Fingerprinting Web Pages and Smartphone Apps from Encrypted Network Traffic with *WebScanner*

Pedro Casas[*], Nikolas Wehner[†], Sarah Wassermann[*], Michael Seufert[†]
[*]AIT Austrian Institute of Technology, [†]University of Würzburg

***Abstract*—Traffic encryption reduces the visibility of Internet Service Providers (ISPs) on the services consumed by their customers. This is particularly challenging for monitoring and analysis of web and apps traffic, which is highly complex and heterogeneous. Loading a web-page or app today requires tens of flows to download the various resources located in distributed cloud servers from different content providers. We introduce *WebScanner*, a web-page and app fingerprinting approach capable to identify all the traffic flows corresponding to individual web-page and app loading sessions within concurrent web pages traffic, enabling highly detailed, per web-page analysis in practical deployments. Different from the state of the art in web and app traffic fingerprinting, *WebScanner* automatically performs the parsing of all the (encrypted) traffic generated by a web visit and its isolation from concurrent traffic, instead of assuming that an external oracle system does so. *WebScanner* also implements a *deep fingerprinting* approach to detect user action-dependent traffic from apps, relying on simple machine learning models and strong input features as fingerprints. Extensive evaluation across a large measurement dataset of popular web pages and mobile apps confirms the outstanding performance of *WebScanner*, identifying the top-500 Alexa websites with precision and recall (P/R) above 95%, isolating their full contents with P/R above 80% for up to 15 concurrent web pages visited by the same device, and detecting specific action-dependent apps traffic with average P/R above 92%.**

I. Introduction

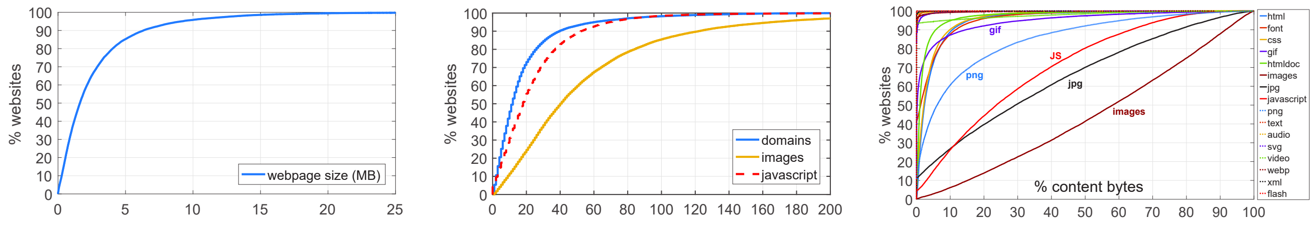
Network traffic monitoring is paramount for Internet Service Providers (ISPs), to improve their visibility, to quickly detect and resolve performance issues, as well as to optimize resources. Approaches like Deep Packet Inspection (DPI) have been traditionally used to obtain application-layer information regarding the health of applications and web services from the network side; however, traffic encryption is limiting the applicability of DPI. To make matters more challenging, modern web and app traffic is served from highly distributed cloud infrastructures through complex content delivery networks, and a single web-page visit or loading of an app can result in tens or even hundreds of end-to-end encrypted traffic flows, generating a difficult to interpret tangle. Network management applications such as performance monitoring, dynamic resource allocation, traffic classification, or even proactive web security (e.g., traffic *honypotting*) require identifying and isolating all the flows corresponding to each single web-page visit or app loading session.

The need for increased visibility on encrypted web traffic has motivated a surge in the conception of Machine-Learning (ML) based approaches to fingerprint and detect the traffic generated by specific web pages and web applications. Naturally, there is a very broad literature in the problem of encrypted traffic classification for web pages and apps, but for the majority of these previous work, they assume that all the traffic flows generated by the web-page/app instance to classify are disentangled and isolated from concurrent web pages traffic and background traffic by some external means, not evaluated as part of their classification pipeline.

This is a major limitation on the applicability of the state of the art in web pages and apps fingerprinting in practice. We therefore introduce *WebScanner*, a web-page and app fingerprinting approach capable to identify all the traffic flows corresponding to individual web-page and app loading sessions within concurrent web pages traffic, enabling highly detailed, per web-page analysis in practical deployments. Different from the state of the art in web and app traffic fingerprinting, *WebScanner* automatically performs the parsing of all the (encrypted) traffic generated by a web visit and its isolation from concurrent traffic, instead of assuming an external oracle system doing it. *WebScanner* can fully disentangle 15 different web pages visited by the same device (or multiple devices sharing the same IP address, e.g., NATing) at the same time (e.g., multi-tab browsing, or background traffic). In the context of this study, a session corresponds to all the flows exchanged between the end-user's device IP and the IPs of the servers hosting the web-page embedded contents, for one specific web-page loading instance. Besides detecting specific web pages/apps and filtering/disentangling their contents (flows), *WebScanner* employs ML techniques to fingerprint and identify different user action-dependent sessions generated by popular apps, such as product search in the Amazon e-commerce app, or Amazon video, app scrolling, etc.

Through extensive evaluation on a broad dataset composed of desktop and smartphone web browsing measurements for the top-500 Alexa websites, as well as instrumented measurements for four popular apps and 11 different user actions, we demonstrate the fine-granularity monitoring capabilities enabled by *WebScanner*, being able to detect single web pages and to isolate all the traffic flows belonging to these with (P)recision and (R)ecall above 95% (detection) and 80% (disentangling), for up to 15 concurrent web pages visited by the same IP address at the same time, and detecting specific action-dependent apps traffic with average P/R above 92%.

The remainder of the paper is organized as follows. Sec. II presents some background on the complexity of current Inter-



(a) Web-page size (MB). (b) Number of domains, connections, and relevant resources. (c) Share of content types.

Fig. 1: Heterogeneity of Internet web pages, for the top 10.000 Alexa websites in 2021.

net web pages, and briefly overviews related work. Sec. III presents the techniques employed by *WebScanner* for web traffic identification and filtering, as well as the fingerprinting, ML-based approach for app user-action classification. Extensive evaluation results are reported in Sec. IV – for web traffic, and in Sec. V – for apps, including the characterization of the traffic measurements, an analysis of the fingerprints, and the assessment on the detection and disentangling of web pages, as well as the evaluation of the app classification approach. Finally, Sec. VI presents concluding remarks.

II. Background & Related Work

To show evidence on the broad heterogeneity and content richness of Internet web pages today, Fig. 1 depicts the (a) size, (b) number of domains and relevant resources, and (c) share of content types, for the top 10.000 Alexa websites in 2021, extracted from the HTTP Archive public dataset (https://httparchive.org/) in Google BigQuery. web pages size and content significantly varies, and a single web-page loading today might require to download contents from up to 200 different domains; half of the pages have more than 20 javascript resources and embed more than 40 images.

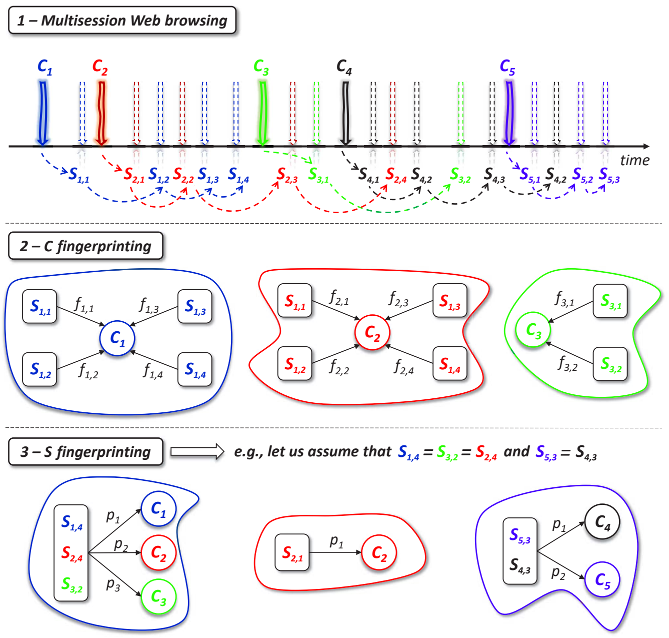
There is a vast literature in the problem of web [3]–[6], [9] and app [10], [11] traffic fingerprinting and classification, for end-to-end encrypted traffic. A complete survey is presented in [7]. Most of these approaches are based on the application of supervised or semi-supervised ML models to a set of features derived from the encrypted traffic. Their target is not exactly the same as ours, as their main application is to identify a certain web-page or app instance, whereas we are interested in identifying and isolating all the traffic belonging to each session; nevertheless, their main limitation is assuming a non-realistic application setup, where all the traffic flows belonging to a single instance are already perfectly isolated from whatever background or concurrent web traffic. In fact, these papers explicitly mention that those cases where background traffic may be present, for example from multi-tab browsing, is out of their scope and left fir future work. This is a clear limitation in the practice, as fingerprint computation becomes noisy and prone to impactful errors. Two papers [6], [8] are mentioned as potential solutions to this disentangling problem, but the proposed solutions are rather limited to the full identification goal of *WebScanner*: the solution presented in [8] is only tested with up to two concurrent web pages, explicitly mentioning that accuracy would be lower for more than two pages; the solution in [6] uses Server Name Indication (SNI), an extension to TLS which directly refers to the specific HTTPS service being accessed, and while this approach is useful to identify different services hosted by the same server, it does not help in isolating the traffic from all the flows coming from completely different servers.

Previous work [12] has taken a similar direction for web traffic disentangling, introducing PAIN; while *WebScanner* has multiple similarities with PAIN, it also has important differences: *WebScanner* traffic identification and disentangling are fully-supervised approaches, based on off-line active measurements and fingerprinting, whereas PAIN is meant for unsupervised operation, and requires passive measurements for web pages accessed by a large number of users. As stated in [12], PAIN cannot monitor a single visit to a specific webpage, whereas that is exactly the target of *WebScanner*. In addition, *WebScanner* uses a double fingerprinting approach to identify web pages directly requested by the user, instead of simply tracking pre-defined web-page names defined by the operator. To conclude, we have recently applied *WebScanner* to the problem of Web QoE inference and analysis [1], fingerprinting and isolating the traffic from independent web pages to extract relevant traffic features for ML-driven QoE inference and analysis. Different from [1], the focus of this paper is on the fingerprinting techniques, both for web pages and apps.

III. Web Disentangling and App Fingerprinting

In practice, and depending on the specific vantage point where the monitoring system is deployed, sessions are concurrent, originating either at individual (e.g., multi-tab browsing) or multiple users, which might even share the same origin IP address if located behind a NAT gateway. Once a user visits a certain website, the browser opens multiple connections to the different servers, fetching all the contents which compose the corresponding web-page, including both local web contents as well as third-party embedded contents, such as advertisements and more, see Fig. 2. We refer to the domain associated with the web-page requested by the user as the *core domain* C, and the subsequently contacted domains as *support domains* S_j. From now on, let us assume we have a total of w core domains or web pages, and d unique support domains.

Next, we introduce an approach to detect and isolate all the flows corresponding to an individual session. *WebScanner* tackles two specific challenges: (i) firstly, it is able to identify



1 – Multisession Web browsing
time
2 – C fingerprinting
3 – S fingerprinting
e.g., let us assume that $S_{1,4} = S_{3,2} = S_{2,4}$ and $S_{5,3} = S_{4,3}$

Fig. 2: WebScanner fingerprinting concept and definitions for web traffic.

all the core domains C_i present in a group of mixed web traffic flows, and (ii) secondly, given a core domain C_i, it can identify all the support domains $S_{i,j}$ associated with it. We refer to the first problem as the *click-stream identification* – i.e., identifying web pages directly requested by the user – and to the second problem as *session disentangling* – i.e., splitting flows into their associated sessions. An additional complexity in these two problems is that a certain domain could be support for multiple different core domains, and that a core domain itself could actually be a support domain for a different web-page. Note that *WebScanner* assumes traffic flows are tagged with their associated domain names, as observed in the DNS queries. In such a scenario, correctly identifying core and support domains of a web session directly translates into a matching reconstruction of the network traffic flows. In this paper we do not address the usage of DNS encryption.

Besides solving the click-stream identification and session disentangling problems, *WebScanner* implements a simple traffic fingerprinting approach to classify app traffic into different user action related classes; the main idea and assumption is that apps (and eventually web pages) can be distinguished by the sequence of the packets' direction, either uplink or downlink.

A. Click-stream Identification

We tackle the click-stream identification problem through a fully supervised, fingerprints-based approach, where fingerprints are derived from the sequence of core and associated support domains observed in multiple web-page loading sessions. Given a collection of w web pages – accessible through their core domains $C_i, i = 1, \ldots, w$ – each visited (i.e., measured) k times during a fingerprint-training period, we construct two types of fingerprints: (i) a core domain fingerprint for each C_i, and (ii) a support domain fingerprint

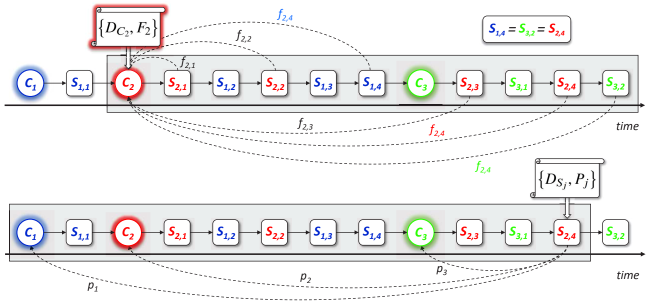


Fig. 3: Application of WebScanner fingerprints for core and support domain detection and association.

for each $S_{i,j}$. A C_i fingerprint corresponds to a list of all the associated $S_{i,j}$ domains observed during the k web-page loading sessions, along with an associated frequency of occurrence, reflecting the percentage of loading sessions where each $S_{i,j}$ is associated to the corresponding C_i. Each of these frequency values ranges from 0+ to 1, where 1 means that this $S_{i,j}$ is always present in loading sessions for web-page C_i, and 0+ that it appears occasionally. On the other hand, a fingerprint for $S_{i,j}$ corresponds to a list of all C_i which this support domain is associated to through the complete set of $w \times k$ measurements, along with the corresponding frequency of occurrence of each C_i; in this case, frequency values represent the percentage of loading sessions each core domain is associated to this support domain, within all the loading sessions where $S_{i,j}$ appears as a support domain. Each frequency value ranges from 0+ to 1, but the sum of all values is always equal to one this time. Fig. 2 explains this concept.

Let us consider web-page C = *instagram.com* as example. A potential fingerprint for C could be the tuple $\{D_C, F\}$, where D_C = {*connect.facebook.net, scontent.xx.fbcdn.net, s.10.facebook.com, static.facebook.com, www.facebook.com, www.instagram.com*} and $F = \{1, 0.014, 0.014, 1, 1, 1\}$. For S = *outlook.live.com* we could have the tuple $\{D_S, P\}$, with D_S = {*live.com*} and $P = \{1\}$. We generalize the fingerprinting notation to the full list of w web pages and d support domains as follows: $C = \{C_i\}$, $\mathcal{D}_C = \{D_{C_i}\}$, $\mathcal{F} = \{F_i\}, i = 1, \ldots, w$, and $\mathcal{S} = \{S_j\}$, $\mathcal{D}_S = \{D_{S_j}\}$, $\mathcal{P} = \{P_j\}, j = 1, \ldots, d$. For the sake of completeness, we specify $F_i = \{f_{i,j}\}, j = 1, \ldots, d_i$, where d_i corresponds to the number of support domains associated to C_i, and $P = \{p_j\}$ – we keep a flat notation for P, as D_S fingerprints can be a mix of any core domain.

We implement different click-stream identification methods, relying on core and support domain fingerprints. The main approach is straightforward – see Fig. 3, and consists of intersecting core and support domain fingerprints with the sequence of temporally observed domains in the traffic, ranking domains based on the corresponding frequencies F and P. In a nutshell, given a core domain C_i, we expect a higher overlap with the list of support domains associated to its fingerprint D_{C_i}; in addition, if more support domains are associated to the same C, it has a higher chance of being an actual core domain and not a support one. The corresponding identification approaches are described in algorithm 1:

Algorithm 1: Click-stream Identification

```
Inputs: $\mathcal{D}$, $\{C, \mathcal{D}_C, \mathcal{F}\}$, $\{\mathcal{S}, \mathcal{D}_{\mathcal{S}}, \mathcal{P}\}$, $\gamma_C$;
Output: $C^o \in \mathcal{D}$;
1  $C^o \leftarrow \mathcal{D} \cap C$;
2  $C^o = \{C_i^o\}, i = 1 \dots n$;
3  $\lambda_C \leftarrow$ zeros($n$);
4  $\lambda_S \leftarrow$ zeros($n$);
5  for ($i = 1;\ i \leqslant n;\ i{+}{+}$) {
6      $\lambda_C(i) \leftarrow \overline{F_i}\left(\mathcal{D}_{t>C_i^o} \cap D_{C_i^o}\right)$;
7      $\lambda_S(i) \leftarrow \overline{P_i}\left(\mathcal{D}_{t>C_i^o} \cap \mathcal{S}_{C_i^o \in \mathcal{D}_S}\right)$;
8  }
9  $\gamma_S \leftarrow \overline{\lambda_S}$;
10 CSF: $C^o \leftarrow C^o(\lambda_C > \gamma_C) \cap C^o(\lambda_S > \gamma_S)$;
11 SF: $C^o \leftarrow C^o(\lambda_S > \gamma_S)$;
12 CF: $C^o \leftarrow C^o(\lambda_C > \gamma_C)$;
13 CN: $C^o \leftarrow C^o$;          /* all $C_i^o$ are selected */
```

Algorithm 2: Web-page Disentangling

```
Inputs: $\mathcal{D}$, $\{C, D_C, F\}$, $\{\mathcal{S}, \mathcal{D}_{\mathcal{S}}, \mathcal{P}\}$;
Output: $\mathcal{S}_o \in \mathcal{D}$;
1  $\mathcal{S}^o \leftarrow \mathcal{D}_{t>C} \cap D_C$;
2  $\mathcal{S}^o = \{S_j^o\}, j = 1 \dots m$;
3  $\lambda_C \leftarrow$ zeros($m$);
4  $\lambda_S \leftarrow$ zeros($m$);
5  for ($j = 1;\ j \leqslant m;\ j{+}{+}$) {
6      $\lambda_C(j) \leftarrow F(S_j^o)$;
7      $\lambda_S(j) \leftarrow P_j^o\left(D_{S_j^o} \cap C\right)$;
8  }
9  $\gamma_C \leftarrow \overline{\lambda_C}$;
10 $\gamma_S \leftarrow \overline{\lambda_S}$;
11 DC: $\mathcal{S}^o \leftarrow \mathcal{S}^o$;      /* all $S_j^o \in D_C$ are selected */
12 DCF: $\mathcal{S}^o \leftarrow \mathcal{S}^o(\lambda_C > \gamma_C) \cap \mathcal{S}^o(\lambda_S > \gamma_S)$;
```

Core and Support Frequency (CSF) – this is the central and most complete approach implemented in *WebScanner*, relying on fingerprints from borh core domains C and support domains S. In the first step, all potential core domains C are pre-selected, based on the list of already fingerprinted core domains – e.g., the top-500 Alexa web pages in this study. For each potential core domain C, a score λ_C is computed, using the frequencies associated to the overlapping support domains S between the sequence of analyzed domains and the domains in the corresponding fingerprint D_C. In addition, using the S fingerprints, potential C are also ranked by their association to the list of analyzed S, computing a second score λ_S. The final selection of C consists of the intersection between the set of C which have a score λ_C higher than a certain threshold γ_C (in the solution we take $\gamma_C = 0.5$), and the top ranked C according to the λ_S score. Fig. 3 briefly describes this concept.

For the sake of evaluation and benchmarking, we also implement three other approaches, which are simpler variations of CSF, including (i) **Support Frequency (SF)** – selects top ranked core domains C according to S fingerprints only, using γ_S = mean(λ_S) as threshold on the λ_S score; (ii) **Core Frequency (CF)** – selects top ranked core domains C according to C fingerprints only, using score λ_C and $\gamma_C = 0.5$; and (iii) **Core Näive (CN)** – selects code domains C based exclusively on the list of known C for which fingerprints are available.

B. Session Disentangling

Once the core domains requested by the user have been detected, the next step consists of identifying the components (i.e., traffic flows) of each of the websites present in the mix of traffic. This corresponds to the identification of the corresponding support domains, for each of the previously detected C. We implemented two different approaches to realize this web session disentangling task – see algorithm 2: (1) **Domains in C fingerprints (DC)** – this is the most basic approach, and consists of identifying all the corresponding S, using C fingerprints. In a nutshell, this is achieved by intersecting the set of S in the corresponding fingerprint D_C, with the list of domains which are observed *temporally after* the specific C we are trying to reconstruct; (2) **Domains in C Fingerprints with Frequencies (DCF)** – this is a more robust approach and the one implemented in the solution, using the same approach as DC, but additionally using C and S fingerprint frequencies F and P to only keep the S with higher probability of belonging to the corresponding C. Note that the web traffic disentangling additionally filters all the background traffic which is not associated to the reconstructed web pages.

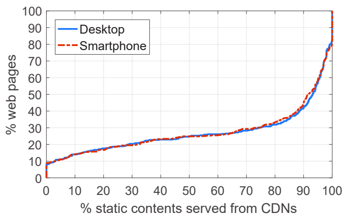
C. App Fingerprinting and Classification

Based on the idea that the packet direction sequences generated by an app (or web-page) generates a specific and unique fingerprint – and specifically for the very first packets, we train ML models on top of very simple traffic features as classifiers to discriminate among different apps and multiple user actions. In particular, we use a standard binary encoding on the uplink/downlink sequence of the first q packets as input features, using +1/-1 for uplink/downlink directions. We complement these q features with the total number of packets in uplink and downlink directions, the total number of flows, and the total size of the first q packets in uplink, and the first q packets in downlink. We trained and tested multiple classifiers, but given that our study is limited to a small number of apps and user actions (4 apps and 11 user actions), we decided to take the simplest model, which already provides highly accurate results. For the sake of simplicity, *WebScanner* uses a linear classifier for the identification task, applying softmax regression to address the multi-class nature of the task.

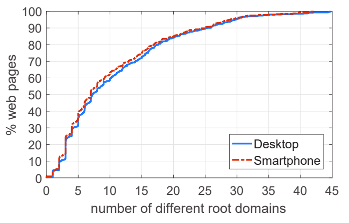
IV. *WebScanner* Evaluation

A. Datasets Characterization

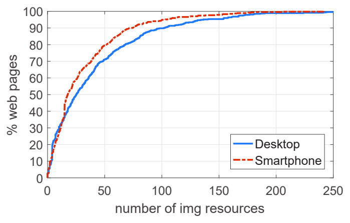
To construct labeled datasets for both web browsing and apps, we use a measurement testbed developed in [2], based



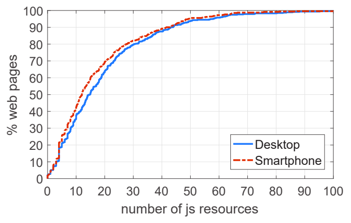
(a) Hosted contents.



(b) Root domains.



(c) Images.

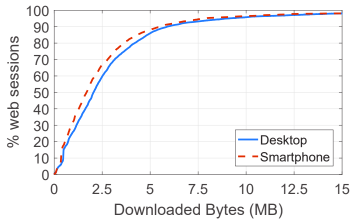


(d) Java-script.

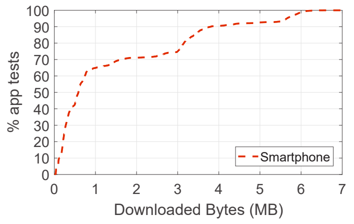
Fig. 4: Web pages characterization, per device type, including CDN hosted contents, number of root domains, and specific image and java-script contents.

dataset	# samples	browser/OS	# pages/app-actions
Web-Desk	25,000	chrome	alexa top-500
Web-Smart	25,000	chrome+android	alexa top-500
APP	13,500	android apps native/web/hybrid	4 apps/11 actions
Total	63,500		

TABLE I: Heterogeneous datasets for fingerprinting and evaluation.



(a) Web Pages.



(b) Apps.

Fig. 5: Downloaded bytes for web pages and app user-actions.

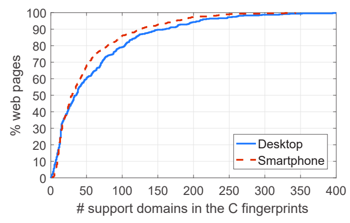
on WebPageTest (WPT) [14] and Appium libraries [15]. This testbed offers network emulation capabilities to simulate different access technologies in terms of network performance (e.g., latency, throughput, packet-loss), which adds variability and heterogeneity to the datasets. For example, the list and order of support domains may vary under different end-to-end delay conditions, due to the application itself, the transport protocols, or the content delivery technology. Laptops and smartphones are used as the instrumented devices, running WPT agents for Linux amd Android, and Chrome is the web browsing application. Tab. I describes the datasets conceived for web and app fingerprinting and validation purposes. Two datasets correspond to web browsing in desktop (*Web-Desk*) and smartphone (*Web-Smart*) devices, in both cases targeting the top-500 most popular websites using Alexa top-sites list (updated to Nov. 2020). The *APP* dataset corresponds to measurements for instrumented apps. These measurements require per-app instrumentation [13], which makes it difficult to scale to a large number of apps. In this study, we instrumented four popular apps covering different app-technologies, including YouTube (YT) and Facebook (FB) – native apps, Amazon (AMZN) – hybrid app, and BBC News (BBC) – web app. The Appium measurement framework allows for the instrumentation of independent *User Actions* (UA) within each app; we therefore tested different types of UA, including *app-startup* (i.e., loading of main page on app opening), *scrolling*, *tab-search*, and menu items/links *clicking*. The complete joint dataset consists of traffic captures for 63,500 individual web-page loading sessions and app UA.

Fig. 4 describes the 500 tested pages in terms of (CDN hosted) contents, per device type. The complexity of the pages is reflected by the (a) fraction of hosted contents and (b) their location at different root domains, with about 30% of the pages embedding resources located at 15 or more different root domains, and with more than 80% of the pages having 30% or more of its contents hosted in distributed CDNs. The biggest share of content bytes in modern web pages corresponds to (c) images and (d) java-script contents, with 60% of the pages having more than 20 embedded image resources and more than 10 java-script resources. Fig. 5 shows the downloaded bytes for (a) web pages and (b) app UA. Desktop-browsed web pages are bigger than those browsed on smartphones, which are optimized for smaller screen sizes. The average page size is 2.7MB in desktop and 2.1MB in smartphone.

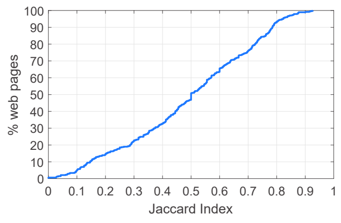
B. *Analysis of fingerprints*

We study web-page fingerprints for desktop and smartphone, separately. The first observation is that *C* fingerprints can be quite big, and that more support domains are observed for desktop web pages. According to Fig. 6(a), while fingerprints consist of less than 25 support domains *S* for about 40% of the top-500 websites, this number grows to more than 80 and 100 for about 20% of the websites in smartphone and desktop, respectively. For better interpretation of the characteristics of the fingerprints, Fig. 8 depicts a graph representation of core and support domains, for the-500 Alexa websites in desktop. In general, and as we see next, fingerprints are markedly different, as noted by the "broccoli"-like shape of the graph. To measure the degree of similarity between fingerprints, we use the well-known Jaccard-Index (JI). A JI = 1 indicates complete similarity, whereas a JI = 0 reflects total difference. Fig. 6(b) depicts fingerprints' similarity between the same web pages accessed in desktop and smarthpone devices. Interestingly, differences are significant between desktop and smartphone, with roughly half of the web pages having a JI below 0.5. This justifies the construction of fingerprints for desktop and smartphone separately.

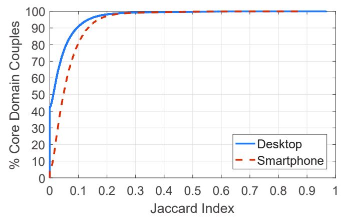
Regarding the identification of web pages and their contents using the produced fingerprints, the less overlapping the fingerprints – i.e., the smaller the JI, the better for the targeted task. Fig. 6(c) reports the JI values for all couples of *C* fingerprints,



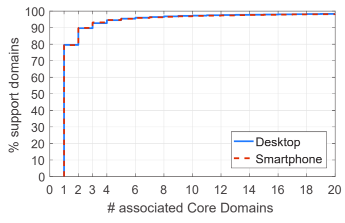
(a) C fingerprints.



(b) C fingerprints similarity for device types.

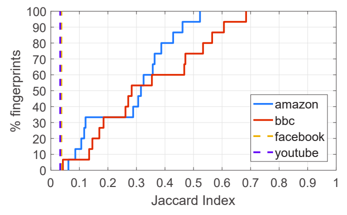


(c) C fingerprints similarity for top-500 web pages.

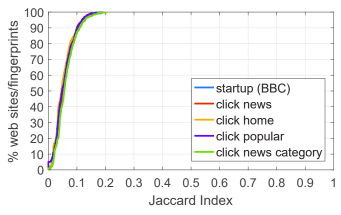


(d) S fingerprints.

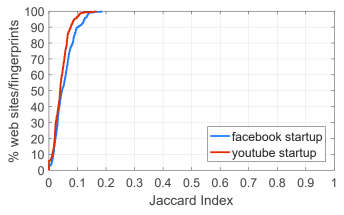
Fig. 6: Fingerprints for desktop and mobile web browsing. High discriminative power is obtained through C and S fingerprints.



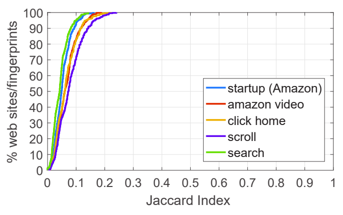
(a) UA + 4 Web fingerprints.



(b) BBC.



(c) YT/FB.



(d) Amazon.

Fig. 7: Analysis of App fingerprints. (a) Similarity within {UA fingerprints + 4 Web fingerprints}. (b-d) Similarity within {per-app UA fingerprints + top-500 Alexa Web fingerprints}. App UA fingerprints have a small overlap with the top-500 Alexa web pages' fingerprints, providing high discriminative power.

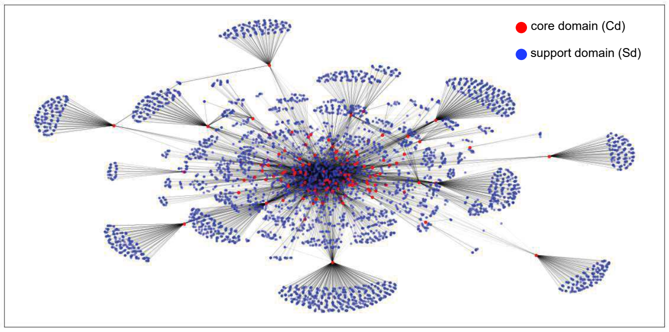


Fig. 8: Graph representation of core/support domains for top-500 Alexa.

for desktop and smartphone. The small JI values confirm a strong discrimination power of C fingerprints, with more than 90% and 80% of the web-page comparisons having a JI below 0.1, for desktop and smartphone, respectively.

In terms of S fingerprints, Fig. 6(d) reports the number of core domains associated to each of the support domains S observed in the studied web pages, for both smartphone and desktop. In total, we identify 12,384 unique support domains for desktop web pages, and 9,601 for smartphone web pages. Roughly 80% of these support domains are associated to only one core domain, enhancing the identification task through a reverse association. About 10% of the support domains are associated to exactly two core domains, and the rest has three or more associated domains. For example, support domains such as *www.google.com* or *www.facebook.com* are present in more than 50% of the web pages. We noted that support domains associated to Google services are in fact the most present in the measurements, most probably linked to both the usage of Chrome and Android as browser and OS, as well as the omnipresence of Google in the web space.

For the sake of completeness, we conclude with an analysis of the fingerprints corresponding to mobile apps, for the different tested UA. We particularly focus on the comparison of these fingerprints to their corresponding web versions, accessed through web browsing in smartphone. Fig. 7(a) shows the JI values for fingerprints corresponding to the four instrumented apps and their smartphone web browsing versions. Fingerprints are generated for each app and for each UA. The overlap between the FB and YT native apps and their corresponding webpages is minimal, resulting in a JI below 0.05. The overlap is significantly higher for the AMZN and BBC apps, which include a much broader set of UA; nevertheless, JI values are below 0.35 for more than 60% of the fingerprint comparisons. Fig. 7(b-d) show the JI values related to the overlap of UA with the complete set of web-pages, accessed in smartphone. JI values are small, below 0.15 for more than 95% of the comparisons, evidencing that app UA fingerprints do not overlap with web-page fingerprints, and can therefore also be used for app clickstream identification and disentangling. In this paper we only use C and S fingerprints for evaluation of the identification and filtering of web pages, and apply the ML-driven approaches for identification of apps.

C. Web Pages Identification and Filtering

We evaluate the different clickstream identification approaches on top of synthetically generated multi-concurrent web pages sessions, considering a different number of concurrent web pages, from two to 15 concurrent web pages. For each number of concurrent web pages, the evaluation corresponds to a set of 10.000 samples synthetically generated. These synthetic concurrent sessions are generated by randomly selecting traffic captures (pcap files) of individual loading sessions, and mixing them together, after properly aligning time references to actually make them concurrent. Fig. 9 reports the obtained results in terms or F1 scores, for desktop and smartphone devices, whereas Tab. II reports precision and recall (P/R) results. F1, P, and R values are high for

	CSF		SF		CF		CN		CSF		SF		CF		CN	
# pages	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R
2	97.5	96.5	61.5	90.0	90.5	98.0	84.4	100	94.7	98.5	35.7	99.0	89.8	99.5	83.9	100
3	96.4	95.0	73.8	81.2	89.7	98.2	83.6	100	98.3	97.3	50.3	98.5	92.6	98.2	88.1	100
5	99.0	92.3	84.9	63.1	89.1	97.7	84.7	100	97.2	97.7	67.9	95.9	92.7	98.7	88.0	100
8	97.1	96.4	70.9	80.9	89.9	98.4	85.4	100	97.8	96.5	80.2	89.1	92.2	97.9	87.4	100
10	96.4	93.9	77.9	74.1	90.3	97.0	86.7	100	98.5	96.5	86.7	83.1	93.4	97.6	89.1	100
15	99.5	91.6	84.8	59.5	91.4	96.2	88.0	100	98.6	95.6	93.0	68.6	92.8	97.4	88.7	100

TABLE II: Clickstream identification performance.

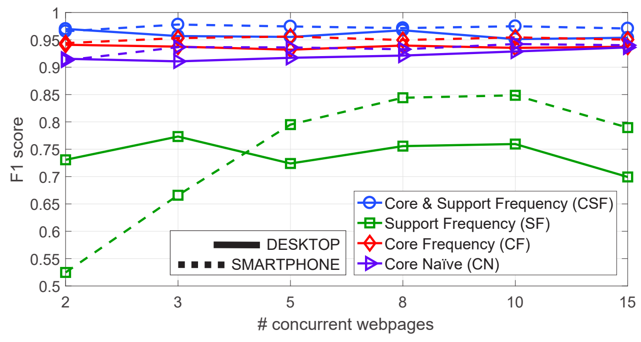


Fig. 9: Clickstream identification performance.

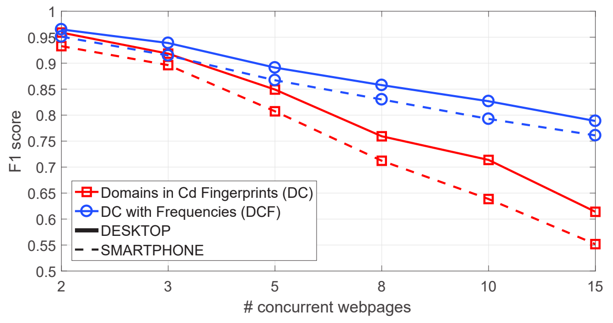


Fig. 10: Web-page disentangling performance.

	DC		DCF		DC		DCF	
# pages	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)	P (%)	R (%)
2	92.1	100	95.9	97.1	87.4	100	93.1	97.2
3	84.9	100	92.5	95.3	81.2	100	90.3	92.7
5	73.8	100	88.7	89.6	67.7	100	85.1	88.4
8	61.2	100	84.4	87.2	55.3	100	81.3	84.8
10	55.5	100	81.3	84.1	46.9	100	77.9	80.7
15	44.3	100	77.5	80.3	38.1	100	74.2	78.1

TABLE III: Web-page disentangling performance.

both device types, pointing to the good discrimination power of the fingerprints. The approach performing the worst is SF, which is expected, as it only relies on S fingerprints. Due to the nature of the approach and the evaluation, CN achieves 100% recall in all tests. As expected, precision drops for this approach, as some S are classified as C. As the number of concurrent pages grows, precision also increases, as the relative number of false positives decreases. CSF (the solution used by *WebScanner*) and CF perform similarly, but additionally using S fingerprints helps to increase precision of the identification approach. We follow a similar approach to evaluate the different web traffic disentangling approaches. In this evaluation, the list of C to reconstruct is given as input, and the task consists of identifying and correctly associating all the S to the correct C. Fig. 10 and Tab. III report the obtained results in terms of F1, P, and R, again for desktop and smartphone mixed web loading sessions, respectively. The basic approach DC achieves 100% recall for all the traffic mixes, based on the fingerprints intersection. However, the approach is not precise, resulting in a significant fraction of misclassifications for S in the traffic mix. Precision drops as we add more concurrent pages, as overlapping S occur more often. Using both C and S fingerprints (SCF, the approach used by *WebScanner*) significantly improves the precision of the disentangling approach, but naturally, sacrificing part of the identification performance in terms of recall. Still, disentangling and filtering performance is very high, even when dealing with 15 concurrent pages, both for smartphone and desktop devices.

V. App Fingerprinting Evaluation

To better explain the fingerprinting concept used by *WebScanner*, Fig. 11 depicts the obtained binary encodings for the different apps and user actions, in this case taking $q = 50$ packets, and for multiple different measurement runs of the apps with varying network performance. Fig 11(1) depicts the startup action for the four apps, Fig 11(2) considers four UA for AMZN – which is the most complex of the instrumented apps, and Fig 11(3) considers the fingerprints obtained for the web versions of the apps, which are conceptually equivalent to the app startup. Each app and action show a unique traffic pattern, supporting the concept for app and web fingerprinting. For example, Facebook shows the most upload packets relative to the first few packets, while BBC and Amazon show the most download packets at the beginning. BBC and Amazon show a quite similar behavior for the first 20 to 30 packets, before BBC starts generating a more regular consecutive upload and download pattern.

A. Detection of Apps

We start by evaluating the performance of *WebScanner* for detecting the startup of each single app. The data is split into 80%/20% training/test data, and 5-fold cross validation is used for training. Fig. 12(a) and Tab. IV report the obtained results in terms of F1 scores, P, and R. The classification accuracy is very high, above 95%, even if using a simple ML model.

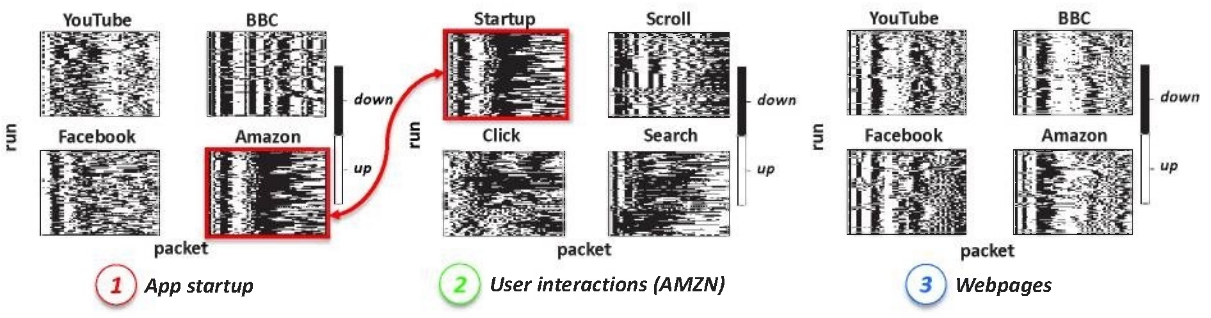
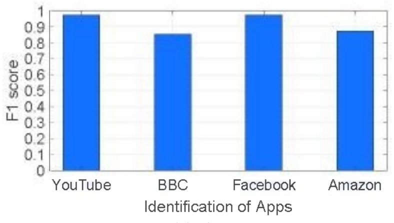
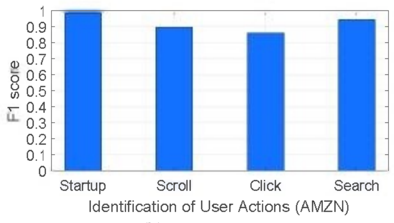


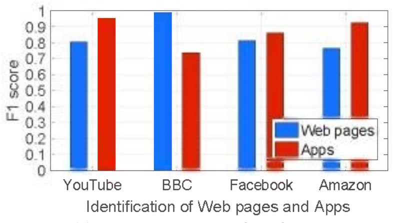
Fig. 11: Traffic patterns showing the observed sequences of upload and download packets.



(a) Apps.



(b) User actions.



(c) Mixing apps and web pages.

Fig. 12: App classification performance.

	Apps Startup				App actions (AMZN)				web pages vs Apps Startup							
	YT	BBC	FB	AMZN	Startup	Scroll	Click	Search	YT_w	BBC_w	FB_w	$AMZN_w$	YT_a	BBC_a	FB_a	$AMZN_a$
P (%)	94.9	85.7	96.1	90.8	98.2	88.6	91.1	91.3	86.6	99.6	76.5	77.6	91.9	85.9	78.8	92.0
R (%)	99.6	85.3	98.8	83.8	99.1	91.1	81.7	98.1	75.3	98.4	86.7	75.3	98.8	64.8	94.7	92.8

TABLE IV: App classification performance.

Most misclassifications occurred for the Amazon and BBC apps, resulting in a F1 score of 85% for Amazon and 88% for BBC. In contrast, almost all instances of the Facebook app and the YouTube app are classified correctly. While these results are encouraging, we acknowledge that for large scale app monitoring, this simple approach will probably not suffice.

B. Detection of User Actions

The same scheme is now applied for the user actions. Fig. 12(b) depicts the obtained results for four of the Amazon user actions, including startup, scrolling, clicking, and searching. The identification of the startup action and the search action hereby result in a F1 score of 96% and 94%, respectively, while most misclassifications happen for the click action – F1 score of 90%. These instances are mistaken for scrolling or searching, which can be explained by the highly variant traffic pattern of the click actions observed before.

C. Detection of Services and corresponding Apps

To conclude, we evaluate the discrimination between app and website requests, considering a mix of both app startup and web-page loading sessions, cf. Fig 11(3). Traffic patterns for websites are noisier, resulting in less visible differences among websites. However, when considering a bigger number of q first packets, again differences between websites can be observed. This reveals that app and web services differ in the way they setup their network connections. We apply *WebScanner* to classify the mix of apps and web pages fingerprints. Fig. 12(c) and Tab. IV report the obtained results. The evaluation resulted in an overall accuracy of 86%, where most misclassifications happen for the BBC app, with a recall of 65%. In contrast, the BBC website is correctly identified in almost all cases. All in all, all web pages and apps can be identified with a F1 score above 70%.

VI. Concluding Remarks

We have presented *WebScanner*, a web and app fingerprinting approach for monitoring of encrypted network traffic, offering web traffic identification and disentangling without assuming a pre-identification and parsing of the encrypted traffic flows. Through extensive evaluation across a large measurement dataset of popular web pages and mobile apps, we have shown that *WebScanner* can identify the top-500 Alexa websites with precision and recall (P/R) above 95%, isolating their full contents with P/R above 80% for up to 15 concurrent web pages visited by the same device, and detecting specific action-dependent apps traffic with average P/R above 92%. Naturally, there are some caveats and limitations in *WebScanner*, for example the need of DNS flow tagging – the approach would not work in case DNS queries are encrypted, the need for re-training of the fingerprints for new deployments – linked to geographically dependencies of content distribution, as well as re-training on a temporal basis. Finally, in terms of scalability, note that *WebScanner* is meant to operate at vantage points closer to the end-users and not at the core of the network.

References

[1] Pedro Casas, Sarah Wassermann, Michael Seufert, Nikolas Wehner, Olivia Dinica, Tobias Hossfeld, "X-Ray Goggles for the ISP: Improving in-Network Web and App QoE Monitoring with Deep Learning," in *6th IFIP Network Traffic Measurement and Analysis Conference (TMA)*, 2022.

[2] Pedro Casas, Sarah Wassermann, Nikolas Wehner, Michael Seufert, Joshua Schüler, Tobias Hossfeld, "Mobile Web and App QoE Monitoring for ISPs - from Encrypted Traffic to Speed Index through Machine Learning," in *13th IFIP Wireless and Mobile Networking Conference (WMNC)*, 2021.

[3] Payap Sirinam, Nate Mathews, Mohammad Saidur Rahman, Matthew Wright, "Triplet Fingerprinting: More Practical and Portable Website Fingerprinting with N-shot Learning," in *26th ACM Conference on Computer and Communications Security (CCS)*, 2019.

[4] Payap Sirinam, Mohsen Imani, Marc Juarez, Matthew Wright, "Deep Fingerprinting: Undermining Website Fingerprinting Defenses with Deep Learning," in *25th ACM Conference on Computer and Communications Security (CCS)*, 2018.

[5] Pierre-Olivier Brissaud , Jérome Francois, Isabelle Chrisment, Thibault Cholez , Olivier Bettan, "Transparent and Service-Agnostic Monitoring of Encrypted Web Traffic," *IEEE Transactions on Network and Service Management*, vol. 16(3), 2019.

[6] Wazen M. Shbair, Thibault Cholez, Jérome Francois, Isabelle Chrisment, "A Multilevel Framework to Identify HTTPS Services," in *15th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2016.

[7] Wazen M. Shbair, Thibault Cholez, Jérome Francois, Isabelle Chrisment, "A Survey of HTTPS Traffic and Services Identification Approaches," *arXiv:2008.08339*, 2020.

[8] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, Ian Goldberg, "Effective Attacks and Provable Defenses for Website Fingerprinting," in *23rd USENIX Security Symposium*, 2014.

[9] Jamie Hayes, George Danezis, "k-fingerprinting: A Robust Scalable Website Fingerprinting Technique," in *25th USENIX Security Symposium*, 2016.

[10] Vincent F. Taylor, Riccardo Spolaor, Mauro Conti, Ivan Martinovic, "AppScanner: Automatic Fingerprinting of Smartphone Apps from Encrypted Network Traffic," in *1st IEEE European Symposium on Security and Privacy (Euro S&P)*, 2016.

[11] G. Aceto, D. Ciuonzo, A. Montieri, and A. Pescap, "Mobile Encrypted Traffic Classification using Deep Learning: Experimental Evaluation, Lessons Learned, and Challenges," *IEEE Transactions on Network and Service Management*, vol. 16(2), 2019.

[12] Martino Trevisan, Idilio Drago, Marco Mellia, "PAIN: A Passive Web Performance Indicator for ISPs," *Computer Networks*, vol. 149, pp. 115-126, 2019.

[13] Ashkan Nikravesh, Qi Alfred Chen, Scott Haseley, Xiao Zhu, Geoffrey Challen, Z. Morley Mao, "QoE Inference and Improvement without End-host Control," in *IEEE/ACM Symposium on Edge Computing (SEC)*, 2018.

[14] Patrick Meenan, "Web-Page Performance Testing," online available at https://github.com/WPO-Foundation/webpagetest, 2021.

[15] Dan Cuellar, "Appium, an Open-source, Cross-platform Test Automation Tool for Native, Hybrid, and Mobile Web and Desktop Apps," online available at https://github.com/appium/appium, 2022.