A Privacy Analysis of Google and Yandex Safe Browsing

Thomas Gerbet
Université Joseph Fourier, France
e-mail: thomas.gerbet@e.ujf-grenoble.fr

Amrit Kumar, Cédric Lauradoux
INRIA, France
e-mail: {amrit.kumar, cedric.lauradoux}@inria.fr

***Abstract*—GOOGLE and YANDEX Safe Browsing are popular services included in many web browsers to prevent users from visiting phishing or malware websites. If these services protect their users from losing private information, they also require that their servers receive browsing information on the very same users. In this paper, we analyze GOOGLE and YANDEX Safe Browsing services from a privacy perspective. We quantify the privacy provided by these services by analyzing the possibility of re-identifying URLs visited by a client. We thereby challenge GOOGLE's privacy policy which claims that GOOGLE cannot recover URLs visited by its users. Our analysis and experimental results show that GOOGLE and YANDEX Safe Browsing can potentially be used as a tool to track specific classes of individuals. Additionally, our investigations on the data currently included in GOOGLE and YANDEX Safe Browsing provides a concrete set of URLs/domains that can be re-identified without much effort.**

Keywords*—*Safe Browsing, Privacy, Tracking

I. INTRODUCTION

In 2008, GOOGLE started a service called GOOGLE Safe Browsing (GSB) [1] to warn and dissuade an end user from visiting phishing and malware web pages. With similar goals, YANDEX followed up with an identical service named YANDEX Safe Browsing (YSB). As of today, all major web browsers including Firefox, Internet Explorer, Safari, Opera and Yandex.Browser feature one of these Safe Browsing (SB) services. The integration of SB services into the browsers has naturally generated an extremely large user base. GSB alone accounts to a billion users until date [2].

The first version of GSB *aka* the `Lookup API`, raised serious privacy concerns: URLs were sent in clear to GOOGLE servers. GOOGLE could potentially capture the complete browsing history of GSB users. GOOGLE later came up with a new version that was conceived as a privacy-friendly service. The same architecture has been adopted by YANDEX for YSB. Essentially, in this new version, a GSB client computes a cryptographic digest of a given URL and checks if its 32-bit prefix matches a local database of malicious prefixes provided by GOOGLE. A database-miss ensures that the URL is safe. While, a hit requires the client to query the server by sending the prefix to eliminate false positives. From the server's reply, the client can eventually determine whether the URL is malicious or not. GOOGLE Chrome Privacy Notice [3] includes a section on GSB. It states that:

> "*Google cannot determine the real URL from this information* (read prefix)."

GOOGLE reiterates this statement in a document concerning the GSB usage in Mozilla Firefox [4]. These guarantees have allowed GSB to be massively used by the end users and even by other web service providers.

Apart from these statements, there is no other privacy analysis of GSB. Our goal is to provide an independent privacy analysis of GSB and its sibling YSB. To this end, we first motivate our work in Section II and then provide a comprehensive description of GSB and YSB in Section III and in Section IV respectively. In Section V, we present a threat model for re-identification and tracking. The SB services employ an anonymization technique that is a combination of *hashing* and *truncation*. Hashing is used to create pseudonyms for URLs. Generating pseudonyms (digests) of the URLs however does not suffice to anonymize the data, and hence truncation is applied to create collisions. Truncation of these pseudonyms ensures that several URLs share the same reduced pseudonym (prefix). We quantify the privacy provided by this solution using a *balls-into-bins* argument and k-anonymity [5] when a single prefix is sent to the servers for a URL (Section VI).

It appears that multiple prefixes can be sent to the servers for a given URL. Indeed, a client does not simply compute the digest of a target URL, rather decomposes the URL into several sub-URLs and computes the digests of all the decompositions (more details in Section III). In theory, distinct URLs can share common prefixes in their decompositions and hence may provide anonymity. We characterize the different cases for such multiple collisions in Section VII and collect their statistics on URLs using the *Common Crawl* dataset [6]. Our experimental analysis estimates the rate of such collisions and shows that hashing and truncation fails to prevent the re-identification of small-sized domains or certain URLs of larger domains. We further materialize this in the form of an algorithm that SB providers could potentially employ for tracking.

In Section VIII, we analyze the databases of malicious prefixes provided by GOOGLE and YANDEX. By crawling their databases, we detect a number of "suspicious" prefixes that we call *orphans*. Orphans trigger communication with the servers, but no full digest corresponds to them. We also observe several URLs which have multiples prefixes included in the blacklists. These provide concrete examples of URLs and domains that can be easily tracked by GOOGLE and YANDEX. We conclude this work with the related work and countermeasures (Section IX). We discuss and evaluate a privacy enhancing technique to reduce information leakage while maintaining the current architecture. Our proposal is efficient and can be directly integrated into SB clients.

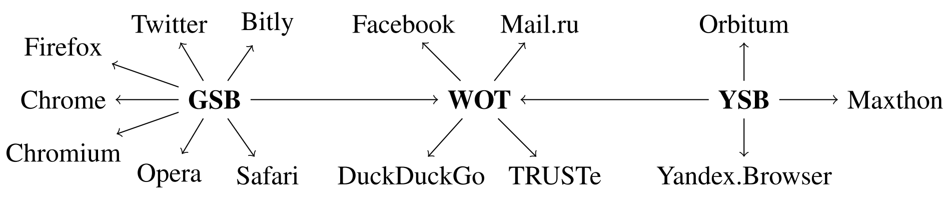

Fig. 1: The Safe Browsing ecosystem.

II. Motivations

Our choice to work on the privacy analysis of GSB and YSB is motivated by two factors. First, being that these are the only two SB services designed to be privacy friendly. The second is their high impact on users and other web services.

Indeed, many software vendors have proposed solutions for SB. Microsoft promotes *SmartScreen URL Filter* [7] and ships it with its products including Internet Explorer, Outlook, Exchange, Windows Live Mail and Entourage. Web of Trust (WOT) [8], and Norton Safe Web (NSW) [9], developed by Symantec are tools operating as a web browser plugin. Finally, McAfee SiteAdvisor [10] is a product very similar to NSW. In general, all SB services filter malicious links by relying on a dynamic blacklist of malicious URLs. However, all the afore-listed SB providers except Google and Yandex provide an SB service that is privacy-unfriendly by design: the URL or a part of it is sent in clear to the servers during the lookup in the blacklists. Furthermore, most of these SB providers agree in their privacy policies that they receive the URLs, but that they are not used to identify, contact, or target ads to users. This is the only guarantee that the users have. To our knowledge, Google and therefore Yandex Safe Browsing are the only SB services with built-in privacy features.

Our second motivation to work particularly on GSB stems from its popularity: it has been included in all major browsers namely Chrome, Chromium, Firefox, Safari and Opera. According to statcounter (statcounter.com), these represent 65% of all the browsers in use. Several other independent web services such as Twitter and Bitly also employ GSB to prevent users from disseminating malicious URLs. Facebook has developed a phishing and malware filter called *Link Shim* [11] that extensively relies on GSB and WOT. Fig. 1 schematically presents the major clients of GSB and YSB.

III. Google Safe Browsing

GSB aims to provide a comprehensive and timely detection of new threats on the Internet. According to a 2012 report [12], Google detects over 9500 new malicious websites everyday and provides warnings for about 300 thousand downloads per day. In the following, we present a comprehensive description of the GSB architecture.

A. Overview

The essential goal of GSB is to warn and dissuade an end user from visiting malicious URLs. The service is implemented at the application layer (HTTP level) of the standard Internet stack. Consequently, whenever a client (typically a browser) attempts to visit a malicious URL, the client can display an interstitial warning page before the suspicious web page is actually requested. In Fig. 2, we present a simplified architecture of the GSB service. Google crawlers harvest malicious URLs from the web and then transmit them to the GSB servers. Clients may then consult the server to check if a link is malicious.

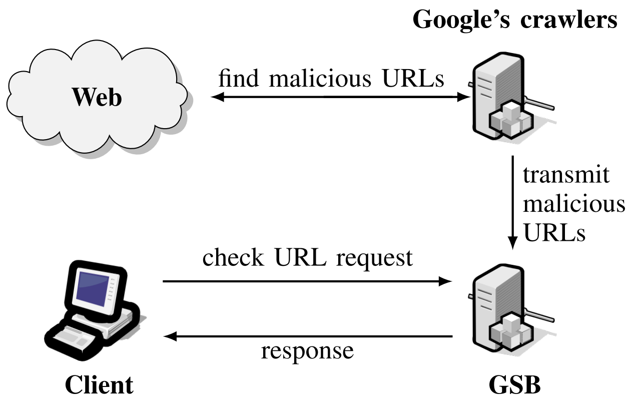

Fig. 2: High level overview of Google Safe Browsing.

GSB classifies malicious URLs into two main categories: malware and phishing. The blacklists and their number of entries are given in Table I. They contain SHA-256 digests of malicious URLs. The lists can either be downloaded partially to only update a local copy or can be obtained in its entirety.

TABLE I: Lists provided by Google Safe Browsing API. Information could not be obtained for cells marked with *.

List name	Description	#prefixes
goog-malware-shavar	malware	317,807
goog-regtest-shavar	test file	29,667
goog-unwanted-shavar	unwanted softw.	*
goog-whitedomain-shavar	unused	1
googpub-phish-shavar	phishing	312,621

GSB was first conceived in the form of a `Lookup API`. Using this API, a client could send the URL to check using an HTTP GET or POST request and the server performed a look up in the malicious lists. However, the API was soon declared deprecated for privacy and efficiency considerations. This was mainly because URLs were sent in clear to the servers and each request implied latency due to the network round-trip. To address these issues, Google currently offers another API: Google Safe Browsing API, described in the sequel.

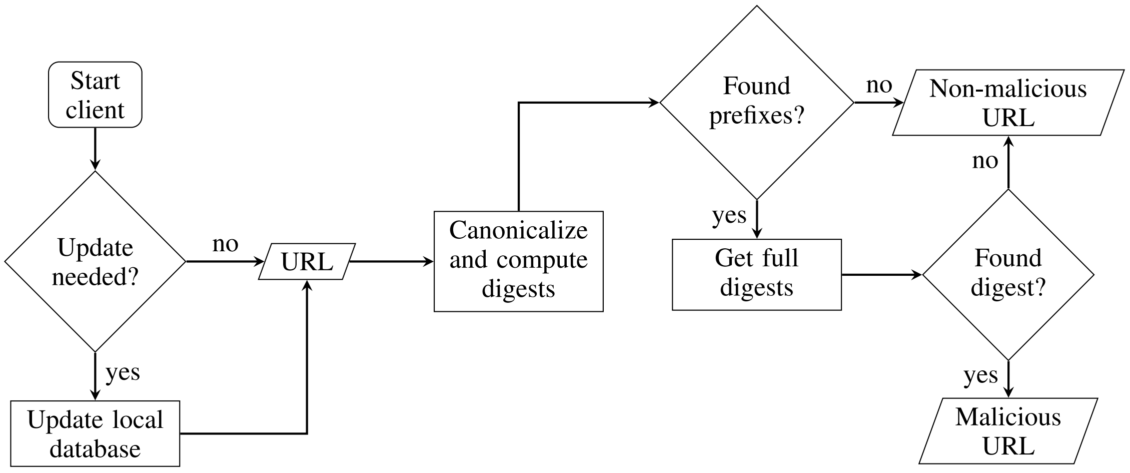

Fig. 3: GOOGLE Safe Browsing: Client's behavior flow chart.

B. *Safe Browsing API v3*

GOOGLE Safe Browsing API v3 is now the reference API. It has been positively received by the community as a major improvement for privacy. In contrast to the `Lookup API`, the client now does not handle a URL directly. Instead, the URL is canonicalized following the URI specifications [13] and then all the *decompositions* of the URL are generated. A decomposition is a URL composed of subdomains and subpaths of the target URL. For the sake of illustration, let us consider the most generic HTTP URL of the form `http://usr:pwd@a.b.c:port/1/2.ext?param=1#frags` (see [14]), where `usr` is a username and `pwd` is the corresponding password, `a.b.c` is a fully-qualified domain name, `port` is a TCP or UDP port number, `1/2.ext` is the URL path, `?param=1` is the query and `#frags` identifies a specific place within a remote resource. All the possible decompositions of the URL in the order they are generated are given below:

```
1  a.b.c/1/2.ext?param=1      5  b.c/1/2.ext?param=1
2  a.b.c/1/2.ext              6  b.c/1/2.ext
3  a.b.c/1/                   7  b.c/1/
4  a.b.c/                     8  b.c/
```

For each decomposition, the client computes a SHA-256 [15] digest. The digest is then checked against a locally stored database which contains 32-bit prefixes of malicious URL digests. If the prefix is not found to be present in the local database, then the URL can be considered safe. However, if there is a match, the queried URL may not necessarily be malicious: it can be a false positive. Consequently, the client must query the GSB server by sending the prefix. The server in response sends all the full digests corresponding to the received prefix. Finally, if the full digest of the client's prefix is not present in the list returned by the server, the URL can be considered safe. Fig 3 summarizes a request through the GSB API.

We note that client performs a lookup for decompositions in the given order. The lookup for all the decompositions is required since the complete URL might not have been included in the blacklists. If any of the decompositions creates a hit in the local database, then the initial link is considered as suspicious and the prefix can be forwarded to the GOOGLE server for a confirmation. If there are more than 1 hits, then all the corresponding prefixes are sent. After receiving the list of full digests corresponding to the suspected prefixes fragments, they are locally stored until an update discards them or when the client restarts, whichever is the earliest. Storing the full digests prevents the network from slowing down due to frequent requests. To maintain the quality of service and limit the amount of resources needed to run the API, GOOGLE has defined for each type of request (malware or phishing) the frequency of queries that clients must restrain to.

1) Local Data Structures: The choice of the data structure to store the prefixes on the client's side is constrained by two factors: fast query time and low memory footprint. GOOGLE has deployed two different data structures until now: *Bloom filters* [16] and *Delta-coded tables* [17]. In an earlier version of Chromium (discontinued since September 2012), a Bloom filter was used. This solution was abandoned to be replaced by delta-coded tables. Unlike classical Bloom filters, this data structure is dynamic, does not have any "intrinsic" false positive probability and yet incurs a lower memory footprint. However, its query time is slower than that of Bloom filters. Even though delta-coded tables do not entail any intrinsic false positive probability, their use to store 32-bit prefixes still generates false positives. False positives arise since several URLs may share the same 32-bit prefix.

2) Safe Browsing Cookie: GSB has often been criticized ever since web browsers have started to use them (see [18]). This is essentially because when implemented inside web browsers, each request to the API also sends a cookie which identifies a client. The cookie sent by browsers is the same as the one used by other services provided by GOOGLE especially the social features such as the *+1 button*. To these criticisms, GOOGLE responded that the cookies were not used to track users but only to monitor the performance of the service on the server-side and to catch bugs (see [19]). Since they are needed by GOOGLE to operate the service, browsers cannot disable it. However, Chromium and Firefox have isolated the SB cookie from the others with the purpose of achieving maximum privacy (see [20]).

IV. Yandex Safe Browsing

YSB comes in the form of an API [21], and also as a security feature in its browser called *Yandex.Browser*. The Yandex Safe Browsing API is compatible with C#, Python and PHP and is a verbatim copy of the GSB API with the only difference that in addition to the phishing and the malware lists provided by Google, the YSB API also includes 17 other blacklists. Each of these lists contains malicious or unsafe links of a given category.

TABLE II: Yandex blacklists. Information could not be obtained for cells marked with *.

List name	Description	#prefixes
goog-malware-shavar	malware	283,211
goog-mobile-only-malware-shavar	mobile malware	2,107
goog-phish-shavar	phishing	31,593
ydx-adult-shavar	adult website	434
ydx-adult-testing-shavar	test file	535
ydx-imgs-shavar	malicious image	0
ydx-malware-shavar	malware	283,211
ydx-mitb-masks-shavar	man-in-the-browser	87
ydx-mobile-only-malware-shavar	malware	2,107
ydx-phish-shavar	phishing	31,593
ydx-porno-hosts-top-shavar	pornography	99,990
ydx-sms-fraud-shavar	sms fraud	10,609
ydx-test-shavar	test file	0
ydx-yellow-shavar	shocking content	209
ydx-yellow-testing-shavar	test file	370
ydx-badcrxids-digestvar	.crx file ids	*
ydx-badbin-digestvar	malicious binary	*
ydx-mitb-uids	man-in-the-browser android app UID	*
ydx-badcrxids-testing-digestvar	test file	*

Table II provides the name and description of the blacklists with the number of prefixes present in each. We highlight that the lists `goog-malware-shavar` and `ydx-malware-shavar` are identical. The same holds for `goog-mobile-only-malware-shavar` and `ydx-mobile-only-malware-shavar`, and for `goog-phish-shavar` and `ydx-phish-shavar` respectively. By comparing the `goog-malware-shavar` lists of Google and Yandex, we observed that there are only 36,547 prefixes in common. Similarly, the phishing lists `googpub-phish-shavar` and `goog-phish-shavar` of Google and Yandex respectively have only 195 prefixes in common. These anomalies exist because the lists might not be up-to-date on the Yandex server.

V. Threat Model

Clients using SB tools and especially web browsers are exposed to several privacy threats. We note that an honest-but-curious SB provider may attempt to reconstruct completely or partly the browsing history of a client from the data sent to the servers. As previously mentioned, this is the case for most SB tools. This is also why the `Lookup API` of GSB was rejected by the community and later discontinued. Another threat posed by SB services consists in revealing if a client has visited some selected web pages and not the full browsing history. By associating traits to pages, the ultimate goal of the SB provider is to detect users' behavior such as political opinions, sexual orientation or allegiance to terrorist groups.

These threats are viable only when Google and Yandex can be justified to be potentially malicious. To this end, we take the self-explanatory example of Mozilla Firefox. In fact, Firefox, a major client of GSB assumes that Google via GSB can indeed be malicious, and takes concrete measures to reduce information leakage to the servers (see Section IX for further details). Moreover, the threat persists even if one assumes that the SB vendors are honest. This is because GSB is open-source, hence any third party may clone it (as currently done by Yandex) and offer a similar service. If the third party is malicious, then it can abusively use GSB for tracking.

It is pertinent to mention that both Google and Yandex have other means to track users, for instance through search queries, search clicks, ad impression/clicks, social network trackers, etc. However, they can not use these tools against users who distrust them and rely on other technologies to navigate on the web, for instance, Firefox with DuckDuckGo search engine. But, since Firefox incorporates GSB, the privacy of such users is still under threat. In other words, SB services may prove to be a handy tweak for Google and Yandex to turn other browsers into a tracking tool of their own. Apparently, around 40% of users do not use Chrome and an equivalent number do not use Google search engine (source statcounter.com). Furthermore, in comparison with traditional tracking techniques the threat posed by SB tracking is more severe. The reason being that unlike these techniques, SB does not necessarily require the client to load the target web page.

We note that the capabilities of SB providers are often immense and that we cannot restrict our adversary to the honest-but-curious model. To reach their goal, we assume that the provider is willing to do anything including tampering with the database. In GSB and YSB, the providers may on their own initiative include values in the prefix database of the client to track them or they can be constrained by a third party (governmental agencies) to do so. Since, Google and Yandex have web indexing capabilities, we safely assume that they maintain the database of all web pages and URLs on the web. We further assume that the SB servers may aggregate requests for full hashes and exploit the temporal correlation between the queries. In the sequel, we analyze GSB and YSB for the afore-mentioned threats. To this end, we consider two cases depending on whether or not more than one prefix per URL is sent by the client to the server.

VI. Single Prefix Match

A simple way to understand GSB and YSB consists in considering them as a probabilistic test run by the client to filter malicious URLs. Whenever, the test executed by the browser is positive, Google or Yandex servers are contacted to remove any ambiguity. While a negative test leaks no information about the URL to Google and Yandex, a positive test sends prefix(es) of certain decompositions of the target URL. In this section, we analyze the privacy provided by GSB and YSB when their servers receive a single prefix for a URL. Studying this case allows us to know if a user's browsing history can be fully/partly constructed.

A. Privacy Metric

In order to exemplify the evoked privacy concern and define a privacy metric, let us consider the following CFP URL for a conference: https://conf.org/2016/cfp.html. Its decompositions are shown in Table III. We suppose that the first decomposition creates a hit in the local database, i.e., `0x24e04dde` is present in one of the blacklists.

TABLE III: Decompositions of a conference CFP URL.

URL	**32-bit prefix**
conf.org/2016/cfp.html	`0x24e04dde`
conf.org/2016/	`0xf9aef594`
conf.org/	`0xed37d926`

Let us consider a situation where a client visits a web page that creates a hit on the prefix `0x24e04dde`. Hence, it sends the prefix to the server. In order to determine if this prefix is enough for GOOGLE or YANDEX to re-identify the corresponding URL, we consider the following privacy metric. The metric is defined as the number of URLs which share a given prefix and is often referred to as the *anonymity set size* in the literature. This measures the uncertainty in re-identifying the URL from a prefix. The higher is the value of the metric, the more difficult is the re-identification and hence better is the privacy achieved. The metric yields a simple yet reliable method to analyze and quantify the information leakage through prefixes. Furthermore, the metric may be viewed as a k-anonymity argument [5] to support the fact that URLs are anonymized through hashing-and-truncation.

B. Analysis

In this section, we compute the afore-defined privacy metric. One may argue that there are infinite number of pre-images for a 32-bit prefix, hence the privacy metric that estimates the uncertainty in re-identification should be infinitely large. However, the crucial point here is that the total number of URLs on the web is finite and hence the privacy metric can at most be finitely small.

We estimate the privacy metric using the probabilistic model of *balls-into-bins*. In the SB context, prefixes represent the bins and URLs are considered as the balls. We are interested in the maximum and the average value that the privacy metric can take. These values respectively measure the worst-case and average-case uncertainty for re-identification. Let us suppose m to be the number of balls and n to be the number of bins. A simple computation shows that the average number of balls in any bin is $\frac{m}{n}$. We further note that, according to the result of Ercal-Ozkaya [22], for a constant $c > 1$, and $m \geq cn\log n$, the minimum number of balls in any bin is $\Theta\left(\frac{m}{n}\right)$. As a result, the minimum value of the metric should not deviate too much from the average value and hence to avoid redundancy, we do not consider it in our analysis. Finally, in order to compute the maximum, we use the following result from Raab and Steger [23].

Theorem 1 (Raab and Steger [23]*):* Let M be the random variable that counts the maximum number of balls into any bin. If we throw m balls independently and uniformly at random into $n = 2^\ell$ bins, then $\Pr[M > k_\alpha] = o(1)$ if $\alpha > 1$ and $\Pr[M > k_\alpha] = 1 - o(1)$ if $0 < \alpha < 1$, where:

$$k_\alpha = \begin{cases} \frac{\log n}{\log \frac{n\log n}{m}}\left(1 + \alpha\frac{\log^{(2)}\left(\frac{n\log n}{m}\right)}{\log\frac{n\log n}{m}}\right), & \text{if } \frac{n}{\mathrm{polylog}(n)} \leq m \ll n\log n, \\ (d_c - 1 - \alpha)\log n, & \text{if } m = c \cdot n\log n, \\ \frac{m}{n} + \alpha\sqrt{2\frac{m}{n}\log n}, & \text{if } n\log n \ll m \leq n \cdot \mathrm{polylog}(n), \\ \frac{m}{n} + \sqrt{\frac{2m\log n}{n}\left(1 - \frac{1}{\alpha}\frac{\log^{(2)} n}{2\log n}\right)}, & \text{if } m \gg n \cdot (\log n)^3. \end{cases}$$

The equation for computing d_c can be found in [23].

In 2008, GOOGLE [24] claimed to know 1 trillion unique URLs. Since then, GOOGLE has reported 30 trillion URLs in 2012 and 60 trillion in 2013. These data are summarized in the first two rows of Table IV. The table also presents the number of domain names recorded by VERISIGN [25].

Using the previous results and the provided Internet data, we compute the maximum and the average value of the metric for unique URLs and domain names. Results are provided for different prefix sizes in Table IV. When GSB was started in 2008, at most 443 URLs matched a given 32-bit prefix. It has increased over the years to reach 14,757 in 2013. Even in the average case, it is hard for GOOGLE and YANDEX to re-identify a URL from a single 32-bit prefix. The case of domain names is slightly different because the space of domain names is much smaller and its dynamic is far slower than the one of URLs. In the worst case, two domain names will collide to the same prefix. Domain names can hence be re-identified with high certainty. However, the server does not know if the received prefix corresponds to a domain name or to a URL.

TABLE IV: Max. and avg. values for URLs and domains with prefix size ℓ. 0^* represents a value close to 0.

	URLs (10^{12})			**domains** (10^6)		
Year	2008	2012	2013	2008	2012	2013
Number	1	30	60	177	252	271
ℓ (bits)	max, avg			max, avg		
16	$2^{28}, 2^{23}$	$2^{28}, 2^{28}$	$2^{29}, 2^{29}$	3101, 2700	4196, 3845	4498, 4135
32	443, 232	7541, 6984	14757, 13969	2, 0.04	3, 0.05	3, 0.06
64	$2, 0^*$	$2, 0^*$	$2, 0^*$	$1, 0^*$	$1, 0^*$	$1, 0^*$
96	$1, 0^*$	$1, 0^*$	$1, 0^*$	$1, 0^*$	$1, 0^*$	$1, 0^*$

Hence, a single prefix per URL does not allow the SB server to reconstruct the browsing history of the client. So far, the solution seems to be privacy preserving as long as the client only reveals a single prefix.

VII. MULTIPLE PREFIX MATCH

In this section, we analyze the case when the backend SB servers receive multiple prefixes. SB servers may receive multiple prefixes for three reasons. First, it could be the result of accidental hits in the local database. This may happen when several decompositions of a non-malicious URL create hits in the database. Second, the SB providers may aggregate prefixes received over time. This could mainly be done to exploit any temporal correlation between the queries. Third and most importantly, in certain cases an SB provider may be forced to include several prefixes for a target URL in the local database. This might be necessary when a domain has a subset of sub-domains and URL paths which host several malicious URLs. Then, the sub-domains and the paths can be blacklisted instead of including each malicious URL in the database. This approach saves memory footprint on the client's side. Consequently, whenever a client visits any of the malicious URLs, multiple prefixes are sent to the servers. We note that one could possibly include only the prefix of the domain to blacklist all its malicious sub-domains and paths. However, this approach also blacklists all non-malicious URLs on the domain. Whence, multiple prefixes are indispensable to prevent certain URLs from being flagged as malicious.

Multiple prefix match for a URL forces the client to send more information to the servers than in the case of a single prefix match. Clearly, the amount of information on the URL obtained by the server is proportional to the actual number of prefixes received or aggregated. In the sequel, we analyze whether multiple prefixes may allow GSB and YSB to re-identify the URL visited by a client.

In order to present a comprehensible privacy analysis, we henceforth consider the simplified case of 2 prefixes. The analysis for the case when the server receives more than 2 prefixes per URL follows in a straightforward manner.

A. Collisions on 2 Prefixes

As in the single prefix case, more than two distinct URLs may yield the same two prefixes. The larger is the number of such URLs, the more difficult is the re-identification. These URLs exist due to three possible types of collisions on 32-bit prefixes. In the first type (Type I), several distinct yet related URLs share common decompositions and these decompositions yield the shared prefixes. We note that 2 distinct URLs are related if they have common sub-domains. The second type of collisions (Type II) is due to distinct yet related URLs that share one decomposition and hence one common prefix, while the other common prefix is due to the collision on truncated digests. Finally, the last type of collisions (Type III) appears when completely unrelated URLs generate the same prefixes. The latter may occur again due to collisions on the truncated digests. In the following, by a Type I URL, we mean a URL that generates a Type I collision with a given URL. We similarly define Type II and Type III URLs for a given URL.

TABLE V: An example with different possible collisions.

		URL	**Decomposition**	**Prefixes**
	Target URL	a.b.c	a.b.c/	A
			b.c/	B
Coll. Type	Type I	g.a.b.c	g.a.b.c/	C
			a.b.c/	A
			b.c/	B
	Type II	g.b.c	g.b.c/	A
			b.c/	B
	Type III	d.e.f	d.e.f/	A
			e.f/	B

In order to illustrate the different possible collisions, we present a set of examples in Table V. We assume that the client visits the target URL `a.b.c` and hence the server receives the corresponding two prefixes, say A and B. The server using these prefixes must determine the exact URL visited by the client. The next 3 URLs exemplify the different collisions.

Clearly, $\mathbb{P}[\text{Type I}] > \mathbb{P}[\text{Type II}] > \mathbb{P}[\text{Type III}]$, where $\mathbb{P}[X]$ denotes the probability of an event X. Under the uniformity assumption of hash functions, a Type III collision is highly unlikely, with a probability of $\frac{1}{2^{64}}$. We note that for Type I and Type II collisions to occur, the URLs must share at least one common decomposition. The probability of these collisions hence depends on the number of decompositions of URLs hosted on the domain. In general, the smaller is the number of decompositions per URL, the lower is the probability that Type I and Type II URLs exist. Moreover, a Type II URL exists only if the number of decompositions on a domain is larger than 2^{32}. We later show that no Type II URL exists by empirically estimating the distribution of decompositions over domains. As a result, the ambiguity in the re-identification can only arise due to Type I collisions. In the following, we discuss the problem of URL re-identification with a focus on URLs that admit Type I collisions.

B. URL Re-identification

We note that a target URL with few decompositions has a very low probability to yield Type I collisions, and hence it can be easily re-identified. In case of URLs with large number of decompositions, the server would require more than 2 prefixes per URL to remove the ambiguity. Nevertheless, the SB provider can still determine the common sub-domain visited by the client using only 2 prefixes. This information may often suffice to identify suspicious behavior when the domain in question pertains to specific traits such as pedophilia or terrorism. It is pertinent to highlight that the SB service provided by WOT collects the domains visited by its clients [26]. Hence, in the scenario when GSB and YSB servers receive multiple prefixes for a URL, the privacy achieved is the same as that ensured by services such as WOT.

Now, let us further analyze the problem of re-identifying URLs for which Type I collisions occur. To this end, we consider an illustrative example of a domain `b.c` that hosts a URL `a.b.c/1` and its decompositions (see Table VI). We assume that these are the only URLs on the domain. The URL generates four decompositions. Two of these decompositions include the domain name `'a'` as a sub-domain while the remaining two do not. These decompositions yield prefixes say A, B, C and D respectively. We note that the considered URL is only a slightly simplified form of the most general URL, where the query part of the URL has been removed [14]. Therefore, it largely represents all the canonicalized URLs.

TABLE VI: A sample URL on `b.c` with its 4 decompositions.

URL	**Decompositions**	**Prefix**
a.b.c/1	a.b.c/1	A
	a.b.c/	B
	b.c/1	C
	b.c/	D

We analyze the following three cases depending on the prefixes sent by the client to the SB server:

- **Case 1.** (A, B) generate hits: If the server receives these prefixes, it can be sure that the client has visited the URL that corresponds to the first prefix A, i.e, `a.b.c/1`. The probability that the re-identification fails is $\mathbb{P}[\text{Type III}] = \frac{1}{2^{64}}$. This holds because we assume that the domain `b.c` hosts only 4 URLs, hence the probability that the re-identification fails is the same as the probability of finding a Type III URL for prefixes (A, B).
- **Case 2.** (C, D) generate hits: In this case, the possible URLs that the client could have visited are: `a.b.c/1`,

a.b.c/ or b.c/1. These URLs correspond to prefixes A, B and C respectively. Hence, in order to remove the ambiguity and re-identify the exact URL visited by the client, the SB provider would include additional prefixes in the local database. If it includes the prefix A, in addition to C and D, then it can learn whether the client visited the URL a.b.c/1 or b.c/1. More precisely, if the client visits a.b.c/1 then prefixes A, C and D will be sent to the server, while if the client visits b.c/1, then only C and D will be sent. Similarly, in order to distinguish whether the client visits a.b.c/ or b.c/, the SB provider would additionally include the prefix B.

- **Case 3.** One of $\{A, B\} \times \{C, D\}$ generates hits: If the prefix A creates a hit, then the visited URL is a.b.c/1, while if the prefix B creates a hit then the client has either visited a.b.c/1 or a.b.c/. As in the previous case, the re-identification requires the SB provider to include an additional prefix A in the prefix database.

As a general rule, all the decompositions that appear before the first prefix are possible candidates for re-identification. Consequently, lower-level domain names and URL paths can be re-identified with a higher certainty than the ones at a higher level. To this end, we consider the case of *leaf* URLs on a domain. We call a URL on a given domain a *leaf*, if it does not belong to the set of decompositions of any other URL hosted on the domain. A leaf URL can also be identified as a leaf node in the domain hierarchy (see Fig. 4). Type I collisions for these URLs can be easily eliminated during re-identification with the help of only two prefixes. The first prefix corresponds to that of the URL itself, while the other one may arbitrarily correspond to any of its decompositions. In the example of Table VI, the URL a.b.c/1 is a leaf URL on the domain b.c, and hence it can be re-identified using prefixes (A, B).

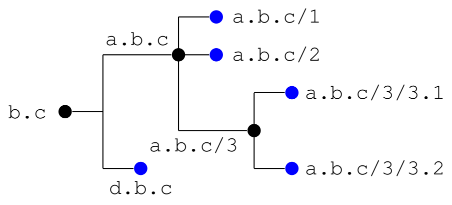

Fig. 4: A sample domain hierarchy. Blue nodes are leaf URLs.

Clearly, only non-leaf URLs contribute to Type I collisions. Hence, in order to re-identify non-leaf nodes, one must include more than 2 prefixes per node. Our observations further raise the question of the distribution of decompositions over domains that we explore in the following section.

C. Statistics on Decompositions

In the following, we support our analysis of the previous section using extensive experiments to estimate the distribution of URLs, decompositions and collisions over domains.

Our experiments have been performed on the web corpus provided by *Common Crawl* [6]. Common Crawl is an open repository of web crawl data collected over the last 7 years. It contains raw web page data, metadata and text extractions. For our experiments, we have used the most recent corpus of April 2015. This crawl archive is over 168 TB in size and holds more than 2.11 billion web pages. We note that the data included in Common Crawl is not exact. For instance, the maximum number of URLs hosted on a domain is of the order of 10^5. However, there are several domains such as wikipedia.org which host over a billion URLs. This peak is due to the fact that crawlers do not systematically collect more pages per site than this bound due to limitations imposed by the server. One may also find small-sized domains for which the crawl archive does not include all web page data, in particular all the URLs on the domain. Despite this bias, the corpus allows us to obtain a global estimate of the size of the web and determine the distribution of URLs and decompositions.

It is worth noticing that popular domains often host more URLs than the non-popular ones. This generally implies that the number of unique URL decompositions and eventual collisions on popular domains are larger than those on random/non-popular ones. We hence consider two datasets in our experiments. Our first dataset contains web pages on the 1 million most popular domains of Alexa [27]. We also collected 1 million random domains from Common crawl and then recovered web pages hosted on these domains. This forms our second dataset. The number of URLs and the total number of decompositions provided by these datasets is given in Table VII. Our dataset on popular domains contains around 1.2 billion URLs, while the one on random domains includes over 427 million URLs. URLs in the Alexa dataset yield around 1.4 billion unique decompositions in total, while the random dataset generates around 1 billion decompositions.

TABLE VII: Our datasets.

Dataset	#Domains	#URLs	#Decompositions
Alexa	1,000,000	1,164,781,417	1,398,540,752
Random	1,000,000	427,675,207	1,020,641,929

Fig. 5a presents the number of URLs hosted on the domains belonging to these two datasets. Clearly, the Alexa domains host larger number of URLs than the random domains. The most number of URLs hosted by a domain from either of the datasets is around 2.7×10^5. We also note that around 61% of the domains in the random dataset are single page domains. Fig. 5b presents the cumulative fraction of URLs for the two datasets. Our results show that for the Alexa dataset, only 19,000 domains cover 80% of all the URLs while, for the random dataset, only 10,000 domains span the same percentage of URLs. These results give strong empirical evidence (due to a large and random dataset) to previous results by Huberman and Adamic [28] that demonstrate that the number of web pages per site is distributed according to a power law. This implies that on a log-log scale the number of pages per site should fall on a straight line. For the random dataset, we fit a power-law distribution of the form:

$$p(x) = \frac{\alpha - 1}{x_{\text{min}}} \left(\frac{x}{x_{\text{min}}} \right)^{-\alpha},$$

where, $x_{\text{min}} = 1$ and the parameter α is estimated as $\hat{\alpha}$:

$$\hat{\alpha} = 1 + n \left(\sum_{i=1}^{n} \ln \frac{x_i}{x_{\text{min}}} \right)^{-1} = 1.312,$$

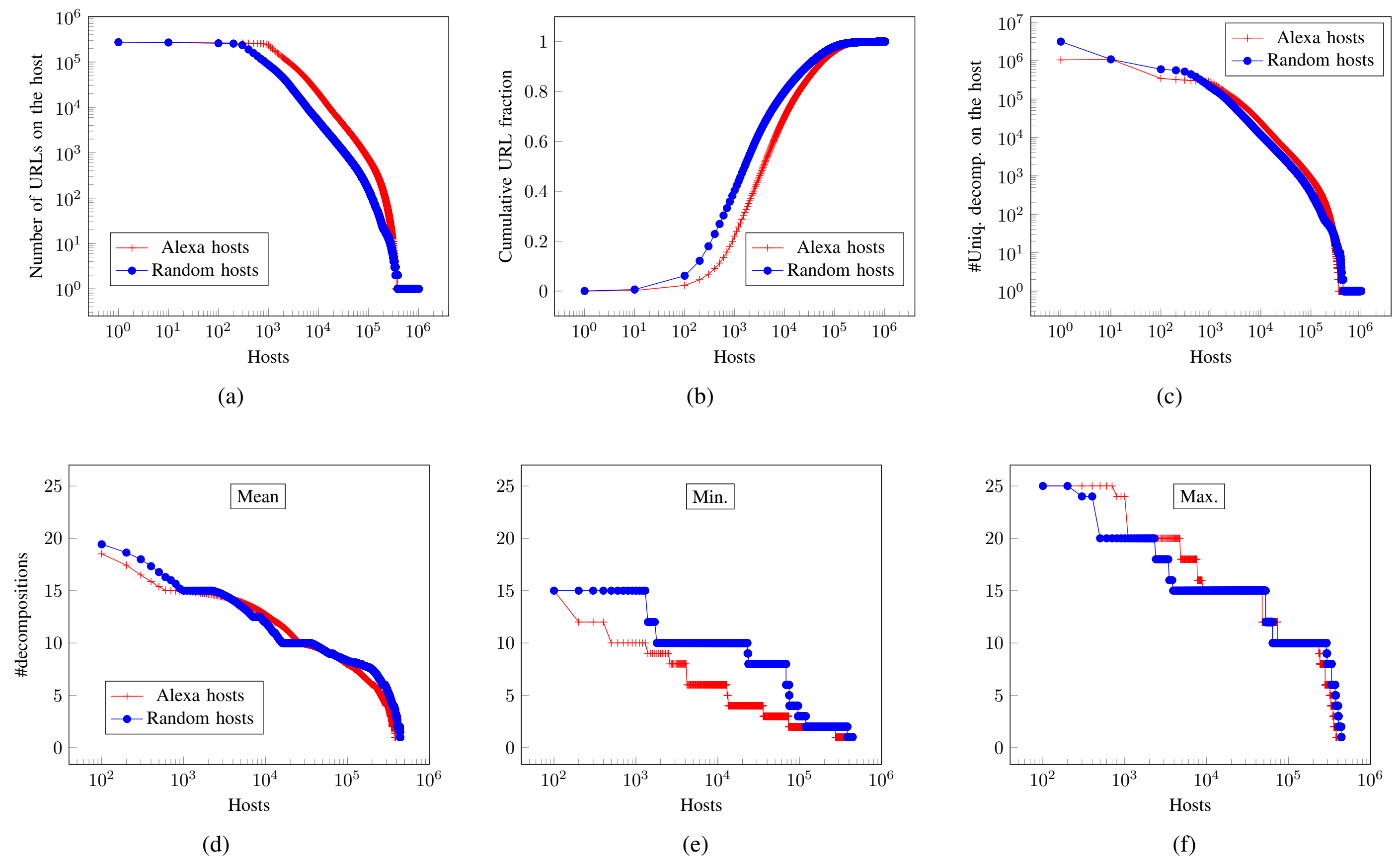

Fig. 5: Distribution of URLs and decompositions on hosts from the two datasets. Figure (a) presents the distribution of URLs over hosts, while (b) presents its cumulative distribution. (c) shows the distribution of decompositions over hosts. (d), (e) and (f) present the mean, minimum and maximum number of URL decompositions on the hosts.

where $\{x_i\}$ are the n data points. The standard error of the estimate is given by: $\sigma = \frac{\hat{\alpha}-1}{\sqrt{n}} = 0.0004$.

Fig. 5c presents the number of unique decompositions per host for the two datasets. These numbers are very close to that of the number of URLs and hence verify a similar power law distribution. The domains that cover the majority of the decompositions contain URLs that are difficult to re-identify (due to Type I collisions). While, the domains representing the tail of the distribution provide URLs which can be re-identified with high certainty using only a few prefixes.

In Fig. 5d,5e,5f, we present the mean, minimum and maximum number of decompositions of URLs per domain in the two datasets. We observe that 51% of the random domains present a maximum of 10 decompositions for a URL, while the same is true for 41% of the Alexa domains. The minimum value on the other hand is higher for random domains: 71% of the Alexa domains have a minimum of 2 per URL, while the fraction is up to 86% in case of random domains. The average number of decompositions for over 46% of the hosts from the two datasets lies in the interval $[1, 5]$. Hence, a URL on these hosts can generate on an average a maximum of $\binom{5}{2} = 10$ Type I collisions on two prefixes. As a result, URLs on these hosts can be re-identified using only a few prefixes.

We now consider Type II collisions. A Type II collision occurs on URLs that share common decompositions. Thus, in order for a Type II collision to occur, the number of decompositions per domain must be at least 2^{32}. However, the maximum number of decompositions per domain from either of the datasets is of the order of 10^7 (see Fig. 5c), which is smaller that 2^{32}. This implies that Type II collisions do not occur for any of the hosts in our datasets.

As for Type I collisions, we observed that the number of collisions found was proportional to the number of unique decompositions on the host. On the one hand, we found several domains from both the datasets for which the number of such collisions was as high as 1 million, while on the other hand, we found over a hundred thousand domains for which the number of such collisions was less than 20. Hence, many of the non-leaf URLs on these hosts can be re-identified by inserting as less as 3 prefixes per URL. Interestingly, we observed that 56% of the domains in the random dataset do not have Type I collisions, while the same is true for around 60% of the domains from the Alexa dataset. Consequently, URLs on these domains can be easily re-identified using only 2 prefixes.

D. A Tracking System based on SB

Our analysis shows that it is possible to re-identify certain URLs whenever multiple prefixes corresponding to them are sent to the servers. Relying on this fact, GOOGLE and YANDEX could potentially build a tracking system based on GSB and YSB. The robustness of the system would depend on the maximum number of prefixes per URL that they choose to include in the client's database. In the following, we denote this parameter by δ. Clearly, the larger is the δ, the more robust is the tracking tool. We note that its value is however chosen according to the memory constraints on the client's side.

Algorithm 1: Prefixes to track a URL

```
Data: link: a URL to be tracked and a bound $\delta$: max.
      #prefixes to be included.
Result: A list track-prefixes of prefixes to be included.
1  decomps, track-prefixes, type1-coll ← [ ]
2  dom ← get_domain(link)
3  urls ← get_urls(dom)
4  for url ∈ urls do
5      decomps ← decomps ∪ get_decomps(url)
6  if |decomps| ≤ 2 then
7      for decomp ∈ decomps do
8          track-prefixes ← track-prefixes ∪
            32-prefix(SHA-256(decomp))
9  else
10     type1-coll ← get_type1_coll(link)
11     common-prefixes ←
        32-prefix(SHA-256(dom)) ∪
        32-prefix(SHA-256(link))
12     if link is a leaf or |type1-coll| == 0 then
13         track-prefixes ← common-prefixes
14     else
15         if |type1-coll| ≤ $\delta$ then
16             track-prefixes ← common-prefixes
17             for type1-link ∈ type1-coll do
18                 track-prefixes ←
                    32-prefix(SHA-256(type1-link))
                    ∪ track-prefixes
19         else
20             track-prefixes ← common-prefixes
               /* Only SLD can be tracked.  */
```

The tracking system would essentially work as follows. First, GOOGLE and YANDEX choose a $\delta \geq 2$, and build a shadow database of prefixes corresponding to at most δ decompositions of the targeted URLs. Second, they push those prefixes in the client's database. GOOGLE and YANDEX can identify individuals (using the SB cookie) each time their servers receive a query with at least two prefixes present in the shadow database.

The crucial point to address is how they may choose the prefixes for a given target URL. In Algorithm 1, we present a simple procedure to obtain these prefixes for a target URL given a bound δ. The algorithm first identifies the domain that hosts the URL, which in most cases will be a Second-Level Domain (SLD) (see Line 2). Using their indexing capabilities, the SB providers then recover all URLs hosted on the domain and then obtain the set of unique decompositions of the URLs (Line 3-5). Now, if the number of such decompositions is less than 3, then the prefixes to be included for the target URL are those corresponding to these decompositions (Line 6-8). Otherwise, the algorithm determines the number of Type I collisions on the URL (Line 9-20). If no Type I collision exists or if the URL represents a leaf, then only two prefixes suffice to re-identify the URL. One of these prefixes is chosen to be the prefix of the URL itself and the other one can be that of any arbitrary decomposition. We however choose to include the prefix of the domain itself (Line 11-13). If the number of Type I collisions is non-zero but less than or equal to δ, then the prefixes of these Type I URLs are also included (Line 15-18). Finally, if the number of Type I collisions is greater than δ, then the URL cannot be precisely tracked. Nevertheless, including the prefix of the URL and that of its domain allows to re-identify the SLD with precision (Line 19-20). We note that if the prefixes are inserted according to this algorithm, the probability that the re-identification fails is $\left(\frac{1}{2^{32}}\right)^{\delta}$.

To illustrate the algorithm, let us consider the following CFP URL for a conference named PETS: petsymposium.org/2016/cfp.php. We first assume that the SB providers wish to identify participants interested in submitting a paper. Hence, the target URL is the CFP URL. The URL has 3 decompositions similar to the ones given in Table III. Since the target URL is a leaf, prefixes for the first and last decompositions would suffice to track a client visiting the target URL. Now, let us consider the case when the SB provider wishes to track a client's visit on petsymposium.org/2016 web page. The target URL yields Type I collisions with: petsymposium.org/2016/links.php and petsymposium.org/2016/faqs.php. Thus, the SB provider would include the prefixes corresponding to these 2 URLs, that of petsymposium.org/2016 and of the domain petsymposium.org/. In total, only 4 prefixes suffice in this case. Now, whenever the SB server receives the last two prefixes, then it learns that the client has visited the target URL. Additionally, this allows the server to track the other 2 URLs that create Type I collisions. The same is possible for the DSN CFP URL, however the SB server would need to include many more prefixes as the SLD of the DSN CFP page has over 50 Type I URLs.

It is also possible to re-identify a URL by aggregating requests sent by the client. This can be achieved by exploiting the temporal correlation between the queries of a user. YANDEX and GOOGLE can identify the requests of a given user thanks to the SB cookie. A user visiting: petsymposium.org/2016/cfp.php (with prefix `0xe70ee6d1`) is very likely to visit the submission website: petsymposium.org/2016/submission/(with prefix `0x716703db`). Instead of looking at a single query, the SB server now needs to correlate two queries. A user making two queries for the prefixes `0xe70ee6d1` and `0x716703db` in a short period of time is planning to submit a paper.

VIII. BLACKLIST ANALYSIS

In this section, we present an analysis of the blacklists provided by GOOGLE and YANDEX. Our analysis has the objective to identify URLs that match multiple prefixes. These URLs hence provide concrete examples of URLs/domains that can be tracked by the SB providers.

A. *Inverting the Digests*

As a first step in our analysis, we recover the prefix lists of GOOGLE and YANDEX. We then use these lists to query their servers using their respective APIs. This allows us to obtain the lists of full digests. Our second step is an attempt to identify the URLs which correspond to these prefixes. To this end, we harvested phishing and malware URLs, domains, and IP addresses from several sources and tested for their belonging to the blacklists of prefixes. The list of all our sources can be found in [29]. We also harvested 1,240,300 malware URLs, 151,331 phishing URLs and 2,488,828 URLs of other categories from BigBlackList [30]. Lastly, we obtained 106,923,807 SLDs from the DNS Census 2013 project [31]. The project provides a public dataset of registered domains and DNS records gathered in the years 2012-2013. We included the last dataset to determine the percentage of prefixes in the local database that correspond to SLDs. A description of all the datasets employed in our analysis is given in Table VIII.

TABLE VIII: Dataset used for inverting 32-bit prefixes.

Dataset	**Description**	#entries
Malware list	malware	1,240,300
Phishing list	phishing	151,331
BigBlackList	malw., phish., porno, others	2,488,828
DNS Census-13	second-level domains	106,923,807

The results of our experiments are shown in Table IX. We observe that reconstruction for GOOGLE prefixes using Malware, Phishing and BigBlackList datasets is inconclusive: 5.9% for malwares and 0.1% for phishing websites. For YANDEX, the situation is better but the majority of the database still remains unknown. However, the DNS Census-13 dataset produces a much larger reconstruction for all the lists except that of the phishing database. The rate is as high as 55% for YANDEX files. We highlight that phishing domains are short-lived and since the DNS Census-13 dataset dates back to 2013, the result of the reconstruction for phishing lists is very limited, only 2.5% for GOOGLE and 5.6% for YANDEX. Nevertheless, these results demonstrate that 20% of the GOOGLE malware list represents SLDs, while 31% of the prefixes in the YANDEX malware lists correspond to SLDs. Relying on our analysis of the previous sections, we may conclude that these prefixes can be re-identified with very high certainty.

It is pertinent to compare the result of our reconstruction with a similar attempt with another list in the past. German censorship federal agency called BPjM maintains a secret list of about 3,000 URLs believed to be unsuitable for women and children. The list is anonymized and distributed in the form of MD5 or SHA-1 hashes as the "BPJM-Modul" [32]. Though similar to the lists handled by GOOGLE and YANDEX, hackers have been able to retrieve 99% of the cleartext entries. We have applied the same approach, yet the reconstruction rate obtained has not been equally high. This proves that in order to reconstruct the database in cleartext, one would need high crawling capabilities and hence it is not achievable for general users. Furthermore, unlike the BPjM list, the blacklists provided by GSB and YSB are extremely dynamic. This requires a user to regularly crawl web pages on the web, which renders the reconstruction even more difficult.

B. Orphan Prefixes

We now look for *orphan prefixes* in GSB and YSB. An entry in the prefix list is called an *orphan* if no 256-bit digest matches it in the corresponding list of full digests. We also look for URLs in the Alexa list which generate an orphan prefix. Table X presents the results of our findings. Both GOOGLE and YANDEX have orphans. While GOOGLE has 159 orphan prefixes, for YANDEX the numbers are astonishingly high. 43% for `ydx-adult-shavar`, 99% for `ydx-phish-shavar`, 95% for `ydx-sms-fraud-shavar` and 100% of the prefixes in `ydx-mitb-masks-shavar` and `ydx-yellow-shavar` are orphans. We did not find any URL in the Alexa list matching a GOOGLE orphan prefix. But there are 660 URLs with one parent: the prefix matches one full digest. For YANDEX, we found 271 URLs matching an orphan prefix and 20,220 URLs with one parent.

The presence of orphan prefixes is very difficult to justify. Moreover, the behavior of a browser on these prefixes is not consistent. Some of the orphan prefixes are considered as false positives by YANDEX while others are declared as true positives. There are three possible explanations to argue the presence of orphans. First, that there is an inconsistency between the prefix lists and the corresponding lists of full digests. This could be due to a misconfiguration, latency in the update or the result of a development error. This particularly might hold for GOOGLE since very few prefixes are orphans. Second, that the services have intentionally noised the database in order to mislead attackers who may try to re-identify URLs from the prefixes. The last argument being that these SB providers might have tampered with their prefixes' database. The presence of large number of orphans for YANDEX proves that it is possible to include any arbitrary prefix in the blacklists.

C. Presence of Multiple Prefixes

The inclusion of multiple prefixes for a URL is not a hypothetical situation. Instead, our experiments with the databases show that GOOGLE and YANDEX indeed include multiple prefixes for a URL. We employ the Alexa list and the BigBlackList as test vectors for our experiments. The Alexa list has been used in our experiments to determine if GOOGLE or YANDEX indulge in any abusive use of SB.

In case of BigBlackList, we found 103 URLs creating 2 hits in the YANDEX prefix lists. Moreover, we found one URL which creates 3 hits and another one which creates 4 hits. The results on the Alexa list are particularly interesting. We found 26 URLs on 2 domains that create 2 hits each in the malware list of GOOGLE. As for the phishing list, we found 1 URL that creates 2 hits. For YANDEX, we found 1352 such URLs distributed over 26 domains. 1158 of these URLs create hits in `ydx-malware-shavar` while the remaining 194 are hits in `ydx-porno-hosts-top-shavar`. We present a subset of these URLs in Table XI. The large number of such URLs is essentially due to Type I collisions. Nevertheless, these URLs are spread over several domains which shows that YANDEX actively includes several prefixes for a URL. This is however less evident for GOOGLE. We reiterate that the corresponding domains and in some cases even the URLs are re-identifiable. This allows YANDEX, for instance, to learn the nationality of a person by the version of xhamster.com he visits. The most interesting example is that of the domain teenslovehugecocks.com. The domain may allow YANDEX to identify pedophilic traits in a user through domain re-identification.

TABLE IX: Matches found with our datasets.

		#matches (%match)			
	list name	**Malware list**	**Phishing list**	**BigBlackList**	**DNS Census-13**
GOOGLE	goog-malware-shavar	18785 (5.9)	351 (0.1)	6208 (1.9)	63271 (20)
	googpub-phish-shavar	632 (0.2)	11155 (3.5)	816 (0.26)	7858 (2.5)
YANDEX	ydx-malware-shavar	44232 (15.6)	417 (0.1)	11288 (3.9)	88299 (31)
	ydx-adult-shavar	29 (6.6)	1 (0.2)	33 (7.6)	201 (46.3)
	ydx-mobile-only-malware-shavar	19 (0.9)	0 (0)	17 (0.8)	790 (37.5)
	ydx-phish-shavar	58 (0.1)	1568 (4.9)	153 (0.47)	1761 (5.6)
	ydx-mitb-masks-shavar	20 (22.9)	0 (0)	1 (1.1)	9 (10.3)
	ydx-porno-hosts-top-shavar	1682 (1.6)	220 (0.2)	11401 (11.40)	55775 (55.7)
	ydx-sms-fraud-shavar	66 (0.6)	1 (0.01)	22 (0.20)	1028 (9.7)
	ydx-yellow-shavar	43 (20)	1 (0.4)	8 (3.8)	76 (36.4)

TABLE X: Distribution of prefixes as the number of full hashes per prefix. Collision with the Alexa list is also given.

		#full hash per prefix				**#Coll. with Alexa list**			
	list name	0	1	2	Total	0	1	2	Total
GOOGLE	goog-malware-shavar	36	317759	12	317807	0	572	0	572
	googpub-phish-shavar	123	312494	4	312621	0	88	0	88
YANDEX	ydx-malware-shavar	4184	279015	12	283211	73	2614	0	2687
	ydx-adult-shavar	184	250	0	434	38	43	0	81
	ydx-mobile-only-malware-shavar	130	1977	0	2107	2	22	0	24
	ydx-phish-shavar	31325	268	0	31593	22	0	0	22
	ydx-mitb-masks-shavar	87	0	0	87	2	0	0	2
	ydx-porno-hosts-top-shavar	240	99750	0	99990	43	17541	0	17584
	ydx-sms-fraud-shavar	10162	447	0	10609	76	3	0	79
	ydx-yellow-shavar	209	0	0	209	15	0	0	15

TABLE XI: A subset of URLs from the Alexa list with multiple matching prefixes in the GOOGLE and YANDEX database.

	URL	**matching decomposition**	**prefix**
GOOGLE	`http://wps3b.17buddies.net/wp/cs_sub_7-2.pwf`	`17buddies.net/wp/cs_sub_7-2.pwf`	`0x18366658`
		`17buddies.net/wp/`	`0x77c1098b`
	`http://www.1001cartes.org/tag/emergency-issues`	`1001cartes.org/tag/emergency-issues`	`0xab5140c7`
		`1001cartes.org/tag/`	`0xc73e0d7b`
YANDEX	`http://fr.xhamster.com/user/video`	`fr.xhamster.com/`	`0xe4fdd86c`
		`xhamster.com/`	`0x3074e021`
	`http://nl.xhamster.com/user/video`	`nl.xhamster.com/`	`0xa95055ff`
		`xhamster.com/`	`0x3074e021`
	`http://m.wickedpictures.com/user/login`	`m.wickedpictures.com/`	`0x7ee8c0cc`
		`wickedpictures.com/`	`0xa7962038`
	`http://m.mofos.com/user/login`	`m.mofos.com/`	`0x6e961650`
		`mofos.com/`	`0x00354501`
	`http://mobile.teenslovehugecocks.com/user/join`	`mobile.teenslovehugecocks.com/`	`0x585667a5`
		`teenslovehugecocks.com/`	`0x92824b5c`

IX. RELATED WORK AND MITIGATIONS

To the best of our knowledge no prior work has studied SB services from a privacy perspective. Due to the purpose and similarity of the service, our work is strongly related to web-search privacy. Indeed, URLs visited by a user and searched keywords reveal extensive information (see [33]). Several solutions to improve web-search privacy can be applied to our case and most notably dummy requests (see [34] for a survey). This solution is currently deployed in Firefox. Each time Firefox makes a query to GSB, some dummy queries are also performed to hide the real one. The dummy requests are deterministically determined with respect to the real request to avoid differential analysis [35]. This countermeasure can improve the level of k-anonymity for a single prefix match. However, re-identification is still possible in the case of multiple prefix match because the probability that two given prefixes are included in the same request as dummies is negligible. Another possible countermeasure consists in redirecting full hash requests through an anonymizing proxy. The essential limitation here is that the client must trust the proxy. Apparently, some proxy services keep server logs of user activity that can be subpoenaed.

Fixing GSB and YSB to prevent any information leakage would ideally require *private information retrieval* [36]. However, none of the existing constructions can scale to the level of SB [37], [38]. Hence, to reduce the amount of information leakage, we propose to query the server *one-prefix-at-a-time*. When a URL has several decompositions matching in the prefixes' database, the prefix corresponding to the root node/decomposition is first queried. Meanwhile, the targeted URL is pre-fetched by the browser and crawled to find if it contains Type I URLs. If the answer from GOOGLE or YANDEX is positive, a warning message is displayed to the user. Otherwise, if Type I URLs exist, then the browser can query the server for the other prefixes. In this case, GOOGLE and YANDEX can only recover the domain but not the full URL. In case no Type I URLs exists, a message can be displayed to warn the user that the service may learn the URL he intends to visit.

In order to evaluate this countermeasure, we have developed a proof-of-concept implementation in Python using SCRAPY [39], a popular web crawler. Among other parameters, SCRAPY allows to configure the timeout to process DNS queries, using the parameter `DNS_TIMEOUT` and the waiting time to download, using `DOWNLOAD_TIMEOUT`. We set these parameters to 30s and measure the cost incurred for 100 random URLs. The tests were performed on a 64-bit processor laptop computer powered by an `Intel Core i7-4600U CPU` at `2.10GHz` with `4MB cache`, `8GB RAM` and running `Linux 3.13.0-36-generic`. In a sequential setting, fetching and processing of a web page took on an average of 0.17s (for pages which do not cause timeout). We however note that this extra processing incurs no overhead when done in parallel while the client makes a full hash request to the SB server.

We note that the crawler is configured to follow a restricted crawling strategy: it can only recover links on the target web page. As a consequence, the crawler may fail to find a Type I URL even when it actually exists. In our experiments, this strategy found Type I URLs in 90% of the cases. A thorough albeit costlier approach would consist in crawling the SLD of the target URL. This ensures that the crawler never errs. The same experiment with a complete SLD crawl required roughly 3 times more time. The underlying crawling strategy hence presents a trade-off between privacy and robustness achieved by the countermeasure.

X. Conclusion

Safe Browsing services are valuable tools to fight malware, phishing and other online frauds. Unlike other Safe Browsing vendors, GOOGLE and YANDEX have made sincere efforts to render their services as private as possible. However, the effect of their anonymization efforts has been largely unclear. We have quantified the privacy provided by these services and have shown that the underlying anonymization technique of hashing and truncation fails when the server receives multiple prefixes for certain classes of URLs.

Our observations on the YANDEX database and to a lesser extent on that of GOOGLE show that it is possible to tamper these databases. These instances could either be deliberate attempts or the results of development errors/misconfigurations or latency. Irrespective of the underlying cause, the service readily transforms into an invisible tracker that is embedded in several software solutions. As future work, we want to design a plugin for Firefox and Chrome to make users aware of the associated privacy issues.

Acknowledgment. This research was partially supported by the Labex PERSYVAL-LAB (ANR-11-LABX-0025-01) funded by the French program Investissement d'avenir.

References

[1] G. Inc., "Safe Browsing API," https://developers.google.com/safe-browsing/.

[2] ——, "Google Transparency Report," Google, Tech. Rep., June 2014, https://bit.ly/1A72tdQ.

[3] Google, "Google Chrome Privacy Notice," https://www.google.com/intl/en/chrome/browser/privacy/, November 2014.

[4] G. Inc., "Google Safe Browsing Service in Mozilla Firefox Version 3," http://bit.ly/1igzX4v, April 2012.

[5] L. Sweeney, "k-Anonymity: A Model for Protecting Privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, no. 5, pp. 557–570, 2002.

[6] "Common Crawl," Online, 2015, http://commoncrawl.org/.

[7] M. Corporation, "Microsoft Security Intelligence Report," Microsoft, Tech. Rep., December 2013, https://bit.ly/1qAfTgt.

[8] L. WOT Services, "Web of Trust," https://www.mywot.com.

[9] Symantec, "Norton Safe Web," https://safeweb.norton.com/.

[10] McAfee, "McAfee Site Advisor," http://www.siteadvisor.com/.

[11] F. Inc., "Link Shim," http://on.fb.me/1he2yEB.

[12] N. Provos, "Safe Browsing - Protecting Web Users for 5 Years and Counting," http://bit.ly/208ra6P.

[13] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986 (INTERNET STANDARD), Internet Engineering Task Force, Jan. 2005.

[14] T. Berners-Lee, L. Masinter, and M. McCahill, "Uniform Resource Locators (URL)," Internet Requests for Comments, RFC Editor, RFC 1738, December 1994, https://www.ietf.org/rfc/rfc1738.txt.

[15] National institute of standards and technology, "Secure Hash Standard (SHS)," National Institute of Standards & Technology, Tech. Rep. FIPS PUB 180-4, march 2012.

[16] B. H. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[17] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein, "Delta encoding in HTTP," Internet Requests for Comments, RFC Editor, RFC 3229, January 2002.

[18] P. D. Kennedy, "Google's Safe Browsing Service is Killing Your Privacy," Online, http://bit.ly/1P2EEMk.

[19] "Chromium bug," Online, http://bit.ly/1Ha458a.

[20] "Mozilla bug," https://bugzilla.mozilla.org/show_bug.cgi?id=368255.

[21] Yandex, "Yandex Safe Browsing," http://api.yandex.com/safebrowsing/.

[22] G. Ercal-Ozkaya, "Routing in Random Ad-Hoc Networkds: Provably Better than Worst-case," Ph.D. dissertation, 2008.

[23] M. Raab and A. Steger, ""Balls into Bins" - A Simple and Tight Analysis," in *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, ser. RANDOM '98. London,UK: Springer-Verlag, 1998, pp. 159–170.

[24] G. Inc., "We knew the web was big..." http://bit.ly/1P4jKwe.

[25] V. Inc., http://verisigninc.com/en_US/innovation/dnib/index.xhtml.

[26] L. WOT Services, "Privacy Policy," https://www.mywot.com/en/privacy.

[27] "Alexa 1M Global Sites," Online, 2015, http://bit.ly/1yhXcgL.

[28] B. A. Huberman and L. A. Adamic, "Internet: Growth dynamics of the World-Wide Web," *Nature*, vol. 401, no. 6749, pp. 131–131, 1999.

[29] Z. S. Corp, https://zeltser.com/malicious-ip-blocklists/.

[30] "BigBlackList," Online, http://urlblacklist.com/.

[31] "DNS Census," Online, 2013, https://dnscensus2013.neocities.org/.

[32] "BPJM Modul," Online, http://bpjmleak.neocities.org/.

[33] R. Jones, R. Kumar, B. Pang, and A. Tomkins, ""I know what you did last summer": query logs and user privacy," in *ACM Conference on Information and Knowledge Management, CIKM 2007, Lisbon, Portugal, November 6-10, 2007*. ACM, 2007, pp. 909–914.

[34] A. Gervais, R. Shokri, A. Singla, S. Capkun, and V. Lenders, "Quantifying Web-Search Privacy," in *ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*. ACM, 2014, pp. 966–977.

[35] D. Veditz, Personal Communication, 2015.

[36] I. Goldberg, "Improving the Robustness of Private Information Retrieval," in *IEEE Symposium on Security and Privacy, S&P '07*, 2007.

[37] R. Sion and B. Carbunar, "On the Practicality of Private Information Retrieval," in *Proceedings of the Network and Distributed System Security Symposium – NDSS 2007*. San Diego, CA, USA: The Internet Society, February 2007.

[38] F. Olumofin and I. Goldberg, "Revisiting the Computational Practicality of Private Information Retrieval," in *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2012.

[39] "Scrapy," http://scrapy.org/.