Vanity Fair: Privacy in Querylog Bundles

Rosie Jones Ravi Kumar Bo Pang Andrew Tomkins
Yahoo! Research
701 First Ave
Sunnyvale, CA 94089.
{jonesr,ravikumar,bopang,atomkins}@yahoo-inc.com

ABSTRACT

A recently proposed approach to address privacy concerns in storing web search querylogs is bundling logs of multiple users together. In this work we investigate privacy leaks that are possible even when querylogs from multiple users are bundled together, without any user or session identifiers. We begin by quantifying users' propensity to issue own-name *vanity* queries and geographically revealing queries. We show that these propensities interact badly with two forms of vulnerabilities in the bundling scheme. First, *structural vulnerabilities* arise due to properties of the heavy tail of the user search frequency distribution, or the distribution of locations that appear within a user's queries. These heavy tails may cause a user to appear visibly different from other users in the same bundle. Second, we demonstrate *analytical vulnerabilities* based on the ability to separate the queries in a bundle into threads corresponding to individual users. These vulnerabilities raise privacy issues suggesting that bundling must be handled with great care.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous

General Terms

Algorithms, Experimentation, Measurements

Keywords

Querylogs, privacy

1. INTRODUCTION

There is much recent interest in the problem of obscuring search log data so that the privacy of individual users is protected, but the data remains useful for as wide a range of purposes as possible. This problem arises in two contexts that are already the subject of active debate. First, search engine companies seek to preserve search log data (for many reasons, including legal restrictions, various types of relevance improvements, and fraud detection), but wish to obfuscate the logs to meet requirements without compromising user privacy. Second, search engine companies wish to share search logs with academics for research purposes, in a way that does not compromise privacy.

In general, anonymizing a search log represents a tradeoff between privacy and fidelity. On one extreme, removing all session information is clearly a big step towards ensuring stronger privacy. However, any detailed user modeling becomes infeasible in such an environment.

Recently, we have seen the proposal of a new technique for providing partial session information: *bundling*, in which user sessions are partitioned into bundles, and each query is associated only with a bundle identifier, rather than a full user identifier.[1] We study the implications of bundling for user privacy, and show that there remain a range of concerns around the technique. Our goal is *not* to propose any specific anonymization scheme, but rather to examine the aspects of bundling as a privacy preserving mechanism.

1.1 Our contributions

Our study is based on a large querylog from Yahoo!. The notion of bundling we use is very liberal: for each query, completely disregard the cookie information and mask the last few bits of the IP address. In this context, the main contributions of this paper are listed below.

Vanity queries. First, we study how effectively a user session can be mapped to either the name of the user generating the session, or the geographic location of the user generating the session. We show that over a 70-day period, as much as 30% of users issue a query for their own name, roughly matching previous surveys of self-reported vanity search behavior. And fully 50% of users issue a query that references a location that shares three or more digits of ZIP code with the user's home location; we also give details on more accurate geographic queries. Based on this analysis, we conclude that an attacker with access to a bundled search log can realistically identify the bundle containing the session of a particular user, or conversely, given the queries in a bundle, she can identify the names or locations of many users in it.

Structural vulnerabilities. Second, we study *structural vulnerabilities*, which arise due to the nature of bundling without any sophisticated analysis of the query content. For example, any user

[1]This technique came into focus in December of 2007 when a major search engine released to the media a series of responses to 24 privacy-related questions posed by Representative Joe Barton (available at `http://searchengineland.com/pdfs/071222-barton.pdf`). In these responses, the search engine proposed that it would employ bundling to anonymize log data. Specifically, it would "anonymize the cookie ID and the last octet (typically one to three digits) of the IP address associated with search queries after 18 months." No subsequent announcement has claimed that the proposed policy is in effect.

who searches more frequently than the average user will contribute a significant fraction of the queries in a bundle, especially if the bundle is small. For bundles created by masking the low-order 8 bits of IP address, we show that more than half a percent of bundles contain a user who issues in excess of 90% of the queries in that bundle.

Similarly, we may consider a more nuanced form of structural vulnerability in which a user queries for "Bend, OR HIV clinic." If the fraction of total users who live in Bend, OR is small compared to the size of a bundle, and the attacker has identified a vanity query consisting of the name of an individual known to live in Bend, then significant information has been leaked—it is highly likely that this user issued the query for the sensitive HIV clinic topic. We characterize the degree of geographic sensitivity of this form.

Analytical vulnerabilities. Finally, we study *analytical vulnerabilities*, in which a careful analysis of the query content allows the bundle to be teased back apart into individual query streams. We explore a variety of approaches to this un-bundling, and show that it is possible to cluster queries into users effectively. In one metric we study, we show that bundles created by truncating the lower octet of the user's IP address can be clustered back into users such that two conditions hold. Fix a query submitted by a given user. First, the cluster containing the query contains on average 60% of the total queries by that user. And second, 60% of queries in the cluster were issued by the given user. We conclude that significant analytical vulnerabilities exist.

Extracting user names. Of independent interest, our study of vanity queries relies on an understanding of the true name of a user. We study the generation of this true name from user profiles and from userids, and as a corollary, we characterize how often information on a person's name is available from these sources.

1.2 Implications for privacy

From an attacker's point of view, if she targets a specific heavy user by knowing his/her name or userid, then she can benefit from the presence of vanity queries to narrow down to the actual bundle that contains the specific heavy user. Our study establishes that vanity queries, besides being abundant, can actually be used to infer the presence or absence of a particular user in a given bundle. Once the attacker focuses on the specific bundle, the structural vulnerability suggests that bundling is virtually ineffective against reasonably heavy users or users who issue unique geographic queries; our analysis also shows that there are sufficiently many structurally vulnerable bundles. Furthermore, the attacker can use the analytical vulnerability techniques to tease out individual user sessions with reasonable accuracy. Thus, the privacy of the user can be compromised, even with our generous notion of bundling.

We therefore conclude that, while bundling represents an interesting potential technique in the privacy toolkit, there remain significant difficulties in employing the technique in practice.

1.3 Organization

The remainder of the paper proceeds as follows. Section 2 covers related work, and Section 3 describes the data we consider. In Section 4 we consider vanity queries and geographically-revealing queries, and the risks these entail. In particular, Section 4.1 studies the extraction of a user's name from profile information and by parsing the userid itself. Finally, Section 5 describes the structural vulnerabilities of bundles, and Section 6 addresses the analytical vulnerability associated with the decomposition of a bundle to recreate the individual user sessions. Section 7 contains concluding remarks.

2. RELATED WORK

The related work falls into four kinds of categories: privacy in querylogs, vanity queries in user search behavior, identifying similar queries and detecting session boundaries, and privacy in general.

Privacy in querylogs. Kumar et al show that anonymizing querylogs by hashing individual tokens does not prevent the decryption of that hash by an attacker with access to another external source of web querylogs [9]. Adar discusses specific schemes for anonymizing querylogs sessions, by removing unique queries, hashing rare queries, and fragmenting into shorter sessions, as well as fragmenting users into topic profiles [1].

Jones et al study the use of simple classifiers to map a sequence of queries into the personal information of the user, such as gender, age, and location [7]. They show that, first, it is easy to build classifiers that perform reasonably well, and second, combining small pieces of information about search terms can be potentially used to identify the session of the user.

Vanity queries. Gates and Whalen found in a survey that most of their users had issued vanity queries in the past [5]. Soghian discusses how existing tools for providing some privacy, such as *TrackMeNot* and *Tor* do not protect the user who issues vanity queries [16]. Almost 25% of surveyed users say they search for themselves online, when we include searches for full name, email address, instant messenger ID, or MySpace address [3]. In Section 4 we empirically analyze the occurrences of vanity queries in web search logs over a period of 70 days.

Identifying similar queries, session boundaries. Similar query identification using web search reformulations have been examined by Jones et al [8]. Queries that are similar in this way will help join together same-session queries from the same user, as we will examine briefly in Section 6.4. Other methods of identifying similar queries are also applicable here. Rey and Jhala look at models of query rewrites in order to distinguish web search query reformulations from queries cooccurring because of user interest patterns [14]. These types of associative matches are things that may help join together different sessions from the same user, as we will examine briefly in Section 6.3. Web searchers also tend to repeat their queries over longer periods of time, especially for navigational or bookmark type queries [17]. These will also help re-identify different sessions for the same user. We do not explicitly model this case, but all edit-distance-based similarity measures will capture this type of similarity. Jansen et al also looked at the use of time and query reformulation as a method for identifying queries from the same session [6].

Privacy. Samarati and Sweeney define a formal notion of privacy they call k*-anonymity* [15]. In this model, one proves that no user can be shown to belong to a group of fewer than k candidates. In our work, we look at whether placing user queries in buckets with large number of users provides in practice a property similar to k-anonymity where k is the number of users in the bundle; we show that without further work, this is not the case.

Novak et al use content similarity to disambiguate and anti-alias users who use multiple pseudonyms to create several identities of themselves [12]. Our work contrasts in that we assume each user has a unique real-world identity, and we are characterizing how easy it is to rediscover this identity. The use of multiple pseudonyms during web search may also be detectable using our approach.

Frankowski et al show that even when users' data is anonymized, their public statements about rare interests can be joined with anonymized data to reveal their identity [4]. This situation has a clear parallel with querylog data, in which users may reveal innocuous interests (types of cooking, sports, movies) on blogs and other forums, and the conjunction of queries on these topics may be used to identify the user.

Backstrom et al look at the problem of identifying users in an anonymized social network [2]. This problem differs in that there is structural information available in the form of links between anonymous users, but no other content. In addition, if users are able to identify themselves in the network, they can then learn more about those linked to them. In the querylog setting, being able to identify one's own queries does not in general help identify other users.

3. DATA

In this section we describe querylog data and user profiles used for our experiments, as well as some data analysis of the distribution of information in querylogs. In all our analysis, we use a subset of daily queries from Yahoo!.

3.1 Querylog Q1

Our first data set is created to study vanity queries. We use querylogs from users issuing at least 100 queries over a 70 day period in 2006. Each query has the cookie information of the browser issuing the query. From many of the cookies, a userid can be reliably extracted. We use these logs in conjunction with the publicly available user profiles for analyzing vanity queries. We refer to these querylogs as Q1.

In order to study vanity queries, Q1 is restricted to queries from cookies such that (a) significant number of queries (at least 100 in our case) are associated with this cookie and (b) its corresponding userid is found in the set of publicly available user profiles. This is obviously a biased sampling of the querylog and cannot serve as the appropriate data set to study characteristics of the bundling scheme. This necessitates the use of a second data set.

3.2 Querylog Q2 and bundles of users

For our second data set, we use a week worth of querylogs chosen from April 9 to April 15th 2007. Each entry has the terms comprising the query, the browser cookie, the originating IP address, and the time stamp. We refer to these querylogs as Q2.

We now describe a study of the basic statistics of creating bundles from user queries. We study a set of just over 100M queries (108,779,740), a subset of the data from querylog Q2, dated April 11, 2007. There are different ways of creating bundles: by hashing the cookie associated with the user or by masking the IP address from which the query originated or both. The first approach gives a more uniform distribution since the cookie is often a reasonably random string; the second may be non-uniform since certain IP addresses might correspond to proxies, ISPs, or other multi-user locations. To study the most difficult case, we do the following to create bundles: we

(1) completely disregard the cookie information, and

(2) mask the IP address, and in particular mask the low-order bits of the IP address.

Note that retaining partial cookie information, such as a hash, would only make this problem easier.

Table 1 shows the basic statistics of the bundles. Since our main focus is to target heavy users, we ignore bundles that are very small. We only consider bundles with at least 100 queries, corresponding to the second column of the table.

Table 2 shows the same statistics that result from masking different numbers of bits in the IP address. From the table we can see that when we mask 8 to 16 bits of the IP address, the average number of users per bundle grows around five times for every four additional bits masked[2].

	All	$>$ 100 queries
Avg. # queries per bundle	86.8	294
Avg. # users per bundle	17.8	57.1
# bundles	1,265,619	303,816

Table 1: Basic statistics for bundles created by masking low-order 8 bits of IP address: all bundles, and only bundles that have at least 100 queries.

# bits masked	8	12	16	24
Avg. # queries per bundle	294	1.2K	6.9K	779K
Avg. # users per bundle	57	248	1.3K	153K
# bundles	303K	85K	15K	141

Table 2: Basic statistics for bundles created by masking low-order bits of IP address, for bundles with at least 100 queries.

4. VANITY QUERIES

Vanity queries are a well-known web phenomenon in which a user issues a query for his or her own name, usually for one of two reasons. First, a user may issue a vanity query in order to assess the ranking of the user's own homepage, blog, or other content. And second, the user may issue a vanity query to discover whether other parties have mentioned the user. If vanity searches are common, it may be possible to associate a query or query session with a user merely by assuming that a query for a person who is not a celebrity is quite likely to contain the name of the user submitting the query. Implications of the popularity of vanity queries are two folds. First, only hashing the cookies (thereby obfuscating the userids) won't suffice: attackers may be able to identify sessions (the ones where vanity queries were issued as well as those with the same obfuscated cookie) for a given user, or conversely identify the user through vanity search in a given thread of queries. Second, for the bundling scheme described in Section 3, "hunting" attackers can use vanity queries to narrow the search for the right bundle from, say, 300K candidates to reasonably few candidates.

Furthermore, as we will discuss in Section 3.2, users also submit queries that contain geographic information, and for natural reasons, such queries will often reference geographic locations that are proximate to the user. For example, a user might enter the query "pizza 95014" because the user lives in postal code 95014 and wishes to find a pizza restaurant. While such queries are not typically the result of vanity *per se*, we will nonetheless refer to them as geographic vanity queries.

The goal of this section is to understand what fraction of users issue vanity queries (names and geographic location), and how effectively these queries can be used to uncover information about the user. To study this, we use querylog Q1. In order to study vanity queries, we first need to extract the user names from the userids.

4.1 Extracting names

In this section we study the process of extracting the true name of a user from the userid. We will use such a process in our experiments, for instance to study the number of users who query for their

[2]Masking the last 8 bits of an IP address maps the IP address to a "class C subnet," which may be associated with a very specific location. So there may be correlation between the addresses of users in each bundle created this way. We do not take advantage of this in our analysis.

own name. At the same time, however, we will use our extraction results to argue that attackers can readily access the true name of some fraction of users in a representative online environment.

There are two natural approaches to converting a userid into names. The first is to employ some external data source that might list the name explicitly, such as the profile page associated with the userid, or the user's signature, or some analysis of the text. We will examine specifically the extraction of names from profile pages. The second natural approach is to extract the names directly from the userid itself. We now explore the coverage of these two approaches, both singly and together.

Extraction from profile. In many websites, each such userid `<u>` has a corresponding public profile page accessible to everyone. To give an example, `http://profiles.yahoo.com/<u>` is a page in the Yahoo! member directory and is semi-structured with fields corresponding to real name, nickname, location, age, marital status, sex, occupation, home page, etc., of the user `<u>`. Not all users necessarily volunteer all the information on a profile page, and some of the information might be deliberately misrepresented.

Despite these caveats, we crawled the profile pages of 744K userids. For each profile page we crawled, we extracted the real name, if present, and parsed it to obtain a firstname-lastname pair (sometimes, including middle names). For our purposes, we do not distinguish between the first and last names and treat it as a set. This method can yield from zero to many names from a single userid.

Extraction from userid. Next, we consider extracting names directly from a parse of the userid. Our intuition in considering this scheme is that many users create ids that contain some morphing of their own name. Thus, we ask the following question: can we extract in a principled manner the names that are contained in a userid?

We proceed by performing a simple segmentation of the name, as follows. The US Census Bureau publishes the 100K most popular firstname-lastname pairs; we gather this data to obtain a list L of 91,909 names (either first or last). Next, we clean the userid by converting every character to lowercase and removing non-alphabetic characters, if any. We then construct a graph based on this cleaned userid. Given a userid of the form $x_1, \ldots, x_n$ where each x_i is an alphanumeric character, we construct a $(n+1)$-node graph G in the following way. The nodes of G correspond to the characters $x_1, \ldots, x_n$, plus a final state x_{n+1}. We place a directed edge (x_i, x_{j+1}) in the graph if one of the following conditions hold:

(1) the substring $x_i \ldots x_j$ is a valid name, according to the list L;

(2) $j = i$, i.e., we allow single-letter names, to account for potential initials.

An example graph constructed for the userid `jackson_andrew` is shown in Figure 4.1. It is clear that the graph has captured many possible names, including the correct ones: `{jackson, andrew}`.

Next, we find the shortest path in this unweighted graph from x_1 to x_{n+1}; this can be accomplished in an efficient manner using breadth-first search. The intuition is that we want to match valid names that are as long as possible, but allowing arbitrary substrings gluing them together. It is clear that every non-consecutive edge in this path of the form $x_i, \ldots, x_j$ where $j > i+1$ corresponds to a valid name according to L. For the userid, we collect the set of names corresponding to all such non-consecutive edges. Note that this can yield from zero to many names from a single userid.

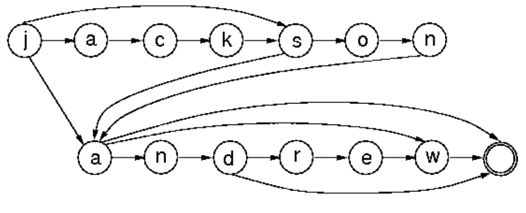

Figure 1: The graph constructed for the userid `jackson_andrew`.

Method	Condition	$k=1$	$k=2$
P_k	$\lvert P(u)\rvert \geq k$	23.36	9.49
S_k	$\lvert S(u)\rvert \geq k$	88.57	41.15
$(P \vee S)_k$	$\lvert P(u)\rvert \geq k \vee \lvert S(u)\rvert \geq k$	89.40	48.43
$(P \wedge S)_k$	$\lvert P(u)\rvert \geq k \wedge \lvert S(u)\rvert \geq k$	19.08	6.32
$(P \cap S)_k$	$\lvert P(u) \cap S(u)\rvert \geq k$	5.64	0.93

Table 3: % userids exhibiting various name extraction conditions. $P(u)$ denotes the set of names found on the profile page for u, and $S(u)$ denotes the set of names found by parsing u.

4.1.1 *Coverage performance*

We now measure the coverage of both schemes. For a userid u, let $P(u)$ denote the set of names obtained using the profile page and let $S(u)$ denote the set of names obtained using the userid segmentation. Since we treat these as sets, it is meaningful to consider their union and intersection, and to place thresholds on their cardinality, in order to obtain a variety of methods for extracting a final set of names from a userid. We define the following methods:

P_k : Returns the set $P(u)$ of names, as long as there are at least k elements in the set $P(u)$.

S_k : Returns the set $S(u)$ of names, as long as there are at least k elements in the set $S(u)$

$(P \vee S)_k$: Returns the set $P(u) \cup S(u)$ of names, as long as either $P(u)$ or $S(u)$ has at least k elements.

$(P \wedge S)_k$: Returns the set $P(u) \cup S(u)$ of names, as long as both $P(u)$ and $S(u)$ have at least k elements.

$(P \cap S)_k$: Returns the set $P(u) \cap S(u)$ of names, as long as the intersection $P(u) \cap S(u)$ is of size at least k.

Table 3 shows how many of the 744K userids have names extracted by each of these methods.

In many user registration systems, users are not required to enter a full name, so it is not surprising that we find even a single name on the profile page for under one quarter of the users. The userid, on the other hand, contains at least one name for almost 90% of users, and two names for over 40%. This confirms our intuition that userids based on first name are extremely common, and those based on two names are in fact quite frequent. If we combine information, we find that two names can be extracted from the combination of profile page and userid in almost 50% of cases.

Comparing the last two lines of the table, we observe that if a name appears in both profile and userid, then 30% of the time, the same name appears in both. Likewise, if we successfully extract at least two names from each of profile and userid, then more than 5% of the time we match exactly on two names. Finally, for around 1% of the userids overall, we are able to extract the same two names from two distinct sources, giving us high confidence that we have

Name extr. method	any-match		all-match	
	absolute	relative	absolute	relative
P_1	2.95	12.6	0.51	2.18
P_2	1.69	17.7	0.51	5.34
S_1	7.68	8.67	0.98	1.10
S_2	4.34	10.55	0.98	2.38
$(P \vee S)_1$	8.85	9.89	1.15	1.29
$(P \vee S)_2$	5.72	11.81	1.15	2.37
$(P \wedge S)_1$	2.50	13.10	0.33	1.73
$(P \wedge S)_2$	2.17	13.30	0.33	2.02
$(P \cap S)_1$	0.96	17.02	0.10	1.77
$(P \cap S)_2$	0.28	30.10	0.10	10.75

Table 4: % users issuing vanity queries according to a particular name extraction method and matching condition. Absolute numbers show fraction of total userids, and relative numbers show fraction of userids for which name information was successfully extracted by the specified method.

successfully identified the name of the user from the publicly available information.

4.2 Name vanity queries

We begin by studying the fraction of users who issue their own name as a query. To do this, we first use the name extraction methods described in Section 4.1 to get potential names for each userid u. Next, we examine the queries issued by userid u. Using a name recognizer that is built using the US Census Bureau data, we first restrict our attention to query terms that are people names. We then count the number of name queries for which at least one term in the extracted name is present (the *any-match* condition), and the number of name queries for which every term in the extracted name is present (the *all-match* condition).

Table 4 shows the fraction of users who issue name vanity queries for a given name extraction method, under either the any-match or all-match conditions.

It is clear from the table that a significant fraction of the users issue vanity queries. 5% of users for whom we extracted two names from the profile page issued a query containing those two names. And over 10% of users for whom we extracted the same two names, using both the profile and a parse of the userid, issued a query containing those two names. To the best of our knowledge, the above represents the first empirical and quantitative analysis of the popular belief regarding the prevalence of vanity queries on the web.

We now examine the characteristics of name vanity queries in more detail, with analysis of the implications in bundle-attack scenarios.

4.2.1 *Recovering user name from sessions*

In this section we consider an attacker who takes a particular session from a querylog and attempts to discover which user generated the session. We focus only on the users who issue name vanity queries and ask the following question: assuming users search for their own names, what is the rank of their names when placed among all name queries they issue? To do this, we once again extract the user names from userids using the name extraction methods of Section 4.1. We retain only name queries as spotted by our name recognizer. We compute the number of times a particular name query was issued by a user, and refer to this as the term frequency (tf) of the query. Since many name queries correspond to popular or celebrity names, we define an analog of inverse document frequency (idf) for a name, computed as the inverse of the logarithm of the number of times the query was issued by any user.

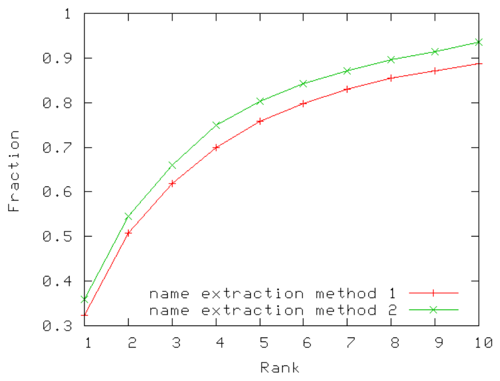

Figure 2: Recovering user name from sessions: recall curves for userid with valid names extracted according to 1 $(P \cap S)_1$ **and 2.** $(P \cap S)_2$**.**

Then, for each user, we rank that user's name queries according to the product of tf and idf scores, and check the rank at which the user's own name appears. Here, we restrict our attention to all-match. We compute the fraction of users whose names are identified among the top name queries they issue.

Figure 2 shows results for two name extraction methods (we observe similar recall curves for other name extraction methods). As shown in the figure, 90% of the users query their own name within the top ten of all name queries they issue (weighted by idf). We believe this is quantitative evidence not only of the popular belief that lots of people issue name vanity queries to the search engines, but also query their name quite often, and it is possible in many cases to study a user's query session and determine the name of the user.

4.2.2 *Recovering sessions for a given user*

In this section we consider an alternate attack methodology. We assume the attacker would like to discover the session of a particular known person, perhaps a neighbor, co-worker, or even a spouse. We therefore ask the reverse question: assuming users search for their own names, given a name, what is the rank of the true user among all users who queried for that name? We consider all names that correspond to at least one userid and accumulate all users who queried for that name at least once (using all-match). As before, we compute idf, but this time with respect to the number of name queries issued by a user; this is to discount users who issue many name queries. We then rank the users based on the product of tf and idf score, and as before we compute the recall position. We plot fraction of users as a function of the rank in Figure 3.

As shown in the figure, recall is extremely high: for both name extraction methods, the recall is between 72% and 81% at position 1, 90% at position 3, and more than 95% at position 10. In other words, given a name, it is highly possible to discover the session corresponding to that user. Thus, given a large collection of query bundles, for those users who have issued name vanity queries (as we noted earlier, this could represent a substantial portion of online users), it is possible to narrow down to a few candidate bundles by examining the name vanity queries.

4.3 Geographic vanity queries

Here, we analyze the fraction of users who issue queries that reference the user's own geographic location. To conduct this study, we make use of the profile data to obtain the geographic location for each user; our set of 744K reference users were chosen such that every profile contains a US postal code.

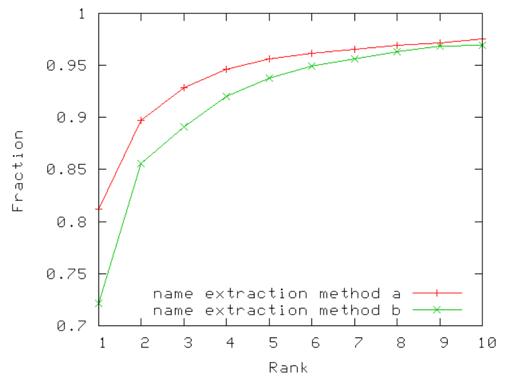

Figure 3: Recovering sessions for a given user: recall curves for name queries, with valid names extracted according to a. $(P \vee S)_2$ and b. $(P \cap S)_2$.

Zipcode Match	$\gamma = 0.1$ $k = 1$	$\gamma = 0.5$ $k = 1$	$\gamma = 0.5$ $k = 5$	$\gamma = 0.5$ $k = 10$
5	13.7	11.7	3.69	1.44
4	25.8	22.6	8.95	4.17
3	52.8	48.5	24.5	12.9

Table 5: % users issuing γ-geovanity queries.

For each query, we apply a black-box geographic classifier based on the internet locality product WhereOnEarth (WOE). Given a query, WOE determines if this query has a locational component and if so, outputs a list of locations at the best guessed granularity (i.e., city, county, state, country) along with a confidence. It also outputs an aggregated confidence that captures how location-specific is the query. We run the WOE classifier for each query and store the top postal code, along with the confidence, for each query.

We say that a query is a γ-*geovanity* query if the confidence output by WOE is at least γ, and the location output by WOE matches the user's location as given in the profile. We consider three levels of postal code matching: the first three digits, first four digits, and all five digits.[3] We also define a positive integer parameter k, and we consider a user to have queried for a location only if the user has issued at least k queries for that location for the given postal code matching granularity. Based on these definitions, we compute the fraction of users who issue geovanity queries. The results are shown in Table 5.

From the table, it is clear that users' queries reveal significant information about their geographic location. Even with the most restrictive form of matching, $\gamma = 0.5$ and $k = 10$, and a match of 5 digits, at least 1% of users issue queries that reveal their location.

5. STRUCTURAL VULNERABILITIES

In this section we characterize the two most patent vulnerabilities that can exist in bundling: (1) there is a single dominant user who issues a very large number of searches relative to an average user in the bundle, and (2) there are users who query for geographical locations, especially locations that are unique.

The results in this section assume that bundles have been formed by masking the low-order eight bits of the IP address, and that we only consider bundles with at least 100 queries.

[3]Note that US postal codes (ZIP codes) are allocated such that ZIP codes with a longer prefix in common tend to be closer together.

Users issuing large number of queries. We now study the case where a single user dominates the bundle and is hence highly visible. Note that the bundles are almost ineffective in this case since the privacy of the dominant user, if his/her identity is known through other means, can be severely compromised.

For a given f, we compute the numbers of users per bundle who issue at least f-fraction of all queries within the bundle. We do this for $f = 0.1, \ldots, 0.9$.

As shown in Table 6, about 3% of the bundles have a user that issued at least half of the queries in the bundle. Even more surprisingly, 0.6% of the bundles have a user who issued at least 90% of the queries in the bundle. These numbers suggest that there are many bundles with dominant users and these bundles are potential sources of leaking private information about these dominant users: in such a bundle, queries for sensitive material can be associated with the dominant user and these associations are correct with very high likelihood.

In fact, if we assume that the distribution of the number of queries per user follows a power law, we can show that any random partition of the users into bundles of reasonable size will always result into many bundles with heavy users. We omit the details of this analytical statement in this version of the paper.

Users issuing geographic queries. We now consider a second natural structural vulnerability, which deals with the geographic location of queries. We described our methodology for recognizing queries that specify a geographic location in Section 4.3. Applying those techniques, we now characterize the extent to which a particular user in a bundle might be the unique user in the bundle querying for that specific geographic location. We define a *heavy user* to be one who queries for at least the average number of queries in a bundle; in our case this represents about a third of all users.

The following table shows that there are reasonable amount of geo-diversity within each bundle.

# zips per bundle	12.26
# zips with only one user	9.05
# heavy users with unique zip	9.90
# heavy users with unique zip and the zip has one user	0.05

About 75% of the postal code (zip) locations in a bundle are associated with only one user. A significant portion of the heavy users (9.9 out of 16.7, average per bundle) are associated with only one postal code. Even more, a reasonable fraction of heavy users (0.3%) are associated with only one postal code and this postal code has exactly one user associated with it. It is easy to see that if a user has a unique postal code and also queries for sensitive material such as HIV clinics or adult services in their area, privacy will be easily compromised.

6. ANALYTICAL VULNERABILITIES

Given a bundle of fibers corresponding to different users, how to extract the individual fibers from this bundle? This is a very special case of a clustering problem, where we would like to take all the queries in a bundle and cluster them to fibers corresponding to different users.

First we describe various measures we use to evaluate the performance of algorithms that find fibers from bundles; we adapt the measures used to compare clusterings. We then describe our main method and analyze the performance of the method on querylog Q2. Next we describe two augmentations to the basic method for which we have preliminary results: modeling using topic switch and modeling using a decision-tree classifier.

% queries	10	20	30	40	50	60	70	80	90
# users per bundle	1.4745	0.3271	0.1157	0.0546	0.0314	0.0199	0.0136	0.0093	0.0060
% bundles	78.60	29.40	11.26	5.42	3.14	1.99	1.36	0.93	0.60

Table 6: Number of users per bundle issuing the given percentage of all queries within the bundle.

6.1 Evaluation measures

Since our problem is akin to clustering, we measure the performance of our algorithms using popular measures to compare clusterings. Let $X = \{x_1, \ldots, x_n\}$ be a ground set and let $\mathcal{C} = \{C_1, \ldots, C_k\}$ and $\mathcal{D} = \{D_1, \ldots, D_\ell\}$ be two clusterings of X. Let $\mathcal{C}(x)$ denote the cluster in $\mathcal{C}$ to which x belongs and let $\mathcal{D}(x)$ denote the cluster in $\mathcal{D}$ to which x belongs. We use three measures: f-measure, a variant of an information measure used by Meila [11], and Rand index.

For each $x \in X$, let

$$\mathrm{prec}(x, \mathcal{D}|\mathcal{C}) = \frac{|\{y \mid \mathcal{D}(y) = \mathcal{D}(x) \wedge \mathcal{C}(y) = \mathcal{C}(x)\}|}{|\{y \mid \mathcal{D}(y) = \mathcal{D}(x)\}|}$$

be the precision of x's cluster in $\mathcal{D}$ with respect to $\mathcal{C}$. Let $\mathrm{prec}(\mathcal{D}|\mathcal{C}) = E_{x \in X}[\mathrm{prec}(x, \mathcal{D}|\mathcal{C})]$ denote the average precision. The *f-measure* is then defined to be the harmonic mean of $\mathrm{prec}(\mathcal{D}|\mathcal{C})$ and $\mathrm{prec}(\mathcal{C}|\mathcal{D})$. The closer the f-measure is to 1, the better the agreement between $\mathcal{C}$ and $\mathcal{D}$.

Meila [11] introduced a variation of information measure (called *Meila measure* in this paper) as a robust way to compare clusterings. Here, a clustering is interpreted as determining a random variable $\mathcal{C}(x)$ when x is picked uniformly at random from X. Meila measure is defined as $H(\mathcal{C}) + H(\mathcal{D}) - 2I(\mathcal{C}, \mathcal{D})$, where $H(\cdot)$ is the binary entropy and $I(\cdot, \cdot)$ is the mutual information. The closer the Meila measure is to zero, the better the agreement between $\mathcal{C}$ and $\mathcal{D}$.

The above two measures have good interpretation and are often used in practice [12]. For completeness, we also consider another commonly deployed measure, *Rand index*, which captures the number of pairwise disagreements between the clusterings $\mathcal{C}$ and $\mathcal{D}$. The closer this measure is to 0, the better the agreement between $\mathcal{C}$ and $\mathcal{D}$.

Since our main goal is to show that the privacy of reasonable heavy users in a bundle is compromised, we restrict our attention to users whose fiber size is strictly more than the average fiber size in the bundle, and measure the clustering performance with respect to the queries issued by these heavy users. Furthermore, we exclude a bundle from evaluation if it contains fewer than 100 queries or if it does not contain a heavy user.

6.2 Main method

We now state a simple method to identify fibers in bundles. Note that efficiency is a major concern: ideally we would like it to run in near-linear time so that it is feasible to process massive querylogs.

The main idea behind our approach is to construct a graph based on similarity between queries and then cluster this graph to obtain the fibers. In fact, the fibers will turn out to be the connected components in this graph, which can be computed in near-linear time using the union-find data structure.

As stated in Section 3, each query in querylog Q2 has various attributes: the terms in the query, the time when the query was issued, and the geo location (if any) assigned by the WOE classifier. We first canonicalize the terms in the query by stemming them and removing stopwords.

	BL1	BL2	g-edges	g, w-edges
	mask-8			
f-measure	0.151	0.257	0.181	0.570
Meila measure	0.551	0.449	0.536	0.238
Rand index	0.151	0.849	0.150	0.117
	f-measure			
mask-12	0.160	0.105	0.187	0.562
mask-16	0.164	0.076	0.186	0.521

Table 7: Performance under different measures (mask-n: bundles formed by masking the low-order n bits of the IP address).

Given the canonicalized queries $q_1, \ldots, q_n$ in a bundle, let G be an n-node graph where each node corresponds to a query. We add an edge from query q_i to q_j in G if one of the following two conditions hold.

(1) The geo location of q_i is the same as that of q_j. These edges are called g-edges or *geo edges*.

(2) q_i and q_j share at least one term. These edges are called w-edges or *word-overlap edges*.

Note that both these operations can be performed efficiently. For instance, to do (2), we create an index of all the terms in all the queries and add edges between q_i and q_j if they are together in a posting list in the index.

Performance results. We consider two simple baselines. In the first baseline (BL1), each query in a bundle is assigned its own fiber; this baseline should perform well when there are many small fibers in a bundle and no dominant users. In the second baseline (BL2), all the queries in a bundle are placed in a single fiber; this baseline should perform well when there is a dominant user in the bundle.

Table 7 summarizes the results. We see that there is a significant improvement over both the baselines BL1 and BL2 across the board. The addition of g-edges already causes some small improvement over BL1, but much more dramatic improvements are brought about by the g, w-edges. In other words, mere word overlap in query terms is a very strong signal indicating two queries are from the same fiber. Interestingly, using only w-edges yields similar performance as g, w-edges. This suggests that even though having g-edges alone provides some amount of information, which enables it to outperform BL1, this information is probably largely subsumed by the w-edges. Note that for the mask-8 setting (evaluated on 100K bundles in Q2), we observed similar qualitative results using different measures, and for subsequent experiments we only report by f-measure.

For mask-12 and mask-16, where bundles contain more fibers (as reflected in lower f-measure for BL2), thus presumably more challenging to de-fiber, the results are similar: the method outperforms the baselines, achieving f-measure over 0.52.

We also experimented with incorporating temporal information (the query time). More specifically, we sort all the queries in a bundle by timestamp, and for each adjacent pair of queries, we tried to decide whether they come from different users. A naive way of incorporating the temporal information (comparing the time-interval between two adjacent queries to a threshold) failed to improve over using only word overlap information (i.e., w-edges) in locating the correct cutting-points. A more sophisticated use of time is possible via a classifier; see Section 6.4.

6.3 Modeling topic switch

As we saw earlier, word overlap appears to be the best performing feature to cluster queries into fibers. However, not all the queries issued by a user share words among them, the interesting case is to estimate how likely it is for a user to have issued two queries with no word overlap. In this section we propose modeling the queries via topics and link two queries if they share a topic. Once we determine the topic of a query in an offline mode, we can always add these additional topic-edges (t-edges) on top of w-edges in the graph G and compute connected components as before.

Our approach is through analyzing word co-occurrence patterns in a given querylog. Again, we focus on simple, scalable methods here, and we simplify the problem to estimating how likely it is for two words w_1 and w_2 to appear in different queries of one user's querylog. Note that words that tend to co-occur within queries are not particularly useful for linking different queries and are not of interest to us here.

Suppose we have access to a querylog of reasonable size, in our case, one day (separate from the week-long data that we test our algorithms on)'s worth of queries. For each given user, all words from queries issued by this user form one bag-of-words representation, that can be treated as a pseudo document reflecting this user's "search interest". We hope to discover interesting related words by analyzing co-occurrence patterns in these pseudo-documents. More specifically, after stopword removal, for each pseudo document with n (content) words in it, we consider each of the $n \times n$ pairs of words. We increment the cooccurrence count for the word-pair $\langle w_1, w_2 \rangle$ for each user who issued the two words in *disjoint* queries and not combined in a query. Let $C(w_1, w_2)$ be the count of cooccurrences received by the pair of words (w_1, w_2) after going through all the users in the log.[4]

Raw counts alone tend to rank highly word pairs with frequent words that are not necessarily closely related to one another, since they may coincidentally appear in queries issued by the same user. We need to adjust for the probability that the two words occurred together by chance. We use the χ^2-measure to discover highly related words[5]. Given the following table of observed counts for (w_1, w_2)

	$W_1 = w_1$	$W_1 \neq w_1$
$W_2 = w_2$	O_{11}	O_{12}
$W_2 \neq w_2$	O_{21}	O_{22}

Here $O_{11} = C(w_1, w_2)$ and O_{ij} can be estimated from $C(\cdot,\cdot)$. The χ^2-measure, defined as $\sum_{i,j}(O_{ij} - E_{ij})^2/E_{ij}$, gives estimates for rejecting the null hypothesis that (w_1, w_2) are unrelated. Therefore, pairs with high χ^2 values are likely to be related terms. Not surprisingly, many of the "related" terms turn out to be query re-writes such as spelling corrections; although these are not the topic switch we are looking for, they are just as useful in linking queries with no word overlap. Still, quite a number of interesting pairs, related to each other for a range of different reasons, are discovered. For instance, $\langle$*Partagas* (a cigar brand), *cigars*$\rangle$, $\langle$*Cybersansar* (related to Nepal), *nepalnews*$\rangle$, $\langle$*Measles*, *smallpox*$\rangle$, and $\langle$*Coloroda*, *Denver*$\rangle$.

For efficiency reasons, we discover topics in the following manner: based on the pairs whose χ^2 values are above a certain threshold, we build a separate graph (for topic-discovery) on the terms involved, with an edge between each pair with high χ^2 value, and form equivalent classes (topics) through clustering in this graph. We experimented with the following two methods. The first is computationally expensive: identify the cliques in this graph and the topic of a node is the identity of the largest clique in which it is present. The second, which is computationally easier, is to identify the connected components in this graph and the topic of a node is the identity of the connected component in which it is present. Once we identify topics for terms being covered by this model, we can add topic edges to the original graph G when two queries share common topics. Table 8 shows the effect of adding the topic edges to mask-8 bundling, evaluated on over 1.4 million bundles sampled from one day's worth of querylog and over 700K bundles from one week's worth of querylog. The effect of using different clustering methods with different coverage over the terms (resulted from different thresholds over χ^2-measure) are examined in more detail in Table 9.

Querylog	BL1	BL2	w-edges	w, t-edges
one day	0.152	0.255	0.575	0.588
one week	0.098	0.235	0.469	0.486

Table 8: Modest improvements to f-measure by using topics.

Method	Clique1	Clique2	Clique3	CC1	CC2	CC3
Coverage	21K	18K	9K	47K	27K	4K
w, t-edges	0.587	0.586	0.576	0.497	0.588	0.575

Table 9: Different clustering methods for topic detection

First, we observe that adding topic edges yields modest improvement in f-measure for both day and week-long data. This may seem marginal, note, however, this is partly due to the low coverage: we observe improvements when topics are identified for only 18K terms. Note that compared to the bundles with one day's worth of querylog, the week-long bundles tend to be bigger as they contain larger fibers (as reflected in the much lower performance of BL1), and this caused the performance of the main method using w-edges to drop, although we observe similar improvement by adding the t-edges. Recall that by masking more bits of the IP address, we also get bigger bundles. In that scenario, it is mostly due to each bundle containing more fibers rather than each fiber being bigger (thus not much change in BL1 performance, but lower performance for BL2), and it didn't have much impact on the main method using w-edges. This is consistent with the fact that our method operates by connecting single queries into fibers, rather than slicing up the bundles as one big fiber.

The best performances of the two methods are comparable. We experimented with a larger range of χ^2-value thresholds with the connected components. It actually hurts the performance if the thresholding is low (CC1); this is because connected components force transitivity of topics. The performance improves as higher thresholds are applied (CC2) and diminishes once again if the threshold is too high to result in a good coverage (CC3).

[4]For efficiency purposes, users with more than 500 queries per day are not included in the calculation, due to the quadratic number of pairs. With better heuristics at early pruning of invalid pairs, these users can be a rich source of co-occurrence patterns.

[5]We have also experimented with other measures, including pointwise mutual information (PMI(w_1, w_2)), as well as a weighted version $C(w_1, w_2) \cdot$ PMI(w_1, w_2), following [10]. But we found them not as effective as the χ^2-measure.

```
fraction-common-initial-words > 0.1
| timelag <= 1709: same (1478.0/27.0)
| timelag > 1709
| | words-in-q1 <= 2: same (478.0/92.0)
| | words-in-q1 > 2: different (220.0/71.0)
```

Figure 4: Portion of the decision tree learned to distinguish pairs of queries from the *same-user* versus *different-users*. Numbers in brackets are correctly classified / incorrectly classified query pairs from a single query bundle.

We also point out that if we break the f-measure into precision and recall and examine the effect of t-edges on these quantities, as one might expect, we observe the following: adding t-edges marginally reduces the precision (since t-edges may contribute to incorrectly linking queries from different user) but results in a better improvement to recall. A more sophisticated topic switch model might yield larger improvements. We leave it as interesting future work.

6.4 Classifier-based approach

In the previous sections we described several approaches to joining queries likely to be from the same user. We expect a typical fiber of queries from a single user to consist of multiple sessions of queries. A single session consists of reformulations, and topically related queries, close together in time. Different sessions from the same user may contain similar terms, when the user re-queries for the same topic, or "bookmark" terms [17]. They may also contain similar geographic information, vanity queries, and queries reflecting the user's age, gender and interests. Weighting the importance of time-lag, word-overlap and topical and demographic similarity may be difficult to do manually. For this reason we turn to the use of machine learning, which allows us to consider all of these factors, while finding the appropriate weights for each automatically.

For this supervised machine learning task we used J48, the Weka [18] implementation of C4.5 [13]. For each bundle in the training data, our training data labels consist of all pairs of queries in that bundle. The label is "same" if the pair of queries are both from the same user, and "different" if the two queries are from different users. The features we used for prediction were from several categories: (1) time, (2) word and character similarity, and (3) session rewrite probabilities. We used 40 bundles for training, giving us 109,038 pairs of queries. A portion of the tree learned is shown in Figure 4.

We evaluated the accuracy of this classifier on 460 bundles of various sizes (including small bundles containing less than 100 queries), containing 8,235,739 pairs of queries. While accuracy is very high (97.5%), we do better on precision (71.2%) than recall (29.9%). This may be because we are able to link within-session queries together reliably, but are not able to link queries from the same user across sessions with different topics.

In order to use the classifier for identifying fibers in bundles, we run the classifier over all pairs of queries in the bundle, and classify them as "same" or "different." Then any pair of queries classified as "same" are joined into the same bundle. Since this takes quadratic time, we were able to evaluate only on 100 bundles: we obtain an improvement in f-measure from 0.601 for word-overlap to 0.623 for the classifier.

7. CONCLUSIONS

In this paper we studied the privacy implications of bundling when used as a tool to enhance the privacy of users in querylogs. We first established that an attacker with access to a bundled querylog can realistically identify names or locations of many users in a bundle. The attacker can also perform the converse task of identifying the bundle, given a user. We then examined the structural vulnerabilities of bundling, in terms of both volume and sensitive queries issued by specific users. We finally considered analytical vulnerabilities, which allows us to decompose a bundle into individual fibers of query sessions, with reasonable accuracy. In summary, bundling is certainly a viable technique in the realm of user privacy; however, there are significant challenges to actually using this in practice.

8. REFERENCES

[1] E. Adar. User 4XXXXX9: Anonymizing query logs. In *Query Logs Workshop at 16th WWW*, 2007.

[2] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou R3579X? Anonymized social networks, hidden patterns, and structural steganography. In *16th WWW*, pages 181–190, 2007.

[3] D. Fallows. Internet search users. `http://www.pewinternet.org/pdfs/PIP\_Searchengine\_users.pdf`.

[4] D. Frankowski, D. Cosley, S. Sen, L. Terveen, and J. Riedl. You are what you say: Privacy risks of public mentions. In *29th SIGIR*, pages 565–572, 2006.

[5] C. Gates and T. Whalen. Private lives: User attitudes towards personal information on the web. Technical Report CS-2005-06, Dalhousie University, 2005.

[6] B. J. Jansen, A. Spink, C. Blakely, and S. Koshman. Defining a session on web search engines. *JASIST*, 58(6):862–871, 2007.

[7] R. Jones, R. Kumar, B. Pang, and A. Tomkins. "I know what you did last summer" – Query logs and user privacy. In *16th CIKM*, pages 909–914, 2007.

[8] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *15th WWW*, pages 387–396, 2006.

[9] R. Kumar, J. Novak, B. Pang, and A. Tomkins. On anonymizing query logs via token-based hashing. In *16th WWW*, pages 629–638, 2007.

[10] C. D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, Cambridge, Massachusetts, 1999.

[11] M. Meila. Comparing clusterings by variation of information. In *16th COLT*, pages 173–187, 2003.

[12] J. Novak, P. Raghavan, and A. Tomkins. Anti-aliasing on the web. In *13th WWW*, pages 30–39, 2004.

[13] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[14] B. Rey and P. Jhala. Mining associations from web query logs. In *Proc. ECML PKDD Workshop on Web Mining*, 2006.

[15] P. Samarati and L. Sweeney. Generalizing data to provide anonymity when disclosing information (abstract). In *17th PODS*, page 188, 1998.

[16] C. Soghoian. The problem of anonymous vanity searches. *SSRN eLibrary*, 2007.

[17] J. Teevan, E. Adar, R. Jones, and M. Potts. Information re-retrieval: Repeat queries in Yahoo's logs. In *30th SIGIR*, pages 151–158, 2007.

[18] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005.