Service-Level Monitoring of HTTPS Traffic

Wazen M. Shbair

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Service-Level Monitoring of HTTPS Traffic

(Identification des Services dans le Trafic HTTPS)

THÈSE

présentée et soutenue publiquement le -

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Wazen M. Shbair

Composition du jury

Rapporteurs :	Hervé Debar	Professeur à Télécom SudParis, Évry, France
	Sandrine Vaton	Professeure à Télécom Bretagne, Brest, France
Examinateurs :	Georg Carle	Professeur à Université Technique de Munich, Allemagne
	Radu State	Directeur de recherche à l'Université du Luxembourg, Luxembourg
	Véronique Cortier	Directrice de recherche CNRS, Nancy, France
Encadrants :	Isabelle Chrisment	Professeure à Télécom Nancy, Nancy, France
	Thibault Cholez	Maître de conférences à Télécom Nancy, Nancy, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Acknowledgement

Firstly, I would like to express my sincere gratitude to my supervisors Isabelle Chrisment, Thibault Cholez and Jérôme François for the continuous support of my PhD study and the related research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better supervisors and mentors for my PhD study.

My sincere thanks also goes to my thesis committee: Hervé DEBAR, Sandrine VATON, Georg CARLE, Véronique CORTIER, and Radu STATE for their insightful comments and encouragement, but also for their questions which encourage me to widen my research from various perspectives.

Besides all, I would like to thank my friends in the MADYNES team for a lovely working atmosphere and also the administration staff in LORIA and INRIA Nancy Grand-Est for making our daily activities going smoothly. Also I specially thank Antoine Goichot for his work with me during this thesis.

I thank the Erasmus Mundus EPIC program for funding my PhD study in the University of Lorraine, with an extended thanks to Delphine Laurant and the Palestinian coordinator in An-Najah National University. Also thanks for the CNRS institute for funding partially my thesis.

I am grateful to the academic stuff in the computer engineering department of the Islamic University of Gaza for their support and help. Thanks to Mohammed Hussein, Shafik Jendia and a particular thank go to Hasan Qunoo for enlightening me the first glance of research.

I would like to thank my closest friends, Fayez Abu-hilou, Tareq Arraj, Mohammed Darbouli, Mohsen Hassan, Mounir Kassir, Hussain AJJ, Basem Shama, Hassan Al Dreimly, Omar Ringa, Motaz Saad and their lovely families for the time we have spent together in the last three years.

My family has always played the greatest part in my life. Thanks to those ones who believe in me, my dad and my mum, for supporting me throughout my study and my life in general. My brothers and sister: Wagdi, Magdy, Ahmed and Arwa I am blessed to have you. The sweetest gratitude goes to my companion Rodina, for leading the way to realize this dream and to my beloved daughter Sanaria, her smile was another factor to overcome the toughest days.

Thanks to all of you. I will remain indebted for you forever.

To The Memory of My Uncle
AHMED JAMIL ESHBAIR
1955-2015

Contents

Chapter 1

General Introduction

Contents

1.1 Context

Web applications have become the lifeblood of many sectors like e-commerce, software edition, media, etc., since they are highly interactive, platform independent and easily accessed from anywhere using web browsers, thanks to web technologies like HTML5, JavaScript or PHP. Hence, the reliability of the supporting infrastructure (i.e., Internet, web servers, local networks) is more crucial than ever. Therefore, network monitoring and analysis tools are fundamental to Internet Service Providers (ISP) and network operators to handle security and performance issues and to avoid costly downtime. Such tools must be continuously adapted to pursue the quick evolution of the network traffic and to cope with modern challenges.

Today's network monitoring solutions are slowly but steadily losing their power due to the global trend toward the encryption of network communications, especially after the National Security Agency (NSA) revelations of massive surveillance programs[1], and the raising awareness of users for privacy questions. Companies like Over The Top (OTT) players (e.g., Akamai Technologies, Google) also want to fully keep the control over their traffic and users' data over the Internet. However, encryption undermines the effectiveness of standard monitoring approaches and makes it difficult to identify and monitor the services behind encrypted traffic, which is essential to properly manage the network.

[1] `https://en.wikipedia.org/wiki/Global_surveillance_disclosures_(2013-present)`

The foremost examples of encryption protocols over the Internet are Transport Layer Security (TLS) and its predecessor Secure Socket Layer (SSL). These protocols have been originally designed to secure banking transactions on the web. However, with the arrival of Web 2.0[2], the TLS/SSL usage has been extended widely. Most of the functionalities previously offered by desktop applications (music, office App., file transfer, email, etc.) are now offered by web applications run in the cloud, and they can be securely accessed through web browsers and mobile applications at any time and from anywhere thanks to the HTTPS[3] protocol.

1.1.1 HTTPS usage trend

The HTTPS protocol allows web applications to secure HTTP over TLS or SSL, which makes it difficult for a third party to infer information about users' interaction with a website using packets sniffer or Man-in-the-Middle (MitM) attacks [1]. As a result, the HTTPS traffic is expanding rapidly alongside the need for Internet users to benefit from security and privacy when surfing the web, and this could explain why web users spend two-thirds of their browsing time over HTTPS websites [2].

The trend of using HTTPS creates an "encryption rush" on Internet industry in two dimensions. The first is the size of HTTPS traffic, and the second is the number of HTTPS websites. According to Cisco 2016 annual security report [3], statistics show that the HTTPS traffic accounts for 57% of all web traffic in October 2015. That number is in line with ISPs: in Europe, French ISPs reported that the amount of encrypted traffic reached 50% of the Internet traffic in 2015 [4] against only 5% back in 2012, while another ISP based in North America expects 65-70% of HTTPS traffic by the end of 2016 [5]. Also the augmentation of HTTPS traffic is recorded by Mozilla telemetry, where it shows that the HTTPS traffic has reached a tipping point and passed the halfway mark of the web traffic [6].

Regarding the number of HTTPS websites, Google Transparency Report [7] shows that more than 50% of the web pages were served using HTTPS in October 2016, as illustrated in Figure 1.1. Also the Netcraft annual survey [8] that has been conducted over 860 million websites shows that the year 2013 has undergone a significant change: websites are more and more being served over HTTPS. Among the Alexa top 1 million websites, more than 68% of them use HTTPS, despite the cost of SSL certificates and the more complex configuration of web servers [9]. To facilitate the movement toward HTTPS, the "Let's Encrypt"[4] program gives free SSL certificates and an automated software to configure servers [2].

In total, there is an increase of 48% of websites using HTTPS and of half a million SSL certificates (+22%) on the Internet over the year 2013 [8]. Another recent motivation for websites' owners to use HTTPS is to improve their rank in search engines. Indeed, Google has announced that they now consider the use of HTTPS as a positive ranking parameter [10].

[2] `https://en.wikipedia.org/wiki/Web_2.0`
[3] Hyper Text Transfer Protocol Secure (HTTPS) or HTTP-over-TLS/SSL
[4] `https://letsencrypt.org/`

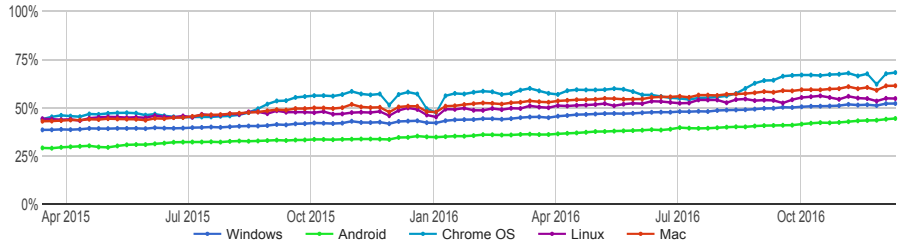


Figure 1.1: Percentage of pages loaded over HTTPS [7]

1.1.2 HTTPS and network monitoring

However, HTTPS is a double edged sword. On one side, the increasing share of the HTTPS traffic has been a mostly positive step toward a more secure web. On the other side, for ISPs and network administrators, HTTPS causes them "blind" to their network traffic and entails their capacity to perform proper network management activities, such as traffic engineering, capacity planning, performance/failure monitoring [11]. For instance, HTTPS prevents network operators from applying Quality of Service (QoS) measurements that give a priority to critical services, or to use caching techniques to reduce network latency and congestion.

From the security perspective, HTTPS makes security monitoring methods unable to understand the traffic and to identify anomalies or malicious activities that can be hidden in encrypted connections [12]. One possible scenario is *Data Exfiltration*, where a local compromised machine transfers sensitive information to an external server controlled by an attacker over an HTTPS channel to circumvent security monitoring systems (e.g., Firewall) and to look like a benign web browsing traffic [13]. The application of security policies to encrypted traffic is also challenging.

The main purpose of the HTTPS protocol is keeping users web browsing activities away from eavesdroppers and tampering. Therefore, appropriate cipher suites are needed and websites' SSL certificate must be verified and trusted. However, HTTPS does not mean that a website's content is not malicious, or welcome in a given network, even with a valid SSL certificate [14]. The authors in [15] find that 13% of websites (over the 10,000 most popular Chinese websites) are using self-signed SSL certificates instead of those issued and verified by a Certificate Authority (CA), after intensive investigation of websites' business information. Even more, based on [16], there is a significant number of phishing websites that use valid SSL certificates issued by trusted CAs to convince clients to trust them. An HTTPS website may be legitimate, but it may embed advertisements that hold malicious JavaScript or viruses, which can be transferred over HTTPS without triggering security check points. Thus, if we do not have the ability to circumvent such websites, we have at least to restrict access to them.

In the field of Internet traffic management, it is important to differentiate between the three following terms: *Identifying*, *Monitoring* and *Filtering*. Identifying is intended to find a fingerprint that can be used as an application identity, such as

specific interaction patterns, keywords or a regular expression string. Monitoring is defined as recording the activities on the network (accessed websites, applications, etc.) for either online or offline investigation. Whereas filtering means restricting access to applications or websites by adding websites URLs to blacklists or using keyword filtering, which is not effective against encrypted HTTPS traffic [17, 18].

Indeed, encrypted content cannot be scanned to detect malware or to check compliance with security policy. Over ten months, from January to October 2015, an investigation over 26 families of malicious web browser add-ons have shown a 40% decrease of the number of infections, however this was a deceptive indication. The real reason was the increasing amount of HTTPS traffic during the experiment period had reduced the ability to detect the infection, as the URL information was no more visible due to encryption [3]. In fact, there is an efficient method to monitor and control the HTTPS traffic based on HTTPS Proxy, where the HTTPS traffic is decrypted in the middle for investigation and re-encrypted again. This easy solution has many supporters from National Security Agencies [19] to security companies [20], and is even discussed for future Internet technical standards [21]. However, such a method cannot be treated lightly as it denies privacy for traffic inspection's sake.

The related work in the field of encrypted traffic monitoring without decryption can be divided in two main parts; the first part is intended to identify at application-level (i.e., Web, P2P, SSH, VoIP, etc.) [22, 23]. While the second one aims to recognize the specific web pages accessed on a given website using website fingerprinting techniques [24–27]. However, identifying the type of encrypted applications is too generic-grain, while the website fingerprinting is too fine-grained, as it works at the page-level with static content, and is no longer adapted to today's web fetching dynamic content from multiple Content Delivery Networks (CDN). For instance, a user accessing Google Maps will be characterized as HTTPS traffic with the first type of technique (application-level), whereas the second type (website fingerprinting) will be only able to identify a specific page of the service. As illustrated in Figure 1.2, accessing the service may not be from the official Google Maps websites, but it can be embedded in another website (e.g., the map displayed on the main page of `www.loria.fr`). We claim that a service-level identification that can identify the precise services accessed through HTTPS is necessary and would constitute a huge step toward the elaboration of monitoring solutions that could properly handle encrypted web traffic. In the previous example, our target is to identify when Google Maps is used independently of the access method.

To summarize this section, the dependability and security of networks and infrastructures will be lowered if we do not have suitable methods to identify, monitor and control the HTTPS traffic, which is quickly becoming the predominant application protocol on the Internet. However, the monitoring of HTTPS traffic has lead to a conflict between security and users' privacy. This is known as "Dilemmas of the Internet age" and has been discussed not only in the academic community but in the overall society and human rights space [28]. The answer to this paradox is complex, as both sides may have valuable arguments. Thus, there is an urgent need for a new

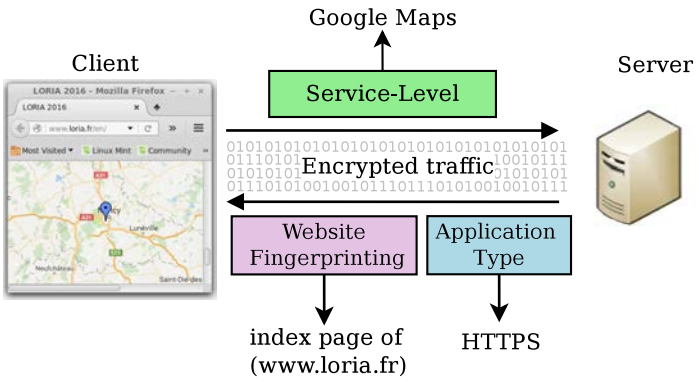


Figure 1.2: Service-level identification

generation of HTTPS traffic monitoring approaches to cope with the global trend of using encryption, while respecting user's privacy over the web.

1.2 Problem Statement

The wide development of HTTPS web applications comes with new issues related to the management of encrypted traffic. Especially, there is an essential need for a new method to monitor HTTPS services, which can be a source of security breaches. Several techniques have been proposed to monitor HTTPS services, however such methods either do not have the proper-level of identification (they do not identify the service) or have a privacy concern related to decryption in the middle (e.g., HTTPS proxy). Therefore, the objective of this thesis is to provide a privacy preserving approach for monitoring HTTPS services at the service-level. Hence, the first challenge is to asses the reliability of a recent method to monitor HTTPS services based on Server Name Indication (SNI), a field of TLS handshake that specifies the name of the accessed service, and to explore and remediate possible weaknesses in the SNI-based method. The second one is to design a more robust method, that do not use header-information, to identify HTTPS services based on traffic pattern or services behaviour. The third challenge is the real-time identification of HTTPS services and how to identify the access to a given HTTPS service very early in the session.

1.3 Our Contribution

Our contribution for the service-level monitoring of HTTPS traffic, as illustrated in Figure 1.3, span over four chapters. In Chapter 2, we explore and study the literature work of HTTPS traffic identification and monitoring based on the necessary steps required to identify HTTPS services. We cover the HTTPS monitoring issues from academic, industrial and legal viewpoints. This step gives a precise view of current HTTPS traffic identification and monitoring techniques and highlights the remaining challenges to be solved.

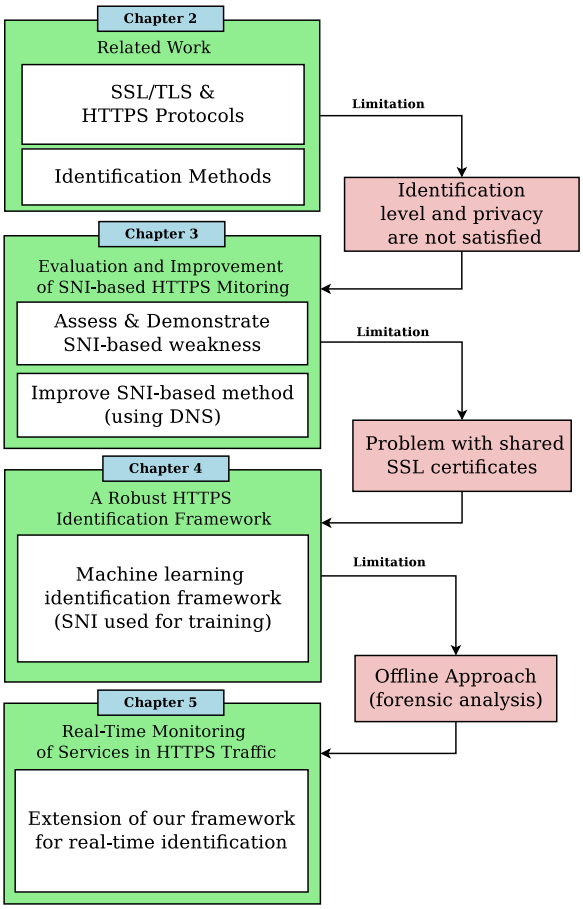


Figure 1.3: Our contributions road map

Chapter 3 evaluates a recent HTTPS monitoring method implemented in many firewall systems, named SNI-based method. We demonstrate that SNI-based monitoring can be easily cheated to bypass firewalls and monitoring systems relying on SNI values. Therefore, we propose an improvement using the Domain Name System (DNS) to remediate the shortage of SNI-based monitoring by detecting suspicious access to HTTPS services. The proposed remediation is evaluated through experiments. Even so, the enhanced SNI-based method is still unable to precisely identify a given HTTPS service if it shares a SSL certificate with other services.

In Chapter 4, we present a more robust method to identify HTTPS services based on traffic pattern rather than inspecting header fields like SNI or using SSL certificate. Hence, we develop a complete framework to identify accessed HTTPS services in a traffic dump. Our framework includes several innovations increasing the identification accuracy and the scalability. We define a new set of statistical features extracted from the encrypted payload. Also, we propose a multi-level classification approach based on machine learning algorithms. The performance of the framework is evaluated using real traffic.

In Chapter 5, we improve our machine learning-based framework to identify HTTPS services in real-time without decryption. By extracting statistical features on TLS handshake packets and progressively on application data packets, we identify HTTPS services very early in the session. We carry out extensive experiments over a significant and open dataset that we have built. In this chapter, we also describe our HTTPS firewall prototype that can identify services in real-time.

Chapter 2

Related work

Contents

2.1 Introduction

The accurate identification and classification of network traffic is the core component of monitoring techniques, since it provides the baseline to evaluate the traffic. Over time, these classification techniques have evolved, started with port-based, IP address-based, DNS-based methods until advanced Deep Packet Inspection (DPI) approaches. The surveys [29–32] are a valuable indicator to understand the evolution of the Internet and how both academic and industrial communities handle traffic identification. Table 2.1 summarizes the main traffic classification goal of these published surveys. The most recent one from Velan et al. [32] focuses on classification approaches for the identification of encrypted traffic over the Internet. They show that, in the past few years, the classification of encrypted traffic in large protocol categories such as IPsec, SSL/TLS, SSH, BitTorrent, Skype, etc. has been widely investigated in the community. They conclude that simply identifying the type of encrypted traffic is not enough and that classification should be improved to identify the underlying services. They also state that much work has been conducted on SSH while TLS should now be at the center of such studies regarding its uttermost importance today.

In this thesis, we go deeper by focusing on the most widely used encrypted application protocol, HTTPS, while trying to go further in the service identification process. We think that this focus is necessary because of the increased amount and complexity of web applications and services using HTTPS [33, 34].

Table 2.1: Main published surveys in the field of Internet traffic classification

Focus	**Covered Period**	**Publish year**	**Survey**
Application identification	2002-2008	2009	[29]
P2P applications identification	1994-2008	2009	[35]
Application identification	2004-2013	2013	[30]
Payload-based identification	2009-2013	2014	[31]
Encrypted traffic identification	2005-2014	2015	[32]

The rest of this chapter proceeds as follows. Section 2.2 provides an overview of web security and the related TLS protocol. Then, as illustrated by Figure 2.1 we detail the scientific related work from the most basic level of protocol identification (TLS, HTTPS), to the finest identification level of naming the websites, as even the web pages accessed through HTTPS. Thus, Section 2.3 considers the identification of TLS among other encrypted traffic types. In Section 2.4, we investigate the methods used to recognize HTTPS applications traffic. Section 2.5 discusses the identification of HTTPS service. Finally, in Section 2.6 we analyse the open research questions and we discuss the trade-off between security, privacy and ethics of monitoring activities.

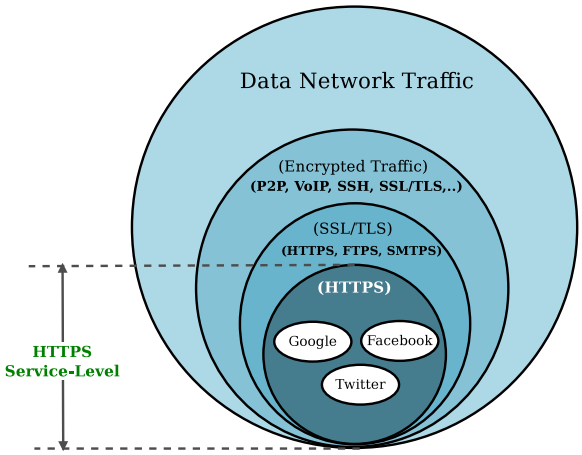


Figure 2.1: Granularity of Internet traffic classification toward HTTPS service monitoring

2.2 Web Security Protocols

With the emergence of the Web 2.0, the increasing number of e-banking, e-government, e-commerce, e-health, etc., web applications is notable. This can be seen as a strong evidence that more and more business activities depend on the web as a way of delivering services. Thus, the web technology had to evolve to satisfy new security and privacy requirements due to the rise of critical web applications [36, 37]. In the context of web application security, the web is a client/server application running over the standard Internet TCP/IP protocols. Therefore, the security protocols have to take into account some considerations when providing methods to secure the web, such as the operating environment (i.e., client/server architecture) and the users. On the client side, web browsers are very easy to use, even if the underlying software is complex and may hold many potential security flaws, in particular when considering plug-in extensions. On the server side, web servers are highly exposed to attacks, which means the level of trust and confidence should be very high. When a web server is down, the damage is not only about money loss but also about enterprise's reputation. Whereas for the web users, most of them are common clients unaware of the security risks that exist and without the experience to handle security problems; such as untrusted SSL certificates warning [38].

There are many potential methods to secure communications, with different mechanisms and their relative location in the TCP/IP protocol layer. Figure 2.2 shows the security protocols related to each layer of the TCP/IP protocol stack. The SSL and TLS security protocols work between the application layer and the transport layer. They use the Public Key Infrastructure (PKI) to provide authentication and then symmetric keys for confidentiality. On the top application layer, we find for example the Secure Shell (SSH) and Pretty Good Privacy (PGP) protocols, which answer to specific security needs of a given application over TCP connection-oriented approach to establish and manage connections [38].

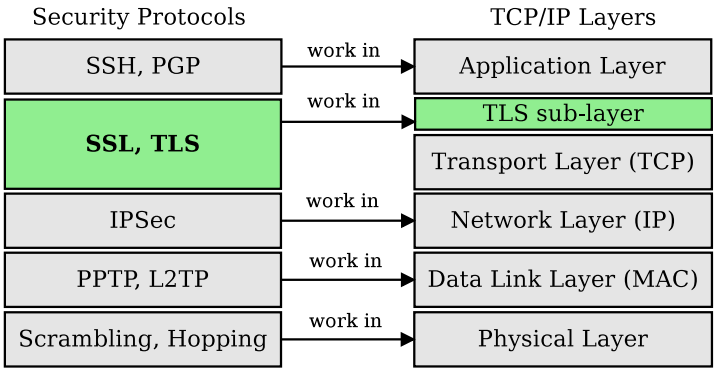


Figure 2.2: The security-related protocols associated with the TCP/IP protocol stack

While all of these security protocols have their advantages, the mainly used ones to secure the web are the SSL and TLS protocols. Therefore, the rest of this section is devoted to discuss the SSL and TLS protocols and how they were designed to enrich web applications with security and the challenges related to their increasing usage.

2.2.1 Introduction to the HTTPS protocol

The HTTPS protocol provides secure web communications, by using HTTP over a secured TLS/SSL connection. To differentiate between HTTP and HTTPS services, the HTTP websites use the port-number 80 by default, while HTTPS ones use the port 443. Here, we need to highlight the difference between HTTPS and encrypted-HTTP. The HTTPS only refers to HTTP-over-TLS/SSL, while encrypted-HTTP can be HTTP over any higher level encrypted connection like VPN, SSH or Tor connections [24]. In the later case, the website's URL starts with "http://". This means that the web server does not provide a secure connection service by itself and the secure link parameters are configured within the chosen method, but not in the server. While in the case of HTTPS, the website's URL starts with "https://" and the website is hosted on an authenticated server that owns a SSL certificate. Thus, a client and a server negotiate the secure connection parameters before exchanging data thanks to HTTP over a dedicated secure link between them [27].

As stated in the introduction, monitoring HTTPS is essential regarding; (1) the fast increase of HTTPS traffic on the Internet; (2) the great diversity of web services that now use the HTTPS protocol to provide privacy and security and, (3) despite HTTPS good intentions, it may be used for illegitimate purposes, such as accessing inappropriate contents or services.

2.2.2 SSL and TLS protocols short history

At the end of 1994, Netscape included the support of the second version of SSL (SSL 2.0) in Netscape Navigator after solving many issues with the first version which

was just used inside Netscape Corporation. In response, Microsoft also introduced in 1995 an encryption protocol named Private Communication Technology (PCT) that was very close to SSL 2.0 [39]. The publication of the two concurrent security protocols created a lot of confusion in the security community, since applications needed to support both for interoperability reasons. To solve this issue, the Internet Engineering Task Force (IETF) formed a working group in 1996 to standardize a unified TLS protocol. After a long discussion with the related parties, the first version of standard protocol (TLS 1.0) appeared in January 1999. In April 2006, the TLS protocol version 1.1 (TLS 1.1) was released, followed by TLS 1.2 in August 2008 which is specified in the RFC5246 [40]. The most recent version of TLS is TLS 1.3, but is still a draft. This last version follows the same specifications but introduces improvements concerning the encryption algorithms parameters and the handshake [41]. Based on this history, we will use the term TLS in the rest of the thesis, while when we talk about the digital certificate, we will use the commonly acknowledge name "SSL certificate".

The primary goal of TLS is to provide a secure channel between authenticated communication parties, to make it impossible for potential third parties to access transmitted data. On the web, the TLS protocol is widely used because it prevents unauthorized view of users' information over the web; no configuration is required from the user side, which makes it easy to be used and all web browsers and web servers fully support TLS natively.

2.2.3 TLS architecture

TLS is not a single protocol but it contains two layers of protocols, as illustrated in Figure 2.3. The top-layer consists of the three handshaking sub-protocols: the Handshake, the Change Cipher Specification, and the Alert protocol. These sub-protocols are used to manage TLS exchanges, as to allow peers to agree on an encryption algorithm and a shared secret key, to authenticate themselves, and to report errors to each other. The lower-layer holds the TLS Record protocol, which can be presented as an envelope for application data and TLS messages from the protocols above. The Record protocol is responsible for splitting data into chunks, which are optionally compressed, authenticated with MAC, encrypted and finally transmitted [40]. The three sub-protocols are introduced in the next subsections.

TLS handshake protocol

The handshake protocol is of prime importance because it defines the first interactions and is responsible for many configuration aspects such as managing cipher suite negotiation, server/client authentication, and session key exchange. During the cipher suite negotiation, a client and a server make agreement about the cipher suite that will be used to exchange data. In authentication, both parties prove their identity using the PKI method. In the session key exchange, they exchange random and special numbers, called the Pre-Master Secret. These numbers are used to create their Master key, which is then applied for encryption when exchanging data.

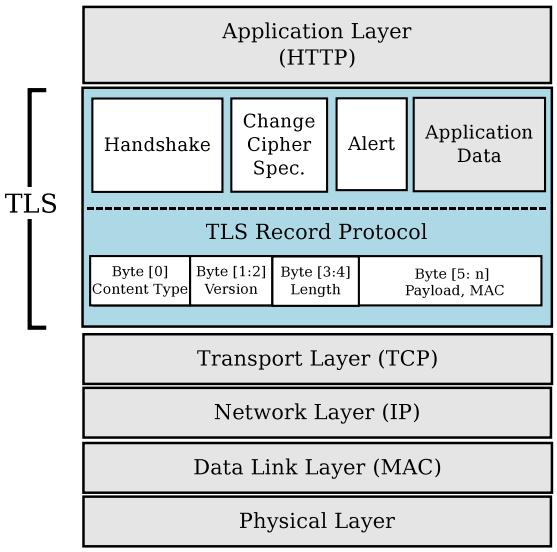


Figure 2.3: The TLS layers and sub-protocols

Figure 2.4 details the sequence of protocol messages for TLS handshake:

1. The `ClientHello` message contains the usable cipher suites, supported extensions and a random number.
2. The `ServerHello` message holds the selected cipher, supported extensions and a random number.
3. The `ServerCertificate` message contains a certificate with a server public key.
4. The `ServerHelloDone` indicates the end of the `ServerHello` and associated messages. If the client receives a request for its certificate, it sends a `ClientCertificate` message.
5. Based on the server random number, the client generates a random Pre-Master Secret, encrypts it with the public key given in the server's certificate and sends it to the server.
6. Both client and server generate a master secret from the Pre-Master Secret and exchanged random values.
7. The client and the server exchange `ChangeCipherSpec` to start using the new keys for encryption.
8. The client sends the `Finished` message to verify that the key exchange and authentication processes were successful.
9. The server sends the `Finished` message to the client to end the handshake phase.

Once both sides have received and validated the `Finished` message from its peer, they can exchange encrypted application data over the new TLS connection.

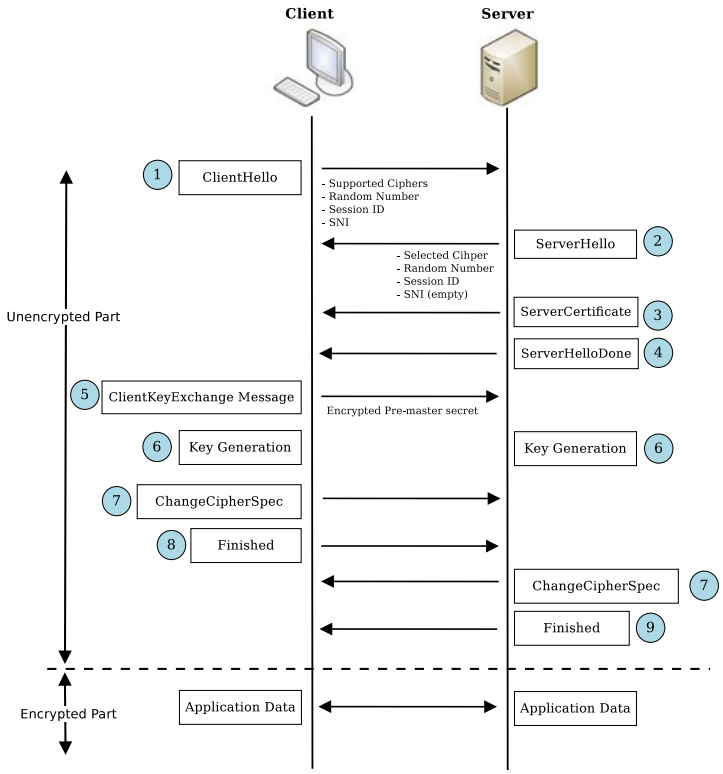


Figure 2.4: The TLS handshake protocol messages sequence

Table 2.2: TLS handshake protocol messages parameters

TLS Handshake Messages	**Parameters**
`ClientHello`	Version, Random, SessionID, Cipher suite, Compression methods, Extensions
`ServerHello`	Version, Random, SessionID, Selected Ciphers, Compression methods, Extension
`Certificate`	Chain of X.509 certificates
`ServerKeyExchange`	Parameters, Signature
`CertificateRequest`	Type, Authorities
`ServerDone`	Null
`CertificateVerify`	Signature
`ClientKeyExchange`	Parameters, Signature
`Finished`	Hash value

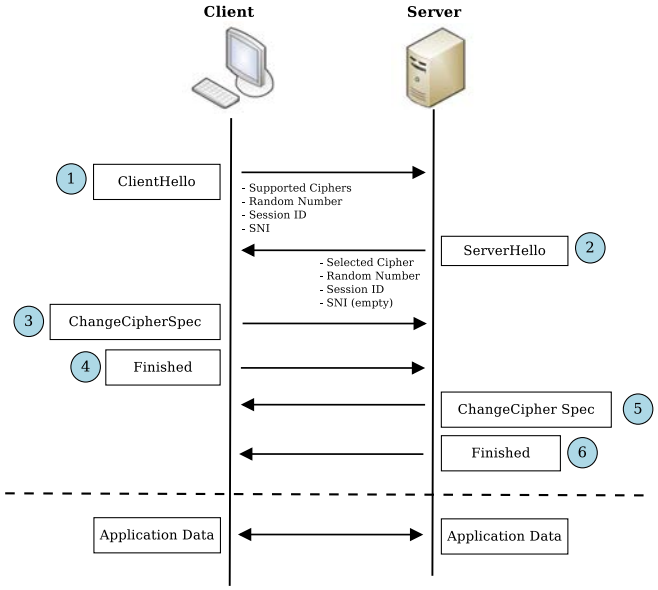


Figure 2.5: Resume TLS handshake protocol messages sequence

Table 2.2 summaries the parameters of TLS handshake messages. Figure 2.5 shows the abbreviated TLS handshake that allows performance gains by only requiring the recomputing of session keys when a TLS connection is resumed. The steps passed over from the full handshake are `ServerCertificate`, `ServerHelloDone` and `ClientKeyExchange`. Since the server and client already know each other and share the master key, those steps are eliminated. The resumed session is triggered by the client submitting the existing session ID from previous connections in the `ClientHello` and the server shall respond with the same session ID in the `ServerHello` (if the IDs are different, a full handshake is required) [40].

Change cipher specification protocol

This protocol changes the encryption algorithm being used by the client and server. Thus, subsequent records will be protected under the newly negotiated algorithm and keys. The protocol consists in a single message that holds a single byte. It also permits a change in the TLS session without having to renegotiate the connection. The `ChangeCipherSpec` message is normally sent at the end of the TLS handshake (i.e., Step 7 in Figure 2.4) [40].

SSL alert protocol

Alerts may be issued at any time, either when the connection has to be closed, or when an error occurs. The Alert messages convey the level of the message (Warning or Fatal) and a description of the alert, which contains either "*Close notify*", "*Unexpected message*" or "*Bad record MAC*" [40].

2.2.4 Key role players in TLS development

The TLS protocol provides the ability to secure communications between a client and a server, which means both parties should be configured to deal with the TLS protocol. Here, we present different implementations of the TLS protocol. We also provide a description of the SSL certificate, which is a keystone in the TLS connection establishment.

TLS implementation libraries

The RCF5246 [40] describes the specifications of TLS version 1.2 and precisely specifies the interaction between a client and a server using the TLS protocol. However, there are many implementations of the same protocol specification. The author of [42] presents a comparison between eight open source TLS libraries; libgcrypt, Libmcrypt, Botan, Crypto++, OpenSSL, Nettle, Beecrypt and Tomcrypt. Regarding his experiments, all TLS libraries contain approximately the same core cipher implementation, although the libraries' performances vary. For example, OpenSSL and Beecrypt have the highest optimization levels, but these libraries only implement a few ciphers, while Tomcrypt, Botan and Crypto++ implement many different ciphers.

In this work, we target the identification of HTTPS services accessed by web browsers (e.g., Firefox, Google Chrome), which use different implementations of the TLS protocol. Mozilla Corporation uses Network Security Services (NSS) libraries to provide the support for TLS in Firefox web browser and other services (e.g., Thunderbird, SeaMonkey) [43]. Previously, Google Chrome also used the NSS libraries, but Google has developed its own fork of OpenSSL, named BoringSSL [44].

SSL certificate

In HTTPS, the client and server complete a TLS handshake during which the server (or sometimes both parties) presents an X.509 digital certificate that holds a public-key associated with the server's domain name. These certificates are assumed to be issued by a trusted CA. As illustrated in Figure 2.6, a X.509 certificate is often linked to a website domain and holds a temporal validity period, a public key, and a digital signature provided by a trusted CA. The web browsers check that the certificate's identity matches the requested domain name, that the certificate is within its validity period, and that the digital signature of the certificate is valid. The certificate's public key is then used by the client to share a session secret with the server in order to establish an end-to-end encrypted channel [45]. There are three types of SSL certificates according to the validation process depth:

- The Domain Validated certificate (DV) asserts that a domain name is mapped to the correct web server (IP address) through DNS. However, this type does not identify organizational information, so it should not be used for commercial websites.

- The Organization Validated certificate (OV) includes additional CA-verified information, such as an organization name and a postal address. These extra validations make OV SSL Certificates more expensive than DV certificates.

- The Extended Validation certificate (EV) uses the highest level of authentication, including diligent human validation of a site's identity and business registration details. Because of the extensive validation, EV is the most expensive among SSL certificates.

To save the cost and management of SSL certificates needed for web applications that require multiple domain names, a single X.509 certificate can be linked to multiple hosts and domains. This method is possible thanks to the X.509v3 specification's support for the `SubjectAltName` field, which allows one certificate to specify more than one host or domain name, and the support for wildcards in both the `CommonName` and `SubjectAltName` fields.

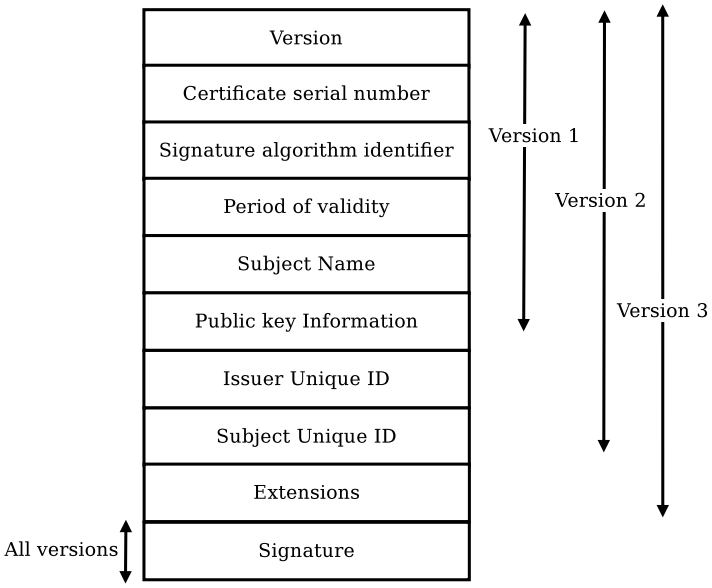


Figure 2.6: X.509 SSL certificate format [46]

2.2.5 Synthesis

In this section, we did a brief introduction about the web security and the related web environment parameters. In particular, we presented the main concept of the TLS protocol, its architecture, format, and the associated sub-protocols (Handshake, Change Cipher Specification, SSL Alert), which will be needed in the rest of the thesis. The following sections concern encrypted traffic identification and discuss the required steps and the possible methods to identify the services behind HTTPS traffic.

2.3 Identification of TLS Traffic

In this section, we review studies that aim to detect TLS traffic among other types of encrypted traffic (i.e., TLS vs. non-TLS). This is motivated by the need to recognize the high-level protocol (TLS) before dealing with sub-protocols running within. The relevant work to identify TLS traffic can be grouped into three methods: port-based, protocol structure-based and, machine learning-based.

2.3.1 Port-based method

The port-based method is a straightforward approach to identify Internet applications and protocols, since the transport layer port numbers are assigned by the Internet Assigned Numbers Authority (IANA). However, to effectively identify the TLS traffic by solely relying on port-number is impossible, since the TLS protocol is widely-used with many application layer protocols. For instance, HTTPS, FTPS and SMTPS protocols use TLS over port 443, 990, 465 respectively. Moreover, the authors in [47] observe that 8% of non-TLS traffic use standard TLS ports, while 6.8% of TLS traffic use ports not officially associated with TLS. This can be explained by misconfigured web servers or users trying to conceal their activities to avoid port-based filtering [1]. This leads to have deeper and more robust identification methods for TLS.

2.3.2 Protocol structure-based method

TLS-level DPI techniques have been used to identify TLS traffic by examining packets' payload to recognize the TLS format. More precisely, studies [47–50] inspect packets' payload for detecting TLS traffic based on the TLS Record Protocol structure. Figure 2.7 shows the content of the first five bytes of the TLS Record.

Bernaille et al. [47] want to identify TLS traffic as early as possible, so they use the standard TLS format to detect `ServerHello` packets. As illustrated in Figure 2.4, the `ServerHello` packet is a part of the TLS handshake protocol and it sets the parameters of the TLS connection (e.g., TLS version, Selected Cipher, etc.). Therefore, the presence of a valid `ServerHello` packet is a strong indication that the monitored flow[5] is a TLS one. The authors in [50] propose a "TLS Traffic Detector" to isolate pure TLS flows, which are then more deeply proceed to recognize services behind them. The TLS detector compares the first 5 bytes of packets payload (i.e., Bytes [0:4] as explained above) with the standard TLS record format to take a decision. It benefits from the idea that the TLS packet payloads should start with the same structure. So checking the first few bytes of the payload for any packet in the flow (not just on the `ServerHello` packet as in [47]) is sufficient to label a flow as TLS.

[5]The flow is a set of packets that have the same 4-tuples (source IP address, source port, destination IP address and destination port)

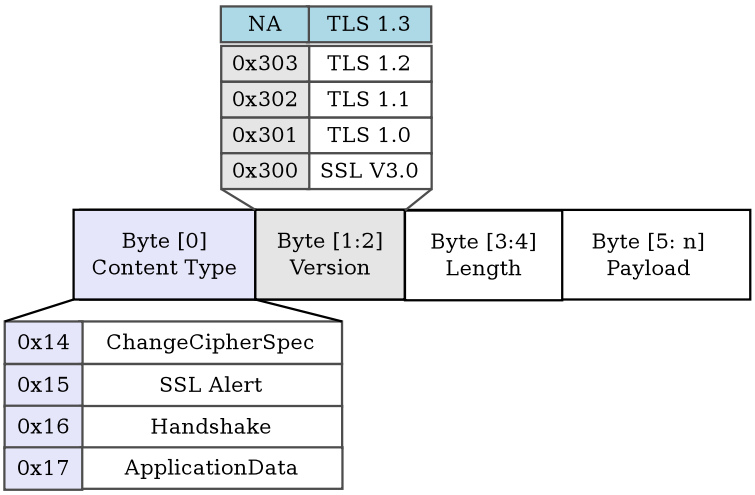


Figure 2.7: TLS record format

Finsterbusch et al. [31] evaluate the OpenDPI[6] approach that has been used for traffic identification based on DPI. OpenDPI is able to classify TLS traffic with an accuracy of 100% by using the information in the TLS Record protocol to identify TLS flows in two phases [49]. In the first phase, it detects a packet which has a valid TLS Record Protocol structure in the payload to read the content type and the TLS version. In the second phase, OpenDPI intercepts the next following packet in the reverse direction. If it has one or more TLS Record protocol structures, then OpenDPI marks this packet as TLS and it continues to check all packets in both directions.

Liu et al. [49] present a structure-based method to detect TLS traffic. The proposed method, named Double Record Protocol Structure Detection (DRPSD), is able to identify TLS traffic using the first 8 packets. The principle idea is based on the fact that to identify the TLS protocol based on the Record Protocol Structure, it is important to count how many Record Protocol Structures are detected. From their experimental results, they find that almost all TLS flows have one or more packets containing two Record Protocol Structures. Thus, they check the packet's payload, if they find double Record Protocol Structures in the payload of a packet, then the corresponding flow is identified as a TLS flow. Using their own private TLS dataset, the DRPSD approach has 99.17% identification accuracy.

2.3.3 Machine learning-based method

Due to the shortage of port-based analysis technique to classify encrypted traffic, more attention has been focused on machine learning algorithms using flow features for traffic identification. Encryption motivates the usage of this novel approach to address the limitations of legacy methods against encrypted traffic. Thus, flow

[6] `https://github.com/thomasbhatia/OpenDPI`

Table 2.3: Machine learning algorithms performance to identify TLS flows [1]

	AdaBoost	**C4.5**	**Naïve Bayes**	**RIPPER**
Accuracy	95.69%	85.13%	89.26%	82.59%
FPR TLS	4%	14%	11%	17%
FPR Non-TLS	2%	1%	1%	1%

statistics such as flow duration, packet size, and inter-arrival time are used as features to build a statistical signature for the TLS protocol [51].

The main requirements before using machine learning methods are training dataset (i.e., solved examples), relevant statistical features, machine learning algorithms and evaluation techniques. As illustrated in Figure 2.8 the learning process is divided into three phases; Training, Classification and Validation. In training, the statistical features and machine learning algorithms are trained to make prediction. The output of the training phase is a model used in the Classification phase to identify unseen data. In the Validation phase, the results of classification are validated to measure the performance of the classification model [30].

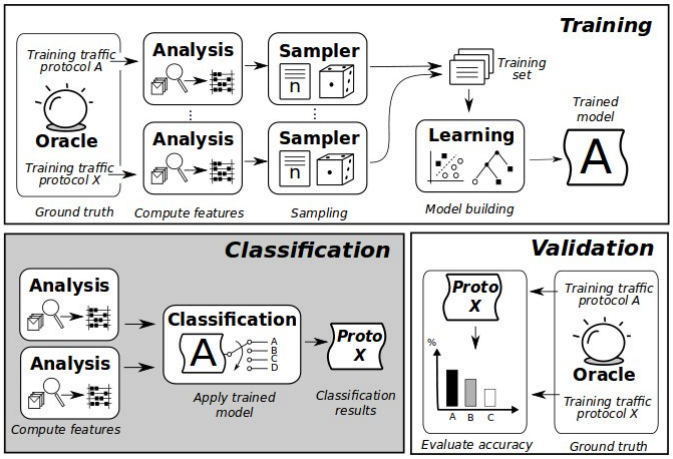


Figure 2.8: Machine learning phases [30]

The feasibility of machine learning in the context of identifying TLS traffic was performed in [1]. Four machine learning algorithms (AdaBoost, C4.5, RIPPER and Naïve Bayesian) were evaluated with 22 statistical features (e.g., Mean, Standard deviation, Max, Min, etc.) computed over the packet size and packets inter-arrival time. For their evaluation, the authors use a private dataset generated locally to assess the TLS traffic identification. As training dataset, they have 50,000 TLS flows labelled as Native-TLS, TLS-Tunneled, or Non-TLS. Table 2.3 presents accuracy and False Positive Rate (FPR) of the machine learning algorithm for identifying TLS flows. The AdaBoost algorithm achieves the highest accuracy with 95% of flows classified correctly as TLS and a 4% False Positive Rate.

Table 2.4: Identification of TLS traffic methods

Method	Features	Accuracy	Publish Year	Reference
Port-based	Port number	-	2007	[47]
Protocol structure	`ServerHello` packets	100%	2007	[47]
	First 8 Packets	99.17%	2012	[49]
	First 2 bidirectional Packets	100%	2014	[31]
	First 5 bytes	100%	2015	[50]
Machine learning (AdaBoost)	Packet size, timing	95%	2011	[1]

2.3.4 Synthesis

To summarize this section, we noticed that the identification of TLS is mainly handled by (1) using the TLS record format; (2) employing machine learning approach over the encrypted payload, as shown in Table 2.4. Based on experimental results given in the related work, we are able to recognize TLS traffic among other types of encrypted traffic with a high level of accuracy.

However, investigating protocols run inside TLS is a totally different challenge, since we need to identify different protocols types (e.g., HTTP, FTP, SMTP, etc.) through TLS encrypted connections. In the following section, we will consider that we have already detected TLS traffic for the next fine-grained level of identification. So we go deeper into the identification of the TLS traffic to detect exactly which TLS flows contain HTTP traffic.

2.4 Identification of HTTPS Traffic

Among application level protocols over TLS, the HTTP protocol is the most used one [32]. Hence, in this section, we consider the studies addressing the challenge of detecting HTTP traffic inside TLS connections (i.e., HTTPS). A direct application to HTTPS monitoring techniques is to check some properties of web traffic against a set of rules provided by an organization to allow or deny the traffic according to the network security policy. Like previously, a large set of techniques is possible from the simplest, with port-based filters, to the most advanced ones with protocol structure-based or machine learning-based techniques.

2.4.1 Port-based method

The port number 443 can be used to identify HTTPS traffic, however port 443 can also be used by malicious applications to hide their activities behind the HTTPS port to mimic a web browsing traffic [1]. Alternatively, some HTTPS web server can be configured to use a different port number, for instance the port 8080 [47].

Unfortunately, this technique does not allow fine-grained monitoring and encompass the whole usage of a protocol without distinguishing between allowed or blocked

websites and services. Given the wide usage of HTTPS today, this method alone is not sufficient to answer operational needs. Beyond web-filtering, other applications, such as P2P clients can easily bypass this type of filtering by using random ports or hiding themselves behind ports of other protocols. Many approaches have been proposed to overcome the usage of non-standard ports with HTTPS. In spite of that, the port number 443 is still widely used in the large body of literature [13, 52–57] to easily collect and build HTTPS dataset for further experiments.

2.4.2 Machine learning-based method

Haffner et al. [11] propose to extract a statistical signature from the packet payload. In the case of unencrypted traffic they extract ASCII words from the data stream as features. For HTTPS, they extract words from the handshake phase, since it is unencrypted as shown in Figure 2.4. The existence and the location of such words in the first 64-Bytes of a reassembled TCP data stream is encoded in a binary vector and used as input for different machine learning algorithms (Naïve Bayes, AdaBoost and Maximum Entropy). The evaluation over dataset from ISP shows that AdaBoost identifies HTTPS traffic with 99.2% accuracy.

Wright et al. [52] demonstrate how web applications behaviour can still be used as a signature to identify the accessed website, even if its traffic is transmitted via HTTPS flows. The authors use the fact that some information remains intact after encryption like packet size, timing, and direction, to identify the common application protocols by applying k-nearest neighbors algorithm (KNN) and Hidden Markov Model (HMM). The KNN algorithm detects HTTPS flows with 100% accuracy and the HMM algorithm performs 88% accuracy.

The authors in [47, 51] share the concept of identifying HTTPS in two steps. In the first step, the TLS traffic is detected based on the protocol-format as discussed in Section 2.3.2, while, in the second step, the HTTP traffic in TLS channels is recognized by applying machine learning methods. In [58], the authors use the size of the first five packets of a TCP connection to identify HTTPS applications with 81.8% accuracy rate. The performance of their classifier has been improved (up to 85%) in [47] by adding a pre-processing phase, where they first detect the TLS traffic based on protocol-format and then identify the HTTP traffic within TLS. Sun et al. [51] propose a hybrid solution, which first detects TLS protocol by inspecting the TLS protocol-format, then applies a machine-learning algorithm (Naïve Bayes) to determine application protocols running over TLS connections. The Naïve Bayes algorithm is used with 8 statistical features; Mean, Maximum, Minimum of packet length, and Mean, Maximum, Minimum of Inter-Arrival time, flow duration and number of packets. Using a private dataset, results show the ability to recognize over 99% of TLS traffic and to detect the HTTPS traffic with 93.13% accuracy.

2.4.3 Synthesis

The related work in the identification of HTTP within the TLS protocol, as illustrated in Table 2.5, has used different methods with an acceptable level of accuracy.

Table 2.5: Identification of HTTPS traffic methods

Method	Features	Accuracy	Pub. Year	Reference
Port-based	Port 443	100%*	2006 2016	[52–54] [13, 55] [56, 57]
Machine learning (KNN, HMM)	Packet size, Timing, Direction	100%, 88% (KNN, HMM)	2006	[52]
Machine learning (AdaBoost)	Keywords	99.2%	2005	[11]
Machine learning (Gaussian Mixture)	TLS-Format, First 5 packets size	85%	2007	[47]
Machine learning (Naïve Bayes)	TLS-Format, Packets size, Timing, Flow duration, Packets number	93.13%	2010	[51]

* If user not malicious (i.e., alters port number)

However, each method claims its own private HTTPS dataset. Due to privacy and security issues, there is no representative full public HTTPS dataset, which prevents others from having a strong and fair comparison of their respective identification accuracy. Levillain et al. [59] analyse some of the existing public HTTPS datasets published between 2010 and 2015. Their investigation shows that some datasets contain only SSL certificates information, while in the other ones the whole TLS answers were truncated. The situation will remain ambiguous in the absence of a reference dataset for all. That leads to another research question about reproducible research and dataset construction [1, 32, 59]. We should also question the representativity of the dataset, since HTTPS nowadays is a multi-purpose protocol (i.e., it can deliver video, music, games, etc.), but it was not the case a couple of years ago. In this work, our challenge is to precisely name the service that generates a given HTTPS flow. Indeed, HTTPS is considered as a single class in the aforementioned papers, even if web applications can provide very different kinds of services. The next section delves more into HTTPS traffic itself, to explore the current methods to name the specific web service that generate a given HTTPS flow.

2.5 Identification of HTTPS Services

The increased complexity of web applications provides the ability to deliver a wide set of services such as online-storage, content providers, social media, video, etc., thanks to Web 2.0, all transmitted via HTTPS connections [50]. The identification of HTTPS services is a serious challenge, since most of the legacy techniques lose their power when facing encryption. Here, we present a large set of solutions that show the higher difficulty of the problem. The possible techniques can be based

on website fingerprinting or machine learning approaches, on values extracted from DNS, IP address, SNI, or SSL certificate, on an HTTPS proxy server or on the acquisition of TLS encryption keys. All these approaches are discussed in the rest of this section.

2.5.1 Website fingerprinting-based method

Identifying the accessed webpages over secure connections is well-known as website fingerprinting, which has been presented in many relevant works [24, 25, 60–62]. Cheng et al. [63] propose one of the earliest method to distinguish the pages visited by users over TLS connections by inspecting the TCP/IP header, which contains payload size and other information. Their technique is based on calculating the size of a downloaded page, which is often unique among all files in a given site. Liberator et al. [24] propose two systems to infer the source of encrypted HTTP traffic (not HTTPS) covered by a SSH-tunnel. The first one is based on the Naïve Bayes classifier and the second one on Jaccard's coefficient. Both systems rely on the packet length while discarding timing information. Herrmann et al. [25] present a multinomial Naïve Bayes classifier based on the normalised frequency distribution of IP packet size for HTTP websites accessed over SSH-tunnel.

Panchenko et al. [26] focus only on Tor, which is out of the scope of this thesis. Panchenko also introduces the concepts of *Open-world* and *Close-world* experiments. In *Close-world*, the training and testing of the classifier takes place over a predefined set of websites, while an *Open-world* indicates the usage of both interested and unknown websites. Miller et al. [27] propose a method to identify the accessed page among 500 pages hosted by the same HTTPS website, based on clustering techniques to classify patterns in traffic. Their results show the possibility to recognize individual pages from the same website accessed over HTTPS with 89% accuracy. They successfully detect the access to the home-page and to many internal-pages from a given website but at the cost of a specific learning at a single website page-level, while more effort is needed to identity embedded services in web pages.

All the aforementioned studies, as compared in [27], are intended to determine the home-page or internal-pages from a website. This is too fine-grained, as it works at the page-level, specially in the case of naming services that offer contents to other web pages. For example, Facebook uses Akamai Content Delivery Network (CDN) to obtain content. So Facebook's contents (photos, videos and profiles information, etc.) are retrieved from "akamaihd.net" and not from "facebook.com". Thus, from the website fingerprinting view, the diversity of the services on the third-party providers are neglected because of the page centric view.

2.5.2 Machine learning-based method

In [23], the authors develop a passive approach for webmail HTTPS traffic identification in order to understand the shift in usage trend regarding mail traffic evolution. Three novel features are proposed (1) service proximity: the existence of a POP, IMAP or SMTP server within a domain is a strong indication that a mail server

exists; (2) activity profiles: clients access their e-mail frequently in a scheduled manner, so it is possible to build daily and weekly profiles; (3) periodicity: the usage of application timers like AJAX technology to periodically (e.g., every 5 minutes) check for new messages creates high frequency time patterns and indicates how the email service is running. These features are used with the Support Vector Machine (SVM) algorithm to differentiate between mail and non-mail services within HTTPS flows. The evaluation over a dataset from ISP shows the ability to identify HTTPS mail related traffic with 93.2% accuracy.

Chen et al. [64] use the traffic pattern of the AutoComplete function, which populates a list of suggested content according to each letter a user enters, such as in Google and Yahoo search engines. This small amount of input data causes state transitions in a web application which generate traffic that can be used to enumerate all possible inputs to match the triggered traffic pattern. Based on real scenarios, they show how such a method can be applied to leak out sensitive information (i.e. the searched keywords) from top online web applications despite the usage of HTTPS. V. Berg. [65] develops a tool that uses encrypted traffic patterns to identify user activities over Google Maps that is now accessed over HTTPS. The tool collects satellite map tiles and builds a database of the different image sizes correlated with their (x,y,z)-triplets coordinates. To identify the accessed region over Google Maps, the tool maps the size of images in HTTPS flow to (x,y,z)-triplets and then clusters the results into a specific region. As a proof of concept, the tool's dataset has been configured with city profiles, where it can correctly detect the transition between such cities.

Dubin et al. [66] intend to determine the name of a video accessed over HTTPS websites such as YouTube. The proposed approach contains three modules: the first one detects a Youtube video based on SNI (e.g., googlevideos.com); while the second one combines several YouTube packets into a peak, defined as a section of traffic where there is a silence before and after. They extract the total number of bits from a peak as the Bit-Per-Peak (BPP) feature. The third module passes the feature to the SVN machine learning classification algorithm. Over a dataset of 10000 YouTube video streams of 100 video titles, they can identify the title of the accessed video with 95 % accuracy.

Korczynski et al. [67] use the TLS handshake messages interaction to build a stochastic fingerprints based on Markov chains for identifying services in HTTPS traffic. The Markov chain states model a sequence of the TLS handshake message types appearing in a single direction flow of a given service from a server to a client. The authors test the proposed approach on 12 HTTPS services such as Twitter, Dropbox over private datasets. Their results show that they can detect Twitter, Dropbox, Gadu-Gadu with 91.13%, 76.36% and 86.26% accuracy respectively.

2.5.3 DNS and IP address-based methods

Prior to any HTTP request is sent a DNS query to resolve the IP address of a host. DNS queries can be used to monitor and filter blacklisted domain addresses. For example, recently, the Turkish government ordered ISPs to block a famous

social networking website. This filtering performed by DNS servers was quickly bypassed by using alternative DNS servers, such as Google DNS servers. Therefore, the identification based on DNS is ineffective if the client behind a monitoring system uses directly the IP address or a local resolution. In this case, no DNS query is sent.

The identification based on the IP address of an HTTPS service is another radical method. However, blocking large websites via IP is very troublesome, since the IP addresses being used for these websites can be very different depending on where the accessing host is located, or whether load balancing is being used across multiple addresses. The IP address ranges to block can be quite large and scattered, not to mention that they are prone to change in time. Even more, nowadays, the "Virtual Hosting" allows a single machine to serve many websites and many web servers are often associated to the same IP address.

These tendencies make monitoring based on IP addresses an inefficient technique to block a specific service or website. For example, blocking a service like Gmail from Google services, while allowing users to access other services like Search or Maps is impossible based only on IP addresses: the whole range of Google's services share the same address space that must be either fully blocked or allowed. Above all, websites' IP addresses can be changed with little interruption in service by updating the DNS record. That makes tracking and updating IP blacklists difficult.

2.5.4 SNI-based method

One recently practical technique to identify HTTPS traffic is named SNI-based monitoring and uses the Server Name Indication (SNI), a field of the TLS handshake. The SNI is a clear string value from the TLS `ClientHello` message that provides a convenient way to know what service is accessed by a new HTTPS connection. SNI-based monitoring has been integrated in many firewall solutions. For example, the Clavister[7] web content filtering system supports both HTTP and HTTPS traffic, where HTTPS filtering is performed based on the SNI value or on the `CommonName` field in the SSL certificate. The Sphirewall[8] firewall system has recently included this SNI-based method to filter HTTPS traffic with the 0.9.9.5 release (July 2013). It can determine the name of the remote host in a HTTPS web request by looking at the SNI value in the initial TLS handshake. Sophos Unified Threat Management (UTM)[9] is another example of a hardware and software network firewall system that recently (in UTM 9.2, released in September 2013) included the SNI-based monitoring feature.

From another perspective, the authors in [68] report that some commercial traffic shaper devices use the SNI extension to detect high bandwidth consuming web applications like Youtube and Netflix. While it has been proven in [69] that the T-Mobile operator in the USA uses the SNI value to identify a set of services that are free of charge for their customers. As mentioned previously, Dubin et al. [66] perform a first filter to get only network flows belonging to Youtube by looking at

[7] `https://www.clavister.com/`
[8] `http://www.sphirewall.net`
[9] `http://www.sophos.com/en-us.aspx`

the SNI extension. For example, if the "googlevideos.com" string is found in the SNI extension, the flow is tagged as Youtube and is processed further to extract features to identify the title of the accessed video using a machine learning method as explained before in Section 2.5.2.

Bortolameotti et al. [13] use the SNI extension in conjunction with SSL certificate information to detect connections toward malicious websites. They examine the claimed SNI value using: (1) Levenshtein distance between the SNI value and top 100 most visited websites; (2) the structure of the "server-name" string in the SNI extension; (3) the format of the "server-name" string, which is a DNS hostname format. Based on their experiments, they are able to detect malicious connections that present SNI with weird values or strings, other than DNS resolvable names, which may be used as a mean for messaging, perhaps used by criminals as a command and control channel.

2.5.5 SSL certificate-based method

Originally, SSL certificates were only used to verify the identities of servers and clients. However, they are also employed to recognize the accessed service over HTTPS flows. Certificate authorities guarantee the identity of a website by digitally signing the website's leaf certificate using a browser-trusted signing certificate. Most recent browsers and operating systems integrate trusted signing certificates known as "Root Certificates" [45]. The TLS protocol enables client software to build secure communications terminated by server/client holding the private key authenticated by the certificate. The most critical attribute, that all HTTPS server certificates hold, is the domain name. This attribute is located in the `CommonName` field under Subject. One or more domains are often included in the `SubjectAlternativeName` field in an X.509 extension [70]. Based on X.509 extension the alternative name field gives the ability to use a single certificate for multiples domains.

Kim et al. [50] use the certificate public information to build SSL/TLS Identification Method (SSIM) to name the services behind HTTPS traffic. The proposed method consists of three modules: (1) a TLS traffic detector module isolates pure TLS traffic before beginning the service identification; (2) the service signature module extracts certificate authority information, Server IP and Session ID from a SSL certificate; (3) the session ID-IP-based service identifier module recognizes non-identified flows from the previous modules by finding a relationship between a server IP and a session ID. Based on their experiments, they can classify 95% of TLS traffic belonging to Google, Facebook and Kakaotalk with about 90% accuracy for the corresponding services. As discussed in the previous subsection, the authors in [13] use also the SSL certificate information (with SNI extension) to detect malicious connections. By investigating the claimed SSL certificate validity, release dates and the content of `SubjectAlternativeName`, they can detect several malicious connections with blacklisted IPs using an expired certificate of Amazon.com.

HTTPS services identification can be based on the `CommonName` field in the SSL server certificate. However, this technique is inefficient as many companies share the same certificate across different services and domain names. For example, blocking

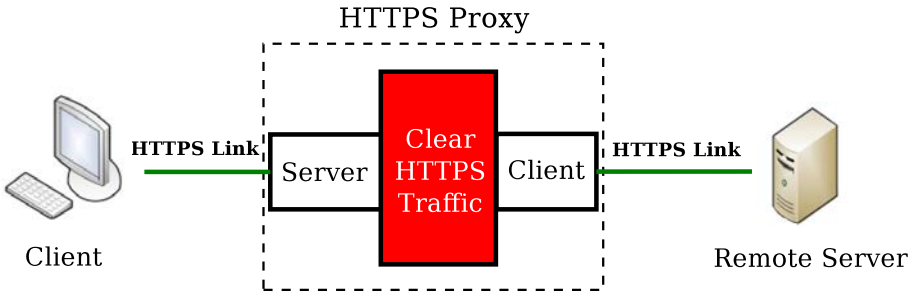


Figure 2.9: HTTPS proxy server

Youtube.com from its SSL certificate would imply to block all the other domain names covered by this certificate and given in `SubjectAltName` (i.e., all the other Google services). In this case, it is impossible by simply looking at the certificate among different Google services to have a fine-grained monitoring between services.

2.5.6 HTTPS proxy-based method

All the above monitoring HTTPS techniques exploit meta-data, but not what encrypted packets actually contain. DPI techniques inspect the content of packets, looking for keywords or regular expressions to identify what the packets hold. However, due to encryption, DPI totally loses its effectiveness [71]. This leads to first decrypt the traffic thanks to an HTTPS proxy, before applying any DPI. The proxy server is a server acting as a MitM. To be able to access encrypted HTTPS traffic between a client and a remote server, it pretends to be the intended remote server and then, it establishes a secure connection to the real server. As shown in Figure 2.9, when a client connects to the remote server via an HTTPS proxy, the client in place connects to the proxy server, which plays the role of a destination server by providing its own SSL certificate. Then, the proxy establishes another secure connection with the real remote server. Thanks to this method, all encrypted web traffic is open to the proxy in clear at the expense of users' privacy.

Existing commercial solutions such as Forefront Threat Management Gateway (TMG) 2010 uses the HTTPS proxy method for HTTPS inspection, which acts as a trusted MitM instead of just processing HTTPS connection blindly [72]. Also, the FireEye[10] product applies the proxy model to provide visibility into untrusted TLS traffic. The product is designed to intercept and forward all desired network traffic for decrypting, examining and then re-encrypting TLS sessions again. FireEye argues that this method responds to the growing number of cyber criminals that use TLS as a cover to get inside organizations and keep being undetected [20].

[10]https://www2.fireeye.com

2.5.7 TLS encryption key acquisition-based method

This last unconventional monitoring solution may be encountered at the government level. There are at least two methods for acquiring the decryption keys, the Key-Recovery mechanism and the cracking of encryption algorithms. In [73] the authors describe the Key-Recovery mechanism or "Key escrow", where all encryption keys are stored in a trusted third party, such as a government, or designated private entities. The third party has the right to access keys for authorized law enforcement purpose. As a result, a government may limit access to HTTPS websites that refuse to share their TLS keys with the escrow system [74]. Exploiting encryption algorithms is different from the preceding ones, as it needs high computation power to be able to crack the encryption. A method for cracking is a flaw in the mathematical algorithm used to encrypt data, such as the factorization problem of widely used public-key cryptosystems. For instance, RSA 768-bit can be broken with a state-of-the-art algorithm and a high computation power [75]. Adrian et al. [76] evaluate the security of Diffie-Hellman key exchange, where they find that 82% of vulnerable servers use a single 512-bit group, which makes it possible to compromise connections of 7% of Alexa top million HTTPS websites.

2.6 Conclusion

HTTPS is quickly becoming the predominant application protocol on the Internet. It answers to the need of Internet users to benefit from security and privacy when accessing the web. But the increasing amount of HTTPS traffic comes with challenges related to its management to guarantee basic network properties such as security, QoS, reliability, etc. The encryption undermines the effectiveness of standard monitoring techniques and makes it difficult for ISPs and network administrators to properly identify services behind HTTPS traffic and to properly apply network management operations.

2.6.1 General analysis of related work

This chapter provides a focused view of HTTPS traffic identification methods, starting from the identification of the lower-level TLS protocol to the precise identification of HTTPS services. We have found that efficient methods exploiting the standard structures of the TLS protocol are able to identify TLS traffic among other types with a high level of accuracy. The identification of HTTP with TLS can also be recognized with acceptable level of accuracy, but the real challenge is the identification of services inside HTTPS traffic.

Many recent approaches, as summarized in Table 2.6, intend to identify HTTPS services based on the plain-text information that appears in the TLS handshake phase, or based on the statistical signature (e.g., machine learning-based methods) of HTTPS web services but they are often dedicated to a specific application (Webmail services, Google Maps, Youtube). While the website fingerprinting technique is not adapted to identify particular services, simpler practical solutions have either

Table 2.6: Identification of services in HTTPS traffic

Method	**Features**	**Identification Level**	**Accuracy**	**Publish Year**	**Reference**
Website fingerprinting	Packet size and Order	Internal Pages	96%	1998	[63]
	Packet size and Direction		89%	2014	[27]
Machine learning (SVM)	Service proximity, Activity profiles, Session duration, Periodicity	Email Services	94.8%	2010	[23]
	Bit-Per-Peak	Video Titles	95%	2016	[66]
Markov Chains	TLS messages order	Services	66%**	2014	[67]
State machine	AutoComplete function	Search Keywords	NA	2010	[64]
Statistical measures	Images size	Google Maps	NA	2011	[65]
SSL certificate	SSL certificate information	Services	90%	2015	[50]
			100%*	2015	[13]
SNI-based	SNI extension information	Malicious Connections	100%*	2015	[13]
		Services	100%*	2015	[68]
		Video Services	100%*	2016	[69]
		Youtube video	100%*	2016	[66]

* If user not malicious (i.e., alters SNI extension value)
** The average of True Positive Rate for 10 services over 3 datasets

privacy or reliability troubles. The HTTPS proxy is an efficient solution, but it breaks privacy as it decrypts the traffic in the middle which is not acceptable for us. Other approaches like IP, DNS, SSL certificates do not work well. The monitoring based on IP addresses faces a real challenge with virtual hosting, where a single IP address serves many hosted websites. The DNS-based method can be bypassed if the client behind a monitoring system uses directly the server IP address. The monitoring based on SSL certificates also has a problem with shared certificates, where a single certificate is used by several services.

Yet, some approaches provide promising results like SNI-based and machine learning-based methods. We have noticed that the reliability of the SNI-based solution to monitor HTTPS services has never been evaluated. For machine leaning-based techniques, there are different methods for HTTPS and TLS identification, but each approach validates its result using a private dataset. This prevents others from having a strong and fair comparison of their respective identification accuracy. The situation will remain ambiguous in the absence of a reference dataset for all.

According to the aforementioned summary, the objective of this thesis is to provide a reliable and privacy preserving identification of services in HTTPS traffic. Thus, the next following chapters aim to define new solutions for HTTPS services identification. First, we deeply assess the SNI-based method in Chapter 3. Second, we motivate the need for a more robust method to identify HTTPS services based on traffic pattern, and not on header information like SNI. Thus, we propose a machine leaning-based approach with dedicated features and a novel multi-level classification technique to identify HTTPS at service-level with high accuracy. Third, we tackle the real-time identification of HTTPS services by refining our approach.

2.6.2 Ethical aspects of HTTPS monitoring

In Section 2.5, we have noticed that there is a very efficient method to monitor and control HTTPS traffic based on HTTPS proxy. Enterprise owners may have certainly good arguments for monitoring and filtering access to their network for security, productivity or responsibility reasons [17]. However, this is not sufficient to legitimate the exposition of employees' private data with an HTTPS proxy. As sensitive data is mostly transmitted via HTTPS services, it is hardly acceptable to trust a third-party to screen this information and break users' privacy. On the opposite side, the IETF has issued the RFC7258 [77] about the monitoring activities legality. It defines that the surveillance by collecting protocol meta-data and application content, traffic analysis (e.g., correlation, timing or measuring packet sizes) and subverting the cryptographic keys is considered as Pervasive Monitoring (PM), what the authors consider as an attack against the privacy of Internet users, and it should be mitigated. From another viewpoint, the authors in [78], claim that large-scale Internet traffic monitoring is ineffective as it is only able to identify trivial crimes, but cannot recognize professional criminals. They also arise another important question: how to guarantee that an administration in power will never abuse the intercepted information to intimidate its opponents. The authors conclude that if online monitoring may fix some problems, it can create even more serious ones.

In conclusion, HTTPS monitoring is a sensitive and complex question. We think that, while challenging, the monitoring of HTTPS services without decryption can provide a viable compromise between the knowledge needed for network management and users' privacy. The research community should focus on proposing new identification techniques offering both security and privacy.

Chapter 3

Evaluation and Improvement of SNI-based HTTPS Monitoring

Contents

3.1 Introduction

The security of data and transactions is a key element to run critical services over the Internet. This wide spread of networks and services makes an increased need for efficient monitoring methods to ensure that network systems and users work within a predefined policy, to avoid some malicious web applications or inappropriate content [79], and to guarantee network performances. That is why enterprise networks statistics [17] show that 66% of companies monitor their employees' network activity. However, it is very hard for an IT organization to properly monitor and manage TLS-encrypted web traffic which may cause security problems [80]. For example, an enterprise's sensitive information may be leaked out over daily used web services or social media that run over encrypted connections, and such leaks are hardly detected [81].

This chapter explores a novel and practical technique to identify HTTPS traffic, named SNI-based monitoring that uses Server Name Indication (SNI), a standardized extension of the TLS handshake. The SNI extension's content provides a direct way to know what service is accessed by a new HTTPS connection. As explained in the previous chapter (Section 2.5.4), many firewall systems and some network operators now use the SNI extension to precisely identify and manage HTTPS traffic.

The rest of this chapter is organized as follows. Section 3.2 provides an overview of the SNI extension and how it is used for HTTPS monitoring and filtering. Section 3.3 evaluates the reliability of the SNI-based filtering and explains our strategies to bypass it. Section 3.4 describes the implementation and evaluation of our tool "Escape" tested against existing firewalls and also presents our investigation of HTTPS servers accessible with a fake SNI. Our solution to improve the SNI-based monitoring is described in Section 3.5. Section 3.6 evaluates the proposed approach and related overhead. Finally, Section 3.7 summarizes and concludes this chapter.

3.2 SNI Extension Overview

In this section, we overview the original use of the SNI extension for virtual hosting, and also the alternative uses like HTTPS monitoring and traffic shaping.

3.2.1 Standard use

Virtual hosting is commonly used to make multiple websites hosted on a single server machine. However, a new challenge has appeared to make it compatible with TLS. Originally Virtual hosting can be either "IP-based" or "name-based". In the first type, a website is mapped to a unique IP address of the server. This is compatible with HTTPS since the server can select the correct SSL certificate based on the IP address. But it is expensive to reserve an IP address for each website, and the server needs several network interfaces to do so. Therefore, in a name-based method, all hosted websites share the same IP address and the server must identify the website based on the HTTP header, by reading the website's URL in the GET request. The problem with this named-based identification when used with HTTPS

is that the HTTP header is sent encrypted once the TLS connection is already established, and thus cannot help to identify the proper certificate. Indeed, since each virtual website has its own SSL certificate, the problem is to select and expose the certificate corresponding to the intended website. Thus, an extension of the protocol was designed to enable TLS to effectively operate with virtual web hosting and overcome this limitation [82].

The SNI extension allows a client to specify the hostname it is attempting to connect to when the TLS negotiation starts, as shown in Figure 3.1. The hostname contains the fully qualified DNS hostname of the server. This makes a server able to have multiple certificates on a single IP address and to ultimately host multiple HTTPS sites using different certificates on the same IP address [45]. So, with the SNI extension, a server is able to present the correct mapping between virtual hosts and the respective certificates [83], thus to provide virtual hosting of HTTPS websites.

In RFC6066 [84], the SNI extension behaviour is explained as follows: in order to provide any of the server names, clients may include an extension of type SNI in the (extended) `ClientHello` message as shown in Figure 3.1. The "extension-data" field of the SNI shall contain `ServerNameList`, where it must at most contain one server-name. A server that receives a `ClientHello` containing the SNI extension uses this information to select an appropriate certificate to be returned to the client. Then, the server answers with an extended `ServerHello`, where the SNI extension should be empty. When the server decides whether or not to accept a request to resume a TLS session, the content of a SNI extension may be used in the look up of the session in the server's session cache. Thus, a client should use the same SNI extension as in the full handshake to request the session resumption, otherwise the server refuses to resume the session and requires the client to start a new full handshake [84]. This behaviour is used to detect if a given server supports SNI extension or not, as we explain later in this chapter.

3.2.2 Deployment and support

Introduced in 2003, the SNI extension has been widely used by browsers, servers and CAs. Most current server software, such as Apache or Microsoft IIS 8, already support the SNI extension [83]. On the browser side, all browsers released in the past 5 years already support the SNI extension, such as Mozilla Firefox, Internet Explorer, Safari or Google Chrome. This means that there is a global trend for supporting the SNI extension and all new versions of web browsers and client-side applications support it. The only limitation concerns older hosts based on Windows-XP (and below) whose old TLS implementation does not consider the SNI extension. However, those hosts should not be allowed to access the web in a security-sensitive context because they miss security updates (since 04/2014 concerning WindowsXP) and are consequently very vulnerable. Moreover, their number is always decreasing and accounts (in September 2016) for only 2.76% of hosts[11].

[11] `http://www.w3counter.com/globalstats.php`

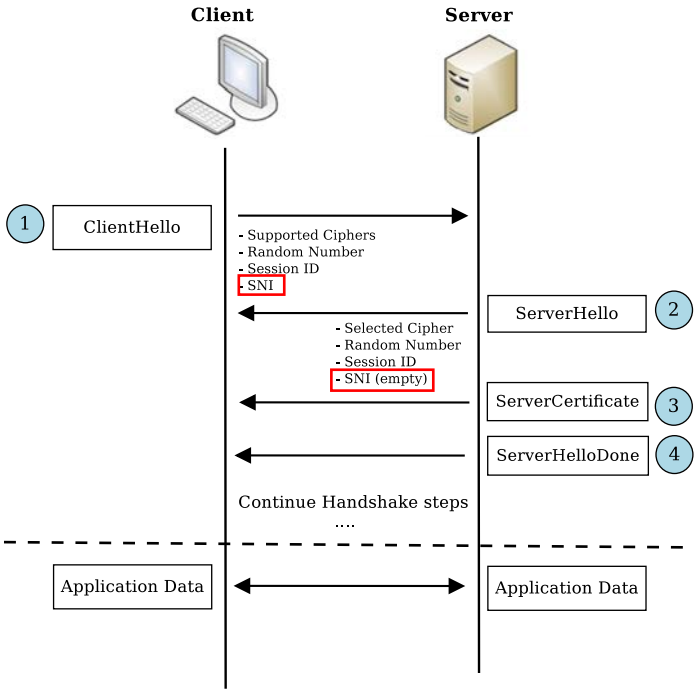


Figure 3.1: SNI extension of the TLS handshake protocol

3.2.3 Alternative uses

Due to the simplicity of extracting the SNI extension content from HTTPS sessions, it has been used for different purposes, not only for security reasons but also for performance by enabling traffic shaping for HTTPS services.

SNI-based HTTPS monitoring

Monitoring based on SNI extension relies on checking the "server-name" value in the extension. This value provides the DNS name of the HTTPS website to be accessed. It is a convenient way (i.e., meaningful string) to know what service is accessed, also the "server-name" value can be compared against a list (i.e., blacklist or whitelist) to enforce HTTPS filtering. As shown in Figure 3.2, the firewall inspects the SNI extension within the `ClientHello` message to check if the "server-name" is in a black/white list or not, and according to the response, the firewall resets the connection or allows the `ClientHello` message to pass through toward the destination server, and further complete the TLS handshake.

Since the content of the SNI extension is a simply extracted value, this is a lightweight process, yet precise to the "service-level", compared to statistical and website fingerprinting techniques to identify the source of HTTPS traffic. It also preserves the privacy of users whose encrypted payload is left untouched.

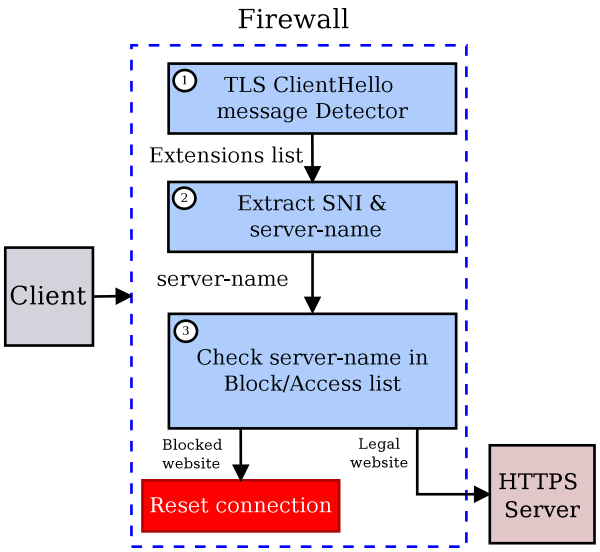


Figure 3.2: SNI monitoring/filtering approach

Traffic shaping

The bandwidth-hungry applications (e.g., YouTube, Netflix) create a real challenge for resource-constrained networks like mobile networks. Thus, mobile operators manage scarce network resources using various techniques such as traffic shaping, in which a given flow is delayed or buffered to prioritize the flow of other services. The main benefit of traffic shaping is to give the priority to critical services (e.g., VoIP) over less-critical traffic, like web browsing. While from another perspective, traffic shapers might rate-limit high-volume flows to avoid network congestion and also to control the impact of bandwidth-consuming applications from affecting other services [85].

In the context of the alternative SNI extension uses, Molvai et al. [68] found that some commercial traffic shapping devices detect applications via the SNI content value, and therefore they trigger the traffic shaping process against a given application. Moreover, as reported in [69] T-Mobile (a mobile operator in the USA) has recently announced the "BingeOn"[12] service, that provides their customers with zero-rates video streams from a large number of partner sites like Youtube, Hulu, Amazon. The principle of zero-rating is that ISPs do not charge users for the traffic related to certain services, often because those services agree to use limited bandwidth resources (limited to 1.5 Mbps). The interesting point is that BingeOn detects when an HTTPS flow concerns a zero-rate service by applying regular expressions on the content of the SNI extension [69] to match pre-defined names.

[12] `http://www.t-mobile.com/offer/binge-on-streaming-video.html`

3.3 Strategies for Bypassing SNI-based Monitoring

The wide support and deployment of the SNI extension makes it a prime choice to identify and manage HTTPS traffic nowadays. However, we will show in this section that SNI-based filtering is not reliable and we will describe two strategies exploiting SNI weaknesses regarding (1) backward compatibility and (2) multiple services using a single certificate. These weaknesses can be used for circumventing middleboxes relying on the SNI extension to monitor and manage HTTPS traffic. In the following sections, we will take the example of a firewall enforcing a blacklist of websites.

3.3.1 Exploiting backward compatibility

According to the related RFC6066 [84], the SNI extension is designed to be backward compatible. This means that if a server does not recognize the SNI extension or the "server-name" value inside, it should still continue the handshake. The first bypassing strategy consists in sending a `ClientHello` message without the SNI extension present or with an alternative "server-name" value during the TLS handshake. In both cases, the firewall will not find the blacklisted name and the `ClientHello` message will pass toward the remote server.

In order to write a proof of concept, a full control over the SNI extension's values is needed. We have customized an HTTPS Java client provided with the Oracle 8 JDK and we added the ability to alter the SNI extension through the TLS socket configuration. We have tested the two following scenarios exploiting the backward compatibility, both were able to bypass the SNI-based filtering.

First Scenario: removing the SNI extension

We initialize the TLS handshake with a `ClientHello` message without SNI. Once the connection is established, an HTTP host header is sent with the host value of the blocked website. Since the HTTP host header is sent after establishing the secure connection, it will be encrypted and it is impossible for the firewall to identify the requested server past the TLS handshake.

Second Scenario: inserting an alternative "server-name" value

It consists in changing the "server-name" field of the SNI extension with an alternate value that is not related with the real name of the web server. For instance, setting the "server-name" value with "www.g00gle.com", "abcde.net", etc. in order to escape the blacklist. We summarize the technique in the following steps, assume we try to access "www.facebook.com":

- Create TLS Java socket with domain name and port 443:
  `TARGET_HTTPS_SERVER=facebook.com`
  `TARGET_HTTPS_PORT=443`

- Configure the TLS socket with a customized SNI object where "server-name" value is faked, then use this socket to connect to a blocked website:
  `server-name=bypassf@ceb00k.com`
- Send (encrypted) HTTP host header with host field holding the right address of the blocked website:
  `GET/HTTP/1.1/r/n`
  `HOST:facebook.com:443`

3.3.2 Exploiting shared server certificate

One property of the certificate standard X.509 is an `AlternativeName` field, which can hold a set of domain names using the same certificate for the TLS handshake. Basically, it allows the service provider to use one server certificate for a set of services. In our context, this can be used to get access to a banned website by sending a SNI value for non-banned websites sharing the same server certificate.

From the firewall side, the connection seems to be legal, but in fact, after completing the handshake, the traffic to the banned website will be totally encrypted and smoothly pass the firewall. Our main example is Google services, since there are many Google services sharing the same Google server's certificate like YouTube and Google Maps. Thus, let us assume YouTube is restricted by an SNI-based filtering solution but Google Maps is not, the bypassing technique works as follows:

- TLS socket with domain name and port 443:
  `TARGET_HTTPS_SERVER= maps.google.com`
  `TARGET_HTTPS_PORT=443`;
- Create SNI object with another service sharing the same certificate:
  `server_name = maps.google.com`
- Get access to Youtube by sending HTTP host header:
  `GET/HTTP/1.1/r/n`
  `HOST:www.youtube.com:443`

As shown in Figure 3.3, we perform the handshake using Google Maps and receive a server certificate for Maps, then we send HTTP host header for "www.youtube.com", and we get all YouTube traffic encrypted with Google Maps server certificate, both websites sharing the same infrastructure. Once the traffic is encrypted, the firewall cannot detect the YouTube traffic based on TLS information.

At the end of this section, we have envisioned and validated two means to bypass SNI-based filtering. Users can easily access prohibited resources by passing through the firewall and without even the need of a third party to circumvent the filtering system. In the next section, we present our large-scale evaluation of these weaknesses and the implementation of a more advanced and convenient tool directly into a web browser.

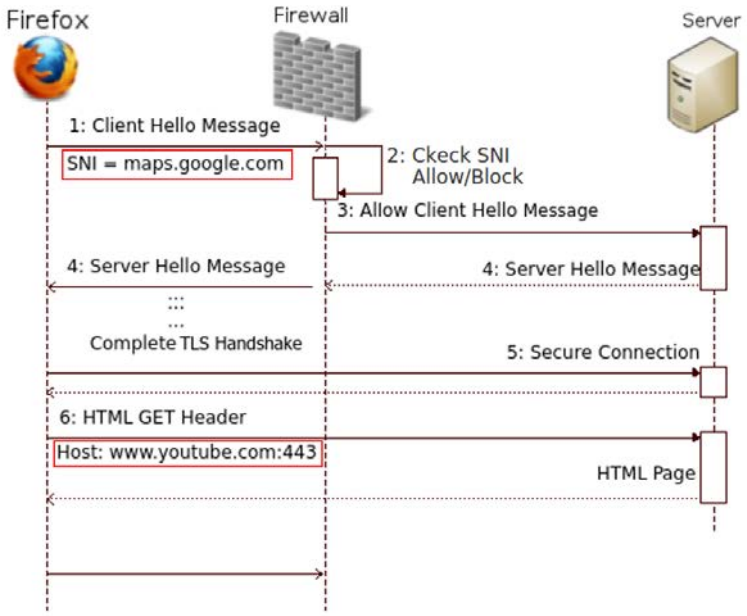


Figure 3.3: Shared SSL certificate scenario to access a blocked service with a legitimate one

3.4 Implementation and Evaluation of Bypassing Strategies

In this section, we first present our Firefox add-on as a proof-of-concept of the SNI-based filtering weaknesses. We then evaluate the support of the SNI extension by many popular web services. Finally we assess the performance of the bypassing strategy.

3.4.1 Implementation of a web browser add-on

As a direct application of the aforementioned results, we present an add-on named Escape[13] that we have developed for the Firefox web browser. The choice of a web browser add-on is motivated by the strong relation between our work and web browsing, the ease of use for Internet users and the support of several architectures and operating systems through the web browser. The core of the add-on is based on another extension, named Convergence[14] that intercepts TLS connections to perform a supplementary check on the certificates. It works as a local MitM inside the web browser, that hacks the web browser requests regarding TLS and creates its own TLS connection with the intended HTTPS server on one side, and with the web browser on the other side.

[13] `http://madynes.loria.fr/Research/Software`
[14] `http://convergence.io/`

Figure 3.4 describes the inner process of Escape as follows:

1. When the web browser (Firefox) connects to HTTPS websites, Escape intercepts the TLS handshake and extracts the SNI extension from the `ClientHello` message (Steps 1-2).
2. Escape creates a TCP socket and completes the TLS handshake with the remote server by using a new SNI value which is either randomly generated or manually configured (Steps 3-6).
3. Escape pursues the initial TLS connection issued from the client and presents its own certificate to the web browser and performs a new TLS handshake (Step 7)
4. The web browser and the web server communicate through the established secured TLS connections through Escape (Steps 8-9).

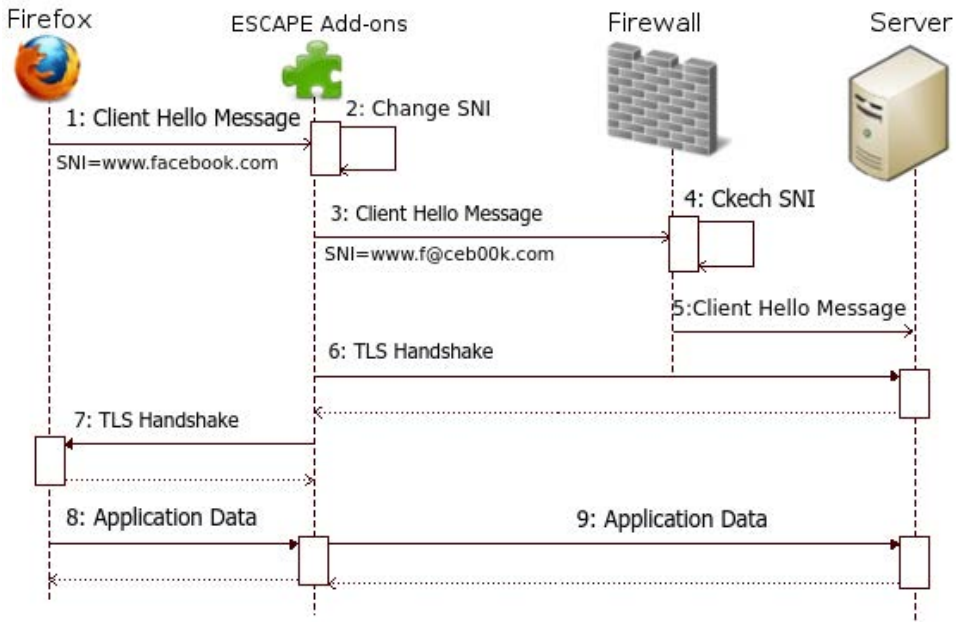


Figure 3.4: Escape add-on interactions

Therefore, Escape provides a convenient technical solution to get the control over the TLS handshake within the web browser in order to be able to modify `ClientHello` messages and to access blocked websites. The modified SNI extension is then processed by two parties, first by a firewall system, the second by the remote server that supports the SNI extension. In Section 3.4.2 we evaluate HTTPS websites behaviour against a fake SNI and in Section 3.4.3, we investigate firewall systems.

3.4.2 Evaluation of HTTPS servers supporting SNI extension

To assess how much the SNI extension is used, we have conducted the first investigation of the SNI deployment with a large set of web servers accessed over a HTTPS connection. Our probing follows RFC6066 [84], which describes how a server must interact in the case of receiving a `ClientHello` message with the SNI extension. If the server supports SNI, it must answer with a `ServerHello` message holding an empty SNI extension (i.e., length equal to 0), otherwise the server responds with a standard `ServerHello` message without SNI extension. Based on this server behaviour, we have implemented a SNI crawling module, where `ServerHello` messages are intercepted to verify if they hold an empty SNI extension.

Basically, TLS messages consist of two sub-layers, the first layer contains either the TLS handshake, TLS Change Cipher Specification or SSL Alert protocol, while the second layer holds the TLS Record protocol, which is known as an envelop for TLS messages and encrypted application data. Thus, let us assume $Record_i$ be the TLS record of TLS $Message_i$. Algorithm 1 describes the procedure to determine if the corresponding server supports the SNI extension or not. `ServerHello` messages are identified with a TLS Record type of 22 and a handshake type of 2. Once the `ServerHello` message is detected, the list of extension is extracted and scanned for an extension with type zero and empty data. According to RFC6066 [84], if this extension is present, we can deduce that the server supports the SNI extension.

Algorithm 1 Detecting the support of the SNI extension

```
1: if Record_i.type == 22 and Record_i.data[0] == 2 then //Detect ServerHello message
2:     ServerHello = TLSServerHello(Record_i.data)
3:     ExtensionList = ServerHello.extensions //Extract extensions list
4:     for ext in ExtensionList do
5:         if ext.type == 0 and ext.length == 0 then
6:             Server supports the SNI extension
7:         else
8:             Server does not support the SNI extension
9:         end if
10:    end for
11: end if
```

Our investigation covers some representative HTTPS domains taken from two lists. The first source is The Internet-Wide Scan Data Repository, which is managed and hosted by the University of Michigan[15]. The selected dataset contains the Alexa Top 1 million domains that respond on port 443 [86]. Instead of using this large amount of domains, we use the top 500 HTTPS websites. The second list[16] (provided by [87]) contains 120 sensitive sites blocked in China, the United Kingdom, and Saudi Arabia. We found that 33 websites out of the 120 are run over HTTPS, and

[15] `https://scans.io/series/443-https-tls-alexa_top1mil`
[16] `https://cs.uwaterloo.ca/~t55wang/knnsitelist.txt`

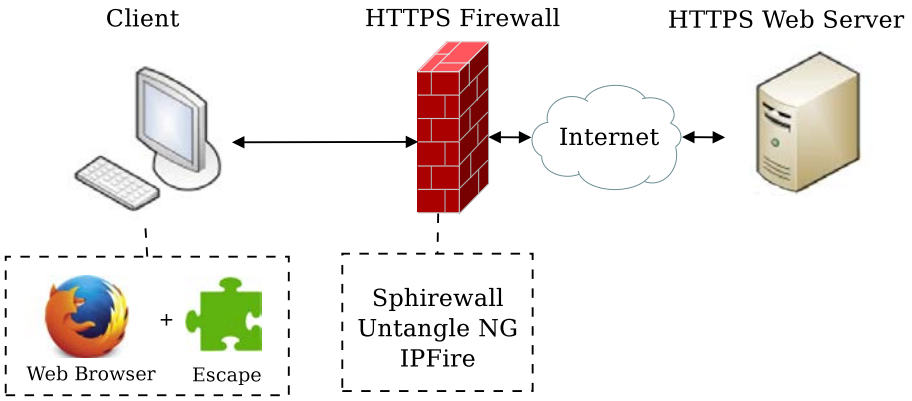


Figure 3.5: Evaluation testbed

we added 14 of them which were not in the first list. In total, we consider 514 HTTPS websites for this investigation.

For each website, we accessed the homepage with a web browser supporting the SNI extension (e.g., Firefox) in a automated way (via scripting) and the network traffic is analysed. After initiating the TLS handshake, the `ServerHello` messages are dissected by our crawler. The results show that 230 websites out of 514 (i.e., 44%) do not support the SNI extension in the way recommended by the RFC6066 [84], while the others perfectly support it. The websites that do not support the SNI extension are hosted on a dedicated server, so that they do not need this extension to select the proper SSL certificate. Monitoring the traffic for such HTTPS websites is a real challenge, because SNI values will simply be ignored on the server side. However, we will see in the next subsection that even servers supporting the SNI extension can still be accessed with a fake-SNI value.

3.4.3 Evaluation of the bypassing strategies

In this section, we first describe the evaluation environment. Second, we focus on how HTTPS servers deal with fake-SNI values. Third, we explore how SNI-based firewall systems interact with a fake SNI. Finally, we present the evaluation results for the shared SSL certificate case.

Evaluation Testbed environment

An evaluation environment was built to show the efficiency of our bypassing techniques and of its implementation in Escape. As shown in Figure 3.5, our evaluation testbed is composed of a client machine (Ubuntu 14.04) with a web browser (Firefox 33), which is configured with our Escape add-on to overwrite the SNI value with fake ones for the 514 websites composing our panel. We access these websites through different firewall systems that rely on the SNI extension for monitoring HTTPS services.

Table 3.1: Example of bypassing SNI filtering using a shared SSL certificate

No	Website	Without SNI	Fake SNI	Shared SSL Certificate
1	Google Search	Pass Filter	Pass Filter	Pass Filter using mail.google.com
2	Youtube	Pass Filter	Pass Filter	Pass Filter using news.google.com
3	Blogger	Pass Filter	Pass Filter	Pass Filter using mail.google.com

Impact on HTTPS websites

We assess the behaviour of several HTTPS servers by observing if the homepage is loaded or not despite the fake-SNI value used to bypass the firewall. The results show that 38 websites detect the fake SNI by showing *"HTTP hostname and TLS SNI hostname mismatch"* message, while 3 websites show a *"Bad Request"* message with no further details. Overall, 8% of websites (41 out of 514) refuse to go further in the TLS handshake when facing a fake SNI value, while 92% of the contacted HTTPS servers ignore this inconsistency according to the backward compatibility principle explained above. Websites like "www.change.org" and "www.uptobox.com" refuse to continue the TLS handshake, since they are both hosted by CloudFlare, a mutualized infrastructure, where a correct SNI value is mandatory to return the correct SSL certificate. Also, we investigate the behaviour of HTTPS servers in the case of shared SSL certificates (explained before in Section 3.3.2). Table 3.1 gives a few tested configurations that assess the fact that the shared certificate weakness can be used to bypass SNI-based monitoring when accessing Google services. It means, we are able to access a Google service which is blacklisted in the firewall configuration. So, we can successfully access services like Google search, YouTube or Blogger using the name of other services which share the same SSL certificate (i.e., mail.google.com, news.google.com, mail.google.com respectively).

Thus, we have shown that the vast majority of HTTPS servers can be accessed with a fake SNI, mostly by exploiting the backward compatibility issues, and the shared-certificates in specific cases. We next consider the impact on firewall systems.

Impact on firewall systems

We evaluate the bypassing strategies against three firewall systems as shown in Figure 3.5. The firewall machine is alternatively configured with one of the following firewall software:

- Sphirewall (version 0.9.9.27)
  An open source firewall system with the recent feature of filtering HTTPS traffic based on SNI.

- Untangle NG firewall[17] (version 10.2.1)
  It is a GNU-Linux based on Debian. The basic version is free with limited features. A paid module (but available as a trial for 14 days) enables HTTPS filtering based on the SNI extension and SSL certificate.
- IPFire[18] (version 2.15)
  An open source Linux firewall system with HTTPS filtering features.

The firewall machine contains two network interfaces, one configured as external interface to the Internet and the other connected to the local private network. The client machine with the web browser and Escape is directly connected to the firewall and has to pass through to access the Internet. We configured each SNI-based firewall software to restrict the access to the 514 HTTPS websites, and to reset the HTTPS connections if such websites are requested. As shown in Figure 3.4, the firewall checks the SNI value against a blacklist to take a decision. The results show that none of the tested firewalls was able to detect the accessed website when a fake SNI is inserted. So we are able to bypass filtering based on SNI and access all the blocked HTTPS services.

The high success rate (92%) of the bypassing strategies motivates the need for a method able to check the SNI value properly before forwarding the `ClientHello` message to the remote destination server. The evaluation also shows that most of TLS servers implement backward compatibility following the RFC6066. This explains why TLS servers always go further in the TLS handshake process. In the next section, we propose an improvement to overcome some of the limitations of the SNI-based monitoring.

3.5 SNI-based Monitoring Improvement

In the previous section, we have demonstrated that the reliability of the SNI-based monitoring is poor because it is easy to forge SNI values to bypass matching rules. Thus, we propose an improvement to SNI-based monitoring techniques to overcome this shortages, with the goal to improve current HTTPS firewall systems.

3.5.1 Architecture

In this work, we propose a robust and efficient solution to verify the content and the veracity of the SNI information defined by the client in order to overcome the aforementioned weaknesses. The main idea is to use the Domain Name System (DNS) information to validate the link between the hostname appearing in the SNI extension and the real IP address of the destination server.

When a website is accessed, prior to any TLS handshake, a DNS query is sent to resolve the IP address of the domain name. Then, a `ClientHello` message is configured with the SNI value holding the domain name and sent to the destination

[17] `https://www.untangle.com/`
[18] `http://www.ipfire.org/`

server IP address previously retrieved from the DNS answer. We cannot simply monitor DNS requests issued by users because malicious users can either resolve the domain name locally, or contact a colluding DNS server to escape detection. However, we believe that the information carried in DNS messages issued from a trusted DNS server can still improve the SNI-based monitoring. Indeed, the DNS information can be used to validate the "server-name" in the SNI extension because, according to the RFC6066[84], this extension must only contain DNS-resolvable hostnames. Moreover, using DNS answers from a trusted DNS server is more stable and reliable than using a pre-configured list of IP addresses, which requires frequent updates. We are in line with other works that use reliable DNS information [88] and remote server IP address [50] to identify network traffic. Figure 3.6 and Algorithm 2 detail the verification procedure of our proposed solution that assesses SNI using a trusted DNS server. The steps are as follows:

1. Inspect the `ClientHello` message and extract the extensions list and the destination server IP address.
2. Search for the SNI extension and extract the "server-name" value.
3. Send a DNS request to a trustworthy server with the "server-name" to get the corresponding IP addresses related to the SNI hostname.
4. Check if information in the DNS response matches the destination server IP address.

Algorithm 2 Validate SNI value with DNS request

if $Record_i.type == 22$ and $Record_i.data[0] == 1$ **then**
  $RealDstIPaddress = DestinationIP(Packet_i)$
  $ClientHello = TLSClientHello(Record_i.data)$
  $ExtensionList = ClientHello.extensions$ //Extract extensions list
  for *ext* in $ExtensionList$ **do**
    if $ext.type == 0$ **then** //SNI extension is founded
      $ServerName = ext.data$
      $DNSResp = DNSRequest(ServerName)$
      if $RealDstIPaddress$ included in $DNSResp$ **then**
        //The destination IP address is related to SNI value
      else
        //Invalid SNI value & Reset the connection
      end if
    end if
  end for
end if

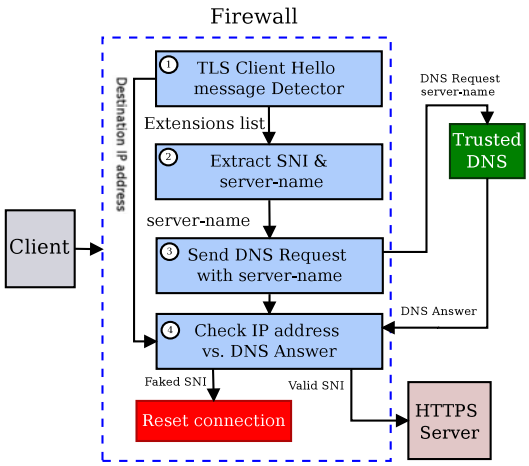


Figure 3.6: SNI verification system using DNS

3.5.2 Fake-SNI detection

In this subsection, we describe how to perform the check to assess the SNI extension from the information present in the DNS response. The fake-SNI value can actually take different forms: (1) a random string; (2) an empty value; (3) another (unfiltered) valid domain. In the first case, the random string will be easily detected because no valid answer will be returned by the DNS server (the "Answer RRs" field of the response will be set to 0, and the reply code will be 0x8183 "No such name"). In the second case, the firewall will detect the empty SNI extension and should block the connection, since all modern systems add an SNI value by default.

The third case will be illustrated by a real scenario, as shown in Figure 3.7. We assume the HTTPS website "twitter.com" is restricted by the SNI-based filtering. To overcome this limitation, a client browser with the Escape add-on can overwrite the SNI value of twitter (twitter.com) with another valid domain (e.g., www.google.com). In the firewall the process goes as follows:

1. The `ClientHello` message is detected and processed to extract the SNI ("www.google.com") and the destination IP address, which should be a twitter's IP address (like 104.244.42.X).

2. We send an independent DNS request with the SNI value (i.e., "www.google.com") to get the IP addresses related to this domain name.

3. The DNS response will contain some Google's IP addresses (like 66.102.1.X).

4. We compare the real destination IP address with those present in the DNS response, what shows a clear mismatch between them. This means that the connection must be considered suspicious by the system and processed accordingly.

5. The `ClientHello` message can be dropped to reset the HTTPS connection.

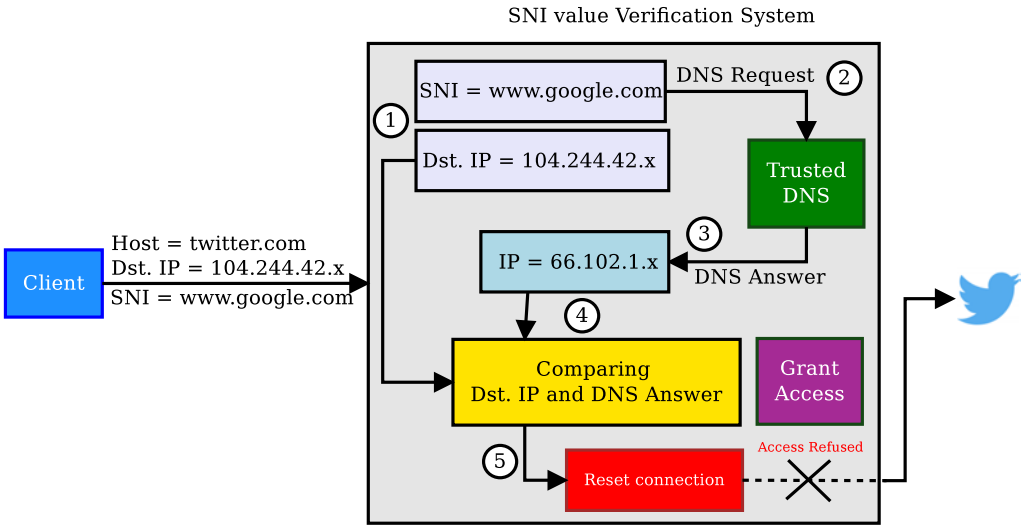


Figure 3.7: Example of fake-SNI detection using DNS

To check the correspondence between the destination server IP address and the ones included in the DNS response, different rules can by applied from an exact IP address match, to more loose rules checking the sub-network match on the first 24 bits. We will see in the evaluation section how our detection strategy can impact the detection rates.

3.6 Evaluation of SNI value Verification based on DNS

The proposed solution was evaluated against the same list of HTTPS websites used previously (Section 3.4.2). The evaluation consists in two parts. In the first one, we assess how the proposed technique behaves with a legit SNI value to evaluate the false-positive rate, while in the second one, we compute the overhead of our solution.

3.6.1 False-Positive rate evaluation

All `ClientHello` messages are inspected for verification, which also includes those with a legit (i.e., unaltered) SNI values. We need to evaluate how the proposed method deals with `ClientHello` messages holding a legit SNI extension, which, if not detected properly, will disturb the monitoring with a high false-positive rate and may alter the global HTTPS connectivity in case of stronger security polices (i.e., filtering). For example, false positives can appear when the DNS is used for load balancing and configured to answer with different IP addresses for a same requested name. For more explanations, the authors of [89] investigate different scenarios that lead to inconsistencies in DNS responses for hostnames related to HTTPS traffic. To assess the consistency between SNI values and DNS responses in a safe environment (i.e., no fake-SNI is present), we calculate the false positive detection rate as the proportion of `ClientHello` messages we would block despite a legit-SNI.

Table 3.2: Identification results regarding the IP address verification strategy

Detection Strategy	# Assessed Connections	True Negatives	False Positives
Exact match	2501	83.75%	16.25%
First 24-bits	2771	92.79%	7.21%
First 16-bits	2935	98.29%	1.71%

As shown in Figure 3.6, the original destination IP address of a `ClientHello` message is compared to the DNS answer that has been requested by using the "server-name" value written in the SNI extension. Our solution is then flexible because we can use different rules to evaluate the `ClientHello` message destination IP address when comparing to the DNS response, for example:

- Look for the full destination IP address (exact 32-bits match) in the DNS response.
- Look for a sub-network match (first 24-bits or first 16-bits match) of the destination IP addresses in the DNS response.

By using the aforementioned methods for comparison, the results show that, all tested 514 HTTPS websites are accessible. But we have investigated further. We then do not only considered the main HTTPS connection of each website, but also the large number of related connections toward HTTPS servers (in total 2986 different connections) that are used to render the complete web pages: load the website content, display advertisement, make statistics, etc. Table 3.2 shows the score achieved by each strategy. With the exact match, our solution validates correctly 83.75% of the legit SNI, while focusing on 24-bits and 16-bits prefix allow us to validate 92.79% and 98.29% respectively, of legit SNI. This can result in an overall 1.71% false positive rate. The low FP value should not alter web browsing and demonstrates the efficiency of using DNS information to validate client's SNI value.

3.6.2 Overhead evaluation

Our additional verification introduces an overhead related to computation and network latency. In our work, we have neglected the computation overhead since the performed checks involve very trivial operations like the extraction of a header field or the comparison of IP addresses. We pay more attention to the increased latency introduced by the communications with a trusted DNS to get the information to make our decision. Thus, we have studied two cases: when the trusted DNS resolver is a global one (Google DNS) or a local one (LORIA DNS, installed in our laboratory network). These two DNS resolvers should exhibit different performances because

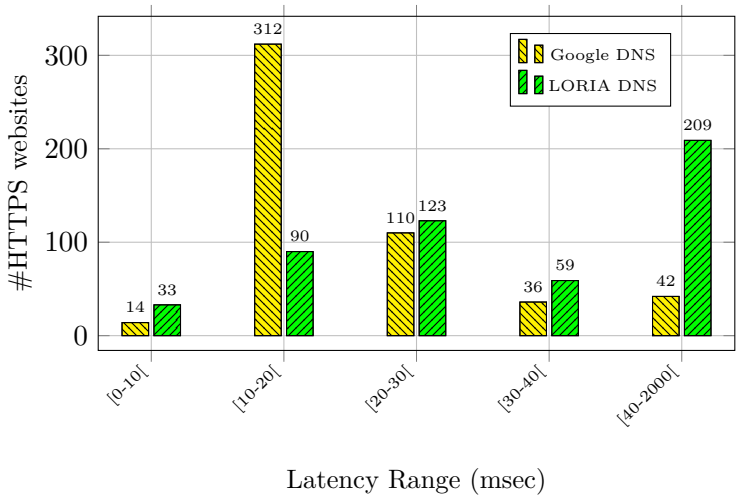


Figure 3.8: DNS latency range per number of HTTPS websites

one is closer in terms of network hops (less network delay), while the second should have a larger cache to answer directly to more requests (better cache hit ratio). Of course, when clients already contact a trusted DNS server to resolve the HTTPS server name (typically, the DNS server of their company or ISP), the DNS response can be directly analysed from the traffic without issuing an additional request, thus introducing no overhead.

Figure 3.8 shows the distribution of the latency for resolving the websites names using Local DNS and Google DNS. The x-axis shows the delay ranges in ms, and y-axis shows the amount of DNS responses. The first range [0-10[ms shows the number of websites for which name resolution takes less that 10 ms. We can notice that, when using a local DNS, 209 out of 514 websites (41%) need at minimum 40 ms to be resolved. However, the global DNS performs a lot better, where 312 out of 514 (61%) need between 10 and 20 ms to get the DNS answer, which is probably due to a very large cache that allows it to answer directly for many names, while the local one needs to forward many requests to the Root-Level of DNS, explaining its high value in the worst delay range [40,2000[.

According to these values, we notice that when using Google DNS, the average added delay is less than 15 ms. Moreover, according to [90], the average server response time (and standard deviation) of HTTPS websites is quite high 483 ($\pm$ 44.1) ms. In comparison, our evaluation results show that we do not alter the browsing experience of users when accessing HTTPS websites.

3.7 Discussion and Analysis

In this chapter, we have provided an in depth view of the latest technique for HTTPS services monitoring and filtering that is based on the SNI field of TLS, and which has been recently used in many firewalls and traffic shaper solutions. We have demonstrated why relying on this SNI extension for identifying and filtering HTTPS ser-

vices is utterly flawed and can be easily cheated. We have shown that the SNI-based monitoring method has two weakness points, regarding (1) backward compatibility and (2) multiple services using a single certificate. We have implemented our proof of concept exploiting SNI weaknesses inside a web browser add-on for Firefox called Escape and that allows users to bypass such an HTTPS filtering in their network. We have conducted the first investigation of the SNI extension deployment with a large set of web servers accessed over HTTPS connections, where the results show that 92% of the HTTPS websites included in the study can be accessed with a fake SNI, while no tested firewall was able to detect the fake SNI. To mitigate this issue, we have proposed a novel DNS-based approach to validate SNI in the context of HTTPS monitoring to verify the relation between the actual destination server and the claimed value of SNI by relying on a trusted DNS service. Finally, experimental results show the ability to overcome the shortage of SNI-based monitoring while having a small false positive rate (1.7%) and small overhead (15ms).

As reported in this chapter, the SNI extension has been used with different objectives, not limited to HTTPS traffic monitoring. It provides an easy and direct solution to services such as T-Mobile's service BingeOn to name the services of a given HTTPS flow. Thus, we believe that our improvement to the SNI-based approach is valuable to enhance the functionality of the less-sensitive SNI-based mnitoring solutions. But for the security monitoring of HTTPS traffic, our fake-SNI detection approach is still unable to precisely identify a given HTTPS service if it shares a SSL certificate with other services, since the claimed domain in the SNI extension to pass the filter operates under the very same IP address range than the blocked service, making it undetectable with our approach. This challenge is the main motivation for the following chapter that proposes a more robust method to identify services in HTTPS traffic, based on the traffic's characteristic itself rather than on a claimed SNI value.

Chapter 4

A Robust Multi-level HTTPS Identification Framework

Contents

4.1 Introduction

The identification of HTTPS traffic is motivated by network administrators who need to gain knowledge about the traffic in their network. In fact, the dependability, the security and the performances of networks and infrastructures can be lowered without suitable and robust methods to identify HTTPS traffic to allow its proper management.

The SNI-based approach we evaluated in the previous chapter is a direct method to monitor HTTPS services without decryption. However, for the security monitoring of HTTPS traffic, this method cannot be used to apply security policies, since it can be easily cheated. Even if we can detect when the SNI value is suspicious thanks to our improved version performing an additional DNS verification, we are still unable to precisely identify an HTTPS service if it shares a SSL certificate with other services. Also the HTTPS proxy-based method is not an option for us due to the privacy issues it raises, as detailed in the related work chapter.

Therefore, in this chapter, we present a privacy preserving framework that allows efficient identification of services in HTTPS traffic without decryption. The framework leverages machine learning to identify precisely HTTPS services from their traffic properties. The remainder of this chapter is organized as follows. Section 4.2 presents the proposed HTTPS identification framework. Section 4.3 presents our methodology, feature set and dataset. Section 4.4 evaluates the results of machine learning algorithms applied to our features and combined with the proposed framework. In Section 4.5 we discuss and conclude the chapter.

4.2 A Multi-level HTTPS Identification Framework

In this section, we design a complete framework to identify accessed HTTPS services in a traffic dump. First, we discuss the current privacy preserving methods for monitoring HTTPS traffic. Second, the multi-level classification paradigm is described. Finally, the framework's components are presented.

4.2.1 Privacy preserving identification methods

As we have discussed in the related work chapter (Chapter 2), there are two approaches to monitor HTTPS traffic without decryption; the first one is identifying the type of applications behind the encrypted traffic, and the second is website fingerprinting. However, identifying the type of encrypted applications is too generic-grain, while the website fingerprinting is too fine-grained, as it works at the page-level with static content, and is no longer adapted to today's web fetching dynamic content from multiple CDNs. Therefore, naming HTTPS services behind given flows is more suitable to manage services run through encrypted connections. For instance, for a given CDN service provider (e.g., Akamai), our framework will identify the service independently of the website used to render its content.

The method proposed in [50] is the only one that shares our goal of classifying encrypted traffic at the service-level. The proposed method generates service signa-

tures from TLS payload data and the server IP address. The certificate publication information field in the TLS certificate is then used to label data. However, this method fails when a single certificate is used for multiple services. For instance, it is impossible to differentiate between Google services and to have a fine-grained identification because of mutualized certificates and IP addresses they exhibit. Using the server IP address as an identifier can also be a problem in the case of virtual hosting or cloud hosting, where different services can be accessed under a same IP address. So we have to define a more reliable method that do not rely only on specific identifiers (e.g., SNI, SSL certificate), but try to fingerprint a service based on its network characteristics.

4.2.2 Multi-Level classification methodology

This approach has been used for the first time in the biology field for classifying the proteins in a hierarchical, tree-like fashion based on shared structural characteristics. First, they group proteins based on their structural class before attempting to assign a protein fold with machine leaning classifier [91]. Most of the existing works have taken a "Flat" view toward Internet traffic classification, focusing on identifying the websites and applications directly, while totally ignoring any hierarchical information. Figure 4.1 shows the principle of the flat classification.

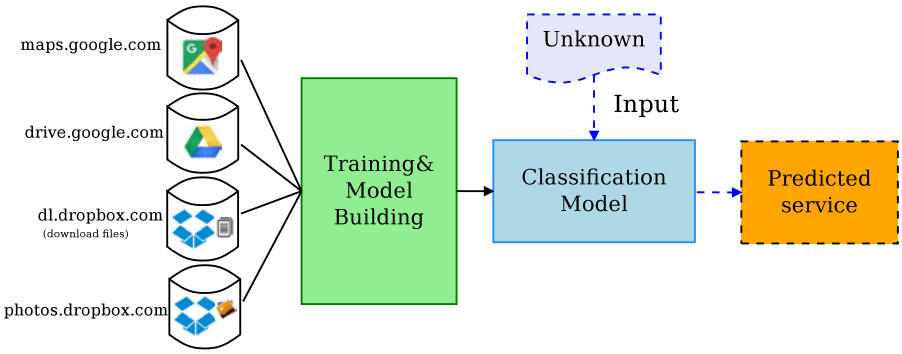


Figure 4.1: Flat classification method

The general idea of the multi-level approach is building a hierarchical structure, where the top level of the hierarchy is referred to as *Class-level* and each class is itself composed of individual *Folds*. This approach can be applied to the classification of HTTPS services based on the SNI extension that includes the domain name of the accessed service, and the domain names are well known to be hierarchical. The root-domain can refer to the *Class-level*, while the sub-domain represents the *Fold-level*. For example, let us assume we have HTTPS traffic for Dropbox services such as "photos.dropbox.com" (to access hosted photos) and "dl.dropbox.com" (to access hosted files) and Google services, such as "maps.google.com" and "drive.google.com". The training and classification hierarchies are built as shown in Figure 4.2. The *Class-level* classifier is built to differentiate between "google.com" and "dropbox.com". While the *Fold-level* contains two separate classifiers for each class (i.e., one for

Google and the second for Dropbox) to classify between their own services. However, in the case of a "flat" view, the classifier would need to distinguish between "photos.dropbox.com", "dl.dropbox.com", "maps.google.com" and "drive.google.com" in one single step.

Above all, this approach is perfectly suitable to identify services that share the same SSL certificate. We can build a dedicated classification model for the services listed in a given SSL certificate. It means, the classifier needs to differentiate between services related to the same *Class-level* what should be easier than trying to identify a service against all known services in a single step. To our knowledge, it is the first time that a multi-level approach is applied to encrypted traffic classification problem. Through the rest of the chapter we will use different terms more adapted to the web context with *Service Provider* referring to the *Class-level*, and the *Service* to the *Fold-level*.

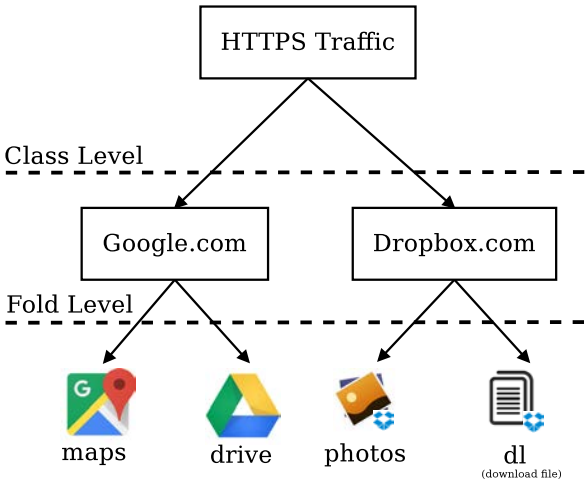


Figure 4.2: Multi-level classification approach

Formally, the multi-level classification approach can be described as follows: Let $S = \{S_0, .., S_H\}$, be a set of service providers, and each element of the set is another set of services $S_i = \{s_i^0, ..., s_i^N\}$, where N represents the number of services belonging to a service provider S_i. For each service s_i^j we define a set of k TLS connections as $T_{i,j,k} = \{t_{i,j,0}, .., t_{i,j,k}\}$, where i is a service provider $0 \leq i \leq |S|$, j is a service of i, $0 \leq j \leq |S_i|$ and $k > 0$. Hence, our classification approach can be defined as a defining function g that can map the right service provider a and service b from a trace $t_{i,j,k}$, where $g(t_{i,j,k}) \rightarrow a, b \quad 0 \leq a \leq |S|, 0 \leq b \leq |S_a|$. Thus, the classification can be put into categories as follows: we consider that an identification is *Perfect* if the service b (and inevitably the service provider a) corresponds to the network trace, *Partial* if the service provider is well identified but not the actual service, and *Invalid* if the proper service provider is not even identified.

4.2.3 Description of framework's components

The HTTPS identification framework includes several innovations increasing the identification accuracy. First, we define a new set of statistical features extracted from the encrypted payload of reassembled TCP connections. Second, we describe how the multi-level classification approach is used for identification. Figure 4.3 gives a flowchart of the proposed framework for identifying the services running in HTTPS connections. The pre-processing phase starts, as shown in Step 1, with the reconstruction of TLS connections extracted from safe HTTPS traces. Step 2 labels TLS connections thanks to the SNI field and builds the hierarchy between services and service providers. In Step 3, statistical features are calculated. These steps are realized in a controlled and safe environment to make SNI values reliable for the learning stage. The next phase is the training and building of classification models, where the first classification level model is built to differentiate between the service providers as shown in Step 4, while in Step 5 a sub-model for each service provider is built to differentiate among the services that belong to a same provider.

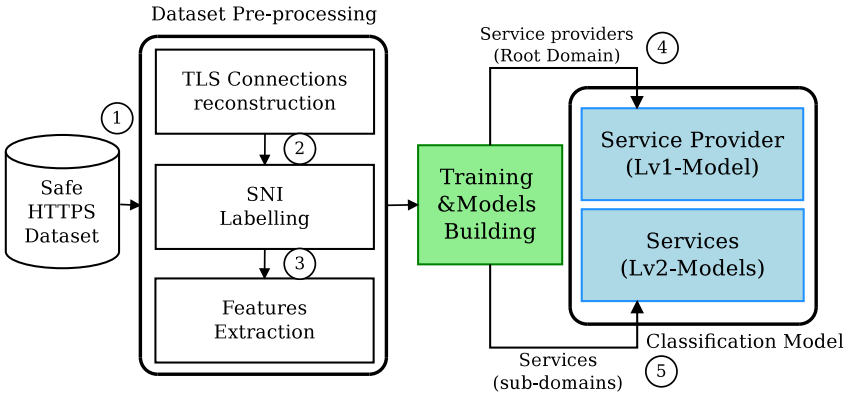


Figure 4.3: The workflow of the HTTPS traffic identification framework regarding models building

The main rationale behind our multi-level approach is to improve the classification model performance by using more precise classification models. The hierarchical representation of the dataset makes the machine learning algorithm (such as the C4.5 algorithm) more concerned (thanks to a specific weight per sub model) about the features that allow distinguishing the services from a given provider. As opposed to the flat view classifier, which needs to re-train the whole model when a new service is added, our approach is also more easily extendable to support a new service provider or a new service. In the first case, we just need to retrain the top-level and add the related service's classifier in the second level. In the second case, where a new service from an existing service provider must be added, we only rebuild the service provider's model.

Figure 4.4 illustrates the procedure in our framework to investigate an HTTPS traffic dump and to name the services inside it as follows:

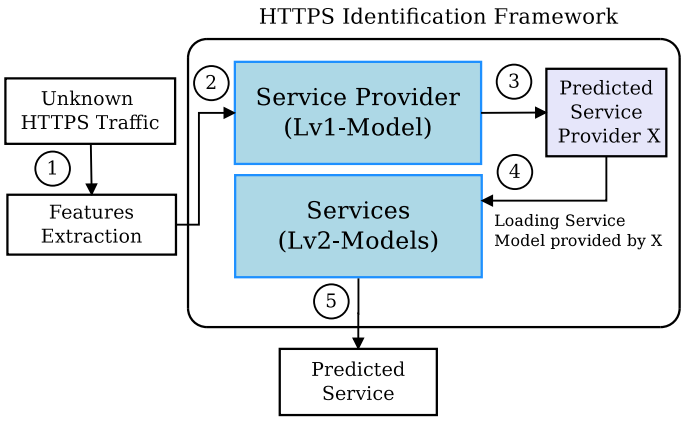


Figure 4.4: HTTPS traffic dump investigation process

1. Statistical features are extracted.
2. The output of Step 1 is used as input for the first level classifier.
3. The service provider is predicted.
4. The specific model that corresponds to the predicted service provider (Lv2-Models) is loaded to identify the precise service.
5. Finally, the name of the predicted service is given.

4.3 Features and Dataset

Machine learning techniques are employed with statistical features over the encrypted payload to identify services run in HTTPS traffic. The common point to all machine learning algorithms is that a feature set is needed to build the model. Then, each algorithm has its own technique to employ these features and distinguish between classes. The selection of the right features according to the problem to solve is essential for obtaining good results.

4.3.1 The statistical features

The proposed framework uses a set of 42 statistical network features for a TLS connection. Some of these features, as listed in Table 4.1, were already used in [92] and [93] for identifying the type of applications run in TLS connections. But we also propose 12 new statistical features related to the encrypted payload, as shown in Table 4.2. These new features are calculated on `ApplicationData` packets, which contain the overall payload after reassembling TCP segments. The reason is that encrypted payload content holds the most valuable data for the client and server, and it gives a closer view of the sequence of information exchanged among communication parties. To be clear, we only calculate statistical features over encrypted payload

without decrypting the content itself. In the following tables, the $\leftrightarrow, \rightarrow$ symbols express the direction of the packet from which the feature is extracted.

Table 4.1: The classical features (30 features) from [93]

Client $\leftrightarrow$ Server
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile)
Client $\rightarrow$ Server
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile)
Server $\rightarrow$ Client
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile)

Having statistical features related to the payload is not a novel idea, the authors in [94] used them for high speed real-time classification, however such features were not applied before to the encrypted traffic identification. The payload features already provided high accuracy when identifying the main type of applications like SSH, FTP, SMTP. Here, we use the benefits of payload statistics to have a fine-grained classification of HTTPS services. Moreover, according to [47], the influence of encryption algorithms on the size of packets is negligible with a small increase ranging from 21 to 33 bytes over the original (unencrypted) size for the most common ciphers. In our work, this result leads us to consider the size of encrypted payload regardless of the type of the encryption algorithm actually used.

Table 4.2: Our 12 additional proposed features over the encrypted payload (6 per direction)

Feature name	**Directions**
Average size	Client→Server, Server→Client
25th percentile size	Client→Server, Server→Client
50th percentile size	Client→Server, Server→Client
75th percentile size	Client→Server, Server→Client
Variance of size	Client→Server, Server→Client
Maximum size	Client→Server, Server→Client

Feature selection

In machine learning, the feature selection is a process by which we automatically search for a subset of original features that will optimize the learning accuracy. Filter methods operate a priori and without knowing the machine learning algorithm, so they are used before any induction takes place [95], contrary to wrapper methods

Table 4.3: The full features set (42 features)

Client ↔ Server
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile)
Client → Server
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile) Encrypted Payload (Mean, 25th, 50th, 75th percentile, Variance, maximum)
Server → Client
Total number of packets, Packet size (Average, 25th,50th,75th percentile, Variance, Maximum), Inter Arrival Time (25th,50th,75th percentile) Encrypted Payload (Mean, 25th, 50th, 75th percentile, Variance, maximum)

that work together with a ML-algorithm to take into account the bias of the algorithm in use. The key benefits of feature selectors algorithms is to reduce over-fitting by removing irrelevant and redundant features. It also improves the identification accuracy and reduces model building time. This leads to machine learning algorithms that train and learn faster.

Based on the experiments of [96], we use the Correlation-based Filter Selection (CFS) to select the most relevant features. Thus, it has been used with the Best-First search method to generate a candidate set of features from the full features set given in Table 4.3. The resulting set of selected features is composed of 18 features, listed in Table 4.4. The CFS selects 10 features from our proposed set out of 12 and 8 features out of 30 from the classical one (i.e., from the state-of-the-art features for traffic classification). This validates the rationale of our proposed features for identifying HTTPS services.

Table 4.4: The 18 selected features

Client ↔ Server
Inter Arrival Time (75th percentile)
Client → Server
Packet size (75th percentile, Maximum), Inter Arrival Time (75th percentile), Encrypted Payload (Mean, 25th, 50th percentile, Variance, maximum)
Server → Client
Packet size (50th percentile, Maximum), Inter Arrival Time (25th, 75th percentile), Encrypted payload(25th, 50th, 75th percentile, variance, maximum)

4.3.2 Selected machine learning algorithms

Many machine learning algorithms have been already used for encrypted traffic classification, such as C4.5, Naïve Bayes, SVM, etc. All machine learning algorithms

Table 4.5: Common machine learning algorithms for encrypted traffic classification

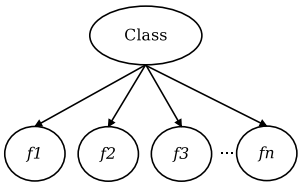
Identification Target	Naïve Bayes	C4.5	Random Forest	RIPPER	SVM	Reference
SSH, Skype	✓	✓		✓		[22]
SSL Traffic	✓	✓		✓		[98]
Skype			✓			[99]
Network Applications (Mail, FTP, WWW)	✓	✓				[100]
	✓	✓				[101]
	✓	✓				[102]
	✓	✓	✓		✓	[103]
		✓			✓	[104]
P2P			✓			[105]

identify traffic by using features as input of their classification method. Therefore, there is no best classification model for all applications but some may perform better than the others in our case. For instance, according to [97], if we want to differentiate traffic generated by multiple protocols, the C4.5 algorithm would be better than SVM algorithm, but if we target a small number of network applications, the SVM would be more suitable. As illustrated in Table 4.5, there are some machine learning algorithms that have been used extensively for encrypted traffic classification. For instance, in [22] the authors used Naïve Bayesian, RIPPER and C4.5 for detecting SSH and Skype traffic. The authors in [98] also use C4.5, RIPPER, and Naïve Bayes techniques to detect TLS traffic, while Li et al. apply the RandomForest algorithm [99]. In [100] the authors find that C4.5 and Naïve Bayes provide a good accuracy in the context of a near real-time identification of network traffic. Williams et al. [101] make a comparison between Naïve Bayes, C4.5, Bayesian Network, based on both computational performance and classification accuracy. Their results show that the these algorithms have almost the same accuracy for identifying network applications, but the C4.5 is faster than the other for identifying network flows. Therefore, for performance purpose, we select two different types of machine learning algorithms, Naïve Bayes and decision trees (C4.5 and RandomForest) to test them and choose the most suitable one for our problem.

Naïve Bayes algorithm

A Naïve Bayes algorithm uses a structure that consists in a parent node and a set of direct children as shown in Figure 4.5, where the root node represents the class and $\{f_1, f_2, ..f_n\}$ are the feature nodes of a particular class. This makes it easier to construct a Naïve Bayes classifier as compared to other classifiers because its structure is provided a priori and therefore it does not need structure learning procedure. Consequently, Naïve Bayes methods need little training time to compute the model used by the classification algorithm [100].

Figure 4.5: Naïve Bayes model



Decision tree algorithms (C4.5 & RandomForest)

Decision tree is one of the most popular family of algorithms in machine learning. It is a hierarchical data structure, where all features are used for building internal decision nodes and terminal leaves that hold the class of the instance [106]. During the training phase, each decision node splits the instances in two or more parts. Then, each path from the root node to leaf nodes forms a decision rule. This is used later in the classification phase. The C4.5 is a well-known decision tree algorithm and it has been widely used for traffic classification purposes. According to [107] the C4.5 advantages are mainly :

1. The C4.5 uses information gain ratio to select splitting attributes, which serve to build decision nodes in order to avoid the bias of selecting attributes.
2. It can handle both discrete and continuous attributes.
3. To avoid over-fitting, the C4.5 prunes the decision tree, which means once a tree has been created, the C4.5 walks-back through the tree to remove useless branches by replacing them with leaf nodes.

Figure 4.6 illustrates an example of a C4.5 decision tree to identify the accessed HTTPS services. Let us assume that we have an unknown HTTPS flow. First, we extract the statistical features, then the features are used as input for the decision tree. For example, if the mean packet size is greater than x (x is threshold calculated by the C4.5 algorithm during the learning phase), then the HTTPS flow is related to service#1, else we move to the next decision node to evaluate the variance of packet size. If the variance is larger than y, then the flow is related to service#2, otherwise it belongs to service#3.

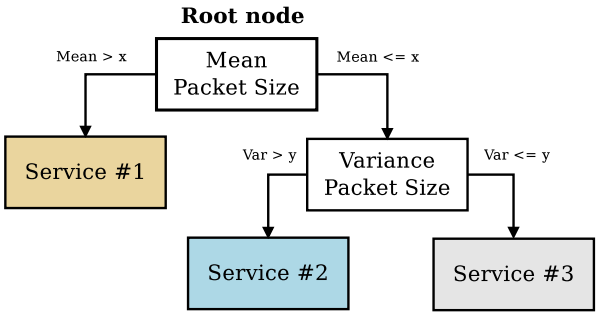


Figure 4.6: C4.5 decision tree using two features to classify services

RandomForest

The RandomForest algorithm is a type of ensemble classification methods, which train several classifiers and combine their results through a voting process [99]. The RandomForest algorithm consists of many tree-structured classifiers, where each tree is constructed with random selection of features. This randomness makes the classification model more robust than a single decision tree, and it helps to avoid the over-fitting on the training data.

Hence, if we need to identify the source of a given HTTPS flow, the extracted features are given to each tree in the forest. Then, each tree predicts the service name and the forest finally selects the class which has the most votes between the trees (i.e., Hard voting) [108]. Another way to get the final result is the Soft voting, where the final decision is taken based on the maximum sum of prediction probabilities, which gives an indication of the validity of the decision [109]. In this work, we use the soft voting to measure the *Confidence Score* of the framework, since it shows the level of agreement between the decision trees in a given forest.

The selection process

To select the most promising machine learning algorithms among Naïve Bayes, C4.5 and RandomForest, we made a first experiment using the full feature set, given in Table 4.3, over the collected dataset introduced in the next section (Section 4.3.3). As shown in Table 4.6, two algorithms have a higher accuracy: C4.5 and RandomForest. So they are used in the rest of this chapter. The selected algorithms will be analysed later in conjunction with different features and the complete identification framework.

Table 4.6: Evaluation of three machine learning algorithms for encrypted traffic classification

Algorithm	NaïveBayes	C4.5	RandomForest
Identification accuracy*	57.9%	87.8%	89%

* The evaluation metrics is present in Section 4.3.4

4.3.3 Private HTTPS dataset collection

We have built an HTTPS dataset to assess our method, since we needed traces with full TLS payloads to be able to compute any features on the traffic, and our investigations did not find any public dataset available with complete HTTPS traces. For example, in a dataset given by the WAND[19] research group, the TLS layer information and payloads are truncated. The authors in [110] collect only the HTTPS server answers between the `ServerHello` and the `ServerHelloDone` messages, i.e., the ciphersuite chosen by the server and the certificate chain sent. However, such datasets are not sufficient for us, since we need the full packets of the TLS sessions.

[19] http://wand.net.nz/

Building a dataset is a research question on his own, but, according to the survey [32], we are in line with similar works that had to build a private dataset to validate their approaches. Our training dataset has been collected in a well controlled environment (our lab) with a dozen of voluntary users, such as students and researchers from our research team. The HTTPS traces of complete user sessions have been collected over 5 months (from July to December 2014). We have used the SNI extension for labelling each HTTPS connection, since it directly refers to the specific HTTPS service that is accessed. In our case, we assume that the training traces come from a safe environment where no fake-SNI values are present during the learning phase.

Table 4.7: Examples of SNI values when accessing Google Maps

maps.google.com	mt0.google.com	mt1.google.com
khm.google.com	khm0.google.com	khm1.google.com
khmdb0.google.com	khmdb1.google.com	maps.gstatic.com

The SNI extension is used as the *Ground Truth* for the classification experiments. The "server-name" field inside the SNI extension is detailed enough to exhibit the service name. Most of the time, a website page needs to issue multiple concurrent requests to render the page's content that is distributed over multiple servers. That can exhibit different names in the SNI field. Table 4.7 illustrates some names related to the Google Maps service that can be seen in the concurrent connections made to render the maps pages. Some names show a common prefix followed by a number. To avoid over-differentiation of services and ease the learning process, we perform a pre-processing of SNI values by removing the numbers and special characters (like dashes) in order to keep the names as meaningful as possible. The obtained dataset is further described in Section 4.3.3.

4.3.4 Performance metrics

The performance of our classifier was evaluated by a K-fold cross-validation process. In K-fold cross-validation, the dataset is randomly split into K roughly equal parts. The algorithm is trained and tested K times, for each $k = \{1, 2, 3, ..., K\}$ the algorithm is trained on $K - 1$ parts and tested on the kth part. The most commonly used metrics to measure the effectiveness of the machine learning algorithms are *Precision*, *Recall*, and *F-Measure*. The *Precision* is defined as the number of True Positives (TP) divided by the number of TP plus the number of False Positives (FP).

$$Precision = \frac{TP}{TP+FP}$$

While the *Recall* is defined as the number of TP over the number of TP plus the number of False Negatives (FN).

$$Recall = \frac{TP}{TP+FN}$$

The *F-Measure* is calculated by the formula as shown below:

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Also, we use Receiver Operating Characteristics (ROC) graphs for visualizing the performance of our classifier. The ROC curve is used to measure the classifier performance regardless of class distribution or error costs. In fact, the classification accuracy score is often insufficient alone for evaluating a classifier performance. Basically the class prediction for each instance is made based on a continuous random variable R, which presents a "Score" computed for the instance using a given features. Given a threshold value T, the instance is classified as "Positive" if $R > T$, and "Negative" otherwise. So by using different value for the classifier threshold T, we get different True Positive Rate (TPR) and False Positive Rate (FPR) points. The ROC graph plots the TPR on the Y axis and FPR the X axis. To explain the plot, we calculate the area under the curve, where the larger area indicates a better classifier [111].

4.4 Evaluation of the Identification Framework

This section evaluates the effectiveness and the accuracy of our solution to identify services running in HTTPS connections. First, we provide a statistical overview of the collected dataset. Second, our proposed features are evaluated with the "Flat" view (i.e., the traditional way to perform classification). Finally, we present the improvement achieved by the proposed multi-level HTTPS identification framework.

4.4.1 HTTPS dataset description

The collected HTTPS dataset contains more than 288901 HTTPS connections for different services accessed by volunteer users. Table 4.8 shows the number of connections for the top services appearing in our dataset. Some services are used very frequently, while others are rarely used leading to a very small number of connections.

Machine learning algorithms need a significant training set, so we have to preprocess the dataset to determine a reasonable threshold for the minimum number of labelled connections per service (i.e., the sufficient number of solved examples for the training phase). In Figure 4.7 we show the relation between the minimum number of labelled HTTPS connections and the total number of different services we can process. For example, the maximum number of services is when considering all services having at least 5-connections per service in the traces, while with a minimum threshold of 50-connections, 263 services can be studied (from 107 distinct service providers). In the rest of this section, we will evaluate the classifier accuracy according to three thresholds 10, 40 and 100 minimum connections per service. The reason behind this selection is to measure how much the identification accuracy is related to the number of examples available for the training phase. This knowledge is also valuable when a new service has to be included for identification, so we can know the minimum number of connections needed to build a robust classifier.

Table 4.8: Services with the highest number of connections in the dataset

Service Provider	**Number of Connections**	**Number of Services**
Univ-lorraine.fr	71595	15
Google.com	47732	29
Akamihd.net	15700	6
Googlevideo.com	4580	1
Twitter.com	3325	3
Youtube.com	3160	1
Facebook.com	3147	4
Yahoo.com	1966	19
Cloudfront.com	773	1
Linkedin.com	467	1

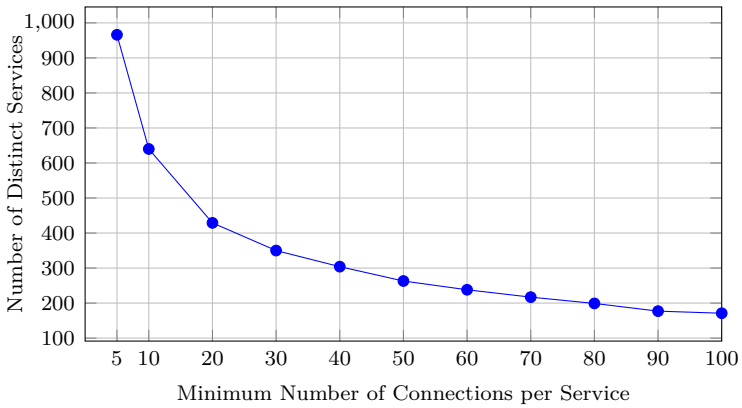


Figure 4.7: Number of distinct services being given a minimum number of related connections in the traces

From another perspective (i.e., the volume of traffic), Figure 4.8 illustrates how the overall number of connections considered in the dataset varies when increasing the minimum number of connections per service. Even with a number set to 100, the number of ignored HTTPS connections is less than 2% of the overall dataset, which means that 98% of the collected traffic is still used for the classification experiments. This can be explained by the fact that web traffic is unbalanced between services, a few of them being very popular and accounting for most of the HTTPS connections.

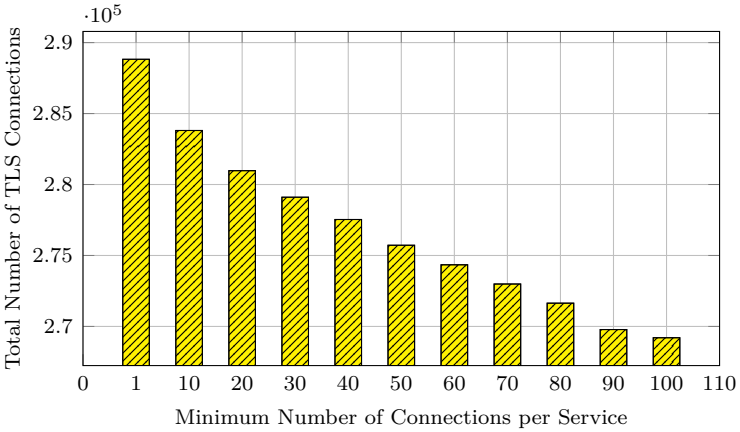


Figure 4.8: Total number of connections in function of the minimum number of connections requested for each service

4.4.2 Feature sets evaluation

The classical features from the state-of-the-art, as described in Table 4.1, are used as the baseline to evaluate our features. Tables 4.9 and 4.10 show the evaluation of the C4.5 and the RandomForest algorithms with different minimum numbers of connections per HTTPS service. The performance metrics are computed per class and the average for all classes are summarized in the following tables to measure the overall performance of the classification. The Weka[20] library was used for this evaluation (i.e., to apply C4.5 and RandomForest algorithms).

Employing the 10-Fold validation for the C4.5 algorithm achieves an average of 83.4%±1.16 precision, as shown in Table 4.9, while the RandomForest achieves 85.7%±0.4 precision. These tables also highlight the relation between the overall classification accuracy and the minimum number of connections needed: the accuracy do not vary much after 70 connections.

By using the full feature set (Table 4.3), the overall accuracy increases by 3 points as described in Table 4.10. The C4.5 algorithm now achieves 86.65%±0.7 precision, while the RandomForest achieves 87.82%±0.68 precision. The reduced set of selected features also achieves better results as shown in Table 4.11, the C4.5 achieves

[20] `http://www.cs.waikato.ac.nz/ml/weka/`

Table 4.9: Classical features with C4.5 and RandomForest

#Connections	**C4.5**			**RandomForest**		
-	Precision	Recall	F-Measure	Precision	Recall	F-Measure
10	81.8%	82%	81.7 %	NA%	NA%	NA%
40	83.3%	83.3%	83.1 %	84.9%	85.5%	84.8%
70	84.1%	84.2%	84%	86%	86.2%	85.7%
100	84.4%	84.7%	84.4 %	86.4%	86.8%	86.3%
Average	**83.4%**	**83.55%**	**83.3%**	**85.76%**	**86.16%**	**85.6%**

Table 4.10: Full features with C4.5 and RandomForest

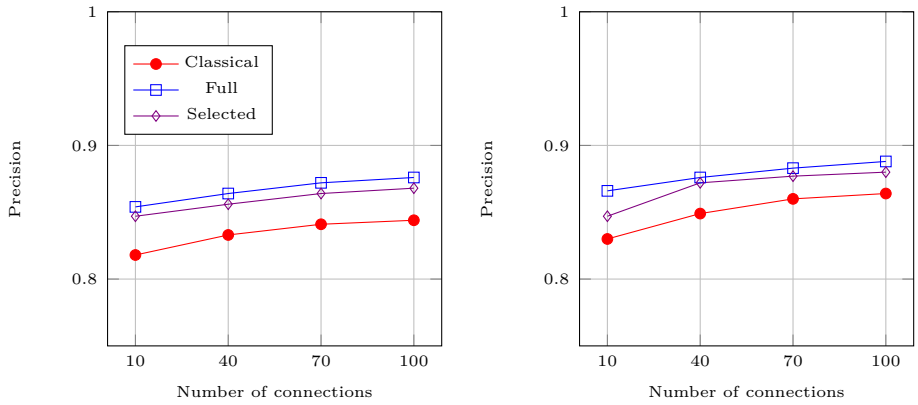
#Connections	**C4.5**			**RandomForest**		
-	Precision	Recall	F-Measure	Precision	Recall	F-Measure
10	85.4%	85.4%	85.2 %	86.6%	87.2%	86.6%
40	86.4%	86.4%	86.2 %	87.6%	87.9%	87.6%
70	87.2%	87.2%	87.1 %	88.3%	88.5%	88.1%
100	87.6%	87.7%	87.5 %	88.8%	89%	88.7%
Average	**86.65%**	**86.67%**	**86.5%**	**87.82%**	**87.86%**	**87.75%**

85.87%±0.64 precision and RandomForest 87.60%±0.10 precision. Figures 4.9(a) and 4.9(b) compare between the two sets of features. The full feature set achieves a higher accuracy compared to the classical one, with both C4.5 and RandomForest algorithms.

Figure 4.10(a) and 4.10(b) show the ROC curves for our classifiers to identify Google services using C4.5 and RandomForest with different features sets. The TPR and FPR points have been calculated by using different threshold values $T \in [0, 1]$, as explained in Section 4.3.4. The performance of a classifier is presented by the Area Under the Curve (AUC). The higher the area is, under the curve, the better the classifier model is. We can notice how the full features model performs better than the model with classical features.

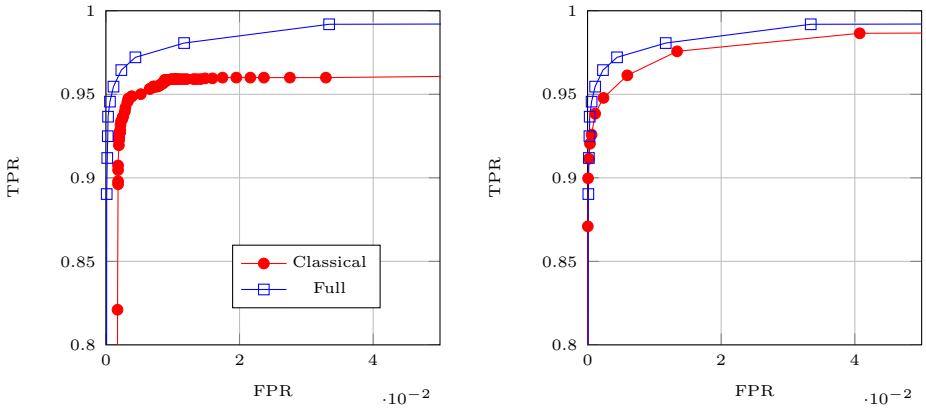
Table 4.11: Selected features with C4.5 and RandomForest

#Connections	**C4.5**			**RandomForest**		
-	Precision	Recall	F-Measure	Precision	Recall	F-Measure
10	84.7%	84.6%	84.6 %	86.6%	87.2%	86.6%
40	85.6%	85.7%	85.5 %	87.2%	87.6%	87.1%
70	86.4%	86.4%	86.3 %	87.7%	87.9%	87.6%
100	86.8%	87%	86.8 %	88%	88.3%	88%
Average	**85.87%**	**85.92%**	**86.05%**	**87.37%**	**87.75%**	**87.32%**



(a) C4.5 (b) RandomForest

Figure 4.9: Precision comparison between classical, full and selected features with C4.5 and RandomForest algorithms



(a) C4.5 (b) RandomForest

Figure 4.10: ROC curves for identifying Google service using classic and full sets of features with C4.5 and RandomForest classifiers

4.4.3 Evaluation of the HTTPS identification framework

The previous section was focused on the evaluation of the different feature sets, and a simple flat identification process was used for that. Based on the aforementioned results, the RandomForest algorithm with the full feature set performs the best, as shown in Table 4.10, obtaining up to 88.8% of accuracy. We now consider our multi-level framework and its specific evaluation (described in Section 4.2.2) and show that we can obtain a better accuracy.

As shown in Figure 4.3 the core component of the proposed framework is the first level model, which predicts the service provider of the HTTPS traffic. Based on the previous section best results, we use RandomForest with 100 connections as minimum number per service to evaluate the framework. We start by evaluating each level to measure the performance of each framework's components and then we evaluate the whole framework as one black box. Concerning the evaluation of the first level model, we still consider the three sets of features (classical, full, selected) to show the improvement of the framework's classifier for each of them. Table 4.12 shows that the full feature set achieve a high level of accuracy for identifying the service provider of HTTPS traffic with 93.6%.

Table 4.12: First level model evaluation with RandomForest

-	**Precision**	**Recall**	**F-Measure**
Classical Features	91.9%	92%	91.7%
Full Features	93.6%	93.7%	93.5%
Selected Features	92.6%	92.8%	92.6%

In the second level of classification, separate classification models are built for each service provider. Each model has been evaluated separately with the same approach used in the first-level. Table 4.13 summarizes the distribution of the overall accuracy for these models. In our dataset, we have 68 distinct service providers that exhibit a service counting more than 100 connections in the traces. There are 51 service providers that have more than 95% of good classification of their own different services. As expected, this result supports the idea that identifying services from the same service provider with a specific model achieves higher accuracy. For example, we can classify among the 19 different Google services, running under "google.com", with more than 93% of perfect identification. Moreover, we can notice that the sets of full and selected features have the same accuracy and and they achieve higher performance than the classical feature set.

The whole framework (Level1&2) has been evaluated by 10-Fold cross validation. The overall accuracy is calculated based on our evaluation method described in Section 4.2.2. The results show that we achieve 93.10% of perfect identification and 2.9% of partial identification.

To assess the confidence level of the framework, we divide the confidence score range of value [0-1] to 11 sub-ranges. The prediction scores are counted for each sub-

Table 4.13: The second level models accuracy with 100 connections as minimum number of connections per service

Accuracy Range	**Number of service providers**		
-	**Classical Features**	**Full Features**	**Selected Features**
100-95%	50	51	51
95-90%	5	5	5
90-80%	6	6	6
Less than 80%	7	6	6

range as shown in Figure 4.11, correct prediction at the top and wrong prediction at the bottom. We can observe that, 86.68% of the predictions are in the sub-ranges [0.8-0.9[, [0.9,1[and 1. Hence, we are almost sure that we identify correctly and so assume the results as relevant. The remaining 13.16% of the predictions is more balanced (46% right and 53% wrong) and will need more analysis, such as DNS or IP address investigation.

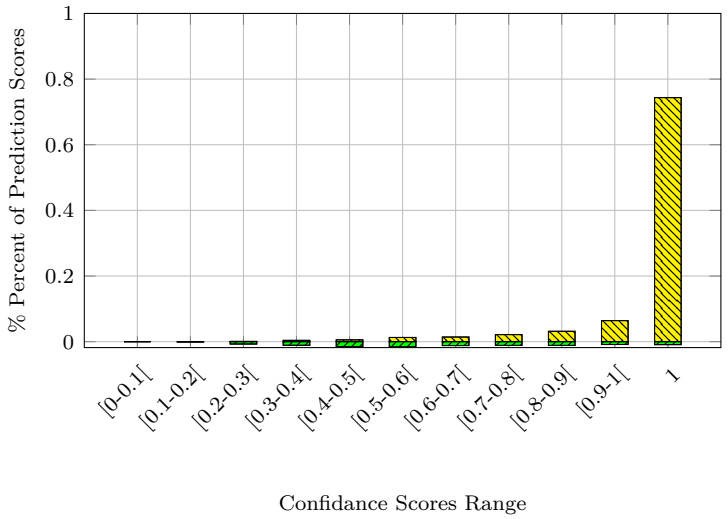


Figure 4.11: Confidence scores hits for the multi-level identification framework

The quick evolution of HTTPS services creates a challenge to machine learning based identification methods. Their statistical features must be re-evaluated regularly to cope with changes. We measure how much our framework is able to identify HTTPS services over time without re-training. Figure 4.12 depicts the classification errors over time. We can notice that even after 23 weeks without new learning phases, we can still identify 80% (error <20%) of HTTPS services.

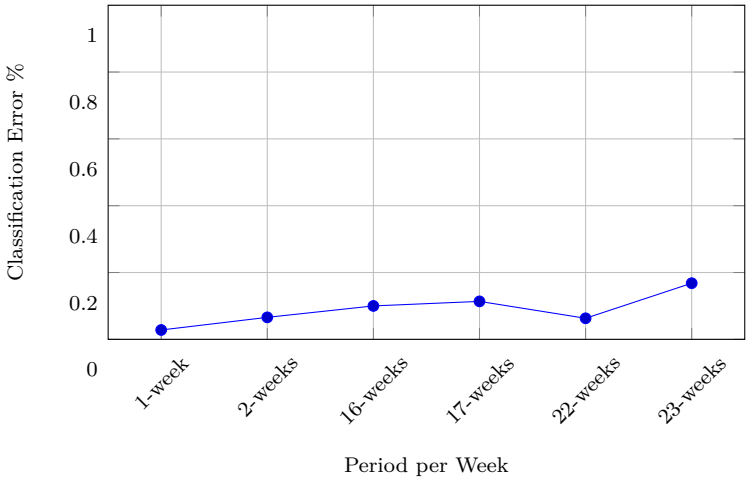


Figure 4.12: Percent of classification error over time

4.5 Discussion and Analysis

Being given the need for reliable yet precise monitoring methods of HTTPS traffic, the objective of this chapter was to propose a method that do not rely on the decryption of the traffic or on protocol header fields, but rather on the properties of the traffic itself. In this chapter, we have presented a privacy preserving framework that allows efficient identification of HTTPS thanks to machine learning techniques. To the best of our knowledge, we are the first to propose such a framework, which is primordial as applications tend to be more and more web-based. Therefore, our main contribution is a complete framework to identify the accessed HTTPS services in a traffic dump. We have also included in the framework several innovations making the identification more accurate and the framework more practical. The first one is a new set of statistical features extracted from the encrypted payload of reconstructed HTTPS connections. The second is a multi-level classification technique, where the main service provider is identified in the top level, while the precise services are recognized in the second level. Our results have shown a high level of accuracy of 93.10%, plus 2.9% of partial identification.

As reported in literature work, the classification of encrypted traffic using machine learning techniques has been mainly applied in the context of offline analysis, where full and complete network flows are available for training and classification. Our framework also work on complete sessions, and so can be only used to identify communications toward forbidden services in the context of a forensic analysis (i.e., after a revealed security issue). Such a tool may help to discover evidential information about the source of a security incident even if the attacker uses a fake SNI.

While interesting, offline identification is limited as it does not allow the management of encrypted traffic, what keeps being our main goal. Indeed, real-time identification is mandatory to allow ISPs and network administrators to perform

the proper network management activities on HTTPS traffic when operating their network. So, we aim to adapt and extend this offline framework to enable real-time analysis and identification of HTTPS services. The real-time identification of HTTPS services is presented in the next chapter.

Chapter 5

Real-Time Monitoring of Services in HTTPS Traffic

Contents

5.1 Introduction

Real-time identification of Internet traffic is vital for network operators. They need to know what is flowing over their network as early as possible thanks to their monitoring infrastructure, so that they can take the proper actions to manage the network, as for example for security, QoS or traffic engineering purposes. Indeed, observing relevant events has to be done as soon as possible to avoid costly inefficient functioning or even downtime. Nowadays, the increasing use of HTTPS undermines the effectiveness of traffic monitoring, making very difficult the proper identification and management of HTTPS flows, contrary to HTTP that is easier monitored in real-time thanks to DPI, by simply looking at the first field of the HTTP header.

There are currently three practical methods to monitor HTTPS services in real-time namely, SNI-based, SSL certificate-based and HTTPS proxy-based. The reliability issues of the first two approaches were explained earlier in Chapter 3. While the HTTPS proxy suffers from huge trust and privacy issues. In this chapter, we present a framework, which is able to identify HTTPS services in real-time, and without simply relying on a single protocol-field but on the traffic pattern itself learned from the first few packets. The rest of chapter is organized as follows. Section 5.2 explores additional related work on HTTPS identification with real-time monitoring requirements. Our approach for a real-time identification of HTTPS services is described in Section 5.3. Section 5.4 presents the experimental evaluation of our solution (features, algorithm and dataset). The implementation of a prototype to identify HTTPS services in real-time is detailed in Section 5.5. Finally, Section 5.6 summarizes and discusses the contributions of the chapter.

5.2 Background on Real-Time Monitoring

Monitoring network traffic in real-time has two main challenges before even classifying: reassembling TCP flows, and gathering a sufficient number of packets needed to classify a flow on the fly. In this section, we first explore the literature work that discusses these challenges, then we address the requirements and the limitations of identifying HTTPS services in real-time.

5.2.1 Related work on real-time network traffic identification

Flow Reassembly

A TCP flow is defined as a set of packets that have the same 4-tuples (source IP address, source port, destination IP address and destination port). Each HTTPS flow contains three types of packets; handshake packets, application data packets and pure TCP control packets (e.g., ACK, SYN). Therefore, the idea of flow demultiplexing is to build a table (e.g., Hash table), where each record contains the packets from a single TCP flow and is accessed by the key of the flow (i.e., the 4-tuple). For each new incoming packet, the 4-tuple key is computed and used for searching in a table to find the corresponding flow of the packet.

Xiong et al. [112] propose a TCP flow reassembly approach for real-time network traffic processing in high-speed networks. They used the *Recently Accessed First* principle to reduce the search cost of the related flow of incoming packets, by bringing the most recently accessed flow record on top of the table, so the next following packets will be mapped quickly. In [102] the authors also apply the same method for flow reassembly: they run packet capture, flow reassembly, and classification with machine learning modules in a pipeline way. Their method achieves reassembly throughput of 24,997.25 flows per second, while the average delivery delay is 0.49 seconds.

Groleat et al. [104] also share the idea of building a real-time classifier based on SVM algorithm. However, their target is to detect the categories of network applications, while we aim to be more precise to identify flows at service-level. In their work, they propose a high-speed flow reassembly algorithm able to handle one million concurrent flows. To achieve this high-speed approach, they use the massive parallelism and low-level network interface access of Field-Programmable Gate Array (FPGA) boards. Their results show they have up to 20 GB/s for flow reassembly.

In our work, we benefit from the approach in [112] to optimize the HTTPS flow reassembly module. We experimentally validate the applied flow reassembly with the Tcpdump[21], pkt2flow[22] and Wireshark[23] tools. The flow reassembly using hardware facility like FPGA is promising, but we keep it for our future work.

Sufficient Number of Packets

The sufficient number of packets is a critical parameter, since packets are required to obtain statistics about the flows that are used as input for identification methods. Therefore, many research works have investigated how many packets are actually needed to obtain sufficient statistics to keep high accuracy levels. Bernaille et al. [47] report using the first 4 packets to early identify applications over TLS traffic like FTP, SMPT or POP3. Kumano et al. [93] investigate the real-time identification of applications type (e.g., P2P, Streaming, etc.) in encrypted traffic. Their results show that they need 70 packets for their real-time identification. Maolini et al. [113] handle the problem of identifying SSH traffic and the underlying protocols (SCP, SFTP and HTTP). Their results show that the first 3 to 7 packets with a K-means clustering algorithm are sufficient to achieve a real-time identification. In [114] the authors use the first 100 packets for a real-time classification between applications like SMTP, POP3, eDonkey and BitTorrent. They have combined K-means and K-nearest neighbour clustering algorithms to construct a lightweight real-time classifier for encrypted traffic.

Based on the aforementioned papers, we can see that values range from 3 to 100 packets. In our case, we cannot clearly deduce from this state-of-the-art the sufficient number of packets we need, since we have a more precise target (i.e.,

[21] `http://www.tcpdump.org/`
[22] `https://github.com/caesar0301/pkt2flow`
[23] `https://www.wireshark.org/`

service-level identification), which is more challenging than the application-level of the aforementioned studies that consider HTTPS traffic as one class. Therefore, we must carefully evaluate the sufficient number of packets that makes it possible to recognize services in HTTPS traffic.

5.2.2 HTTPS services identification in real-time

The increased complexity of web applications, which provide the ability to deliver very different kinds of services, such as email, online storage, video streaming, maps, motivates the identification of HTTPS services in real-time. Intuitively, offline identification methods should give better results since they depend on stable and complete flows' statistics. In offline approaches, the identification process starts after network flows have been terminated and only then statistical features are computed regardless of time and computation complexity. However, the problem is harder in the context of real-time identification where the identification runs in parallel with flows and the goal is to name the HTTPS service behind a live flow as soon as possible, by using as little information as possible without sacrificing accuracy. Real-time identification methods are at the core of middle-boxes like Network Intrusion Detection Systems (NIDS) to identify and manage services, for example to prevent access to unsafe websites. Also, it can be used to apply QoS by triggering automated reallocation of network resources for high priority services such as webmail services [115].

Recently, efforts have been made to use machine learning techniques and to make them efficient for real-time identification. The authors in [93,102,114,116,117] apply different machine learning approaches (C4.5, REPTree, K-means, SVM, Naive Bayes) for identifying the application type (e.g., P2P, SSH, BitTorrent) of encrypted traffic in real-time manner with acceptable level of accuracy and overhead. We are going further into the identification process by identifying specific HTTPS services in real-time.

5.3 Real-Time Framework for Monitoring HTTPS Services

This section presents our methodology for identifying and monitoring HTTPS services in real-time. We first present the formal representation of the problem. Then, we explore the features which are calculated online over reassembled flows and the machine learning algorithm we use. Finally, the architecture of the proposed solution is provided.

5.3.1 Overview

Our proposed framework contains two phases. The first phase includes offline learning and model building and the second phase concerns real-time flow reassembly and

features extraction. For the real-time identification, we use TLS handshake packets and the first application data packets. The reasons behind this selection are:

- Minimizing the overhead of the flow reassembly step because we have to reassemble only a few packets. Therefore, in the rest of this chapter, we mean by flow reassembly, the regrouping of the TLS handshake packets and a few number of application data packets.
- Minimizing the effect of out-of-order packets, which is a challenge in real-time identification systems [58]. Therefore, we benefit from the fact that the TLS handshake messages must arrive in-order to establish the TLS connection.
- Increasing the extendability of the proposed approach to cope with the increased complexity of web applications that can be easily extended and which behaviour and statistical-signature may change significantly over time.

Yet, the continuous evolution of web applications creates an overhead to machine learning based identification methods that must re-evaluate their statistical features to cope with changes. Thus, using TLS handshake messages and only few application data packets should reduce the effect of traffic characteristics changes in time, since handshake messages are only used to establish TLS connections and thus should be more stable in time. To our knowledge, it is the first time that TLS handshake and few application data packets are reassembled in real-time to identify HTTPS services.

Formally, the identification of HTTPS services can be described as follows. Let us assume $X = \{x_1, x_2, ..., x_n\}$ a set of HTTPS flows. A flow instance x_i contains h TLS handshake packets, and d application data packets. A flow x_i is characterized by a vector of r features (e.g., packets mean size). Also let $Y = \{y_1, y_2, ..., y_s\}$ be the set of known HTTPS services, where s is the total number of known HTTPS services. The y_i is the HTTPS service behind an HTTPS flow x_i. Thus, our target is to learn the mapping of a $r-$ dimensional variable X to Y.

5.3.2 The statistical features

Identifying HTTPS services in real-time requires to limit the choice of features to those that can be computed quickly on partial flows. Also, a packet number threshold must be determined, which is the sufficient number of packets that must have been seen from a flow for the identification engine to start the extraction of features and perform the identification [114]. The proper threshold value will be discussed and evaluated later.

According to our public HTTPS dataset (presented in Section 5.4), the number of handshake packets is not fixed among TLS sessions, since the TLS handshake can take three or four packets, or even more depending on whether a client and a server start a new fresh handshake or resume a previous one. Therefore, we use the start of exchanging application data packets as the indicator of the handshake phase completion. Then, we further consider the first application data packets.

Table 5.1: Our 36 features over TLS handshake and application data packets

Common features (9 features per direction)
Packet size (Average, 25th, 50th, 75th Percentile, Variance, Maximum) Inter arrival time (25th,50th,75th Percentile)
TLS handshake header features (6 features)
`ClientHello` packet (length of Session ID, number of cipher suites, length of extensions list)
`ServerHello` packet (length of Session ID, used cipher suite, length of extension list)
Application data packets features (6 features per direction)
Packet size (Average, 25th, 50th, 75th Percentile, Variance, Maximum)

Hence, we use a features set that contains three group of features; the first group is common to all packets (i.e., TLS handshake and application data packets), the second one is related to the TLS handshake header features, while the third group is related to application data packets. Table 5.1 summarized the 36 features we have selected for identifying HTTPS services in real-time.

5.3.3 Selected machine learning algorithm

Once the features are computed, they are used as input for a machine learning algorithm. In Section 4.3.2, we have presented the most popular machine learning algorithms for encrypted traffic classification. Thus in this chapter, we only consider the C4.5 algorithm. There are several reasons behind this choice but the main reason is based on the performance comparison between different machine learning algorithms conducted in [101] and where the results show that the C4.5 algorithm is very fast and efficient, what becomes particularly important in the context of real-time identification. It also has a low computational overhead and is simple to implement [116]. Finally, the C4.5 requires less learning time and classification time. Nonetheless, to complete this work, we envisioned to investigate other machine learning algorithms in future works.

5.3.4 Architecture

Figure 5.1 illustrates the phases of the proposed approach. First, we run the offline part to build the classification model, which will be used in the real-time phase for identifying HTTPS services. In the offline phase, the events proceed as follows:

- (A) Building and pre-processing a representative HTTPS dataset.
- (B) Extraction of valid HTTPS flows.
- (C) Use of SNI extension for labelling.
- (D) Computation of features, which are used as input of step (E) for building the classification model using the C4.5 algorithm.

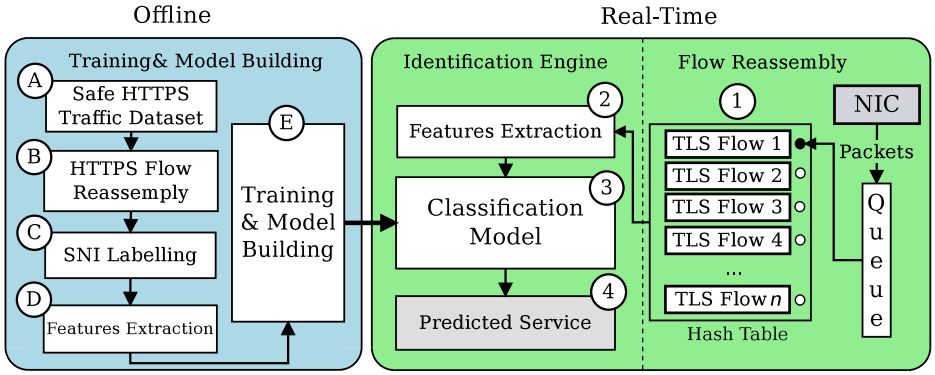


Figure 5.1: Framework architecture for the real-time monitoring of HTTPS services

The real-time phase consists in the flow reassembly process and the identification engine. The core of the identification engine is prepared in the offline phase. The flow reassembly module demultiplexes the TLS handshake and application data packets into the corresponding flow. Compared to the framework we presented before in Chapter 4, the main difference is that the input of the classification model is now a flow collected and reconstructed in real-time that contains the TLS handshake and a few application data packets. In the previous framework, the input was a completed TLS session. Figure 5.1 illustrates the interactions of the proposed approach in the real-time phase as follows:

- (1) The reassembly module receives TLS packets from the Network Interface Card (NIC) and demultiplexes incoming packets.
- (2) When the number of packets in a flow reaches a given threshold, the feature extraction over the flow packets is triggered.
- (3) Computed features are given to the pre-built classification model for identification.
- (4) Finally, the predicted HTTPS service is given. This output can be used to manage the traffic like applying QoS or security policies to HTTPS services.

5.4 Evaluation of the Real-Time Monitoring Framework

In this section, we evaluate the key parameters that may affect the accuracy of the proposed solution: the HTTPS dataset and the threshold value that defines the sufficient number of packets needed to identify HTTPS services in real-time. For the evaluation, we use the features set described in Section 5.3.2 with the C4.5 machine learning algorithm. Except if explicitly mentioned, we apply the 10-fold cross validation to compute the identification accuracy as our main evaluation metric (which means we use 90% of data for training and 10% for testing). We use the scikit-learn[24] library to implement and test the different C4.5 identification models.

[24] `http://scikit-learn.org/stable/index.html`

5.4.1 Public HTTPS dataset

To train and build a classification model, we need a representative dataset of HTTPS traffic dump. However, there is a lack of open full HTTPS datasets including packets' payload. The main reason is probably related to privacy and security risks of publishing a raw dataset that contains users' activity over HTTPS websites. But, packet's payload is mandatory for us to be able to compute the statistical features over encrypted payload we use to feed the machine learning algorithm.

Data Collection

We have built our own dataset by visiting HTTPS websites on a daily basis during two weeks, with both Google Chrome and Firefox web browsers. In Section 4.3, we have already presented an HTTPS dataset. The reasons for building a new one are as follows: (1) the old dataset was outdated, since it was collected in July 2014 and we need to deal with a fresh dataset since flows will be identified in real-time; (2) the old one contains 263 services only, while in the new dataset we target 1000 HTTPS services to better validate our real-time identification approach.

The websites we consider to build the new dataset are based on the list of the top HTTPS websites referenced in the Internet-Wide Scan Data Repository[25]. With an automated script we have loaded the main page of each website, which is sufficient to get the beginning of the flows and not create privacy issues, since no real users have generated the traffic. The dataset we made contains 487312 flows related to 7977 different HTTPS services for a total size of 55 GB. This higher number of services is due to many HTTPS flows from third-party content providers that are used to render the complete web pages: load the website content, display advertisement, make statistics, etc. The collection process took place over two weeks. Therefore, we consider only the HTTPS services that appear at least 14 times in our traces and are included in the original target list. We have per HTTPS service at minimum 14 labelled flows to train the classification model. The dataset has been pre-processed by eliminating uncompleted flows with no application data packets. We indeed need to be sure that we use valid flows to train our model.

The training dataset is divided in two parts according to the web browser used to access the websites. Some flows were issued by Google Chrome (70771 labelled flows related to 1384 HTTPS services), and the others by Firefox (56116 labelled flows related to 890 HTTPS services). Therefore, we evaluate our approach over HTTPS services accessed by Google Chrome and Firefox to be sure that the classification results are not sensitive to the client but just to the accessed service. For the sake of reproducibility, our HTTPS dataset is publicly available[26] with full encrypted payloads and the TLS information layer. We hope to contribute in solving the absence of reference HTTPS datasets [32].

[25] `https://scans.io/series/443-https-tls-alexa`
[26] `http://betternet.lhs.loria.fr/datasets/https/`

Overview of the HTTPS Dataset

As explained before, our approach relies on features calculated over the TLS handshake packets and over application data packets. In this section, we explore the number of TLS handshake packets and of application data packets present in the different flows that constitute our HTTPS dataset.

Figure 5.2 illustrates that the number of TLS handshake packets is between 3 and 5 packets. We can notice close yet different distributions between Google Chrome and Firefox. The majority of flows that have 4 TLS handshake packets, 75% for Google Chrome and 66% for Firefox. Regarding the application data packets, Figure 5.3 illustrates the proportion of flows in our dataset in function of the total number of application data packets they have. We find many short HTTPS flows with only one or two application data packets (49% for Google Chrome, 53% for Firefox), while a relatively small number of flows are long with more than 9 application data packets (18% for Google Chrome, 18% for Firefox), even if they represent more in terms of the proportion of data exchanged.

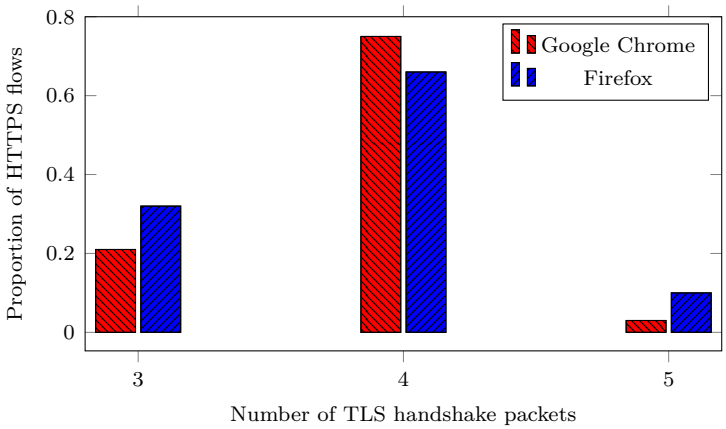


Figure 5.2: Distribution of the number of TLS handshake packets in the dataset

In Figure 5.4, we study the Cumulative Distribution Function (CDF) for the number of application data packets per flow. A significant proportion of flows (49%) consists of one or two application data packets, while flows with 9 or more application data packets account for about 18%. From another point of view, Figure 5.5, shows the contribution of the different flows regarding their number of application data packets to the total size of the dataset. The flows with one application data packet account for 15.3% of the total traffic size with Google Chrome, and 14.13% with Firefox. We notice that short flows (i.e., with less than 9 application data packets) account for almost two thirds of the total size ($\simeq$ 64%) while long flows account for a bit more than the last third ($\simeq$ 35%)). This can be explained by the specific nature of web traffic that vehiculates a lot of small objects like CSS files, HTML titles, JavaScript, etc. So these short TCP flows are almost related to simple HTTP GET and POST requests used to get the web pages' content [118]. These statistics reflects the way the dataset was generated (i.e. by loading the front page of each

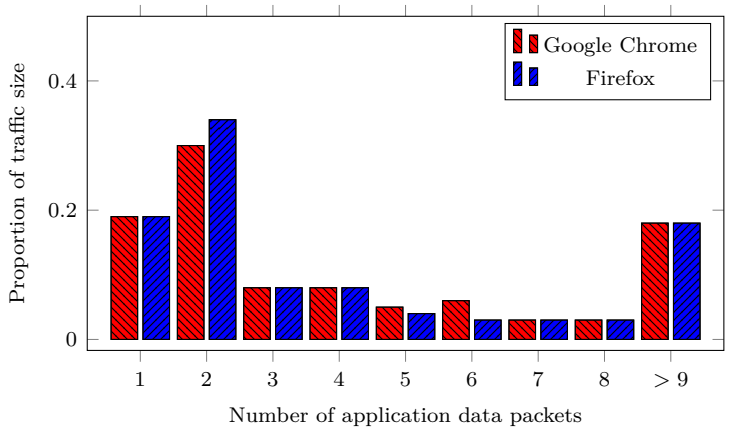


Figure 5.3: Distribution of the number of application data packets in the dataset

website) and may not be representative of HTTPS traffic at a larger scale which may hold other kinds of traffic, for example large sessions of video streaming over HTTPS. Anyway, our goal being to identify services by using very few packets, the dataset is perfectly adapted to our objective.

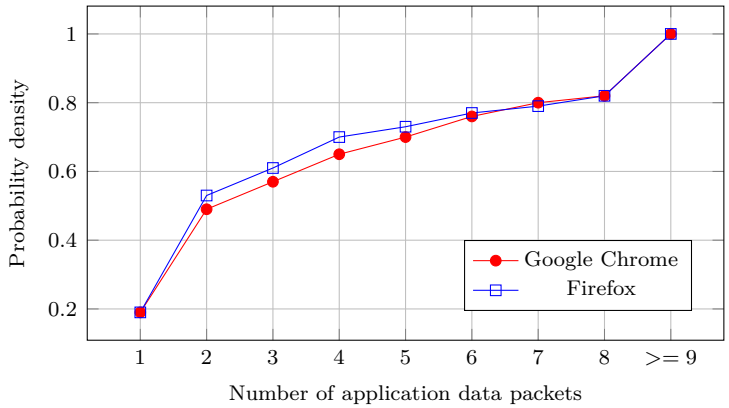


Figure 5.4: CDF for the number of application data packets per flow

5.4.2 Evaluation of the number of application data packets characterizing a flow

As shown in the previous section, the majority of flows have 3 to 4 TLS handshake packets, however there is a large variety in the number of application data packets per flow. Hence, our target is to use the minimum number of application data packets that enables an accurate identification of HTTPS services in real-time. In this part of evaluation, we need to measure how the accuracy evolves with the number of application data packets we consider.

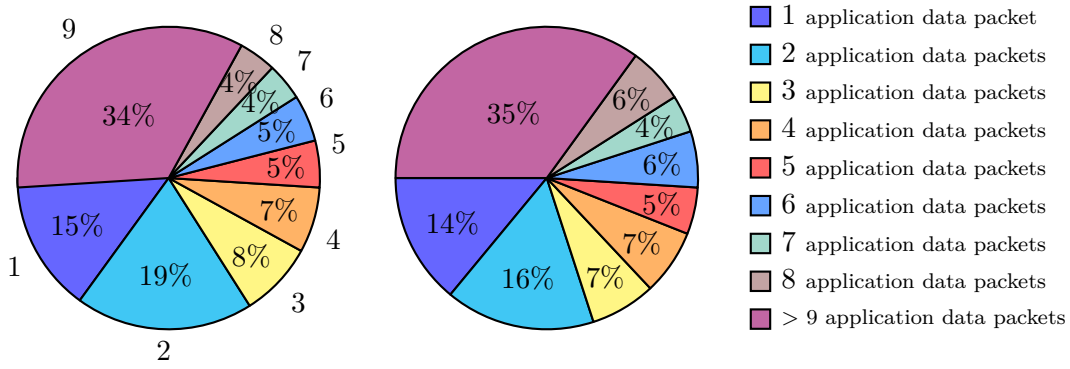


Figure 5.5: The participation in the dataset size per flow length (Left: Google Chrome, Right: Firefox)

First, we consider long HTTPS flows (i.e., flows that have at least 15 application data packets). We aim to assess the accuracy to identify HTTPS services behind such long flows using different d numbers of application data packets where $d \in \{0, 1, 2, 3, 4, 5, 6, 9, 12, 15\}$. Hence, 10 classification models were built. Each model was created using all the TLS handshake packets and d application data packets. For example, the first model consists in considering only the TLS handshake packets (i.e., no application data packets $d = 0$) to build the model and identify HTTPS services, while the second model includes TLS handshake packets plus one application data packet ($d = 1$) and so on. In the training dataset, we find 171 different HTTPS services accessed with Google Chrome and 124 HTTPS services accessed with Firefox having more than 15 application data packets. Figure 5.6 shows the relation between the number of application data packets considered by the model and the overall identification accuracy. As expected, increasing the number of application data packets improves the accuracy. However, after 9 application data packets the identification accuracy is almost stable, for Google Chrome we get $91.66\% \pm 0.57$, and for Firefox $96.33\% \pm 0.47$. Therefore, we can identify HTTPS services behind long flows quite early by using the first 9 application data packets only and without decreasing the accuracy.

However, as shown in Figure 5.3, if 18% of Google Chrome and 18% of Firefox flows have more than 9 application data packets, the vast majority of flows have less (82% for Google Chrome, 82% for Firefox). Thus, we need a classification model able to handle various number of application data packets (i.e., from 0 to 9). Figure 5.7 shows the performance of C4.5 models that have been built by progressively increasing the maximum number of application data packets considered when building the model and applied to identify all HTTPS services (1384 of Google Chrome and 890 of Firefox) including those with less than 15 application data packets. For instance, if we use up to three application data packets, HTTPS flows with one, two or three data packets are fully given for training and testing, while longer flows are limited to the three first application data packets. We can notice that with no application data packet (only handshake packets), the classification of Google Chrome and Firefox

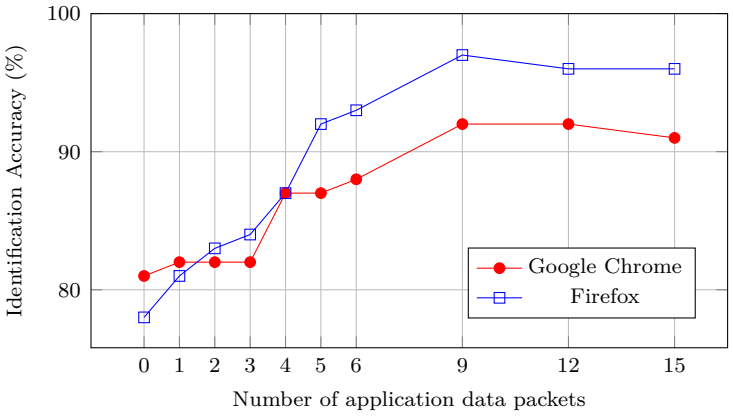


Figure 5.6: Impact of the number of application data packets considered on the identification accuracy, when identifying long flows

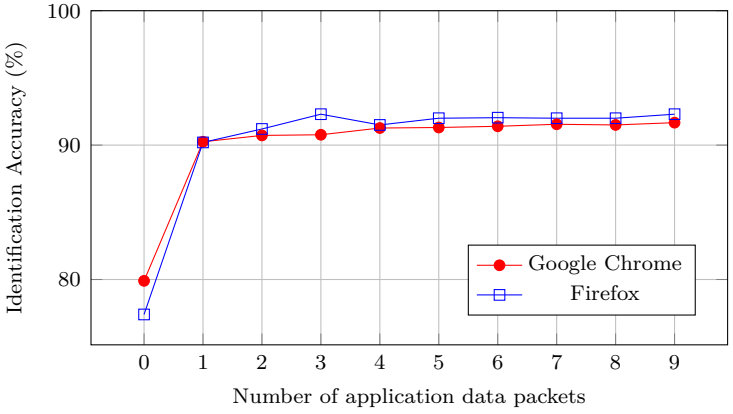


Figure 5.7: Impact of the number of application data packets considered on the identification accuracy when identifying all flows

traffic achieves 79.9% and 77.4% identification accuracy respectively. However, the accuracy can be increased to 90.25% and 90.2% if we only add one application data packet. While, if we include up to 4 application data packets, the accuracy rises respectively to 91.27% and 91.5%.

Decoupling the number of application packets used in training phase from testing phase

Here, we make a deeper analysis to assess the performance of the proposed method. Mainly, we measure the identification accuracy if we give less or more application data packets when trying to identify a service than the number used for training. In other words, we need to evaluate the case when the number of application data packets for learning and the one for testing are not equal.

Varying the number of application data packets is valid from the ML-algorithm point of view because we use flow-level features and not packet-level features. Therefore, the space used to perform the classification is the same and the number of features used does not vary with the number of application packets. For example, computing the maximum packet-size of a flow over 3 or 5 packets only counts as a single feature. However, the measured value of the feature may change regarding the number of input packets used for its computation.

Let us assume this scenario, we train a classification model to identify a given HTTPS service X using 4 handshake packets plus up to d application data packets. This means that features related to application data packets (see Table 5.1), will be computed over d or less application data packets. However, we deal with application data packets, which number is not fixed and varies according to the amount of data to be exchanged. So if the model is given features from a flow X computed over 4 handshake packets but with r application data packets, where $r \neq d$, we need to assess how much the accuracy of the model will be affected. In another words, we want to know what is the best number of packets d to consider when building the model and how another identification threshold (fewer or higher) can affect (or not) the accuracy. In this part, we still validate our approach using flows accessed by Google Chrome and Firefox. Table 5.2 shows the accuracy using a C4.5 model with up to d application packets, where $d \in \{1, 2, 3, 4, 5, 6, 9\}$ to identify HTTPS services accessed by Google Chrome, and Table 5.3 for the ones accessed by Firefox. For example, when we use $d = 5$, it means that up to 5 application data packets were used to train the model. So we include all the packets for flows with 1 to 5 application data packets, while for longer flows (i.e., with more than 5 application data packets) we use only the first five application data packets. Typically, the interesting result is to see how a short training threshold affects the identification accuracy of long flows and how a high threshold affects the identification of short flows, in order to find a good trade-off. Please note that 10-fold cross validation was not used for this experiment to limit the noise. Consequently, when training and testing thresholds are aligned, the accuracy is always of 100% but this result is not interesting and has been removed from the tables. It is also redundant with the experiment leading to Figure 5.7 that used 10-fold cross validation and aligned

testing/training thresholds.

In the first row in Table 5.2, we build a C4.5 classification model by calculating our features over the TLS handshake packets plus a single application data packet. Then this model is tested to identify flows by looking at different numbers of application data packets. The idea is to assess the identification accuracy by using only one application data packet for training. For instance the second column (in the first row) shows an accuracy of 96% for identifying flows by looking at their first two application data packets. The last column (still in the first row) gives the identification accuracy 90% for identifying flows with up to 9 application data packets, with the same training configuration. From this table, we can see that with a low training threshold the accuracy decreases when the testing threshold increases, but with a high training threshold, accuracy decreases even faster when the testing threshold lowers.

For visualizing the performance of the classification model, we use the Box plot diagram to present the overall identification accuracy under different training scenarios. The box plot is a standardized method for displaying the distribution of data based summary of five numbers: minimum, first quartile, median, third quartile, and maximum. Figure 5.8 and 5.9 show the performance of the overall identification accuracy according to the number of application data packets used to train the model for Google Chrome and Firefox respectively. For instance, the first box represents the performance of a classification model that has been built using at most one application data packets (corresponding to the first row of Table 5.2), and can be read like this: the central rectangle spans the first quartile (89%) to the third quartile (91.75%), and the segment inside the rectangle shows the median (90%) and "whiskers" above and below the box show the locations of the minimum (88%) and maximum (96%).

According to these results (for both Google Chrome and Firefox), we can see a trade-off between the number of allowed application data packets and the level of identification accuracy. Thus, we find that using a training model that has been built using up to 4 or 5 application data packets offers more balanced results to identify both short and long HTTPS flows. As shown in Table 5.2 and Table 5.3 using up to 5 application data packets achieves (75%, 73%) identification accuracy with HTTPS flow with a single application data packet and (94%, 96%) to identify ones with 9 application data packets for Google Chrome and Firefox respectively.

The conclusion is that we achieve better identification results by lowering the learning to the first 4 or 5 application data packets, which is good for our real-time identification objective.

5.4.3 Classification model generalized for multiple clients

In previous sections, we considered different classification models for each web browser; one model for services accessed by Google Chrome and another one for services accessed by Firefox. However, for building a general classification model, we want to know if the browser used to train the model affects its validity for other sources of traffic. In theory, it should not because the features are designed to

Table 5.2: Identifying HTTPS services accessed by Google Chrome

		Testing (up to d)								
		1	2	3	4	5	6	7	8	9
Training (up to d)	1	-	96%	94%	91%	90%	90%	90%	90%	90%
	2	83%	-	95%	90%	89%	90%	89%	88%	87%
	3	80%	95%	-	95%	94%	93%	92%	92%	91%
	4	72%	92%	96%	-	98%	97%	96%	94%	93%
	5	75%	89%	94%	97%	-	98%	97%	95%	94%
	6	73%	88%	92%	96%	98%	-	99%	97%	96%
	7	74%	88%	92%	95%	97%	99%	-	98%	97%
	8	73%	88%	92%	95%	96%	98%	99%	-	99%
	9	71%	86%	93%	95%	96%	97%	98%	99%	-

Table 5.3: Identifying HTTPS services accessed by Firefox

		Testing (up to d)								
		1	2	3	4	5	6	7	8	9
Training (up to d)	1	-	84%	79%	77%	76%	76%	76%	76%	76%
	2	83%	-	91%	84%	81%	80%	79%	78%	78%
	3	76%	93%	-	96%	95%	93%	92%	91%	90%
	4	73%	88%	96%	-	98%	96%	95%	94%	93%
	5	73%	88%	94%	97%	-	98%	97%	96%	96%
	6	70%	85%	91%	96%	98%	-	99%	98%	98%
	7	72%	85%	90%	94%	97%	99%	-	99%	99%
	8	73%	86%	90%	93%	96%	98%	99%	-	100%
	9	74%	86%	90%	93%	98%	98%	99%	100%	-

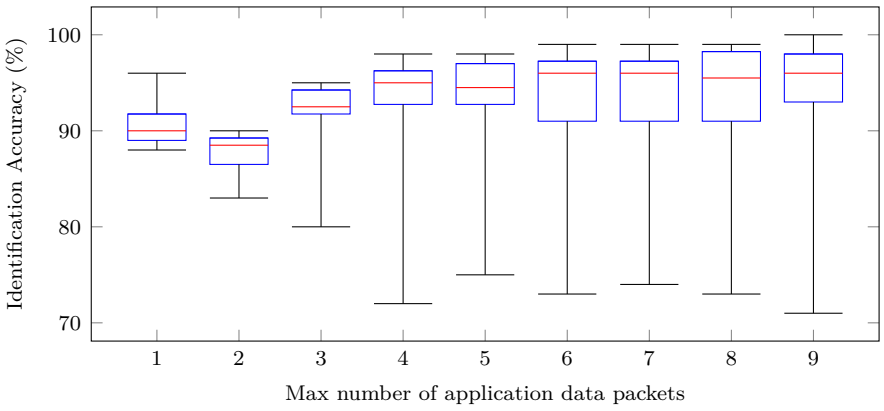


Figure 5.8: C4.5 classification models performance to identify HTTPS services accessed by Google Chrome

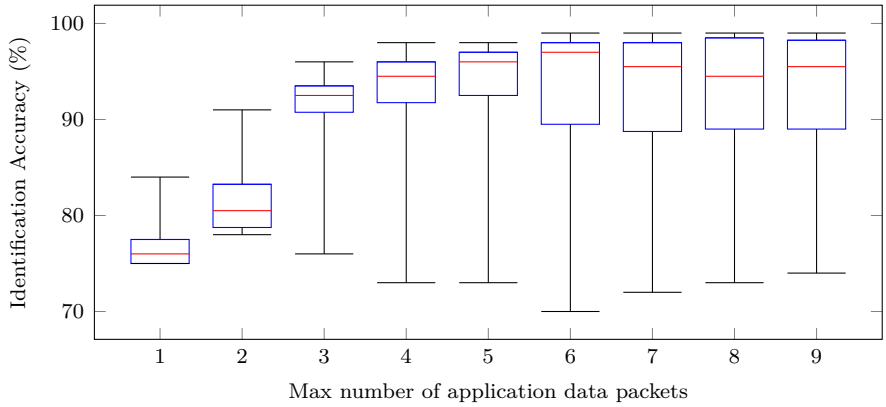


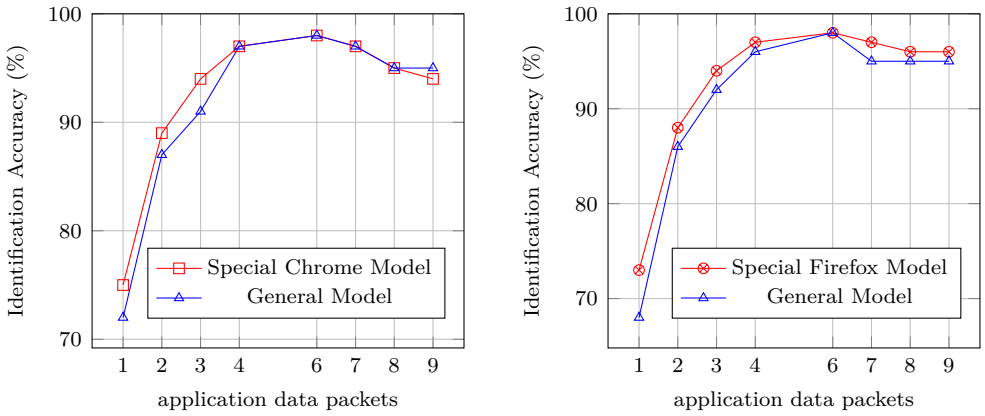
Figure 5.9: C4.5 classification models performance to identify HTTPS services accessed by Firefox

identify the service.

The authors in [119] studied the effect of using different operating systems (Windows, Linux, Mac) and different web browsers (Internet Explorer, Google Chrome, Firefox) on the identification of some HTTPS services. Their results show that there is a strong relation between the client side environment (i.e., OS and Web browser) and the identification of the service of a given HTTPS flow (offline analysis). In our experiments, we use Ubuntu as operating system and, as mentioned earlier, Google Chrome and Firefox web browsers. Thus, the questions arises as to whether it is possible to train a classification model using HTTPS flows from Google Chrome to identify HTTPS flows from Firefox or vice-versa?

The experimental results show that, if we use Firefox's HTTPS flows for training the model and Google Chrome's HTTPS flows for testing, the identification accuracy is low (19%) and this result is inline with [119]. One possible reason of this low accuracy is that each browser uses a different implementation of TLS library: Firefox uses the NSS library and Google Chrome uses the BoringSSL library as explained before in Section 2.2.4, which affects the collected features over different clients. Therefore, we need several separate models for identifying HTTPS services depending on the client. However, this approach complicates the identification in real-time, since we don't always have a prior knowledge about the used web browser. As a result, to provide a general classification model that is able to deal with HTTPS flows regardless the web browser, we have combined Google Chrome and Firefox flows for training to generate a single and more generic model. For testing, we use this general model to assess if it can identify HTTPS services in the traffic from either Google Chrome or Firefox.

Table 5.4 shows the identification accuracy of using one general model trained with both Google Chrome and Firefox flows. For testing, we use flows from each web browser separately. We observe that the behaviour of the general classification model is similar to the dedicated one. It means, with a low training threshold the accuracy decreases when testing threshold increases, but with a high training



(a) HTTPS services accessed by Google Chrome

(b) HTTPS services accessed by Firefox

Figure 5.10: Dedicated classification models vs. the general classification model

threshold, accuracy decreases even faster when the testing threshold lowers. So we have decided to consider using a training threshold of 5 application data packets to build a general model, and for testing we use all flows (i.e., we do not apply cross validation) from both Firefox and Google Chrome. Figures 5.10a, 5.10b compare the performance of the general model to identify HTTPS services accessed by Google Chrome and Firefox against a the dedicated models per web browser. We notice that the overall identification does not change significantly by using the general model. Based on these results, we conclude that it is possible to benefit from using one general model to identify HTTPS services in real-time regardless of the web browser used to access a given service if the training dataset already represents clients' diversity.

5.4.4 Extending the multi-level classification framework

In Chapter 4, we have discussed a multi-level classification, where we identify HTTPS services in two steps. In the first one, we predict the service provider (e.g., google.com), and in the second one, we identify the service itself (e.g., drive.google.com). In this section, we combine this multi-level classification process together with the real-time identification.

Figure 5.11 shows the complete resulting architecture combining both aspects. Thus, as we discussed before, during the offline phase we create a Service-Providers model (Lv1-Model) and, for each provider, we build a dedicated model (Lv2-Models). To make the framework compatible with real-time, we use the real-time features set described in Section 5.3.2.

So the final identification process will be as follows:

- (1) The flow reassembly module receives the TLS packets from the NIC and demultiplexes incoming packets to the related flows.

Table 5.4: The identification accuracy of a classification model trained with both Google Chrome and Firefox HTTPS flows

#App. for Training	Web Browser	Testing (up to d) application data packets								
		1	**2**	**3**	**4**	5	**6**	**7**	**8**	**9**
up to 1	Chrome	-	97%	95%	90%	89%	89%	88%	88%	88%
	Firefox	-	93%	88%	86%	84%	84%	84%	83%	83%
up to 2	Chrome	82%	-	95%	93%	92%	92%	92%	91%	91%
	Firefox	80%	-	94%	92%	91%	90%	89%	89%	89%
up to 3	Chrome	77%	93%	-	97%	95%	94%	93%	92%	91%
	Firefox	69%	90%	-	96%	94%	93%	91%	90%	89%
up to 4	Chrome	73%	90%	95%	-	97%	97%	95%	93%	93%
	Firefox	69%	87%	95%	-	97%	95%	93%	92%	91%
up to 5	Chrome	72%	87%	91%	97%	-	98%	97%	95%	95%
	Firefox	68%	86%	92%	96%	-	98%	95%	95%	95%
up to 6	Chrome	70%	85%	88%	95%	98%	-	99%	97%	96%
	Firefox	68%	85%	91%	95%	98%	-	98%	97%	97%
up to 7	Chrome	69%	84%	89%	93%	97%	99%	-	98%	97%
	Firefox	89%	84%	89%	92%	97%	99%	-	99%	99%
up to 8	Chrome	74%	88%	93%	95%	97%	98%	99%	-	99%
	Firefox	70%	85%	90%	96%	98%	99%	99%	-	99%
up to 9	Chrome	73%	88%	92%	95%	96%	97%	98%	99%	-
	Firefox	71%	85%	90%	92%	95%	98%	99%	100%	-

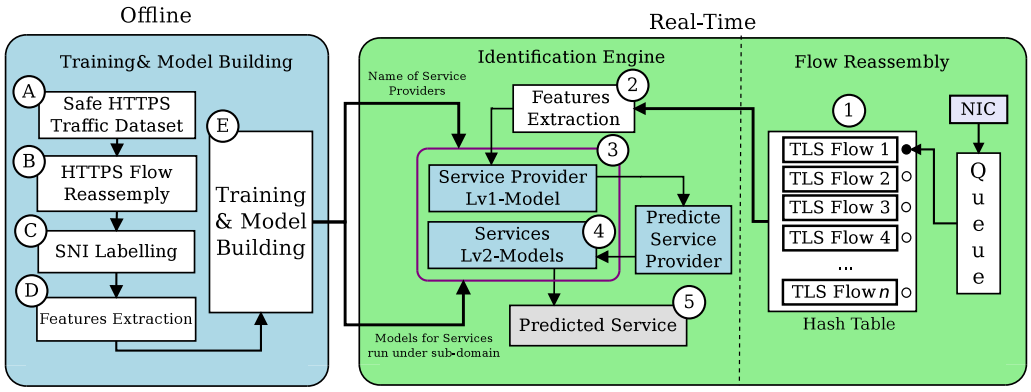


Figure 5.11: Integration of the multi-level classification for identifying HTTPS services in real-time

Table 5.5: Multi-level identification accuracy of HTTPS services

	Testing (up to d) application data packets								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Partial identification	3%	3%	2%	2%	2%	2%	1%	1%	1%
Perfect Identification	90%	90%	91%	91%	92%	92%	92%	92%	93%
Invalid identification	7%	7%	7%	7%	6%	6%	7%	7%	6%

- (2) When the number of packets in a flow reaches the threshold, the feature extraction over the flow packets is triggered.
- (3) Computed features are firstly passed to the Lv1-model (service-provider) to predict the service provider of the given HTTPS flow.
- (4) A specific model corresponding to the predicted service provider (Lv2-Models) is loaded to name the HTTPS service.
- (5) Finally, the name of the predicted service is delivered.

To perform the evaluation, we apply the multi-level classification approach to the combined traces of Firefox and Google Chrome used for training, in order to obtain a general model. According to our previous results, we use a threshold of 5 application data packets for training and we evaluate the accuracy over different testing threshold values d. We apply the cross-validation $k = 10$ to produce the final results and we use our previous evaluation approach that can cope with the multi-level classification framework (i.e., Perfect, Partial, an Invalid identification). Table 5.5 shows the evaluation results, where the perfect identification rate slightly increases when increasing the threshold value. However, after the value of 5 the accuracy does not improve significantly, which supports our choice for a threshold of 5 application data packets for both testing and training that we have selected in the previous section. Also, we can notice that the perfect identification accuracy (compared to a flat classification) does not increase significantly. However, if we consider both partial and perfect identifications, we increase the identification rate up to 3%. Identifying the service provider of a given HTTPS flow can be useful to trigger deeper investigation or other identification approaches.

5.5 Real-Time HTTPS Firewall Prototype

As a direct application of the previous results, we have developed a prototype that is able to identify HTTPS services in real-time. The prototype operates by extending the iptables/netfilter architecture. So it can be easily integrated into existing Linux middleboxes. As implementation language we use the Python for its flexibility and to keep the prototype code easier to modify and test.

5.5.1 Prototype architecture

The proposed prototype operates, as illustrated in Figure 5.12, in three levels; the first is hardware level, where packets are sniffed using iptables rules toward the ker-

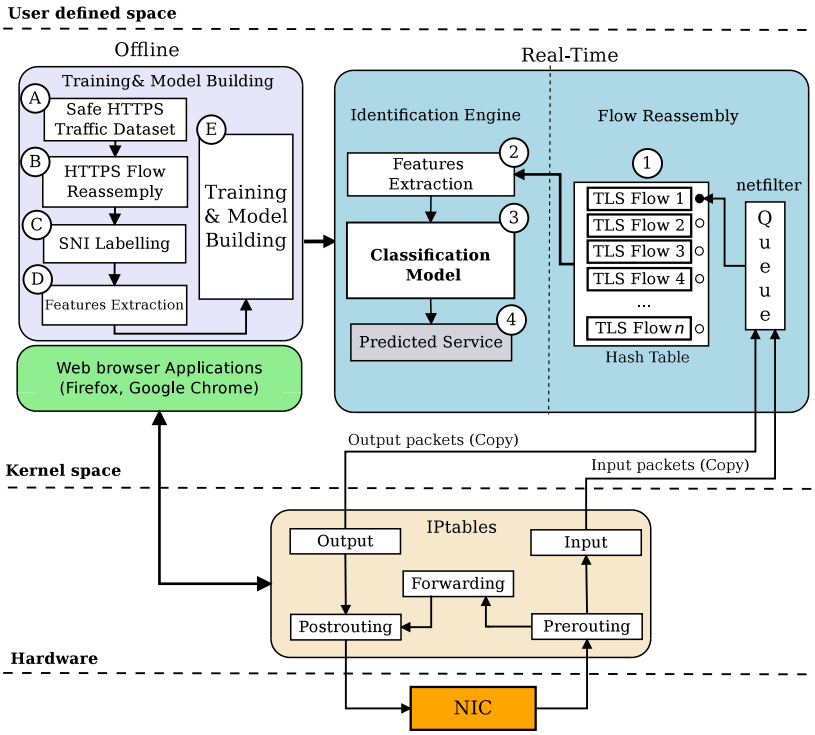


Figure 5.12: Real-time HTTPS services identification prototype flowchart

nel space (second level) and then the packets are copied in the user defined space (third level) for multiplexing. Hence, we divide the prototype into three modules; the packet capture module, the reassembly module, and the machine learning identification module.

Packet capture module

HTTPS packets are captured and forwarded from the iptables queue (i.e., network level) to the user space (i.e., application level). We benefit from the libnetfilter_queue[27] to gain access to HTTPS packets queued using iptables rules, which forwards all input packets (from the remote HTTPS service) and output packets (from the local web browser) with port number 443 to the reassembly module to make a copy, and re-inject them back to the network level to allow the communication to continue. Then the arrived packets are handled by the reassembly module.

The reassembly module

As explained before, we apply the proposed approach in [112] to optimize the HTTPS flow reassembly module. Taking into account that the monitoring system needs to handle thousands of TCP flows, we use the dictionary class to store the flows' record.

[27] `https://github.com/chifflier/nfqueue-bindings`

The dictionary is accessed by a key that should be unique for each flow (i.e., the 4-tuples). Based on that, the TCP flow is presented as a list of TCP packet objects, which are stored based on their arrival order and can be accessed using the flow-key. Upon receiving TCP packets from the netfilter queue, the system calculates a hash key using the 4-tuple (source IP address, source port, destination IP address, destination port). This key is used as a key to find the corresponding flow in the dictionary object.

Algorithm 3 Flow reassembly algorithm

```
1: P =ReceivePacket()
2: Info=ParsePacket(P)  // the key is calculated
3: if Info.SYN then // a new SYN packet is arrived
4:     R =Get-BCRT-Record(Info.key)
5:     if R is invalid and ! Info.ACK then
6:         Insert-BCRT-Record(Info) // a new flow is started
7:     else R is valid
8:         if Info.ACK then
9:             UpdateRecord(Info)
10:            Insert-FCRT-Record(Info)
11:        end if
12:    end if
13: else// non SYN packet arrived
14:    Insert-FCRT-Record(Info)
15: end if
```

Algorithm 3 explains the flow reassembly process. The module has two record sets (i.e., dictionary objects), the first is Budding Connection Record Table (BCRT) that is used to store a SYN packet until the related ACK packet arrived, then the SYN packet is moved to the Formal Connection Record Table (FCRT) to create a new flow record. Then, for each TCP packet without a SYN flag, the module looks for the related flow record in the FCRT using the packet's key. If the key is found in the FCRT, then the packet is added to the corresponding flow, otherwise it is discarded. As soon as the number of packets in a given flow reaches pre-defined threshold limit, the identification module is triggered.

Machine learning identification module

The identification module has been implemented using the scikit-learn[28] library. The module's tasks are: to extract features (given in Section 5.3.2) for a given HTTPS flow reconstructed by the reassembly module, and to run the C4.5 algorithm to predict the HTTPS service of the flow.

[28] `http://scikit-learn.org/stable/index.html`

5.5.2 Performance evaluation

In this section, we present the results of a few performance metrics of our prototype. As illustrated in Figure 5.13, the evaluation environment consists in a client machine configured to automatically open HTTPS websites, while our prototype runs on a PC with 64-bit Intel Xeon Processor@3.80GHz, DDR3 32GB memory and OS Linux-Mint 17.0.

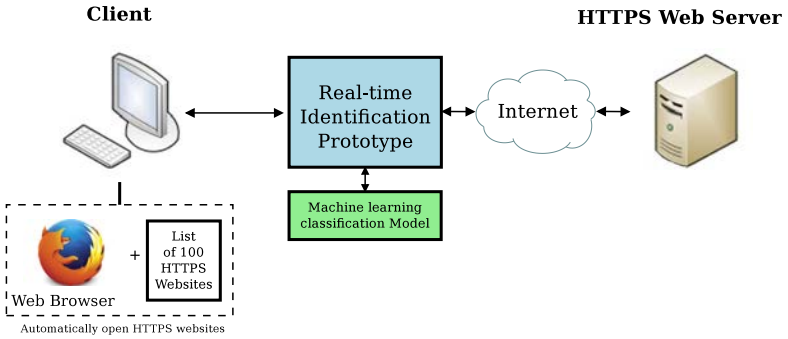


Figure 5.13: The prototype evaluation environment

First, we measure the amount of delay needed for identifying HTTPS services in real-time using TLS handshake packets and 9 application data packets (i.e., the worst case). This delay must be short, so we do not alter the browsing experience of users that access HTTPS websites through our identification framework. In our prototype, we have two potential sources of delay: the packet capture process using iptables and the machine learning identification procedure (i.e., the features extraction and the execution of the C4.5 algorithm). Concerning the first source of delay, as we just make a copy of captured packets without any treatment, we consider that this delay is negligible, and we only focus on the second one. The results show that we have in average 2.02±0.86 ms delay to treat a single HTTPS flow, from the trigger of the flow reassembly to the final identification. According to [33], using HTTPS instead of HTTP adds an extra delay of 500 ms when using fibre links, while over 3G mobile network the delay is up to 1.2 seconds (depending on the RTT value). So the delay added by the framework is negligible for users and web applications because it is several orders of magnitude lower than the latency introduced by the protocol itself.

Second, to measure the identification accuracy of our approach in real-time, we need to integrate the overhead delay of capturing packets and reconstructing flows, to make it sure that the training and testing will be done in the same conditions. Unfortunately, the public dataset was not captured through the framework and consequently this traffic cannot be used to evaluate the real-time prototype but only the model on which it is based (see Section 5.4.2). The main issue with the public dataset set is the collection process. TCP packets are dumped from the NIC to the dump file with no further delay. However in our prototype, TCP packets are captured from the NIC to netfilter queue (i.e., iptable) then forwarded to our framework

for copying and re-injected back to the NIC. This process add a signification delay for the Inter packets arrival time, and this will make features calculated for training and the one for testing are not aligned.

Therefore, we have built a traffic collector that collects traffic at the same point than our real-time identification prototype. The collector receives packets from the iptables queue, parses the packets into flows and extract features that will be saved for building a new training dataset. For this new evaluation focused on the prototype implementation, we have built a small dataset using our collector visited the top 100 HTTPS websites. The training was configured to use a threshold of 5 application data packets. The collected dataset contains 109513 records related to 582 HTTPS services. We use the cross-validation (k=10) and C4.5 to evaluate the identification accuracy. The results show that we have 95.96% of accuracy. This higher accuracy can be explained because the top 100 HTTPS services are easier to differentiate than the top 1000 used to evaluate the models: the more services, the higher the probability than a couple of them show similar features.

Third, we study the prototype's flow throughput. This is the number of flow per second (fps) the framework can reassemble and identify. From the previous experiment, we measure that the prototype is able to process at a rate of 31874 fps. This throughput is higher than the flow throughput given in [102](24997 fps) thanks to using a smaller number of packets per flow before starting the identification process. Based on these results, the prototype performs well with a low delay overhead and a high flow throughput, and also good identification accuracy which is perfectly in line with the previous evaluation of the model alone. In conclusion, all tests confirm that our prototype satisfies the real-time requirement for monitoring HTTPS services.

5.6 Discussion and Analysis

In this chapter, we have tackled the real-time identification of HTTPS services. We have proposed a novel approach to identify services based on the TLS handshake packets and a few number of application data packets. The proposed approach was evaluated from different side views. First, we have presented an overview of our public HTTPS dataset collection to participate in solving the issue of a reference HTTPS dataset. Second, we have studied the sufficient number of application data packets needed to get a high identification accuracy. The results show that we can identify HTTPS services using the TLS handshake packet and up to 9 application data packets without loosing accuracy. So we need at most 13 packets (4 TLS handshake packets + 9 application data packets) to recognize the HTTPS service behind a flow, what is very early in the session compared to others values in related work that need 70 or 100 packets. Third, we have evaluated the relation between the identification accuracy and different threshold values. Experimental results show that the best overall classification is obtained by considering up to 5 application data packets. This lower number allows a better recognition of the large amount of short flows with 90% overall accuracy.

Then we have presented the general classification model to identify flows which are either accessed by Google Chrome or Firefox web browsers. We show that using a mixed dataset results in a minor drop in the identification accuracy compared to using dedicated classification models for each web browser. Also, we combined the multi-level classification to our the real-time identification. Even if the accuracy does not improve significantly, we shall benefit from the enhanced flexibility and the extendibility of that framework. As a direct application of our results, we have presented a real-time identification prototype to identify HTTPS services. Experimental tests show that our approach has a low overhead delay of 2.02 msec and high flow throughput of 31874 fps.

The findings of this chapter can be considered from different sides. From one side, we can identify HTTPS services in real-time by using a small number of application data packets in addition to the TLS handshake packets. From another side, we can also consider to have a trade-off between security and accuracy by tuning the threshold value. So if accessing to a forbidden service generates a high risk, then we can set the model to identify services using only handshake packets to prevent any exchange of application data packets, even if this configuration may affect other legitimate services due to its lower accuracy. On the opposite, for other kinds of application focused on QoS or accounting, for example while monitoring video streaming or file download over HTTPS connections that generate long flows, we may consider more packets for identification, the small amount of data exchanged being considered less critical. Hence, by using 9 application data packets these services can be detected with approximately 92 % accuracy. We achieved a similar accuracy as in [23] (93%) to identify services in HTTPS, but we do it in real-time, and we identify many types of services, not just web-mail ones. So we consider that this contribution significantly improve to the state-of-the-art of HTTPS monitoring approaches.

For the security point of view, if we use 9 application data packets to gain a higher accuracy, the amount of leaked data is still expected and under control. After removing the headers, the payload of a TCP data packet is 1460 Bytes. So up to 13 kB (9*1460) can be exchanged overall. But based on our dataset, we found that the average size of the encrypted payload transmitted from client to server is only 306 Bytes, and 331 Bytes in the opposite direction. Hence, if we allow 9 application data packets to be exchanged, the average amount of data will be in reality around (9*318) Bytes or 2.8 kB, before we are able to identify the flow and to decide if the flow is allowed or not.

Chapter 6

General Conclusion

Many reports demonstrate the continuous increasing of HTTPS traffic, in parallel with the global intention from key role players in the Internet Industry to deploy encryption everywhere. One of the main motivation toward the "encryption rush" is the raising awareness of users for privacy questions as response to the NSA revelations. As a reaction, the web browsing through HTTPS becomes the norm. However, encryption undermines the effectiveness of network traffic identification approaches, and so of monitoring activities which are known as a fundamental component to network management and security. For instance, to apply security policies, it is important to have a proper understanding of the services behind the encrypted traffic travelling through the network. In this dissertation, we focused on the encrypted traffic generated by HTTPS services. There are two challenges with monitoring HTTPS services; the first is the proper level of identification for HTTPS traffic, and the second is the reliability and the privacy issues affecting currently used methods. Therefore, our research question was to find reliable and privacy-preserving techniques to monitor, with a proper level of identification, the increasing amount of HTTPS traffic.

Thus, our contributions covered four main points; first we have studied and proposed a taxonomy of literature work in the field of HTTPS traffic identification and monitoring based on the necessary steps required to identify HTTPS services. From the most basic level of protocol identification (TLS, HTTPS), to the finest identification of precise services. Second, we have provided an in depth view of a recent technique for HTTPS services monitoring and filtering that is based on the Server Name Indication (SNI) a field of the TLS handshake, and which is used in many firewalls and traffic shaper solutions. We have identified several flaws in the current usage of SNI-based HTTPS monitoring that make it unreliable and proposed improvements to this method. Third, we have claimed that a service-level identification of HTTPS traffic is appropriate for operational needs. Thus, we have proposed a robust framework to identify the accessed HTTPS services from a traffic dump, without relying neither on a header field nor on the payload content. The framework has been built based on a new set of statistical features combined with machine learning algorithms and a novel multi-level classification approach. We have evaluated

this framework using a real-work traffic collected from well-controlled environment. Fourth, the real-time identification of HTTPS services has been addressed. We have proposed improvements to our framework to monitor HTTPS services in real-time. By extracting statistical features on the TLS handshake packets and progressively on few application data packets, we can identify HTTPS services very early in the session. We have evaluated the real-time monitoring approach though extensive experiments and an implementation of a real-time HTTPS firewall prototype.

6.1 Achievements

Analysing the literature work in the field of HTTPS traffic identification shows that the identification of TLS and HTTPS protocol is well mastered. More precise levels (i.e., service-level) keep being challenging despite recent advances in this research field. Indeed, website fingerprinting is too fine-grained (page-level) and identifying the application-type (HTTPS) is too coarse-grained, and are not adapted to operational needs. Moreover, HTTPS proxy, while efficient, has other issues related to users' privacy that it breaks, and other solutions may succeed to identify a given service (webmail, maps) but are too specific.

New machine learning methods have been used widely in related work for identifying encrypted traffic applications. However, each proposed method is applied over a private dataset, what makes any comparison between such methods unfair. The main reason of the lack of public HTTPS dataset is the privacy and security risk of publishing raw dataset that contains users activity over HTTPS websites. Therefore, in the current existence dataset, the TLS layer information are truncated. To participate in solving this problem, we have published our private HTTPS dataset of crawling top 1000 websites, with full payload and TLS information.

In order to the assess the reliability of SNI-based method, we have demonstrated why relying on SNI extension for identifying and monitoring HTTPS services is utterly flawed and can be easily cheated. We have shown that the SNI-based method has two weakness points, regarding (1) backward compatibility and (2) multiple services using a single certificate. We have implemented our proof of concept exploiting SNI-based method weaknesses inside a web browser plug-in for Firefox called "Escape" and that allows users to bypass such HTTPS filtering in their network. We have conducted the first investigation of SNI extension deployment with a large set of web servers accessed over HTTPS connections, where the results show that 92% of the HTTPS websites included in the study can be accessed with a fake SNI. To mitigate this issue, a novel DNS-based approach to validate SNI values has been proposed and consists in verifying the relation between the actual destination server and the claimed value of the SNI extension by relying on a trusted DNS service. The experimental results show the ability to overcome the shortage of SNI-based monitoring, while having a small false positive rate (1.7%) and a small overhead (15 ms). As reported in literature work, the SNI field is used for different goals beyond its initial purpose. It is not limited to security monitoring, but also provides an easy and direct solution for ISPs to name the services of a given HTTPS flow, like for

the T-Mobile's service BingeOn. Thus, we believe our improvement to SNI-based monitoring is valuable and can be applied to enhance the less-sensitive SNI-based solutions.

We argue that the SNI-based method cannot be used effectively to enforce security polices on encrypted traffic by verifying a single header field. Even the enhanced SNI-based method is still unable to precisely identify a given HTTPS service if it shares a SSL certificate with others. To provide a more robust method for identifying HTTPS services, we have proposed a complete framework based on machine learning algorithms to identify what are the HTTPS services accessed in a traffic dump. Our framework includes several innovations making the identification accurate and the framework scalable. The first is a new set of statistical features extracted from the encrypted payload of reconstructed HTTPS connections. The second is a multi-level classification technique, where in the top level the main service provider is identified, while in the second level the precise services are recognized. We defined an evaluation method on a private dataset and we obtained a high level of accuracy of 93.10% for identifying HTTPS services in a given traffic dump. A direct application of the proposed framework could be to investigate the access of prohibited or suspicious services over HTTPS during a forensic analysis: services that pretend to be from a given provider but are not classified as such may be manually investigated when searching for the source of a security incident.

Finally, we have addressed the real-time identification of HTTPS service as our most challenging contribution. We improved our framework to cope with real-time identification requirements such as the number of packets, features and the proper machine learning algorithm. So we extract statistical features on TLS handshake packets and progressively on application data packets. Using these few first packets (a round 10 packets) makes it possible to identify HTTPS services very early in the session. Experiments results over a significant and open dataset show that our method offers up to 92% accuracy thanks to the C4.5 algorithm, our features set and the small number of packets we use. A real prototype implementation and tests highlight that our method only adds a low overhead delay (2.02ms) but can achieve a high flow processing throughput (31874 flows/s).

According to the aforementioned contributions, we believe that we can monitor services in HTTPS traffic with a good level of accuracy and high confidence either in an offline or in a real-time manner thanks to machine learning methods, well-tuned statistical features, the novel classification paradigm and the optimized usage of the TLS handshake and application data packets. Despite the more challenging problem of monitoring HTTPS services without decryption, we still achieve a better accuracy than similar related work using machine-learning. From one side, the proposed solution provides more robust method than the SNI-based method, while from the another side, it preserves the users' privacy on the contrary of solutions using an HTTPS proxy. Our solution makes it possible to have a new balance between security and privacy and so making a significant progress for the network management community. From another perspective, the proposed approaches in this thesis can even be combined as bricks of LEGO. It means, one could build

a monitoring system that uses the improved SNI-based verification to detect flows with a fake SNI, and then delegates to the real-time identification framework the prediction of the actual HTTPS services behind them.

6.2 Future Work

This thesis opens many perspectives in different research directions for employing machine learning for enhanced network monitoring. First, we present complementary studies for our solution. Second, we discuss the possible extensions of this thesis as future work.

Our solution needs to be enhanced to identify HTTPS service against traffic morphing techniques (that are used by anonymous network like Tor), which may change the statistical fingerprint of flows and make them unrecognizable by our framework. Indeed, encryption is a basic level of obfuscation against direct traffic inspection, while other approaches such as traffic morphing add extra protection against the potential leakage of meta-data like network features. Hence, we envisage to improve our solution to handle this challenge. For instance, by adding a pre-step to detect the existence traffic morphing on a flow and then by selecting a subset of features that are more robust against traffic morphing. At another level, HTTPS will be progressively replaced by the HTTP/2 protocol [120] in the upcoming years. Thus, we will have to adapt our architecture to monitor services in HTTP/2 traffic. This could be challenging according to the ability of HTTP2 to multiplex and stream several contents in a same flow.

On-the-fly retraining of the identification engine is also interesting, so the machine leaning-based identification framework can be automatically updated alongside the changes in a given service. That will make the HTTPS services monitoring solution always efficient without the need for a full periodic re-training process. Also, we are interested in studying how the HTTPS service's statistical signature is robust against the changes in network environment. We have already demonstrated that the framework is client agnostic if well trained, but the local network characteristics should also be considered. For instance, the authors in [121] discuss how to isolate the noise from the network conditions, such as the size of the TCP congestion window and the variable routing queueing time. They aim to extract inter-packet time features that are really related to the application itself. Their results show that they are able to remove the noise and make the inter-packet time features independent of the network conditions. As future work, it would make sense to assess how far our features depend on the network conditions and how to avoid this type of noise. This would allow the learning phase to occur everywhere whereas has been local so far.

As an extension to this thesis, a comparison between different machine learning methods over a reference HTTPS dataset and within a unified experimental environment is still missing. The open HTTPS dataset we made and published is a first step in this direction, yet not sufficient. Such a study will make future efforts to improve or invent new machine learning approaches based on a solid ground of knowledge.

We believe that the existence of such a knowledge will add more confidence for testing and deploying machine learning methods in the network security industry other than staying in research cycle. This perspective is strongly supported by a Cisco's recent survey (August 2016) [122] about the trends in network security monitoring, where 29% of experts in network security support the research on security solutions that use machine learning approaches.

This thesis also can be extended by employing hardware acceleration techniques for packets parsing, flows reconstruction and machine learning algorithms execution. As explained in [104], the FPGA technology allows processing network traffic with high throughput. Also, would accelerate machine learning based traffic classifiers. The authors in [123] use FPGA to improve the performance of a C4.5 decision tree classifier (the algorithm we applied in our frameworks). Their experimental results show high throughput of 550 Gb/s (assuming a 40 Byte packet size). Therefore, we envisage that we could use this technique to build a new generation of hardware HTTPS firewall.

List of Publications

The contributions of this thesis have been published in a number of international conferences and were presented in different event as detailed below:

Selective International Conferences

- Wazen Shbair, Thibault Cholez, Jerome François, and Isabelle Chrisment, Improving SNI-based HTTPS Security Monitoring, IEEE International Workshop on Security Testing and Monitoring (STAM2016) co-located with The 36th IEEE International Conference on Distributed Computing Systems (ICDCS 2016), July 2016, Nara, Japan. Link
- Wazen Shbair, Thibault Cholez, Jerome François, and Isabelle Chrisment, A Multi-level Framework to Identify HTTPS Services, The 28th IEEE/IFIP Network Operations and Management Symposium (NOMS), April 2016, Istanbul, Turkey. Link
- Wazen Shbair, Thibault Cholez, Antoine Goichot, Isabelle Chrisment, Efficiently Bypassing SNI-based HTTPS Filtering, The 14th IFIP/IEEE Symposium on Integrated Network and Service Management (IM),Experience Session, October 2015, Ottawa, Canada. Link

Talks, Demo and Posters

International Audience:

- Wazen Shbair, Thibault Cholez, Jerome François, Isabelle Chrisment, Security analytics: geo-labeled Android logs and bypassing HTTPS filtering, la 1ère rencontre franco-allemande académie-industrie sur la cybersécurité, Nancy, France, December 2016 [Demo].
- Wazen Shbair, Thibault Cholez, Jerome François, Isabelle Chrisment, Real-Time Identification of Services in HTTPS Traffic, Masdin workshop, Metz, France, December 2016 [Talk].
- Wazen Shbair, Thibault Cholez, Jerome François, Isabelle Chrisment, Security Monitoring of HTTPS Traffic, Workshop on Intrusion Detection and NetFlow Analysis (WIN), Interdisciplinary Center for Security, Reliability and Trust (SnT), Luxembourg, July 2016 [Talk].

- Wazen Shbair, Thibault Cholez, Jerome François, and Isabelle Chrisment, HTTPS Traffic Classification, Network Machine Learning Research Group (NMLRG), Internet Research Task Force (IRTF), Buenos Aires, Argentina, April 2016 [Talk]. (Link)

National Audience:

- Wazen Shbair, Thibault Cholez, A.Goichot, Isabelle Chrisment, Efficiently Bypassing SNI-based HTTPS Filtering, Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information (RESSI),Toulouse, France, May 2016 [Talk].
- Wazen Shbair, Thibault Cholez, Isabelle Chrisment , An Open and Flexible Architecture for Monitoring HTTPS Traffic, Annual PhD students Conference IAEM Lorraine 2015, LORIA, Nancy, France, October 2015 [Poster] (Link)

Software and Dataset

- *Escape*:
  A Firefox web browser add-on, which we designed and developed in the context of evaluating HTTPS traffic monitoring techniques. It offers the ability to bypass SNI-based firewalls that use SNI to monitor HTTPS websites.(Link)
- *A Multi-level Framework to Identify HTTPS Services:*
  We present our framework, which includes machine learning techniques and a multi-level classification approach to identify the accessed HTTPS services in a given traffic dump without decryption. (Link)
- *Open HTTPS dataset*:
  Acknowledging the absence of a public HTTPS traffic dataset, we provide a HTTPS dataset contains full HTTPS raw PCAP files captured while crawling the top 1000 HTTPS websites. The scan was made on a daily basis over two weeks, using both Google Chrome and Mozilla Firefox Web browsers. (Link)
- *Prototype for real-time identification of HTTPS services*:
  Our prototype operates by extending the iptables/netfilter architecture. It receives and demultiplexes the arriving HTTPS packets to a related flow. As soon as the number of packets in a given flow reaches a threshold, the identification engine extracts the features and runs the C4.5 machine learning algorithm to predict the HTTPS service of the flow. (Link)

List of Figures

List of Tables

Glossary

ACL : Access Control List
AJAX : Asynchronous JavaScript And XML
CA : Certificate Authority
CDN : Content Delivery Networks
CFS : Correlation-based Filter Selection
DPI : Deep Packet Inspection
DNS : Domain Name System
FTP : File Transfer Protocol
HMM : Hidden Markov Model
HTTP : Hyper Text Transfer Protocol
HTTPS : Hyper Text Transfer Protocol Secure
IANA : Internet Assigned Numbers Authority
IDS : Intrusion Detection System
IETF : Internet Engineering Task Force
IMAP : Internet Message Access Protocol
IPsec : IP security
ISP : Internet Service Provider
MAC : Message Authentication Codes
MitM: Man-in-the-Middle
PGP : Pretty Good Privacy
PHP : Personal Home Page
PKI : Public Key Infrastructure
PKT : Private Communication Technology
POP : Post Office Protocol
QoS : Quality of Service
RFC : Request For Comments
ROC : Receiver Operating Characteristics
SMTP : Simple Mail Transfer Protocol
SNI : Server Name Indication
SSH : Secure SHell
SSL : Secure Socket Layer
SVM : Support Vector Machine
TLS : Transport Layer Security
ToR : The Onion Routing
URL : Universal Resource Locator
VPN : Virtual Private Network

Bibliography

[1] C. McCarthy *et al.*, "An investigation on identifying SSL traffic," in *Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on*. IEEE, 2011, pp. 115–122.

[2] M. Aertsen, M. Korczyński, G. Moura, S. Tajalizadehkhoob, and J. v. d. Berg, "No domain left behind: is let's encrypt democratizing encryption?" *arXiv preprint arXiv:1612.03005*, 2016.

[3] "Cisco 2016 Annual Security Report," http://www.cisco.com/c/dam/assets/offers/pdfs/cisco-asr-2016.pdf, (Online, retrieved: 30 May, 2016).

[4] "Report 2015-0832 from the French regulatory authority for telecommunications (ARCEP). Structure de l'usage de la bande passante des reseaux d'acces a internet sur le territoire francais," "http://www.arcep.fr/uploads/tx_gsavis/15-0832.pdf", ([In French], retrieved: 07 July, 2015).

[5] "Global internet phenomena spotlight: Encrypted internet traffic," https://www.sandvine.com/downloads/general/global-internet-phenomena/2015/encrypted-internet-traffic.pdf, (Online, retrieved: 11 May, 2016).

[6] "HTTPS adoption has reached the tipping point," https://www.troyhunt.com/https-adoption-has-reached-the-tipping-point/, (Online, retrieved: 3 February, 2017).

[7] "Google Transparency Report, HTTPS Usage," https://www.google.com/transparencyreport/https/metrics/?hl=en, (Online, retrieved: 18 December, 2016).

[8] "January 2014 web server survey," "http://news.netcraft.com/archives/2014/01/03/january-2014-web-server-survey.html", (Online, retrieved: 29 August, 2016).

[9] T. Mori, T. Inoue, A. Shimoda, K. Sato, S. Harada, K. Ishibashi, and S. Goto, "Statistical estimation of the names of https servers with domain name graphs," *Computer Communications*, 2016.

[10] "HTTPS as a ranking signal," https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html, (Online, retrieved: 3 September, 2016).

[11] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, "ACAS: automated construction of application signatures," in *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*. ACM, 2005, pp. 197–202.

[12] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," *EURASIP Journal on Information Security*, vol. 2016, no. 1, p. 1, 2016.

[13] R. Bortolameotti, A. Peter, M. H. Everts, and D. Bolzoni, "Indicators of malicious SSL connections," in *Network and System Security*. Springer, 2015, pp. 162–175.

[14] "HTTPS working for malicious users," https://securelist.com/blog/research/57323/https-working-for-malicious-users, (Online, retrieved: 3 September, 2016).

[15] P. Chen, N. Nikiforakis, L. Desmet, and C. Huygens, "Security analysis of the chinese web: How well is it protected?" in *Proceedings of the 2014 Workshop on Cyber Security Analytics, Intelligence and Automation*. ACM, 2014, pp. 3–9.

[16] S. Pukkawanna, G. Blanc, J. Garcia-Alfaro, Y. Kadobayashi, and H. Debar, "Classification of SSL servers based on their SSL handshake for automated security assessment," in *2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS)*. IEEE, 2014, pp. 30–39.

[17] "Internet filtering and monitoring in your business," http://www.currentware.com/whitepapers/Internet-Filtering-and-Monitoring-in-your-BusinessWhite-Paper.pdf, (Online, retrieved: 19 April, 2016).

[18] X. Xu, Z. M. Mao, and J. A. Halderman, "Internet censorship in China: Where does the filtering occur?" in *Proceedings of the 12th International Conference on Passive and Active Measurement*, ser. PAM'11. Springer, 2011, pp. 133–142.

[19] "French national agency for the security of information systems (ANSSI), Security recommendation regarding the analysis of HTTPS traffic," http://www.ssi.gouv.fr/uploads/IMG/pdf/NP_TLS_NoteTech.pdf, (Online, retrieved: 30 May, 2016).

[20] "FireEye SSL intercept appliance, expose attacks hiding in SSL traffic," https://www.fireeye.com/content/dam/fireeye-www/global/en/products/pdfs/ds-ssl-intercept.pdf, (Datasheet, Online, retrieved : 22 May, 2016).

[21] "Explicit trusted proxy in HTTP/2.0 (internet-draft)," https://tools.ietf.org/html/draft-loreto-httpbis-trusted-proxy20-01, 2014, (Online, retrieved: 29 August, 2016).

[22] R. Alshammari *et al.*, "Machine learning based encrypted traffic classification: identifying SSH and Skype," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on.* IEEE, 2009, pp. 1–8.

[23] D. Schatzmann, W. Mühlbauer, T. Spyropoulos, and X. Dimitropoulos, "Digging into HTTPS: flow-based classification of webmail traffic," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement.* ACM, 2010, pp. 322–327.

[24] M. Liberatore and B. N. Levine, "Inferring the source of encrypted HTTP connections," in *Proceedings of the 13th ACM conference on Computer and communications security.* ACM, 2006, pp. 255–263.

[25] D. Herrmann, R. Wendolsky, and H. Federrath, "Website fingerprinting: attacking popular privacy enhancing technologies with the multinomial naïve-bayes classifier," in *Proceedings of the 2009 ACM workshop on Cloud computing security.* ACM, 2009, pp. 31–42.

[26] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel, "Website fingerprinting in onion routing based anonymization networks," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society.* ACM, 2011, pp. 103–114.

[27] B. Miller, L. Huang, A. D. Joseph, and J. D. Tygar, "I know why you went to the clinic: Risks and realization of HTTPS traffic analysis," in *Privacy Enhancing Technologies.* Springer, 2014, pp. 143–163.

[28] "The right to privacy in the digital age," http://www.ohchr.org/EN/NewsEvents/Pages/Therighttoprivacyinthedigitalage.aspx, (Online, retrieved: 18 December, 2016).

[29] A. Callado, C. Kamienski, G. Szabó, B. P. Gerö, J. Kelner, S. Fernandes, and D. Sadok, "A survey on internet traffic identification," *Communications Surveys & Tutorials, IEEE*, vol. 11, no. 3, pp. 37–52, 2009.

[30] S. Valenti, D. Rossi, A. Dainotti, A. Pescapè, A. Finamore, and M. Mellia, "Reviewing traffic classification," in *Data Traffic Monitoring and Analysis.* Springer, 2013, pp. 123–147.

[31] M. Finsterbusch, C. Richter, E. Rocha, J.-A. Muller, and K. Hanssgen, "A survey of payload-based traffic classification approaches," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 2, pp. 1135–1156, 2014.

[32] P. Velan, M. Čermák, P. Čeleda, and M. Drašar, "A survey of methods for encrypted traffic classification and analysis," *International Journal of Network Management*, vol. 25, no. 5, pp. 355–374, 2015.

[33] D. Naylor, A. Finamore, I. Leontiadis, Y. Grunenberger, M. Mellia, M. Munafò, K. Papagiannaki, and P. Steenkiste, "The cost of the S in HTTPS," in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 2014, pp. 133–140.

[34] M. Husák, M. Cermak, T. Jirsik, and P. Celeda, "Network-based HTTPS client identification using SSL/TLS fingerprinting," in *10th International Conference on Availability, Reliability and Security (ARES)*. IEEE, 2015, pp. 389–396.

[35] M. Zhang, W. John, K. Claffy, and N. Brownlee, "State of the art in traffic classification: A research review," in *PAM Student Workshop*, 2009, pp. 3–4.

[36] G. Canfora and C. A. Visaggio, "A set of features to detect web security threats," *Journal of Computer Virology and Hacking Techniques*, pp. 1–19, 2016.

[37] C. Huang, J. Liu, Y. Fang, and Z. Zuo, "A study on web security incidents in China by analyzing vulnerability disclosure platforms," *Computers & Security*, vol. 58, pp. 47–62, 2016.

[38] W. Stallings, *Cryptography and network security: principles and practices*. Pearson Education India, 2006.

[39] R. Oppliger, *SSL and TLS: Theory and Practice*. Artech House, Inc., 2009, [Book].

[40] T. Dierks and E. Rescorla, "RFC 5246 - the transport layer security (TLS) protocol version 1.2," http://tools.ietf.org/html/rfc5246, 2008.

[41] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," Internet-Draft, IETF, Tech. Rep., October 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tls-tls13-10

[42] T. Bingmann, "Speedtest and comparsion of open-source cryptography libraries and compiler flags," https://panthema.net/2008/0714-cryptography-speedtest-comparison/, 2008, (Online, retrieved: 28 August, 2016).

[43] M. D. Network, "Mozilla network security services (nss)," https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS.

[44] "Boringssl," https://boringssl.googlesource.com/boringssl/, (Online, retrieved: 13 December, 2016).

[45] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate Ecosystem," in *Proceedings of the 2013 Conference on Internet Measurement Conference*, ser. IMC '13. ACM, 2013, pp. 291–304.

[46] "X.509 public key certificates," https://msdn.microsoft.com/en-us/library/windows/desktop/bb540819(v=vs.85).aspx, (Online, retrieved: 13 December, 2016).

[47] L. Bernaille and R. Teixeira, "Early recognition of encrypted applications," in *Passive and Active Network Measurement*. Springer, 2007, pp. 165–175.

[48] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[49] C. Liu, G. Sun, and Y. Xue, "DRPSD: An novel method of identifying SSL/TLS traffic," in *World Automation Congress (WAC), 2012*. IEEE, 2012, pp. 415–419.

[50] S.-M. Kim, Y.-H. Goo, M.-S. Kim, S.-G. Choi, and M.-J. Choi, "A method for service identification of SSL/TLS encrypted traffic with the relation of session ID and Server IP," in *Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific*. IEEE, 2015, pp. 487–490.

[51] G.-L. Sun, Y. Xue, Y. Dong, D. Wang, and C. Li, "An novel hybrid method for effectively classifying encrypted traffic," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[52] C. V. Wright, F. Monrose, and G. M. Masson, "On inferring application protocol behaviors in encrypted network traffic," *The Journal of Machine Learning Research*, vol. 7, pp. 2745–2769, 2006.

[53] R. Holz, L. Braun, N. Kammenhuber, and G. Carle, "The SSL landscape: a thorough analysis of the x. 509 PKI using active and passive measurements," in *Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference*. ACM, 2011, pp. 427–444.

[54] Z. Durumeric, J. Kasten, M. Bailey, and J. A. Halderman, "Analysis of the HTTPS certificate ecosystem," in *Proceedings of the 2013 Internet Measurement Conference*. ACM, 2013, pp. 291–304.

[55] A. Hilts, "Scandinista! analyzing TLS handshake scans and HTTPS browsing by website category," in *Workshop on Surveillance & Technology. PETS*, 2015.

[56] P. Velan, J. Medková, T. Jirsík, and P. Čeleda, "Network traffic characterisation using flow-based statistics," in *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*. IEEE, 2016, pp. 907–912.

[57] M. Husák, M. Čermák, T. Jirsík, and P. Čeleda, "HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting," *EURASIP J. Inf. Secur.*, vol. 2016, no. 1, pp. 30:1–30:14, Dec. 2016. [Online]. Available: http://dx.doi.org/10.1186/s13635-016-0030-7

[58] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 2, pp. 23–26, 2006.

[59] O. Levillain, M. Tury, and N. Vivet, "concerto: A methodology towards reproducible analyses of TLS datasets," *IACR Cryptology ePrint Archive*, vol. 2017, p. 20, 2017. [Online]. Available: http://eprint.iacr.org/2017/020

[60] G. D. Bissias, M. Liberatore, D. Jensen, and B. N. Levine, "Privacy vulnerabilities in encrypted HTTP streams," *Lecture notes in computer science*, vol. 3856, p. 1, 2006.

[61] L. Lu, E.-C. Chang, and M. C. Chan, "Website fingerprinting and identification using ordered feature sequences," in *Computer Security–ESORICS 2010*. Springer, 2010, pp. 199–214.

[62] A. Pironti, P.-Y. Strub, and K. Bhargavan, "Identifying website users by TLS traffic analysis: New attacks and effective countermeasures," INRIA, Tech. Rep., 2012.

[63] H. Cheng and R. Avnur, "Traffic analysis of SSL encrypted web browsing," https://people.eecs.berkeley.edu/~daw/teaching/cs261-f98/projects/final-reports/ronathan-heyning.ps, (Online, retrieved: 14 September, 2016).

[64] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 191–206.

[65] V. Berg, "SSL traffic analysis attacks," http://2011.ruxcon.org.au/2011-talks/ssl-traffic-analysis-attacks/, (Online, retrieved: 31 May, 2016).

[66] R. Dubin, A. Dvir, O. Pele, and O. Hadar, "I know what you saw last minute-encrypted HTTP adaptive video streaming title classification," *arXiv preprint arXiv:1602.00490*, 2016.

[67] M. Korczynski and A. Duda, "Markov chain fingerprinting to classify encrypted traffic," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 781–789.

[68] A. Molavi Kakhki, A. Razaghpanah, A. Li, H. Koo, R. Golani, D. Choffnes, P. Gill, and A. Mislove, "Identifying traffic differentiation in mobile networks," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. ACM, 2015, pp. 239–251.

[69] A. M. Kakhki, F. Li, D. Choffnes, E. Katz-Bassett, and A. Mislove, "Bingeon under the microscope: Understanding T-Mobiles zero-rating implementation," in *Proceedings of the 2016 workshop on QoE-based Analysis and Management of Data Communication Networks*. ACM, 2016, pp. 43–48.

[70] J. Clark and P. C. van Oorschot, "SoK: SSL and HTTPS: revisiting past challenges and evaluating certificate trust model enhancements," in *IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 511–525.

[71] S. J. Murdoch and R. Anderson, "Tools and technology of internet filtering," *Access Denied: The Practice and Policy of Global Internet Filtering, ed. Ronald J. Deibert, John Palfrey, Rafal Rohozinski, and Jonathan Zittrain (Cambridge, MA: MIT Press)*, pp. 57–72, 2008.

[72] "Configuring HTTPS inspection with Forefront threat management gateway (TMG)," http://www.isaserver.org/articles-tutorials/configuration-general/Configuring-HTTPS-Inspection-Forefront-Threat-Management-Gateway-TMG-2010.html, (Online, retrieved: 4 January, 2016).

[73] H. Abelson, R. Anderson, S. M. Bellovin, J. Benalob, M. Blaze, W. Diffie, J. Gilmore, P. G. Neumann, R. L. Rivest, J. I. Schiller, and B. Schneier, "The risks of key recovery, key escrow, and trusted third-party encryption," *World Wide Web J.*, vol. 2, no. 3, pp. 241–257, Jun. 1997.

[74] B. D. McGinnes, "Cleaning a HTTPS feed," https://www.cla.asn.au/documents/CleanFeedHTTPS.pdf, 2010, (Online, retrieved: 2 February, 2017).

[75] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik *et al.*, "Factorization of a 768-bit RSA modulus," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 333–350.

[76] D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J. A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta *et al.*, "Imperfect forward secrecy: How Diffie-Hellman fails in practice," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 5–17.

[77] S. Farrell and H. Tschofenig, "RFC7258: Pervasive monitoring is an attack," 2014. [Online]. Available: https://tools.ietf.org/html/rfc7258

[78] M. A. Caloyannides, "Is privacy really constraining security or is this a red herring?" *Security & Privacy, IEEE*, vol. 2, no. 4, pp. 86–87, 2004.

[79] M. Dilman and D. Raz, "Efficient reactive monitoring," *Selected Areas in Communications, IEEE Journal on*, vol. 20, no. 4, pp. 668–676, 2002.

[80] Bloxx, "Protecting your network against risky SSL traffic," https://www.bloxx.com/media/1360/bloxx_whitepaper_protectnetwork_fromssl_us.pdf, 2012, (Online, retrieved: 11 May, 2016).

[81] "Managing encrypted traffic with Blue Coat solutions," https://www.bluecoat.com/solutions/managing-ssl-and-https-traffic, (Online, retrieved: 15 September, 2015).

[82] S. Blake-Wilson, M. Nystrom, D. Hopwood, J. Mikkelsen, and T. Wright, "RFC 3546: Transport layer security (TLS) extensions," 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3546.txt

[83] L. Völker, M. Noe, O. P. Waldhorst, C. Werle, and C. Sorge, "Can internet users protect themselves? challenges and techniques of automated protection of HTTP communication," *Computer communications*, vol. 34, no. 3, pp. 457–467, 2011.

[84] D. Eastlake, "RFC 6066-the transport layer security (TLS) extensions: Extension definitions," 2011. [Online]. Available: http://tools.ietf.org/html/rfc6066

[85] T. Flach, P. Papageorge, A. Terzis, L. Pedrosa, Y. Cheng, T. Karim, E. Katz-Bassett, and R. Govindan, "An internet-wide analysis of traffic policing," in *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*. ACM, 2016, pp. 468–482.

[86] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman, "A search engine backed by Internet-wide scanning," in *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, Oct. 2015.

[87] T. Wang, X. Cai, R. Nithyanand, R. Johnson, and I. Goldberg, "Effective attacks and provable defenses for website fingerprinting." in *USENIX Security*, 2014, pp. 143–157.

[88] P. Foremski, C. Callegari, and M. Pagano, "DNS-Class: immediate classification of IP flows using DNS," *International Journal of Network Management*, vol. 24, no. 4, pp. 272–288, 2014.

[89] T. Mori, T. Inoue, A. Shimoda, K. Sato, K. Ishibashi, and S. Goto, "SFMap: Inferring services over encrypted web flows using dynamical domain name graphs," in *Traffic Monitoring and Analysis: 7th International Workshop, TMA 2015 Proceedings*. Springer, 2015, pp. 126–139.

[90] M. Prandini, M. Ramilli, W. Cerroni, and F. Callegati, "Splitting the HTTPS stream to attack secure web connections," *IEEE Security and Privacy*, vol. 8, no. 6, pp. 80–84, 2010.

[91] K. Marsolo, S. Parthasarathy, and C. Ding, "A multi-level approach to SCOP fold recognition," in *Bioinformatics and Bioengineering, BIBE 2005. Fifth IEEE Symposium on*. IEEE, 2005, pp. 57–64.

[92] Y. Okada, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka, "Comparisons of machine learning algorithms for application identification of encrypted traffic," in *Machine Learning and Applications and Workshops (ICMLA), 2011 10th International Conference on*, vol. 2. IEEE, 2011, pp. 358–361.

[93] Y. Kumano, S. Ata, N. Nakamura, Y. Nakahira, and I. Oka, "Towards real-time processing for application identification of encrypted traffic," in *Computing, Networking and Communications (ICNC), 2014 International Conference on*. IEEE, 2014, pp. 136–140.

[94] F. Dehghani, N. Movahhedinia, M. R. Khayyambashi, and S. Kianian, "Real-time traffic classification based on statistical and payload content features," in *Intelligent Systems and Applications (ISA), 2010 2nd International Workshop on*. IEEE, 2010, pp. 1–4.

[95] M. A. Hall and L. A. Smith, "Feature subset selection: a correlation based filter approach," in *1997 International Conference on Neural Information Processing and Intelligent Information Systems*. Springer, 1997, pp. 855–858.

[96] H. Kim, K. C. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, "Internet traffic classification demystified: myths, caveats, and the best practices," in *Proceedings of the 2008 ACM CoNEXT conference*. ACM, 2008, p. 11.

[97] Y. Xue, D. Wang, and L. Zhang, "Traffic classification: Issues and challenges," in *2013 International Conference on Computing, Networking and Communications (ICNC)*, 2013, pp. 545–549.

[98] C. McCarthy *et al.*, "An investigation on identifying SSL traffic," in *Computational Intelligence for Security and Defense Applications (CISDA), 2011 IEEE Symposium on*. IEEE, 2011, pp. 115–122.

[99] J. Li, S. Zhang, Y. Xuan, and Y. Sun, "Identifying Skype traffic by random forest," in *2007 International Conference on Wireless Communications, Networking and Mobile Computing*. IEEE, 2007, pp. 2841–2844.

[100] K. Singh, S. Agrawal, and B. Sohi, "A near real-time IP traffic classification using machine learning," *International Journal of Intelligent Systems and Applications*, vol. 5, no. 3, p. 83, 2013.

[101] N. Williams, S. Zander, and G. Armitage, "A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 5, pp. 5–16, 2006.

[102] S. S. L. Pereira, J. L. d. C. e Silva, and J. E. B. Maia, "NTCS: A real time flow-based network traffic classification system," in *10th International Conference on Network and Service Management (CNSM) and Workshop*. IEEE, 2014, pp. 368–371.

[103] C. Gu, S. Zhang, and Y. Sun, "Realtime encrypted traffic identification using machine learning," *Journal of Software*, vol. 6, no. 6, pp. 1009–1016, 2011.

[104] T. Groléat, S. Vaton, and M. Arzel, "High-speed flow-based classification on FPGA," *International journal of network management*, vol. 24, no. 4, pp. 253–271, 2014.

[105] B. Hullár, S. Laki, and A. Gyorgy, "Early identification of peer-to-peer traffic," in *2011 IEEE International Conference on Communications (ICC)*. IEEE, 2011, pp. 1–6.

[106] R. Alshammari and A. N. Zincir-Heywood, "Can encrypted traffic be identified without port numbers, IP addresses and payload inspection?" *Computer networks*, vol. 55, no. 6, pp. 1326–1350, 2011.

[107] W. Dai and W. Ji, "A mapreduce implementation of C4.5 decision tree algorithm," *International Journal of Database Theory and Application*, vol. 7, no. 1, pp. 49–60, 2014.

[108] J. Zhang and M. Zulkernine, "Network intrusion detection using Random Forests," in *Third Annual Conference on Privacy, Security and Trust, Proceedings*, October 12-14, 2005.

[109] S. Marchal, J. François, R. State, and T. Engel, "Phishscore: Hacking phishers' minds," in *Network and Service Management (CNSM), 2014 10th International Conference on*. IEEE, 2014, pp. 46–54.

[110] O. Levillain, A. Ébalard, B. Morin, and H. Debar, "One year of SSL internet measurement," in *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 2012, pp. 11–20.

[111] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2005, [Book].

[112] B. Xiong, C. Xiao-su, and C. Ning, "A real-time TCP stream reassembly mechanism in high-speed network," *South West Jiaotong University*, vol. 17, no. 3, pp. 185–191, 2009.

[113] G. Maiolini, A. Baiocchi, A. Iacovazzi, and A. Rizzi, "Real time identification of SSH encrypted application flows by using cluster analysis techniques," in *International Conference on Research in Networking*. Springer, 2009, pp. 182–194.

[114] R. Bar-Yanai, M. Langberg, D. Peleg, and L. Roditty, "Realtime classification for encrypted traffic," in *International Symposium on Experimental Algorithms*. Springer, 2010, pp. 373–385.

[115] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *Communications Surveys & Tutorials, IEEE*, vol. 10, no. 4, pp. 56–76, 2008.

[116] J. Li, S. Zhang, Y. Lu, and J. Yan, "Real-time P2P traffic identification," in *IEEE GLOBECOM 2008-2008 IEEE Global Telecommunications Conference*. IEEE, 2008, pp. 1–5.

[117] C. Gu, S. Zhang, and Y. Sun, "Realtime encrypted traffic identification using machine learning," *Journal of Software*, vol. 6, no. 6, pp. 1009–1016, 2011.

[118] L. Qian and B. E. Carpenter, "A flow-based performance analysis of TCP and TCP applications," in *2012 18th IEEE International Conference on Networks (ICON)*. IEEE, 2012, pp. 41–45.

[119] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele, "Analyzing HTTPS encrypted traffic to identify user operating system, browser and application," *arXiv preprint arXiv:1603.04865*, 2016.

[120] M. Belshe, BitGo, R. Peon, Google, M. Thomson, and Mozilla, "RFC 7540 - Hypertext Transfer Protocol Version 2 (HTTP/2)," 2015. [Online]. Available: https://tools.ietf.org/html/rfc7540

[121] M. Jaber, R. Cascella, and C. Barakat, "Can we trust the inter-packet time for traffic classification?" Inria, Research Report, Jun. 2011. [Online]. Available: https://hal.inria.fr/inria-00546794

[122] J. Oltsik, "Network security monitoring trends," http://www.cisco.com/c/dam/en/us/products/collateral/security/stealthwatch/esg-research-insight.pdf, (Online, retrieved: 21 December, 2016).

[123] D. Tong, L. Sun, K. Matam, and V. Prasanna, "High throughput and programmable online traffic classifier on FPGA," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2013, pp. 255–264.

Abstract
Service-Level Monitoring of HTTPS Traffic

The global trend toward an encrypted web quickly made HTTPS traffic the dominant share of Internet traffic. However, the continuous increase of HTTPS traffic comes with new challenges related to the network management and security analysis of encrypted traffic. Therefore, there is an essential need for new methods to monitor, with a proper level of identification, the increasing number of HTTPS traffic. In this thesis, we first investigate a recent technique for HTTPS services monitoring that is based on the Server Name Indication (SNI) field of the TLS handshake, and which has been recently used in many firewall solutions. We show thanks to a web browser add-on called Escape that we designed and implemented, how this method can be easily bypassed. Experiments show that that 92% of the HTTPS websites included in the study can be accessed with a fake SNI. To mitigate this issue, we propose a novel DNS-based approach to validate the claimed value of SNI. The evaluation show the ability to overcome the shortage of SNI-based monitoring while having a small false positive rate (1.7%) and small overhead (15 ms). Second, we claim that a service-level identification of HTTPS traffic is appropriate for operational needs. Thus, we propose a robust framework to identify the accessed HTTPS services from a traffic dump, without relying neither on a header field nor on the payload content. The framework uses a new set of statistical features combined with machine learning algorithms and a novel multi-level classification approach. Our evaluation based on real traffic shows that we can identify encrypted HTTPS services with 93% identification accuracy. Our solution can be used in a new network forensic analysis. Third, the real-time identification of HTTPS services has been tackled. We have improved our framework to monitor HTTPS services in real-time. Indeed, observing relevant events in network traffic have to be done as soon as possible to avoid costly inefficient functioning or even downtime. By extracting statistical features over the TLS handshake packets and a few application data packets (10 packets), we can identify HTTPS services very early in the session. The obtained results and a prototype implementation show that our method offers 92% identification accuracy, high HTTPS flow processing throughput (31874 flow/second), and a low overhead delay of 2.02 ms. According to the aforementioned contributions, we believe that we can monitor services in HTTPS traffic with a good level of accuracy and high confidence either in offline or real-time manners.

Keywords: HTTPS, Security monitoring, Traffic analysis, Firewall

Résumé

Identification des Services dans le Trafic HTTPS

Depuis 2012, la proportion du trafic Internet relative au web chiffré ne cesse de croître, faisant de HTTPS le principal protocole applicatif aujourd'hui. Cependant, ce développement rapide du trafic HTTPS engendre de nouveaux défis liés à la supervision et à l'analyse du trafic web chiffré. En effet, pour pouvoir gérer les réseaux, et ainsi garantir leur performance et leur sécurité, il est nécessaire d'avoir un certain niveau de connaissance sur le trafic transitant, ce qui est rendu difficile par un chiffrement systématique. Il y a donc un besoin essentiel de nouvelles méthodes permettant de superviser, avec un niveau approprié d'identification, la proportion croissante de trafic HTTPS. Dans cette thèse, nous dressons tout d'abord un bilan des différentes techniques d'identification de trafic et constatons l'absence de solution permettant une identification du trafic HTTPS à la fois précise (a niveau des services) et respectueuse de la vie privée des utilisateurs (sans déchiffrement).

Nous nous intéressons dans un premier temps à une technique récente, néanmoins déjà déployée, permettant la supervision du trafic HTTPS grâce à l'inspection du champ SNI, extension du protocole TLS. Nous montrons que deux stratégies permettent de contourner cette méthode, la rendant peu fiable. Grâce à un plug-in pour navigateur implantant ces stratégies de contournement, nous montrons que 92% des sites web les plus populaires sont accessibles malgré un filtrage basé sur le champ SNI. Comme remédiation, nous proposons une procédure de vérification supplémentaire basée sur un serveur DNS de confiance. Les résultats expérimentaux montrent que cette solution pragmatique est efficace, avec seulement 1,7% de faux positifs, mais elle ne peut être appliquée pour distinguer les services d'un même fournisseur.

Ensuite, nous proposons une architecture qui permet l'identification des services dans le trafic HTTPS, sans déchiffrement, en se basant sur l'apprentissage automatique des caractéristiques intrinsèques aux flux réseau complets d'un service, plutôt que sur des informations issues des champs protocolaires. Nous avons ainsi défini un nouvel ensemble de caractéristiques statistiques combinées avec une identification à deux niveaux, identifiant d'abord le fournisseur de services, puis le service en particulier avec une précision de 93%, selon notre évaluation à partir de trafic réel.

Enfin, nous améliorons cette architecture afin de permettre l'identification du trafic en temps réel en ne considérant que les premiers paquets des flux plutôt que leur totalité. L'apprentissage se fait ainsi sur l'initialisation de la connexion TLS et les cinq premiers paquets applicatifs, permettant une identification très tôt dans la session. Pour évaluer notre approche, nous avons constitué un dataset comportant les flux complets de chargement des principaux sites web et l'avons rendu public pour comparaison. La précision obtenue atteint 92%. Nous présentons également un prototype de logiciel reconstituant les flux HTTPS en temps réel puis les identifiant. Celui-ci peut traiter 31874 flux par seconde et n'ajoute que 2,02ms de délai.

Mots-clés: HTTPS, Supervision, Sécurité, Analyse de trafic, Pare-feu

Résumé de la thèse en français

Identification des Services dans le Trafic HTTPS

1 Introduction

Toutes les études récentes montrent que le Web repose de plus en plus sur le protocole HTTPS. D'un côté, HTTPS fournit aux utilisateurs des propriétés essentielles de sécurité et de confidentialité pour leurs communications, mais de l'autre côté, il soulève des défis et des problèmes importants pour les organisations, défis liés à la gestion du trafic chiffré (filtrage, détection d'anomalie, etc.). Les solutions de supervision actuelles du trafic HTTPS ont soit des problèmes éthiques, liés à la rupture du secret des communications par un déchiffrement systématique du trafic par des équipements intermédiaires (HTTPS proxy), soit des problèmes de fiabilité en ne permettant pas une identification précise du trafic. Par conséquent, notre question de recherche consiste à trouver des techniques pour superviser le trafic HTTPS qui sont à la fois efficaces, robustes, tout en respectant la vie privée des utilisateurs.

Dans cette thèse, notre contribution quant à la détection des services dans le trafic HTTPS est organisée en quatre sections. Dans la section 2, nous étudions l'état de l'art concernant l'identification du trafic HTTPS en fonction des étapes nécessaires aboutissant à l'identification des services. La section 3 évalue ensuite une méthode de supervision récente du trafic HTTPS qui se base sur le champs SNI de TLS et qui est mise en œuvre dans de nombreux systèmes de pare-feu. Dans la section 4, nous présentons une méthode plus robuste pour identifier les services HTTPS basée sur l'apprentissage automatique des caractéristiques intrinsèques du trafic des divers services. La section 5 améliore cette approche pour permettre l'identification en temps réel du trafic HTTPS. Enfin, une conclusion générale et les perspectives de recherche sont présentées.

2 Supervision réseau et protocole HTTPS

Le protocole HTTPS permet aux applications Web de sécuriser les communications HTTP en les transportant via TLS, ce qui rend difficile pour un tiers de déduire des informations sur l'interaction des utilisateurs avec un site Web. Cependant, HTTPS ne signifie pas que le contenu d'un site Web n'est pas malveillant ou accepté dans

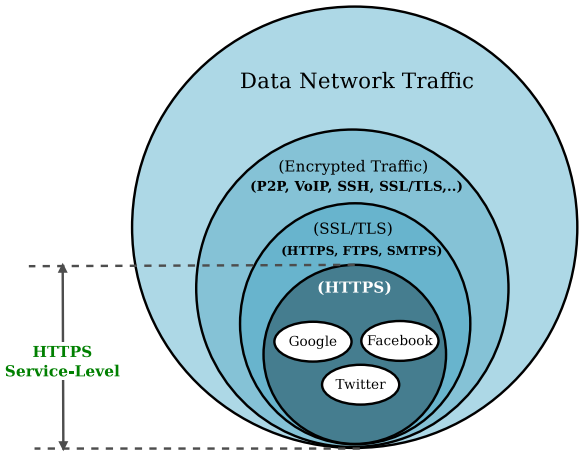


Figure 1: Granularity of Internet traffic classification toward HTTPS service monitoring

un réseau donné. Ainsi, nous avons étudié et proposé une taxonomie des travaux existants dans le domaine de l'identification et de la supervision du trafic HTTPS en fonction des étapes nécessaires pour identifier les services accédés, comme le montre la Figure 1.

Nous avons constaté qu'il existe des méthodes efficaces pour identifier le trafic TLS qui sont basées sur l'exploitation de la structure propre au protocole (PDU). L'identification du protocole HTTP parmi d'autres usages de TLS peut également être réalisée avec un niveau de précision acceptable, mais le défi réel reste l'identification des services à l'intérieur même du trafic HTTPS. Il semble exister des approches prometteuses pour ce problème comme les méthodes basées sur l'analyse du champ SNI de TLS ou sur l'apprentissage automatique des caractéristiques du trafic. Cependant, nous avons remarqué que la fiabilité de la solution basée sur SNI n'a jamais été évaluée rigoureusement, bien qu'étant déjà implantée dans plusieurs produits.

3 Évaluation et amélioration de la supervision HTTPS par SNI

Dans cette section, nous étudions une technique récente de supervision et de filtrage des services HTTPS inspectant le champ SNI de TLS et qui a été récemment implantée dans de nombreux firewalls et gestionnaires de trafic. Nous montrons que cette méthode comporte deux points faibles qui concernent (1) la rétro-compatibilité requise par le standard et (2) les services multiples utilisant un même certificat. Nous avons mis en œuvre un démonstrateur exploitant ces faiblesses à travers un module appelé Escape pour le navigateur Web Firefox et qui permet aux utilisateurs de contourner un tel filtrage HTTPS mis en place dans leur réseau, en remplaçant la valeur du champs SNI à la volée lors de l'établissement d'une connexion TLS par une autre valeur non filtrée par le pare-feu. Nous avons évalué l'étendue de cette

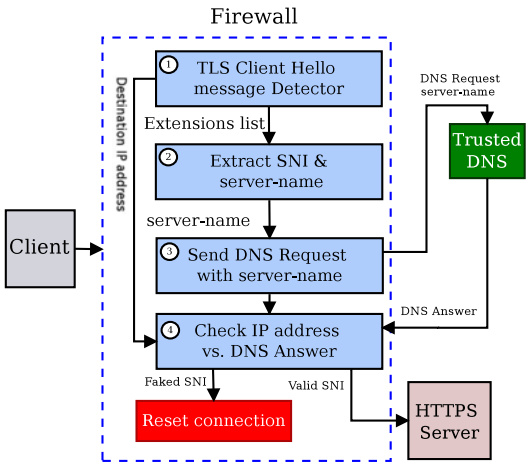


Figure 2: SNI verification system using DNS

faiblesse en interrogeant un échantillon significatif de serveurs Web populaires accessibles par HTTPS. Les résultats montrent que 92% des sites Web inclus dans l'étude peuvent être consultés avec un faux SNI, alors qu'aucun des pare-feu considérés n'a pu détecter le contournement. Pour atténuer ce problème, nous avons proposé une amélioration du mécanisme de détection actuel ajoutant une vérification supplémentaire mettant à contribution le service DNS afin de valider le champ SNI renseigné par l'utilisateur. Ainsi, la relation entre le serveur de destination contacté (adresse IP) et la valeur revendiquée par le SNI (nom DNS) peut être vérifiée par un servur DNS de confiance comme le montre la Figure 2. Les résultats expérimentaux attestent de l'efficacité de cette solution qui permet de limiter efficacement cette faiblesse des pare-feu utilisant SNI, en induisant un faible taux de faux positif (1,7%) et un faible surcoût (15 ms).

Cependant, notre approche, si elle permet de détecter efficacement de faux SNI, n'est toujours pas capable d'identifier un service HTTPS donné s'il partage son certificat SSL avec d'autres services. En effet, un service voisin peut être renseigné dans l'extension SNI pour passer le filtrage et utiliser les mêmes serveurs que le service bloqué, rendant l'usurpation indétectable avec notre approche. Ce défi est la principale motivation pour le chapitre suivant qui propose une méthode plus robuste pour identifier les services dans le trafic HTTPS car désormais fonction de caractéristiques intrinsèques au trafic du service lui-même, plutôt que sur d'un champ protocolaire particulier.

4 Un framework à plusieurs niveaux pour l'identification de services HTTPS

L'objectif de cette section est de proposer une méthode de supervision fiable du trafic HTTPS qui ne repose ni sur le déchiffrement du trafic, ni sur des champs d'en-tête protocolaire peu fiables comme SNI, mais plutôt sur les propriétés du trafic lui-même.

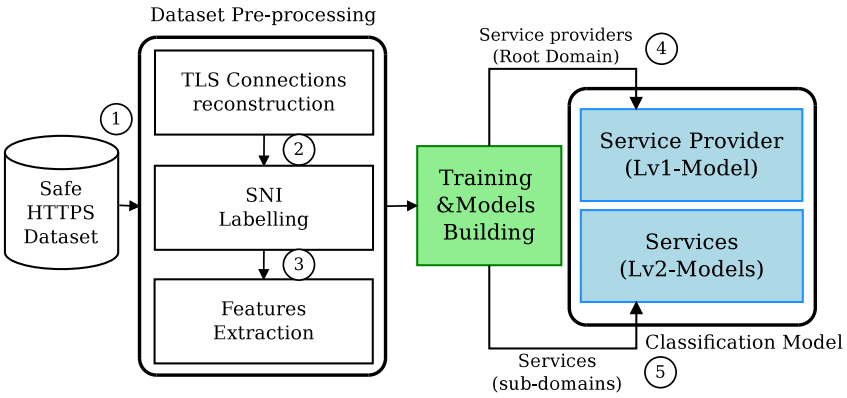


Figure 3: The workflow of the HTTPS traffic identification framework regarding models building

Par conséquent, nous présentons un framework qui permet une identification efficace des services HTTPS grâce à des techniques d'apprentissage automatique tout en respectant un certain niveau de vie privée. À notre connaissance, nous sommes les premiers à proposer un tel framework pouvant offrir un compromis entre les besoins de supervision des opérateurs et la vie privée des utilisateurs. Nous considérons dans un premier temps l'identification de services dans des traces complètes précédemment collectées, par exemple dans un cadre d'analyse forensic. Le framework que nous proposons inclue plusieurs innovations améliorant la précision de l'identification et rendant facilitant l'apprentissage. La première consiste en un nouvel ensemble de caractéristiques statistiques extraites des données applicatives chiffrées dans connexions HTTPS reconstruites. La seconde est une technique de classification à plusieurs niveaux, comme illustré dans la Figure 3, où le fournisseur de service est identifié à un premier niveau, tandis que les services précis de celui-ci sont différenciés grâce à un modèle particulier au second niveau. Nos résultats montrent qu'un haut niveau de précision de 93.10 % est atteint, plus 2,9 % d'identification partielle (fournisseur de service reconnu mais pas le service). Comme mentionné dans l'état de l'art, la classification du trafic chiffré à l'aide de techniques d'apprentissage automatique a été principalement appliquée dans le contexte de l'analyse hors ligne, où des flux de réseau complets sont disponibles pour l'apprentissage et la classification.

Cette approche reste limitée dans son application car elle ne permet pas la gestion du trafic chiffré en transit, ce qui reste notre objectif principal. En effet, l'identification en temps réel est obligatoire pour permettre aux FAI et aux administrateurs de pouvoir gérer le trafic HTTPS lors de l'exploitation de leur réseau. Nous voulons donc améliorer le framework pour permettre l'analyse et l'identification en temps réel des services HTTPS, ce qui est présenté dans la section suivante.

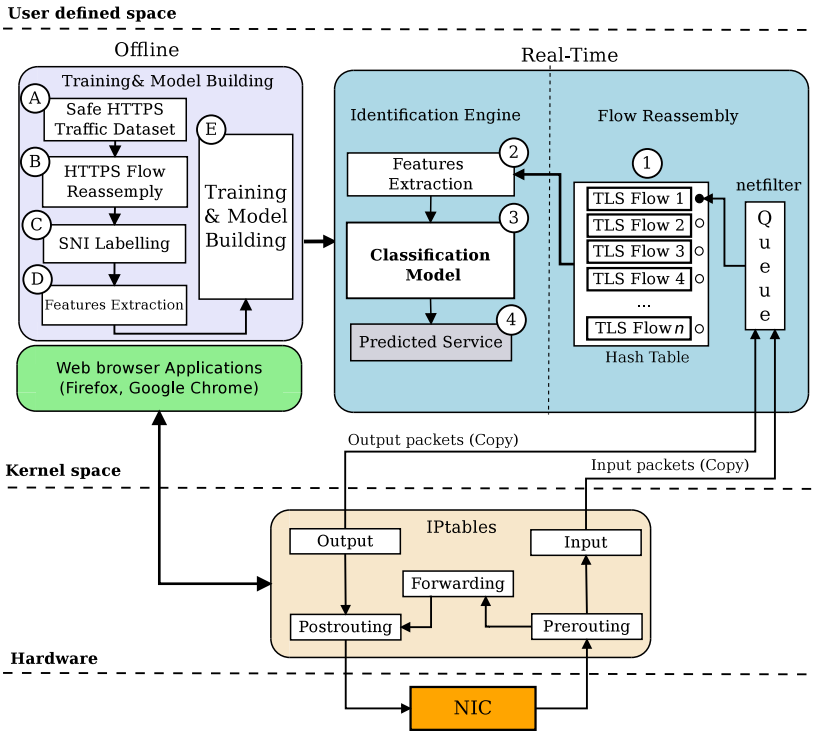


Figure 4: Real-time HTTPS services identification prototype flowchart

5 Identification en temps réel de services HTTPS

Cette section traite de l'identification en temps réel des services HTTPS. Nous proposons une nouvelle approche pour identifier les services qui limite son observation aux paquets d'initialisation de la connexion TLS et à un petit nombre de paquets applicatifs. Cette solution a été étudiée sous différents angles. Tout d'abord, nous avons constitué et décrit un jeu de données public de trafic HTTPS, ceci afin de garantir la reproductibilité de nos résultats. Deuxièmement, nous avons étudié le nombre de paquets de données applicatives nécessaires pour obtenir une bonne précision d'identification. Les résultats montrent que les paquets d'initialisation et 9 paquets applicatifs sont nécessaires pour identifier sans perte de précision les longues sessions HTTPS, soit 13 paquets au total. Ce résultat se positionne très bien parmi l'état de l'art puisque d'autres travaux similaires nécessitent jusqu'à 70, voire 100 paquets pour des résultats inférieurs. Troisièmement, nous avons évalué la relation entre la précision de l'identification et différentes valeurs de seuil limitant le nombre de paquets applicatifs observés. Les résultats expérimentaux montrent que globalement, la meilleure classification sur l'ensemble de notre jeu de données est obtenue en considérant jusqu'à 5 paquets applicatifs. En effet, la majorité des flux HTTPS étant courts, observer jusqu'à 5 paquets applicatifs permet une meilleure précision globale atteignant 92%.

En tant qu'application directe de notre approche, nous avons réalisé un prototype pour identifier en temps réel les services HTTPS. Le prototype proposé nécessite plusieurs étapes, comme illustré dans la Figure 4. La première est au niveau matériel où les paquets sont copiés et dirigé vers l'espace noyau grâce à des règles iptables, puis sont copiés dans l'espace utilisateur pour la reconstruction du flux et l'exécution de l'algorithme de classification. L'évaluation des performances du prototype montrent des valeurs compatibles avec notre objectif, avec faible retard de 2,02 ms et un débit élevé de 31874 flux traités par seconde.

6 Conclusion générale

6.1 Travail réalisé

L'augmentation importante du trafic HTTPS ces dernières années crée un besoin urgent de nouvelles méthodes pour superviser et gérer ce trafic. Dans cette thèse, nous traitons ce problème en quatre étapes. Tout d'abord, nous avons étudié et proposé une taxonomie de l'état de l'art dans le domaine de l'identification du trafic HTTPS en fonction des étapes nécessaires pour identifier les services. Deuxièmement, nous avons étudié en détail une technique récente de supervision et de filtrage des services HTTPS basée sur l'extension SNI de TLS. Nous avons identifié plusieurs défauts dans l'utilisation actuelle de celle-ci qui la rend peu fiable et avons proposé une amélioration efficace sollicitant un serveur DNS. Troisièmement, nous avons proposé un framework permettant d'identifier les services HTTPS accédés dans des traces réseau complètes, ceci sans compter ni sur un champ d'en-tête particulier, ni sur le déchiffrement des données applicatives. Le framework inclue un nouvel ensemble d'indicateurs statistiques combiné à des algorithmes d'apprentissage automatique et à une nouvelle approche de classification multi-niveaux qui permettent ensemble d'obtenir une précision élevée d'identification. Quatrièmement, l'identification en temps réel des services HTTPS a été abordée. Nous avons proposé des améliorations à notre framework pour superviser les services HTTPS en temps réel. En extrayant des indicateurs statistiques des paquets d'initialisation de TLS et progressivement sur quelques paquets applicatifs, nous pouvons identifier les services HTTPS très tôt dans la session. Nous avons évalué cette ultime approche de supervision en temps réel grâce à des expériences approfondies réalisée sur un jeu données public que nous avons constitué, ainsi que grâce à la mise en œuvre d'un prototype fonctionnel de pare-feu HTTPS offrant des performances encourageantes.

6.2 Perspectives de recherche

Cette thèse ouvre de nombreuses perspectives visant à améliorer notre solution et la rendre pérenne. Tout d'abord, il faudrait pouvoir résister aux techniques de morphing du trafic, qui sont notamment utilisées par les réseaux anonymes comme Tor et visent à modifier l'empreinte statistique des flux éviter toute identification. À un autre niveau, HTTPS sera progressivement remplacé par le protocole HTTP/2 dans les années à venir ce qui implique d'adapter notre architecture pour pouvoir

identifier les services dans le trafic HTTP/2. Cette adaptation est un réel défi du fait de la capacité du protocole HTTP2 de pouvoir multiplexer différents contenus dans un même flux.

L'apprentissage à la volée du moteur d'identification est également intéressante, de sorte que celui-ci puisse progressivement prendre en compte les évolutions du trafic pour les différents services afin de maintenir un bon taux d'identification sans avoir besoin d'un processus de rééducation périodique complet. Enfin,les performances de l'architecture peuvent être améliorées par l'utilisation de FPGA qui peuvent accélérer matériellement l'analyse des paquets et notamment l'exécution de l'algorithme de classification C4.5.