Improving SNI-based HTTPS Security Monitoring

Wazen M. Shbair, Thibault Cholez, Jérôme François, Isabelle Chrisment

Improving SNI-based HTTPS Security Monitoring

Wazen M. Shbair *, Thibault Cholez *, Jérôme François †, Isabelle Chrisment *
* University of Lorraine, LORIA, UMR 7503, Vandoeuvre-les-Nancy, F-54506, France
Email: {shbair.wazen, thibault.cholez, jerome.francois, isabelle.chrisment}@loria.fr
† INRIA Nancy-Grand Est, 615 rue du Jardin Botanique, 54600 Villers-les-Nancy, France

***Abstract*—Recent surveys show that the proportion of encrypted web traffic is quickly increasing. On one side, it provides users with essential properties of security and privacy, but on the other side, it raises important challenges and issues for organizations, related to the security monitoring of encrypted traffic (filtering, anomaly detection, etc.). This paper proposes to improve a recent technique for HTTPS traffic monitoring that is based on the Server Name Indication (SNI) field of TLS and which has been implemented in many firewall solutions. This method currently has some weaknesses that can be used to bypass firewalls by overwriting the SNI value of new TLS connections. Our investigation shows that 92% of the HTTPS websites surveyed in this paper can be accessed with fake-SNI. Our approach verifies the coherence between the real destination server and the claimed value of SNI by relying on a trusted DNS service. Experimental results show the ability to overcome the shortage of SNI-based monitoring by detecting forged SNI values while having a very small false positive rate (1.7%). The overhead of our solution only adds negligible delays to access HTTPS websites. The proposed method opens the door to improve global HTTPS monitoring and firewall systems.**

I. Introduction

Recent surveys show that the proportion of HTTPS traffic is quickly increasing. According to a recent report from French ISPs [1], the amount of encrypted traffic reached 50% of the Internet traffic in 2015 against only 5% back in 2012. Such steady increase is related to many factors such as: HTTPS adoption by giant websites like Facebook and Youtube [2], and the increasing computation power in network devices that makes complex encryption/decryption applicable to high-speed Internet [3]. This change in trends adds new challenges regarding network monitoring and management for security (filtering, anomaly detection, etc.) and related to the identification of encrypted traffic, especially HTTPS which supports many services.

HTTPS monitoring is related to network monitoring techniques, which provide knowledge about traffic flowing through a network. Thus, if appropriate data are monitored, it is possible to detect network attacks, security policy violations and malicious activities [4]. In this context, the terms *Monitoring* and *Filtering* are widely used when we need to have control over Internet usage. However, *Monitoring* is defined as recording the activities on the Internet (accessed websites, applications, etc.), while *Filtering* means restricting access to applications or websites that the administrator wants users to avoid by adding websites URLs to blacklists or using keyword filtering [5], [6]. We need to monitor first to be able to filter.

HTTPS (i.e. HTTP over SSL/TLS) is the most commonly used encrypted protocol [7], [8], thus the accurate and efficient identification of HTTPS traffic is essential for managing and controlling current and future data networks [9]. The related work in this field can be classified into two branches: the *Academic* work and the more *Practical* solutions to which this work is more related. The scientific related work propose techniques like Website Fingerprinting, Machine Learning techniques and Markov Chain models to recognize the HTTPS traffic. A recent survey by Velan et al. [10] presents these relevant work. However, these techniques are still under development and research and have many challenges related to applicability and accuracy. In the meantime, practical solutions have been implemented in firewall systems to monitor HTTPS traffic, such as HTTPS proxy [11] and monitoring based on SSL certificates. These methods still have many issues related either to privacy or efficiency [12].

In this paper, we explore one widely used practical technique to identify HTTPS traffic, named SNI-based monitoring that uses Server Name Indication (SNI), a field of the SSL/TLS handshake, to identify the accessed websites in HTTPS connections. The SNI is a clear string value from TLS ClientHello messages 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 (Sphirewall[1], Untangle NG[2], IPFire[3], etc.). In our previous work [12], we demonstrated that the reliability of SNI-based filtering is poor because it is easy to forge SNI values to bypass such protection. In this paper, we propose an improvement to SNI-based monitoring techniques to overcome such shortages, with the goal to improve HTTPS firewall systems. Our contribution is threefold: (1) we provide the first estimation of the percentage of HTTPS servers still accessible despite SNI-based filtering, (2) we propose a solution to detect fake-SNI and (3) we evaluate the efficiency of our solution regarding detection rates and overhead.

The rest of the paper is organized as follows: Section II and III respectively present the related work and an overview of SNI. Our solution to overcome fake-SNI is described in Section IV. Section V presents our investigation of HTTPS servers accessible with a fake-SNI. Section VI evaluates the proposed approach and related overhead. Finally, Section VII concludes the paper.

II. Related work

The 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

[1] http://www.sphirewall.net
[2] https://www.untangle.com
[3] http://www.ipfire.org

content [13]. Enterprise networks statistics [5] show that 66% of companies monitor their employees' network activity. For example almost 44% of online video is being viewed during work time and 29% of adult websites are accessed from work computers. Relevant work on HTTPS monitoring relies on the information exchanged during SSL/TLS handshake, such as monitoring the SSL certificate [14] [15], monitoring the handshake interactions sequence [16] and, more recently, monitoring SNI [12] (what we aim to improve). Identifying websites based on TLS interactions leverages machine learning techniques and is prone to insufficient training and identification errors while certificate monitoring is limited when certificates are shared among services or often changed.

However the most widespread approaches for HTTPS security monitoring rely on decryption. They are based on HTTPS Proxy Server, SSL/TLS encryption keys retrieval or cracking the encryption algorithms. In the first approach, a proxy server is placed between client and server and pretends to be the intended remote server. Then, the proxy establishes a secure connection to the correct server. As shown in Figure 1, when a client connects to the remote server via a HTTPS proxy, the client connects in reality to the proxy server, which plays the role of destination server by providing its own SSL certificate. Finally, the proxy establishes another secure connection with the real remote server. Thanks to this basic method, all encrypted web traffic is open to the proxy in clear but at the cost of trust and privacy.

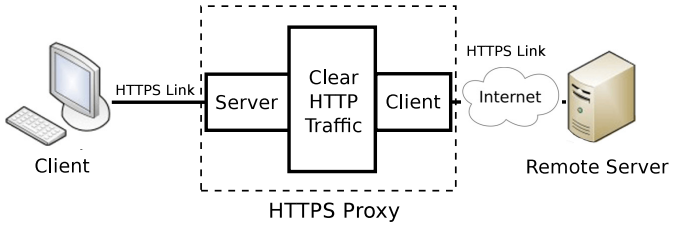

Fig. 1: HTTPS Proxy Server

SSL/TLS encryption keys can be retrieved thanks to a "Key Recovery" mechanism or "Key escrow" [17]. All encryption keys are kept in a trusted third party, such as government or private entities, which have 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 SSL/TLS key with the escrow system. Cracking the encryption algorithms is a very different approach which relies on flaws either in the encryption protocol or in the mathematical algorithm used to encrypt data. Using a full brute force attack is very unlikely because of the time needed to perform it (billions of years to break a 128-bit encryption).

The aforementioned approaches have many issues related to privacy (decryption on users' data) or computation complexity. We believe that improving SNI-based monitoring can make it a powerful method for managing HTTPS traffic. Indeed, SNI monitoring offers a better trade-off between security and privacy as private data stay fully encrypted while only the destination service is known.

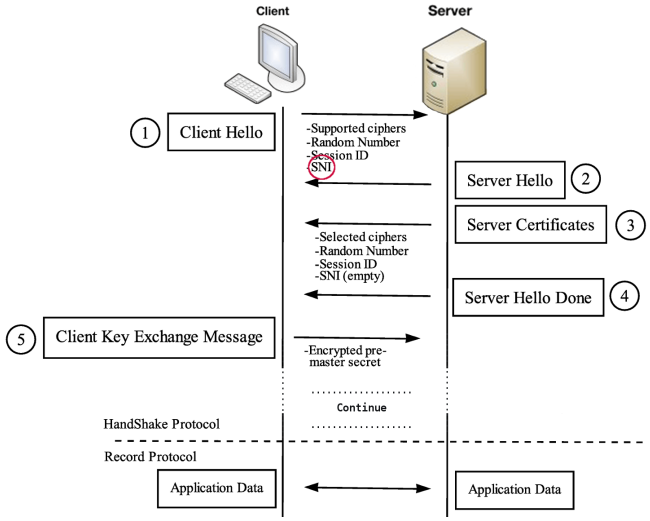

Fig. 2: SSL/TLS Handshake protocol

III. Background on SNI usage

A. Standard usage of SNI

In parallel with virtual hosting techniques that make multiple websites hosted on a single machine, appeared a new challenge to make it compatible with SSL/TLS. Virtual hosting can be either "IP-based" or "name-based". In the first type, each SSL certificate is mapped to a unique IP-address, so the server depends on the IP address to answer with the correct SSL certificate. But it is expensive to reserve an IP address for each website, therefore in a name-based method, all hosted websites share the same IP address and the server can easily retrieve the right certificate from the HTTP header, by reading the website's URL in the GET request. The problem with the named-based certificate identification used with HTTPS is that the HTTP header is sent encrypted after the SSL/TLS connection is established, and thus cannot be used to identify the certificate. Since each virtual website has its own SSL certificate, the problem is to select and expose the proper certificate corresponding to the intended website, but this was not envisioned when SSL/TLS was designed. Thus, an extension of the protocol was used to enable TLS to operate effectively with HTTP and overcome this limitation [18].

The Server Name Indication is used during the SSL/TLS handshake protocol and specifies in plain text the requested hostname in Client Hello messages (i.e. Step 1 in Figure 2) of the SSL/TLS handshake [19]. More precisely, the behaviour of SNI is presented as follows: in order to provide the "server-name", client may include an extension of type SNI in the extended Client Hello message. The "extension-data" field of SNI shall contain at most one hostname, thus a server is able to present the correct SSL certificate according to the hostname appearing in the SNI [20].

B. Alternative usage: SNI-based HTTPS Monitoring

Monitoring based on SNI relies on checking the "server-name" value in SNI. 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 and

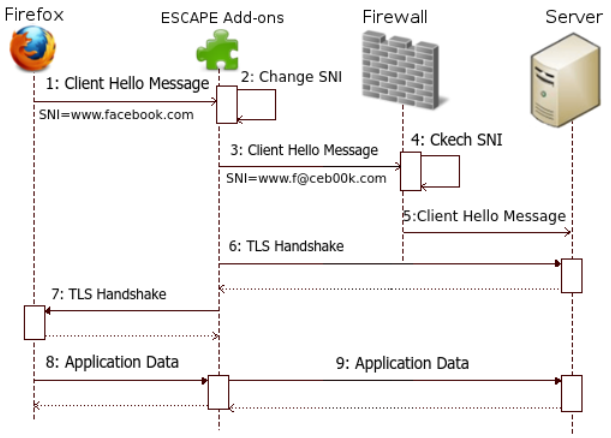

Fig. 3: Escape Add-on interactions from [12]

TABLE I: Example of *Alternative Name* values in Youtube's SSL certificate

*.google.com	*.android.com	*.cloud.google.com
*.google.ca	*.google.co.jp	*.google.fr
*.googlevideo.com	*.youtube.com	*.url.google.com

the "server-name" can be compared against a list to enforce HTTPS filtering. In [12], we designed and developed a tool, named Escape, which can be used to bypass such type of filtering by replacing the SNI value with a fake one. The tool can exploit two weaknesses: the first exploits backward compatibility and is based on the fact that, according to the RFC [20], if a remote server does not understand the SNI, it should continue the handshake. Initially, this was done to ensure that HTTPS servers are still accessible to clients which do not support the SNI extension. However, the backward compatibility can be used to bypass a SNI-based firewall by changing the "server-name" in SNI with random values or with unblocked domain names. Figure 3 shows how the backward compatibility is exploited for bypassing firewalls thanks to Escape. In this example, a firewall restricts access to Facebook but Escape is able to change the SNI from "www.facebook.com" to "www.f@ceb00k.com", so that the client hello message will bypass the firewall as the fake domain "f@ceb00k.com" is not blocked. The second weakness is the *Alternative Name* field of the certificate standard X.509, which can holds multiple domain names, so one certificate for multiple domains. Table I gives a short list of the *Certificate Subject Alternative Name* fields appearing in the Youtube.com SSL certificate. In this context, shared certificates can be used to access a restricted website by sending SNI for a unrestricted one sharing the same SSL certificate. We consider that this weakness is less critical as it only allows a user to access services from the same provider or the same hosting cloud. The solution we propose in the next section aims to mitigate the "backward compatibility bypassing strategy" that can totally mask the accessed website.

IV. SNI-based Monitoring Improvement

A. Architecture

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

When a website is accessed, prior to any SSL/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 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 [12], 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 SNI because, according to the RFC [20], SNI 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 [21] and remote server IP address [22] to identify network traffic.

Figure 4 shows the verification procedure of our proposed solution that assesses SNI using a trusted DNS server. The steps are as follow:

1) Inspect the ClientHello message and extract the extensions list and the destination server IP address.
2) Search for 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.

B. Fake-SNI Detection

In this subsection, we describe how to perform the check to assess the SNI from the information in the DNS response. The fake-SNI 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 and should block the connection since all modern systems add an SNI value by default. The only issue is when considering older hosts based on Windows-XP (and below) whose TLS implementation does not support SNI. However, those hosts should not be allowed to access the World-Wide-Web in a security-sensitive context because they miss security updates (since 04/2014 concerning

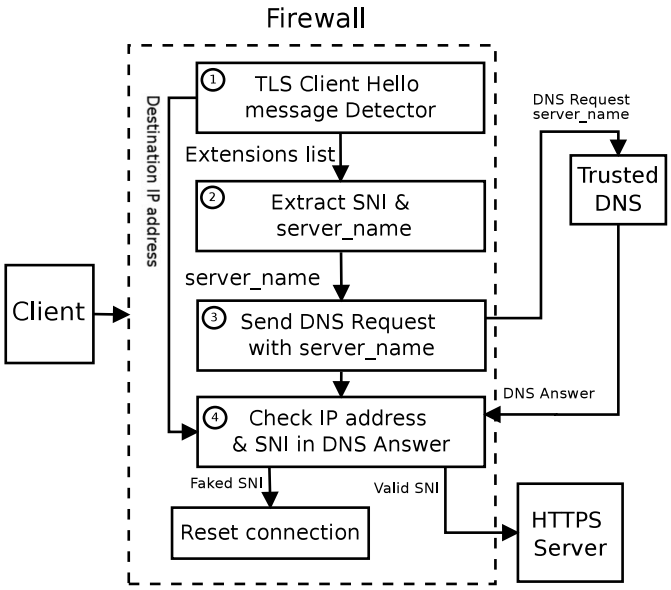

Fig. 4: Client SNI verification using DNS

WindowsXP) and are consequently very vulnerable. Moreover, their number is always decreasing and now accounts for only 4.24% of hosts[4].

The third case will be illustrated by a real scenario. We assume the HTTPS website "twitter.com" is restricted by SNI-based filtering. To overcome this limitation, a client browser with Escape tool can overwrite the SNI of Twitter (twitter.com) with another valid domain (www.google.com). In the firewall, the processed ClientHello message destination IP-address is a Twitter's IP address (like 104.244.42.X), while the SNI contains "www.google.com". Our module then sends an independent DNS request with SNI value (i.e "www.google.com") to get the IP addresses related to this domain name. The DNS response contains some Google's IP addresses (like 66.102.1.X). Comparing the real destination IP address with those in the DNS response shows a clear mismatch between them, which means that the connection must be considered suspicious by the system and processed accordingly. For instance, the ClientHello message can be dropped to reset the HTTPS connection.

To check the correspondence between the destination server IP address and the ones included in the DNS response, different rules can by applied from the a rigorous check of the 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 the detection strategy impacts detection rates.

V. Evaluation of HTTPS servers permeable by fake-SNI

A. Evaluation of HTTPS servers supporting SNI

Nowadays, the SNI extension is supported by most of browsers, such as Mozilla Firefox, Safari or Google Chrome, and by servers software, like Apache and Microsoft IIS 8. To assess how much SNI is used, we conducted the first investigation of SNI deployment with a large set of web servers accessed over a HTTPS connection. Our probing follows RFC6066 [20], which describes how a server must interact in the case of receiving a Client Hello message with SNI extension. If the server supports SNI, it answers with a Server Hello message holding a SNI extension, with an empty value (i.e length equal 0), otherwise the server responds with the standard ServerHello message without extension. Based on this server behaviour, we implemented a SNI crawling module, where ServerHello messages are intercepted to verify if they hold an empty SNI extension.

Basically, SSL/TLS messages consist of two sub-layers, the first layer contains either SSL/TLS handshake, SSL/TLS Change Cipher Specification or SSL Alert protocol, while the second layer holds the SSL/TLS Record protocol, which is known as an envelop for application data. We intercept Server Hello messages to determine if the corresponding server support SNI or not. Server Hello messages are identified with a TLS Record type of 22 and an handshake type of 2. Once the Server hello message is detected, the list of extension is extracted and scanned for an extension with type zero and empty data. According to [20], if this extension is present, the server supports SNI. Otherwise, it does not.

Our investigation looks at HTTPS domains taken from two lists. The first source is The Internet-Wide Scan Data Repository, which is managed and hosted by University of Michigan[5]. The selected dataset contains the Alexa Top 1 million domains that respond on port 443. Instead of using this large amount of domain, we use the top 500 HTTPS websites. The second list is provided by Wang et al.[6] and contains 120 sensitive sites blocked in China, the United Kingdom, and Saudi Arabia. We found 33 websites out of the 120 run over HTTPS and we added 14 of them which were not in the first list. In total, we consider 514 HTTPS websites for this study.

For each website, we accessed the index page with a Web browser supporting SNI (i.e Firefox). After initiating the SSL/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 SNI in the way recommended by the RFC, while the others perfectly support SNI. The websites that do not support SNI are probably hosted on a dedicated server, so that they do not need this extension to select the proper SSL certificate. Monitoring HTTPS websites that do not use SNI is a real challenge since fake-SNI values will be ignored on the server side. A client is then able to overwrite the SNI as the server do not care about the actual value. However, we will see in the next subsection that most servers supporting SNI can even be accessed with a fake-SNI.

B. Evaluation of the bypassing strategy

In this section, we study how HTTPS websites deal with a fake or unknown SNI, in particular if they implement the backward compatibility described in the RFC [20]. We configured a SNI-based firewall system, named Sphirewall, to restrict the access to the 514 HTTPS websites and to reset the HTTPS connection if such websites are requested. The client browser (Firefox 33, on Ubuntu 14.04) is configured

[4]http://www.w3counter.com/globalstats.php

[5]https://scans.io/series/443-https-tls-alexa_top1mil

[6]https://cs.uwaterloo.ca/ t55wang/wf.html

with Escape, which overwrites the SNI of the 514 websites with faked ones to bypass the firewall. As shown in Figure 3, the firewall checks the SNI value against a black list to take a decision. We assess the behaviour of the HTTPS servers by observing if the page loads or not despite the fake-SNI 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. Overall, 8% of websites (41 out of 514) refuse to go further in the SSL/TLS handshake when facing a fake-SNI, while 92% of the contacted HTTPS servers ignore this inconsistency according to the backward compatibility principle explained above. For instance, websites like "www.change.org" and "www.uptobox.com" refuse the fake SNI, since both are hosted by CloudFlare, where a correct SNI is mandatory to return the correct SSL certificate.

The high success rate (92%) of the bypassing strategy motivates the need for a method able to check the SNI properly before forwarding the ClientHello to the remote destination server. In the next section, we evaluate our proposed method.

VI. Evaluation of SNI verification based on DNS

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

A. Evaluation of the false-positive rate

All ClientHello messages are inspected for verification, which also includes the ones with a legit (i.e. unaltered) SNI. We need to evaluate how the proposed method deals with ClientHello messages holding a legit SNI, 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 ACL (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 [23] investigate different scenario 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 blocked despite a legit-SNI.

As shown in Figure 4, the original destination IP address of a ClientHello message is compared with the DNS answer that has been requested by using the "server-name" value written in SNI. 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 16-bits) of the destination IP address in the DNS response

TABLE II: 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%

By using the aforementioned methods for comparison, the results show that, all tested 514 HTTPS websites are accessible. But we investigated further. We then not only considered the main HTTPS connection of each web site, but also the large number related connections toward HTTPS servers (2986) that are used to render the complete web pages: load the website content, display advertisement, make statistics, etc. Table II 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 allows us to validate up to 98.29% of legit SNI. This results in a overall 1.71% false positive rate. This low FP value should not alter web browsing and demonstrates the efficiency of using DNS information to validate client SNI.

B. Evaluation of the overhead

Our SNI verification can introduce an overhead related to computation and network latency. In our work, we neglect 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 study two cases: when the trusted DNS server is a global one (Google DNS) or a local one (LORIA DNS, installed in our laboratory network). These two DNS servers should exhibit different performances because 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). Of course, when clients already contact a trusted DNS server to resolve the HTTPS server name, the DNS response can be directly analysed without issuing an additional request.

Figure 5 shows the latency ranges of DNS queries towards both DNS servers. 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 10ms. We can notice that the local DNS performs a bit better in this range thanks to the reduced network delay to reach it. However, in the next ranges of delay, the global DNS performs a lot better, especially with a high value in the range [10,20[ms 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 Root-Level Domain Servers 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 [24], the average server response time (and standard deviation) is quite high with HTTPS 483 ($\pm$ 44.1) ms. Our results show that we do not alter the browsing experience of users when accessing HTTPS websites.

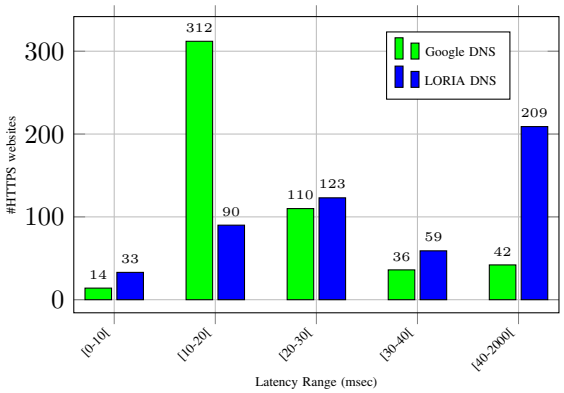

Fig. 5: DNS latency range per number of HTTPS websites

VII. Conclusion

Currently, the proportion of encrypted traffic is quickly increasing. On one side, it provides users with essential properties of security and privacy, but also raises important challenges and issues related to the security monitoring of encrypted traffic. The contribution of this work is threefold. We conducted the first investigation of SNI 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, we proposed a novel DNS-based approach to validate SNI in the context of HTTPS security monitoring that consists in verifying the relation between the actual destination server and the claimed value of SNI based 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). We thus believe that this work represents an important development in the field of HTTPS monitoring, and can be applied to improve global HTTPS monitoring and firewall systems. Our future work will integrate the proposed method in a firewall system for HTTPS.

Acknowledgement

This work was partially funded by Flamingo, a Network of Excellence project (ICT-318488) supported by the European Commission under its FP7 Programme, and by HuMa, a project funded by Bpifrance and Region Lorraine under the FUI 19 framework.

References

[1] "Report 2015-0832 from the French regulatory authority for telecommunications (ARCEP)," [In French], Accessed: 19/04/2015. [Online]. Available: http://www.arcep.fr/uploads/tx_gsavis/15-0832.pdf

[2] 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.

[3] Z. Cao, G. Xiong, Y. Zhao, Z. Li, and L. Guo, "A survey on encrypted traffic classification," in *Applications and Techniques in Information Security*. Springer, 2014, pp. 73–81.

[4] 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.

[5] "Internet filtering and monitoring in your business," http://www.currentware.com/whitepapers/Internet-Filtering-and-Monitoring-in-your-BusinessWhite-Paper.pdf, Accessed: 19/04/2016.

[6] 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.

[7] W. Shbair, T. Cholez, J. Francois, and I. Chrisment, "A multi-level framework to identify https services," in *IEEE/IFIP Network Operations and Management Symposium*, 2016.

[8] 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.

[9] Z. Cao, S. Cao, G. Xiong, and L. Guo, "Progress in study of encrypted traffic classification," in *Trustworthy Computing and Services*. Springer, 2013, pp. 78–86.

[10] 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.

[11] "Safety recommendations concerning the analysis HTTPS stream," http://www.ssi.gouv.fr/uploads/IMG/pdf/NP_TLS_NoteTech.pdf, [In French], Accessed: 19/04/2015.

[12] W. M. Shbair, T. Cholez, A. Goichot, and I. Chrisment, "Efficiently bypassing SNI-based HTTPS filtering," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 990–995.

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

[14] 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.

[15] 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.

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

[17] 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.

[18] 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

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

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.