On the Privacy of Web Search Based on Query Obfuscation: A Case Study of TrackMeNot

Sai Teja Peddinti and Nitesh Saxena

Computer Science and Engineering
Polytechnic Institute of New York University
psaiteja@cis.poly.edu, nsaxena@poly.edu

Abstract. Web Search is one of the most rapidly growing applications on the internet today. However, the current practice followed by most search engines – of logging and analyzing users' queries – raises serious privacy concerns. One viable solution to search privacy is *query obfuscation*, whereby a client-side software attempts to mask real user queries via injection of certain noisy queries. In contrast to other privacy-preserving search mechanisms, query obfuscation does not require server-side modifications or a third party infrastructure, thus allowing for ready deployment at the discretion of privacy-conscious users. In this paper, our higher level goal is to analyze whether query obfuscation can preserve users' privacy in practice against an adversarial search engine. We focus on TrackMeNot (TMN) [10, 20], a popular search privacy tool based on the principle of query obfuscation. We demonstrate that a search engine, equipped with only a short-term history of a user's search queries, can break the privacy guarantees of TMN by only utilizing off-the-shelf machine learning classifiers.

Keywords: Web Search Privacy, Query Obfuscation, Noisy Queries

1 Introduction

With an enormous amount and wide variety of data available on the web today, web search has emerged as one of the most important services. In the recent past, the prevalent practice followed by search engines – of logging and analyzing users' web search queries – has received considerable attention from media and public as well as researchers all over the world. The issue was first brought into limelight in August 2005 in the wake of US Department of Justice's subpoena to Google for a week's worth of search query records [15]. This was followed by publishing of AOL's three month (pseudonymized) search query logs, from which identities of certain users had been extracted based on personal information embedded in their queries [9, 2]. Right after, other media reports shed more light on how several major search engines (Yahoo!, AOL, MSN and Google) log, store and analyze individual search query logs.

Archiving search queries, from search engine's perspective, is inherently useful for improving the efficiency of search and quality of search results, and for revenue generation through sponsored search advertising. However, it has serious

privacy implications for the users of the search services. Some common examples of search behavior that can have an *explicit* adverse effect on a user's privacy, when queries are logged, include – searching for information on a particular disease the user or a family member might be suffering from, searching for one's social security number or phone number just to verify if it exists on the web, locating directions, subscribing to news items, and "ego-surfing[1]." Additionally, and perhaps more seriously, query logs can also be used for *implicit* privacy violations. By implicit, we mean that the sensitive information can not be learned directly, but has to be extracted from a user's queries via profiling and aggregation methods or data mining techniques. For instance, it is possible to infer a user's income level from the brands of products he or she often searches for [21].

A number of techniques have been proposed to address the problem of search privacy. One class of these techniques involves third-party infrastructure such as a proxy, e.g., Scroogle [18] or an anonymizing network [17], e.g. Tor [19]. These approaches, however, require the user to impose (unwanted) trust onto third-party servers and usually have performance penalties. Another body of work applicable to web search privacy is on private information retrieval (PIR) protocols [14]. Current PIR protocols, unfortunately, are not feasible to be deployed in practice due to high computation and communication overheads.

A third class of solutions, which is the focus of this paper, is based on the principle of *query obfuscation*. Basically, the idea is that a client-side software injects *noisy* queries into the stream of queries transmitted to the search engine; if the engine is unable to distinguish between noisy queries and real user queries, user profiling may not be possible, thereby preventing implicit privacy violations. A query obfuscation technique does not require any server-side modifications and allows for ready deployment at the discretion of privacy-conscious users.

Our Contributions: A higher level goal of this work is to analyze how effective query obfuscation can be – in preserving users' privacy in practice – against an adversarial search engine. To this end, we focus on TrackMeNot (TMN) [10, 20], a real-world search privacy tool based on query obfuscation (the only one we are aware of). TMN is implemented as Mozilla Firefox plugin that attempts to hide user queries in a stream of programmatically generated search queries, which mimic or simulate the user's search behavior.

As we discuss in the following section, TMN has taken necessary measures to simulate user's search behavior and generate noisy queries as similar as possible to user's queries. TMN has also evolved considerably over time shaping into a potentially robust and popular query obfuscation tool.[2] [3] We set out to investigate whether it is still possible (and to what extent) for an adversarial search engine – equipped with users' search histories – to filter out TMN queries

[1] It is the prevalent practice of searching for one's own name, on a popular search engine, to see what results appear.

[2] Currently, TMN's plugin version 0.6.719 has been downloaded 390,909 times.

[3] We refer the reader to Bruce Schneier's criticism of TMN and subsequent discussion, following TMN's introduction back in 2006 [3].

using off-the-shelf machine learning classifiers and thus undermine the privacy guarantees provided by TMN.

We answer the above question affirmatively. We selected 60 users from the publicly-available AOL search logs and treated them as users of the TMN software. For each of these users, we measured the efficiency of some known machine learning classifiers with respect to two metrics: (1) percentage of *correctly identified user queries*, and (2) percentage of *TMN queries incorrectly identified* as user queries. If there are u user queries and t TMN queries, recorded by the search engine, and a classifier predicted $u' + t'$ queries as user queries, where u' corresponds to correctly identified user queries and t' corresponds to incorrectly identified TMN queries, then our two metrics are given by u'/u and t'/t, respectively. The classifier is said to be doing a good job if u'/u is close to 1 and t'/t is close to 0, i.e, percentage of correctly classified user queries is close to 100% and percentage of incorrectly classified TMN queries is close to 0%. Through our current experiments, we are able to achieve an average accuracy of 48.88% for identifying user queries, while the percentage of incorrectly classified TMN queries is only 0.02%. We also observed that queries corresponding to some of the users could be identified with 100% and greater than 80% accuracies, whereas for others, the identification rate was less than 10%. Based on our results, we can conclude that most users are susceptible to privacy violations even while using TMN, some of them being significantly more vulnerable than others.

In terms of related work, we find that theoretical models have previously been developed to bring insights into the effectiveness of query obfuscation for search privacy [24]. We are also recently made aware of a short paper [4], which presents a brief analysis of TMN using search logs from a single user (see Section 2.1 of [4]). Current paper represents the first step, to the best of authors' knowledge, towards a larger scale analysis of TMN using existing classifiers.

We also note that the problem considered in this paper is different from the problem of identifying queries from an anonymized search log (see, e.g., [12, 11]). First, an adversary in our application is the search engine itself and not a third party attempting to de-anonymize a search log. Second, unlike a third party, the search engine is already in possession of users' search history using which it can effectively train a classifier. Moreover, the goals of our study are also different; we are interested in evaluating known classifiers to study our problem so as to keep our attacks simple and easy enough for an unsophisticated adversary.

The rest of this paper is organized as follows. In Section 2, we discuss TMN's query generation. In Section 3, we present our experimentation methodology and set-up, user selection criteria and query logging methods. This is followed by Section 4, where we put forward our query classification results, and finally, some discussion based on our results in Section 5.

2 Background: TMN Query Generation

In this section, we discuss TMN query generation process. We first try to understand this process based on what was reported in [10], and then, for deeper insights, inspect TMN's source code [20].

2.1 Understanding TMN from the Literature

As mentioned earlier, TMN hides the user queries in a stream of programmatically generated search queries, which mimic or simulate the user's search behavior. TMN maintains a dynamic query list, which is instantiated with an initial seed list of queries obtained from popular RSS feeds and publicly available recent searches. Later, individual queries from this list are randomly selected and substituted with query-like words from HTTP response messages returned by the search engine for actual user queries. Over time, each TMN instance develops a unique set of queries and adapts itself to the user's search behavior and mimics the user more closely.

TMN employs a "Selective Click-Through" mechanism, which simulates the user behavior of clicking on the query results returned and listed by the search engine. It uses regular expressions to avoid clicking on revenue generating advertisements, and thus claims to leave the web business model unharmed. It keeps track of the user searches by monitoring all outgoing HTTP requests from the browser using the "Real Time Search Awareness" mechanism. The "Live Header Maps" feature enables TMN to adapt dynamically to the specific client browser data, such as browser version and operating system details, helping TMN to use the exact set of headers that the browser uses. TMN also implements "Burst Mode" queries in order to incorporate the common user behavior of firing related queries in immediate succession as part of a query session.

With all these features, TMN is believed to be a good simulator of user's searching behavior. However, it has certain drawbacks as mentioned in [10]. TMN can not mask a user's private information (e.g., names or phone numbers) included in the search queries themselves, and it can not prevent user identification based on the IP address or cookies typically used by search engines. In order to hide one's IP address while searching, TMN developers [10] recommend the use of anonymizing networks, such as Tor [19]. Bruce Schneier, in his blog [3], also commented about the weaknesses of TMN. Though most of the raised objections have already been addressed in the latest version of TMN, some of them are noteworthy, such as the problem of "hot-button issue" searches. This problem may occur when TMN itself generates sensitive search queries, e.g., those involving "HIV","drug-use" and "bombings", and which might be problematic for TMN users. The TMN authors claim that this problem can be prevented by configuring the initial RSS input feeds and thus controlling the type of queries sent by TMN. Based on these discussions, we can say that TMN (potentially) only provides protection against aggregation and profiling of individual search queries by adversarial search engines. With and without the use of TMN, user's area of interest would be exposed to the adversary, but when using TMN, the actual search queries would be masked in a stream of related queries. The better the simulated queries resemble the actual user queries, the better are the chances for TMN to hide the actual user queries.

2.2 Understanding TMN from the Source Code

In order to obtain a deeper understanding of TMN, we analyzed the supporting code of TMN's Firefox extension. Mozilla extensions which are written in XUL

and JavaScript, provide an easy way to develop new applications on top of the basic Firefox browser platform. The XUL language extends the GUI of the browser while the JavaScript helps in defining the functionality.

When TMN is installed on the Firefox browser, it creates a default query seed file along with a query list. This query list is initialized with some queries extracted from the default or supplied RSS feeds and this list is padded with some queries from the default seed file if the queries extracted are less in number. Once the query list is generated, a search is scheduled immediately without any delay (delay is 0 seconds). (Later on, some non-zero delay values are specified to schedule a new search based on the query generation frequency chosen by the user, using the TMN control panel, and some random offset value).

After the delay timeout, a random query from the query list is selected to perform a search. With some probability, the query is modified to be only the longest word or a negated word is concatenated to the query, such as "word1 word2 - word2" or quotation marks are added. Sometimes, if "Burst Mode" is enabled, a sequence of related queries might be generated from the selected query by omitting some keywords at random. These Burst Mode queries are sent within short intervals of time, so as to form a chain of related searches.

TMN maintains a list of headers and URLs for each search engine, and an entry in these lists gets updated with new headers and URLs when the user performs a search on the corresponding search engines. The previously selected and modified query is added to the URL, which is then encoded and an XMLHttpRequest is generated for the encoded URL with updated header fields. TMN saves this last query fired and displays it on the Firefox status bar; it also stores this URL in search history for later reference. When there is a state change in the XMLHttpRequest sent, i.e., when a response is received from the server, an appropriate action is taken based on the HTTP status response. If an error occurs, it is logged. If the HTTP status response is *OK*, based on some probability, TMN tries to simulate the user click-throughs. To this end, TMN identifies the links on the HTML response sent by the search engine, processes these links and picks one of them at random. After some delay, another XMLHttpRequest is generated with the selected link, thereby simulating the user behavior of clicking a link. TMN does not process the returned html response for this click-through link. If Burst Mode is enabled, TMN schedules the next search with the following keyword in sequence. If it is not under Burst Mode, the HTML response is processed, and keywords from the textual content on the web page are identified and extracted. TMN then picks at random a new keyword from this extracted keyword list and adds it to the query list by replacing a query at a randomly picked index in the querylist. This new query list is saved and written to the TMN seed file. TMN again schedules new search at a timeout value with an offset and this procedure is repeated.

In this way, the TMN seed file gets updated with keywords extracted from the web results returned by the search engine, for the queries fired by the user. In the long run, TMN gets adapted to a query content the user is interested in and generates better queries making it (potentially) much harder for the search

engine to differentiate the noisy queries from the original user queries. Because some form of randomization occurs at each and every step, it is impossible for two TMN instances to generate the same set of TMN queries.

3 Experimental Study of TMN: Preliminaries

Based on our discussion in previous section, we find that TMN has taken necessary measures to simulate user's search behavior and generate noisy queries as similar as possible to user's queries. TMN has also evolved considerably over time resulting in a potentially robust and popular query obfuscation tool. In this work, we set out to investigate whether it is still possible (and to what extent) for an adversarial search engine – equipped with users search histories – to filter out TMN queries using off-the-shelf machine learning classifiers and thus undermine the privacy guarantees provided by TMN. In our *adversarial model*, we assumed that the search engine is adversarial and its goal is to distinguish between TMN and user queries for profiling and aggregation purposes. We also assumed that the engine would have access to user's search histories for a certain duration until the point the user starts using the TMN software. We considered a passive adversarial search engine, the one who only works with and analyses the queries received from the users, and in particular, does not inject manipulated responses to the user in an attempt to distinguish between TMN and user queries.

In order to pursue our study, we should work with real user queries. To this end, one possibility was to seek users who may volunteer to use TMN and let us record all outgoing (user as well as TMN) search queries fired from their machines. However, due to the privacy concerns (which form the basis for our work), it was not feasible to recruit such volunteering users.

To address the above problem, we used a novel experimental methodology. We worked with the AOL search data [1] and modeled or simulated the existing user queries in a way they would have appeared to the search engine if the users were using TMN. The AOL search data was well suited for this purpose because it consists of a large number of real user queries (21 million), corresponding to a large user base (650,000) and spanning over a reasonably long period of time (3 months). Though the AOL logs correspond to a different time period (year 2006), it does not affect our experiment because we concentrate on the query content alone and do not consider the associated query timestamps, as we discuss later in the paper (see Section 4.3). Since most queries do not have temporal dependence, we proceed with the use of historical AOL search logs for our experiments.

We selected a few users from the AOL logs and simulated their behavior of issuing queries to the search engine while TMN is installed and running on their machines. TMN is a Mozilla extension and these extensions, installed on a Firefox browser, operate only on one user profile – the one on which it was installed. Hence, we can have multiple Firefox user profiles, each with its own independent TMN instance, simulating a different user.

Due to the resource limitations on a single machine, it was difficult to run many Firefox user profiles simultaneously. To remedy this, we used the Plan-

etLab [16] system, a global distributed research network used by researchers to develop network applications and run network simulations. PlanetLab resources are assigned to the users as a resource slice, and these slices are instantiated by assigning nodes to it. Each of these nodes need to be configured with the experimental environment, which in our case is a working Mozilla Firefox browser with the TMN plugin installed.

3.1 Categorizing AOL Data

As mentioned earlier, the AOL logs span across three months: March, April and May of 2006. We use the last month's data for simulating the user and reserve the data from the first two months as the user history which we assume is available to the search engine before the user started using TMN. The former will be used as a test set and the latter as a training set, in case of our supervised machine learning classifiers (see Section 4.3). We selected a set of 60 test users from the AOL logs. This selection is based on AOL users' behavior as observed across four different categories (discussed below) so that a wide variety of users are covered. These categories are directly or indirectly related to TMN's query generation process. For obtaining the statistics across each category, we considered all the 650,000 AOL users, and thus each user falls into one user group in each category. While choosing our 60 test users, we made sure that there exist as few user intersections as possible across different categories so as to report the results for 60 different users.

Number of Queries: Over a period of time, different users fire different number of queries. TMN's efficiency in masking real user queries depends on the number of searches performed by the user. More the number of queries sent by the user, better are the chances for TMN to adapt itself to the user search categories and the content the user is interested in. We calculated the total number of searches performed by each user and plotted the number of users across different query bands. From Figure 1 (a power law distribution), we can see that most users lie below the 500 query mark with the bulk of them performing less than 100 searches over a three month duration. The rest are spread across the graph in small numbers. The same characteristics are also seen in the graphs plotting the maximum number of queries fired in a day, a week and one month versus the number of users in each query band. We thus combined these four into the same category (i.e., number of queries over a 3 month period).

Average Query Frequency: Users have different querying rates, which may turn out to be an identifying feature for TMN. TMN provides an option to set the frequency at which (noisy) queries are sent to the search engine. If this frequency varies significantly from the actual user's timing pattern, then it becomes easy to filter out the TMN queries. Hence, we computed the average timing difference between successive queries for each user and plotted the number of users across different time bands, as shown in Figure 2.

Sensitive Query Content: The contents of search queries obviously varies from user to user and TMN should be successful in masking these queries, especially those which are sensitive in nature. We considered two broad classes for

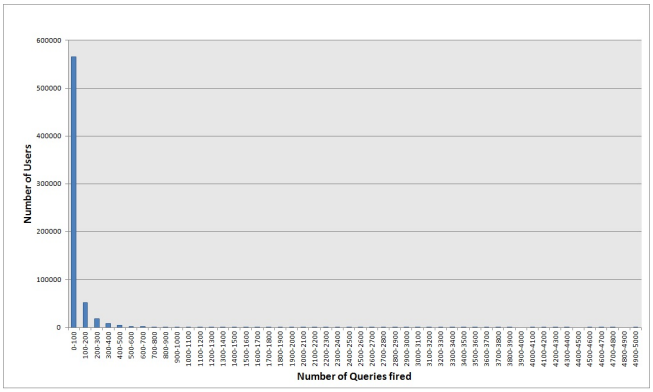

Fig. 1. Number of Queries

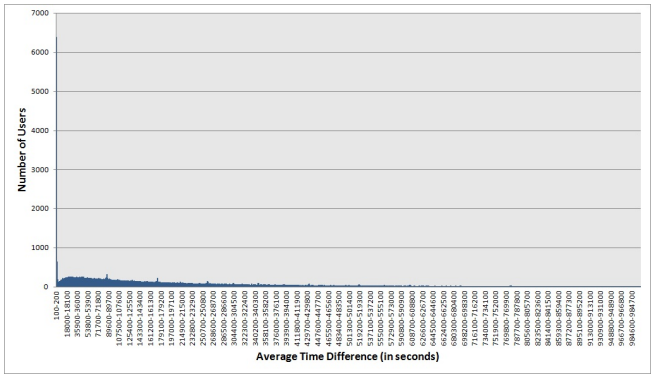

Fig. 2. Average Query Frequency

the query content: sensitive and insensitive. Sensitive queries are those, which a user may not be willing to reveal to the outside world, such as his/her medical condition, interest in weaponry (considering the alarming increase in terrorism), those related to child abuse and pornography, and so on. On the other hand, a user may not mind the public taking notice of his/her insensitive queries, such as those related to movie interests, sports, and education.

Manually identifying the sensitive/insensitive query distribution for each user is very cumbersome. To remedy this, we resorted to machine learning techniques for classification of query content. We manually labeled a small subset of queries into sensitive and insensitive categories (we referred to various press articles discussing sensitive queries that appeared in the AOL logs [9, 2]), and trained a Naive Bayes classifier with this data . This classifier was later used to classify the rest of the user queries. The cross-validation accuracy of this classifier on the manually labeled set was 68.0095%. Having identified each user's queries into the sensitive and insensitive categories, we plotted a graph indicating the number of users across different sensitive/insensitive percentages (see Figure 3). The graph is alarming and contrary to what one would normally expect. A large number of users were classified to be making sensitive queries. This anomaly

could be because of the way we trained the classifier; while training, we labeled the complete query in the training set to be sensitive or insensitive instead of just selecting some relevant keywords, because we did not want the filtering mechanism to miss queries – such as "how to kill your wife" – which are not necessarily keyword sensitive.

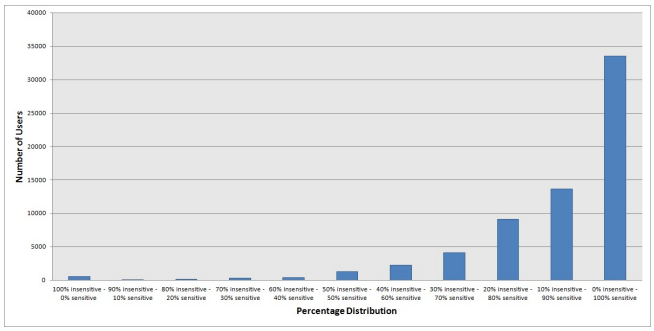

Fig. 3. Sensitive/Insensitive Distribution

Weekday/Weekend Distribution: Some users perform web search only during their office hours over the weekdays and some only over the weekends. If such users start using TMN on their desktop machines (neglecting laptops or notebooks, due to frequent periods of inactivity when these devices are put to sleep), it may be easier to separate out the TMN queries. Speculating this as an important issue, we calculated the number of queries fired by each user over weekdays and weekends. We categorized users into three groups – those who search only over weekdays, those who search only over weekends and those who distribute their queries between weekdays and weekends. Figure 4 provides a graphical distribution of this data.

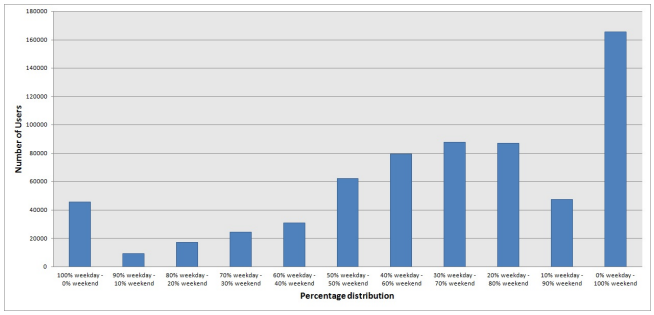

Fig. 4. Weekday/Weekend Distribution

3.2 Selecting Users

Having observed different statistics, we next needed to select various users for our study. We decided to select 15 AOL users from each category discussed above, i.e., a total of 60 users.

Number of Queries: From Figure 1, we find that most users are below the 100 query mark, and of these, more than 70% perform less than 30 searches during

the three month period. Thus, we selected 8 users at random from the set of users who fire less than 30 queries, 5 users from the set of users who fire less than 100 queries and 2 users who pose more than 100 queries.

Average Query Frequency: The graph in Figure 2 is smooth everywhere except for a sharp peak at 200 seconds. To take this into account, we randomly selected 5 users from the set of users with an average query frequency of less than 200 seconds, 5 users lying near the second rounded peak at 35000 seconds, and the remaining 5 from the set with more than a million seconds average gap between successive queries.

Sensitive Query Content: Since there are a large number of users in the 100% sensitive band (Figure 3), we randomly selected six users from this set. Five users are selected from the 30% insensitive - 70% sensitive band, two users from 10% sensitive - 90% insensitive band, and the remaining two from 100% insensitive query set.

Weekday/Weekend Distribution: Based on the distribution in Figure 4, we equally divided the choice of users among those who fire all their queries over weekdays, those who distribute 40% on weekday and 60% on weekend, and those who search only during weekends.

3.3 Experimental Set-Up and Implementation

After the user selection, the task ahead was to simulate the user logs while a TMN instance per user is running, and record all the resulting queries. We are using PlanetLab to run multiple Firefox instances. Once a resource "slice" has been assigned to a PlanetLab user account, "nodes" need to be allocated to the slice to utilize the resources. Each allocated node behaves as a separate Unix operating system, with basic utilities pre-installed and a provision to install any necessary softwares and updates.

Sixty nodes (corresponding to each selected user) were allocated to the slice, and each of these nodes maintains one Firefox user profile. Since Mozilla is a GUI application and X11 forwarding (necessary to run GUI applications over SSH connections) is not enabled on the PlanetLab machines due to security reasons, we had to install a VNC server on each of the nodes, which provides a GUI enabled remote access to these machines. Google was chosen as our (adversarial) search engine.

To simulate user's search behavior as per AOL log files, another Mozilla extension has been developed which reads the user logs and fires the queries at their respective timestamps listed in the logs. Similar to the TMN plugin, the new plugin also generates the user queries as XmlHttpRequests. The html response – from the server – to these queries is processed by TMN, since TMN does not find the corresponding request URL in its database (see Section 2). TMN treats the webpage to be a valid response to an actual user query and adapts itself to the new data – the exact behavior we need. Since the AOL user logs belong to a different time frame (year 2006), they were translated to the present time. The average query frequencies of TMN instances were chosen at random so as to keep them as close as possible to the real user behavior. We also ran 5 additional

TMN instances with varying average TMN query frequency, for the same user, on our local machines in order to evaluate the effect of query frequency on the level of privacy provided by TMN. After configuring the necessary settings on PlanetLab machines, both the user log simulator and TMN were started. These experiments were conducted for a period of one month, and backup of the logs was taken at regular intervals.

4 Classification of User and TMN Queries

For our machine learning requirements, we used WEKA [23], an open source software which supports many machine learning algorithms and data preprocessing options. We used this off-the-shelf machine learning toolkit in order to estimate the accuracy with which we (adversarial search engine) can filter user queries, from the pool of user and TMN queries we obtained as described in previous section.

Two main categories of machine learning algorithms which can be used for our application are clustering and classification algorithms. Classification algorithms assign labels to quantities after being trained on a labeled training set. Clustering algorithms, without any prior knowledge of labeled data, try to group the data into groups (clusters), such that elements in a group share some common features. Classification is a supervised mechanism, where we need to train the classifier on some labeled training set, and determine its classification accuracy by labeling data in the test set. Clustering is the unsupervised version which gathers information about the data from an unlabeled training set and divides the test set into clusters [23].

4.1 Preparing the Data

The pool of simulated user and TMN query logs, collected over the one month period (as discussed in previous section), form our test data which needs to be clustered or classified. We labeled each query in the test data as a user or TMN query, since we want to test the accuracy of machine learning algorithms after categorizing the queries. The data includes the query, its label and the timestamp when the query was fired. For indicating the time, we used WEKA's DATE attribute in "yyyy-MM-dd HH:mm:ss" format. The queries are strings and WEKA can not directly handle string attributes. So we used a preprocessing filter, called *StringToWordVector*, which breaks down the words in the string and converts them into numeric attributes. Each string gets converted into a word vector of 1s and 0s in these attributes, where '1' indicates the presence and '0' indicates the absence of the word in the string.

4.2 Clustering Algorithms

We started with the unsupervised/clustering schemes since they are simple and potentially more powerful (as no labeled training is needed). We tested the performance of well known clustering algorithms, such as SimpleKMeans, Farthest

First and EMClusterer [6] with *default parameters* using the *Classes-to-Clusters* evaluation mode. In this testing mode, the pre-assigned labels are masked and the data gets processed using the other attributes. Once the clusters are formed, the labels are unmasked and the majority class in each cluster is determined to find the accuracy of the algorithm as per these labels. However, the clustering algorithms with default parameters could not distinguish user queries from those of TMN and placed both types of queries into the same (TMN) cluster, for all of our test users. We note that it is possible to achieve better user query identification results by fine tuning the parameters of the clustering algorithms or applying other procedures, such as n-grams. However, since our goal is to identify the efficiencies using simple off-the-shelf machine learning tools with no parameter optimization, we defer this task to future work, and rather concentrate on classification algorithms.

4.3 Classification Algorithms

Since clustering with default parameters performed poorly, we decided to work with supervised/classification algorithms which are trained on prior labeled data. While training the classifiers, we need to have sample data corresponding to both the user and TMN classes (labels). If only one of user or TMN training data is used, all the queries would get classified into the same class since there are no identifying features available about the other class. The training set for the user queries was obtained from AOL two month user history, as discussed in Section 3.1. To obtain the TMN training set, we used the logs from a TMN instance which was run independently of all our simulations on a desktop machine for a period of one week.

With the training and test sets at our disposal, we next needed to choose the classifiers for our study. Out of several classifiers applicable to our scenario, based on their performance in few preliminary tests, we selected five algorithms: Logistic (Regression), Alternating Decision Trees (ADTree), Random Forest, Random Tree and ZeroR. For the sake of completeness, a brief description of each of these classifiers is provided below:

- *Logistic (Regression):* Regression classifier models are used to predict the probability of occurrence of an event by trying to fit the data to a logisitic curve. Logistic regression is mainly used when there are two classes of data, but multinomial versions also exist [5].
- *Alternating Decision Trees:* It is a decision tree algorithm containing decision and prediction nodes. These decision nodes specify a condition while the prediction nodes contain a number. In traditional decision trees, we travel along one path from the root, but here we simultaneously travel along many paths upto the leaf prediction nodes and the end result is determined by considering all the prediction node values covered [22].
- *RandomForest:* It is a collection of classification trees, in which the input is made to travel across all the trees and the final decision is made based on voting [13].

- *RandomTree:* It considers K randomly chosen attributes at each node in the tree and provides an estimation of class probabilities [8].
- *ZeroR:* This algorithm identifies the majority class label and classifies every element with the majority label, thereby providing the threshold accuracy that should be provided by other classifiers [7].

Query and Date Attributes: To check for the influence of each of the attributes (query and date) on the classification, we tested the performance of the above four classifiers (except ZeroR as its user accuracy is 0% due to a large TMN query set) across the following three settings for a couple of test users. Our goal was to determine to what extent these attributes might be useful for classification.

1. Considering only date and label value attributes
2. Considering only query and label value attributes
3. Considering both query and date along with label value attributes.

The results obtained are indicated in Table 1. (The percentages indicate the fractions of user queries correctly identified by the classifiers; the TMN query misclassification rates were close to 0% in most cases and are not listed). We can clearly see that out of the three settings, considering only query attribute along with label values provides the maximum accuracy. Including the date attribute reduces the accuracy and considering only the date attribute yields the worst accuracy. Therefore, for the analysis of rest of the experimental data, we neglect the date attribute and consider only query and label values as the data to be classified. Since Naive Bayes is a standard classifier which can be used when date attribute is not considered, we replaced ADTree with Naive Bayes classifier for the rest of our analysis.

Table 1. % of user queries correctly classified with different attributes

		Classifier Accuracies			
		Logistic	**ADTree**	**Random Forest**	**Random Tree**
User1	**Only Query**	92.59%	82.22%	92.59%	89.63%
	Only Date	14.44%	13.7%	13.7%	13.7%
	Both Query and Date	92.59%	13.7%	89.63%	46.30%
User2	**Only Query**	85.19%	85.71%	86.77%	86.24%
	Only Date	3.17%	0.53%	0.53%	0.53%
	Both Query and Date	10.58%	0.53%	68.25%	0.53%

TMN Average Query Frequency: To test for the effect of TMN's average query frequency in protecting users' privacy, we ran another 5 simulations apart from the 60 simulations considered before. Each of these 5 simulations, simulated the same user but with different TMN query frequencies – 10 per minute, 5 per minute, 1 per minute, 30 per hour and 1 per hour. After one month, these TMN logs were analyzed using the shortlisted classifiers. The results obtained for Naive Bayes and Logistic (Regression), which yielded the best accuracies, are depicted in Table 2. Though the performance of Naive Bayes was varying a

little, the Logistic regression classifier was found to have a constant accuracy. This suggests that using different query frequencies would more or less provide the same level of accuracy. In other words, higher TMN frequency may not help in hiding user's query, contrary to one's intuition.

Table 2. % of user queries correctly classified for different TMN query frequencies

TMN Query Frequency	User Query Accuracies		TMN Query Misclassifications	
	Naive Bayes	**Logistic (Regression)**	**Naive Bayes**	**Logistic (Regression)**
10 per Minute	6.25%	56.25%	0%	0.06%
5 per Minute	0%	56.25%	0%	0.02%
1 per Minute	56.25%	56.25%	0%	0.12%
30 per Hour	56.25%	56.25%	0%	0%
10 per hour	56.25%	56.25%	0%	0%

Independent User History: Since using an independent TMN log for training the classifier turned out to be helpful in identifying the user queries with good accuracies, we performed a test to validate whether any user log data other than the actual user history would also give similar results (if this were the case, the search engine would not need access to every user's history of searches). To this end, we considered four users, $user_1$, $user_2$, $user_3$ and $user_4$, from the AOL log data. Now, instead of using a user's history to train the classifier for that user, we used the history of $user_4$ as the training data and tried to classify $user_1$, $user_2$ and $user_3$ simulated queries from their respective TMN query pools using Logistic, RandomForest, RandomTree and Naive Bayes (substituted with ADTree, as described before) classifiers. In all the cases, none of the user queries were identified correctly, however. That is, the accuracy turned out to be 0%.

Our analysis above shows that an independent user log is not helpful in distinguishing between user and TMN queries, but an independent TMN log is. One reason for this could be that the independent TMN log was functional around the same time frame as other TMN instances (i.e., it was run along with other TMN instances). Note that an adversarial search engine can also produce such updated TMN log from time to time for training the classifiers.

We note that many users are not likely to pay attention to the RSS feeds chosen for query generation and may use the default ones. Thus, in our experiments, we used the default RSS feeds thereby generating the same initial seed list of queries. We have not closed the browsers while conducting our experiments because of the common practice among users to put their computers to sleep and re-invoke them instead of switching them off and rebooting the machines each time, and also due to their tendency to continue using the browser without restarting unless it crashes. We acknowledge that not closing the browsers might affect the efficiency of TMN, because TMN updates the query list with new keywords from RSS feeds every time the browser restarts.

4.4 Classification Results

After collecting the query and label data from the 60 user simulations, we were ready to execute the selected classifiers. We built the training set with user

history log and an independent TMN log as discussed previously. The results of classifiers over the test data are depicted in Table 3. For simplicity, we have not listed the results for all the classifiers; rather we only report the performance of the standard Naive Bayes classifier and the maximum accuracy achieved by the other three classifiers (Random Forest, Random Tree and Logistic). Also, the accuracies shown are the *mean accuracies* of the users belonging to different AOL categories (as defined in Section 3.2). We find that for all users, the classifiers did a very good job of correctly identifying almost all TMN queries; average TMN query misclassification rate was close to 0.02%. In other words, there were only a very few TMN queries which were wrongly classified.

Table 3. Mean accuracies of user queries and mean misclassifications of TMN queries for each category of users

No. of Queries	Users	User Accuracy (%)		TMN Misclassif. (%)	
		Naive Bayes	Max.	Naive Bayes	Max.
0-10	8	6.15	11.52	0	0.07
11-100	5	7.08	33.14	0	0.25
100+	2	18.71	33.86	0.06	0.29

Average Query Freq. (sec)	Users	User Accuracy (%)		TMN Misclassif. (%)	
		Naive Bayes	Max.	Naive Bayes	Max.
0-100	5	28.16	40.41	0.03	0.01
35000	5	30.83	71.86	0.01	0.01
$> 10^6$	5	9.23	36.28	0	0

Senst. Query Content	Users	User Accuracy (%)		TMN Misclassif. (%)	
		Naive Bayes	Max.	Naive Bayes	Max.
0%	2	60	60	0	0
10%	2	61.46	64.79	0	0
70%	5	45.53	63.96	0.02	0.14
100%	6	23.97	39.02	0	0.16

Weekday/ Weekend Dist.	Users	User Accuracy (%)		TMN Misclassif. (%)	
		Naive Bayes	Max.	Naive Bayes	Max.
Only weekdays	5	12.28	12.28	0	0
Only weekends	5	23.2	99.99	57.26	0.08
Distributed	5	1.22	99.92	86.94	0.08

The accuracies for identifying the user queries were not very high in general; average accuracy over all users was 48.88%. In most cases, the classifier was able to identify a reasonable fraction of user queries correctly. However, there were indeed some cases (e.g., the one for Sensitive Query Content and one for Average Query Frequency categories) where 100% accuracy was achieved in identifying the user queries. There were 4 other user instances for which more than 80% accuracies were achieved.

5 Discussion of Results

In this section, we discuss and attempt to interpret the results obtained in Section 4. The first key insight from our results is that the classifiers were very accurate in identifying the TMN queries (mean misclassification rate over all users was only

0.02%). In other words, very few TMN queries were wrongly identified as user queries. This is perhaps because the TMN query log – using which the classifiers were trained – consisted of a reasonably large number (42334) of TMN queries (although only corresponding to a week's period) which was likely sufficient to extract features for identifying TMN queries. Recall that this log was generated around the same time frame as our test user instances, which might have been helpful in correct classification of TMN queries. Note that an adversarial search engine can also produce such updated TMN log from time to time for training the classifiers. A very low rate of misclassification of TMN queries implies that any query classified as a user query is indeed a user query with a significantly high probability.

The classification accuracies for user queries, on the other hand, were not as good as they were for TMN queries (we obtained a mean user query identification accuracy of 48.88% over all users). One possible reason for relatively low accuracy in this case is that we were only able to leverage users' two-month history for training purposes. Since a large number of users only fired less than 100 queries (as seen from Figure 1) over 3 months, the classifiers did not have a large number of user queries to work with. Due to this reason, perhaps it was not possible to derive identifying characteristics for user queries in a number of cases. We believe that, in practice, the search engines can utilize long-term search histories available to them prior to a user starts using the TMN software, resulting in much better accuracies. Even with our current average identification rates of about 50%, the search engine can identify 50% of user queries (since almost no TMN queries were incorrectly classified, as discussed above) and still use them for profiling and aggregation purposes. Note also that our accuracies were found to vary significantly across different users. We observed that queries corresponding to some of the users could be identified with 100% and greater than 80% accuracies, whereas for others, the identification rate was less than 10%. Based on our current experiments, we can conclude that most users are susceptible to privacy violations even while using TMN, and some of these users are significantly more vulnerable than others (as we discuss below).

Looking at Table 3, we can make inferences regarding which users are possibly more vulnerable based on our different categories: number of queries, average query frequency, sensitive query content and weekday/weekend distributions. User query identification accuracies seem to be slightly improving with the number of queries posed by the users. Although the misclassification rates are increasing very slightly, we can ignore them considering a good improvement in user query classification rate. These results are justifiable because more the number of queries sent by a user, more are the chances to identify user query patterns and hence better are the accuracies. Users with very fast (less than 100 sec) and very slow (more than 1 million seconds) average querying frequencies seem significantly less vulnerable compared to those with mediocre (35,000 seconds) frequencies. The very fast and very slow category users are those who send very few queries in immediate succession or spread their queries across 3 months duration. Since the queries available for analysis are few, the accura-

cies are bound to be less for these users compared to the ones belonging to the mediocre category.

We do not notice any significant effect of the sensitivity of query content on classification accuracies. However, for users who did not pose any insensitive queries (based on our categorization in Section 3.1), accuracies were found to be relatively lower. Therefore, based on our sensitive query classification, the users who fire a larger fraction of sensitive queries were better camouflaged by TMN than those who fire a larger fraction of insensitive queries. This might be because of the presence of many sensitive queries in the initial query set generated from the default RSS feeds.

Users who engage in web search only during weekdays turned out to be much better protected compared to those who pose queries only over weekends (queries of such users can be identified with almost 100% success). This is because if users send queries only during weekends, then whatever queries are seen during weekday must be generated by TMN, allowing for easy identification. Finally, from Table 2, we also observed that using different TMN average query frequencies would more or less provide the same level of accuracy. In other words, higher TMN frequency may not help in hiding user's query, contrary to one's intuition.

In summary, our results indicate that TMN is very susceptible to machine learning attacks. In fact, TMN could be weaker than what our attacks imply. This is because we only used some simple off-the-shelf classifiers with default parameters and this itself resulted in considerable accuracies. Use of better and stronger machine learning algorithms, with optimized parameters, is very likely to further increase the accuracies.

6 Conclusions and Future Work

In this paper, we focused on TrackMeNot (TMN), a real-world search privacy tool based on query obfuscation. We demonstrated that a search engine, equipped with only a short-term history of user's search queries, can break the privacy guarantees of TMN by only utilizing off-the-shelf machine learning classifiers. More specifically, by treating a selected set of 60 users – from the publicly-available AOL search logs – as users of the TMN software, we showed that user queries can be identified with an average accuracy of 48.88%, while the average TMN query misclassification rate was only 0.02%.

In the future, we are interested in exploring designs of novel classifiers which can take into account other attributes (such as query timestamps) and possibly improve identification of users' queries. Classifier and clustering accuracies can be improved by selecting better classifiers and fine tuning their parameters. We defer this task of improving the efficiency by optimized parameter selection to future work.

Acknowledgments. We are grateful to our shepherd Rachel Greenstadt and PETS'10 anonymous reviewers for their insightful feedback. We also thank Lisa Hellerstein for discussion on machine learning classifiers and her helpful comments on our work, and the developers of TMN – Helen Nissenbaum and Vincent Toubiana – for their useful suggestions on a previous draft of the paper.

References

1. AOL Search Log Mirrors, http://www.gregsadetsky.com/aol-data/.
2. M. Barbaro and T. J. Zeller. A Face Is Exposed for AOL Searcher No. 4417749. In *The New York Times*, August 09 2006.
3. Bruce Schneier: Schneier on Security: TrackMeNot. Available at: http://www.schneier.com/blog/archives/2006/08/trackmenot_1.html, 2006.
4. R. Chow and P. Golle. Faking contextual data for fun, profit, and privacy. In *ACM workshop on Privacy in the electronic society (WPES)*, 2009.
5. DTREG - Software For Predictive Modeling and Forecasting. Logistic regression. Available at http://www.dtreg.com/logistic.htm, Feb 2010.
6. R. Evans. Clustering for Clasification. In *Master's thesis, Computer Science, University of Waikato*, 2007. http://adt.waikato.ac.nz/uploads/approved/adt-uow20070730.091151/public/02whole.pdf.
7. E. Frank. ZeroR. Available at http://weka.sourceforge.net/doc/weka/classifiers/rules/ZeroR.html, Feb 2010.
8. E. Frank and R. Kirkby. Random tree. Available at ttp://weka.sourceforge.net/doc/weka/classifiers/trees/RandomTree.html, Feb 2010.
9. S. Hansell. Marketers Trace Paths Users Leave on Internet. In *The New York Times*, September 15 2006.
10. D. Howe and H. Nissenbaum. TrackMeNot: Resisting Surveillance in Web Search. In *On the Identity Trail: Privacy, Anonymity and Identity in a Networked Society, Ian Kerr, Carole Lucock and Valerie Steeves (editors)*, 2008.
11. R. Jones, R. Kumar, B. Pang, and A. Tomkins. "i know what you did last summer": query logs and user privacy. In *Conference on information and knowledge management (CIKM)*, 2007.
12. R. Jones, R. Kumar, B. Pang, and A. Tomkins. Vanity fair: privacy in querylog bundles. In *Conference on Information and knowledge management (CIKM)*, 2008.
13. R. Kirkby. Random forest. Available at http://weka.sourceforge.net/doc/weka/classifiers/trees/RandomForest.html, Feb 2010.
14. E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Symposium on Foundations of Computer Science (FOCS)*, 1997.
15. NYTimes: Google Resists U.S. Subpoena of Search Data, http://www.nytimes.com/2006/01/20/technology/20google.html?_r=1.
16. PlanetLab: An open platform for developing, deploying, and accessing planetary-scale services. Available at: http://www.planet-lab.org/.
17. F. Saint-Jean, A. Johnson, D. Boneh, and J. Feigenbaum. Private web search. In *ACM workshop on Privacy in Electronic Society (WPES)*, 2007.
18. Scroogle.org, http://scroogle.org/.
19. Tor Anonymizing Network, http://www.torproject.org/.
20. TrackMeNot: Browser Plugin, http://www.mrl.nyu.edu/~dhowe/trackmenot/.
21. B. Trancer. Click: What millions of people are doing online and why it matters. Hyperion, 2008.
22. Wikipedia. Alternating decision tree. Available at http://en.wikipedia.org/wiki/Alternating_decision_tree, Feb 2010.
23. I. Witten and E. Frank. *Data Mining–Practical Machine Learning Tools and Techniques, Second Edition*. Elsevier, 2005.
24. S. Ye, S. F. Wu, R. Pandey, and H. Chen. Noise injection for search privacy protection. In *Conference on Computational Science and Engineering (CSE)*, 2009.