications depending on some heuristics specific to the type of application (i.e. background application or browser). We assessed whether new applications align with known and trusted behaviour focusing on observed patterns and thresholds within HTTP attributes and response codes. For browser applications, a new application will be considered legitimate if its content, visited sites, and successful server responses exhibit similarity to established patterns. Similarly, a new background application will be deemed legitimate if its behaviour closely resembles that of recognized ones, evidenced by fewer failed requests and URL request patterns consistent with those of known applications.

These are some of the behaviours inspected while adding a new application fingerprint during the detection module:

- The *URI request structure*. While background applications are programmed to use a fixed structure in their request (i.e., same type of headers and similar content and size), browsers are directly influenced by users. In fact, browsers connections are characterised by patterns that are almost unique across the entire web, given the wide range of resources that can be accessed. This makes harder to find a unique model, which can differentiate two browsers (e.g., Chrome and Firefox);
- The number of possible *domains* applications can communicate with. This number is huge for browsers in respect of the number of domains a background application makes HTTP requests to. Thus, every day we would expect to see a browser to contact different servers, while background applications usually keep on reaching to the same set of domains;
- The *MIME types* of the transmitted resource. Depending on the application type, we would expect various format resources in servers' responses. For example, it is common to observe charset encoding and many different MIME types on flows initiated by a browser. On the other hand, this is not a common behaviour in background applications' communications.

VI. Experimental Results

In this section we describe the three different datasets used during our research. We evaluate the detection performance of our solution on the datasets used.

A. Datasets

In our evaluation we made use of Data Exfiltration Malware (DEM) dataset analysed in DECANTeR as to analyse the efectiveness of the solution to label malicious traffic flows. In addition to these datasets, we deployed our solution on an international organisation (OD), which results will be discussed in this section.

Data Exfiltration Malware[2] (DEM). This dataset was used to evaluate DECANTeR. It contains network captures (.pcap) mainly of info-stealer malware run on a virtual machine (VM) for roughly 60 minutes per sample. The samples belong to 8 families of information-stealer malware: iSpy, Shakti, FareIT, CosmicDuke, Ursnif, Pony, Dridex, and SpyEye. We ran our solution on this dataset to show effectiveness of the solution in detecting anomalous traffic. To evaluate our solution we used the same labels for benign and malicious traffic used by DECANTeR. In particular, all requests that had same user-agent as the applications in the VMs traffic (without the execution of the malware). The other requests are considered malicious, including browser requests as malware are the only ones that could have used browsers user-agents.

Organisation Dataset (OD). Our system has been deployed to an international organisation monitoring both incoming and

[1]https://github.com/lsanomaly

[2]https://scs.ewi.utwente.nl/downloads/

outgoing HTTP traffic on a network link with thousands of hosts. nProbe [28] has been used for exporting the flows without sampling, while IPFIXcol [38] was used for storing the flows. Thus, these flows were processed by our system. We used real data to avoid possible biases by running the system with data captured in a lab (e.g., a fixed set of installed applications or operating systems). Over a period of two weeks, we obtained 979.463 HTTP requests generated by network traffic in 252 hosts. In total we saw 3126 User-Agent deriving from the applications installed in these hosts.

B. Experimental Setup

As we wanted to evaluate both our solution and DECANTeR we followed the same process to label malicious and benign requests in our datasets. We used the first day of traffic for the training phase and the rest of the captured traffic for the testing. As the training mode is a trusted scenario, where only legitimate flows are seen, we manually checked if there were malicious fingerprints, and if so, we removed them. For the OD dataset we used the help of external threat intelligence services[3] and with indicators of compromise we have obtained from a professional threat intelligence provider.

In our solution the operator is required to help the solution to learn by checking the legitimacy of alerted flows are related to new background applications or browser on the monitored hosts. In the case the fingerprint is considered legitimate this will be added to the fingerprints database. This is similar to how DECANTeR handle unseen traffic and it is also a normal case seen in most of the solutions experienced from professional threat intelligence providers. In our experimental setup, we assume that the operator is actively updating the trained fingerprints database.

C. Results

Solution	Dataset	Accuracy	Precision	Recall	F1-score
IPFIX Solution	DEM	99.06%	99.77%	99.27%	99.40%
IPFIX Solution	OD	98.60%	11.78%	91.60%	21.09%

TABLE II: Comparison of both solutions on background application contained in Organisation Dataset.

As shown in Table II we ran our framework on the DEM and OD datasets, containing both background and application flows.

We consider a message to be: true positive (TP) if the flow was assessed as anomalous and we had indicators of compromise; false positive (FP), if the flow was assessed as anomalous and we had no indicators of compromise; false negative (FN) if the flow was assessed as legitimate and we had indicators of compromise; and true negative (TN) if the flow was assessed as legitimate and we had no indicators of compromise.

We used perfomance metrics to compare each evaluation:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

[3]We have used the information provided by ThreatCrowd (https://www.threatcrowd. org/) and VirusTotal (https://virustotal.com/)

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$F1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

These performance metrics related to DEM provide insights into the effectiveness of the model in detecting anomalous flows, with high precision and recall, resulting in a high F1-score. This is probably due to the fact that new applications are not seen as this is a closed dataset. So we have mainly new flows coming from malware, flows that are not added to the trained dataset as they don't comply with the heuristics mention in Section V-C (i.e., URI structure, domains and MIME types).

Regarding the OD dataset, the framework analysed a total of 262,916 flows. Among these, 1064 flows triggered alerts as new applications, leading to the addition of new fingerprints to the database. False negatives primarily occurred with applications utilizing user-agents such as "Transmission", "Clementine" and "curl". Notably, the majority of false positives (72% corresponding to 2201 flows) from background applications were attributed to just two applications on two separate monitored hosts, identified by the user-agents "MEGAsync" and "p2/1.1.300.v20161004-0244". Further investigation is required to determine why these behaviors were not recognized as those of new legitimate applications by the detection module.

We counted the number of times that an operator would have intervened during the analysis. The number was 1,924 times. Of these, 502 were related to new fingerprints added to the fingerprinting database, while around 1,000 were related to two single users behaving in an uncommon way. In particular, user-agent from HTTP request headers, of what it seems to be browsers applications, appeared to be malformed. These strings contained a mixture of characters, symbols, and sequences that deviated from the usual pattern for a User-Agent string.This anomaly, which could be attributed to bot activity, script execution, or data corruption, was observed on two monitored hosts.

We compare our solution to DECANTeR only on background applications as we didn't have all features to permit DECANTeR to properly identify browsers traffics (i.e., Accept-Language field):

Solution	Accuracy	Precision	Recall	F1-score
DECANTeR	93.53%	0.14%	11.67%	0.25%
IPFIX Solution	98.01%	2.75%	73.67%	5.35%

TABLE III: Comparison of both solutions on background application contained in Organisation Dataset.

The IPFIX solution shows better performance than DECANTeR in terms of accuracy, recall, and F1-score.

However, both solutions exhibit relatively low precision, suggesting that there is room for improvement in accurately identifying true threats while minimizing false alarms. A way to reduce false positive alerts would be to increase the training phase period, so all applications running on the monitored hosts would be already present on trained fingerprint database. In some cases during the testing phase, a monitored host was observed with only one flow available for testing. Meanwhile, the maximum number of flows collected from a single monitored host was 13,615, with a median of 432 flows per host.

To conclude, this evaluation shows that the solution proposed for fingerprinting application traffic proposed is a good alternative, in terms of detection rate with an accuracy of 98%. One possible factor contributing to this discrepancy could be the variation in the collected HTTP headers. DECANTeR appears to gather a comprehensive set of headers including Accept, Accept-Encoding, Accept-Language, Connection, Cookie, DNT, X-Requested-With, among others, whereas on OD these headers are not considered potentially influencing the analysis. In this case the IPFIX solution requires less information related to the HTTP headers to detect anomalous traffic on background applications.

VII. Discussion

We had issues in the way nProbe extracted flow records regarding the HTTP headers. In particular, we sometimes had some malformed HTTP header strings from browser connections working with the third dataset (i.e., OD). For example, for some hosts we saw one or two connections with malformed User-Agent strings, such as "Moz", or "Mozilla/5.0 (Wind". Looking deeper to the flow record we found out that those connections were from the host's browser. This issue was half fixed by using edit distance algorithms and finding out if there was some correspondence with an already seen application. However, depending on how badly the string was cut it could have simply slip, and as they are not frequent it would have been flagged as anomalous (i.e., false positives). Further understanding of the mechanism through which nProbe HTTP plugin exports HTTP headers needs to be done, in order to fix this issue from the beginning of our framework avoiding unnecessary processing cost and alerts output from the system.

We have some parallelism with the features used by Decanter. Specifically, how the traffic was labelled and the features used. Being able to see the difference between requests from the same application, helps to give a better look on what is actual being sent out, while in our case we don't have this level of details. Another distinction lies in our approach, which extends beyond scrutinizing outgoing traffic. We looked into responses within the fingerprint generation module as well checking the status code. This capability enables us to assess communication, whether within a well-configured client-server scenario or when dealing with applications within our system that solely transmit data without anticipating responses, or perhaps only expecting a 404 error (indicating requested content not found).

We had issues univocally identifying a browser from the features collected at the beginning of the experiment. In particular, we would have some issues if a malware application was able to imitate the specific browser user by the user on that host. The issue is caused by the entropy seen in the traffic generated by the browsers to different hosts that generated different MIME types and Responses when requested in different times for instance.

Evasion. Our solution is not easy to evade with simple evasion techniques such as spoofing user agent or other header values. To successfully mask data exfiltration, a malicious actor would need to replicate both outgoing features and responses from an external server of an existing application on the monitored hosts. However, this does not make our solution impossible to evade as we have mentioned previously in the article. In fact, the identifying browsers with the features used is still not robust enough to univocally identify different browsers by the type of data sent and response from the servers.

Recent research from Davis et al. [16] analyses the validity of using human expertise to blindly choose the most relevant features for them. Thus, the authors generate a total of 1,141 for each HTTP connection and select the top 25 features to be used. The result achieved a detection rate above 99.93% at a false positive rate below 0.01%. However, the huge number of features considered make the solution unusable in large enterprise networks. In particular, it requires the usage of unigrams of the request body, which extends beyond our research focus. For this research we constructed the set of features used by manually inspecting network traffic produced by both malicious and normal traffic to find discriminating attributes. We also review other related works.

VIII. Related Work

In this section we review several works that explore different network-based monitoring techniques.

A. *Application fingerprinting approaches*

Application fingerprinting is based on active and passive techniques in which information about different features of the environment, web browser and OS are extracted. While for active fingerprinting different techniques are used, such as JavaScript to query information about the list of browser plugins or screen resolution, passive fingerprinting techniques analyse network traffic. Bortolameotti et al. presented in DECANTeR [6] a system for detection of anomalous HTTP network traffic by passively extracting fingerprints of benign applications running on the host. This process involves extracting information from clustered POST and GET requests in the form of Host header value, constant header fields, average request size, User-Agent header value, Accept-Language header value, and the size of outgoing information in the cluster. In another work, Headprint [45], Bortolameotti et al. presented a system for the detection of anomalous traffic that does not rely on a small set of predefined features to model the message content, but it automatically identifies the most characterizing content directly. Both these techniques covered both web browsers and background applications running on workstation and server. However, they require more computational and storage cost compared to our work to monitor and process network traffic.

John Althouse, et al. [41] distinguishes itself from passive application fingerprinting methodologies by focusing on

SSL/TLS handshake patterns as a means of identifying application behavior.

Several works contribute to our comprehension of mobile app behavior through varied approaches. Miskovic et al. proposed [42] AppPrint, an automatic fingerprinting technique, which uses different HTTP features in combination with machine learning algorithms. Similarly, Taylor et al. [43] extends this capability to encrypted traffic scenarios using statistical features of the packet length (e.g., mean, skew) of incoming, outgoing, and complete flows to train traditional machine learning models. A step further is taken by Van Ede et al., which introduced FlowPrint a semi-supervised approach for fingerprinting mobile application that creates fingerprints based on 5 min captures of traffic and strong correlations (i.e., cross-correlations). Moreover, their approach uses Jaccard similarity [46] to cluster flows with fingerprints created from a set of network destinations to form a maximal clique in the correlation graph. All these solutions explore mobile applications communication to generate mobile application fingerprints, which differ to common workstation and servers network traffic.

B. Threat-specific approaches

These works have the common goal of understanding the characteristics and patterns of malicious traffic. Some works analyse and cluster malicious traffic into families to train classifiers to spot generic C&C communication channels. For example, the work of Perdisci et al. [9] propose a system that cluster network-level behaviour of malware by focusing on URL similarities among traces of malicious HTTP traffic.

Other works focus on specific traffic patterns generated by botnets [10][11][12]. In particular, Botfinder [12] uses machine learning to identify key features of C&C communication, based on the observation of traffic generated by bots in a controlled environment. Instead, the research of Al-Bataineh et al. [13] focus on the network behaviour of the Zeus malware to identify its unique pattern in HTTP POST requests.

All these works help to identify patterns of unknown malware attacks, as long as they have common features to the malware families used during the training phase. However, effective analysis is hard to achieve as they require a deep understanding of malware specific actions and characteristics, done by tools performing static or dynamic analysis. Thus, they end up requiring a huge dataset of malware samples to properly train a detection model. On the other hand, our focus is on benign network traffic so that we can detect anomalies without any prior knowledge on malware.

C. Anomaly-based approaches

Unlike the previous approach, these works use only the legitimate traffic generated by hosts within the organisation to detect potentially a wide range of novel attacks. However, they strongly depend upon the quality of the patterns chosen to define the baseline 'normality'. If this is not properly defined, the approach can miss the detection of new malware or even miss known ones.

Web Tap [14] applies anomaly detection techniques and statistical model to the application layer to find unusual HTTP request headers indicative of a tunnel. Thus, its goal is to spot non-standard web browsing traffic. Nevertheless, the evaluation of the system showed that 12% of 767 alerts was wrongly classified as malicious (i.e., false positives). An improvement of Web Tap in terms of false positive is obtained by another tool, called DUMONT [15]. It is a similar system that uses a hierarchy of one-class Support Vector Machine (SVM) to classify flows as legitimate or malicious. It uses lengths, structure, timing and entropy calculated from HTTP requests as key features. Even though they had a lower amount of false positive, their average detection rate was 89.3% as it is strongly user-dependent. Both these tools (Web Tap and DUMONT) are constrained to the fact that they need a high computational and storage costs. Not because they can easily be evaded with encryption techniques, as they inspect the payload. In addition to that, they do not provide any solution in case of changes in the host behaviour as new applications are installed or old ones are updated.

A recent research from Davis et al. [16] analyses the validity of using human expertise to blindly choose the most relevant features for them. Thus, the authors generate a total of 1141 for each HTTP connection and select the top 25 features to be used. The result outperforms the previously mentioned solutions achieving a detection rate above 99.93% at a false positive rate below 0.01%. However, the huge number of features considered make the solution unusable in large enterprise networks.

Regarding data exfiltration detection methods using IPFIX, the only known research was done by Marchetti et al. [17]. Their framework can identify and rank suspicious hosts in a large organisation possibly involved in data exfiltration. They only use network traffic data, such as the number of bytes uploaded, the number of flows initiated, and the number of external IP addresses related to an internal host. As it does not need to inspect payload data, their framework can work even for encrypted communications. Nevertheless, their list of hosts with high 'suspiciousness score' results constrained by the knowledge and availability of security analysts. On the other hand, we aim to obtain a list of anomalous connection alerts raised by the system.

IX. Conclusions and Future Work

In this work we have shown how NetFlow/IPFIX technologies can be used to detect anomalous communications in HTTP traffic. This solution requires a training phase without malicious data in order to generate an initial set of legitimate fingerprints of applications in host machines. Our approach gathers and analyses only traffic data extracted from headers field. The proposed set of features can detect the application type and possible data exfiltration of each flow analysed. The effectiveness of the proposed solution has been evaluated using two different datasets containing data from malware and from an organisation network. We have evaluated the system against other state of the art solution (i.e., DECANTeR), showing that with less features it can correctly detect anomalous traffic on background applications. Future work includes the integration of additional features in nProbe able to distinguish different kind of browsers and a deeper performance analysis using a bigger dataset with thousands of hosts.

REFERENCES

[1] David McCandless. World's Biggest Data Breaches &Hacks. http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

[2] Verizon. Verizon Data Breach Investigations Report (2022). https://www.verizon.com/business/resources/reports/dbir/

[3] IBM Security. Cost of Data Breach: Report 2022. https://www.ibm.com/reports/data-breach

[4] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, May 2016.

[5] Kerry G. Coffman and Andrew M. Odlyzko. Internet growth: Is there a "Moore's Law" for data traffic?. *Handbook of massive data sets.* Springer, Boston, MA, 2002. 47-93.

[6] Riccardo Bortolameotti, Thijs van Ede, Marco Caselli, Rick Hofstede, Maarten H. Everts, Willem Jonker, Pieter Hartel and Andreas Peter. DECANTeR: DEteCtion of Anomalous outbouNd HTTP TRaffic by Passive Application Fingerprinting. *Proceedings of the 33rd Annual Computer Security Applications Conference.* ACM, 2017.

[7] Petr Matousek, Ondrej Rysavy, Matej Gregr, and Martin Vymlatil. Towards identification of operating systems from the internet traffic: IPFIX monitoring with fingerprinting and clustering. In *Data Communication Networking (DCNET), 2014 5th International Conference on*, pages 1–7. IEEE, 2014.

[8] The Mitre Corportation. Commonly Used Ports. https://attack.mitre.org/techniques/T0885/

[9] Roberto Perdisci, Wenke Lee, and Nick Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *NSDI*, volume 10, page 14, 2010.

[10] Guofei Gu, Phillip A Porras, Vinod Yegneswaran, Martin W Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Usenix Security*, volume 7, pages 1–16, 2007.

[11] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, volume 5, pages 139–154, 2008.

[12] Florian Tegeler, Xiaoming Fu, Giovanni Vigna, and Christopher Kruegel. Botfinder: Finding bots in network traffic without deep packet inspection. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 349–360. ACM, 2012.

[13] Areej Al-Bataineh and Gregory White. Analysis and detection of malicious data exfiltration in web traffic. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pages 26–31. IEEE, 2012.

[14] Kevin Borders and Atul Prakash. Web tap: detecting covert web traffic. In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 110–120. ACM, 2004.

[15] Guido Schwenk and Konrad Rieck. Adaptive detection of covert communication in http requests. In *Computer Network Defense (EC2ND), 2011 Seventh European Conference on*, pages 25–32. IEEE, 2011.

[16] Jonathan J Davis and Ernest Foo. Automated feature engineering for HTTP tunnel detection. *Computers &Security*, 59:166–185, 2016.

[17] Mirco Marchetti, Fabio Pierazzi, Michele Colajanni, and Alessandro Guido. Analysis of high volumes of network traffic for advanced persistent threat detection. *Computer Networks*, 109:127–141, 2016.

[18] Frank L Greitzer, Lars J Kangas, Christine F Noonan, A Dalton, and RE Hohimer. Identifying at-risk employees: A behavioral model for predicting potential insider threats. Report PNNL-19665. *Pacific Northwest National Laboratory (PNNL), Richland, WA (US)*, 2010.

[19] Ivo Friedberg, Florian Skopik, Giuseppe Settanni, and Roman Fiedler. Combating advanced persistent threats: From network event correlation to incident detection. *Computers &Security*, 48:35–57, 2015.

[20] Guodong Zhao, Ke Xu, Lei Xu, and Bo Wu. Detecting APT Malware Infections Based on Malicious DNS and Traffic Analysis. *IEEE Access*, 3:1132–1142, 2015.

[21] Singh, Abhay Pratap, and Mahendra Singh. A comparative review of malware analysis and detection in HTTPs traffic. In *International Journal of Computing and Digital Systems* 10.1 (2021): 111-123.

[22] Aldo Cortesi, Maximilian Hils, Thomas Kriechbaumer, and contributors. mitmproxy: A free and open source interactive HTTPS proxy. https://mitmproxy. org/

[23] SSL Forward Proxy - Decryption https://docs.paloaltonetworks.com/

[24] Wilkens, Florian, et al. Passive, transparent, and selective TLS decryption for network security monitoring. In *IFIP International Conference on ICT Systems Security and Privacy Protection.* Cham: Springer International Publishing, 2022.

[25] Ruoti, Scott, et al. User attitudes toward the inspection of encrypted traffic. In *Twelfth Symposium on Usable Privacy and Security.* 2016.

[26] Rick Hofstede, Pavel Cˇeleda, Brian Trammell, Idilio Drago, Ramin Sadre, Anna Sperotto, and Aiko Pras. Flow monitoring explained: From packet capture to data analysis with netflow and ipfix. *IEEE Communications Surveys &Tutorials*, 16(4):2037–2064, 2014.

[27] B. Claise, B. Trammell, and P. Aitken. Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information. *RFC 7011 (Internet Standard), Internet Engineering Task Force (IETF)*, September 2013.

[28] Luca Deri. nProbe: An Open Source NetFlow Probe for Gigabit Networks. In *Proceedings of the TERENA Networking Conference*, TNC'03, 2003.

[29] Eckersley, Peter. How unique is your web browser?. *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, Berlin, Heidelberg, 2010.

[30] Ryan Van Antwerp. Exfiltration techniques: An examination and emulation. *PhD thesis, University of Delaware*, 2011.

[31] Annarita Giani, Vincent H Berk, and George V Cybenko. Data exfiltration and covert channels. In *Defense and Security Symposium*, pages 620103–620103. International Society for Optics and Photonics, 2006.

[32] Annarita Giani, Vincent H Berk, and George V Cybenko. Data exfiltration and covert channels. In *Defense and Security Symposium, pages 620103–620103. International Society for Optics and Photonics*, 2006.

[33] Clint Andrew Hall. Web presentation layer bootstrapping for accessibility and performance. In *Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibility (W4A)*. ACM, 2009. p. 67-74.

[34] Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten Van Steen, Felix C Freiling, and Norbert Pohlmann. Sandnet: Network traffic analysis of malicious software. In *Proceedings of the First Workshop on Building Analysis Datasets and Gathering Experience Returns for Security*. ACM, 2011. p. 78–88.

[35] Mary M. Moya, and Don R. Hush. Network constraints and multi-objective optimization for one-class classification. *Neural Networks*, 1996, 9.3: 463-474.

[36] Shehroz S. Khan, and Michael G. Madden. A survey of recent trends in one class classification. In *Irish Conference on Artificial Intelligence and Cognitive Science*. Springer, Berlin, Heidelberg, 2009. p. 188-197.

[37] John A. Quinn, and Masashi Sugiyama. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters*, 2014, 40: 36-40.

[38] Petr Velan; KREJČÍ, Radek Krejcˇì. Flow information storage assessment using IPFIXcol. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*. Springer, Berlin, Heidelberg, 2012. p. 155-158.

[39] Google. HTTPS encryption on the web. https://transparencyreport.google.com/https/overview

[40] Frediani, Carola, et al. Attacco ai pirati. L'affondamento di Hacking Team: tutti i segreti del datagate italiano. (2015)

[41] John Althouse, et al. TLS Fingerprinting with JA3 and JA3S

[42] Miskovic, Stanislav, et al. Appprint: automatic fingerprinting of mobile applications in network traffic. In *Passive and Active Measurement: 16th International Conference,* PAM 2015, New York, NY, USA, March 19-20, 2015, Proceedings 16. Springer International Publishing, 2015.

[43] Taylor, Vincent F., et al. Appscanner: Automatic fingerprinting of smartphone apps from encrypted network traffic. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 2016.

[44] Van Ede, Thijs, et al. Flowprint: Semi-supervised mobile-app fingerprinting on encrypted network traffic. In *Network and distributed system security symposium (NDSS).* Vol. 27. 2020.

[45] Bortolameotti, Riccardo, et al. Headprint: detecting anomalous communications through header-based application fingerprinting. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing.* 2020.

[46] Paul Jaccard. The Distribution of the Flora of the Alpine Zone. New Phytologist, 1912.

APPENDIX A
FIGURES AND TABLES

Figure 2 shows a comparison between different algorithms used on DEM for the browser identification module on our solution used to identify the type of application.

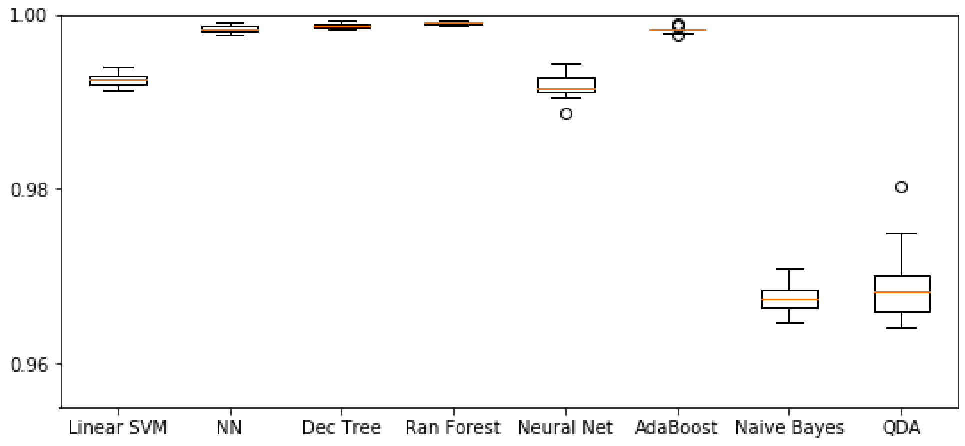

Fig. 2: Comparison of the results between different classification models tested on browser identification module.

Figure 3 shows a visual example of output generated by the analysis of certain type of features generated by browsers.

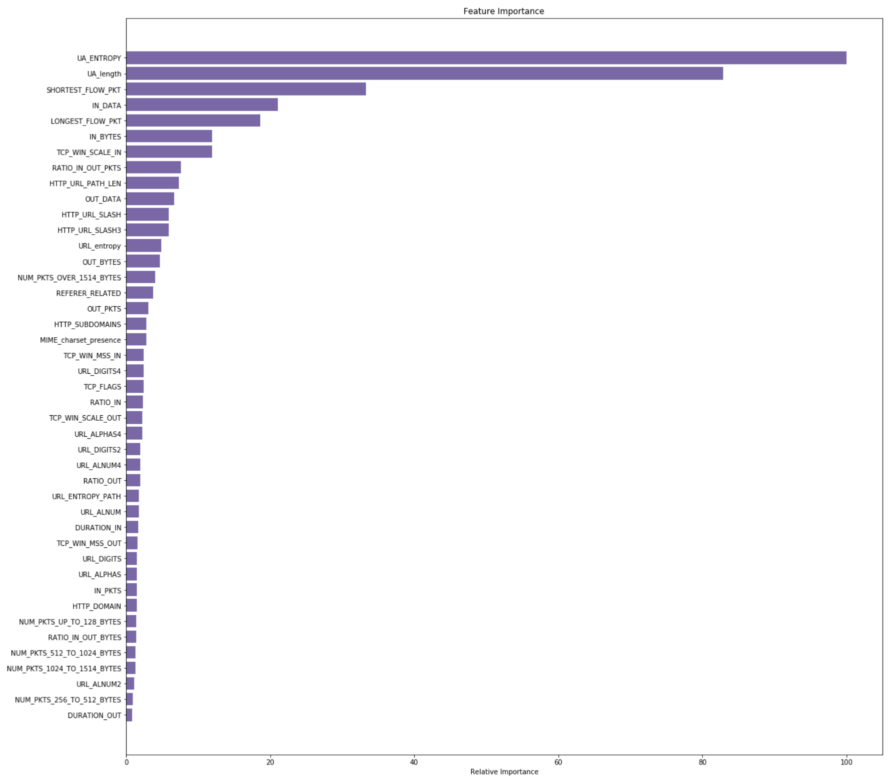

Fig. 3: Feature importance for network flows generated by browsers.