# ESQABE: Predicting Encrypted Search Queries

Isaac Meers[1(✉)], Mariano Di Martino[1], Peter Quax[2], and Wim Lamotte[1]

[1] UHasselt - Hasselt University - tUL, Expertise Centre for Digital Media (EDM), Wetenschapspark 2, 3590 Diepenbeek, Belgium
isaac.meers@uhasselt.be

[2] UHasselt - Hasselt University - tUL - Flanders Make, Expertise Center for Digital Media (EDM), Wetenschapspark 2, 3590 Diepenbeek, Belgium

**Abstract.** All popular search engines implement HTTPS to protect the privacy of their users. Unfortunately, HTTPS encryption only covers Application layer headers and information will still leak through side-channels and other protocols used in a conversation between browser and server. This paper presents a novel eavesdropping approach called ESQABE, which combines these sources of information in order to determine what a subject is querying a search engine for in a real-life situation. To achieve this goal, packet length and timing information of the autocomplete functionality are used in combination with the home page contents of the search result links subsequently opened by the user. ESQABE is evaluated by automated tests using realistic search queries and based on real-life behavior. The technique is able to correctly predict the search query in 33% of the cases which is a significant improvement when compared to related work. In 41% of the cases, the correct query was included in the top 3 of most likely predictions. In most other cases no prediction could be made. To better protect the user, we contribute a browser extension that effectively hides the search query for the eavesdropper. The tool not only protects users but also visualizes what information is leaking to an eavesdropper.

**Keywords:** Eavesdropping · Privacy · Network security

## 1 Introduction

Search queries are often exploited as a valuable source to predict user interests. In contrast to simply opening a website for any reason, entering a search query in an engine often implies an actual interest in a specific subject. This could include highly personal (and thus, privacy-sensitive) information about users such as shopping interests, religious beliefs, political preferences, etc. Obtaining an anonymized set of queries has already led to the identification of a real person [1].

To protect the privacy of users, internet traffic nowadays is mostly encrypted. According to the Google Transparency Report[1], US users spent 97% of their time

[1] https://transparencyreport.google.com/https/overview.


Published by Springer Nature Switzerland AG 2021
A. Jøsang et al. (Eds.): SEC 2021, IFIP AICT 625, pp. 103–117, 2021.
https://doi.org/10.1007/978-3-030-78120-0_7

in Chrome on HTTPS websites in 2020. The main advantage of applying encryption from a privacy perspective is the protection against eavesdropping or man-in-the-middle attacks. However, HTTPS is an application layer protocol and does not cover metadata of lower layer protocols, such as IP addresses on the network layer. The amount of unencrypted information per packet is limited, but eavesdroppers can combine it with other sources and use it to identify their subjects. When a user searches, a lot of traffic which leaks metadata is generated. First, the autocomplete functionality generates traffic to keep the list of suggestions up to date. Next, when the user submits his query, the list of search results which contains hyperlinks to relevant websites is loaded. And when the user visits these websites, the browser will establish new HTTPS connections to the relevant servers to download web pages. This results in a temporary increase in network traffic flowing between the user and the internet. In this work, this meta-information is used to identify search queries entered by users. But as noted in Sect. 3.6, not all search engines are vulnerable and especially privacy aware search engines do a better job at protecting their users by sending less information to their servers. Concretely, the following contributions are presented in this paper:

- A novel approach, ESQABE, to determine the contents of search queries entered by subjects using only information available to a passive eavesdropper (i.e., HTTPS encrypted traffic). ESQABE outperforms existing approaches as it is able to correctly predict the search query in over 32% of the cases where other approaches as KREEP [8] only reach 15% on top-50 accuracy.
- A browser extension that protects users against the eavesdroppers using ESQABE, but also educates which data still leaks to an eavesdropper.
- The source code of both ESQABE and the protection tool, as well as the dataset of traces used for evaluation, are made available at https://github.com/isaacme/ESQABE.

## 2 Related Work

**Table 1.** Comparison of related work with ESQABE

| | KREEP (2019) [8] | Oh et al. (2017) [9] | Chen et al. (2010) [2] | ESQABE |
|---|---|---|---|---|
| Protocols | HTTPS | Tor | Wi-Fi WPA | HTTPS, DNS |
| Data needed | Language model and avg. typing speeds | Fingerprints for target keywords | None | None |
| Targeted keywords | In dictionary (>12k words) | 300 targets | Unlimited | Unlimited |
| Information sources | Auto-complete requests | Fingerprinting TOR-traffic | Auto-complete responses | Auto-complete + subsequent requests |

Other attempts that try to identify search queries sent by a user, operate on a different level compared to ESQABE. Table 1 shows a comparison of the major differences of ESQABE to the related works described in this section.

KREEP [8] – a passive eavesdropping approach – identifies the complete search query while only using the requests generated by the autocomplete functionality. They first extract the requests which are sent when a new character is typed into the search bar and use these to determine the total query length. Secondly, they tokenize these requests (separating spaces from other characters) to find the length of the different words in the query. Based on the inter arrival times between keystrokes, a neural network identifies words using a dictionary. The training set of this network was composed of keystrokes recorded from 83000 typists using QWERTY keyboards and typing English words. The predictions of the neural network are filtered based on the info of the tokenization algorithm. Finally, a language model is used to further reduce these predictions to a list of 50 hypothesis queries. For 15% of the tested search queries in [8], the generated list of 50 hypothesis queries included the actual query. This is clearly less effective than our approach, but the results cannot be compared directly as will be described in Sect. 4.

ESQABE builds further on the detection and tokenization algorithms of KREEP, these are discussed in detail in Sect. 3.2. In contrast to our approach, KREEP assumes the user is typing in English and using a QWERTY keyboard. It is also assumed that all words in the search query appear in the language model used by the eavesdropper. This limits the real-world effectiveness of KREEP, as users might search for brand names (typically limited in a dictionary) or in other languages, neither of which are included in the model. In ESQABE, we also investigate subsequent traffic (like site visits), but in KREEP solely the timings of keystrokes based on packet inter-arrival times are used.

In [9], website fingerprinting approaches are used to fingerprint individual queries or keywords. They take into account all traffic generated when searching, starting from the moment the user commences typing until 5 s after the results page is loaded. Instead of using HTTPS captures directly, the captures contained traffic as generated by the Tor browser. This attack identifies if the query contains one of the 300 predefined targeted keywords with a precision of 91% and a recall of 81%. Which specific keyword from the list was found could be determined with 48% accuracy. The use case of this approach differs from ours as [9] can be used to filter specific keywords in firewalls while ESQABE focuses more on query reconstruction without a limited predefined list.

An older attack [2] describes a method relying on the sizes of the responses for the autocomplete traffic. In contrast to our approach, this attack is described to work with Wi-Fi WPA encrypted packets and thus cannot use IP addresses to filter the autocomplete traffic. They determine the query incrementally by trying to add each character (a-z, ␣) and keeping the one with the response sizes closest to the captured response. This attack assumes that the eavesdropper who tries to guess the query gets the same suggestions as the target. But, this attack should be reevaluated as suggestions generated by search engines nowadays are tailored to the specific user resulting in different response sizes.

# 3 ESQABE: Encrypted Search Query Analysis by Eavesdropping

In this section we describe ESQABE, an approach that makes it possible for a passive eavesdropper to determine a user's search query from encrypted network traffic. The main idea of ESQABE is to combine the available metadata of the encrypted communication with information gathered from the unencrypted network traffic. This is subdivided in several steps of information retrieval:

1. The lengths of the words in the search query are derived from the autocomplete traffic. This information is used to generate a *pattern* of the query. This step is based on the detection and tokenization algorithms of KREEP.
2. A list of the search results opened by the user is composed. The sudden increase of bandwidth usage caused by opening a new website is used to detect when the user opened a result. Info from DNS queries and TLS Client Hello messages is used to identify which website the result linked to.
3. The home pages of these websites are visited and the words matching the patterns of the first step are extracted. For each word extracted, the amount of occurrences on the home pages is counted.
4. If a Wikipedia article was among the results the user opened, its fingerprint is used to reduce the list of potential queries.

Finally, a ranking of possible search queries is generated. The larger the amount of occurrences, the higher the word appears in the ranking.

## 3.1 Prerequisites

We assume the eavesdropper has the ability to capture encrypted network traffic traces from the client containing the entire search action (beginning when a user starts typing their query and ending when they stop opening results). These traces do not only contain the communication between the browser and the search engine server, but also other traffic generated by the computer (e.g., DNS traffic). This traffic can easily be captured by the owners of the internet gateways users connect with. Apart from home and company networks these also include those of hotels, airports, museums etc. On open Wi-Fi networks, not only the owners but also other users and even passersby can capture enough traffic. ISP's and regimes in less democratic countries are also positioned on the path between client and server. Users connected through a VPN hide the necessary data from most of the previous instances but make the data available to the VPN provider who they have to trust. The following realistic assumptions are made:

– All captured connections use HTTP/1.1 or HTTP/2. HTTP/3 and QUIC are out-of-scope of this paper. As Google Search on Google Chrome utilizes QUIC, a significant amount of queries is not directly susceptible to ESQABE. Nevertheless, a large number of queries is made using HTTP/1.1 and HTTP/2 (e.g., on networks blocking UDP, other browsers etc.).

- All websites visited by the user are encrypted through HTTPS. Extraction of information from HTTP traffic is straightforward as it is not encrypted. For this reason HTTP traffic is considered out-of-scope.
- It is assumed that for communication only TLS encryption is used. So IP, TCP and TLS headers are captured unencrypted.
- The user typed their complete search query without typographical mistakes or corrections. Some search engines correct minor mistakes after query submission. If the words of the correction are of the same length as the words in the erroneous query, then the effectiveness of ESQABE will not be decreased. According to [3], only 10–15% of queries entered contain errors.
- ESQABE is optimized for the extraction of search queries made using Google Search. Nonetheless, the techniques described can be used on other search engines as long as they provide an autocomplete functionality. This assumption is discussed in depth in Sect. 3.6.

### 3.2 Step 1: Extracting Search Query Length

**Table 2.** An overview of autocomplete requests on Google Search while entering *Hasselt University* as a query using HTTP/2. On the left, the details as visible in the browser are shown. On the right, the sizes of the packets as visible by an eavesdropper.

| Actual request | | | Eavesdropper | |
|---|---|---|---|---|
| Query parameter | HTTP header (B) | Increment | Frame length (B) | Increment |
| ... | | | | |
| hassel | 720 | +1 | 206 | +1 |
| hasselt | 721 | +1 | 206 | +0 |
| hasselt%20 | 724 | +3 | 208 | +2 |
| hasselt%20u | 725 | +1 | 209 | +1 |
| ... | | | | |
| hasselt%20university | 735 | +1 | 216 | +1 |

The first step taken in ESQABE is extracting the traffic generated by the autocomplete functionality to determine the lengths of the different words in the search query. As search engines encrypt their application layer data with HTTPS, extraction methods need to be based on the information available in the non-encrypted headers of underlying protocols. The method used to achieve this is based on the detection and tokenization mechanisms from KREEP [8].

Detecting the packets is based on the fact that the only difference between all autocomplete requests is the query parameter. As shown in the example of Table 2, the size of the header always increments by one byte for each new character added. A space is encoded as `%20` and thus uses 3 bytes. For websites using HPACK compression with HTTP/2, these increments are not as consistent. HPACK uses a static Huffman code to encode string literals in which most

alphanumeric characters are represented in 5 to 7 bits (in contrary to 8 bits with HTTP/1.1). As string literals cannot end in the middle of a byte, padding is used to fill the gap. When the added character is encoded in less bits than the padding in the previous request, the new request will have the same size. Nevertheless, this pattern of packets with incrementing size is still distinguishable from other traffic when HPACK is used. The right side of Table 2 shows an example of what the eavesdropper sees when HPACK is used, note that due to the compression a space causes a 2 byte instead of a 3 byte increment.

The detection of KREEP often fails on longer traces we used. KREEP is not optimized for traces that include the visits of the search engine results afterwards. To overcome this issue, a maximum is set on the time between key presses. Especially in the case of Google Search, detection of anomalies due to a parameter called `gs_mss` is improved. Not much details about the parameter are known, but when it suddenly appears it contains the query as typed until that point and keeps the same value for all subsequent requests. The increment in size it causes can be calculated and equals the length of the encoded query until this point plus the length of `&gs_mss=` which is 8 characters. Apart from adaptations to the length detection algorithm, a filter was added to only include IP addresses from the search engine – based on a reverse DNS lookup (for example, Google hosts are all within the domain `1e100.net`).

After the autocomplete requests are extracted, they are analyzed in the tokenization phase in order to determine the length of the different words in the query. The fact that a space is encoded as `%20`, which causes a 2 or 3 byte increment, makes it distinguishable from other (single-byte) characters for an eavesdropper. This way we can easily split the query into separate words. Note that not all search engines encode a space as `%20`; some use `+` which is encoded using a single byte. For this limited amount of search engines the tokenization approach cannot be used and as a consequence these search engines are not vulnerable to ESQABE. A comparison of the applicability of popular search engines is discussed in Sect. 3.6.

### 3.3 Step 2: Identifying Opened Search Results

Once a user has submitted a search query, a list of results is shown - including hyperlinks to the relevant web page. Opening a web page from this list implies that the user leaves the search engine and sets up (a) connection(s) to a new website. Although the IP address of the server of this website is visible to the eavesdropper, this does not imply that the specific website on that IP address can be determined. This is due to the very common use of name-based virtual hosting, where multiple sites are hosted on the same server. In this case, distinction between sites is made by its hostname in the `host` in the HTTP headers, which are encrypted by HTTPS and not visible to the eavesdropper. Nevertheless, even in such conditions, identification of the website is rather straightforward as its hostname is sent in different protocol layers as well. DNS and the TLS Server Name Indication extension (SNI) default to sending the hostname in plain text. In ESQABE, these hostnames are linked to connections by investigating the SNI

extension if available. For all other cases, a dictionary is created by examining captured DNS responses. If for one IP address several hostnames are found, the last entry that happened before the TLS Client Hello of the connection is used. The cases in which encrypted DNS or TLS Encrypted Client Hello are used will be discussed in Sect. 5.

Extracting the hostnames from the complete network traffic trace provides the eavesdropper with preliminary insights into what the user has connected to. However, this list not only contains hostnames of websites the subject intentionally visited as a result of the search query, but also includes others as a consequence of background traffic or third party resources used by these websites (e.g. scripts, images and style sheets). For example, the response of `uhasselt.be` indicates to the browser that it needs to fetch a script from `cdn.jsdelivr.net`. The web browser of the subject can only start to open new connections for these resources after the response of `uhasselt.be` starts arriving. Thus, the browser will always connect to the website requested by the user first (in this example `uhasselt.be`) and fetch the extra resources afterwards. Opening a web page causes a significant amount of data to be transferred between the subject and the internet. An eavesdropper will see this sudden increase in bytes transferred from and to their subject and can use this to identify a web page visit.

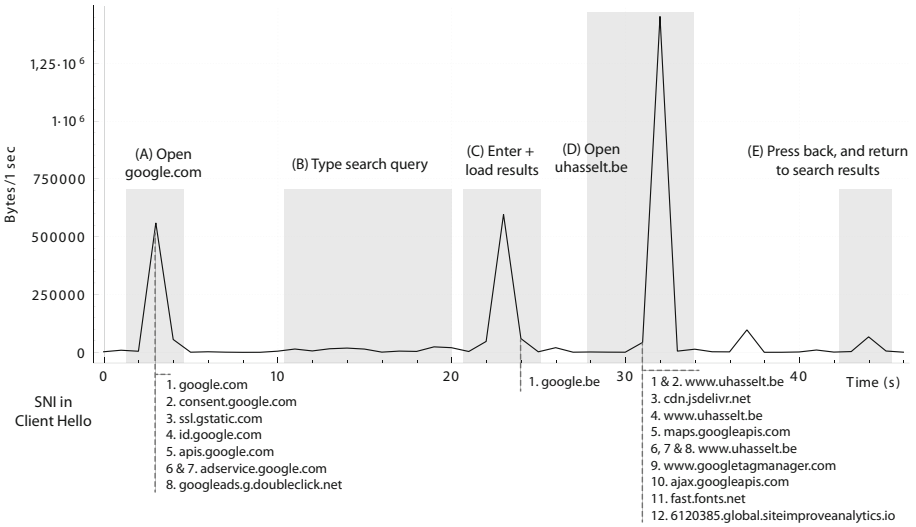


**Fig. 1.** Visualization of the amount of data when a user searches for Hasselt University and opens the university website. Below the graph, all SNI values for TLS Client Hello messages are shown (grouped by 2 s time frame).

The graph in Fig. 1 shows the characteristics of a page load visually. Phase A, C and D can all easily be detected due to the sudden change in data-rate. The amount of background traffic, that is being generated at the same time as

the user is visiting a web page, is insignificant in comparison to the page load and does not affect the graph. Users executing other more network-heavy apps generate more traffic and make the extraction of the page load harder. These are often related to streaming services and games for which filters could be written based on known IP ranges or domain names. The bottom of the graph shows the new TLS connections set up by the subject during the capture of this trace.

With all this information, we propose the following strategy to obtain a realistic list of websites visited by the subject. The algorithm assumes a limited time frame in which the search took place. The timestamp of the last autocomplete packet (step 1) is the starting point of the algorithm.

1. The first step is to extract all new connections opened in the given time frame. As web pages are loaded using HTTPS, each connection to a new website is set up using a TLS handshake. These all start with a client sending a *Client Hello* message, which is used as an indicator for a new connection.
2. The *Server Name Indication* extension or dictionary as described in Sect. 3.3 is used to link each connection with the hostname of the visited website.
3. At the beginning of the time frame, the subject just landed on the page containing the search results. From here, ESQABE iterates chronologically over all new connections. For each connection the following is done:
   (a) The total size of the traffic between the subject and the internet is calculated for a range of `MAX_LOAD_TIME` seconds after the *Client Hello* message of the connection.
   (b) If the resulting size is larger than `MIN_PAGE_SIZE`, it is expected that a web page was loaded right after the connection was set up. The hostname of the website is added to the list of potential visits.
   (c) The connections set up during the load of the page are those initiated by the page itself (i.e., resources such as images and scripts) and were not explicitly opened by the subject. To avoid these connections from being detected as real visits, all connections in the spike are ignored. The next connection processed is the first after the spike in network traffic.

The algorithm results in a list of websites the subject potentially visited after the search results page was loaded. In the algorithm two constant threshold values are used. These need to be tuned, depending on the network conditions for the connections ESQABE is deployed on.

- `MAX_LOAD_TIME`: The maximum expected load time of a web page. In practice this needs to be the length of the spike in traffic (for example phase D in Fig. 1) and can be determined automatically.
- `MIN_PAGE_SIZE`: The minimum size of a web page. If this value is too small, system background traffic will be classified as website visit. But if this value is too large, legitimate website visits could be ignored.

### 3.4 Step 3: Visiting Home Pages of Websites

Combining both the information about the length of the different words in the search query, as well as the identity of the websites visited subsequently, we

now try to deduce the complete search query. Websites mentioned in the search results are assumed to contain at least some information related to the search query, otherwise they would not have appeared in the search results. However, as explained earlier, we only know the hostname of the visited results and not the specific pages on that site. Fortunately, this does not imply that the home page cannot provide us with valuable information. We split this into several options:

- **The result links to the home page.** This is the most trivial case. When the subject opened a search result pointing to the home page of a website, this page will contain at least a part of the search query. Otherwise, it would not have appeared in the results. This happens when the user, for example, searches for a specific company, website, person or a keyword a company advertises on.
- **The result links to a Wikipedia article.** Wikipedia articles very often show up in the top search results. According to [6], a Wikipedia article appears in the first four results for 35% of their list of popular queries. For 73% of the queries of the dataset used in Sect. 4, the first page of search results contains a links to a Wikipedia article. As Wikipedia contains information about all sorts of topics, scraping its home page will not provide any useful information at all for ESQABE. Therefore, for Wikipedia pages, another approach is taken and described in Sect. 3.5.
- **The result links to another platform or webshop.** Platforms and webshops, like Booking.com, AirBnB, Steam and IMDB, provide a lot of items. On their home pages, they only feature a fraction of their assortment. This is however not implemented and left for future work, but we expect that depending on the platform, a similar approach to Wikipedia can be taken.
- **In all other cases,** the home page still provides a lot of relevant information about the topic. For example, company websites often feature several of their products on their home page, the home page of a television station lists the programs they broadcast and news websites feature their recent articles.

For each website in the list obtained in the previous section, the word(s) that match(es) the pattern of the search query are extracted. This is done by opening the home pages of the candidate websites with the Selenium WebDriver. To extract the word combinations, the following approach is taken for each website:

1. The home page of the website is fully loaded (until the `onload` event fires).
2. A small scroll movement is executed to trigger the loading of content below the fold, which due to lazy loading may not have been loaded otherwise.
3. All visible text is extracted from the `<body>` using the `innerText` property.
4. The strings of the title tag, image alt-tags, and selected page meta-tags (name, description, og:title, twitter:title, etc.) are extracted separately.
5. Once this extraction is finished, a regular expression generated from the word lengths of the search query is used to extract all matching strings.
6. The resulting sets of strings are combined in an associative array of counters with the string serving as key, ordered by decreasing number of appearances.

### 3.5 Step 4: Wikipedia

The home page of Wikipedia does not provide useful information for ESQABE. As there are millions of articles published on Wikipedia, it is impossible for an eavesdropper to determine which article was visited without additional information. However, if another result was opened besides the Wikipedia article, we can use the visit to Wikipedia to reduce the list of potential search queries by comparing web page fingerprints.

For each potential search query, the Wikipedia API is consulted to retrieve the corresponding article(s). Only the articles with the same title as the search query are selected. Sometimes a lookup will return a disambiguation page; in that case the 5 uppermost articles on this page are selected. If no relevant article can be found for a potential query, this query is ignored. Each selected article is visited in Google Chrome and Mozilla Firefox and separate network traffic traces are captured. These are provided (labeled) as the training set to a naive Bayes classifier as described by [7]. Note that these classifiers can also cause false positives. An existing implementation by Dyer et al. [5] is used in ESQABE. Eventually the capture of the Wikipedia traffic by the subject is tested against this classifier to determine which Wikipedia article was visited. The search query associated to this article is placed at the top of the list of potential search queries.

### 3.6 Vulnerable Search Engines

While we have mainly considered Google Search, other engines utilizing autocomplete functionality can be vulnerable to ESQABE as well. These engines all use HTTPS. The autocomplete requests vary subtly between search engines, as shown in Table 3. Some of them require small tweaks to ESQABE, especially during the phase of length detection, but the main concept remains unchanged.

**Table 3.** Comparison of applicability to multiple search engines

| | Google | Baidu | Bing | Yahoo | Yandex | DuckDuckGo | Startpage.com |
|---|---|---|---|---|---|---|---|
| HTTP | HTTP/2 | HTTP/1.1 | HTTP/2 | HTTP/2 | HTTP/2 | HTTP/2 | HTTP/2 |
| Space | %20 | %20 | %20 | %20 | + | + | + |
| Trigger | new character | new character | new character | new character | new character | new character 300 ms throttle | timeout |
| Special | `gs_mss` counter | counter | counter | / | / | / | random string |
| Applicable? | Yes | Yes | Yes | Yes | Only total length | No | No |

Firstly, the absence of HTTP/2 (e.g., Baidu) in autocomplete traffic makes it easier to detect sequences as every new character causes a 1 byte increase. Secondly, some search engines encode their spaces as a `+` instead of a `%20` in the autocomplete request, making a space indistinguishable from an alphanumeric character. However, while the length of the individual words in the query cannot be determined, their total length can still be utilized. Thirdly, DuckDuckGo

throttles its autocomplete requests for 300 ms, dropping redundant requests occurring within this interval. As the amount of requests is used by the eavesdropper to determine the length of the query, DuckDuckGo is not vulnerable to our method. Fourthly, Startpage.com goes a step further and does only request new suggestions at the moment the subject stops typing. In addition, they even include a header containing a random length string which is changed at every request. Consequently, Startpage.com is also not vulnerable to ESQABE. In contrary to the other search engines, DuckDuckGo and Startpage.com are mainly focused on protecting the privacy of their users. Apart from not using their data for advertisement purposes, we also discovered that they are better in protecting their users from potential eavesdroppers.

## 4 Experimental Evaluation

### 4.1 Approach

To simulate realistic searching behavior, queries from real life datasets were used in our automated testing approach. Few datasets are available due to obvious privacy concerns [1], but agencies specialized in search engine optimization keep lists of the globally most popular queries[2]. We used an extraction of this list from May 2020 which is calculated using data from the last 6 months. To also include regional queries, the most popular terms from Google Trends[3] were added. As search results were opened automatically we filtered out keywords related to adult and illegal content to prevent security and legal risks. The list was generated during May 2020 and consists of 452 unique queries.

The tests were all automated through a script that uses the Selenium WebDriver. All tests were executed using version 76 of Firefox on a device running macOS 10.14. As the exploited characteristics originate in the web application itself or in user behavior, the browser should not have large impact on attack performance. At the beginning of every set of tests, a fresh Firefox profile was generated, clearing all browser caches. During each test, TShark was used to capture a network traffic trace, just like a possible eavesdropper would. Firefox was instructed to generate an SSLKEYLOGFILE for debugging purposes; this way the trace could be decrypted if necessary for further investigation. For all queries in the dataset, three captures were executed with respectively one, two and three results opened afterwards.

A real user visiting a website is mimicked by the following steps:

1. Open a fresh browser window and navigate to https://google.com.
2. After the page is fully loaded, click on the search box.
3. Mimic a user who is typing the query by entering one character at a time and leaving pauses of 0.3 s between two characters. Note that the inter key interval has no impact on the attack performance on search engines like Google Search, which do not implement throttling.

[2] https://www.mondovo.com/keywords/most-searched-words-on-google/.
[3] https://trends.google.com/.

4. When done with typing, press enter and wait for the results page to load.
5. Extract the hyperlinks of the search results from the results page, and pick the results that will be opened. To simulate a real visitor, the results chosen are based on the actual click through rates of search results on a certain position. As Google does not provide this data, they are based on the analysis of 5 million search query logs by an external company[4].
6. A selected page is opened, and after the page has been loaded, a small scroll movement is executed to simulate a user searching for information. After a delay of 3 s, the back button is pressed and the next page is opened. This step is repeated until all selected pages are visited.
7. After the visits, the browser tab is closed and the capture is finished.

### 4.2 Results

**Table 4.** Correctly identified query length

| Type of trace | KREEP short[a] | KREEP long[b] | ESQABE long[b] |
|---|---|---|---|
| Correct total length | 99.70% | 19.13% | 92.90% |
| Correct word length | 74.89% | 19.03% | 87.97% |

[a] Captured traffic traces end after query was completely typed. Measurements as described in [8].
[b] Traces generated as described in Sect. 4.1

Detecting the lengths of the different words of the search query is the first step of ESQABE. The results of this steps are shown in Table 4. We noticed that the original version of KREEP underperformed in our tests in comparison to the measurements in [8]. The traces used to evaluate KREEP stop immediately after the query was typed and do not include the load of the search results page. Our traces are longer as they also contained the visits to the search result links, which caused issues with the detection phase. These issues appeared because the approach taken to detect the autocomplete traffic was not restrictive enough. For example, requests made by the browser to load the search results page were often identified as autocomplete requests. In ESQABE we made improvements to KREEP to overcome these issues as described earlier in Sect. 3.2.

The next step is the detection of the visited search results. The goal of this step is to identify the domain name of the result which the user opened. In 78.80% of all search results opened by the subject, the domain name was identified correctly by the algorithm. The algorithm failed sometimes when the subject opened multiple results of the same domain as in these cases no new TLS handshake took place. In other cases the algorithm wrongly identified a domain which hosts assets as the search result opened by the user. And in some cases small results were ignored as they were categorized as background traffic.

[4] https://backlinko.com/google-ctr-stats.

**Table 5.** Accuracy of ESQABE for complete search query identification. The table shows the amount of test cases for which the correct query could be determined.

| # of opened search results | Correct guess | Top-3 accuracy |
|---|---|---|
| 1 | 27% | 35% |
| 2 | 35% | 44% |
| 3 | 36% | 46% |
| 1, 2 or 3 | 33% | 41% |

And finally, the total performance of ESQABE is evaluated and shown in Table 5. The more search results are visited by the subject, the more information is available to the eavesdropper and the more search queries are detected correctly. Succeeding in, or failing with this approach is highly dependant on the search queries and the websites visited afterwards, as already discussed in Sect. 3.4. Overall, ESQABE outperforms KREEP as it succeeded in predicting the specific query for 33% of our tests. In [8], KREEP returned 50 hypothesis queries for each test case and in up to only 15% of the cases this included the actual query. However, note, that the results cannot be compared directly as the testing strategies differ. KREEP was tested on pieces of sentences extracted from the Enron email corpus and English gigaword newswire corpus as for these key inter-arrival intervals are available. These extractions are valid English clauses, but do not reflect what a real user would enter in a search engine. We, on the other hand, need realistic queries to evaluate ESQABE. This means that the results of KREEP and ESQABE cannot be directly compared side by side; a dataset of keystrokes of realistic search queries would need to be available.

Note that ESQABE has its limits. It is mostly interesting for eavesdroppers who target a specific victim and follow them for a longer period of time to observe patters in the search behavior. Then, they can take into account the second and third guesses which are often correct as shown in Table 5. Also note, we simulated a user who is focused on one task. A user who extensively multi tasks and constantly switches between websites can load other pages while making a web search. This can currently not be detected and can mislead ESQABE. However, some search engines generate requests for logging when a user opens a result, it is left for further research if this can be used to differentiate traffic.

## 5 Defense Mechanisms

It is clear that HTTPS on its own is insufficient to protect users from eavesdroppers. The combination of limited pieces of – sometimes seemingly innocent – information, can enable an eavesdropper to find out a lot about their subjects. Luckily, when a piece of information is missing, ESQABE and similar methods become harder or even impossible to execute. Using minor defense mechanisms can yield a significant impact, even without limiting functionality for end users. We created a browser extension that implements multiple defense mechanisms.

The first set of defenses focuses on preventing the detection of query length. Some of these were already suggested in [8]. As described earlier, Startpage.com appears not to be vulnerable for our eavesdropping approach and is used as a source of inspiration as well. A first, non-intrusive, approach is the addition of padding to the autocomplete requests. Adding random padding – even a small amount – makes it impossible to tokenize the words of the search query. The differences between a space and a normal character disappear in the padding.

Padding does not protect against detection of total query length. Rather, it will make length guessing only harder as the pattern is less distinguishable. To effectively hide the length, the request length should be in range of other background traffic on the same TCP connection (e.g., traffic for analytics), which is impossible to guess for future requests. Combining random padding with throttling of the requests would effectively hide query length for the eavesdropper as ESQABE counts the autocomplete requests for its guess. When throttling is enabled in the extension, it intercepts the autocomplete requests and drops them during certain intervals. However, throttling will impact the user experience as the refresh rate of the suggestions is limited. The approach of DuckDuckGo is to wait at least 300ms between requests, which is less intrusive than Startpage.com which debounces for 500ms. Just as Startpage.com, our extension also provides an option to completely turn off autocompletion.

The second line of defense is to make it impossible to find the domain the subject connected to. Encrypting DNS and TLS Client Hello messages would be a good starting point, as extraction then needs to shift to other less straightforward approaches. New protocols as DNS-over-HTTPS and TLS Encrypted Client Hello (ECH) are created to solve these problems. Our browser extension guides the user in the processes to enable ECH and shows a warning when it is not enabled. However, these protocols do have their flaws as well. In [10], a fingerprinting attack is discussed which can be used to identify selected domains from DNS-over-HTTPS traffic. In [4], another approach is proposed based on extraction from reverse DNS and the Subject Alternative Name field present in HTTPS certificates. This technique does not need the generation of fingerprints in advance and achieved an accuracy of 50.5% when tested on all websites in the Tranco top 6000. Encrypted DNS and ECH make the detection harder but not impossible for the eavesdropper. Another option is using decoy traffic to mislead eavesdroppers. This approach involves additional network traffic which can delay the effective page load. In our extension it is possible to enable random visits to other Wikipedia articles in the background.

## 6 Conclusion

In this work we have shown that eavesdroppers can combine pieces of seemingly innocent leaked information in order to obtain more intelligence about their subjects. We have described ESQABE, a technique to derive the text string a user entered as a query in a web search engine from information available to an eavesdropper. The approach is, contrary to previous work, applicable in a real

world scenario independent of the language and contents of the query. In our open world approach only a limited number of assumptions is made and with a correct answer in 33% of all test cases, we have shown the viability of our approach. There are surely opportunities to extend ESQABE in order to avoid certain limitations and increase its effectiveness. For example, the applicability of this approach on traffic traces containing QUIC traffic needs further investigation. However, as every key press generates a new autocomplete request, they still have the potential to be recognized by an eavesdropper. We also created a browser extension which implements basic defense mechanisms against ESQABE. Source code for both ESQABE and the browser extension is made available at https://github.com/isaacme/ESQABE, together with the captured datasets.

## References


1. Barbaro, M., Zeller, T.J.: A face is exposed for AOL searcher no. 4417749. https://www.nytimes.com/2006/08/09/technology/09aol.html. Accessed 23 Nov 2020
2. Chen, S., Wang, R., Wang, X., Zhang, K.: Side-channel leaks in web applications: a reality today, a challenge tomorrow. In: 2010 IEEE Symposium on Security and Privacy, pp. 191–206 (2010). https://doi.org/10.1109/SP.2010.20
3. Cucerzan, S., Brill, E.: Spelling correction as an iterative process that exploits the collective knowledge of web users. In: Proceedings of EMNLP 2004. pp. 293–300 (July 2004), https://www.aclweb.org/anthology/W04-3238
4. Di Martino, M., Quax, P., Lamotte, W.: Knocking on IPs: identifying https websites for zero-rated traffic. Secur. Commun. Networks (2020). https://doi.org/10.1155/2020/7285786
5. Dyer, K.P., Coull, S.E., Ristenpart, T., Shrimpton, T.: Peek-a-boo, I still see you: why efficient traffic analysis countermeasures fail. In: 2012 IEEE Symposium on S&P, pp. 332–346. IEEE (2012). https://doi.org/10.1109/SP.2012.28
6. Lewandowski, D., Spree, U.: Ranking of wikipedia articles in search engines revisited: fair ranking for reasonable quality? J. Am. Soc. Inf. Sci. Technol. **62**(1), 117–132 (2011). https://doi.org/10.1002/asi.21423
7. Liberatore, M., Levine, B.N.: Inferring the source of encrypted http connections. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, New York, pp. 255–263. CCS 2006. Association for Computing Machinery (2006). https://doi.org/10.1145/1180405.1180437
8. Monaco, J.V.: What are you searching for? a remote keylogging attack on search engine autocomplete. In: 28th {USENIX} Security Symposium ({USENIX} Security 19), pp. 959–976 (2019)
9. Oh, S.E., Li, S., Hopper, N.: Fingerprinting keywords in search queries over tor. Proc. Priv. Enhancing Technol. **4**, 251–270 (2017). https://doi.org/10.1515/popets-2017-0048
10. Siby, S., Marc, J., Diaz, C., Vallina-Rodriguez, N., Troncoso, C.: Encrypted DNS privacy? → a traffic analysis perspective. In: NDSS. Internet Society (2020). https://doi.org/10.14722/ndss.2020.24301