Private Information Disclosure from Web Searches

Claude Castelluccia[1], Emiliano De Cristofaro[2], Daniele Perito[1]

[1] INRIA Rhone Alpes, Montbonnot, France
[2] University of California, Irvine

Abstract. As the amount of personal information stored at remote service providers increases, so does the danger of data theft. When connections to remote services are made in the clear and authenticated sessions are kept using HTTP cookies, intercepting private traffic becomes easy to achieve. In this paper, we focus on the world largest service provider – Google. First, with the exception of a few services only accessible over HTTPS (e.g., Gmail), we find that many Google services are vulnerable to simple session hijacking attacks. Next, we present the Historiographer, a novel attack that reconstructs the web search history of Google users – Google's Web History – even though this service is supposedly protected from session hijacking by a stricter access control policy. The Historiographer uses a reconstruction technique inferring search history from the personalized suggestions fed by the Google search engine. We validate our technique through experiments conducted over real network traffic and discuss possible countermeasures. Our attacks are general and not only specific to Google, and highlight privacy concerns of mixed architectures mixing secure and insecure connections.

1 Introduction

With the emergence of cloud-based computing, users store an increasing amount of information at remote service providers. Cloud-based services often come at no cost for the users, while service providers leverage considerable amounts of user profiling information to deliver targeted advertisement. However, storing large amounts of personal information to external providers raises privacy concerns. Privacy advocates have highlighted the conceptual and practical dangers of personal data exposure over the Internet [12, 14–16].

In this paper, we analyze private information potentially leaked from web searches to third parties, rather than focusing on data disclosed to service providers.

Being the world's largest service provider, we focus on the case of Google. In particular, we analyze one Google service: Web History: It provides users with personalized search results based on the history of their searches and navigation. The history is accessible at `http://google.com/history`.

Web searches have been shown to be often sensitive [16]. Any information leaked from search histories could endanger user privacy. For example, it is likely that search histories contain personal health-related information: a recent research has, in fact, successfully correlated the spread of influenza and the number of related search queries divided by region [18]. Similarly, searches may be related to political or religious views, sexual orientation, etc. Also, AOL's release in 2006 of 20 million nominally anonymized searches underlined that search queries contain private information [10].

The privacy of personal data stored by service providers has been long threatened by the well-known attacks consisting of hijacking user's HTTP cookies.[1] These attacks have been addressed by Google in several ways. For instance, "sensitive" services such as Gmail now enforce secure HTTPS communication by default and transmit authentication cookies only over encrypted connections. Regarding Google Web History, the login page states: "To help protect your privacy, we'll sometimes ask you to verify your password even though you're already signed in. This may happen more frequently for services like Web History which involves your personal information". Frequently requesting users to re-enter their credentials can thwart the session hijacking attack. However, as illustrated in this paper, such an attack can still be effective if a user has just signed in. Moreover, we show that search histories can still be reconstructed even though the Web History page is inaccessible by hijacking cookies.

The Historiographer. To this end, we successfully design the Historiographer, an attack that reconstructs the history of web searches conducted by users on Google. The Historiographer uses the fact that users signed in any Google service receive suggestions for their search queries based on previously-searched keywords. Since Google Web Search transmits authentication cookies in the clear, the Historiographer—monitoring the network—can capture this cookie and use the search suggestions to reconstruct a user's search history. We refer to Section 3 for more details on the reconstruction technique.

Contributions. This paper makes the following contributions:

1. We show that the Google infrastructure is vulnerable to the Historiographer, a new attack that reconstructs part of the search history of users.
2. We show that the well known session hijacking attack is still applicable to many Google services. More specifically, we evaluate the security of several Google services, including Web History, against this simple attack and report the number of services vulnerable along with the amount and type of information potentially disclosed by each service.
3. We conduct an experimental analysis over network traces from a research institution, a Tor [1] exit node, and the 20 million anonymized searches released by AOL in 2006, in order to assess the number of potential victims and the accuracy of our attack. Results show that almost one third of monitored users were signed in their Google accounts and, among them, a half had Web History enabled, thus being vulnerable to our attack. Finally, we show that data from several other Google services can be collected with a simple session hijacking attack.

Paper Organization. The rest of the paper is organized as follows. Section 2 presents the necessary technical background. Section 3 details the new Historiographer attack. Section 4 describes our experimental evaluations on real network traffic, and estimates the number of potential victims and the accuracy of the Historiographer. Independently of Historiographer, this section also evaluates the additional information leaked from Google's services through simple session hijacking. Section 5 discusses possible countermeasures to thwart the Historiographer attack, while Section 6 overviews related

[1] In a session hijacking attack, an attacker monitoring the network captures an authentication cookie and impersonates a user. In Section 6, we will discuss several related vulnerabilities.

work. Section 7 concludes the exposition. Finally, in Section 8, we discuss the actions taken by Google in response to our findings.

2 Background

In the following, we present background information on several aspects discussed throughout the rest of the paper: the HTTP cookies, and the Google architecture.

2.1 HTTP Cookies

The need of maintaining sessions in HTTP emerged with the creation of the first web applications (e.g., e-commerce websites), as HTTP is a stateless protocol. RFC2109 [22] and RFC2965 [23] specified a standard way to create stateful sessions with HTTP requests and responses. They describe two new headers, Cookie and Set-Cookie, which carry state information between participating origin servers and user agents. A Cookie, which contains a unique identifier, is typically used to store user preferences or to store an authentication token. Cookies are set by the server as follows. After an incoming HTTP request, a server sends back a HTTP response containing an HTTP header, referred to as Set-Cookie, requesting the browser to store one or several cookies. Such a header is in the form of *name=value*, the so-called "cookie crumb". As a result, provided that the user agent enables cookies, every subsequent HTTP request to a server on the same domain will include the cookie in the Cookie HTTP header. A cookie may also include an expiration date[2], or a flag to mark it *secure*. In the latter case, the browser will send the secure cookie only over encrypted channels, such as SSL. A set-cookie header may optionally contain a *domain* attribute, which specifies the domain validity of the cookie. If this attribute is set, the cookie is referred to as domain cookie, as opposed to host cookie which is not specific to any particular sub-domain. For example, as we will present in Table 1, a user accessing Google's Calendar receives a domain cookie for `calendar.google.com` as an authentication token. Such a cookie is then to be included in every subsequent HTTP requests to the domain. In contrast, other Google's applications (such as the Search, History or Maps) only set host cookies, which are used across different services and domains. Finally, a set-cookie header may specify a path attribute to identify the subset of URLs for the cookie's validity. For example, as we will present in Table 2a, a user that signs in Google receives three cookies, namely `SID`, `SSID`, and `LSID`. While the latter only applies to the path "*/account*", the other two are can be used for different paths.

2.2 Google Architecture

As we mention in Section 1, we focus on the case of the world's largest service provider, i.e., Google. This section describes the Google architecture[3].

[2] If an expiration date is provided, cookies survive across browser sessions, and are then called "persistent". Otherwise, the cookie is deleted when closing the browser.

[3] Since not all the components of the Google architecture are public, some of the details presented in this section might not be completely accurate.

Google Web Products. Google offers more than 40 free Web services, including several search engines (e.g. Google Web Search), maps (Google Maps), as well as personalized subscription-based services like email (Gmail), documents (Google Docs), photos (Picasa), videos (Youtube), Web history. Even though some services can be used without registration (e.g., search), other are user-specific (e.g. Gmail) and require user authentication. Most of the services can be used by means of a single Google account, a combination of username and password. However, services that do not mandate registration provide extra features if users are signed in. For instance, an authenticated user can obtain personalized, potentially more accurate, search results on Google Maps based on her default location.

Google Web History. This *opt-out* service – previously known as Google Search History and Personalized Search – is implemented by Google to provide signed-in users with personalized search results based on the history of their searches and navigation. Furthermore, users typing search queries in the Web interface are prompted with *suggestions* resulting from their history. To this end, Google tracks all Web searches performed by a signed-in user (with Web History service enabled), as well as the target web pages clicked from the search result page. This service may be further enhanced by installing the Google Toolbar, allowing Google to also track *all* visited web sites, independently from the use of the search engine. Google Web History also provides a Web interface at `google.com/history`, allowing users to view and delete their history. Users are given the choice to *pause* Web History by accessing their account. Nevertheless, Google customizes searches and provides suggestions based on data recorded before pause. Note that Google is offering Personalized Search not only to signed-in but also to signed-out users. In fact, for these users Google performs the customization using the information linked to the user's browser with the help of an "anonymous" cookie. Specifically, Google stores up to 180 days of activity linked to such cookie. Again, users can explicitly disable this feature [3].

Google Authentication. Google services are accessible with a single set of credentials, composed by a pair username/password. Different services are usually hosted as subdomains of `google.com` (or other Top-Level Domains for different countries) and offer seamless integration between each other to minimize the need for users to reenter their credentials. Integration is achieved through the Accounts service. In practice, requests to authenticate to a Google service are redirected to the Accounts page where the user is asked to enter her username and password. If authentication succeeds, a browser cookie is set (or refreshed) to track the session and the user is redirected back to the page that was originally requested. An illustration of this mechanism is provided in Fig. 1.

Access to Google Accounts is always secured using HTTPS. However, subsequent connections might revert back to simple HTTP depending on the requested service. For example connection to Maps Search are established with HTTP whereas HTTPS access to Gmail is enforced.

Table 1 compares several Google Services. It may be the case that services considered more sensitive are protected by HTTPS, whereas those judged less sensitive are left unencrypted. In particular, we noticed that the use of HTTPS is mandatory for some services (e.g., Gmail), while impossible for others (e.g., Search). Additionally, there are

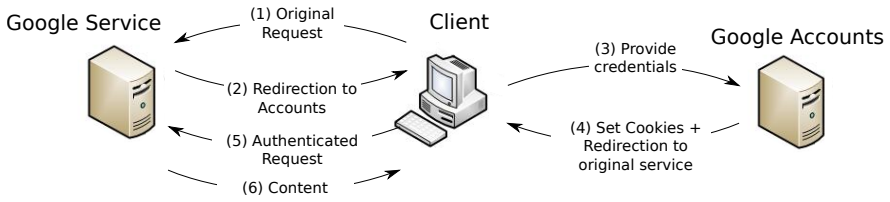

Fig. 1: The Google Accounts authentication management for Google services.

services accessed on HTTP by default, but users may force a secure connection specifying `https://` in the URL.

Service Name	Default Connect.	HTTPS Support	Domain specific cookie	Purpose
Search	HTTP	no	no	Web search
Maps	HTTP	no	no	Maps search
Reader	HTTP	yes	no	RSS/Atom feed reader
Contacts	HTTP	yes	no	Address book manager
History	HTTP	yes	no	Search history manager
Gmail	HTTPS	mand.	no	Web mail application
Accounts	HTTPS	mand.	no	Google account manager
News	HTTP	no	no	News aggregator
Bookmarks	HTTP	yes	no	Bookmark manager
Docs	HTTP	yes	yes	Office application
Calendar	HTTP	yes	yes	Calendar application
Groups	HTTP	yes	yes	Discussion groups application
Books	HTTP	no	no	Personalized digital library

Table 1: Some of Google's services

Google cookies. Authenticated sessions are kept by means of cookies that are set by Accounts upon successful authentication. Two cookies, called `SID` and `SSID`, are used as authentication tokens across most services[4] for unencrypted and encrypted connections, respectively. We believe their names might stand for *Session ID* and *Secure Session ID*[5].

A description of several Google cookies is reflected in Tables 2a and 2b. Note that: (1) `SID` and `SSID` are valid for all Google sub-domains and are used to authenticate users to several services, (2) `SID` is not a secure cookie, i.e., it is sent on every connection to Google, while `SSID` is only sent over encrypted connections, and (3) `NID` represents the "anonymous" cookie used to track unlogged users. There are also a num-

[4] All services that do not use domain cookies, such as Maps, History, Search, Reader, Books and Contacts – see Table 1.

[5] An additional list of domain-specific cookies, such as those for `docs.google.com` or `calendar.google.com`, are sent in the clear text but are *set* only over a secure connection upon user access.

ber of cookies not reported, which are used for miscellaneous purposes, e.g., to store language or search interface preferences.

In our study, we will focus on the SID cookie, providing authenticated access to most unencrypted services. In particular the SID cookie is sent in all web searches. It is used by Google to identify the requesting account, populate the account's Web History and provide personalized web results and suggestions.

3 Historiographer: Reconstructing Search History

3.1 Attack Overview

In the following, we present the Historiographer, an attack aiming to reconstruct users' search histories stored by Google. The attack consists of two steps.

First, it hijacks a session stealing the victim's SID cookie. This can be done, for example, by eavesdropping on her traffic, and in particular on any request to a Google service, such as Google search. Eavesdropping can be performed by listening on a local wired network, an open wireless network, such as a campus network, or by deploying a Tor exit node (as detailed in Section 4). This does not necessary involve compromising nodes, and therefore does not require special skills.

Second, it reconstructs the Web History using a *partial precise* inference attack [17]. We recall that an inference attack is a technique used to disclose sensitive and protected information from presumably non-sensitive data. In this setting, we reconstruct part of the potentially privacy-sensitive Web History from web searches. The technique is *partial*, because, as shown in Section 3.2, it does not always reconstruct the whole history. Finally, it is *precise* since it infers accurate items from the Web History without introducing errors, as opposed to imprecise inference techniques that do it with a certain probability.

Note that any user, in particular if equipped with a mobile device, is likely to access the Internet via an unencrypted wireless channel at some point of time. As soon as she signs in Google when connected to such unprotected networks, she becomes vulnerable to our attack. Furthermore, the attack is effective even if the user is careful and never inputs sensitive information during "insecure" browsing sessions over unencrypted wireless channels.

Cookie-Name	Secure	Domain	Path	Purpose
SID	no	google.com	/	authentication token
SSID	yes	google.com	/	secure authentication token
LSID	yes	google.com	/accounts	secure authentication token

(a) Google's cookies for signed in users

Cookie-Name	Secure	Domain	Path	Purpose
NID	no	google.com	/	track unlogged users
PREF	no	google.com	/	store search settings (e.g., language)

(b) Google's cookies for not signed in users

Table 2: Description of the type and purposes of some cookies used in the Google platform.

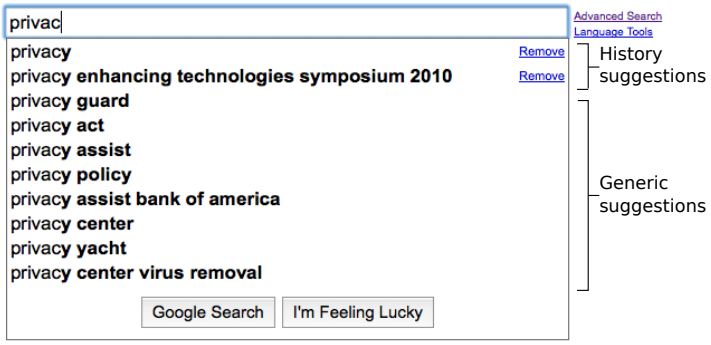

Fig. 2: An example of Google Search Suggestions.

3.2 Reconstructing the web history through inference

Web history access control. Authenticated users can consult, modify, pause or delete their complete history by accessing the Web History service. The history can be consulted as an HTML page or an RSS feed. However, as mentioned above, Google access control policy for Web History differs from the one implemented in other services. In fact, users are frequently asked to re-enter their credentials even though they are already authenticated. Preliminary tests showed that this mechanism is used quite frequently, and in such a case a session hijacker would be prevented from downloading the history.

Exploiting the search suggestion feature. However, a feature provided by Google, namely the *search suggestions*, helped us circumvent the access control enforced for Web History. As mentioned in Section 2, Google search engine offers contextual information in the search interface that can be derived from the user's search history. Specifically, whenever a prefix is typed in the search box, an Ajax [21] request is sent to a Google server, which replies with a list of associated keywords. Fig. 2 presents an example of a user typing the prefix "`privac`" in the search box. The user is then prompted with a list of related keywords to auto-complete the search, i.e., *search suggestions*. These keywords can either be based: (1) on Google's ranking of similarity (we call them *generic* search suggestions), or (2) on user's search history (we call them *history* search suggestions). Note that history search suggestions are only sent to the user if the typed prefix corresponds to search queries that are in the Web History and were followed by a "click" on one of the results. We call these queries: "clicked" queries. History search suggestions are visually distinguishable from the generic ones, since they include a link to remove them. This is reflected in the Javascript code, as history search suggestions have a flag set to differentiate them. The access to the web server that implements suggestions is carried out using Ajax and every request is authenticated sending an `SID` cookie, which can be easily eavesdropped and hijacked. Therefore, once an `SID` cookie has been captured, the user's Web History can be reconstructed using the suggestion service: Historiographer steals a user authentication cookie and then sequentially requests possible prefixes to the suggestion server to recover keywords coming from the history.

Reconstruction algorithms. In order to reconstruct a search history, the Historiographer needs to ask for suggestions for different prefixes. Hence, we need to carefully select the list of possible prefixes to use, since the keywords in the history are unknown. We encounter the following obstacles: (1) the number of requests for suggestions should be kept to a minimum, in order to be as stealthy as possible; (2) at most three replies come from the suggestion history upon each suggestion request, limiting the amount of information discovered with each request; and (3) suggestions are only returned for two-letter (or longer) prefixes, preventing from simply looking for all letters in the given alphabet. A naïve (*brute-force*) approach would involve requesting all the possible two- or three-letter prefixes to harvest the replies coming from the history. However, this would already require $26^2 = 676$ requests for two-letter and $26^3 = 17,576$ for three-letter combinations in the English alphabet, hence relatively high numbers that might lead to detection. Instead, the Historiographer employs a more sophisticated technique: it requests only prefixes that are common in a given language. For instance, if one considers English, there are only 7 words starting with the two-letter prefix `oo`, while no word starts with the prefix `qr`. Whereas, the most used prefix results to be `co`, used in 3223 words. It is then reasonable to expect that in the search history there are more entries starting with the letters `co` than with `qr`. As a result, we proceed as follows: We extract all two-letter prefixes from a reference corpus, order them by frequency, and we select only the prefixes in the $90th$ percentile. We used two different reference corpora in our experiments: the English dictionary and the AOL dataset of 20 million anonymized searches that was released in 2006 [10]. However, they both achieved very similar performance. For the English dictionary, this yields a total of 121 two-letter combinations and reduces the number of requests and the fingerprint of the attack[6]. Further, we notice that at most 3 search suggestions can come from the history for each requested prefix. Thus, if we get exactly 3 suggestions from the history, there are either 3 or more search queries starting with the corresponding prefix. This is a potential indication that this prefix is particularly frequent in the history, and it is worth being further explored. Hence, whenever we encounter a two-letter prefix producing 3 suggestions, we add another letter to the prefix and we repeat the request. Note that the resulting three-letter prefix is again generated by extracting the most common three-letter prefixes from the dictionary and not by simply adding every possible letter in the alphabet. Fig. 3 visually depicts this procedure: The prefix `co` produces 3 results and is further explored, contrary to `de` and `ya` who produce only 2 (resp., 1) results. A description of the achieved accuracy and the related overhead in terms of requests is provided in Section 4.2.

Implementation. We implemented the Historiographer as a Perl application. It is part of a more complete tool that: (i) captures traffic from a network interface, (ii) recognizes cookies sent to and from Google servers, and (iii) then uses them to hijack sessions and retrieve personal information. Web History is only one of the services the software collects data from.

[6]Different languages can be supported by simply changing the alphabet and the reference corpus.

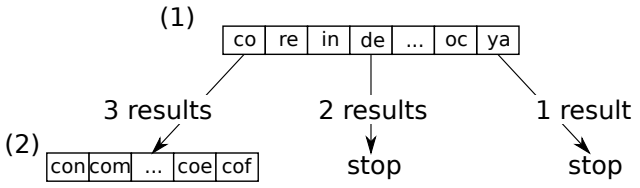

Fig. 3: Smart tree approach. To reconstruct large portions of the search history, we start with the most common two-letter prefixes (1). If a prefix produces 3 suggestions, then we descend in the tree (2).

3.3 Beyond the Historiographer: Exploiting Personalized Results

The Historiographer attack uses Personalized Search to leak information from a user's Web History. However, one could also use the so-called Personalized Results, i.e., the fact that search queries on Google often produce different results based on the user's search history. We present an example of this in Fig. 4. If the results contain at least one linked page previously accessed by the user, the "View customizations" link appears at the top right corner of the result page. One can easily identify the visited linked pages (e.g., `http://petsymposium.org/2010/` in Fig. 4) since they are marked with a tag reporting the number of visits (e.g., 8), and the date of the last visit (e.g., March 1st). Therefore, an adversary can verify that specific keywords belonging to a user's search history using the Personalized Results. We call such an attack a *targeted check*. Note that the adversary does not have to test the exact matching keyword searched by the user. It is enough to make a related search that includes the visited linked page in the results. For instance, assuming that a user has searched for *PETS 2010* and *Oakland 2010*, and has then clicked on the related links `http://petsymposium.org/2010/` and `http://oakland09.cs.virginia.edu/`. A subsequent search for the keyword *Privacy* would produce a result page with the "View customizations" link. Looking at the result page produced by only one request, an adversary can find out that the above pages were visited and conclude that the user is interested in *privacy*, in *PETS 2010* and *IEEE Security and Privacy*. The adversary could then try other keywords and broaden the information leakage or profile user's interests. Note that this attack can be amplified with the exploitation of the new Google's **Star** service that allows users to mark their favorite web sites. With **stars**, a user can mark his favorite sites by simply clicking the star marker on any search result or map. As a result of this action, these sites will appear in a special list next time the identical or a related search is performed. This feature gives even more power to the adversary. Note that this attack only applies to signed-in Google users with Web History enabled (a significant proportion of Google users as showed in Sec. 4). However, as discussed in Sec. 2, Google provides customized searches to signed-out users too, using an "anonymous" cookie. Therefore, we believe that a similar attack can be designed for signed-out users as well, although the history would be limited to the life of this cookie.

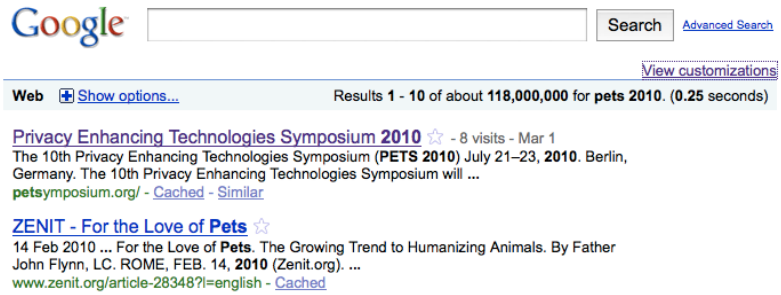

Fig. 4: An example of Google Personalized Results.

4 Measurements and Analysis

Given the private nature of the information gathered and the difficulty of having users willing to disclose them, we conducted four different experiments. These experiments were aimed at collecting data to estimate: (1) the number of potential victims that access Google services while being signed-in and, among them, how many have Web History enabled; (2) the accuracy and the cost of the Historiographer; (3) the amount of private data that can be retrieved from other services with the simple session hijacking attack; (4) the applicabity of the Historiographer on smart phones.

4.1 Estimating the number of potential victims

In order to estimate the number of potential victims of our attack, we conducted an experimental analysis on the network traces collected from a research center with about 500-600 daily users and a Tor exit node. We collected one week of network traffic during February 2010. The goal was to measure the percentage of users using Google while signed-in, and that having the Web History service enabled. Note that only aggregate data was stored. The data collected from the research center was analyzed passively, i.e., no session was actually hijacked.

In order to count the number of users from a network trace, one needs reliable identifiers to filter out duplicate queries or changes of network identifiers, e.g. IP churn. Luckily we could use cookies gathered from the network captures to identify single users. As explained above, Google issues persistent cookies both to signed-in and not signed-in users. Among them we chose to use `SID` cookies to identify signed-in users and `NID` cookies to identify not signed-in ones. Furthermore, in order to count the number of users with history enabled, our application looked for a particular link to the History service that is included in each search result page. The results of test are presented in Table 3a. Around one third of the users resulted to be signed-in while using Google services, including web searches. Furthermore, about half of the users with an account have history enabled. The limited size and the lack of randomness in the choice of our sample, does not allow us to draw conclusions about the entire population of users. However, if we combine our results with the above mentioned popularity of Google services, it would appear that a significant portion of web users are at risk.

Experiment	Number of Google users	Number of users signed-in	Number of signed-in users with History enabled
Research center	1502	543 (36.1%)	223 (14.8%)
Tor Exit Node	1893	872 (46.1%)	441 (23.29%)

(a) Measurements on network traces

User ID	$\mathbf{n_h}$	$\mathbf{n_c}$	$\mathbf{n_s}$	Recall	$\mathbf{n_{requests}}$	History Activation date
1	751	442	308	0.69	680	Aug 08, 2009
2	318	142	99	0.69	368	Mar 10, 2008
3	621	321	176	0.54	483	May 16, 2009
4	520	248	169	0.68	400	May 22, 2007
5	657	309	231	0.75	601	Feb 06, 2009
6	389	202	130	0.64	365	Fen 12, 2009
7	690	337	201	0.60	560	Jul 18, 2008
8	416	219	143	0.65	399	Aug 09, 2006
9	228	127	69	0.54	211	Aug 20, 2008
10	306	164	118	0.72	334	Sep 27, 2009
11	1567	930	506	0.54	740	Oct 26, 2009
12	1163	680	533	0.78	823	Dec 4, 2009

(b) Results from volunteers

Type of information leaked	Corresponding service	Number of Accounts accessible	Mean number of entries collected
Complete (unrestricted) Search History	History	45(5%)	123
Blogs followed on Reader	Reader	139(15%)	14
Address book	Contacts	766(87%)	189
Maps search history	Maps	696(79%)	22
Default address on Maps	Maps	52(5%)	1
Financial portfolio	Portfolio	11(1%)	8
First/Last name	Maps profile	661(75%)	1
Bookmarks	Bookmarks	236(27%)	79

(c) Aggregate Information analyzed from 872 Tor users.

Table 3: Results from the three experiments.

4.2 Estimating Historiographer's accuracy

Volunteers. In order to evaluate the extent of potential leakage of private information from Google web searches, we turned to volunteers. It would have been otherwise impossible to conduct our study on uninformed users without incurring legal and ethical issues. We aimed at evaluating the accuracy of the Historiographer at reconstructing web histories. To this end, we "attacked" the accounts of 10 volunteers using our software and measured its accuracy. The performance of the Historiographer at reconstructing search histories can be measured in terms of *recall*. For every user u, we call H the set of entries in u's history, H_c the subset of searches whose results were clicked by u, and S the set of entries reconstructed from suggestions. We denote $n_h = |H|$, $n_c = |H_c|$ and $n_s = |S|$. Since suggestions are only given for "clicked" queries, the recall R of our reconstruction algorithm can be measured as the ratio $R = \frac{n_s}{n_c}$. Results are reflected in Table 3b. The Historiographer reconstructs a significant portion of a user's history, with a mean recall of 0.65. The mean number of requests per user to reconstruct the history was 440. Since users are kept signed-in for two weeks, these requests can be made at a low pace to increase stealthiness. For instance, an attacker could issue a request every hour and still expect to retrieve 65% of the "clicked" queries. Also, the

recall can arbitrarily be increased by increasing the number of requests. On average, with about 2000 requests, we can obtain a mean recall of 0.81. The mean recall lowers to 0.34 when considering the ratio of reconstructed entries over the complete set H. Recall that the Historiographer can only recover "clicked" queries, although a complete history typically contains more information and additionally stores the time and the frequency of searches. We argue that only recovering "clicked" queries is not a tremendous limitation. When inspecting volunteers' history, we noticed that "clicked" queries are often corrections of generic or misspelled queries. A more accurate analysis of this phenomenon is left for future work. Note also that the Google's algorithm producing keyword suggestions is based on several parameters, such as dates and frequencies of searches and visited web sites. Therefore, we believe that the accuracy and the amount of information that can be retrieved by the Historiographer could be further improved with a deeper understanding of the underlying algorithm. On the other hand, it appears that the likelihood that an entry in the history is returned as a suggestion decreases over time, which could negatively affect the recall for older entries.

AOL Dataset. Next, we tested our attack on a wider sample. We used the anonymized query dataset released by AOL in 2006, containing 20 million searched made by $650,000$ users. From the dataset, we constructed the search history of each user. Then, simulating the search suggestions fed by Google drawing from the histories, we estimated the recall of our reconstruction technique. The mean recall was 0.64, an accuracy similar to that obtained for the volunteers.

4.3 Additional Information Leakage via Session Hijacking

As mentioned above, in addition to the Historiographer, an attacker can hijack a user's session to access several Google services. This section evaluates the extent of the information leaked. We ran our software for a week on a Tor exit node, and we analyzed 872 Google accounts. We stress that our software only generated aggregate data automatically and discarded the information immediately. Note that we used Tor only as a way to collect anonymized network traces. This cannot, by any means, be considered as an attack against Tor. In fact, even considering a malicious Tor exit node, the attacks can be prevented by using the appropriate tool configuration to block cookies transmitted over HTTP. (For more information, we refer to [4, 5]). However, we point out that a significant number of users are not aware of the dangers. In fact, they authenticate to Google while connected in Tor and do not block HTTP cookies, thus endangering their anonymity and privacy to potential malicious Tor exit nodes. In fact, a malicious entity could set up a Tor exit node to hijack cookies and reconstruct search histories. The security design underlying the Tor network guarantees that the malicious Tor exit node, although potentially able to access unencrypted traffic, is not able to learn the origin of such traffic. However, it may take the malicious node just one Google `SID` cookie to reconstruct a user's search history, the searched locations, the default location, etc., thus significantly increasing the probability of identifying a user. Additional example applications include RIAA tracking users that ever searched—although connected into Tor—for torrent files related to unlicensed material.

Session Hijacking Attack. By means of session hijacking, we tried to access the following information: locations searched on Maps (along with the "default location",

when available); blogs followed on Reader; full Web History (when accessible without re-entering credentials); finance portfolio; bookmarks. For each of them, we counted the number of entries retrieved and reported the mean over the 872 accounts. Table 3c summarizes the obtained results. We point out that for 5% of the accounts, we accessed the Web History page without being asked to re-enter credentials (simply replaying the `SID` cookie). We stress that the session hijack had a significant success rate for many popular services. For instance, we retrieved 79% of the searched locations on Maps and the 87% of address books (Contacts). Also, we were able to retrieve the first and last name associated to the account in 75% of cases. Unfortunately, *these numbers translate into a significant amount of personal (and identifying) information leaked through session hijacking*. Notably, the information collected from the Maps service was composed of maps queries coming from the histories of the users. Similarly to history suggestions, users that access Maps are presented with entries that come from the locations they previously searched for. Differently from search suggestions, Maps suggestions are not the result of an prefix based Ajax query to a remote Google server. Instead, for signed-in users, the page at `maps.google.com` includes a Javascript array that includes *a*ll previous searches. Accessing this information only requires retrieving the web page once and does not require the use of the Historiographer. The provided information is very detailed and includes: the exact location searched (address:), the time, in seconds since the Epoch, it was searched (created:) and the number of times the location was searched (count:). The information collected this way is of the same kind of the one collected by the Historiographer but referred to maps searches instead of generic web searches. However, the specifics of the design of Maps suggestions make the attack on this service much easier. We can only speculate on the reasons behind such a design. One could be that, since Maps history is relatively small in mean size 3c, it is more efficient to send all the information at once, rather than relying on multiple Ajax requests and replies. Whatever the reasons, this design makes location information stored on Google more vulnerable to session hijacking than search history.

4.4 Web history and Smart Phones

With the increasing number of smart phones users, search history is likely to be strongly correlated with users' location of the users. We noticed that Google maintains a separated Web History when the search page is accessed from an iPhone. Such a history has a less strict access control policy. Similarly to Google Maps, the whole search history is sent as a Javascript list embedded in the page. Supposedly, this information is presented only when using the iPhone. However, one just needs to set the appropriate user agent string when accessing Google (for example through the User Agent Switcher Firefox extension [7]. Then, replaying the `SID` cookie, the whole Web History becomes accessible, with a single page access. We tested this strategy on the set of volunteers. We were able to retrieve their iPhone search history from a regular PC by switching the PC's browser user agent to an iPhone user agent, and hijacking the victims' `SID` cookies.

5 Possible Countermeasures

The vulnerability targeted by the Historiographer is difficult to address because of the complexity and scale of the Google architecture, as well as the performance and usability requirements. However, we discuss some possible countermeasures. For instance, users could take the following precautions, simultaneously: (i) always log out from any Google service when performing a search, (ii) disable the Web History service, and (iii) disable personalization from anonymous cookies or always delete Google cookies, similarly to what is suggested by the Electronic Frontier Foundation On the other hand, Google could either: (i) discontinue the Personalized Search service, or (ii) let the users choose to enforce HTTPS for web searches (for instance, by clicking on a special link when surfing from *insecure* networks) and trade off speed with privacy. However, one can argue that solutions preventing personalized searches may degrade the service, whereas the use of HTTPS on Web Search[7] may be too expensive to put in place. Evidence of this is given by the impossibility of accessing Google search page via HTTPS and by the concerns already expressed by Google regarding the performance of using HTTPS for Gmail [9].

Compartmentalized Searches. We propose an additional mitigation technique that would allow to keep the Personalized Search service. Specifically, we propose that Google could keep separate histories based on the networks from which user's searches originate. Then, it can provide different search suggestions (and personalized results) based on different locations. We imagine an extension to the `google.com/history` web page to allow a user to configure such locations and the privacy settings related to them. Although this would not solve all possible information leakage, it would compartmentalize user's private information: Consider for instance an employee reluctant to reveal personal information to her employer (e.g., that she is looking for another job). Fearing that her navigation within the company network is monitored, she might avoid accessing potentially "compromising" information. If she signs in Google from the company network, however, her search history —containing for instance "compromising" searches made from home—(and more) can be leaked.

Binding authentication cookies to IP addresses. Several web sites, e.g., LiveJournal [2], allow user agents to bind the authentication cookies to the current IP address. In other words, the server does not accept an authentication cookie that originates from a different IP address. However, this technique is not always enforced due to drawbacks on the usability of the service. For example, "mobile" users, whose IP address often changes, would be forced to frequently re-enter their credentials. However, depending on the network configuration, binding cookies to IP addresses could not be enough to prevent session hijack. For instance, an attacker operating on a local network could succeed by poisoning the ARP table on the local Ethernet switch. Note also that at the moment Google allows a single account to be signed-in from multiple locations and with multiple IP addresses (although some services such as Gmail display the number of simultaneous connections at the bottom of the page).

[7] Note that adopting HTTPS only for the Web History web page would not prevent the Historiographer, but only the access to the page.

6 Related Work

To the best of our knowledge, this work is the first to focus on the private information leaked from web searches to third parties. In the following, we present the most relevant work to several concepts and tools that we use.

Session hijacking. Since their early appearances, the use of cookies to maintain authenticated sessions has lead the way to *session hijacking* attacks (see for instance [19]). These attacks are quite simple: an attacker monitoring network traffic may sniff an authentication cookie and replay it to impersonate another user. For this reason, sensitive web applications should always employ secure cookies, i.e., authentication cookies that are only transmitted over encrypted channels. However, this simple countermeasure is not always effective. For instance, in 2008 the Cookiemonster attack [24] highlighted vulnerabilities derived from an improper mixed support of secure and insecure connections. In response to this work, Google sett HTTPS in Gmail by default [26]. Although this attack—as well as simple session hijacking—could not be be used to hijack the Web History, it is an interesting example of vulnerabilities in web applications that do not *properly* provide mixed HTTP/HTTPS support.

Privacy Threats. Recent work has discussed potential privacy threats related to cloud service providers. For instance, [14] discussed potential threats and countermeasures associated with many forms of web activity—focusing on Google—related to the information collected by service providers. However, as opposed to our work, this paper focuses on the privacy threats against the service provider. Another direction was taken in [16, 15] to assess user perception on alleged privacy threats by interviewing users. Among the other interesting results, it has been shown that more than 80% of users admitted to having conducted searches for information they would not want disclosed to their current or future employer. Finally, independently of our work, it was recently shown that popular online applications may leak private data to a network eavesdropper even over encrypted web connections [13]. In particular, an adversary could exploit the autocompletion mechanism of popular search engines to infer the victim's search queries. When a user types the first letter in a search query, the search engine sends that character to the server, and the server replies with a list of suggested completions. As the size of that list depends on the character typed, an attacker can deduce which letter was typed. When the second letter is entered, another request is sent to the server, and another encrypted response sent back to the client, which allows the attacker to infer the second character; and so on. As a result, the attacker guesses the search query, despite the communication is encrypted. This result is complementary to our work: It allows recovering search requests over encrypted channels. However, the attack does not work if the victim is logged-in and the suggestions received are personalized. In contrast, our attack retrieves parts of the victim's search history using these personalized suggestions, although our attack does not work over encrypted links.

Limiting personal information disclosure. Several techniques have been proposed to avoid user profiling and reduce the amount of information potentially leaked. For instance, the Firefox extension Trackemenot [20] periodically issues randomized search-queries to search engines to populate a user's search history with (non-clicked) queries

and hide real queries. However, this would not prevent the Historiographer from retrieving "clicked" queries from the history and retrieve sensitive information.

7 Discussion

This paper has presented a study of the private information disclosed to third parties from web searches. We showed that the well known session hijacking attack is still applicable to many Google services, and we presented the Historiographer, an attack that reconstructs Google's search histories from simple web searches. We have validated our technique through a large-scale experimental analysis.

We argue that solutions should be quickly deployed to protect users against these two types of attacks. The session hijacking attack is harmful not only because it allows an attacker to collect a lot of private information, including sometimes the search history, but also because it can be exploited to add potentially *compromising* entries [25]. It can also be used to modify the search results displayed to the victim. In fact, Google allows to delete or promote—i.e., show as first—results using a button associated to them. An adversary hijacking a session cookie can perform searches on the victim's behalf and influence the results corresponding to these searches as she wishes. For instance, this attack can be a powerful tool for censorship, as it can be used to remove or promote some pages displayed after a Google search.

The Historiographer can be used to reconstruct part of the Web History, when, for example, the simple session hijacking attack is not applicable. In addition, it can be used as an oracle to perform *targeted checks*, e.g., to verify the existence in the search history of specific keywords. The Historiographer is an *amplification* attack, and therefore is much more powerful than a simple eavesdropping attack: It not only allows an attacker to eavesdrop on the victim's search requests, but also allows him to retrieve the victim's *previous* search requests, *possibly performed from different networks and even different computers*. Also, the Historiographer is *non-destructive*, i.e., it does not affect user data. The number of potential victims is very high, since any signed-in user is at risk as soon as she issues a single Google search request from an unencrypted network, such as an open wireless network at an airport or a cafe.

These attacks deserve serious attention since Web Histories contain sensitive information. Any information leaked from Web search histories could endanger user privacy. Information retrieved from the search history could also be combined with other publicly available data, such as that published on social networks to accurately profile and/or identify target users. Furthermore, since the Historiographer also works for Google searches performed from mobile devices and such searches contain also localized results, one could use location-based services to also track users' movements and locations.

Although the Historiographer builds on features specific to the Google architecture, our goal is not to attack Google nor any particular service provider. Instead, we highlight the general problem of protecting the privacy of sensitive data when using a mixed architecture with both secure and insecure connections. As mentioned in [8], Google is not the only provider which leaves its customers vulnerable to data theft and account hijacking. As a matter of fact, the Bing search engine recently added a similar function-

ality to Personalized Suggestions. Users receive suggestions based on their previous searches and they can access the full search history [6]. Differently from Google, Bing only uses anonymous cookies for this purpose and stores the search history only up to 29 days. However, in Bing the full history is accessible via a simple session hijacking. We defer to future work a complete analysis of Bing and other search engines.

8 Afterword

While this paper was under submission (March 2010) we disclosed it to Google to allow them to react to it. Google has been very responsive to our research and has taken some actions to fix some of the highlighted issues. After receiving a preliminary report, Google temporarily disabled the personalized suggestions (note, however, that they were never disabled on smart phones), and switched the Web History and Bookmark services to HTTPS (thus, preventing session hijacking on these services) [8]. Later on, Google countered the Historiographer attack by encrypting back-end server requests associated with the personalized Maps and Search suggestion services. We provide a detailed description and discussion on the possible shortcomings of this solution in [11]. We also detail a possible way the Historiographer could work against Google's solution, albeit with a different and slightly more powerful attacker. It is also noteworthy that the proposed solutions do not prevent potential leakage resulting from personalized results (see Section 4.3). Furthermore, as of today (beginning of May), searches conducted from smart phones are still vulnerable (see Section 4.4) and session hijacking is still effective on the following services: Reader, Contacts and Portfolio.

References

1. Tor: anonymity online. http://www.torproject.org/.
2. LiveJournal. http://www.livejournal.com/support/faqbrowse.bml?faqid=135, Retrieved February 2010.
3. Personalized search for everyone. http://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html, Retrieved February 2010.
4. Privoxy. http://www.privoxy.org/, Retrieved February 2010.
5. Torbutton. http://www.torproject.org/torbutton/, Retrieved February 2010.
6. Bing Autosuggest. http://bit.ly/bxPk9g, Retrieved March 2010.
7. User Agent Switcher firefox plugin. https://addons.mozilla.org/en-US/firefox/addon/59, Retrieved March 2010.
8. A. Acquisti et al. Ensuring adequate security in Google's cloud based services. http://www.wired.com/images_blogs/threatlevel/2009/06/google-letter-final2.pdf, 2009.
9. A. Whitten (Google). HTTPS security for web application. http://googleonlinesecurity.blogspot.com/2009/06/https-security-for-web-applications.html, 2009.
10. M. Barbaro and T. Zeller. A face is exposed for AOL searcher no. 4417749. *New York Times*, 9:2008, 2006.

[8] For more details on the ongoing development of this project, refer to http://planete.inrialpes.fr/projects/private-information-disclosure-from-web-searches.

11. C. Castelluccia, E. D. Cristofaro, and D. Perito. The historiographer reloaded. Technical report, INRIA, May 2010.
12. R. Cellan-Jones. Web creator rejects net tracking. http://news.bbc.co.uk/2/hi/7299875.stm, 2008.
13. S. Chen, R. Wang, X. Wang, and K. Zhang. Side-Channel Leaks in Web Applications: a Reality Today, a Challenge Tomorrow. In *IEEE Security and Privacy Symposium'10*, 2010.
14. G. Conti. Googling considered harmful. In *Workshop on New Security Paradigms*, pages 76–85, 2006.
15. G. Conti. *Googling Security: How Much Does Google Know About You?* Addison-Wesley, 2009.
16. G. Conti and E. Sobiesk. An honest man has nothing to fear: user perceptions on web-based information disclosure. In *SOUPS'07*, pages 112–121, 2007.
17. C. Farkas and S. Jajodia. The inference problem: a survey. *ACM SIGKDD Explorations Newsletter*, 4(2):6–11, 2002.
18. J. Ginsberg, M. Mohebbi, R. Patel, L. Brammer, M. Smolinski, and L. Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012–1014, 2008.
19. R. Graham. SideJacking with Hamster. http://erratasec.blogspot.com/2007/08/sidejacking-with-hamster_05.html, 2007.
20. D. Howe and H. Nissenbaum. TrackMeNot. http://mrl.nyu.edu/~dhowe/trackmenot/, 2008.
21. Jesse James Garrett. Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php, 2005.
22. D. Kristol and L. Montulli. RFC2109: HTTP State Management Mechanism. *IETF*, 1997.
23. D. Kristol and L. Montulli. RFC2965: HTTP State Management Mechanism. *IETF*, 2000.
24. M. Perry. CookieMonster: Cookie Hijacking. http://fscked.org/projects/cookiemonster, 2008.
25. J. Robertson. Internet Virus Frames Users For Child Porn. http://www.huffingtonpost.com/2009/11/09/internet-virus-frames-use_n_350426.html, 2009.
26. S. Schillace. Default https access for Gmail. http://gmailblog.blogspot.com/2010/01/default-https-access-for-gmail.html, Retrieved February 2010.