UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Deducing search queries from encrypted network traffic

Isaac Meers
Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :
Prof. dr. Peter QUAX

2019
2020

De transnationale Universiteit Limburg is een uniek samenwerkingsverband van twee universiteiten in twee landen: de Universiteit Hasselt en Maastricht University.

UHASSELT
KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

Maastricht University

Faculteit Wetenschappen
School voor Informatietechnologie

master in de informatica

Masterthesis

Deducing search queries from encrypted network traffic

Isaac Meers

Scriptie ingediend tot het behalen van de graad van master in de informatica

PROMOTOR :

Prof. dr. Peter QUAX

HASSELT UNIVERSITY

Abstract

Master in Computer Science

Deducing search queries from encrypted network traffic

by Isaac MEERS

User data is has become more valuable than ever before. Companies are interested for the purpose of targeted advertisements and attackers are looking for information to perform social engineering attacks. HTTPS was introduced to encrypt and secure website visits and make it impossible for attackers to eavesdrop on the contents of the communication. But, HTTPS does not encrypt everything and information does still leak. This theses presents a novel approach called ESQABE which uses this information in order to determine what a victim is searching for using a search engine. This is done by combining several different pieces of information as for example the length of packets and the websites visited afterwards. ESQABE is evaluated by automated tests and could correctly predict the search query in more than 32% of the cases. In more than 41% it even appeared in a list of three suggestions made. In order to protect the user, a browser extension was created which effectively hides the search query.

Acknowledgements

The realisation of this thesis was not possible without the help of several people.

At first, I want to thank my mentor Mariano Di Martino for his insights and feedback. I have found the discussions during our weekly meetings extremely interesting and they helped me completing this thesis. He has put an extensive amount of work in answering my question and proofreading my thesis for which I am very grateful.

I would also like to thank my promotor Prof. dr. Peter Quax for giving me the opportunity to work on this interesting topic. His tips, feedback and support were vital in order to achieve this result.

In extend, I want to thank the staff of the networking and security research group for their thoughts on my work.

Furthermore, I am grateful for my friends. We did not only have fun together but they also provided me with feedback about the topic and my writing skills.

And at last, I want to thank my family and especially my parents for supporting me in pursuing this Master's degree. Their patience and support was indispensable during stressful times.

Contents

Abstract i

Acknowledgements ii

1 Introduction 1

2 Background 3
- 2.1 Eavesdroppers and man-in-the-middle attacks . . . 3
- 2.2 HTTP over TLS . . . 4
- 2.3 HTTP/2 . . . 6
  - 2.3.1 Streams and Multiplexing . . . 7
  - 2.3.2 HPACK - Header Compression for HTTP/2 . . . 7
- 2.4 Domain names and IP addresses . . . 10
  - 2.4.1 DNS: Domain Name Service . . . 10
  - 2.4.2 Name-based virtual hosting . . . 12
  - 2.4.3 SNI: Server Name Indication . . . 12
- 2.5 Webpage fingerprinting (WPF) . . . 13

3 ESQABE: Encrypted Search Query Ascertainment By Eavesdropping 15
- 3.1 Attack description . . . 15
- 3.2 Extracting search query timings and length . . . 17
  - 3.2.1 KREEP: Remote keylogging attack on search engine autocomplete . . . 17
  - 3.2.2 Integration in ESQABE . . . 19
  - 3.2.3 Experiment: Can HTTP/2 multiplexing be/become an issue? . . . 22
- 3.3 Domain names . . . 22
  - 3.3.1 Fingerprinting DNS-over-HTTPS requests . . . 23
  - 3.3.2 OpenKnock: Identifying HTTPS Websites in an open world scenario . . . 24
- 3.4 Detecting the visited websites . . . 24
  - 3.4.1 Information available for the eavesdropper . . . 24
  - 3.4.2 Implementation . . . 27
- 3.5 Visiting web pages . . . 28
- 3.6 Using Wikipedia fingerprints . . . 29
  - 3.6.1 Selecting potential Wikipedia articles . . . 30
  - 3.6.2 Identifying the visited article . . . 31
  - 3.6.3 Broadening this approach . . . 32
- 3.7 Extraction of non-Google search queries . . . 32

4 Evaluation of ESQABE 34
- 4.1 User Study . . . 34
  - 4.1.1 Early approaches . . . 34
  - 4.1.2 Testing approach . . . 35

4.1.3 Technical setup . . . 36
4.1.4 Cancellation of the user tests . . . 36
4.2 Automated tests . . . 36
4.3 Results . . . 37
4.3.1 Detecting query length . . . 37
4.3.2 Detection of the visited search results . . . 38
4.3.3 Identification of search queries . . . 39
4.4 Comparison with KREEP . . . 39
4.5 Potential improvements . . . 40
4.5.1 Using WikiData . . . 40
4.5.2 Detection of typographical errors . . . 41
4.5.3 Exploiting incremental HPACK . . . 41
4.5.4 Search it yourself . . . 41

5 Defences **42**
5.1 Defending against search query length leakage . . . 42
5.1.1 Random padding . . . 42
5.1.2 Throttling requests . . . 43
5.2 Hiding domain names . . . 44
5.3 Misleading ESQABE . . . 45
5.4 Educating users . . . 45

6 Conclusion & reflection **47**
6.1 Conclusion . . . 47
6.2 Reflection . . . 48

A Dutch Summary **49**
A.1 Inleiding . . . 49
A.2 Geëncrypteerd verkeer afluisteren . . . 50
A.3 Het achterhalen van zoekopdrachten . . . 52
A.4 Evaluatie van ESQABE . . . 53
A.5 Gebruikers beschermen . . . 54
A.6 Conclusie . . . 55

Bibliography **56**

Chapter 1

Introduction

> Personal data is the oil of the 21st century, a resource worth billions to those who can most effectively extract and refine it [Dance et al., 2018].

Data may not be as vendible as oil, it still is an enormous valuable resource for companies [Martínez, 2019]. For example, according to the financial report of Facebook, the average user of the platform generated a revenue of $6.03 during the first quarter of 2020 [Facebook, 2020]. Facebook does not ask its users to pay a fee, but uses the data to generate revenue. This is common practice for internet companies, they use the data of their customers to sell specific targeted advertisement spots on their platforms. As targeted advertisements achieve better results until a certain extend, companies like to sell these spots as they often do not have a direct channel to target these people themselves [Farahat and Bailey, 2012].

For companies which literally live on the internet, like Google, Facebook and Amazon, it is rather easy to collect user data as they can monitor the actions their customers execute on their platforms. However, they are not the only companies which have access to these online actions. Every company involved in the transportation of these actions will have access to the raw data an end user transmits. These companies do not only include the internet service provider at home, but all places a user connects to the internet. For example, locations as a hotel, airport, restaurant, café, amusement park, public transport, and many more provide internet access to their customers. And as providers of internet, these companies can also eavesdrop on their customers [Cheng et al., 2013].

But not only legitimate companies try to eavesdrop on their users, also attackers try to exploit these situations. In order to execute a successful social engineering attack for example, attackers need to know some details about their victims. This can be the job of the victims, their hobbies, political interests or others [Krombholz et al., 2015]. One way to gather those, is by eavesdropping on the network connection of the victim. In the case of an open Wi-Fi network, this can be as easy as listening with an antenna [Cheng et al., 2013; Sombatruang et al., 2018], but also on several private networks it appears to be possible to eavesdrop on traffic [Vanhoef and Piessens, 2017]. Attackers might even setup an evil twin access point, with which they try to mislead victims to connect to their network by pretending to be legitimate [Lanze et al., 2014].

Luckily for the victims, not all raw data makes sense to the eavesdroppers. In the early days, this raw data contained a lot of information which these intermediates could read. The internet was originally built as a network between universities and they all trusted each other. Privacy and security were not an important factor, in

fact every intermediate transferring the communication could read all the contents. This, however, is changing, and multiple new standards were introduced to protect the data and privacy of the end user. For example, HTTPS encrypts HTTP traffic and assures the end users they are communicating with the websites they indented to visit and that no one else can read their communications [Rescorla, 2000].

In fact, HTTPS became one of the main protection mechanisms of the internet as it effectively hides the contents of the requests and responses between two instances. In main stream media, the protocol is often suggested for end users as one of the methods to protect themselves when using public Wi-Fi networks [Nielo, 2018]. This is true to a certain extend. HTTPS protects users from leaking their passwords and valuable information, but as will be described in the following chapters: it does not hide everything. For example, domain names of the websites visited are still visible. Eavesdroppers can still try to learn more about their victim, but how far can they go, what is interesting for them and can victims protect themselves? Eavesdroppers are not just looking for a list of websites visited by their victims but they try to gather information with which they can build a profile of their victim. This eventually resulted in the following research questions:

- Is it possible for an eavesdropper to deduce what a victim is searching for on a search engine in a real life situation?
  - Can an eavesdropper detect when a victim is searching?
  - Can an eavesdropper deduce which search results the victim opens?
- If an eavesdropper can deduce search queries, how can victims protect themselves against and hide their search queries?

Search queries were not coincidentally chosen as the main subject. Connections with servers in general can be setup for various reasons, for example background traffic by the operating system or opening an uninteresting link from an email. Search queries, in contrary, are made by a user when he is actually looking for some piece of information. The end user is probably interested in the topic he is searching for, and this is of interest for advertisers [Rose and Levinson, 2004; Taghavi et al., 2012]. Take the following example: a user is sitting in a fast food restaurant, connects to the Wi-Fi, and searches for new smartphones. If the restaurant knows this, they can try to sell advertisement space on their television screens to smartphone manufacturers. These will be interested if the restaurant tells them a customer is searching for a new one. However, they cannot deduce this by simply looking at the traffic and seeing communication with a server from Apple, as this can be background traffic by the smartphone itself. If they know the search query, they can be more sure of the incentives of their customer.

The questions will be answered as a feasibility study where a new technique is created by building further on existing techniques and novel approaches. Chapter 2 gives an introduction in the techniques already used by eavesdroppers and the protection mechanisms which are already introduced. Chapter 3 answers the first question from the perspective of an attacker, and the effectiveness is evaluated in chapter 4. The second question is answered from the perspective of a victim in chapter 5. The eventual result of this thesis will not only show the possibility of a new attack but immediately provide defences.

Chapter 2

Background

In this chapter, the reader is introduced to some aspects of networking and security used in this thesis. It is expected that the reader has a basic knowledge of computer networking, like the Ethernet and IP protocols. In this section, first the basic idea of an eavesdropper or man-in-the-middle is defined. This way, the reader will be able to use the mindset of these attackers to gain deeper insight into multiple protocols and techniques used to create the internet as we know it today. Viewing it form an attacker perspective will show which valuable information is leaked by a potential victim.

2.1 Eavesdroppers and man-in-the-middle attacks

Communication between two parties over the internet travels through cables, air, and is processed in a lot of intermediate devices. When an attacker manages to become part of the path between these parties and exploits this position, the attacker is called a man-in-the-middle. While communication between the original parties goes on, the attacker can eavesdrop and alter the messages exchanged. A successful execution of a man-in-the-middle attack assures that both communicating parties do not notice that their communication is read and/or altered. The main difference between man-in-the-middle attackers and eavesdroppers is the altering behaviour. Eavesdroppers function as taps and do not interfere in the network traffic they can intercept, they behave passively. Man-in-the-middle attackers, on the other hand, behave actively, they can for example alter, delete, or delay packets they see passing by. Gaining a position in path is often easier for an eavesdropper than a man-in-the-middle, as the eavesdropper requires less technical abilities.

Gaining this position can be done in different ways. An attacker can use a Wi-Fi antenna and intercept all traffic of users connected to open networks. Password protected networks normally encrypt their traffic, but a flaw in the WPA2 personal protocol enabled an attack called KRACK, with which attackers could decrypt this traffic [Vanhoef and Piessens, 2017]. Not everyone needs to set up an attack to gain a position in the path. ISPs, for example, posses the infrastructure used by a user to connect to the internet. As a consequence, a user sends all his traffic to the ISP which needs to forward it, but the ISP can also access the data inside the packet. Actually, everyone providing internet access to users own a device which is part of the path. This also includes hotel, cafe, and restaurant owners which provide their customers with internet access.

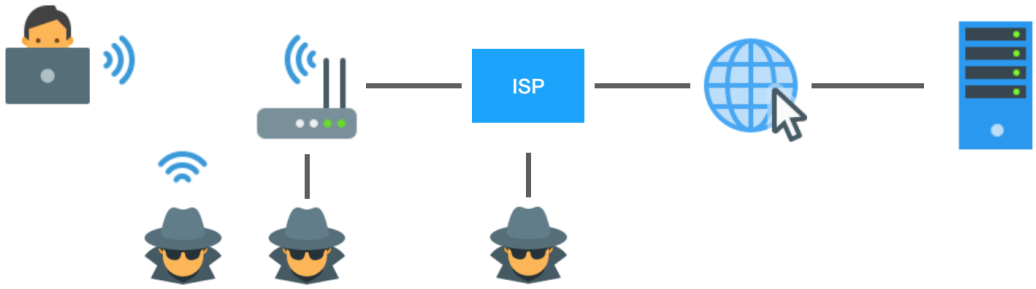


FIGURE 2.1: Possible positions of an eavesdropper.
Icons from Icons8

Luckily for the internet user, being in the right position does not imply that an attacker can read all communication. When two parties are encrypting their communication, a man-in-the-middle has to have access to the encryption key to be able to decrypt the communication. The issues of possible eavesdroppers on the path are already known since the introduction of the internet which was built on trust. Users started to protect themselves and these days a lot of internet traffic is encrypted. According to Google, 84% of the webpages loaded by Chrome for Windows were served using HTTPS encryption in 2019 [*HTTPS encryption on the web - Transparency Report*]. An attacker does not necessarily need to gain a position on the path to be able to eavesdrop. For example, other researchers succeeded in gaining a position inside the web browser using a browser extension [Rauti and Leppänen, 2012]. In the browser, they have direct access to the plain text data and capturing the encryption key is not necessary anymore.

In the next sections of this thesis, when an eavesdropper or a man-in-the-middle attack is mentioned, we suppose the attacker sits in between the two communicating parties. The attacker only has access to the data which passes through him and has no access to the plain text data if it was encrypted by the sender, in contrary to a man-in-the-browser. The positions are illustrated in figure 2.1. An example can be a restaurant that provides free internet to its customers or an attacker who uses an antenna to eavesdrop on open Wi-Fi networks.

2.2 HTTP over TLS

The HTTP protocol was originally built without privacy and security in mind. All HTTP traffic is not encrypted and the user needs to trust the network that it is just doing what he asks for. This makes communication with the protocol vulnerable to attacks like eavesdropping and content hijacking. And, the end user also has no way to validate if the content he receives is really coming from the server he wanted to connect with. An adversary could act as if he is the real server and send a fake response to the user.

HTTPS was introduced to fix these problems by providing encryption, data integrity and authentication. To accomplish this, HTTP messages are encrypted using Transport Layer Security (TLS), so an adversary cannot read or modify the communication. Clients can verify the identity of the server they want to communicate with by

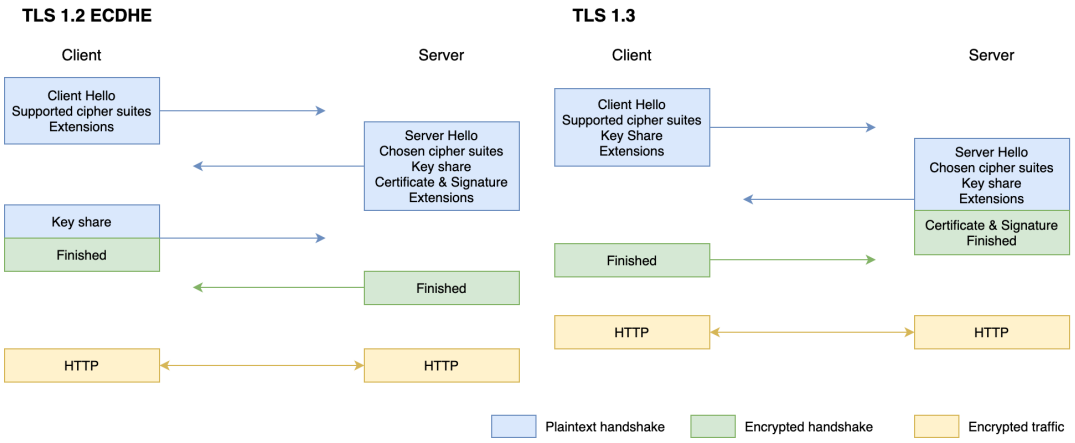


FIGURE 2.2: A side-by-side overview of a TLS handshake of v1.2 (left) and v1.3 (right) [Valsorda, 2016]

validating the certificate of the server using Certificate Authorities [Rescorla, 2000][1].

These measurements require some extra steps by the client and the server before they can start communicating. These steps are called the TLS handshake. Next, we describe the handshake of TLS v1.2 using the elliptic-curve Diffie-Hellman key exchange (ECDHE). Communication over HTTPS is initiated by a client opening a TCP connection to a web server using the TCP 3-way handshake. After this connection is set up, the TLS handshake is performed. In this handshake, the server tries to prove his identity to the client and they exchange messages to make some agreements about the encryption algorithms used. The client initiates the handshake by sending a *ClientHello* message. In this message the client includes information about the version of TLS used and the encryption algorithms he supports. The server replies to this message with a *ServerHello* message containing the encryption algorithm it chose, its certificate containing its public key and a key share. The client verifies the certificate against a Certificate Authority to be sure he is talking to the right server. If the certificate is valid, he shares his key share with the server. Now they both calculate the key they are going to use for the communication[2]. When the calculations are done, they send a finished message to each other, note that this message is already encrypted using the freshly generated key. An overview of this handshake is drawn on the left side of figure 2.2 [Rescorla and Dierks, 2008].

In TLS v1.3, a new approach to this handshake is taken to make it not only faster, but also more secure. The *Client Hello* message is not only used to announce the supported cipher suites, the client can already include multiple key shares in this message. So when the server chooses a cipher suite from the list, he can use the corresponding key share of the client to immediately generate the encryption key and send his own key share back to the client. As the actual key can be generated earlier in the process, more parts of the handshake can be encrypted than with TLS v1.2. This difference is shown with other colors in figure 2.2. In TLS v1.2 only the *finished* messages and the actual traffic are encrypted. This implies that the certificate is sent in plain text from the client to the server. An adversary eavesdropping the connection can use this certificate to identify the owner of the server which the client

[1] For completeness, there are exceptions in which HTTPS does not require a certificate but these are often not supported by web browsers.

[2] An in depth explanation of the Diffie-Hellman Key Excange algorithm can be found in RFC 2631 - `https://tools.ietf.org/html/rfc2631`

FIGURE 2.3: A partial screenshot of the TLS certificate used on `uhasselt.be`

is communicating with. In TLS v1.3, on the other hand, the server can already encrypt its certificate when sending it to the client as the key is known earlier. Now an eavesdropper cannot read the certificate, which contains valuable information about the visited website. This includes the domain name and name of the organisation, as for example show in figure 2.3.

It is important to mention that even in TLS v1.3, the initial *Client Hello* and *Server Hello* messages are sent in plain text. These initial messages contain multiple extensions to make additional agreements about for example the cipher suite or the requested website. As these messages and extensions are not encrypted, the information is visible to an eavesdropper who can exploit it [Rescorla, 2018].

After the handshake is finished, both parties will exchange HTTP traffic using the TLS stream. This way, their HTTP frames are encrypted and an attacker cannot eavesdrop anymore on the actual data exchanged. But some metadata about the communication is still visible. To understand why, HTTPS needs to be put in the bigger picture. As shown in figure 2.4, TLS is positioned in the application layer of the TCP/IP stack. In this stack, lower-layer protocols package higher layer protocols. So only HTTP is packed into TLS, all the other layers are not encrypted by TLS. This, however, is not without reason. When routing the packet over the internet its destination needs to be known by the routers to be able to perform routing. An adversary eavesdropping the traffic, who can sometimes be a router itself, can also read this metadata. Other information like timestamps and encrypted package sizes are easily deducible by inspecting the characteristics of the packet passing by. The use of TLS encryption does not influence the original packet size [Rescorla, 2018].

2.3 HTTP/2

Not only TLS v1.2 could be sped up, some characteristics of HTTP/1.1 have a negative impact on the overall performance when browsing. To impove this performance, HTTP/2 was created [Belshe et al., 2015]. Some design changes of HTTP/2 also have an impact on the metadata leaked in encrypted traffic. Moreover, all major HTTP/2 client implementations[3] require HTTP/2 to be encrypted, making the adoption of TLS even higher. In this section changes impacting our attack are described.

[3] Firefox, Chrome, Safari, Opera, IE and Edge

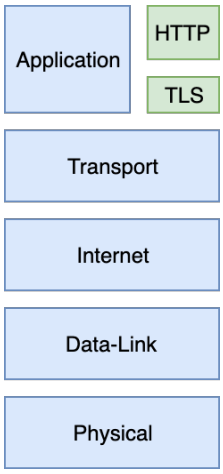


FIGURE 2.4: The position of TLS in the TCP/IP stack

2.3.1 Streams and Multiplexing

HTTP/1.1 only supports one active request at a time, which causes head-of-line blocking. Browsers tried to solve this by opening multiple parallel TCP connections, which each could handle separate resources. This is not an ideal solution as for example re-prioritisation is not possible. That is why HTTP/2 introduces multiplexing and streams. Multiple HTTP requests/responses can be send over the same TCP connection, browsers and servers can split messages into multiple frames and schedule these as they prefer. This implies that a request, for example, can be split over multiple TLS records or TCP segments, but also that multiple frames can be combined in a single TLS record / TCP segment [Grigorik, 2014].

This behaviour influences the amount of information visible to an eavesdropper. As normally he could assume a single TLS record represented a single HTTP frame, but this is not the case anymore. And an eavesdropper needs to keep this in mind when analysing captured traffic.

2.3.2 HPACK - Header Compression for HTTP/2

All versions of HTTP use Header fields to enhance the requests and responses with various types of metadata. Some header fields differ for each message as for example the `:path` header in a request. This header identifies the specific resource which is requested by the browser. But other header fields are the same for multiple requests or responses as for example the `user-agent`, `cache-control` or `cookie` headers. The last one in particular is often used to add some kind of state to the stateless HTTP protocol and is not changed often over the time of a website visit.

In an attempt to optimise bandwidth usage, the HTTP/2 protocol defines a method to compresses header fields [Peon and Ruellan, 2015]. The original approach introduced by HTTP/2s predecessor SPDY, using the *DEFLATE* format, was vulnerable to the CRIME attack where the compression leaked information when it was encrypted [Rizzo and Duong, 2012]. This is why HPACK was introduced for HTTP/2.

HPACK uses three principles to compress the headers:

- A Static Table: This table contains a predefined and unchangeable list of 61 header fields. This list is made up of the most frequently used fields by popular

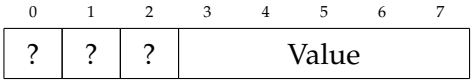


FIGURE 2.5: HPACK Integer value encoding within a 5-bit prefix [Peon and Ruellan, 2015]

websites. Some headers which have a limited amount of frequent values are stored together with their value as a single entry.

- A Dynamic Table: This table is of a limited size an is initially empty. It is filled gradually with new headers encountered during the connection. The table is maintained in first-in, first-out order, so when the table is full and a new element needs to be added, the oldest entry is removed. The protocol does not require all entries to be added into the table, it is up to the encoder to decide if it is efficient to add one. For example, adding the `:path` header to the table would probably be inefficient as it changes for every request.
- Huffman encoding: String literals which do not have an entry in one of the tables are encoded using a static Huffman code defined in the RFC. The Huffman coding table is generated especially for HTTP headers using a large real world sample.

The index of every entry in both tables is used as the unique identifier for this entry. An HPACK encoder can use this identifier to express the value it resembles and does not need to literally encode the value. The use of the dynamic table enables an encoder to only send the literal value of a header once and use the index for the following messages.

HPACK compression makes life of the eavesdropper harder, an eavesdropper investigating encrypted HTTP/2 traffic will for example struggle with the dynamic table. Using this implies that when a client sends the same request twice to the server, the second request can be significantly shorter than the first one. As this dynamic table can confuse an eavesdropper, he needs to take into account the fact that this can happen.

The compression of string literals, on the other hand, can possibly provide interesting information to an eavesdropper. To fully understand this, the encoding of integers is explained first as these are used to express string length. For storage optimisation purposes, integer encodings do not have to start at the beginning of an octet but can start anywhere within an octet. On the other hand, the encodings do always fill up until the end of an octet. The part of the integer encoding which finishes the first octet is called the prefix. For example in figure 2.5, the integer encoding starts at the fourth bit of the octet making it a 5-bit prefix.

Integers (I) strictly less than $2^N - 1$ can be encoded in an N-bit prefix and do not need additional octets. When the integer is larger or equal, the prefix is filled with 1's and the remaining part of the integer (I - $2^N - 1$) is encoded in a list of one or more octets. The first bit of an octet is used to report which octet is the last one. The other 7 bits are used for the value as shown in figure 2.6. Theoretically, an integer of the size $2^N - 1$ can be encoded in an N-bit prefix. But as this would be the same as filling the prefix with 1's, a decoder cannot know if an extra octet is following. The number of extra octets (O) needed for the encoding of $I > 2^N - 1$ can be calculated

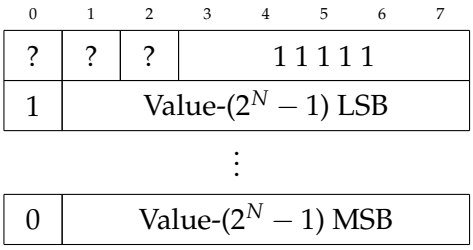


FIGURE 2.6: HPACK Integer value encoding after a 5-bit prefix [Peon and Ruellan, 2015]

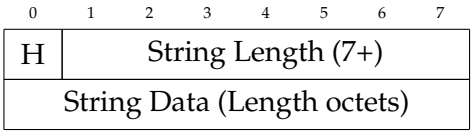


FIGURE 2.7: HPACK String Literal representation [Peon and Ruellan, 2015]

using the formula

$$O = \lceil (\underbrace{\lfloor \log_2(I - (2^N - 1)) \rfloor + 1}_{(1)}) / \underbrace{7}_{(2)} \rceil$$

where (1) represents the number of bits needed for the binary representation of $I - 2^N - 1$ and (2) the number of bits used for data in an octet.

Figure 2.7 depicts the bit-level representation of a string literal value. The first bit is used to state if the strings are encoded in ASCII or using the static Huffman encoding. Then the string length is encoded as an integer with a 7-bit prefix. At last, the string data is encoded in octets. If ASCII encoding is used, each character is encoded using one byte. When Huffman codes are used, this can be less than one byte. To make sure string literals are represented using complete octets, padding is added at the and of the encoding.

A previous attack on HPACK compression called PETAL [Tan and Nahata, 2013] stated that the use of the fixed Huffman coding table does not cause enough information leakage to be practical exploitable by a man-in-the-middle. However, when a sequence of multiple requests can be observed this can provide the eavesdropper of useful information as will be described in the next chapter.

The size of string literals depends on the number of characters. When ASCII representation is used, the string data part grows one byte for each character added. With the static Huffman code used by HPACK, the addition of a hexadecimal character does not have to cause a growth in size. For example, the encoding of *ii* is *00110 00110* which is 10 bits long. The last 6 bits will be padded, so the total encoding results in a length of 2 octets. But the encoding of *iii* is 15 bits long, this will be padded with 1 bit resulting in an encoding with a total length of also 2 octets. As the smallest static Huffman code has a length of 5 bits, two characters will be encoded in at least 10 bits which does not fit inside 1 octet. So, the addition of two characters will always cause an increase of at least one byte.

Besides, the string length encoding can cause an extra byte grow when its integer representation needs an extra octet. Calculated according to the formula above, table

String Length	Extra octets needed
0-126	0
127-254	1
255-16510	2
16511-2097278	3

TABLE 2.1: Extra octets needed for string length encoding

2.1 gives an overview of the additional octets needed for string lengths smaller than 2097279 characters.

2.4 Domain names and IP addresses

Just as in the real world, every location available in the world wide web has its own unique address, namely an IP address. These IP addresses are used to route internet traffic from their source to their correct destination, and form a crucial part of the world wide web. IP addresses are build in such a way they describe where to find the particular server on the internet. But, these lists of numbers are hard to memorise for end-users as they do not care about the location of the resource they try to visit, for them only the content matters. This is why domain names exist, they provide an alternative address which can be remembered easily by users and functions as the identifier of their favourite website. Using domain names also provides benefits for website owners. These domain names do not include any information about their location inside the internet, so websites can keep their domain name even if they move to another server.

2.4.1 DNS: Domain Name Service

Domain names, however, did not replace IP addresses, they just provide an alias for the same address. And this is where DNS comes into play. The DNS protocol is used to translate domain names into IP addresses. It can be seen as a giant telephone book for the internet. Several companies provide this information by DNS resolvers. These resolvers can be asked about the IP address of a website by just providing the domain name. And interestingly for an eavesdropper, this all happens over the internet.

When a client connects to the internet, he needs to provide the IP address of a DNS server. Mostly, this is done automatically by the DHCP protocol. The moment a client tries to visit a website, the DNS client software will send a query to a DNS resolver containing the domain name. This DNS resolver responds with an appropriate answer containing the IP address of the requested website. DNS has always been an essential component of the internet and already exists since its launch. As multiple protocols invented back in the early days, security and privacy were ignored. All DNS-messages between the client and the server are sent without encryption and are readable for everybody who can intercept them [Bortzmeyer, 2015; Kurose and Ross, 2017].

HTTPS may encrypt the actual communication between the client and the server, the DNS traffic still leaks information to an eavesdropper. Figure 2.8 shows an example of the query and answer sections from a DNS response for `uhasselt.be`. As can be

```
▼ Queries
  ▼ uhasselt.be: type AAAA, class IN
      Name: uhasselt.be
      [Name Length: 11]
      [Label Count: 2]
      Type: AAAA (IPv6 Address) (28)
      Class: IN (0x0001)
```

```
▼ Answers
  ▼ uhasselt.be: type AAAA, class IN, addr 2001:6a8:2100:500::3
      Name: uhasselt.be
      Type: AAAA (IPv6 Address) (28)
      Class: IN (0x0001)
      Time to live: 3600 (1 hour)
      Data length: 16
      AAAA Address: 2001:6a8:2100:500::3
```

FIGURE 2.8: An example of the DNS query and answer sections `uhasselt.be` as can be seen by an eavesdropper.

seen, an eavesdropper can easily deduce which websites his victim is visiting. This property has already been (mis)used for multiple different purposes. For example, the Great Firewall of China appears to block websites by inspecting and tampering DNS traffic [Zittrain and Edelman, 2003] or in the United Kingdom ISPs are required to enable parental controls and implement this using DNS filtering [Fidler, 2019].

These privacy and security issues made researchers propose many different alternatives to DNS. Eventually two new protocols, namely DNS-over-TLS (DoT) and DNS-over-HTTPS (DoH), started to be adopted to prevent this kind of information leakage. With DoT, a DNS client needs to setup a secure TLS connection with a DNS resolver first. After this, the DNS queries sent using this connection are encrypted and unreadable for an eavesdropper [Hu et al., 2016]. While with DoH, DNS messages are embedded into HTTPS messages. DoH can take advantage of upgrades of the HTTP protocol and so it can easily evolve over time [Hoffman and McManus, 2018]. For example, DoH can also make use of HTTP/3 and improve its performance and security that way. In contrast to DoT, DoH traffic uses the same transport layer port as normal HTTPS traffic, making it more difficult to do DNS traffic analysis on a network. Both Google Chrome and Mozilla Firefox offer support for DoH, and are starting to roll it out by default during 2020 [The Chromium Project, 2019; Mozilla Support, 2019; Deckelmann, 2020].

With the introduction of DoT and DoH, eavesdroppers cannot spy on users their DNS traffic anymore. Firefox provides users with a list of well known providers of DoH from which they can choose one. This approach is differently from the fact that normally network administrators and ISPs could push the use of a particular resolver when a user connects to their network using DHCP. They often pushed their own resolver as they used it to implement governmental restrictions or company policies to block websites. To prevent users of having connectivity issues or companies from banning Firefox, Mozilla also introduced the principle of the canary domain when they started enabling DoH by default. Before Firefox attempts to use DoH on a specific network, it will first try to resolve the canary domain, `use-application-dns.net`, using the resolver provided by the operation system of the device. The resolve will usually be the resolver provided by the network administrator. If the resolver returns a positive answer, meaning he could resolve the domain, Firefox will start using DoH. In any other case, it will fallback to using the DNS resolver provided by the operating system. A man-in-the-middle who wants to observe DNS traffic can actually tamper a valid response for the canary domain and cause Firefox to use normal DNS. Note that this is a temporary measure until the Internet standards body creates a standard way of signalling the presence of DNS-based content filtering[4]. A more advanced user can always override this setting to

[4]The responsible working group: `https://datatracker.ietf.org/wg/abcd/`

ignore this canary domain check and force DoH. In that case, Firefox will never try to use plain text DNS [Savage and Lazar, 2019].

It is clear that both DoT and DoH provide an initial protection against eavesdroppers. But other privacy concerns are raising as for example the limited number of resolvers which enable these services. These providers would gain more insight into the behaviour of their users as they often already provide other cloud services [Borgolte et al., 2019]. These consequences are still subject to further research and DNS is not the only protocol leaking domain names to eavesdroppers.

2.4.2 Name-based virtual hosting

After receiving an answer from a DNS resolver, the web browser knows to which IP address he needs to sent his HTTP GET request. But just sending an HTTP GET request will often not work as web servers can be configured to host multiple websites. These websites are called virtual hosts. The most common way of doing this is by using name-based virtual hosting. When a user tries to connect directly to the IP address without mentioning the domain name, the server does not know which website it has to serve. This is why the `Host` header was introduced for HTTP requests. In this header, the client needs to insert the domain name of the website his request is intended for. Hosting multiple websites on the same server is common practice on the internet as this optimises the use of limited resources. For example, websites hosted at shared hosting companies or served via CDN providers share IP addresses [Cloudflare Support, 2019]. In HTTP/1.1 this `Host` header field is required and in HTTP/2 this is replaced by the `:authority` pseudo-header [Fielding and Reschke, 2014; Belshe et al., 2015]. These headers are not interesting from the perspective of an eavesdropper as they are a part of the HTTP message which is embedded inside an encrypted TLS stream.

2.4.3 SNI: Server Name Indication

There however is an additional issue for the web server providing virtual hosts. When a user visits an HTTPS-encrypted website, the browser first needs to set up a secure TLS connection before the actual HTTP request can be sent. This implies that during the TLS handshake, a web server hosting multiple websites does not yet know which website the user wants to connect to. The headers, including `Host` or `:authority`, are not yet available during the TLS handshake phase. This raises an issue, different websites often use different certificates to secure their TLS connections. However, the web browser excepts to receive only one certificate which he will validate during the handshake phase. When a server returns a wrong certificate (with a mismatching common name), the browser throws an error stating that the connection is not private and the connection is aborted.

To solve this problem, the TLS Server Name Indication (SNI) extension was introduced in RFC 3546 [Blake-Wilson et al., 2003] and is supported by all major browsers [*Server Name Indication*]. An SNI enabled browser adds this extension field in the *Client Hello* message of the TLS handshake. This enables the server to detect which website the client wants to visit so he can return the right certificate. As explained in section 2.2, this *Client Hello* message is sent unencrypted by the client as he does not yet share an encryption key with the server at this point. An eavesdropper can intercept the packet and determine the website a user is visiting as for example shown in figure 2.9.

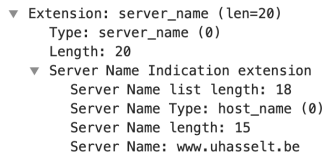


FIGURE 2.9: An example of the SNI extension in a Client Hello message as can be seen by an eavesdropper when visiting `uhasselt.be`.

This triggered the security community. Encrypted DNS hides the domain name for an eavesdropper and since TLS v1.3 the certificate, also containing the domain name, is sent in the encrypted part of the handshake. But, the SNI extension would still leak the domain name to an eavesdropper. This is why a new extension to TLS v1.3, called encrypted SNI (ESNI), is under development and planned to be submitted for standardisation in 2021 [Rescorla et al., 2020]. As its name says, this extension provides a client and server a way to communicate the domain name in the *Client Hello* message without leaking it to a possible eavesdropper by simply encrypting it.

To be able to encrypt the server name indication, the web server and the client need to share an encryption key. For the ESNI extension, a Diffie-Hellmann key exchange mechanism is used. The public key share from the server is distributed to clients via a special DNS record for the domain. After generating the shared key, the client can encrypt the server name and add it into the *Client Hello* message together with its public key share, this provides the server with enough information to decrypt the server name [Rescorla et al., 2020].

Support for this extension is expected once ESNI becomes accepted as a standard. Firefox already started to support the ESNI extension in 2018 with one of the first draft versions [Savage and Lazar, 2018]. Google Chrome, in contrary, awaits until the official standardisation of the extension [*Issue 908132: FR: Support for Encrypted SNI (ESNI)*]. Not only browser support is crucial to get the extension deployed, also website owners need to update their server software and enable the ESNI extension before a client can effectively hide the website he is visiting.

As not only the SNI extension leaks information to an eavesdropper, the development of this extension seams to shift in the direction of completely encrypted *Client Hello* messages, but this is not yet clear at time of writing of this thesis [Rescorla et al., 2020].

2.5 Webpage fingerprinting (WPF)

The previous sections show that an eavesdropper can identify the domain name of the website a victim has visited, even if he was using TLS encryption to protect his traffic. However, this does not imply the eavesdropper can identify which specific page on the website the victim was visiting. And this is were webpage fingerprinting (WPF) comes into play.

The goal of TLS traffic encryption, as stated in the RFC, is to provide a secure channel between to communicating peers [Rescorla, 2018]. A client using TLS can trust that he is communicating with the real website he wants to connect to and that the traffic is unreadable and unadaptable by a potential man-in-the-middle. It however does not protect against traffic analysis attacks as, for example, webpage fingerprinting

attacks. These already exist since the introduction of the first versions of TLS (called SSL back then) and aim to identify the specific web page a victim is visiting.

Webpage finterprinting (WPF) attacks start by generating a target set of webpages they want to be able to identify in encrypted traffic. For all webpages selected, the attacker needs to generate a distinctive signature. This signature consists of properties a potential eavesdropper has access to, for example the amount of network packets travelling to the victim when a new page is loaded. These fingerprints are used at the moment the attacker captures some encrypted traffic. He then extracts the necessary features and calculates the similarity score of this traffic with the fingerprints stored in his database. As internet webpages change constantly, the attacker needs to maintain his database by frequently updating the signatures of the target pages [Cheng and Avnur, 1998].

Qixiang Sun et al., 2002 were the first to apply these techniques in practice. The number of objects requested when the web page is loaded and the respective sizes of each of these objects were used as a signature for the different web pages. To evaluate the effectiveness of this attack, they generated signatures for just over 2000 web pages which served as the target set. They then tested their approach by measuring the Jaccard similarity between this target set and the samples of 100 000 web pages. They eventually reached an identification rate of about 75% with a false positive rate of only 1.5%, proving the possible effectiveness of WPF techniques.

Chapter 3

ESQABE: Encrypted Search Query Ascertainment By Eavesdropping

As shown in chapter 2, not all traffic is encrypted and encryption does not mean everything is hidden. In this chapter, we describe ESQABE, a collection of existing and novel techniques which make it possible for an eavesdropper to determine a user's search query by using this information. ESQABE tries to show that combining multiple techniques can improve attack performance, and that the impact of an attack can be bigger than expected. Possible defences against ESQABE will be discussed in chapter 5 after a thorough evaluation in chapter 4.

3.1 Attack description

The goal of the adversary in this attack is to determine a search query which the user entered into a search engine from a network trace. The approach followed by ESQABE exploits not only characteristics of network communications and protocols but also human behaviour when they are searching for something.

Some assumptions are made about the victim and the attacker which are needed to make the attack work. This listing serves as an overview, and the assumptions will be discussed further in the course of this chapter.

- It is assumed that the attacker already possesses a network traffic trace of the communication between the end user and the world wide web. We assume this trace not only contains the actual communication but also TLS handshakes and DNS traffic. These traces can be gathered by an eavesdropper as described in section 2.1.
- Communication took place using HTTP/1.1 or HTTP/2. HTTP/3 and QUIC are not supported.
- All websites visited by the victim need to be encrypted using HTTPS. As mentioned in chapter 2, more than 80% of the websites worldwide provide HTTPS and search engines will rank pages using HTTPS higher than pages served over plain HTTP [Bahajji and Illyes, 2014]. Pages served over HTTP would provide even more information to the eavesdropper, but as they are very rare in search results this extra information is ignored and ESQABE focuses on the more common HTTPS traffic.

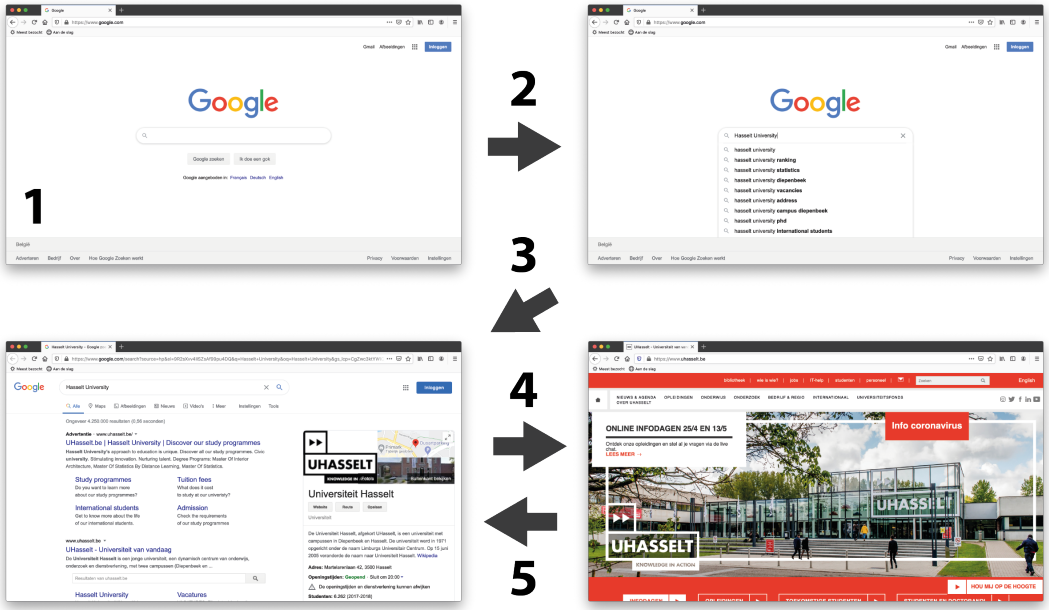


FIGURE 3.1: The scenario followed by the example victim

- It is assumed that for communication only TLS encryption is used. So IP, TCP and TLS headers are not encrypted as would be the case when using an SSH or VPN tunnel.
- The victim typed his complete search query without typographical mistakes. Mistakes where a single character is swapped can occur without decreasing the effectiveness of the attack. According to Cucerzan and Brill, 2004, only 10-15% of search queries entered by users contain errors. The detection of potential typographical errors is left for future work.
- ESQABE, as described in this chapter, is optimised for the extraction of search queries made using Google. But, the techniques described can be used on other search engines as long as they have an autocomplete functionality. This assumption is discussed in depth in section 3.7.

To support the explanation of the multiple techniques used in the attack, an example victim will be introduced. In this example, the next scenario is followed:

1. The victim opens `google.com` in his web browser
2. The victim types the search query: *Hasselt University*
3. The victim presses enter and waits for the results to load
4. The victim opens the result which points to the university website `uhasselt.be`. He observes the homepage, but does not click on anything.
5. The victim presses on the back-button of the web browser and returns to the results page.

Screenshots of these steps are shown in figure 3.1.

3.2 Extracting search query timings and length

The first step of ESQABE is to distinguish traffic related to search queries from an unfiltered network trace. As search engines encrypt their traffic with HTTPS, extraction methods need to be based on the information available in the non-encrypted headers. Once separated, this traffic will be analysed to determine the length and timing information about the search query. This information will be important in the next steps as it can largely reduce the number of possible search queries the victim could have typed. To extract this information, the approach followed by ESQABE builds further on an existing attack called KREEP.

3.2.1 KREEP: Remote keylogging attack on search engine autocomplete

KREEP [Monaco, 2019] is a man-in-the-middle attack that exploits the way search engines provide suggestions to autocomplete a user his search query. While the user is typing, these suggestions appear on the screen as shown in figure 3.2. In this example, the user already typed *Hasselt Uni* and Google is making a suggestion to complete the query as *Hasselt University*. Every time the user types a new character, a new list of suggestions appears. To achieve this, the interface of the search engine sends a request for new suggestions to the server. The currently typed part of the query is included as a GET-parameter in the request and the server responds with the updated list of suggestions. The goal of KREEP is to determine the complete search query by only investigating the traffic generated by this autocomplete functionality. As already mentioned, due to encryption KREEP has no access to the contents of the communication, it only uses metadata to determine the query.

In this paper, some assumptions about the behaviour of users are made. The user needs to type in the complete query and can only use the alphanumeric keys, underscores, dashes or the space bar. Manually moving the cursor will also cause incorrect suggestions. The words in the query need to be in the dictionary used and KREEP was only evaluated using QWERTY keyboards. As the timing of key presses is extremely important for the final word determination, the scope of KREEP is limited to search engines which generate new requests on a `keydown` event.

The attack described in [Monaco, 2019] consists of a pipeline of four steps. The source code, written in Python, is made available by the author and an heavily adapted version is used in ESQABE.

The first step is detection and tokenization. The autocompletion traffic needs to be extracted from a network traffic trace which also contains other website visits and background traffic. To extract this traffic, behavioural patterns of the autocomplete functionality are exploited by KREEP. The most important behaviour is that every time the user types a new character into the input field, a new list of suggestions is generated. As these suggestions are generated at the server, the browser needs to make a network request for every new list of suggestions. The bottom of figure 3.2 shows these requests for a user who typed *Hasselt Uni*. These request are all of the same format. The only change between them is the value of the GET-parameter `q` which contains the string typed into the search box until then. So every time the user types a new character, the query in the request grows exactly one byte. When the user types a space, which is represented by "%20" in the GET-parameter, then three bytes are added. As the other HTTP headers do not change in between requests, an eavesdropper can even observe this incremental pattern in the sizes of the encrypted TCP payload.

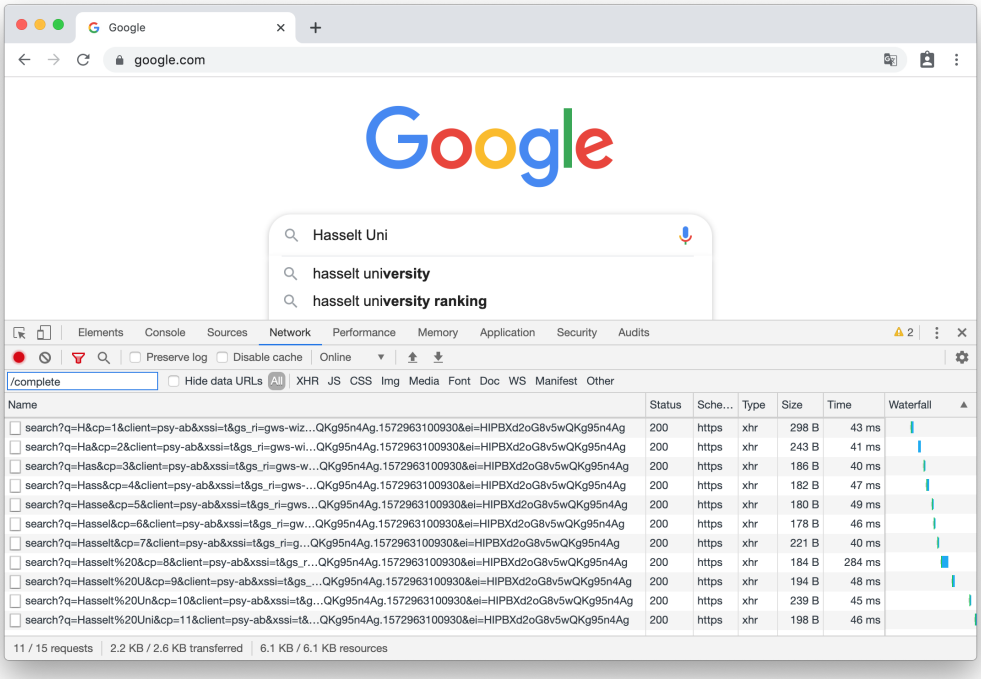


FIGURE 3.2: A screenshot of Google suggesting queries with at the bottom the requests made to the server

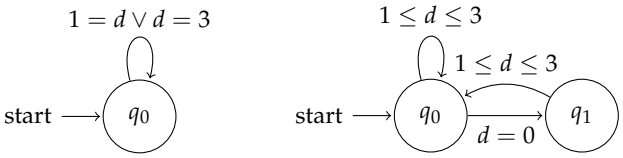


FIGURE 3.3: Two DFAs that accept sequences resembling autocomplete traffic with d = difference packet size. Left: DFA for HTTP without HPACK compression. Right: DFA for HTTP/2 with HPACK compression

This pattern of sizes of the autocompletion requests can be used to separate it from the other background traffic. To achieve this for KREEP, a sequence detector DFA is used. For all network packets captured, the size of the TCP payload is passed to the DFA. The longest subset of packets matching the DFA is probably the set of autocomplete requests. As one new character or space triggers the refresh of the suggestions, only sets of packets incrementing with one or three bytes in size are expected. This DFA is shown at the left side of figure 3.3. However, this is not the case when the search engine uses HTTP/2 with HPACK compression and static Huffman codes, as for example Google does. As mentioned in section 2.3.2, the use of this compression method can make that an extra character in the request does not cause an extra byte in the compressed packet. So, packets with increments of zero or two bytes in size are also accepted. Three consecutive packets where the size does not increment are not accepted, as this impossible with the static Huffman codes used by HPACK. The HTTP/2 variant of the DFA is shown at the right side of figure 3.3.

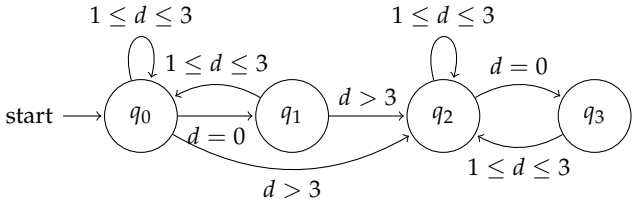


FIGURE 3.4: DFA that accepts sequences resembling Google autocomplete traffic with d = difference in packet size

After the autocomplete packets are extracted from the network trace, KREEP determines the length of the different words in the query. Spaces cause a 2- to 3-byte increase in packet size, while all other characters cause 0- to 1-byte increases. This way a new autocomplete request containing a space character can be distinguished from a request containing a normal character.

At this point, KREEP has a list of word sizes. This list is used to generate a filtered version of their dictionary, based on the word sizes. Finally, they try to identify the typed words by using keystroke timings and predefined language models. To achieve this, KREEP uses a three-layer neural network to predict the key probabilities.

The approach described above, is the approach in general and in the ideal case. But, each different search engine has its own special characteristics. Google adds a `gs_mss` parameter to the request containing the partially completed query at some point. The function of this parameter is unknown as Google has made no documentation available about the parameter. In the original paper, it is suggested that this parameter occurs after about 12 characters typed. Our tests showed that the moment of appearance is highly variable, ranging from after 5 characters typed to 40 characters typed. The original implementation of KREEP accepts three bigger jumps[1] than 3 bytes, in an increase of package size. The complete DFA for Google, as described by KREEP, is shown in figure 3.4.

3.2.2 Integration in ESQABE

The keystroke detection and tokenisation steps described by KREEP form the first step of ESQABE. While integrating these steps, several issues were discovered. Multiple adaptations and refinements to the implementation provided by Monaco, 2019 are made. The eventual implementation is faster, reaches higher accuracy and fits better for our purposes.

Optimisation for large traces The original implementation is not created for extraction of search queries out of realistic (large) traces. For example, the user may use a cloud file hosting service like Dropbox which is synchronizing with the server at the same time. All packets are first grouped by their destination IP address and protocol and then tested against the DFA for determining autocompletion traffic. Trying to match traffic with a DFA is a computationally intensive task as these lists of packet groups consist of a larger multitude of packets. We speed this up by adding a filter before sending the traffic to the DFA. Our implementation groups packets by TCP stream and filters out all packets that do not have Google as destination.

[1] The DFA described in the paper only accepts one jump, it's not clear why the implementation accepts three jumps.

The fact that Google has given each of its servers a hostname containing *1e100.net* [*What is 1e100.net?*] is exploited to achieve this. For each external IP address, the corresponding hostname is retrieved via a reverse DNS lookup. All packets with IP addresses that do have a hostname which does not contain *1e100.net* are filtered out. This improvement also causes less false positives as less packets can coincidentally match the DFA. Other search engines do not have to implement this behaviour, in that case the traffic can be filtered by using the domain name associated with the destination IP address of the packet. The association of the domain name with the IP address is discussed in section 3.3.

Take timings into account Another issue which occurred with these lagers traces is that, due to the large search space, random packets with a packet size which could follow on the auto completion traffic are matched as well. This caused KREEP to return wrong word lengths. To reduce the probability that this happens, the DFA was extended with the constraint that there may be a maximum of three seconds in between two packets.

From TCP packets to TLS records The original KREEP implementation is based on the size of the payload of the TCP packets. But with the use of TLS, multiple TLS records can be multiplexed in a single TCP packet. This multiplexing behaviour confuses KREEP as these multiplexed TCP packets are overlooked by the DFA. As it is only the payload of the TLS records that is encrypted, the distinction between the records is visible for an eavesdropper. ESQABE uses the adapted implementation where the key logger is based on the sizes of TLS records.

Important to notice is that the multiplexing of other TCP traffic with the auto complete traffic does not happen that often. The requests for new suggestions are only sent when a new character is typed. When the user starts typing, the complete search page is already loaded so there are almost no requests to multiplex with. However, there is sporadic user tracking traffic which can be multiplexed with the auto complete traffic.

Not only can these requests be multiplexed into a single TCP packet, they can also be multiplexed on HTTP/2 level. This could raise a problem as the traffic at HTTP/2 level is completely encrypted by TLS and not visible to an eavesdropper. This behaviour is browser dependant and is described in 3.2.3.

`gs_mss` **matching improvement** As already mentioned in the previous section, the behaviour of this parameter described by KREEP did not match the actual behaviour. This caused wrong patterns to match the DFA, resulting in a wrong query length. Investigation of the `gs_mss` parameter showed a more precise matching is possible as the size is predictable. The moment of appearance of this parameter is still unpredictable and undocumented by Google, but it appears to be linked to the contents of the query, for example when gibberish is entered the parameter appears a lot sooner. At this point ESQABE does not know what a user is searching for so this characteristic cannot be exploited. The contents and size of the parameter, on the other hand, are predictable. The parameter contains the part of the query typed until that point. It is best shown in an example:

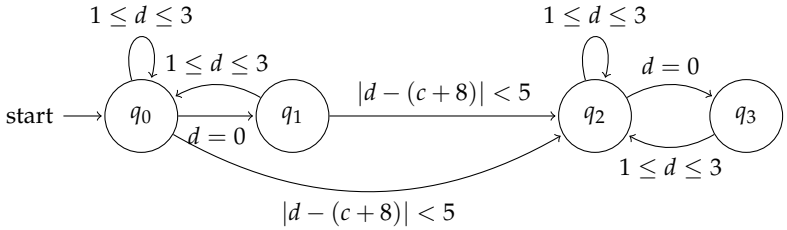


FIGURE 3.5: DFA that accepts sequences resembling Google autocomplete traffic with d = difference packet size with last packet and c is difference in packet size with first packet

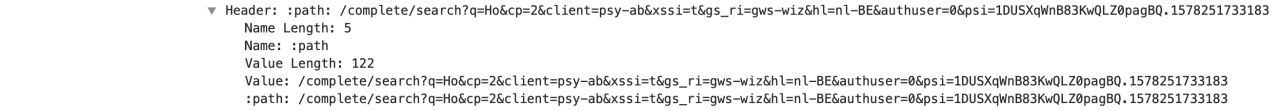


FIGURE 3.6: A screenshot of Wireshark showing the HTTP/2 `path` header for an autocomplete request

1. A user already typed *misdes* which resulted in a request to `/complete/search?q=misdes&cp=6`[2]
2. The user now types an *c*, this results in a request to `/complete/search?q=misdesc&cp=7&gs_mss=misdes`.
3. In all subsequent autocomplete requests, the `gs_mss` parameter stays present with the same value.

The new matching behaviour is shown in figure 3.5. The edges with $d > 3$ are replaced by $|d - (c + 8)| < 5$, with c is the difference in packet size between the current and the first packet. The size of `&gs_mss=` is represented by 8 bytes and the total difference needs to be smaller than 5 as it includes an extra character, which can be a space, and extra increases caused by HPACK.

HPACK and tokenization difficulties The tokenization method followed by KREEP has another issue. When using HPACK to encode strings, rollover issues can occur. As explained in section 2.3.2, every string encoding starts with an integer representing the string length. As shown in figure 3.6, the URL of the autocomplete request contains more than only a part of the query. In the example, the complete `path` parameter exists out of 122 characters. So an extra byte will be added when 5 more characters are typed because the string length integer needs an extra byte for its encoding. This can result in a 2-byte increase in packet size when only one non-space character is added and a 4-byte increase when a space was added. As the other parameters in the URL are unpredictable, it cannot be determined if this was a space or a normal character in a word. Having two such byte roll-overs is very rare as the query would have to be at least 128 characters. This can be deduced from table 2.1 in section 2.3.2 which gives an overview of the string length encoding size and the fact that search queries have an average length of 3 terms [Taghavi et al., 2012]. The fact that a space can cause a 4-byte increase needs to be taken into account.

[2]Only relevant parts of the URLs in this example are shown. All the other parts do not change.

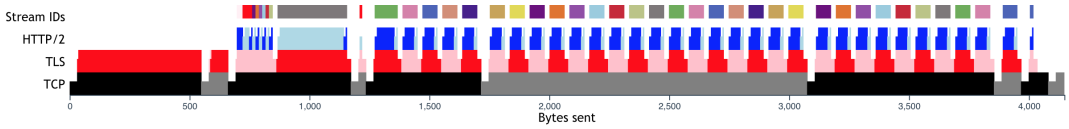


FIGURE 3.7: Packetization view for 30 HTTP/2 request in Firefox

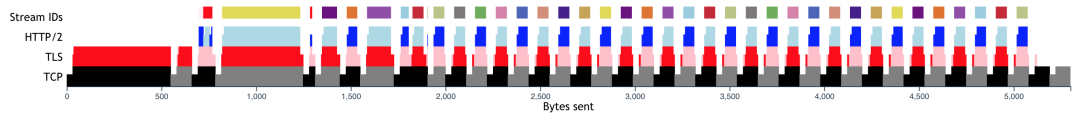


FIGURE 3.8: Packetization view for 30 HTTP/2 request in Google Chrome

3.2.3 Experiment: Can HTTP/2 multiplexing be/become an issue?

A single TCP packet can contain multiple TLS records, an eavesdropper can see the size of these individual records. But, if HTTP/2 streams are combined into a single TLS record, an eavesdropper cannot known this. As KREEP heavily supports on the fact that only the query itself has influence to the packet lengths, multiplexing multiple HTTP/2 streams into a single TLS record will cause detection problems. It is important to know how browsers behave and if their behaviour can be or become a problem for KREEP.

Research by Morla, 2017 showed that only 5% of the TLS records they retrieved from HTTP/2 servers contained more than one HTTP/2 frame. However, for requests, it is not a server but the browser which decides when to multiplex multiple HTTP/2 frames into a single TLS record. To experiment with this behaviour, a small webpage was created which fetches 30 other very small resources from the same server at the same time, this in order to try how a browser behaves.

The tests executed with Google Chrome and Firefox showed different behaviours. To analyse the traces in a quick way, the network packet traces were imported into the *packetization* view of *qvis*[3]. Figures 3.7 and 3.8 show an example visualisation for respectively Firefox and Chrome. The top line shows different HTTP/2 streams, the requests in this case. The second line shows the HTTP/2 frames, notice that Firefox sends a window update in every request. The third line shows the different TLS records and the last line the different TCP segments. Firefox appears to pack every HTTP/2 request in a separate TLS record, while Chrome in this case even did sent each HTTP/2 request in a separate TCP segment.

So, currently, the two major browser implementations appear to not multiplex multiple HTTP/2 requests into a single TLS stream. This makes it possible for an eavesdropper to deduce the separate lengths of the queries. If, browsers would change this behaviour in the future, this does not necessarily need to become a problem. As already explained, the autocomplete requests are sent at the moment the user types a new character and at these moments there is few traffic to multiplex with.

3.3 Domain names

After a user entered his search query, a list of results is shown. The user will open a result and visit a website which differs from the search engine. The domain names

[3] `https://qvis.edm.uhasselt.be`

of these websites provide a valuable source of information about what the victim is searching for. For example, when a victim visits *apple.com*, he is probably searching for something related to technology. In this example, the domain name provides category information about the search term. But more importantly, domain names make it also possible for the eavesdropper to visit the website the victim is browsing. ESQABE can extract the content of the homepage and compare it with possible search queries.

As discussed in section 2.4, domain names are sent multiple times by the client to multiple different services. Originally this was sent all unencrypted and in the clear, but new mechanisms try to protect internet users from this kind of information leakage by adding encryption. Some of these techniques start to be adopted, others are still under development in 2020. It however is still unclear if the domain names will be hidden in the future as not only attackers find this information valuable, but also internet providers and network engineers who use them to provide several services as for example parental control. Solutions to enable this services to keep on working need to be found and provide potential entry points for attackers as with the canary domain discussed earlier.

The implementation of ESQABE will mainly abuse the availability of the Server Name Indication (SNI) extension which is not yet sent encrypted and is visible for an eavesdropper. This makes it possible to label every single TCP connection in the network capture trace with the corresponding domain name, even if multiple websites on the same host are accessed. With the proposition of Encrypted SNI, this can become impossible in the future, this in combination with the encryption of DNS traffic seems to make an attack as ESQABE impossible. But, that is not true, as multiple alternative approaches to determine the domain name are already examined by other researchers.

3.3.1 Fingerprinting DNS-over-HTTPS requests

A first approach is to deduce the domain name using the DNS-over-HTTPS (DoH) and DNS-over-TLS (DoT) traffic. Siby et al., 2020 found out these new protocols appear to be vulnerable to a fingerprinting attack. A user visiting a website will not only need to request the IP address of this specific domain name but also the addresses of the servers hosting external resources used by the website. These include for example an external library, images hosted on CDNs, tracking scripts and others. The packet sizes of this complete set of DoH traffic appear to have a high intra-class and a low inter-class variance which makes them highly suitable for a fingerprinting attack. DoH traffic cannot be extracted directly as it runs on the same port as HTTPS traffic and so can be called disguised. But as the amount of DNS servers offering DoH is limited, an eavesdropper can start with filtering on IP addresses from known resolvers. TLS streams used for DoH also appear to show other behaviour than other TLS streams. These DoH streams are very long lived to avoid the overhead of the TLS handshake for each new request and the requests and responses are typically shorter than with normal HTTPS trafic. However, when HTTPS traffic is mixed with DNS traffic in the same TLS stream the researchers did not succeed in their goal to extract the DoH traffic. To deploy this attack, the eavesdropper needs to generate fingerprints for all the domains he is interested in and needs to keep them updated over time as websites often make changes. When using this fingerprinting method, the eavesdropper can only detect the domain names he knows about and

for which he generated samples. This is a large difference with the straight forward extraction of domain names out of the original DNS traffic.

3.3.2 OpenKnock: Identifying HTTPS Websites in an open world scenario

The previous approach required the eavesdropper to generate the fingerprints of all possible websites on beforehand. This approach would limit the effectiveness of ESQABE. Di Martino et al., 2020 proposed another approach for circumventing encrypted DNS and SNI called OpenKnock. This approach is based on domain name extraction from reverse DNS and the Subject Alternative Name field present in HTTPS certificates. This technique does not require the generation of fingerprints in advance, which enables the attack to work in an open world scenario. Tested on a list of 3940 top websites, this attack was able to correctly predict 66.1% of the websites and only predicted 9.9% of the websites falsely.

3.4 Detecting the visited websites

The list of visited domain names created in the previous step consists of all domain names which are found or deduced from the complete network trace which was created by the eavesdropper. This implies that this list does not only contain domain names the victim intentionally visited but also domain names which are a consequence of the visit and even domain names visited by the operating system in the background.

For example, the victim visits `google.com`, searches for *Hasselt University* and clicks on the search result containing the official website `uhasselt.be`. When the eavesdropper feeds this trace to ESQABE, it is able to extract 13 different domain names. It does include `google.com` and `uhasselt.be`, but it also includes for example `cdn.jsdelivr.net`, `ajax.googleapis.com`, `fast.fonts.net` and many more. The amount of domain names can differ between circumstances, but in the majority of the situations it will be more than two.

The victim, however, did never specifically visit the websites behind these extra domain names, he only visited `google.com` and `uhasselt.be`. But as his target websites reference specific files from these servers to enhance their website with other features, the domain names of these feature providers are also present in the network trace. These domains do not contain any references to *Hasselt University* and are not in the interest of ESQABE. These domains could even mislead ESQABE, as they often provide information related to the products they deliver. This could make ESQABE erroneous think that the victim searched for something related to their products. To obtain better results, ESQABE needs to have a way to filter these uninteresting domain names and only maintain a list of domain names which the victim effectively visited.

3.4.1 Information available for the eavesdropper

A first approach is to filter domain names known to only host assets for other websites to include, for example `googleapis.com` (which hosts JavaScript libraries [*Google APIs*]) or `adobess.com` (which is used by Adobe Creative Cloud for authentication and authorisation purposes [*Adobe Creative Cloud Network Endpoints*]). Using such a

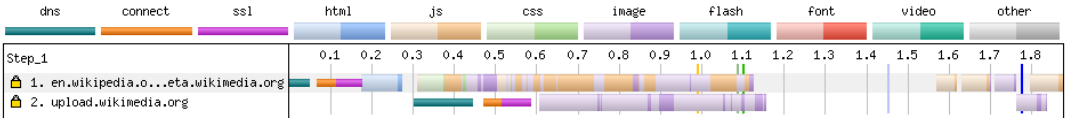


FIGURE 3.9: A screenshot of Web Page Test (connection view) showing the different connections setup when visiting `https://en.wikipedia.org/wiki/Hasselt_University`. The abbreviated value behind 1. contains: en.wikipedia.org,meta.wikimedia.org

blacklist can be a good first step, but it will always be incomplete and will rapidly become outdated.

A second approach is to use the characteristics of network traffic to identify the initial website which was visited. At the moment the user clicks to open a website, the web browser will lookup the IP address of the server and open a new TCP connection. They establish a TLS tunnel and the browser sends his HTTP GET request to which the server starts answering with an HTTP response. It is only from the moment the response starts arriving at the client that he can find linked resources required by the website which are hosted on other servers for which new TCP connections need to be set up [Kurose and Ross, 2017]. For example, in figure 3.9, a connection overview is shown for a user visiting the Hasselt University Wikipedia-page. A connection to `upload.wikimedia.org` is setup only after the initial connection was setup. In this example, HTML was fully loaded before the other connection was setup, this however does not always need to be the case. When a website uses preconnect headers for example, the browser can setup a connection to another server even before the HTML content was fully loaded [Mihajlija, 2019].

This chain of connections can be correlated with the fact that a user needs to fetch the content of the web pages over the internet, also passing the eavesdropper. The average web page size on the internet in April 2020 is 2031.9 KB for desktop pages and 1864.2 KB for mobile pages with 75% of the mobile pages being larger than 882.1 KB [*Report - Page Weigth*]. So, an eavesdropper will see an increase in bytes and packets transferred to his victim on the moment his victim loads a web page.

Take figure 3.10, this graph is made with the trace an eavesdropper could capture for the victim of the example in section 3.1. The actions of step 1, 3 and 4 cause the users web browser to load a new web page. The human eye can easily distinguish these actions from the graph as they cause a peak in the amount of packets per second. This would be even clearer if the victim was focused on searching only. But to make the example more realistic, the victim was running his mail client in the background and had file synchronisation services as Google Drive turned on. However, the traffic they generated was so rare in comparison with a page load it does not disturb the graph.

The bottom of the graph shows the new TLS connections set up by the victim during the capture of this trace. This was done by extracting all TLS *Client Hello*-messages and connecting them to the domain name they belong to. As all *Client Hello*-messages in this capture contain the Server Name Extension, they could be directly linked to the correct domain name. If a connection uses the Encrypted Server Name Extension, the IP address of the connection could be correlated with the domain name retrieved as described in section 3.3. To not clutter the graph, connections setup within two seconds of each other are grouped and arranged in the order they appeared. Some connections in the captured trace related to system traffic, these are

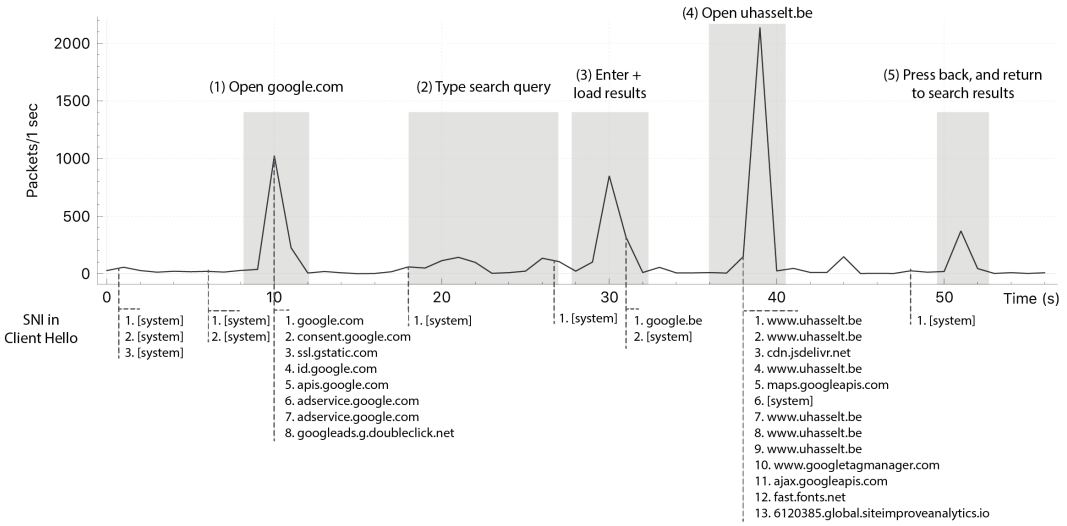


FIGURE 3.10: Visualisation of the number of TCP segments when a user searches for Hasselt University and opens the university website. Below the graph, all SNI values for TLS Client Hello messages are shown. *[system]* labels represent connections setup by other processes running in the operating system.

also visualised because an eavesdropper does not know if these are related to the system or to the visited web page. These system connections are redacted as they contained private information.

The graph is made completely of information available for the eavesdropper (except for the annotations of the steps). With the information available in the graph, the eavesdropper can determine the user visited `google.com` around the tenth second as this is the first connection setup before a peak in packets. He can also determine the users visits `uhasselt.be` after about 38 seconds using the same reasoning. Important to note here is that the system traffic is totally unpredictable and it can happen that a new system connection is setup just before the user visits a web page. Most of the system domain names are filterable by the human eye as they often mention their purpose in the domain name. For now, assume an automatic implementation can filter these out with a blacklist. The other annotated regions in the figure cannot be determined from this graph and are only shown for clarification.

This behaviour only happens if the web page is not fully cached. If the browser appears to have a version of the page available in his cache, he will use it. Depending on if the complete page is available in the cache, this website visit will possibly not appear inside the eavesdropped network trace. But as caching the complete web page prevents website owners of quickly publishing updates they often forbid browser to us a cached HTML page without first validating if it is still up-to-date at the server. In 2019, over half of them made their HTML pages non-cacheable and only 10% provided a cache TTL > 0 [Calvano, 2019]. But even if they are permitted, browsers do not retain a copy of every web page a user visited during their whole lifetime. This would use too much storage space and become inefficient, browsers use heuristics that prefer often visited pages to stay longer inside their browser cache. If a user makes a web search, it can be expected he did not just visit this page as he would probably not be searching for it. Combining these arguments, ESQABE assumes pages are not completely available in cache and a browser requests them from the server.

3.4.2 Implementation

With the insights gained, ESQABE was extended to automatically detect the web sites the victim effectively visited. In this step of the implementation it is expected the time frame wherein the victim opens websites related to his search query is delimited. The timestamp of the last autocomplete packet is taken as the front delimiter, as ESQABE is not interested in traffic which happened before the search query.

1. The first step is to extract all new connections opened in the given time frame. As websites 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. In 2020, most of these *Client Hello* messages still contain the *Server Name Indication* extension which makes it straightforward to link each connection with the visited domain. However, when this header is not present, the IP address of the server can be looked up in the dictionary resulting from the techniques in section 3.3.
3. At the beginning of the time frame, the victim just landed on the page containing the search results. From here, ESQABE iterates chronologically over all new connections. So, for each connection:
   (a) The total size of the traffic between the victim 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 then `MIN_PAGE_SIZE`, it is expected a web page was loaded immediately after the connection was setup. The domain name associated with this connection is added to the list of potential visits.
   (c) The connections setup during the load of the page are connections setup by the page itself and were not explicitly opened by the victim. All connections setup while more traffic is flowing between the client and the internet are skipped. The next connection processed is the first after the spike in network traffic.

The algorithm results in a list of websites the victim probably visited after the search results page was loaded. In the algorithm two constant threshold values are used. These need to be chosen depending on the network conditions ESQABE is deployed on.

- `MAX_LOAD_TIME`: The maximum expected loading time of a web page. In practice this needs to be the length of the spike in traffic and can probably be determined automatically.
- `MIN_PAGE_SIZE`: The minimum size of a web page. If this value is too small, system background traffic can be classified as website visit. But if this value is too big, legitimate website visits could be ignored. This size can vary over time, the transfer size of web pages and its resources is rising but as computer get faster, users tend to leave open more application which on their turn cause an increase in background traffic [*Report - Page Weigth*]. There are opportunities to automatically determine this value, but these are left for future work.

Not that this technique has its limits. 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, depending on the search engine a difference between a click on a search result and other actions can possibly be detected. Some search engines log every click a user makes, so this tracking information would occur when the user opens a link but not when he does something else. The feasibilty of this approach is left for future work.

3.5 Visiting web pages

At this point ESQABE knows the the length of the different words in the search query and knows which websites the victim visited after making the search query. The websites visited do have to contain some information about the search query, otherwise they would not have appeared in the search results. However, ESQABE does not actually know which page of the website is visited by the victim, it only knows the domain name of the website containing the web page. This does not imply that the home page cannot provide us with valuable information. Take the following situations:

- **The result links to the home page.** This is the most trivial case. When the victim opened a search result pointing to the home page of a website, the home page will contain at least a part of the search query as it otherwise would not have been appeared in the results. This happens when the user for example searches for a specific company, person, a keyword a company advertises on ...
- **The result links to a Wikipedia article.** Wikipedia articles appear very often in the top search results. According to [Lewandowski and Spree, 2011], Wikipedia appeared in the visible area of four results for 35% of their list of popular queries. For the queries of the dataset used in chapter 4, with 73% of the queries, the first page contains a result what links to a Wikipedia article. As Wikipedia contains information about all sorts of topics, scraping its homepage will not provide any information at all for ESQABE. For Wikipedia pages another approach is taken as described in section 3.6.
- **The result links to another platform or webshop** Platforms and webshops like Booking.com, Steam, AirBnB, IMDB, Netflix... these platforms all provide a gigantic number of items. On their homepages they only feature a fraction of everything they have available online. Depending on the platform a same approach as for Wikipedia can be taken.
- **In all other cases,** the home page can still provide relevant information to the topic. For example a company website can feature several products on their homepage or the homepage of a television station can list all programs they broadcast ...

Now, ESQABE tries to automatically extract relevant information from these homepages by extracting word(s) that match the pattern(s) determined by KREEP. ESQABE uses the Selenium WebDriver[4] to open the homepages of the candidate websites in a real web browser as modern websites often fetch extra content using JavaScript. To extract the word combinations the following approach is taken for each

[4] selenium.dev

website:

1. ESQABE visits the homepage of the website and waits until it is fully loaded.
2. A limited scroll movement is executed to trigger the loading of content under the fold, which due to lazy loading[5] may not have been loaded otherwise.
3. After the visit, all text is extracted from the HTML body using the `innerText` attribute. As defined in the HTML Living Standard [*HTML - Living Standard: DOM*], this attribute returns all the text inside the element and its children as rendered. This also implies that tags as for example `<p>Hasselt <strong>University</strong></p>` are returned as *Hasselt University*. The string which `innerText` returns highly resembles what would be on a users clipboard if he selected the whole page and copied it.
4. The strings of the title tag, image alt-tags, and selected page meta-tags[6] are extracted separately.
5. Once this extraction is finished, RegEx patterns generated from the word lengths of KREEP are used to extract all matching strings.
6. The resulting sets of strings are combined in a counter with as key the string and ranked by number of appearances. Note, these counters are case sensitive and for example *Hasselt University* and *HASSELT UNIVERSITY* will be counted separately.

The counters of all websites are summed up at the end to generate one big counter for each string. At this point the counters of the same strings, but with different casing are merged together. As some websites change the casing of words for styling purposes, e.g. capitalised titles, ESQABE uses for each word the most common casing in the counter. This is of importance for the step described in section 3.6.

3.6 Using Wikipedia fingerprints

As discovered during the evaluation of ESQABE, Wikipedia pages do often appear in the results of search engines. Other researchers found Wikipedia results to appear in the visible area of four results for 35% of their queries. Initially, this was not interesting for ESQABE. The homepage of Wikipedia does not provide any information relevant to a search query, as it is a collection of a multitude of Wikipedia articles. But, this does not imply Wikipedia cannot be an interesting information source for ESQABE. In contrary, if a Wikipedia page is visited by the victim it can be used to reduce the list of potential search queries to only one as will be described in this section.

In total more than 35 million Wikipedia articles exist in more than 300 different languages [Wikipedia contributors, 2020a]. All these articles try to cover a single subject with all relevant information. Each article is identified by its unique title. This title, according to the guidelines, needs to be short, natural, distinguishable and recognisable [Wikipedia contributors, 2020b].

[5] Websites which implement lazy loading do not load the full web page on the initial load, but start loading content when a user starts interacting. Mostly this includes content under the fold, which is initially invisible but becomes visible if the user makes a scroll movement.

[6] name, description, og:site_name, og:title, og:description, twitter:title, twitter:description

When a visit to a Wikipedia article is detected by the eavesdropper, ESQABE will try to determine which article the victim visited. The list of potential search queries, which resulted from the algorithm in the previous section, will serve as the entry point for this quest. For each possible search query, there will be searched for a matching Wikipedia article. These selected pages will be compared to the actual traffic retrieved by the eavesdropper and so it will be estimated which article was visited by the victim.

3.6.1 Selecting potential Wikipedia articles

As the task of making these fingerprints is time-intensive, this list of potential Wikipedia articles needs to be kept as short as possible. This is why ESQABE only tries to find Wikipedia articles with a title matching the possible search query. To implement this, ESQABE makes use of the Wikipedia Python package[7] which uses the Wikipedia API[8] to provide its services. The implementation is straight forward, but some details have to be kept in mind.

First, Wikipedia article titles use the most commonly recognisable name of the subject. But their contributors also provide redirection pages from other, less commonly known names or synonyms. For example a user who tries to open the article with title *Paul Hewson* will be redirected to the article with title as title *Bono*. As Google results always show the direct link to the real article, in this attack only the final article URL is used.

Secondly, Wikipedia article titles are case sensitive and do always start with a capital letter. A miscapitalisation can cause the API to return no responses potentially resulting into a wrong result. In the previous section a limited attempt to avoid miscapitalisation is already discussed, which avoids capitalisation issues due to website design. For common mistakes, Wikipedia provides redirection pages, which will be automatically followed and ESQABE does not need to worry about this [Wikipedia contributors, 2020c].

Third, Wikipedia consist out of more than 300 different versions for different languages. Each language is maintained separately and has its own separated API endpoint. They all run the same application but on different sub domains of `wikipedia.org`. To compare the correct language version of the page to the eavesdropped traffic, it is important to use the correct API endpoint. As a sub domain is in practice a different domain, this is already extracted as described in section 3.3 and can be used to determine the correct endpoint.

Last, some words have multiple meanings. For example, Java is an isle and a programming language. For these pages Wikipedia uses special disambiguation pages. These pages contain a list of articles with the same name, but with a different meaning. In the current version, all these pages are added to the list of candidate articles. In theory it could be possible for ESQABE to filter several pages by comparing their contents with the contents of home pages of the other websites, but this is not investigated and is left open for further research.

Searching for possible Wikipedia pages caused an unexpected advantage to be gained. This list of potential search queries also contains entries which do match the requested pattern but are completely nonsensical. For example, an execution of the

[7] `https://pypi.org/project/wikipedia/`
[8] `https://www.mediawiki.org/wiki/API:Main_page`

attack on a victim searching for *Beat Saber* also resulted in the potential search query *game where*. Or when the victim searched for the Belgian television show *Piet Piraat*, this resulted in ESQABE returning *onze andere* which is *our other* in English. These examples consist of valid words which can occur in sentences like this, but they do not make sense on their own. As a consequence, there does not exist a Wikipedia article with this title and the suggestion can be deleted.

3.6.2 Identifying the visited article

In order to identify the visited article, a small web page fingerprinting attack is executed. The implementation used in ESQABE and described next is based on a naive Bayes classifier as described by Liberatore and Levine, 2006 and an implementation of this classifier by Dyer et al., 2012.

The classifier uses supervised machine learning algorithms to achieve the identification. This means the classifier is trained on traces with known sources and functions in a closed world. The set of traces retrieved in advance is called the training set. The classifier can only identify web pages he already trained on, which makes the training set is also called the target set of the attacker. Generating and maintaining a dataset of fingerprints for all 53 million Wikipedia articles is an unrealistic task [Wikipedia contributors, 2020a]. This is why the implementation of ESQABE generates the fingerprints of the potential articles, selected in the previous section, on the fly. The potential articles are visited by ESQABE, one at the time, while a network capture is running in the background. As this capture does not communicate with the browser, it observes the same data as an eavesdropper would. The main difference is that these website visits are run on demand and so the trace can be labelled with the article title to serve as an entry in the training set. To improve the quality of this training set, the web pages are visited multiple times by both Mozilla Firefox and Google Chrome as the sizes of the requests and response change by browser and do not always be consistent due to, for example, changing headers.

Next, the traces of the training set need to be described by the attributes as used by the Liberatore classifier where a trace together with its attributes is called an *instance* i. Every instance is labelled by its *class* C_i, which is the title of the Wikipedia article. The list of attributes X_i of the instance are used to describe each packet captured by the eavesdropper. Each attribute represents the amount of packets of a certain size in a certain direction (incoming or outgoing). The size of a packet is defined as the amount of bytes of the captured link-layer payload. This set consisting of (X_i, C_i) tuples is then passed to the algorithm.

The classification method by Liberatore and Levine, 2006 is based on a naive Bayes classifier. For the web page captured by the eavesdropper, the list of attributes is also calculated and called Y. This list is then passed to the classifier which will first calculate the conditional probability $P(C_i|Y)$ by using Bayes' rule: $P(C_i|Y) = \frac{P(Y|C_i)P(C_i)}{P(Y)}$. The classifier calculates this probability for all possible classes C_i and eventually selects the class with the highest probability as the article probably visited by the victim. It is assumed that every article has the same chance to be visited, so the probability $P(C_i)$ is the same for each C_i. And the $P(Y|C_i)$ is calculated by the use of normal kernel density estimation based on the attributes of the training set.

The naive Bayes classifier itself is not re-implemented for ESQABE but uses the existing implementation, `weka.classifiers.bayes.NaiveBayes`, of the Weka toolkit

	Google	Baidu	Bing	Yahoo	Yandex	DuckDuckGo	Startpage.com
Domain	google.com	baidu.com	bing.com	yahoo.com	yandex.com	duckduckgo.com	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
Differences	gs_mms GET-parameter and counter	counter	counter	/	/	/	random string
Applicable?	Yes	Yes	Yes	Yes	Only total length	Only in specific cases	No

TABLE 3.1: Comparison of applicability to multiple search engines

from Witten et al., 2011. In principle, other classifiers can be used as well but a comparison for this specific scenario is left for future work.

3.6.3 Broadening this approach

In this section, this fingerprinting approach is only discussed for Wikipedia pages as they are most common. But, one could see the other possibilities which can be taken. For example, the same approach could potentially be taken for Instagram pages of celebrities or for product pages on webshops like Amazon. As long as there is a potential list of pages to start from, multiple other platform websites which host a multitude in topics could fit into this approach.

For example, research by Di Martino et al., 2019 showed that fingerprinting social media pages involves certain troubles. The researchers describe an attack, IUPTIS, which makes it possible to identify a social media page for a man-in-the-middle. More specifically, to be able to run IUPTIS, the eavesdropper will need to have the possibility to delay several network packets and make a live analysis of the network trace.

3.7 Extraction of non-Google search queries

ESQABE as described above is focused on the detection of search queries made with Google. However, in practice the techniques described are extendable to other search engines which use the autocomplete functionality as well. The first step using KREEP will require a limited amount of fine tuning as some autocomplete functionalities may have different behaviours, take for example the `gs_mss` parameter Google uses as described in section 3.2.2. The other steps are search engine independent and will not require any modification. This section shortly discusses the applicability to several other popular search engines. To examine the applicability for each search engine, we entered several different queries and examined them using a network protocol analyser. The results of this comparison are summarised in table 3.1. Only HTTPS variants of the search engines are studied, some search engines as for example Baidu still make the HTTP version available to their users, but in that case search query extraction is trivial.

In contrary to all other search engines, Baidu did not yet upgrade to HTTP/2 which means the advantages and disadvantages of HPACK compression are not present in these autocomplete network requests. The absence of HPACK makes it easier to detect sequences as characters always cause a single byte increases and spaces a three byte increase. On the other hand, the results cannot be pruned based on the incremental HPACK compression as described in section 4.5.3.

Search engines which encode their spaces as a + instead of a %20 in the auto-complete request make it impossible for an eavesdropper to distinguish a space character from an alphanumeric character. Consequently, the eavesdropper can determine the length of the search query but he cannot determine the length of the different words in the query as he cannot split them into parts. The effectiveness of the attack on these search engines is not yet investigated and is left for future work. In research by Taghavi et al., 2012, 15% of the queries consisted of only one word. For these queries, ESQABE would already be applicable in theory.

An investigation of the source code of DuckDuckGo showed the search engine throttles its autocomplete requests for 300 milliseconds. This implies that for users typing with a speed of more than one key per 300 milliseconds not all all requests will be send. A large study on typing behaviour observed an average inter-key interval of 238.656 milliseconds (SD = 111.6 ms) with fast typists averaging about 120 milliseconds (SD = 11 ms) [Dhakal et al., 2018]. So for an average typist, this will impact the amount of auto complete requests and the pattern visible for the eavesdropper. With HPACK compression, ESQABE needs to rely on the number of packets to determine the query length as a new character added to the query does not need to cause a byte increase.

Startpage.com appears to be not vunerable at all. Every request for new suggestions does not only include the search query it also includes a variable `ixrc` which appears to contain a random string of a random length which is changed with every request. This implies that the autocomplete requests for Startpage.com do not have an increasing packet length and so ESQABE cannot detect the length of the search query.

Chapter 4

Evaluation of ESQABE

The attack described in the previous section works theoretically, but its applicability in practice needs to be evaluated. This section describes the followed approaches, which consists out of two parts. First, the viability of ESQABE was planned to be tested with a user study. And secondly, an automatic test approach is followed to reach a higher scale.

4.1 User Study

The applicability of the previous attack in real-life situations depends on the behaviour of how end-users search the web. The first goal of this study is to retrieve real-life search behaviour which users tend to make in everyday use. This includes the queries entered by the test subjects and the websites they open from the list of results. The second goal of this study is to determine the realism of the assumptions stated in section 3.1.

4.1.1 Early approaches

The first, and most straightforward method would be to ask users if they would install a tool which automatically logs search queries and consecutive actions taken by the end-user. But, this path was not taken as users could make for example medical related search queries which involve legal complexity in storage. Users would have to give up an important part of their privacy to participate in this test, so this approach was considered as not executable.

So, simulating and triggering real life search behaviour in a simulated lab environment is the real challenge of this study. Search queries normally happen in spontaneous situations where the user is trying to figure something out mostly triggered by a certain context. The first, and most simplistic approach is to just ask users some questions where they have to find answers to. These questions are rather simple as for example: "Who is the president of Malta?". The questions need to be hard enough so the subject does not know the answer right away and needs to search. However, when just testing this in a small group of subjects, they all entered the question directly into Google and the questions were so easy for Google it could just answer right away as shown in figure 4.1.

The second attempt in creating a real life situation used questions based on images. This path was taken, as users cannot literally enter images in the search engine and need to create search queries themselves to solve the question. An example of such a question is: "What is the name of the ship pictured in figure 4.2?". For the purpose

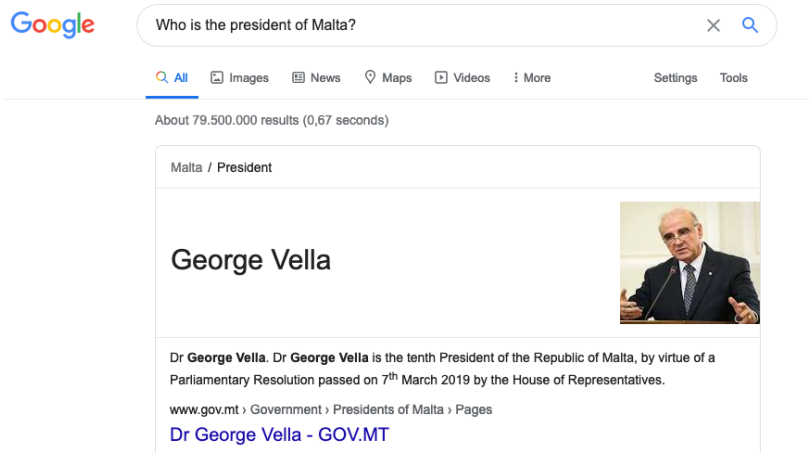


FIGURE 4.1: A knowledge panel by Google immediately answering the question

of the test, the test subjects would not be required to find an effective answer to the questions, the focus lies on the queries they created and the websites they visited. A set of five questions of this format were tested on several fellow students of the author to give a first impression if this approach could work. The search queries generated by the test subjects appeared to be artificial and the test subjects did not feel as if they would make real searches this way.

4.1.2 Testing approach

As the previous approaches resulted in unrealistic search behaviour, another approach was prepared for the user study. In this approach the test subject is asked to pick a situation from a list which he can best resemble with but did not experience recently. The last requirement is made to avoid the subject from having prior knowledge which could result in an unrealistic search. Some examples of the situations on the list are:

- Your mobile phone broke down and your searching for a new one.
- You are planning to travel to a certain destination and are preparing the trip.
- You are having visitors and need to prepare a meal.
- You are tasked to give a presentation about a specific brand (e.g. Coca-Cola).

Each test subject would be tested in two stages. In the first stage, no additional requirements are asked and the user is tasked to search as he would at home. The results of this stage would be used to provide an answer to the following questions:

- How often does a user make a typing mistake while typing his search query?
- How often does a user not finish his query and choose a suggestions from the list?

These questions test the assumptions made for KREEP in section 3.1. In the second stage, the test subject is asked to focus on typing the search queries without mistakes and to type them completely without using the suggestions. The results of this stage would be used to test the applicability of the complete test if the assumptions of KREEP are met.

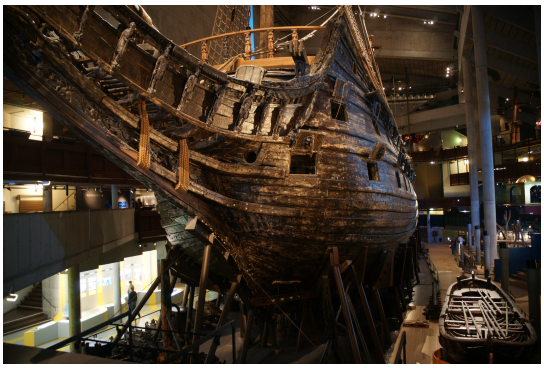


FIGURE 4.2: Image of the Swedish Vasa ship
Vasa from JavierKohen, licensed under CC BY-SA 3.0

4.1.3 Technical setup

During the test the subject has access to a computer running the Firefox web browser. All browser data is cleared in between participants. On this computer a network packet capture program is running to generate a log of all network packets sent and received. The web browser is configured to dump the TLS key logs in order to be able to decrypt the network packet traces when investigating the test runs afterwards. A browser extension is installed in the browser to generate extra logs about the user behaviour. This extension generates a log file containing a list of all queries entered and the websites visited by the subject. This log file would later be used to automatically evaluate ESQABE.

4.1.4 Cancellation of the user tests

The user tests were fully prepared, but could eventually not be executed on real subjects due to the world wide outbreak of COVID-19 [Hui et al., 2020]. In order to slow down the spread of this virus, the Belgian government decided that human contact should be limited to the absolute minimum and the university closed for students [Hasselt University, 2020]. This situation made it impossible to execute this user study as the participants were required to use the provided computer in a controlled environment.

4.2 Automated tests

As an alternative to the user tests, an automated testing approach is followed to evaluate the effectiveness of ESQABE. To simulate a realistic victim, a list of queries was created consisting of several sources. A first set of queries was provided by Yahoo in context of their Webscope program[1]. However, the sample of queries made available was extracted in January 2009, which is a long time ago. This list also contained a lot of search queries related to illegal and adult content on the internet. As the search queries would be visited automatically, these were filtered out manually to prevent security and legal risks. The second (more limited) set of queries is extracted from Google Trends[2], where the most popular search queries of the last 2 years are extracted. The total list of queries selected for this test consisted of 452 unique queries.

[1] https://webscope.sandbox.yahoo.com/
[2] https://trends.google.com/trends/

The tests where all run by an automated script which uses the Selenium WebDriver[3]. For each test run the number of results opened by the victim was specified. All tests were run using the Firefox web browser on a device running macOS 10.14. At the beginning of every set of tests, a fresh Firefox profile was generated which did not contain any browser caches. During each test, TShark[4] was used to capture a network traffic trace just as 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.

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 and prepare to start typing.
3. Mimic a user who is typing the query by entering one character at the time and leaving pauses of 0.3 seconds between two characters.
4. When done 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 which 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 these rates, they are based on numbers of an external company, Blacklinko, which made Search Engine Optimisation its core business [Dean, 2019]. They analysed a dataset of 5 million search query logs to conclude their ratings.
6. A selected page is opened, and after the page was loaded a small scroll movement was executed to simulate a user searching for information. After a delay of 3 seconds, 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.

For all queries in the dataset three captures were executed with respectively one, two and three results opened afterwards. After the generation of the traces they were automatically passed through to ESQABE and the results were evaluated.

4.3 Results

ESQABE is created gradually, the steps taken to introduce certain improvements are based on the results found while testing a certain step. This can be seen in the following testing approach.

4.3.1 Detecting query length

The first and one of the most important steps of ESQABE is to determine the length of the words in the search query. As this is used to build upon in the next steps, a high accuracy needs to be reached. If this step is incorrect, it is impossible for ESQABE to generate correct results.

[3] `https://selenium.dev`
[4] `https://www.wireshark.org`

	KREEP short trace[a]	**KREEP** long trace[b]	**ESQABE** long trace[b]
Correct total length	98.70%	19.13%	92.90%
Correct word length	74.89%	19.03%	87.97%

[a] Captured traffic traces end after query was completely typed and does not include load of results page. Results as described in [Monaco, 2019].
[b] Traces generated as described in section 4.2

TABLE 4.1: Correctly identified query length (Firefox)

The approach taken by ESQABE is heavily based on the approach as described by Monaco, 2019 as used in KREEP. According to the evaluation of KREEP, it should be able to detect the length of the search query almost perfectly. Their tests showed that in 98.70% of the traces captured when using the Firefox browser, they could correctly detect the total length. When using Google Chrome, an even higher rate of 99.72% was reached. As the detection of the lengths of the separate words already needs a correct total length, these rates are slightly lower with 74.89% and 81.12% for respectively Firefox and Chrome. They executed their tests using 4000 unique queries on Google for both browsers. These queries were generated of a dataset consisting of randomly selected fragments from the Enron email corpus and the Gigaword Newswire corpus. These corpora were retyped by 168000 volunteers for another study and the logs of these typist were used as test sets [Dhakal et al., 2018]. Every testrun was executed in a fresh browser window with all cookies cleared. Two seconds after the page was fully loaded, they started replaying the keystrokes as they appeared in the dataset. According to the example trace made available in the public code repository[5], their traces stopped capturing after the query was typed completely and the results page was not loaded.

This, however, is no realistic behaviour. Users who enter a search query do this because they are looking for something and will probably press enter and visit the search page. To evaluate the behaviour of KREEP in a realistic scenario, it was tested using 1095 traces generated as described in the previous section. This traces did include packets from the user opening the results page and visiting several results. The original KREEP implementation determined the correct search query length for only 19.13% of the traces, with a correct recognition of spaces in only 19.03%. This was a lot lower than the results as described by Monaco, 2019. The adapted version of this attack, as discussed in section 3.2.2, is used in ESQABE and did perform better. In 92.90% of the traces the correct length was detected and in 87.97% of the traces the length of the individual words was guessed correctly. The detection of word lengths is even higher than the initial amounts reported by KREEP. The results are summarised in table 4.1.

4.3.2 Detection of the visited search results

For the evaluation of the detection of the visited web pages, the same approach is followed. The results are shown in table 4.2. 78.80% of all results opened by the victim were identified correctly by the algorithm. However, the situation in which a website is visited multiple times appears to cause problems with the algorithm as web browsers keep several connections opened. In this case no new TLS handshake

[5] https://github.com/vmonaco/kreep

# of opened search results[a]	Individual detection rate	Group detection rate[b]
1	79.70%	79.70%
2	78.39%	65.82%
3	78.77%	61.38%
1, 2 or 3	78.80%	69.59%

[a] The amount of results opened by the victim after entering the search query.
[b] The percentage of queries for which all results opened were detected correctly.

TABLE 4.2: Accuracy of ESQABE for the detection of the websites opened from the search results page

# of opened search results	Correct query on 1st place	Correct query in top 3
1	28.68%	34.77%
2	34.67%	44.22%
3	36.30%	46.20%
1, 2 or 3	32.96%	41.36%

TABLE 4.3: Accuracy of ESQABE for the search query identification

is performed and the algorithm guesses that one of the domains hosting the external resources was the visited web page.

4.3.3 Identification of search queries

And finally, the correct amount of suggestions is evaluated. These results are shown in table 4.3. The more results are visited by the victim, the more information available to the eavesdropper and the more search queries are guessed correctly.

Succeeding in, or failing with this approach is highly dependant on the search queries and the websites visited afterwards and multiple factors come into play. But the most important factor is the detection of the length of the search query. If this is determined incorrectly, the eventual result is certainly wrong.

4.4 Comparison with KREEP

ESQABE and KREEP [Monaco, 2019] have the same goal, identifying which search query the user typed when using a search engine as an eavesdropper. Both use the same method for the determination of the search query length. However, all other techniques used by the attacks are completely different. KREEP does only utilise the traffic generated by search engine autocomplete but ESQABE also uses traffic generated by the user when opening the result pages. KREEP on the other hand requires a real using typing the query on the keyboard, as the time between consecutive key presses is used to guess the word typed. And in contrary to ESQABE which searches for words on websites, KREEP uses a language model to generate hypothesis queries. This means KREEP is limited to the words included in the language model.

Testing KREEP the way ESQABE was tested and visa versa is not possible because both approaches made different abstractions. The testing approach used to evaluate ESQABE as described in section 4.2 does enter a search query on a fixed speed.

But for KREEP the time between characters is important, and the are not available for the search query dataset used. The approach used to test KREEP, on the other hand, used random sentences from emails and news articles which do not represent realistic search queries as ESQABE does require.

This is why comparing KREEP and ESQABE is difficult. The query length detection algorithm is already compared in section 4.3.1. The adapted version of ESQABE clearly outperforms KREEP in realistic traces. Both attacks do differ in the way they deliver their estimations of the actual search query. KREEP delivers a list of 50 potential queries, some with only minor changes while ESQABE delivers a ranking of 0 to 3 potential queries. In the case of KREEP, the actual search query appeared in the list of 50 suggestions in only 15-16% of the tests executed, depending on the browser used. While for ESQABE, the query appeared in the top 3 for 34-46% depending on the amount of results opened. Note that these are the results generated by different testing approaches, but in the overall view, ESQABE does outperform KREEP.

4.5 Potential improvements

ESQABE cannot guess queries with a score of 100%, so improvements are still possible. In this section potential improvements are discussed based on the observations made in the latest experiments.

4.5.1 Using WikiData

Some search queries are related to topics which appear to be hard to find. An example of these topics is travel related search queries. Some travel websites are focused on a specific region of interest or a specific country, and in these cases the homepage often provides enough information. But in other cases, the travel related website serves information about destinations all over the world. In this case, the homepage only shows a small excerpt of what is available on the website. Examples of these websites in travel related search queries are AirBnb and Booking.com. For travel related websites for example, it can be interesting to test if a region or city was the main search query for example.

To automatically identify the topic of these interesting websites WikiData[6] can be used. WikiData is a sister project of Wikipedia, but focuses on providing open structured data on the internet [Wikidata contributors, 2020]. A large website or platform, like these examples[7][8], all have an entry in WikiData and these entries can be found by filtering where the *Official Website*[9] property equals the target domain name. The entries of these websites in WikiData are linked to a certain industry in which they are active. For example, AirBnB is labeled as a website in the tourism industry, which on its turn is a subclass of the travel industry. And Booking.com, on the other hand, is labeled as a website in the travel industry directly. This could make it possible for ESQABE to automatically detect the topic of the website and choose an appropriate list of words.

[6] `https://www.wikidata.org/`
[7] AirBnB: `https://www.wikidata.org/wiki/Q63327`
[8] Booking.com: `https://www.wikidata.org/wiki/Q4035313`
[9] `https://www.wikidata.org/wiki/Property:P856`

However, travel is not the only branch for which this approach can be interesting. Other examples can be lists of television series when websites of broadcasters are opened and so on.

4.5.2 Detection of typographical errors

At this point, the evaluation of ESQABE expected the search queries to be typed without any mistake. As the reader will probably know, nobody types without a mistake. The amount of typographical errors made in search queries appeared to be rather limited (10-15%) in 2004 [Cucerzan and Brill, 2004], but there is no more recent research which proves this is still the case.

Typographical errors and movements of the cursor make it impossible for the attack as described by KREEP to detect the individual word length. Possible future work could try to detect these movements and typographical errors to extend the cases in which ESQABE can be executed. Cursor movements or the deletion of letters in the query do not cause any extra traffic at all, so a potential attack can only support on the information available. We see opportunities to detect the total length even if changes were made because changing a character causes a new autocomplete request with the same length. The main challenge will be to deduce this request from other new autocomplete requests of the same size where the request size did not increment due to HPACK.

4.5.3 Exploiting incremental HPACK

HPACK does not leak useful information to the eavesdropper in general. But, when it is used incrementally as in the autocomplete traffic, interesting information can be deduced [Monaco, 2019]. As described in section 2.3.2, HPACK uses a static Huffman table to encode string literals. This way, not all alphanumeric characters are encoded in 7 bits as with ASCII, but in a variable number of bits ranging from 5 to 8 as is the case with the table used by HPACK. This characteristic causes that some autocomplete requests are of the same length, even if an additional character was added. This could be used to filter the results of ESQABE even further.

4.5.4 Search it yourself

Another possible improvement is to test the potential search queries and validate which results appear on the page. This can provide an indication to ESQABE if he made the right guess. However, as search several search engines personalise the results [Horling and Kulick, 2009], this cannot be seen as a perfect solution. Personalization can happen by using many different factors, but some will be the same for the eavesdropper running ESQABE and the victim as for example the location. When the attack is run in an environment which uses Network Address Translation then even the IP address can be the same.

Chapter 5

Defences

It is clear that the current mechanisms in place to protect users from eavesdroppers are not sufficient. In this chapter, possible defences and countermeasures are discussed so users can protect themselves better against these eavesdroppers. Also, a multi purpose browser extension is proposed in order to protect and educate end-users about the information what is still leaked to possible eavesdroppers. It is clear that the combination of small amounts of information can cause eavesdroppers to know a lot about their victims. The browser extension automatically activates all implemented defences, but they can be turned off by the user on request.

5.1 Defending against search query length leakage

The first defence, makes it impossible for ESQABE to determine the length of the different words in the search query by looking at the autocomplete traffic. As discussed in section 3.7, not all search engines appear to be vulnerable to this attack and their behaviours serve as an inspiration for the defences. Startpage.com, a search engine which focuses on privacy, effectively prevents the detection of the search query length by the combination of several approaches. Table 5.1, for example, illustrates which autocomplete requests are visible for an eavesdropper when the victim typed *Hasselt University*.

5.1.1 Random padding

A first countermeasure is the introduction of random padding. Every autocomplete request contains an extra HTTP GET parameter `idx` which contains a string of a random amounts of bytes. This on the other hand causes the eventual record size visible to the eavesdropper to not be incremental anymore. As Startpage.com does not track its users, there is no other traffic flowing between the victim and their servers, so the requests are all consecutive. But it is for an eavesdropper impossible to detect possible spaces or the eventual search query length.

Adding random padding to encrypted payloads is not a new invention. In fact, both TLS [Rescorla and Dierks, 2008] and HTTP/2 [Belshe et al., 2015] provide methods to introduce padding in their headers or payloads. HTTP/2 explicitly states: "Padding can be used to obscure the exact size of frame content and is provided to mitigate specific attacks within HTTP" [Belshe et al., 2015]. But, adding padding is not a magical, one size fits all solution. Research by Dyer et al., 2012 showed that adding packet to packets did not protect against website fingerprinting attacks, as these are mainly based on the coarse features. In large training stets, the amount

of padding averages out and did not have a big impact on the performance of the tested fingerprinting attacks.

In this case however, there is a difference. To extract the length of the search query, fine grained details are used. The smallest adaption of length can cause the attack to fail. Detection of spaces and the exploitation of incremental HPACK compression are impossible if the lengths cannot be extracted exactly. The extraction of the length of a search query is based on the detection of the incremental pattern, which will not be the case anymore if random padding is introduced.

Introducing this extra padding, introduces an extra cost as this involves sending nonsensical bytes over the internet. The number of requests were padding is added, is rather limited which limits the impact of the overhead. In the end, the end user will not notice a difference if some extra bytes are added to the request.

The padding behaviour is implemented in the browser extension by introducing a new header in the HTTP request as these extensions have no access to lower layer protocols. The extension adds a new header called `X-Padding` to each auto complete request. The payload of this header is a random string of a random length between 0 and 40 bytes. To generate the random value and length the `Crypto.getRandomValues()` method is used, in favour of the `Math.random()` method which does not provide cryptographically secure random numbers and could be predicted by the eavesdropper [Watson, 2017; Stark et al., 2009].

5.1.2 Throttling requests

Another possible defence is to only show new suggestions if the user stops typing for a small timeout. One could argument that a user who keeps typing is not paying attention to the suggestions until he stops. Startpage.com was the only search engine taking this approach. In the example shown in table 5.1, *hasselt* was typed very slowly and suggestions were updated frequently, but *university* was typed a lot more fluently which made the list of suggestions update a lot less frequently.

Search engines which use this throttling behaviour send a lot less auto complete requests, so an eavesdropper does get less requests on which they can support their judgement about the length of the query. Due to the use of HPACK compression, the amount of characters is not known anymore to the eavesdropper. For example, a requests with a one byte increase could represent one or two characters, as one of both could have caused, but not necessarily, a zero byte increase.

However, throttling requests can impact the user experience as updates are not shown as frequently as with real time updates. The eventual suggestions are only shown after a small delay which can feel slow, so throttling is not a preferred solution.

In the browser extension, the user can choose two approaches to introduce throttling. The first option is to throttle the autocomplete request as Startpage.com and DuckDuckGo do. In this mode, requests are kept in a waiting queue and only sent if no new request arrives within 500 ms. The second option is to turn of the auto complete suggestions altogether. This last option does influence the user experience with the search engine, but can assure a very privacy sensitive user that no data is leaked to an eavesdropper. When no auto complete requests are sent, an eavesdropper can not use it to deduce search query length.

Actual request		**Visible for eavesdropper**	
Request for	Random bytes	Record Size	Timestamp (s)
h	49	318	16.66
ha	38	148	18.23
hass	30	143	19.77
hasse	48	158	20.63
hasselt	7	126	21.99
hasselt u	41	156	23.35
hasselt un	36	155	24.18
hasselt university	31	155	26.63

TABLE 5.1: An overview of auto complete requests on Startpage.com for an example query of *Hasselt University*

5.2 Hiding domain names

As explained in section 2.4, other approaches are already proposed to hide domain names for an eavesdropper. And it appears that plain text domain names will start belonging to the past. The encryption of domain names will not only benefit victims which are targets for eavesdroppers but also people where for example governments censor the internet using the plain text domain name. However, these governments do not like the fact that the encryption of all domain names can evade their filtering systems and try to find ways to avoid it. For example, Chai et al., 2019 stress the importance of ESNI becoming dominant so these regimes cannot block all traffic which is using ESNI as it would lead to blocking to much false positives.

However, currently, there is still a way to go. To enable the use of ESNI [Rescorla et al., 2020], the site owners need to update their DNS records to include the key used for ESNI, their server software needs to support ESNI and the client browser has to know how to use it. Of the top 1 million websites 10.9% supported the extension in 2019, which is still under development [Chai et al., 2019]. The large CDN provider Cloudflare and web browser Firefox, started to implement draft versions of the protocol since 2018 to already start with the first adoption of the protocol but there is still a long way to go.

In the meanwhile there already exists attacks as OpenKnock by Di Martino et al., 2020 which provide ways for eavesdroppers to still determine a domain name the victim visited even if it was encrypted. Or extraction from the OCSP protocol which several browsers use to verify if a certificate is not revoked yet. They do this by sending a serial number of the certificate in plain text to a server so they can verify the certificate. This request can also be captured by an eavesdropper. This serial number is unique and an eavesdropper can use a website like `https://crt.sh/` to search the associated certificate. The *Common Name* or *Subject Alternative Name* field provides the list of hostnames the certificate is used for [McElroy, 2019]. In this case an attacker can narrow the scope, but these certificates can contain a multitude of domains, so an eavesdropper will not have 100% certainty. This issue is not yet broadly discussed, but a Firefox issue[1] is already opened to search for a defence.

[1] `https://bugzilla.mozilla.org/show_bug.cgi?id=1535235`

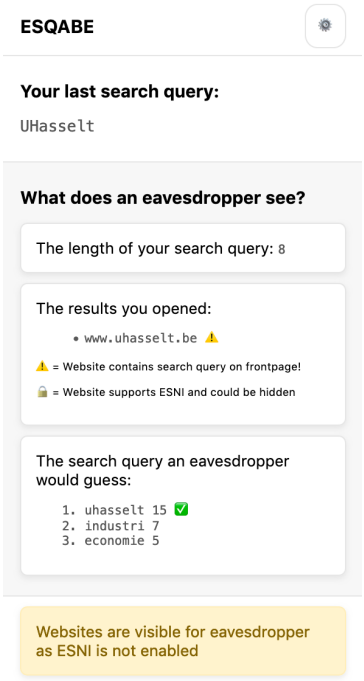


FIGURE 5.1: Screenshot of the ESQABE browser extension popup

The browser plugin shows if a website does support the use of encrypted SNI and checks your browser if the extension is already enabled. To check if a domain supports ESNI, the extension checks the availability of the DNS record containing the public key for ESNI. Depending on which version of server and client is implemented, this can be on a different place. For the first supported version by Firefox and Cloudflare, the key could be found in a `TXT`-record on the `_esni`-subdomain of the website.

5.3 Misleading ESQABE

Apart from hiding the length of the search query, ESQABE can still detect the webpages which were effectively visited As an attacker using ESQABE knows these websites were visited in context of the search query, he can learn the victim probably had the intention to open a website about this subject.

Effectively hiding the traffic for ESQABE is not possible as the eavesdropper has gained a position on the path between the victim and the server of the website. They can however mislead the eavesdropper by visiting other websites unrelated to the search query so an eavesdropper does not know what traffic is legitimate and what is used to mislead him. Doing this at moment a result is opened makes the attack described in section 3.4 harder. But, this approach also involves extra network traffic which can delay the effective page load. b

5.4 Educating users

The browser extension not only functions as a defence, it also shows its users what an eavesdropper using ESQABE would possibly see. The visualisation is shown in figure 5.1. Running a complete instance of ESQABE would require the user to

install the tool including packet capture software. This is a cumbersome task to do and would probably make the tool less handy to use. This is why the behaviour of ESQABE is simulated inside the browser extension.

The simulation in the browser differs from the the real tool as it has access to the not encrypted requests the browser is going to make. Generating the lists of results opened is a trivial task in the extension, and in contrary to ESQABE will always be correct. But, to be able the generate the predictions as ESQABE would do, the browser extension does also visit the homepage of the websites to extract the list of suggestions. The Wikipedia fingerprinting is not implemented in this extension.

The goal of the extension is to show users what an eavesdropper could still know about your search behaviour although HTTPS is used. Domains that support ESNI are shown with a padlock and are even crossed out when both DNS over HTTPS and ESNI are enabled in the browser. The keywords extracted are ranked as ESQABE would do and the correct query is highlighted with a green checkmark.

Chapter 6

Conclusion & reflection

6.1 Conclusion

The first internet was build on trust and security, privacy measurements were absent. When the popularity of the internet started to grow, large steps were taken into protecting the users of the internet. And mainly thanks to HTTPS and TLS large improvements were made. But as shown in this thesis, users browsing the web still leak information which can be used by eavesdroppers. The main topic of this thesis focused on ESQABE, describing the combination of techniques which can result in the eavesdropper finding the search query a victim entered. The techniques may not provide much information when used separate, but the combination of these techniques is the strength of ESQABE.

The first research question can be answered with yes, it is possible for an eavesdropper to determine what a victim is searching for. However, not every search query is deducible as there are some requirements to make it possible. Several search engines are not vulnerable, and the user has to type his query completely. Apart from this, the applicability of ESQABE was shown in a real life testing scenario. The moment a search query happened could be detected in every case and 78.80 % of the visited websites were detected correctly. It is shown that an eavesdropper could correctly suggest the search query as one of the three bests in roughly 41% of the test cases. A user study is described but had to be cancelled due to the worldwide COVID-19 outbreak, but can still be executed in future work.

The introduction of HTTPS was a major step towards the protection of the privacy of the internet user against potential eavesdroppers. However, they appear to insufficient against eavesdroppers who keep finding new ways to spy on end users. The introduction of encrypted DNS and ESNI can cause a real revolution as these will hide the last plain text occurrences of a host name in network traffic. This will make it impossible for eavesdroppers to know for sure which domains have been visited and potential censorship is made a lot harder. The first steps are already taken, and victims can start to protect themselves against these eavesdroppers. Hiding the length of their search query, or even the moment they search can be done with the implemented browser extension and making connections with random domain names can make it harder to determine which results were opened.

There is still room for improvement as already suggested in this thesis. Several proposal can probably increase the correctness of ESQABE, but all these suggestions do not evade the defences discussed. It is the combination of the multiple sources of information which make it possible for an eavesdropper to detect a search query. Without domain names and the query length ESQABE will not be possible.

6.2 Reflection

Writing a master thesis is the last achievement of five years at the university. The project is of such a scale which I have never done before. I was unsure about what to expect, but I can certainly tell I have learned a lot while writing this thesis. This ranged from new information about protocols and technical information to how to write (and rewrite) an English text. It certainly sparked my interest into cyber security even more. I have also learned that the introduction of new encryption protocols are not always fantastic as they for example involve centralising some infrastructure.

The right evaluation approach was crucial for this thesis. With an attack as ESQABE proposed without evaluation it would be guessing if it actually worked. Everybody told me that evaluation needs time. I even started planning this early in the process. But in the end, it will always take more time than expected.

The outbreak of COVID-19 came unexpected and the quarantine measurements forced me to change the evaluation strategy. This was unfortunate as the preparations were already taken and the proposed scheme was already tested in informal environments. Eventually, I think the alternative approach taken for the evaluation functions a good alternative within the limits in place.

One thing I have certainly grown in is how to determine if something is interesting to take a look at, and how long one needs to dwell on something. In the beginning, I took to much time researching paths that led to nowhere. Eventually, I think I have learned to find the right balance.

Overall, I am certain I have learned a lot writing this thesis. Not only about cyber security, but about doing computer science research in general.

Appendix A

Dutch Summary

A.1 Inleiding

Het oorspronkelijke internet is gebouwd als een netwerk tussen verschillende universiteiten. Het werd als een veilige omgeving gezien waar niemand iets kwaadaardig zou uitsteken en dus was beveilging ook niet noodzakelijk. Het internet is echter gegroeid en informatieveiligheid is steeds belangrijker geworden. Aan de andere kant is de waarde van die informatie sterk gestegen. Doordat bijvoorbeeld Facebook en Google weten wat hun gebruikers interessant vinden, kunnen ze gerichte reclameruimte verkopen aan hun adverteerders. Doordat die advertenties zo gericht zijn, leveren ze vaak betere resultaten op aan een adverteerder en kunnen Google en Facebook advertenties duurder verkopen. Van deze bedrijven is algemeen geweten dat ze informatie van hun gebruikers bijhouden, maar gebruikers lekken ook op andere manieren gegevens.

Elke online actie veroorzaakt communicatie tussen de gebruiker en een server over het internet. Dat gaat echter niet over een rechtstreekse lijn maar passeert via verschillende tussenliggende instanties. Als een gebruiker thuis zit is dat zijn internet service provider, maar buitenshuis wordt er vaak verbonden met publieke Wi-Fi netwerken. In een café, een restaurant, een pretpark, het openbaar vervoer, er zijn heel veel plekken die internet aanbieden aan het bezoekers. Deze bedrijven hebben dan toegang tot de ruwe acties die gebruikers op hun netwerk uitvoeren.

Daarbuiten zijn er ook kwaadaardige aanvallers die geïnteresseerd zijn in de persoonlijke gegevens van internetgebruikers. Denk bijvoorbeeld aan een aanvaller die een social engineering attack wil uitvoeren. Deze moet dan bepaalde details over zijn slachtoffer te weten zien te komen. Op een open, onbeveiligd Wi-Fi netwerk is meeluisteren eenvoudig aangezien er daar geen extra beveiliging toegevoegd wordt. Voor een gesloten, beveiligd netwerk is dit al een stuk moeilijker, maar ook daar zijn er reeds aanvallen bekend die meeluisteren mogelijk maakten zoals bijvoorbeeld [Vanhoef en Piessens, 2017].

Niet alle ruwe data die onderschept kan worden is even interessant voor iemand die aan het meeluisteren is. Tegenwoordig wordt het gros van deze gegevens geëncrypteerd. Uit cijfers van Google Chrome blijkt dat zelfs meer dan 80% van de geopende webpagina's gebruik maken van HTTPS [*HTTPS encryption on the web - Transparency Report*]. HTTPS verkeer dat onderschept wordt is onverstaanbaar voor een aanvaller en de inhoud van de communicatie is beschermd. Maar HTTPS bevindt zich in de applicatielaag, wat betekent dat alle bovenliggende lagen niet geëncrypteerd zijn en er dus nog steeds informatie lekt. De vraag is echter: Hoe interessant is die

informatie en wat kan er uit afgeleid worden? Dat heeft geleid tot de uiteindelijke onderzoeksvragen van deze thesis:

- Is het mogelijk voor een afluisterende aanvaller om te achterhalen waar zijn slachtoffer naar aan het zoeken is op een zoekmachine in een realistisch scenario?
  - Kan die aanvaller zien wanneer zijn slachtoffer aan het zoeken is?
  - Kan die aanvaller achterhalen welke resultaten zijn slachtoffer open geklikt heeft?
- En als een aanvaller die zoekopdracht kan achterhalen, hoe kunnen potentiële slachtoffers zich daartegen beschermen en hun zoekopdrachten verborgen houden?

De keuze van zoekopdrachten is niet toevallig. De effectieve zoekopdracht is al verborgen door HTTPS, wat het probleem niet triviaal maakt. Daarbij is zo een zoekopdracht effectief interessant voor een aanvaller. Een aanvaller kan zien met welke servers zijn slachtoffer verbindt, maar dat kan een vertekend beeld geven over de interesses van zijn slachtoffer. Een toestel dat verbinding maakt met een server van Apple doet dit niet altijd op aanvraag van zijn gebruiker. Deze toestellen doen dit om bijvoorbeeld informatie te synchroniseren. Maar een zoekopdracht is altijd door de gebruiker geïnitieerd en toont een bepaalde interesse in dat onderwerp aan. Men gaat namelijk niet iets opzoeken waar men totaal niet mee bezig is.

Om een antwoord te bieden op de onderzoeksvragen is deze thesis opgevat als een haalbaarheidsstudie waarbij er een tool, genaamd ESQABE, geïmplementeerd is die probeert om zoekresultaten te achterhalen. De werking van deze tool is geëvalueerd en er zijn ook beschermingsmaatregelen geïmplementeerd in een browser extensie. Op die manier kunnen gebruikers zich beschermen en is het voor een aanvaller niet meer mogelijk om ESQABE te gebruiken om zoekopdrachten te achterhalen.

A.2 Geëncrypteerd verkeer afluisteren

In deze thesis wordt er uitgegaan van een bepaald type aanvaller namelijk de afluisteraar. Deze aanvaller heeft zich ergens in het pad kunnen plaatsen tussen zijn slachtoffer en de rest van het internet. Dat kan bijvoorbeeld een internet service provider zijn, de beheerder van het netwerk waarmee het slachtoffer verbonden is of een aanvaller die aan het meeluisteren is naar draadloos verkeer. In het laatste geval wordt ervan uitgegaan dat eventuele encryptie zoals bij WPA2 reeds gekraakt is zoals bijvoorbeeld in KRACK [Vanhoef en Piessens, 2017]. Op die manier heeft de aanvaller toegang tot het volledige verkeer van een gebruiker naar het internet. Dat verkeer houdt niet alleen de geëncrypteerde HTTPS pakketten in maar ook onder andere DNS verkeer en TLS handshakes. Verder verwachten we ook dat er geen gebruik gemaakt wordt van andere beveiligingsmaatregelen zoals een VPN of SSH tunnel. Wel verwachten we dat alle websites die de gebruiker bezoekt beveiligd zijn met HTTPS. We laten websites over HTTP, zonder beveiliging, buiten beschouwing omdat deze steeds zeldzamer worden en daar extractie van informatie triviaal is.

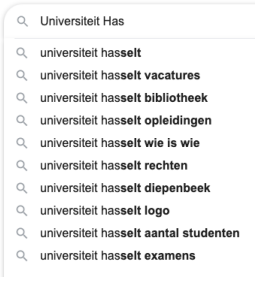
HTTPS wordt ook wel HTTP over TLS genoemd omdat het gebruik maakt van het TLS protocol wat zich in de applicatielaag situeert. Bij HTTPS wordt het volledige frame geëncrypteerd en zijn zowel de HTTP-body als -headers niet zichtbaar voor een afluisteraar. Maar TLS is een protocol in de applicatielaag, dat betekent dat alle

protocollen in de lager gelegen lagen niet geëncrypteerd worden. Dat is logisch, want een tussenliggende router moet bijvoorbeeld kunnen zien wat het IP-adres van de bestemming is om het pakket correct door te kunnen sturen. Ook heeft TLS encryptie geen invloed op de totale lengte van het HTTP pakket, geëncrypteerd of niet, de lengte blijft steeds behouden. Dat is dus zichtbaar voor een aanvaller.

Een aanvaller heeft ook toegang tot de domeinnamen van de websites waarmee zijn slachtoffer verbindt. Deze lekken uit in verschillende andere protocollen dan HTTP die niet beveiligd zijn. Het eerste voorbeeld daarvan is de Domain Name Service (DNS), ook wel het adresboek van het internet genoemd. Om een domeinnaam in een IP-adres om te zetten stuurt het besturingssysteem een DNS-request naar een resolver. Het antwoord hierop bevat de lijst van IP-adressen die verwijzen naar die domeinnaam. Standaard gebeurt die communicatie zonder encryptie en is dat dus perfect op te vangen voor een afluisteraar. Echter zijn recentelijk DNS-over-HTTPS (DoH) en DNS-over-TLS (DoT) geïntroduceerd die het mogelijk maken om deze communicatie te versleutelen [Hu e.a., 2016; Hoffman en McManus, 2018]. Hier zijn echter nadelen aan verbonden, denk bijvoorbeeld aan een bedrijfsfirewall die bepaalde websites wil blokkeren en vaak steunt op dat DNS verkeer. Er worden daar oplossingen voor gezocht die dat in de toekomst toch mogelijk zouden kunnen maken.

De introductie van HTTPS introduceerde nog een ander probleem op het internet. In de header van HTTP pakketten zit namelijk een `Host` of `:authority` header die aangeeft voor welke website het pakket bedoeld is. Dit zorgt er voor dat er op één server meerdere websites gehost kunnen worden. Echter, om een TLS verbinding op te zetten is er nood aan een TLS handshake. In die handshake wordt het encryptie algoritme vastgelegd en stuurt de server een certificaat naar de gebruiker om zijn authenticiteit te bewijzen. Maar, hier zit een probleem. Verschillende websites gebruiken namelijk verschillende certificaten en als de aangesproken server meerdere websites host, weet hij op dat moment niet welk certificaat hij zou moeten terugsturen. Daarom werd de Server Name Extension (SNI) geïntroduceerd in TLS [Blake-Wilson e.a., 2003]. Deze extensie kan worden toegevoegd aan het eerste bericht dat naar de server verstuurd wordt. Zo weet de server met welke website de gebruiker verbinding wil maken en kan hij het juiste certificaat terugsturen. Maar die extensie is niet geëncrypteerd want op dat moment is er geen gemeenschappelijke encryptiesleutel afgesproken. Men is hier op zoek naar oplossingen zoals bijvoorbeeld Encrypted SNI wat de hostname encrypteert met een sleutel beschikbaar in DNS.

Verder is er al veel onderzoek beschikbaar over Webpage fingerprinting aanvallen (WPF). Deze aanvallen zijn gebaseerd op het idee dat geëncrypteerd verkeer de effectieve inhoud misschien wel verbergt, maar dat de vorm telkens hetzelfde blijft. Men kan dit bekijken als een statistische analyse van het verkeer. Van verschillende webpagina's waarin de aanvaller geïnteresseerd is, maakt deze zogenaamde vingerafdrukken. Dat is een collectie van karakteristieke kenmerken van die webpagina. Hiervoor bezoekt de aanvaller die pagina's verschillende keren en probeert hij zo bepaalde eigenschappen te extraheren. Voorbeelden hiervan zijn de groottes van de verschillende pakketten of alle IP-adressen die betrokken zijn in het laden van die webpagina.



FIGUUR A.1: Een voorbeeld van de automatische aanvulling bij Google.

A.3 Het achterhalen van zoekopdrachten

Tot nu toe heeft de aanvaller al op verschillende manieren informatie kunnen achterhalen over zijn slachtoffer. Al die informatie kan nu gecombineerd worden om te achterhalen waar een gebruiker naar aan het zoeken is. Dat is de kern van deze thesis: de beschrijving van ESQABE, een aanval die het mogelijk maakt om uit alle informatie die nog lekt zoekopdrachten te achterhalen. De implementatie van ESQABE werd gefocust op Google omdat elke zoekmachine zijn eigen karakteristieken heeft om uit te buiten, maar de aanval is uitbreidbaar naar andere zoekmachines zoals aangehaald wordt in de thesis.

De eerste stap is het achterhalen van de lengte van de zoekopdracht. Hiervoor wordt gebruikt gemaakt van informatie die lekt door de automatische aanvulling van zoekmachines. Een voorbeeld van deze functionaliteit is weergegeven in figuur A.1. Elke keer als de gebruiker een nieuw karakter intypt, wordt er een nieuwe lijst van suggesties getoond. Aangezien deze lijst op de servers van de zoekmachine gegenereerd wordt, moet er telkens een request naar de servers van de zoekmachine verstuurd worden. Dat request bevat de onvolledige zoekopdracht die zich op dat moment in het zoekveld bevindt. Maar in die requests valt er een patroon te herkennen, opeenvolgende requests verschillen namelijk maar 0 of 1 byte in grootte. Dat is ook zichtbaar voor een afluisteraar die op die manier kan achterhalen wat de lengte van de zoekopdracht was op basis van het aantal requests. Doordat een spatie geëncodeerd wordt als `%20` is er in dat geval een verschil in grootte van 2 of 3 bytes op te merken. Op die manier kan een aanvaller bepalen hoe lang de verschillende woorden van de zoekopdracht van zijn slachtoffer zijn. Er is wel een voorwaarde aan deze aanval verbonden: het slachtoffer moet zijn volledige zoekopdracht intypen, anders zal de lengte niet kloppen. Deze stap is gebaseerd op een bestaande aanval genaamd KREEP, maar is uitgebreid en aangepast om efficiënter en effectiever te werken in real-life scenarios [Monaco, 2019].

Als volgende stap worden alle verschillende domeinnamen geëxtraheerd uit het verkeer dat de aanvaller heeft kunnen aftappen. Zoals eerder al gesuggereerd is de lijst van domeinnamen die een attacker kan afleiden veel langer dan alleen de websites die de gebruiker effectief bezocht heeft. Daarom zal ESQABE nu proberen om enkel de websites te extraheren die geopend zijn doordat een gebruiker een nieuw resultaat geopend heeft. Dat gebeurt aan de hand van elke specifieke kenmerken. Als iemand een nieuwe website opent dan zal een afluisteraar een tijdelijke stijging waarnemen in het aantal pakketten dat per seconde passeert. Dit omdat een website vaak verschillende resources zoals afbeeldingen en scripts moet inladen. De eerste verbinding van zo een piek zal echter altijd de website zijn waarmee de gebruiker

effectief wou verbinden. Die website moet de webbrowser namelijk instructies geven over de locatie waarvan de andere resources binnengehaald moeten worden. Let wel, we spreken hier over domeinnamen en niet over de volledige url. Het pad bevindt zich namelijk in een HTTP header die ook geëncrypteerd is en dus niet zichtbaar voor een aanvaller.

Nu dat de aanvaller zowel de lengte van de woorden van de zoekquery als de bezochte websites kent, kan hij deze informatie gebruiken om mogelijke zoekopdrachten af te leiden. De belangrijkste aanpak bestaat eruit dat de aanvaller de startpagina van de verschillende websites gaat bezoeken. Die pagina's kunnen informatie prijsgeven over een bepaalde zoekopdracht. Neem nu bijvoorbeeld een gebruiker die zoekt naar "iPhone" en terechtkomt op de specifieke pagina over de iPhone op de website van Apple. De afluisteraar zal dan zien dat zijn slachtoffer een zoekopdracht van zes karakters ingetypt heeft en dat hij apple.com bezocht. Op de hoofdpagina van Apple staat verschillende keren het woord "iPhone", wat dan de zoekquery was. Het is echter niet het enige woord van zes karakters op de startpagina, maar de lijst is eerder van beperkte aard.

Echter, voor bepaalde websites zal de hoofdpagina de aanvaller niet veel wijzer maken. Als het slachtoffer bijvoorbeeld de Wikipediapagina van "iPhone" opende, dan zal de aanvaller niet veel kunnen leren van de startpagina van Wikipedia. Wikipedia beschikt over in totaal meer dan 35 miljoen artikelen, die worden niet allemaal op de hoofdpagina beschreven. Maar, Wikipedia kan wel gebruikt worden om de lijst van suggesties te filteren. Zo kan er een Webpage Fingerprinting Attack uitgevoerd worden op een zeer beperkte training set. Want, de aanvaller kan voor elke potentiële zoekopdracht het bijhorende Wikipedia artikel opzoeken. Van die artikelen kan hij een fingerprint maken en die vergelijken met het verkeer dat hij van zijn slachtoffer opgevangen heeft om zo te achterhalen welk artikel bezocht werd.

Verder kan de incrementele HPACK compressie ook gebruikt worden om bepaalde termen te filteren. In HTTP/2 kunnen de headers van een request gecompresseerd worden aan de hand van HPACK compressie. Vandaar dat de autocomplete requests niet consistent 1 byte groter worden, maar soms ook dezelfde grootte behouden. Dit omdat karakters voorgesteld worden door een variabele hoeveelheid bits, namelijk tussen 5 en 8.

A.4 Evaluatie van ESQABE

Aangezien deze effectiviteit van deze aanval afhankelijk is van factoren was er nood aan een grondige evaluatie. Die was oorspronkelijk opgevat als een gebruikerstest. Omwille van de uitbraak van de wereldwijde pandemie veroorzaakt door COVID-19 en de daarbij horende maatregelen was het niet mogelijk dit uit te voeren. De aanpak voor het testen is beschreven in het 4de hoofdstuk van de thesis.

Uiteindelijk is er dan geopteerd voor een automatische evaluatie gebaseerd op een dataset van veel gebruikte zoekopdrachten vrijgegeven door Yahoo aangevuld met zoekopdrachten uit Google Trends. Voor deze evaluatie is het gedrag van een gebruiker gesimuleerd voor 450 zoekopdrachten. De test is meermaals uitgevoerd met telkens een verschillend aantal resultaten wat geopend wordt. De computer typt automatisch de zoekopdracht in en opent enkele resultaten. Welke resultaten er geopend worden, wordt willekeurig bepaald. Het is echter zo dat resultaten die hoger gerangschikt worden vaker geopend worden, ook dat is geïntegreerd in de tests.

Aantal zoekresultaten geopend	**Geïdentificeerd**	**Geïdentificeerd in top 3**
1	28.68%	34.77%
2	34.67%	44.22%
3	36.30%	46.20%
1, 2 of 3	32.96%	41.36%

TABEL A.1: Correct geïdentificeerde zoekopdrachten, deze tabel is een kopie van tabel 4.3

Volgens de verwachtingen presteert ESQABE beter als er meer resultaten geopend worden zoals weergegeven in tabel A.1. Dit is te verwachten, omdat het een gok blijft of dat de hoofdpagina van een website effectief een deel van de zoekopdracht bevat, als er meer websites geopend worden is de kans groter dat de zoekopdracht terug te vinden is. Verder worden er in de thesis nog mogelijke verbeteringen voorgesteld.

A.5 Gebruikers beschermen

Gebruikers kunnen zich op verschillende manieren beschermen tegen de aanval, besproken in ESQABE. Om dat eenvoudiger te maken is er in deze thesis een browser extensie geïmplementeerd die enkele van deze beschermingen uitvoert. Eén daarvan is bijvoorbeeld het toevoegen van strings van willekeurige lengte aan de autocomplete requests. Hierdoor verdwijnt het patroon dat zichtbaar is voor een aanvaller en wordt het moeilijker om te achterhalen welk verkeer afkomstig is van de autocomplete functionaliteit. Zoekmachines als DuckDuckGo en Startpage.com hebben, misschien zelfs onbewust, een extra maatregel doorgevoerd die ook beschikbaar is in de extensie. Dit gaat namelijk over het beperken van het aantal requests. Zo sturen deze diensten pas een aanvraag voor nieuwe suggesties op het moment dat een gebruiker gedurende enkele milliseconden gestopt is met typen. Dit kan gedaan worden om de server minder te belasten, maar zorgt er ook voor dat er veel minder informatie lekt naar de aanvaller. De informatie die toch lekt is te weinig om de lengte van de woorden in de zoekopdracht te achterhalen.

Het verbergen van de domeinnamen is eerder afhankelijk van de evoluties in de internet protocollen. Daar worden, zoals reeds vermeld, al verschillende stappen gezet. Kijk naar bijvoorbeeld DNS-over-HTTPS en Encrypted SNI (ESNI) / Client Hello. De browser extensie is niet in staat om deze domeinnamen te verbergen aangezien daarvoor ook de server op de hoogte moet zijn. Wel toont de extensie of een bepaalde website al dan niet de encrypted SNI extensie ondersteunt en is het mogelijk om “misleiden” in te schakelen. Dan zal de extensie op het moment dat de gebruiker een bepaalde website opent ook gelijktijdig verbinding maken met andere domeinen of pagina’s. Wat bijvoorbeeld fingerprinting dan weer een stuk moeilijker maakt.

Behalve beschermen heeft de extensie ook een didactische functie. Hij stimuleert namelijk wat een afluisteraar zou kunnen achterhalen en toont dit aan de gebruiker. Daardoor is een gebruiker zich meer bewust van het feit wat iemand kan zien die wil meeluisteren.

A.6 Conclusie

De onderzoeksvragen van de thesis kunnen kort beantwoord worden. Het is mogelijk voor een aanvaller om een gebruiker af te luisteren en de zoekopdracht te achterhalen. Zo slaagt hij er eenvoudig in om te achterhalen wanneer een gebruiker aan het zoeken is en kan hij zien welke websites er geopend werden na de zoekopdracht. De aanval is geïmplementeerd en kan geautomatiseerd uitgevoerd worden door een aanvaller. Er zijn echter eenvoudige manieren om de gebruiker te beschermen tegen deze aanvallen die dan ook geïmplementeerd zijn in een browserextensie.

Bibliography

Adobe. *Adobe Creative Cloud Network Endpoints*. URL: https://helpx.adobe.com/enterprise/kb/network-endpoints.html (visited on 04/28/2020).

Bahajji, Zineb Ait and Gary Illyes (Aug. 6, 2014). *HTTPS as a ranking signal*. URL: https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html (visited on 04/28/2020).

Belshe, Mike, Roberto Peon, and Martin Thomson (May 2015). *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540. DOI: 10.17487/RFC7540. URL: https://rfc-editor.org/rfc/rfc7540.txt.

Blake-Wilson, Simon et al. (June 2003). *Transport Layer Security (TLS) Extensions*. RFC 3546. DOI: 10.17487/RFC3546. URL: https://rfc-editor.org/rfc/rfc3546.txt.

Borgolte, Kevin et al. (2019). "How DNS over HTTPS is Reshaping Privacy, Performance, and Policy in the Internet Ecosystem". In: *TPRC 47*. DOI: 10.2139/ssrn.3427563.

Bortzmeyer, Stéphane (Aug. 2015). *DNS Privacy Considerations*. RFC 7626. DOI: 10.17487/RFC7626. URL: https://rfc-editor.org/rfc/rfc7626.txt.

Calvano, Paul (2019). "Caching". In: *Web Almanac*. URL: https://almanac.httparchive.org/en/2019/caching8.

Can I use? *Server Name Indication*. URL: https://caniuse.com/#feat=sni (visited on 12/08/2019).

Chai, Zimo, Amirhossein Ghafari, and Amir Houmansadr (Aug. 2019). "On the Importance of Encrypted-SNI (ESNI) to Censorship Circumvention". In: *9th USENIX Workshop on Free and Open Communications on the Internet (FOCI 19)*. Santa Clara, CA: USENIX Association. URL: https://www.usenix.org/conference/foci19/presentation/chai.

Cheng, Heyning and Ron Avnur (1998). "Traffic analysis of SSL Encrypted Web Browsing". In: *URL citeseer. ist. psu. edu/656522. html*.

Cheng, N. et al. (2013). "Characterizing privacy leakage of public WiFi networks for users on travel". In: *2013 Proceedings IEEE INFOCOM*, pp. 2769–2777. DOI: 10.1109/INFCOM.2013.6567086.

Cloudflare Support (Oct. 2019). *How does Cloudflare work?* URL: https://support.cloudflare.com/hc/en-us/articles/205177068-How-does-Cloudflare-work- (visited on 11/29/2019).

Cucerzan, Silviu and Eric Brill (July 2004). "Spelling Correction as an Iterative Process that Exploits the Collective Knowledge of Web Users". In: *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*. Barcelona, Spain: Association for Computational Linguistics, pp. 293–300. URL: https://www.aclweb.org/anthology/W04-3238.

Dance, Gabriel J.X., Michael LaForgia, and Nicholas Confessore (Dec. 18, 2018). "As Facebook Raised a Privacy Wall, It Carved an Opening for Tech Giants". In: *The New York Times*. URL: https://www.nytimes.com/2018/12/18/technology/facebook-privacy.html (visited on 06/05/2020).

Dean, Brian (Aug. 2019). *We analyzed 5 million Google search results - Here's What We Learned About Organic Click Through Rate*. URL: https://backlinko.com/google-ctr-stats (visited on 05/15/2020).

Deckelmann, Selena (Feb. 2020). *Firefox continues push to bring DNS over HTTPS by default for US users*. Mozilla. URL: https://blog.mozilla.org/blog/2020/02/25/firefox-continues-push-to-bring-dns-over-https-by-default-for-us-users/ (visited on 05/15/2020).

Dhakal, Vivek et al. (2018). "Observations on Typing from 136 Million Keystrokes". In: *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM. DOI: https://doi.org/10.1145/3173574.3174220.

Di Martino, Mariano, Peter Quax, and Wim Lamotte (2019). "Realistically Fingerprinting Social Media Webpages in HTTPS Traffic". In: *Proceedings of the 14th International Conference on Availability, Reliability and Security*. ARES '19. Canterbury, CA, United Kingdom: Association for Computing Machinery. ISBN: 9781450371643. DOI: 10.1145/3339252.3341478. URL: https://doi.org/10.1145/3339252.3341478.

– (2020). "Knocking on IPs: Identifying HTTPS Websites for Zero-Rated Traffic". In: *Manuscript submitted for publication*.

Dyer, K. P. et al. (2012). "Peek-a-Boo, I Still See You: Why Efficient Traffic Analysis Countermeasures Fail". In: *2012 IEEE Symposium on Security and Privacy*. IEEE, pp. 332–346.

Facebook (Apr. 2020). *Facebook Q1 2020 Results*. URL: https://s21.q4cdn.com/399680738/files/doc_financials/2020/q1/Q1-2020-FB-Earnings-Presentation.pdf (visited on 06/05/2020).

Farahat, Ayman and Michael C. Bailey (2012). "How Effective is Targeted Advertising?" In: *Proceedings of the 21st International Conference on World Wide Web*. WWW '12. Lyon, France: Association for Computing Machinery, 111–120. ISBN: 9781450312295. DOI: 10.1145/2187836.2187852. URL: https://doi.org/10.1145/2187836.2187852.

Fidler, Andy (2019). *A UK ISP view on DNS over HTTPS*. British Telecom. URL: https://www.icann.org/sites/default/files/packages/ids-2019/08-fidler-icann-dns-symposium-a-uk-isp-view-on-doh-issue-11may19-en.pdf (visited on 02/17/2020).

Fielding, Roy T. and Julian Reschke (June 2014). *Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*. RFC 7230. DOI: 10.17487/RFC7230. URL: https://rfc-editor.org/rfc/rfc7230.txt.

Google. *Google APIs*. URL: https://googleapis.github.io/ (visited on 04/28/2020).

– *HTTPS encryption on the web - Transparency Report*. URL: https://transparencyreport.google.com/https/overview (visited on 11/29/2019).

– *What is 1e100.net?* URL: https://support.google.com/faqs/answer/174717 (visited on 11/05/2019).

Grigorik, I. (2014). *High Performance Browser Networking*. Shroff Publishers & Distr. ISBN: 9789351104711. URL: https://hpbn.co/.

Hasselt University (2020). *Coronavirus*. URL: https://www.uhasselt.be/Coronavirus (visited on 05/24/2020).

Hoffman, Paul E. and Patrick McManus (Oct. 2018). *DNS Queries over HTTPS (DoH)*. RFC 8484. DOI: 10.17487/RFC8484. URL: https://rfc-editor.org/rfc/rfc8484.txt.

Horling, Bryan and Matthew Kulick (Dec. 2009). *Personalized Search for everyone*. URL: https://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html (visited on 06/03/2020).

HTML - Living Standard: DOM. https://html.spec.whatwg.org/multipage/dom.html. Accessed: 2020-04-22.

httparchive.org. *Report - Page Weigth*. URL: https://httparchive.org/reports/page-weight (visited on 04/28/2020).

Hu, Zi et al. (May 2016). *Specification for DNS over Transport Layer Security (TLS)*. RFC 7858. DOI: 10.17487/RFC7858. URL: https://rfc-editor.org/rfc/rfc7858.txt.

Hui, David S. et al. (Feb. 2020). "The continuing 2019-nCoV epidemic threat of novel coronaviruses to global health — The latest 2019 novel coronavirus outbreak in Wuhan, China". English. In: *International Journal of Infectious Diseases* 91, pp. 264–266. ISSN: 1201-9712. DOI: 10.1016/j.ijid.2020.01.009.

Krombholz, Katharina et al. (2015). "Advanced social engineering attacks". In: *Journal of Information Security and Applications* 22. Special Issue on Security of Information and Networks, pp. 113 –122. ISSN: 2214-2126. DOI: https://doi.org/10.1016/j.jisa.2014.09.005. URL: http://www.sciencedirect.com/science/article/pii/S2214212614001343.

Kurose, J. and K. Ross (2017). *Computer Networking: A Top-Down Approach, Global Edition*. 7th ed. Pearson Education Limited. ISBN: 9781292153605. URL: https://books.google.be/books?id=IUh1DQAAQBAJ.

Lanze, Fabian et al. (2014). "Undesired Relatives: Protection Mechanisms against the Evil Twin Attack in IEEE 802.11". In: *Proceedings of the 10th ACM Symposium on QoS and Security for Wireless and Mobile Networks*. Q2SWinet '14. Montreal, QC, Canada: Association for Computing Machinery, 87–94. ISBN: 9781450330275. DOI: 10.1145/2642687.2642691. URL: https://doi.org/10.1145/2642687.2642691.

Lewandowski, Dirk and Ulrike Spree (2011). "Ranking of Wikipedia articles in search engines revisited: Fair ranking for reasonable quality?" In: *Journal of the American Society for Information Science and Technology* 62.1, pp. 117–132. DOI: 10.1002/asi.21423, eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.21423. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.21423.

Liberatore, Marc and Brian Neil Levine (2006). "Inferring the Source of Encrypted HTTP Connections". In: *Proceedings of the 13th ACM Conference on Computer and Communications Security*. CCS '06. Alexandria, Virginia, USA: Association for Computing Machinery, 255–263. ISBN: 1595935185. DOI: 10.1145/1180405.1180437. URL: https://doi.org/10.1145/1180405.1180437.

Martínez, Antonio García (Feb. 26, 2019). "No, Data Is Not the New Oil". In: *WIRED*. URL: https://www.wired.com/story/no-data-is-not-the-new-oil/ (visited on 06/05/2020).

McElroy, Sean (Jan. 2019). *Despite DoH and ESNI, with OCSP, web activity is insecure and not private*. URL: http://blog.seanmcelroy.com/2019/01/05/ocsp-web-activity-is-not-private/ (visited on 04/28/2020).

Mihajlija, Milica (July 2019). *Establish network connections early to improve perceived page speed*. URL: https://web.dev/preconnect-and-dns-prefetch/ (visited on 05/20/2020).

Monaco, John V (2019). "What are you searching for? a remote keylogging attack on search engine autocomplete". In: *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pp. 959–976.

Morla, Ricardo (2017). "Effect of Pipelining and Multiplexing in Estimating HTTP/2.0 Web Object Sizes". In: *CoRR* abs/1707.00641. arXiv: 1707.00641. URL: http://arxiv.org/abs/1707.00641.

Mozilla Support (2019). *Firefox DNS-over-HTTPS*. URL: https://support.mozilla.org/en-US/kb/firefox-dns-over-https (visited on 12/08/2019).

Nielo, David (May 8, 2018). "Simple Steps to Protect Yourself on Public Wi-Fi". In: *WIRED*. URL: https://www.wired.com/story/public-wifi-safety-tips/ (visited on 06/05/2020).

Peon, Roberto and Herve Ruellan (May 2015). *HPACK: Header Compression for HTTP/2*. RFC 7541. DOI: 10.17487/RFC7541. URL: https://rfc-editor.org/rfc/rfc7541.txt.

Qixiang Sun et al. (2002). "Statistical identification of encrypted Web browsing traffic". In: *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 19–30.

Rauti, Sampsa and Ville Leppänen (2012). "Browser Extension-Based Man-in-the-Browser Attacks against Ajax Applications with Countermeasures". In: *Proceedings of the 13th International Conference on Computer Systems and Technologies*. CompSysTech '12. Ruse, Bulgaria: Association for Computing Machinery, 251–258. ISBN: 9781450311939. DOI: 10.1145/2383276.2383314. URL: https://doi.org/10.1145/2383276.2383314.

Rescorla, Eric (May 2000). *HTTP Over TLS*. RFC 2818. DOI: 10.17487/RFC2818. URL: https://rfc-editor.org/rfc/rfc2818.txt.

– (Aug. 2018). *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. DOI: 10.17487/RFC8446. URL: https://rfc-editor.org/rfc/rfc8446.txt.

Rescorla, Eric and Tim Dierks (Aug. 2008). *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246. DOI: 10.17487/RFC5246. URL: https://rfc-editor.org/rfc/rfc5246.txt.

Rescorla, Eric et al. (Mar. 2020). *Encrypted Server Name Indication for TLS 1.3*. Internet-Draft draft-ietf-tls-esni-06. Work in Progress. Internet Engineering Task Force. 27 pp. URL: https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-06.

Rizzo, Juliano and Thai Duong (2012). *The CRIME attack*. Ekoparty. URL: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU (visited on 12/27/2019).

Rose, Daniel E. and Danny Levinson (2004). "Understanding User Goals in Web Search". In: *Proceedings of the 13th International Conference on World Wide Web*. WWW '04. New York, NY, USA: Association for Computing Machinery, 13–19. ISBN: 158113844X. DOI: 10.1145/988672.988675. URL: https://doi.org/10.1145/988672.988675.

Savage, Joni and Angela Lazar (2018). *Encrypted SNI Comes to Firefox Nightly*. URL: https://blog.mozilla.org/security/2018/10/18/encrypted-sni-comes-to-firefox-nightly/ (visited on 05/12/2020).

– (2019). *Canary domain - use-application-dns.net*. URL: https://support.mozilla.org/en-US/kb/canary-domain-use-application-dnsnet (visited on 05/12/2020).

Siby, Sandra et al. (2020). "Encrypted DNS –> Privacy? A Traffic Analysis Perspective". In: *Network and Distributed System Security Symposium (NDSS 2020)*. Internet Society. URL: https://www.ndss-symposium.org/wp-content/uploads/2020/02/24301-paper.pdf.

Sombatruang, N. et al. (2018). "The continued risks of unsecured public Wi-Fi and why users keep using it: Evidence from Japan". In: *2018 16th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, pp. 1–11. DOI: 10.1109/PST.2018.8514208.

Stark, E., M. Hamburg, and D. Boneh (2009). "Symmetric Cryptography in Javascript". In: *2009 Annual Computer Security Applications Conference*. IEEE, pp. 373–381. DOI: 10.1109/ACSAC.2009.42.

Taghavi, Mona et al. (2012). "An analysis of web proxy logs with query distribution pattern approach for search engines". In: *Computer Standards & Interfaces* 34.1, pp. 162 –170. ISSN: 0920-5489. DOI: https://doi.org/10.1016/j.csi.2011.07.001. URL: http://www.sciencedirect.com/science/article/pii/S0920548911000808.

Tan, Jiaqi and Jayvardhan Nahata (2013). *Petal: Preset encoding table information leakage*. Tech. rep. Citeseer. URL: https://www.pdl.cmu.edu/PDL-FTP/associated/CMU-PDL-13-106.pdf.

The Chromium Project. *Issue 908132: FR: Support for Encrypted SNI (ESNI)*. URL: https://bugs.chromium.org/p/chromium/issues/detail?id=908132 (visited on 05/12/2020).

– (2019). *DNS over HTTPS (aka DoH)*. URL: https://www.chromium.org/developers/dns-over-https (visited on 12/08/2019).

Valsorda, Filippo (Sept. 2016). *An overview of TLS 1.3 and Q&A*. URL: https://blog.cloudflare.com/tls-1-3-overview-and-q-and-a/ (visited on 12/27/2019).

Vanhoef, Mathy and Frank Piessens (2017). "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. Dallas, Texas, USA: Association for Computing Machinery, 1313–1328. ISBN: 9781450349468. DOI: 10.1145/3133956.3134027. URL: https://doi.org/10.1145/3133956.3134027.

Watson, Mark (Jan. 2017). *Web Cryptography API*. W3C Recommendation. W3C. URL: https://www.w3.org/TR/2017/REC-WebCryptoAPI-20170126/.

Wikidata contributors (2020). *Wikidata:Main Page — Wikidata*, URL: https://www.wikidata.org/wiki/Wikidata:Main_Page (visited on 05/28/2020).

Wikipedia contributors (2020a). *Wikipedia:About — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia\%3AAbout&oldid=952427410 (visited on 05/12/2020).

– (2020b). *Wikipedia:Article Titles — Wikipedia, The Free Encyclopedia*. URL: http://en.wikipedia.org/w/index.php?title=Wikipedia\%3AArticle\%20titles&oldid=953494343 (visited on 05/12/2020).

– (2020c). *Wikipedia:Naming conventions (capitalization) — Wikipedia, The Free Encyclopedia*. URL: https://en.wikipedia.org/wiki/Wikipedia:Naming_conventions_(capitalization) (visited on 05/12/2020).

Witten, Ian H., Eibe Frank, and Mark A. Hall (2011). "Chapter 10 - Introduction to Weka". In: *Data Mining: Practical Machine Learning Tools and Techniques (Third Edition)*. Ed. by Ian H. Witten, Eibe Frank, and Mark A. Hall. Third Edition. The Morgan Kaufmann Series in Data Management Systems. Boston: Morgan Kaufmann, pp. 403 –406. ISBN: 978-0-12-374856-0. DOI: https://doi.org/10.1016/B978-0-12-374856-0.00010-9. URL: http://www.sciencedirect.com/science/article/pii/B9780123748560000109.

Zittrain, J. and B. Edelman (2003). "Internet filtering in China". In: *IEEE Internet Computing* 7.2, pp. 70–77.