Safe Browsing Services: to Track, Censor and Protect

Thomas Gerbet, Amrit Kumar, Cédric Lauradoux

Safe Browsing Services: to Track, Censor and Protect

Thomas Gerbet, Amrit Kumar, Cédric Lauradoux

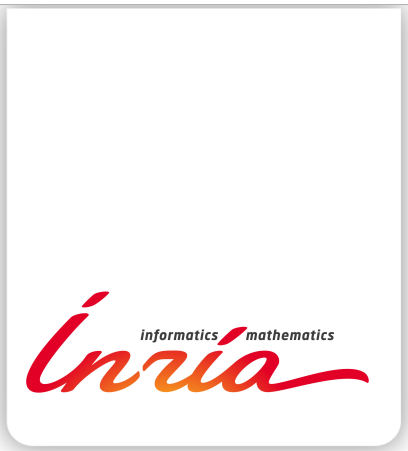
RESEARCH REPORT

N° 8686

February 2015

Project-Teams PRIVATICS

ISSN 0249-6399 ISRN INRIA/RR--8686--FR+ENG



Safe Browsing Services: to Track, Censor and Protect

Thomas Gerbet, Amrit Kumar, Cédric Lauradoux

Project-Teams PRIVATICS

Research Report n° 8686 — February 2015 — 25 pages

Abstract: Users often inadvertently click malicious phishing or malware website links, and in doing so they sacrifice secret information and sometimes even fully compromise their devices. The purveyors of these sites are highly motivated and hence construct intelligently scripted URLs to remain inconspicuous over the Internet. In light of the ever increasing number of such URLs, new ingenious strategies have been invented to detect them and inform the end user when he is tempted to access such a link. At the forefront lies the *Safe Browsing* technique, which queries a blacklist to identify unsafe websites and notify users and webmasters allowing them to protect themselves from harm.

In this paper, we show how Safe Browsing services work and how they damage the privacy of their users. We study the privacy of high impact Safe Browsing services including Google and Yandex to understand how much information they can collect. We hereby warn their users that these services can be easily used for surveillance to track specific classes of individuals or to censor certain websites. Google and Yandex are currently not accountable for the entries included in their blacklist. Until a major advance in the field of private information retrieval, Safe Browsing can not be made private and the current services can not be trusted. We provide evidences that Yandex has all probes in place to track the users of targeted websites.

Key-words: Malware, phishing, surveillance

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Safe Browsing Services: to Track, Censor and Protect

Résumé : Les utilisateurs cliquent parfois par inadvertance sur des sites de phishing ou de malware et ce faisant ils sacrifient des informations secrètes et parfois même compromettent leur sécurité. Les créateurs de ces sites sont très motivés et sont capables de construire des URLs intelligemment scriptées. Compte tenu du nombre sans cesse croissant de tels URLs, de nouvelles stratégies ont été inventées pour les détecter et informer l'utilisateur final quand il est tenté d'accéder à un tel lien. Au premier rang de ces techniques se trouve les services de Safe Browsing, qui interrogent une liste noire de sites dangereux et qui avertissent les utilisateurs leur permettant de se protéger.

Dans cet article, nous montrons comment les services Safe Browsing endommagent la vie privée de leurs utilisateurs. Nous étudions la vie privée de systèmes Google et Yandex pour comprendre quelles informations ils peuvent recueillir. Nous avertissons leurs utilisateurs que ces les services peuvent être facilement utilisés pour la surveillance de suivre des classes spécifiques d'individus ou de censurer certains sites. Google et Yandex ne sont pas actuellement responsables des entrées inclus dans leur liste noire. Jusqu'à une avancée majeure dans le domaine de la recherche d'informations privées, les services de Safe Browsing ne peuvent pas être respectueux de la vie privé. Nous fournissons des preuves que Yandex a toutes les sondes en place pour suivre les utilisateurs de sites Web ciblés.

Mots-clés : Malware, phishing, surveillance

1 Introduction

Communications purporting to be from popular social web sites, auction sites, banks, online payment processors or IT administrators are rampantly used to lure unsuspecting public. These attempts popularly known as *phishing*, aim to acquire sensitive information such as usernames, passwords, and credit card details by masquerading as a trustworthy entity. In general, these attacks rely on seemingly authentic URLs to entice a user to transmit sensitive data to the back-end malicious entities or to install harmful piece of code aka *malware* on their device.

These malicious URLs are often hard to be visually detected, whence the *raison d'être* of more reliable, automated and robust way of detecting them. *Safe Browsing* is such a service that enables browsers to check a URL against suspected phishing or malware pages' database. The browser can then warn the user about the potential danger of visiting the page. Essentially, a Safe Browsing enabled browser computes digests of the provided URL and checks if the 32-bit prefixes of the digests match a local database. If there is a match, a server is queried to ensure that it is not a false positive.

All the major browsers such as Chrome, Firefox, Internet Explorer, Safari, Opera and Yandex.Browser include such a service. Furthermore, Safe Browsing services are used by a huge number of users, for instance, Google claims more than a billion users of its Safe Browsing services until date [Inc14].

In this paper, we investigate Safe Browsing services and describe several privacy issues with Google and Yandex. To this end, we first provide a comprehensive description of Google and Yandex Safe Browsing. We then describe how these prefixes can leak information to the server and how Google and Yandex can turn and employ Safe Browsing for surveillance. We exhibit several URLs which can be re-identified by Yandex: Yandex can know if you prefer the adult sites `www.wickedpictures` or `www.mofos` because they both trigger Safe Browsing (false positive) at their login page and too many information are sent to the server. By crawling the database of Google and Yandex servers, we obtain a list of "suspicious" prefixes. These prefixes do not match any full digest on the server's side. They can be the result of an inconsistency between the server and the client or they can be prefixes inserted at the client side by the server to track specific URLs.

We even observe some very sensitive URLs blocked by Yandex. It would be very easy for Google and Yandex to censor a URL by including it Safe Browsing. Accidents have already occurred in the past with Safe Browsing services: some popular websites have been blocked by Google Safe Browsing by accident. It was the case for Bitly[1] in October 2014. For less popular sites, the situation is more complicated because it may take a while before somebody discovers the issue. This is the case for the website of the NGO *Secours Islamique* blocked by Yandex.Browser but not by regular Chromium, Firefox or Safari.

Our observations might be some artifacts/errors or might be evidence that Safe Browsing services are used deliberately for surveillance and censorship. We can not prove that Google or Yandex tamper with the blacklists of their Safe Browsing services on purpose. However, our observations, artifacts or not, shed light on risks of Safe Browsing has it is currently done. As a general conclusion on our observations, it seems that Yandex Safe Browsing accumulates all the possibles sins.

Our contribution can be summarizes as follows. We analyze Safe Browsing services and focus on the solution of Google and Yandex. We demonstrate that URLs can be re-identified despite the measures to anonymize them. As a consequence, we show that they can be turned into an out-of-control surveillance system. We support our claims by some analysis of the current databases. We also discuss a possible mitigation and censorship issues.

[1] `https://twitter.com/bitly/status/526015781533134848`

After this introduction, the paper is organized as follows. Section 2 provides a general overview of the Safe Browsing services. Then, we briefly describe in Section 3 all the major Safe Browsing mechanisms which are not privacy-friendly. It includes Microsoft SmartScreen filter among others. We give a detailed description of GOOGLE Safe Browsing and YANDEX Safe Browsing in Section 4 and Section 5 respectively. The main contribution of the paper is given in Section 6 and Section 7. These sections analyze the privacy implications of using Safe Browsing services and provide a *mitigation* mechanism to reduce the privacy risks. We give some related work in Section 8 and conclude the paper.

2 Safe Browsing: Overview

The essential goal of any Safe Browsing mechanism is to warn and dissuade an end user from accessing potentially malicious URLs. All the existing Safe Browsing services filter malicious links by relying either on a dynamic blacklist of malicious URLs, domains and IP addresses or a whitelist of *safe-to-navigate* websites. The dynamic nature of this list incorporates the fluctuating behavior of malicious domains: a *safe-to-navigate* domain transforming into a malicious one and vice versa. A *safe-to-navigate* website may turn into a malicious one when hackers inject malicious code into it. Conversely, a domain owner may clean his malicious webpage which eventually is reconsidered as non-malicious. Since Safe Browsing features are often included in the web browsers, the service is implemented in such a way that it does not considerably effect the browser usability. Hence, to ensure the robustness of the service, Safe browsing implements fast lookup data structures which may generate *false positives* : a non-malicious URL may get detected as a malicious one.

In addition to providing the basic warnings to the users, Safe Browsing services often provide webmaster tools. These tools allow a user to report malicious links unknown to the service. The submitted link upon analysis may get included in the blacklist. Symmetrically, a user or an administrator may explicitly ask the service to remove the malicious flag from a given domain.

GOOGLE first started this service in 2008 and has since proposed three different APIs. Since the inception of GOOGLE Safe Browsing, other web service providers have proposed similar solutions to detect malicious websites and inform the user about them. YANDEX [Yan] provides a GOOGLE-type API. YANDEX API has been used to implement the Safe Browser add-on developed by *Web of Trust* [WS]. Microsoft promotes *SmartScreen URL Filter* [Cor13] and ships it with its products: IE, Outlook.com, Outlook, Exchange, Windows Live Mail and Entourage. Norton Safe Web (NSW) [Symb], developed by Symantec is another tool operating as a web browser plugin, and requires Internet Explorer 6 or Firefox 3. Finally, McAfee SiteAdvisor [McAa] is another product very similar to NSW.

The afore-cited Safe Browsing services are the most popular ones and are proposed by established companies in the domain of security and web service. In the following sections, we investigate the afore-cited Safe Browsing services for privacy and security risks.

3 Privacy-unfriendly Services

In this section, we discuss briefly the Safe Browsing services which agree that they recover each URL visited by a user. These mechanisms by design are not privacy-friendly and can track each user and monitor his activities over the web. This includes MICROSOFT SmartScreen filter, Web of Trust (WOT), Norton Safe Web and McAfee SiteAdvisor.

3.1 Microsoft SmartScreen

SmartScreen filter [Mica] was first introduced as a malware and phishing filter in Internet Explorer (IE)8. It extends the phishing-only filter integrated in IE7 to include sites that offer malicious downloads or employ social engineering techniques to incite a user to download and run a malicious software. Microsoft claims to have effectively blocked 2 to 5 million attacks a day by socially engineered malwares and downloads for its IE8 and IE9 customers.

The SmartScreen feature is disabled by default. Unlike GOOGLE or YANDEX Safe Browsing, SmartScreen filter checks URLs against a whitelist of sites known not to be potential security risks. Essentially, SmartScreen is a reputation based system. The technique consists in a simple and direct *look-up* in a cloud-based database of potentially malicious links. Essentially, it requires the client to send the URL to the distant server and obtain a response for its presence in the database. In case of files downloaded using IE, the hash of the file (SHA256) is sent to the server hosting the database along with other information.

Since the release of Windows 8, SmartScreen has been extended to become a system feature of Windows, and hence is not solely restricted to IE. With the advent of IE9, a new feature has been added to SmartScreen: *SmartScreen Application Reputation*. It aims to protect users from unknown malware programs. This software reputation scheme is backboned by Authenticode signature [Micb, Micc].

Microsoft's Privacy webpage for IE mentions that: "*When you use SmartScreen Filter to check websites automatically or manually, the address of the website you are visiting will be sent to Microsoft with standard computer information and the SmartScreen Filter version number. Information that may be associated with the address, such as search terms or data that you entered in forms, might be included.*"

For files, MICROSOFT has a similar statement. Microsoft asserts that the information described above is only used to analyze performance and improve the quality of service. However, this is the only guarantee available to a user.

3.2 Web of Trust

Web of Trust (WOT) [WS] is a free browser addon released in 2007. The addon makes query to a reputation database to know whether or not a website is trustworthy. It uses several sources including Phishtank and YANDEX Safe Browsing. The addon automatically creates a random identifier for a user. At each query, this identifier, the date and time of the request and the host name for each site a user visits are sent to WOT servers to recover the rating information. Clearly, WOT can track a user.

3.3 Norton Safe Web

Norton Safe Web [Symb] abbreviated as NSW is developed by Symantec and delivers information about websites based on automated analysis and user feedback. The feature is included in its product Norton Internet Security and Norton 360. A limited, standalone version of Safe Web, known as Norton Safe Web Lite is also available as a freeware. Safe Web operates as a web browser plugin and requires IE6 or Firefox 3 or later or GOOGLE Chrome. Recently, Symantec has teamed up with Ask.com to provide on top of Safe Web, a search environment called Norton Safe Search (NSS) [Syma] which offers visual site ratings even within search results. Even when NSS is disabled, NSW continues to annotate searches on the default search engine.

Symantec asserts that during the search "*Ask.com collects information including: IP address, the origin of the search ... and may share this information with a third party*".

3.4 McAfee SiteAdvisor

McAfee SiteAdvisor (SA) [McAa] is plugin available for Windows (Firefox, Internet Explorer and Chrome) and for Mac OS (Firefox and Safari). It also provides a secure search box to block risky sites and provides one of these 4 ratings to each site, Safe, Caution, Warning, Unknown. An extended and paid version of the software comes with SA Live [McAc]. With SA Live software, even emails and instant messages get checked for malicious links. SA Live also scans downloaded files for threats. Depending on the chosen protection level, download protection will stop the download and warn users of the risks. Protection against malicious downloads is not available in the standard SA freeware.

SA employs a lookup mechanism to contact the rating server [McAb] and caches the rating in memory for a period of time which is based on a value from the rating server. The default is 30mins. SA also generates a log file which is regularly sent to ePolicy Orchestrator for reporting purposes. SA currently sends logs only through HTTP and hence can be easily eavesdropped.

As a conclusion, these services are not privacy friendly: the URL or part of it is sent in clear to the servers. To our knowledge, Google and Yandex are the only Safe Browsing services with built-in privacy features.

4 Google Safe Browsing

Google Safe Browsing (GSB) [sba] aims to provide a comprehensive and timely detection of new threats on the Internet. We highlight that the Google Safe Browsing service is not restricted to search, but it also extends to adds. It has further opened paths for new services such as instantaneous phishing and download protection, i.e., protection against malicious *drive-by-downloads*, chrome extension for malware scanning and Android application protection. According to a 2012 report [Pro], Google detects over $9,500$ new malicious websites everyday and provides warnings for about 300 thousand downloads per day.

Fig. 1 shows a simplified architecture of the Google Safe Browsing service. Google crawlers continuously harvest malicious URLs available on the web and then transmit them to the safe browsing server. Clients can then consult the server to check if a link is malicious.

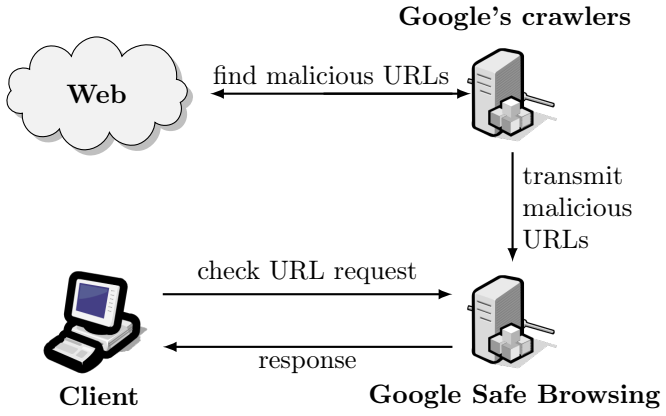


Figure 1: High level overview of Google Safe Browsing.

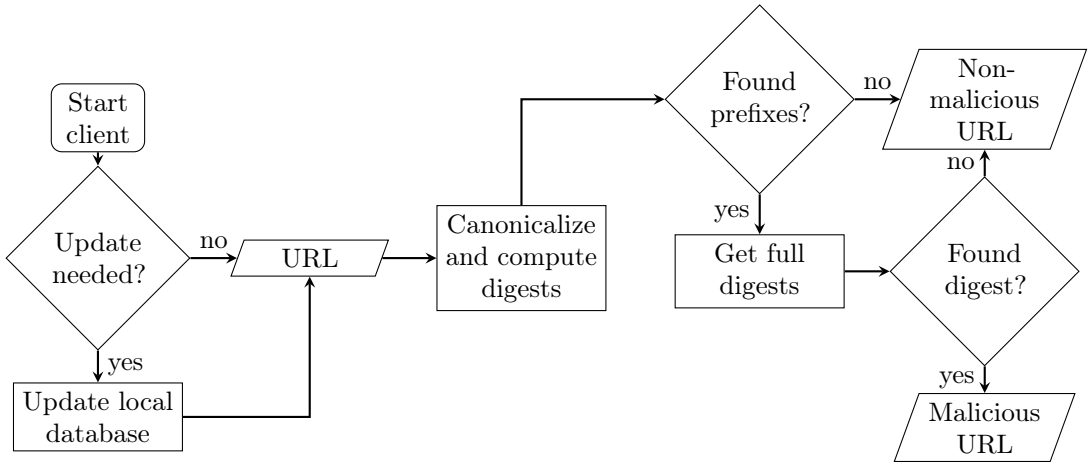


Figure 2: Google Safe Browsing: Client's behavior flow chart.

Google Safe Browsing classifies malicious URLs into two lists: malware and phishing. The names and the number of entries per list is given in Table 1. They contain hashes of malicious URLs and can either be downloaded partially to only update the local copy by adding new hashes and removing old ones or can be downloaded in its entirety. The lists can be accessed by clients using two different APIs, software developers choose the one they prefer according to the constraints they have. In the following, we describe these APIs.

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

Table 1: Lists used by Google Safe Browsing API.

4.1 Lookup API

Google Lookup API is a quite simple interface to query the state of a URL. Clients send URLs they need to check using HTTP GET or POST requests and the server's response contains a direct answer for each URL. The response is generated by looking up in the malicious lists stored on the distant server. This is straightforward and easy to implement for developers, but has drawbacks in terms of privacy and performance. Indeed, URLs are sent in clear to Google servers and each request implies latency due to a network round-trip. To solve these issues, Google offers another API: Google Safe Browsing API described below.

4.2 Safe Browsing API

Google Safe Browsing API is more complex than the Lookup API and consists in sharing a part of the processing with the client. This API was received positively by the community as a major improvement for the privacy. In contrast to the Lookup API, the client now does not handle a

		Data structure (MBytes)			
		Delta-coded table		**Bloom filter**	
Prefix size (bits)	**Raw data**	**size**	**comp. ratio**	**size**	**comp. ratio**
32	2.5	1.3	1.9		0.8
64	5.1	3.9	1.3		1.7
80	6.4	5.1	1.2	3	2.1
128	10.2	8.9	1.1		3.4
256	20.3	19.1	1.06		6.7

Table 2: Client cache for different prefix sizes.

URL directly. Instead, the URL is canonicalized following the URI specifications [BLFM05] and hashed with SHA-256 [Nat12]. Digest is then checked against a locally stored database which contains 32 bits 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 is not necessarily malicious: it can be a false positive. The client must query the remote Google service to get all the full digests of malicious URLs corresponding to the given prefix. Finally, if the full digest of the client is not present in the list returned by the server, the URL could be considered safe. We note that most of the malicious links are short-lived and hence the lists and the local database are regularly updated to incorporate this dynamic behavior. Fig. 2 summarizes a request to the Google Safe Browsing API from a client's point of view.

We note that the lookup requires the client to test several URLs corresponding to a given URL. This is because the complete URL might not have been included in the blacklists. For instance, in order to check whether `http://a.b.c/1/2.html?param=1` is malicious, the client will lookup for the following 8 strings:

```
a.b.c/1/2.html?param=1
a.b.c/1/2.html
a.b.c/
a.b.c/1/
b.c/1/2.html?param=1
b.c/1/2.html
b.c/
b.c/1/
```

If any of the above URLs creates a hit in the local database, then the initial link can safely be considered as malicious and can be forwarded to the Google server for a confirmation. After receiving the list of 256 bits digests corresponding to the suspected prefixes fragments, they are locally stored until an update discards them. Storing the full digests prevents the network from slowing down due to frequent requests. To maintain the quality of service and limiting the amount of resources needed to run the API, Google has defined for each type of requests (malware or phishing) the frequency of queries that clients must restrain to.

4.2.1 Data Format

The prefixes blacklist is provided by Google and each client has to store it locally. Each blacklist is comprised of a series of *chunks*. Chunks consists of two kinds: *add chunk* and *sub chunk*. A chunk is identified by an ID number and contains new digests to add to the list (aka add chunk) or ID numbers of older chunks which have to be discarded (aka sub chunk). When a client updates his database, he sends the lists of chunks' ID numbers that he already owns and gets back only the new chunks.

4.3 Local Data Structures

The choice of data structure to store the prefixes is constrained by two factors: fast query time and low memory footprint. Two different data structures have been employed by GOOGLE until now: *Bloom filters* [Blo70] and *Delta-coded tables* [MKD+02] described below.

In early versions of Chromium (discontinued since September 2012), a Bloom filter [Blo70] was used to store the prefixes' database on the client's side (more details on Bloom filters can be found in [BM05]). The Chromium development team used Bloom filter with 20 hash functions. The hash function used is the one proposed by Jenkins [Jen96]. They also set the minimum size of the Bloom filter to 25000 bytes and the maximum size to 3Mb. This solution was abandoned because the original Bloom filter is not dynamic: suppression can not be made.

The CHROMIUM development team has switched to another data structure in the more recent releases. Unlike the classical Bloom filter, this data structure called delta-coded table [MKD+02] is dynamic and yet incurs a low memory footprint. Indeed, memory usage is a critical point for web browsers, especially when they are used in memory constrained mobile environments. Furthermore, unlike Bloom filter, the current data structure, does not have any "intrinsic" false positive probability and uses less memory. However, its query time is slower than Bloom filter. The balance between the query time and the memory usage seems suitable with the operational constraints.

In case of delta-coded prefix table, the 32-bit prefixes are sorted, and the difference of the consecutive prefixes is computed. The final representation requires two arrays, storing prefix differences using 16 bits. An index handles the case when 16 bits can not encode a difference and provides a quicker random access. A maximum of 100 differences between each index is allowed.

Example 4.1 (Delta-Coded Prefix Table) *Consider the sorted prefixes:* $\{20, 42, 60000, 200000, 210000\}$. *Two vectors are considered: index vector and delta vector. To store the prefixes into the delta-coded table, first the difference set is computed i.e.* $diff = \{20, 22, 59958, 140000, 10000\}$. *This set includes the first prefix. A pair composed of the first prefix and the size of the initial delta vector i.e.* $(20, 0)$ *is then inserted into the index vector. Since, the subset* $\{22, 59958\}$ *of the prefixes can be encoded using less than* 16 *bits, the delta vector will now contain the list of these differences i.e.* $\{22, 59958\}$. *The following difference,* 140000 *however cannot be encoded using* 16 *bits, hence a new index is assigned, and the pair composed of the prefix and the current size of the delta vector* $(200000, 2)$ *is inserted in the index vector. Finally, the last delta* 10000 *is inserted in the delta vector since it can be encoded using 16 bits.*

Even though the delta-coded table does not entail false positives, its use to store 32 bit prefixes in CHROMIUM indeed leads to false positives. False positives arise due to the fact that several URLs may have the same 32 bit prefix.

We have re-implemented all the local data structure to understand why Bloom filters were abandoned and why GOOGLE has chosen 32-bit prefixes. The results of our coding are shown in Table 2. If 32 bit prefixes are stored, the raw database requires 2.5 MB of space. Storing these prefixes using delta-coded table would only require 1.3 MB of memory, hence GOOGLE achieves a compression ratio of 1.9. For the same raw data, Bloom filter would require 3 MB of space. This justifies GOOGLE's choice of delta-coded table over Bloom filter. However, starting from 80 bit prefixes, Bloom filter outperforms delta-coded table.

4.4 Safe Browsing Cookie

GOOGLE Safe Browsing services have been criticized since web browsers have started to use them. Indeed, when implemented inside web browsers each request to the Safe Browsing API also sends

a cookie which allows to identify a user. The cookie sent by the web browsers is the same as the one used with other services provided by GOOGLE especially the social features such as the *+1 button*.

To these criticism, GOOGLE responded that cookies were not used to track users but only to monitor the performance of the Safe Browsing services on the server-side and to catch bugs [2].

Since the cookie is needed by GOOGLE to operate the service, web browsers can not disable it. Chromium and Firefox have decided to isolate the Safe Browsing cookie from the others with the purpose of protecting users as much as possible [3].

5 Yandex Safe Browsing

YANDEX Safe Browsing (YSB) service comes in the form of an API [Yan] 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 GOOGLE Safe Browsing API with the only difference being that in addition to the phishing and malware lists proposed by GOOGLE, the API also includes 17 other blacklists. Each of these blacklists contains malicious or unsafe links of a given category. For instance, `ydx-sms-fraud-shavar` contains prefixes of links widely used in sms fraud. Similarly, `ydx-badbin-digestvar` corresponds to links to malicious binary code.

In Addition, YANDEX provides an API to check whether or not a URL corresponds to an adult website. It works exactly as the API for other blacklists and looks up for prefixes into two files: `ydx-porno-hosts-top-shavar` a list of prefixes corresponding to around 100,000 porn websites and `ydx-adult-shavar`, a list of 434 adult websites and adds.

Table 3 provides the name and description of the blacklists and the number of prefixes in each blacklist. In fact the lists `goog-malware-shavar` and `ydx-malware-shavar` are identical in YANDEX scheme. The same holds for `*-mobile-only-malware-shavar` and also for `*-phish-shavar`. By comparing the two `goog-malware-shavar` lists of GOOGLE and YANDEX, we observed that there are only 36547 prefixes in common. Similarly, for phishing lists `googpub-phish-shavar` and `goog-phish-shavar` of GOOGLE and YANDEX respectively have only 195 prefixes in common. These anomalies might be because the lists are not up-to-date on the YANDEX server.

6 Surveillance with GSB and YSB

As previously discussed, the GOOGLE Safe Browsing cookie has drawn severe criticisms in the past. In this section, we show that the privacy issues with the Safe Browsing mechanism are not restricted to the use of cookies. In fact, all the existing mechanisms create far deeper privacy concerns. Our analysis of the Safe Browsing mechanisms provides two other important privacy breaches. First, we identify how the URLs can be re-identified by GOOGLE and YANDEX using the received prefix(es) during the query, hence leaking a user's activity on the Web. Second, we explain how it is possible to track a user's visit to a targeted URL.

6.1 URL Re-identification

A simple way to understand GOOGLE and YANDEX Safe Browsing consists in considering them as a probabilistic test run by the browser to filter malicious URLs. As a probabilistic solution, they have true positives, false positives, etc. When the test executed by the browser is positive, GOOGLE or YANDEX are contacted to remove any ambiguity. In reply, the server sends the list of

[2] `https://code.google.com/p/chromium/issues/detail?id=103243`
[3] `https://bugzilla.mozilla.org/show_bug.cgi?id=368255`

Table 3: Yandex Blacklists. The 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	*

full hashes corresponding to the user's query (which may consist of more than 1 prefix): if there is a match, the browser proceeds in alerting the user. However, when the test is negative, the URL is safe (from the point of view of the service).

While a negative test leaks no information about the URL to Google and Yandex, a positive test sends prefix(es) of the decomposition of the URL to Google and Yandex. The exact number of prefixes sent depends on the number of hits in the local database. Table 4 summarizes all the cases with respect to two factors: existence in the prefix lists and if a connection with the distant server is required.

Event	**Existence**	**Connection**
True positive	Yes	Yes
True negative	No	No
False positive	Yes	Yes
False negative	No	No

Table 4: Events leaking information in GSB and YSB.

When the test in the local database is positive, the browser sends prefix(es) to Google or Yandex. It is important to know if those prefixes are enough to recover the URL, hence learning a user's activity. Let us consider the example of `https://petsymposium.org/2015/cfp.php`. Its decomposition is shown in Table 5.

Let us consider the situation where the second decomposition creates a hit in the local database, i.e, `0x7b45bc0c` is present in one of the prefix lists. To determine if the prefix `0x7b45bc0c` corresponding to the second decomposition is enough for Google or Yandex to recover the URL, we need to know the number of unique URLs (or a rough estimate) and the maximum number of URLs which share the same prefix. Our analysis is provided below.

We use a result from Raab and Steger [RS98] on *balls into bins* to evaluate the maximum

URL	32-bit prefix
`https://petsymposium.org/2015/cfp.php`	`0xbd27e944`
`https://petsymposium.org/2015/`	`0x7b45bc0c`
`https://petsymposium.org/`	`0xf34e4a27`

Table 5: Decomposition of a URL and its different prefixes.

number of URLs that can share the same ℓ bits prefix.

Theorem 1 (Raab and Steger [RS98]) *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:*

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

The equation for computing d_c can be found in [RS98].

If $n = 2^\ell$ is the number of prefixes and m is the number of unique URLs, then M is the maximum number of URLs matching a given prefix. M is also the maximum uncertainty (in terms of number of URLs) that Google or Yandex can have over re-identifying the URL for a received prefix. It can be viewed as a k-anonymity argument [Swe02] to support the fact that URLs are anonymized through hashing and truncation to ℓ bits. Actually, the correct argument from the user's point of view would be to consider the case of the bin having the least number of balls. However, we are taking the attacker's point of view: for Google and Yandex, M is the worst case for URLs re-identification.

In 2008, Google [goo] claimed to know 1 trillion unique URLs. Since then, Google officials have reported 30 Trillion in 2012 and 60 Trillion in 2013. These values are summarized in Table 6 in addition to the number of domain names provided by verisign in its reports [Inc].

Year	# unique URLs (Google)	# of domains
2008	1 Trillion	177 Million
2012	30 Trillion	252 Million
2013	60 Trillion	271 Million

Table 6: Internet facts.

Using the previous theorem and Table 6, we compute in Table 7 the values M for URLs and domain names for different prefix sizes over the last few years. When Google Safe Browsing was started in 2008 ($\ell = 32$), at most 443 URLs matched a given prefix. This number has increased over the years to reach 14757 in 2013. In the worst case, it is hard for Google and Yandex to re-identify a URL from its 32-bit prefix. However, if the URL is a domain name (second level for instance), the situation is different because the domain name space is 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. So far, the solution seems to be privacy preserving as long as we only reveal a single prefix.

	M **for URLs**			M **for domain**		
ℓ (bits)	2008	2012	2013	2008	2012	2013
16	2^{28}	2^{28}	2^{29}	253	363	388
32	443	7541	14757	2	3	3
64	2	2	2	1	1	1
96	1	1	1	1	1	1

Table 7: M for URLs and domain names with different prefix sizes ℓ.

However, for GSB and YSB, the server can receive several prefixes for a single URL: k decompositions of the URL can be tested as positive. Randomly choosing two URLs for which the decomposition matches exactly the same 2 prefixes is unlikely: 2^{-64}. It implies that if there is more than one match among the decompositions, the URL can be re-identified.

A surveillance system based on YANDEX or GOOGLE Safe Browsing will work as follows. First, they build a shadow database of prefixes corresponding to at least two decompositions of the targeted URLs. Second, they insert/push those prefixes in the client's database. GOOGLE and YANDEX can identify interesting individuals each time their server receives a query with a combination of at least two prefixes matching in the shadow database.

To illustrate our discovery let us use the URL of PETS conference Call-for-Paper (CFP) webpage. We assume that the corresponding prefix: `0xbd27e944` is included in the prefix list. According to Table 7, if the server receives this prefix, it can not tell for sure that the user has visited the following CFP webpage:

`https://petsymposium.org/2015/cfp.php`.

It might be another colliding page. Let us now assume that `0x7b45bc0c` and `0xf34e4a27` are both in the prefixes' list. When a user enters the previous URL, it will be decomposed into:

- `https://petsymposium.org/2015/`,
- `https://petsymposium.org`.

The prefixes corresponding to the decomposition both match the database. The server will receive the two prefixes and re-identify the URL by querying a shadow database.

Actually it is also possible to re-identify URL even if a single prefix of the decomposition matches the database. Indeed, it is possible to exploit temporal correlation between the queries of a user. YANDEX and GOOGLE can identify the requests of a given user thanks to the Safe browsing cookie. A user visiting:

`https://petsymposium.org/2015/cfp.php`

is very likely to visit the submission website:

`https://submit.petsymposium.org/hotcrp/pets2015-02/`.

Instead of looking at a single query, we need to correlate two queries. A user making two queries for the prefixes `0xb5e3d5ce` and `0x7b45bc0c` in a short period of time is planning to submit a paper.

Nobody can blame GOOGLE or YANDEX for these false positives because they can create malware and phishing websites matching those prefixes. It will be the perfect crime: unlikely to be detected and impossible to be caught because it is trivial to forge alibis (forging second pre-image is discussed later in the paper). The rest of the section is dedicated to finding prefixes that can be tracked in the existing system. We also discuss at the end how to fight this surveillance system.

6.2 Understanding the Blacklists

As a first step in our study, we recover the blacklists of GOOGLE and YANDEX. We use the lists of prefixes to query GOOGLE and YANDEX servers using the API. This allows us to obtain the list of all the full digests corresponding to malware, phishing and adult sites.

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 [Cor]. We were unable to use all the sources in the list since several of these (`stopbadware.org` for instance) do not allow to use their lists for comparison purposes. We have not used them to respect their terms of use. In total, we harvested 1240300 malware URLs and 151331 phishing URLs. We then query GOOGLE and YANDEX Safe Browsing with those URLs. The results of our findings are shown in Table 8. The findings reveal that the reconstruction for GOOGLE prefixes is rather 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.

		Malware list		**Phishing list**	
	list name	#matches	%matches	#matches	%matches
GOOGLE	goog-malware-shavar	18785	5.9	351	0.1
	googpub-phish-shavar	632	0.2	11155	3.5
YANDEX	ydx-malware-shavar	44232	15.6	417	0.1
	ydx-adult-shavar	29	6.6	1	0.2
	ydx-mobile-only-malware-shavar	19	0.9	0	0
	ydx-phish-shavar	58	0.1	1568	4.9
	ydx-mitb-masks-shavar	20	22.9	0	0
	ydx-porno-hosts-top-shavar	1682	1.6	220	0.2
	ydx-sms-fraud-shavar	66	0.6	1	0.01
	ydx-yellow-shavar	43	20	1	0.4

Table 8: Number of matches with 1240300 malware websites and 151331 phishing websites.

In order to know how many URLs generate more than 2 hits in the blacklist, we have used the Alexa list of the most popular websites (10^6 URLs). We found nothing for GOOGLE, i.e., all the URLs create at most 1 hit in the malicious list by GOOGLE. The result for YANDEX is left in Appendix B (Table 12). To summarize, we found 12 URLs for which 2 prefixes in the decomposition create hits in the malicious lists. One URL creates a hit in `ydx-malware-shavar` while the remaining are hits in `ydx-porno-hosts-top-shavar`.

YANDEX can moreover know when a user attempts to login to the adult site `m.mofos.com` and when he logs out. YANDEX can also know the nationality of a person by the version of `xhamster` he visits.

Our observations are conclusive for YANDEX but not for GOOGLE. However, it is easy to think that GOOGLE could be requested to include specific prefixes to track certain individuals.

6.3 Orphan Prefixes

After this first point, we look for *orphan prefixes* in YANDEX and GOOGLE Safe Browsing. We say that a prefix is orphan if there is no full digest matching it. We also look for URLs of Alexa list which match an orphan prefix. The result of our findings is presented in Table 9. Both GOOGLE and YANDEX have orphans: 159 for the two files of GOOGLE (0.02% of their prefixes) and 46521 for YANDEX (10.8%). The proportion of orphans for YANDEX is clearly not negligible.

We did not find any URL in Alexa list matching a GOOGLE orphan prefix. But there are 572+88 URLs of Alexa with one parent: the prefix matches one full digest. For YANDEX, we found 271 URLs matching an orphan prefix and 20220 URLs with one parent.

		#full hash per prefix				#Coll. with TopAlexa			
	list name	0	1	2	Total	0	1	2	Total
GOOGLE	goog-malware-shavar	36	317,759	12	317,807	0	572	0	572
	googpub-phish-shavar	123	312,494	4	312,621	0	88	0	88
YANDEX	ydx-malware-shavar	4,184	279,015	12	283,211	73	2,614	0	2,687
	ydx-adult-shavar	184	250	0	434	38	43	0	81
	ydx-mobile-only-malware-shavar	130	1,977	0	2,107	2	22	0	24
	ydx-phish-shavar	31,325	268	0	31,593	22	0	0	22
	ydx-mitb-masks-shavar	87	0	0	87	2	0	0	2
	ydx-porno-hosts-top-shavar	240	99,750	0	99,990	43	17,541	0	17,584
	ydx-sms-fraud-shavar	10,162	447	0	10,609	76	3	0	79
	ydx-yellow-shavar	209	0	0	209	15	0	0	15

Table 9: Distribution of prefixes in the lists provided by GOOGLE and YANDEX as the number of full hashes per prefix. For instance, there are 36 orphan prefixes in `goog-malware-shavar` list for which their is no full hash. We also provide the number of sites of Top Alexa which trigers an orphan prefix.

Orphans are false negatives. If a prefix has one or more parents it can be a false positive or a true positive. Only GOOGLE and YANDEX know the status of these prefixes. We can offer two explanations: with respect to an honest and an attacker's perspective. An honest perspective could be that there are gaps between the client database and the server database. The paranoid alternative is that GOOGLE and YANDEX tamper with their prefixes' database for tracking purpose. We have observed GOOGLE service for five months and the presence of orphans is persistent and their number is constant. We can not prove with certainty that they are actually tracking their users but we show that they have wide and open opportunities.

Clearly, GOOGLE and YANDEX Safe Browsing are not good for mass tracking. They can excel in dedicated tracking too. It can be for positive applications such as tracking visits to illegal sites (to track terrorists or pedophiles). But there is also a risk that the tool can be used to track people or individuals belonging to minority groups. As a conclusion, some true positives or false positives can leak browsing information and can be used to recover URLs for YANDEX Safe Browsing. We have no direct evidence for GOOGLE but there is no reason it would not work.

6.4 Hashing for privacy

We have two solutions to protect ourselves from Safe Browsing services: disable them or making useless. Disabling Safe Browsing implies losing a valuable service. Making the surveillance useless implies that GOOGLE or YANDEX can not distinguish the prefixes corresponding to a targeted URL from another irrelevant URL and if many people trigger the surveillance system. To achieve these goals we need to create URLs with legitimate content sharing the same 32-bit prefixes as those of the services' entries. This essentially consists in finding *multiple (second) pre-images* over 32-bit prefixes (for more details and definitions Appendix A). Once these websites are online,

they will attract users and will trigger the system. GOOGLE will start to be unable to distinguish the targeted people from the mass.

This is due to the fact that the certainty with which the Safe Browsing servers could re-identify the URL given prefixes gets decreased. The factor of uncertainty depends on the number of multiple (second) pre-images found. This idea is very similar to the algorithmic complexity attack introduced by Crosby and Wallach [CW03].

We have written a second pre-image search engine in Python. In order to ensure that the URLs are human readable, we have employed fake-factory[4] (version 0.2) a python package to generate fake but human readable URLs. The domain name corresponding to each generated URL is also checked for its availability using the python module `pywhois`: a wrapper for the Linux `whois` command. Our scheme can also be tuned to generate URLs for a given existing domain. Our brute-force-search being highly parallelizable, we exploit the module Parallel Python[5]. Our experiments were performed on a cluster with CPython 2.6.6 interpreter. The cluster runs 64-bit processors powered by an Intel QEMU Virtual CPU (cpu64-rhel6), with 32 cores running at 2199 MHz. The machine has 4MB cache and is running Centos Linux 2.6.32.

The goal of the search engine was to search for second pre-images for 1 million popular webpages of Alexa. Fig. 3 presents results obtained at the end of 1 week. The total number of second pre-images found was $111,027,928$. Since the Alexa list contains 1 million entries, as expected the number of second pre-images found per URL is highly concentrated around 100. For around $38,000$ URLs in Alexa list, we found 110 second pre-images per URL. The tail corresponds to the URLs for which we found the largest number of second pre-images. Finding second pre-images for a 32-bit prefix is fast and inexpensive.

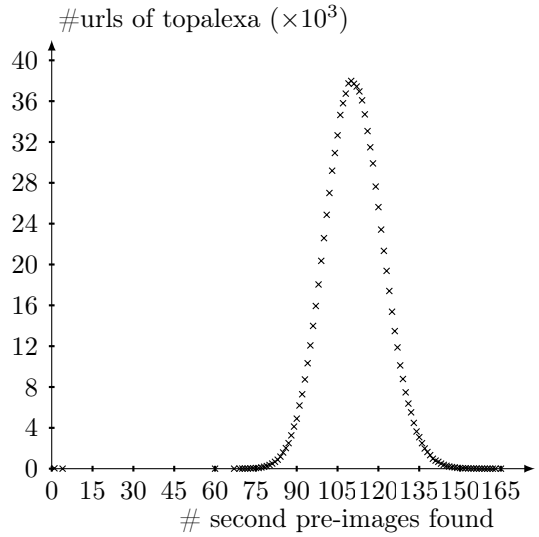


Figure 3: Number of second pre-images found per URL in the Alexa list of 1 million URLs.

To give a concrete case, we can create fake profiles on `xhamster` to protect the URL `http://fr.xhamster.com/user/kmille`, which corresponds to the channel of a user 'kmille'. This URL is listed in Appendix B since the prefixes of decompositions `fr.xhamster.com` and `xhamster.com` are in the prefix lists. The channel 'kmille' on `xhamster` is not in the database but let us assume

[4] `https://pypi.python.org/pypi/fake-factory/0.2`
[5] `http://www.parallelpython.com/`

it is. In order to protect the privacy of a user visiting the channel, we create fakes profiles on `xhamster` which share the same prefix and decomposition. Table 10 shows the values we obtain for `http://fr.xhamster.com/user/kmille`. It took us a night of computation to obtain them. These fake profiles hide user's visit to the channel since it is impossible for the Safe Browsing servers to know from the prefixes only if a user has visited the authentic URL or one of the fakes.

http://fr.xhamster.com/user/1375396439	e235ec60
http://fr.xhamster.com/user/6488033553	
http://fr.xhamster.com/user/13745385596	
http://fr.xhamster.com/user/14732183940	
http://fr.xhamster.com/user/16593185848	

Table 10: Fake profiles to protect kmille channel viewers.

To give a concrete case, we can create fake profiles on `xhamster.com` to protect the URL `http://fr.xhamster.com/user/kmille`. This URL is listed in Appendix B because the prefixes of `fr.xhamster.com` and `xhamster.com` are in the database. The channel of user kmille is not in the database but let is assume it is. We can create fakes profiles which share the same prefix and decomposition. Table 10 shows the values we obtain for `http://fr.xhamster.com/user/kmille`. It took us a night of computation to obtain them. It is impossible for the tracker to know from the prefixes only if you have visited the authentic URL or one of the fakes.

It is interesting to notice that Google or Yandex can use the same tactic to justify the inclusion of certain prefixes in the database. For a given targeted website, they can create several malware pages matching the site prefixes. If anyone wants to blame them, they can still use the URLs of their malwares as an alibi.

7 Censorship

We want to conclude our analysis of Google and Yandex Safe Browsing by some considerations on censorship. By nature, a Safe Browsing service censors malicious URLs. To censor an URL, Google and Yandex need to push the prefixes into the client database and to store the full digest in their servers. Without the computational power of Google, it is hard to re-identify the malicious URLs from Google full digests. However, we still want to check if abuse occurs. We limit our investigation to the URLs from Alexa in which trigger Safe Browsing (both orphans or with one parent). We attempt to categorize these URLs into four categories: religion, war/crime, disease and sex. The result of our classification can be found in Appendix C(Table 13). Amongst these URLs, we discover several interesting cases. The webpage of the French NGO "*secours islamique*" is a notable one. When a user attempts to visit the webpage of the NGO `www.secours-islamique.org` through Yandex.Browser, he is redirected to the warning page of Yandex Safe Browsing. The page is considered as safe by Google Safe Browsing, WOT, Microsoft filter and other malwares lists we have explored. We eventually find two records for `www.secours-islamique.org` at `stopbadware.org`. Two URLs of the website were blacklisted by Google in 2013-2014 , for a period of 23 and 146 days respectively.

There are two possible explanations behind the presence of the NGO's webpage in the Yandex malware list. Either, the list is not updated or the prefix of the webpage has been kept intentionally in the list so as to censor the webpage. After contacting them, we realize that the people from *Secours Islamique* were unaware of the situation.

The root of the problem is that Google or Yandex are the only one to control the content of the databases of prefixes and of the dangerous websites. Until this situation is not changed,

	Quarantine Period	
Path blacklisted	Start	End
ramadan-2011/	2013-DEC-21	2014-JAN-13
projet/	2013-AUG-20	2014-JAN-13

Table 11: Records for `secours-islamique.org` from `stopbadware.org`.

suspicion will stay on these services. Google, Yandex, Microsoft and the others must be made accountable for their blacklists. A first step will be to publish those blacklists. All the companies involved in the Safe Browsing business are supporting `stopbadware.org`. StopBadware is a nonprofit anti-malware organization. They advertise 1,610,355 malwares in their database. However, getting their database comes with a lot of restriction usage. We can not compare solutions for instance. Single queries are possible but you need to know what you are looking for. It limits the possibilities in term of accountability. Moreover, privacy and transparency are not mentioned in their goals. We also need an international surveillance network to ensure the consistency of these services and that they behave well.

8 Related work

Browsers have always been under scrutiny especially for their privacy implications. Aggarwal et al. [ABJB10] provide an analysis of private browsing modes in Firefox, Internet Explorer, Chrome and Safari. They show that in most cases, private browsing does not hide IP address and browser fingerprint (à la Panopticlick [Eck10]) and that some browsers make half hearted attempt if not at all to hide public state of the browser. Lerner et al [LEPK13] provide a static type system to analyze JavaScript extensions for observation of private browsing mode. The authors use the type system to verify compliance of Firefox extensions with private-browsing. The findings reveal that several extensions violate private-browsing objectives while others exhibit dubious behavior. Several works on privacy enhancing techniques such as [SRG97] and Tor-based extensions such as [Per, Rom] focus on hiding the client's IP address. Doppelganger [SK06] is a tool that focuses on cookie privacy. Bugnosis [AM02] is a Firefox extension that warns users about server-side tracking using web bugs.

A key question for the privacy of Safe Browsing services is the practicability of *private information retrieval* (PIR). If PIR schemes are usable and can be deployed at a large scale then Safe Browsing can be made private. The cost of PIR is a question highly debated by the community. Sion and Carbunar [SC07] evaluate the performance of single database PIR scheme. The authors show that the deployment of non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude less time-efficient than trivially transferring the entire database. The study primarily considers the computational PIR protocol of [KO97]. The authors argue that *a PIR is practical if and only if per-bit server side complexity is faster than a bit transfer*. With a normal desktop machine, trivial transfer (at 10MBps) of the database is 35 times faster than PIR. This ultimately restricts the use of PIR protocols for low bandwidths (tens of KBps). Olumofin and Goldberg [OG12] refute the general interpretation [SC07] that no PIR scheme can be more efficient that the trivial one. Authors evaluate two multi-server information-theoretic PIR schemes by Chor et al. [CGKS95] and by Goldberg [Gol07] as well as a single-server lattice-based scheme by Aguilar-Melchor and Gaborit [MG08]. It is hard to imagine a company using multi-server approach for its Save Browsing services: *it seems to the authors that PIR is not yet usable right for a large scale deployment.*

We highlight that Safe Browsing services are only one of the many prevalent web-malware

detection systems: *Virtual Machine client honeypots*, *Browser Emulator client honeypots*, *Classification based on domain reputation* and *Anti-Virus engines*. The VM-based detection systems [WBJ+06, MBGL06, MBD+07] typically detect exploitation of web browsers by monitoring changes to the OS, such as the creation of new processes, or changes to the file system or registry. A browser emulator creates a browser-type environment and uses dynamic analysis to identify exploits. Several works [CKV10, Naz09] follow this approach and emulate a browser to extract features from web pages that indicate malicious behavior. In the absence of malicious payloads, it is also possible to take a content-agnostic approach to classify webpages based on the reputation of the hosting infrastructure. Some of these techniques [APD+10, FKP10] leverage on DNS properties to predict new malicious domains based on an initial seed. Finally, Anti-Virus systems operate by scanning payloads for known indicators of maliciousness. These indicators are identified by signatures, which must be continuously update to identify new threats. The main advantage of these alternatives to Safe Browsing services is that they can be run locally on the users computers without disclosing information to a third party. However, we have not checked if the existing products are privacy-friendly.

9 Conclusions

Safe Browsing services are valuable tools to fight malwares, phishing and other online frauds. Google and Yandex have made a real effort to make their services as private as possible. Unfortunately, it does not work and we have shown in our paper why the design of these two services is flawed: hashing and truncation have failed again. The situation is grim because none of the Safe Browsing services respects the privacy their users.

Our theoretical considerations are also supported by observations on Yandex database and to a lesser extend on Google. It may be the results of development errors, misconfigurations or deliberate tracking and censorship. Our goal is certainly not to blame or accuse any entity, but rather to alarm users on the dangers of the Safe Browsing services. After reading the paper, the reader must be convinced that Google and Yandex Safe Browsing are dangerous.

The only definitive answer for the problems related to Safe Browsing is private information retrieval. New efficient schemes are critical and can improve dramatically the privacy of Safe Browsing.

References

[AB12] Jean-Philippe Aumasson and Daniel J. Bernstein. SipHash: A Fast Short-Input PRF. In *Progress in Cryptology - INDOCRYPT 2012*, Lecture Notes in Computer Science 7668, pages 489–508, Kolkata, India, December 2012. Springer.

[ABJB10] Gaurav Aggarwal, Elie Bursztein, Collin Jackson, and Dan Boneh. An Analysis of Private Browsing Modes in Modern Browsers. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 79–94, 2010.

[AM02] Adil Alsaid and David Martin. Detecting web bugs with bugnosis: Privacy advocacy through education. In *Privacy Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers*, pages 13–26, 2002.

[APD+10] Manos Antonakakis, Roberto Perdisci, David Dagon, Wenke Lee, and Nick Feamster. Building a dynamic reputation system for DNS. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 273–290, 2010.

[App10] Austin Appleby. smhasher - Test your hash functions., 2010. `https://code.google.com/p/smhasher/`.

[BLFM05] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (INTERNET STANDARD), January 2005. Updated by RFCs 6874, 7320.

[Blo70] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM*, 13(7):422–426, 1970.

[BM05] Andrei Broder and Michael Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4), 2005.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *Annual Symposium on Foundations of Computer Science, FOCS 1995*, 1995.

[CKV10] Marco Cova, Christopher Kruegel, and Giovanni Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 281–290, New York, NY, USA, 2010. ACM.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Cor] Zelster Security Corp. `https://zeltser.com/malicious-ip-blocklists/`.

[Cor13] Microsoft Corporation. Microsoft Security Intelligence Report. Technical report, Microsoft, December 2013. `https://bit.ly/1qAfTgt`.

[CW03] Scott A. Crosby and Dan S. Wallach. Denial of Service via Algorithmic Complexity Attacks. In *USENIX Security Symposium*, Lecture Notes in Computer Science 7668, pages 3–3, Washington, USA, December 2003. USENIX Association.

[Dan12] Quynh Dang. Recommendation for Applications Using Approved Hash Algorithms. Technical Report SP 800-107 Revision 1, National Institute of Standards & Technology, august 2012.

[Eck10] Peter Eckersley. How unique is your web browser? In *Proceedings of the 10th International Conference on Privacy Enhancing Technologies*, PETS'10, pages 1–18, Berlin, Heidelberg, 2010. Springer-Verlag.

[FKP10] Márk Félegyházi, Christian Kreibich, and Vern Paxson. On the potential of proactive domain blacklisting. In *LEET*, 2010.

[Gol07] I. Goldberg. Improving the Robustness of Private Information Retrieval. In *Security and Privacy, 2007. S&P '07. IEEE Symposium on*, 2007.

[goo] `http://googleblog.blogspot.fr/2008/07/we-knew-web-was-big.html`.

[Inc] Verisign Inc. http://www.verisigninc.com/en_US/innovation/dnib/index.xhtml.

[Inc14] Google Inc. Google Transparency Report. Technical report, Google, June 2014. https://bit.ly/1A72tdQ.

[Jen96] Robert Jenkins. A Hash Function for Hash Table Lookup, 1996. http://www.burtleburtle.net/bob/hash/doobs.html.

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is Not Needed: Single Database, Computationally-private Information Retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, 1997.

[LEPK13] Benjamin S. Lerner, Liam Elberty, Neal Poole, and Shriram Krishnamurthi. Verifying web browser extensions' compliance with private-browsing mode. In *Computer Security - ESORICS 2013 - 18th European Symposium on Research in Computer Security, Egham, UK, September 9-13, 2013. Proceedings*, pages 57–74, 2013.

[MBD+07] Alexander Moshchuk, Tanya Bragin, Damien Deville, Steven D. Gribble, and Henry M. Levy. Spyproxy: Execution-based detection of malicious web content. In *Proceedings of the 16th USENIX Security Symposium, Boston, MA, USA, August 6-10, 2007*, 2007.

[MBGL06] Alexander Moshchuk, Tanya Bragin, Steven D. Gribble, and Henry M. Levy. A crawler-based study of spyware in the web. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA*, 2006.

[McAa] McAfee. McAfee Site Advisor. http://www.siteadvisor.com/.

[McAb] McAfee. McAfee Site Advisor FAQ. https://kc.mcafee.com/corporate/index?page=content&id=KB73457.

[McAc] McAfee. McAfee Site Advisor Live. http://home.mcafee.com/store/siteadvisor-live.

[MG08] C.A. Melchor and P. Gaborit. A fast private information retrieval protocol. In *Information Theory, 2008. ISIT 2008. IEEE International Symposium on*, 2008.

[Mica] Microsoft. SmartScreen Application Reputation. http://blogs.msdn.com/b/ie/archive/2011/05/17/smartscreen-174-application-reputation-in-ie9.aspx.

[Micb] Microsoft. SmartScreen: Authenticode Code Signing. http://blogs.msdn.com/b/ieinternals/archive/2011/03/22/authenticode-code-signing-for-developers-for-file-downloads-building-smartscreen-applicati
aspx.

[Micc] Microsoft. Windows Logo. https://msdn.microsoft.com/en-us/windows/dd203105.aspx.

[MKD+02] J. Mogul, B. Krishnamurthy, F. Douglis, A. Feldmann, Y. Goland, A. van Hoff, and D. Hellerstein. Delta encoding in HTTP. RFC 3229, RFC Editor, January 2002. http://tools.ietf.org/html/rfc3229.

[MVO96] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., 1st edition, 1996.

[Nat12] National institute of standards and technology. Secure Hash Standard (SHS). Technical Report FIPS PUB 180-4, National Institute of Standards & Technology, march 2012.

[Nat14] National institute of standards and technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. Technical Report FIPS PUB 202, National Institute of Standards & Technology, may 2014. draft.

[Naz09] Jose Nazario. Phoneyc: A virtual client honeypot. In *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, Berkeley, CA, USA, 2009. USENIX Association.

[OG12] Femi Olumofin and Ian Goldberg. Revisiting the Computational Practicality of Private Information Retrieval. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2012.

[Per] Mike Perry. Torbutton. `https://www.torproject.org/docs/torbutton/`.

[Pro] Niels Provos. Safe Browsing - Protecting Web Users for 5 Years and Counting. `http://googleonlinesecurity.blogspot.fr/2012/06/safe-browsing-protecting-web-users-for.html`.

[Rom] Sasha Romanosky. FoxTor: helping protect your identity while browsing online. `cups.cs.cmu.edu/foxtor`.

[RS98] Martin Raab and Angelika Steger. "Balls into Bins" - A Simple and Tight Analysis. In *Proceedings of the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, RANDOM '98, pages 159–170, London,UK, 1998. Springer-Verlag.

[sba] Safe Browsing API. `https://developers.google.com/safe-browsing/`.

[SC07] Radu Sion and Bogdan Carbunar. On the Practicality of Private Information Retrieval. In *Proceedings of the Network and Distributed System Security Symposium – NDSS 2007*, San Diego, CA, USA, February 2007. The Internet Society.

[SK06] Umesh Shankar and Chris Karlof. Doppelganger: Better browser privacy without the bother. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 154–167, New York, NY, USA, 2006. ACM.

[SRG97] Paul F. Syverson, Michael G. Reed, and David M. Goldschlag. Private web browsing. *Journal of Computer Security*, 5(3):237–248, 1997.

[Swe02] Latanya Sweeney. k-Anonymity: A Model for Protecting Privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(5):557–570, 2002.

[Syma] Symantec. Norton Safe Search. `http://nortonsafe.search.ask.com/`.

[Symb] Symantec. Norton Safe Web. `https://safeweb.norton.com/`.

[WBJ+06] Yi-Min Wang, Doug Beck, Xuxian Jiang, Roussi Roussev, Chad Verbowski, Shuo Chen, and Samuel T. King. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *NDSS*, 2006.

[WS] Ltd WOT Services. Web of trust. `https://www.mywot.com`.

[Yan] Yandex. Yandex Safe Browsing. `http://api.yandex.com/safebrowsing/`.

A Hash functions

A hash function compresses inputs of arbitrary length to digest/hash of fixed size. It is a very popular primitive used in algorithms [CLRS09] and in cryptography/security [MVO96]. The design of a "*good hash function*" depends on the field of application. Non-cryptographic hash functions such as MurmurHash [App10] or Jenkins hash [Jen96] are designed to be fast, to uniformly distribute their outputs and to satisfy several other criteria such as the avalanche test [App10]. The SMHASHER suite [App10] provides a good overview of these functions and the tests they must satisfy.

The design of a cryptographic hash function is very different. Cryptographic hash functions are slower than their non-cryptographic siblings. Furthermore, in order to resist to an adversary, a cryptographic hash function h must verify the following properties:

- **Pre-image resistance:** Given a digest d, it is computationally infeasible to find an input x, such that $h(x) = d$.
- **Second pre-image resistance:** Given an input x and the digest $h(x)$, it is computationally infeasible to find another input $x' \neq x$, such that $h(x) = h(x')$.
- **Collision resistance:** It is computationally infeasible to find two inputs $x \neq x'$ such that $h(x) = h(x')$.

Non-cryptographic hash functions are not designed to satisfy these properties [CW03, AB12]. Let us consider a hash function h with ℓ-bit digests. The choice of ℓ is critical for cryptographic hash functions because the basic complexities for finding pre-image, second pre-image and collisions are 2^ℓ, 2^ℓ and $2^{\ell/2}$ respectively. A cryptographic hash function is considered secure if there is no attack with a lower complexity. The NIST recommendation [Dan12] for cryptographic hash functions are SHA-256, SHA-384, SHA-512 [Nat12] and SHA-3 [Nat14].

We provide in this paragraph two natural extensions of the previous concepts: *truncated digests* and *multiple pre-images* or *multiple second pre-images*. In fact, many applications need to reduce the size of the full digest. Truncated digests must be carefully used as explained in [Dan12], since it is commonly assumed that for a truncated digest of ℓ' bits the security is reduced at least to $2^{\ell'}$ (pre-image and second pre-image) and $2^{\ell'/2}$ (collision). If ℓ' is too small, computation of pre-image, second pre-image or collisions are feasible. For instance, in GOOGLE Safe Browsing (Section 4), SHA-256 is applied to URLs but the digests are truncated to 32 bits *vs* 256 bits of SHA-256. With 32 bits of digest, pre-image, second pre-image can be computed after 2^{32} brute force computations. We also define *multiple pre-images*, where, given a digest d, we wish to compute n pre-images x_i such that $h(x_i) = d$. Accordingly, we define *multiple second pre-images*, where given x' and $h(x')$, we wish to compute n second pre-images x_i such that $h(x_i) = h(x')$.

B Multiple prefix URLs

We present in Table 12, URLs in the Alexa list for which more than 1 decompositions were found to be present in the malicious prefix lists of YANDEX. The first URL:

`http://torr.comoj.com`

creates two hits in the `ydx-malware-shavar` file of YANDEX. The remaining 11 create two hits in the `ydx-porno-hosts-top-shavar` file of YANDEX.

URL	matching decomposition	prefix
http://torr.comoj.com/	torr.comoj.com/	`0x18e3177e`
	comoj.com/	`0xc748b1b8`
http://rpc.weblogs.com/	rpc.weblogs.com/	`0xfb8c1dea`
	weblogs.com/	`0x58c465bd`
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.mofos.com/user/login	m.mofos.com/	`0x6e961650`
	mofos.com/	`0x00354501`
http://fr.xhamster.com/user/kmille	fr.xhamster.com/	`0xe4fdd86c`
	xhamster.com/	`0x3074e021`
http://de.xhamster.com/user/video	de.xhamster.com/	`0x0215bac9`
	xhamster.com/	`0x3074e021`
http://nl.xhamster.com/user/ppbbg	nl.xhamster.com/	`0xa95055ff`
	xhamster.com/	`0x3074e021`
http://mobile.teenslovehugecocks.com/user/join	mobile.teenslovehugecocks.com/	`0x585667a5`
	teenslovehugecocks.com/	`0x92824b5c`
http://nl.xhamster.com/user/photo	nl.xhamster.com/	`0xa95055ff`
	xhamster.com/	`0x3074e021`
http://m.mofos.com/user/logout	m.mofos.com/	`0x6e961650`
	mofos.com/	`0x00354501`
http://m.wickedpictures.com/user/login	m.wickedpictures.com/	`0x7ee8c0cc`
	wickedpictures.com/	`0xa7962038`

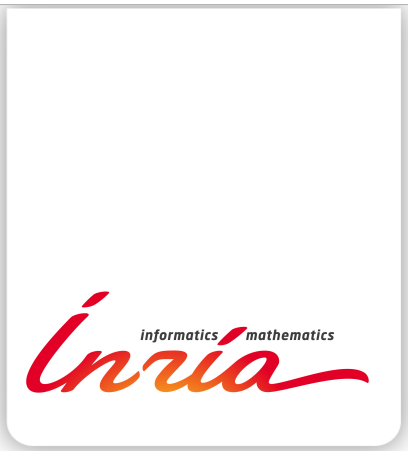
Table 12: URLs with multiple matching prefixes.

C Characterizing Alexa prefixes

We present here Table 13 which characterizes Alexa prefixes which are considered as malicious. We classify the corresponding URLs into four important categories: religion, war/crime, disease, and sex.

		Orphans				One parent			
	list name	**religion**	**war/crime**	**disease**	**sex**	**religion**	**war/crime**	**disease**	**sex**
Google	goog-malware-shavar	na	na	na	na	1	6	2	34
	googpub-phish-shavar	na	na	na	na	0	2	0	1
Yandex	ydx-malware-shavar	4	0	3	11	17	21	22	79
	ydx-adult-shavar	0	1	0	32	0	0	0	13
	ydx-mobile-only-malware-shavar	1	0	1	1	0	0	0	4
	ydx-phish-shavar	0	0	1	0	na	na	na	na
	ydx-mitb-masks-shavar	0	0	0	0	na	na	na	na
	ydx-porno-hosts-top-shavar	0	0	0	43	26	58	26	6882
	ydx-sms-fraud-shavar	0	0	0	20	0	0	0	0
	ydx-yellow-shavar	0	0	0	3	na	na	na	na

Table 13: Characterising TopAlexa websites whose 32-bit prefix is present in the Yandex and Google lists.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399