Efficiently Bypassing SNI-based HTTPS Filtering

Wazen M. Shbair, Thibault Cholez, Antoine Goichot, Isabelle Chrisment

Efficiently Bypassing SNI-based HTTPS Filtering

Wazen M. Shbair *, Thibault Cholez *, Antoine Goichot *, Isabelle Chrisment *
* University of Lorraine, LORIA, UMR 7503, Vandoeuvre-les-Nancy, F-54506, France
Email: {shbair.wazen, thibault.cholez, isabelle.chrisment}@loria.fr

***Abstract*—Encrypted Internet traffic is an essential element to enable security and privacy in the Internet. Surveys show that websites are more and more being served over HTTPS. They highlight an increase of 48% of sites using TLS over the past year, justifying the tendency that the Web is going to be encrypted. This motivates the development of new tools and methods to monitor and filter HTTPS traffic. This paper handles the latest technique for HTTPS traffic filtering that is based on the Server Name Indication (SNI) field of TLS and which has been recently implemented in many firewall solutions. Our main contribution is an evaluation of the reliability of this SNI extension for properly identifying and filtering HTTPS traffic. We show that SNI has two weaknesses, regarding (1) backward compatibility and (2) multiple services using a single certificate. We demonstrate thanks to a web browser plug-in called "Escape" that we designed and implemented, how these weaknesses can be practically used to bypass firewalls and monitoring systems relying on SNI. The results show positive evaluation (firewall's rules successfully bypassed) for all tested websites.**

I. Introduction

A. Background on HTTPS and TLS

The Internet has been combined with every aspect of our life, social media, education, health, sport and finance. Today, we are transferring more sensitive information than ever. This traffic navigates through the Internet in secure connections, encrypted with Secure Socket Layer (SSL). In 1999, the Internet Engineering Task Force (IETF) adopted this protocol as a standard known as Transport Layer Security (TLS) [1], which is basically used to secure HTTP into HTTPS. Through the paper, we use the term TLS to indicate both protocols.

TLS is an encryption protocol that operates under the application layer. Its goal provides secure communication channels for the wide set of application layer protocols. In the case of HTTPS, when a client first connects, the client and server complete a TLS handshake during which the server presents an X.509 digital certificate, which is used to identify and authenticate the server to the client. This certificate includes the identity of the server (e.g. website domain), a temporal validity period, a public key, and a digital signature provided by a trusted third party. The client checks 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 cryptographic channel [2]. The latest version of TLS (1.2) specifies two-sub protocols, TLS Record Protocol and TLS handshake protocol, assuring confidentiality and integrity of traffic between communication parts [3].

Surveys show that 25%-35% of enterprise traffic is TLS-encrypted, and the number may reach 70% in some companies. TLS encryption is the most commonly used solution to protect the privacy of that traffic. In January 2014, Netcraft[1] conducted a survey on 860 million websites and showed that the year 2013 has undergone a significant change: websites are more and more being served over HTTPS, probably after the fact that the usage of TLS has been highlighted as a defence after a series of spying revelations. In total, there is an increase of 48% of websites using TLS and of half a million TLS certificates (+22%) on the Internet since January 2013. For instance, to secure communications and protect users' privacy, most of social media websites (Facebook, Twitter, etc.), web-based email services (Yahoo, Gmail, etc.), Cloud storage providers (Dropbox), use HTTPS to secure the link between users and servers.

B. Web Filtering: between Security Policies and Censorship

The security of data and transactions is a keystone element in commercial and government data networks. The security challenges have become a top priority for all organizations using information technology. However, it is very hard for an IT organization to properly monitor and manage TLS-encrypted web traffic which may cause some problems [4]. Indeed encrypted content cannot be scanned to detect malware or to check/verify compliance with security policy. For example, TLS can be used to hide content that users do not need in the workplace and to expose organization sensitive data to security risk. So, there is a real need for monitoring and control techniques related to HTTPS traffic while trying to keep users privacy.

If, in some sensitive organizations, the filtering of encrypted traffic can be legitimate, on the opposite, a state-enforced large scale web traffic filtering is a real threat against people's privacy and freedom that is regularly used in some countries . The authors of [5] state that the practice of Internet filtering and surveillance is becoming more commonplace and sophisticated.

C. Outline

In this context, this paper aims to investigate for the first time a new technique that relies on Server Name Indication (SNI) extension of TLS [6] for filtering HTTPS traffic and we demonstrate that it can be easily bypassed. The rest of the paper is organized as follows. Section II describes legacy filtering techniques when applied to HTTPS. Section III introduces SNI and how it is used for HTTPS filtering, Section IV evaluates the reliability of SNI-based filtering and explains our strategies to bypass it, Section V describes the implementation and evaluation of our tool "Escape" which

[1] http://news.netcraft.com/archives/2014/01/03/january-2014-web-server-survey.html

against existing firewalls. Finally, Section VI concludes the paper.

II. Legacy Web-Filtering Techniques

Web filtering techniques 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. A large set of techniques is possible from the simplest, with port-based filters, to the most advanced ones with Deep Packet Inspection (DPI) or statistical based techniques. In this section, we present several technical solutions to implement web filtering and we discuss their relevance when they are used to filter HTTPS traffic.

A. Port-based Filtering

Internet applications and protocols can be identified by well-known transport layer ports assigned by IANA, for example port 80 for HTTP, 21 for FTP, 443 for HTTPS, etc. As a result, applications could be identified by extracting the port-value from the packets headers. Unfortunately, this technique does not allow fine-grained filtering and affects the whole usage of a protocol without distinguishing between allowed or blocked websites and services. Given the wide usage of HTTPS today, this method is obsolete. 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 [7].

B. DNS Filtering

Prior to any HTTP request is sent a Domain Name System (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. Also, the DNS filtering is ineffective if the client behind a filtering system uses directly the IP address or a local resolver. In this case, no DNS query is sent. Beside bypassing HTTPS filtering, our tool "Escape" provides ways to overcome DNS filtering.

C. IP Address Filtering

Filtering the IP address of a web server is another radical method. In fact, nowadays, the "Virtual Hosting" allows a single machine to serve many websites and many Web servers are often associated to the same IP address. Alternatively, a single domain name may correspond to many IP addresses to improve fault tolerance and load balancing.

These two tendencies make filtering based on IP address 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 others like Search or Maps is impossible based only on IP addresses: the whole range of Googles IP addresses is either blocked or allowed.

D. Certificate Filtering

Certificate authorities (CAs) provide digital certificates for websites, which are used for identification and authentication. CAs 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 are integrated with trusted signing certificates known as "Root Certificates" [2]. TLS protocol enables client software to build secure communication terminated by server/client holding the private key associated with the certificate. The most critical attribute, that all HTTPS server certificates hold is the domain name. This attribute is located in the CommonName (CN) field under Subject. One or more domains are often included in the subject alternative name field in an X.509 extension [8]. Based on X.509 extension the alternative name field gives the ability to use single certificate for multiples domains.

HTTPS filtering can be made based on the CommonName (CN) in the server certificate. Like IP address filtering, this technique has proven to be inefficient as many companies share the same certificate across different services and domain names. For example, to blacklist youtube.com based on its certificate, all the other domain names covered by this certificate and given in SubjectAltName (i.e. all the other Google services) must also be blacklisted. In this case, it is impossible to differentiate between the different Google services based on the certificate and to have a fine-grained filtering between services.

E. Payload-based Techniques

All the above filtering traffic techniques are based on metadata, but not on what packets contain. The payload-based or Deep Packet Inspection (DPI) techniques inspect the content of packets, looking for keywords or regular expressions to identify what the packets hold [9].

Analyzing an encrypted payload is of course another challenge, beyond the current capacities of real time processing. Thus, the payload-based techniques totally lose their effectiveness against encrypted traffic [10].

F. HTTP Proxy Filtering

An HTTP proxy acts as a man in the middle that processes and forwards users' HTTP requests. It can be used for both performance purposes (caching) and security when blocking websites. The filter can be a simple blacklist, or since the full content of the requested web pages is available, a filter against some keywords or expressions [9].

However, in the case of HTTPS, the proxy cannot directly process HTTP requests that are in the encrypted traffic. To use the HTTPS proxy, the client needs to issue a secured connection to the proxy which will do itself the connection to the secured web server. This setup allows the proxy to access all encrypted web traffic in clear between the client and the server at the cost of huge trust and privacy issues. Because sensitive data is transmitted through HTTPS, it is hardly acceptable to trust a third-party to screen these information and break our privacy.

III. Overview of the Server Name Indication Extension

Because of all these limitations, firewalls and Web content filtering solutions now use the Server Name Indication (SNI) field of TLS to precisely identify and filter HTTPS traffic. The current section provides an overview of SNI and how it is used to filter HTTPS.

A. What is the SNI Extension for?

HTTPS initially faced a deployment issue with virtual hosting technique, which allows the hosting of many named instances on a single server thanks to the HTTP host header. This header usually transports the fully qualified domain name of the (virtual) web server the client wants to reach. Thereby, the web server is able to run multiple virtual sites on just a single IP address, using the host header for multiplexing. However, HTTPS encrypts the host header, so the web server cannot access this information before finishing the TLS handshake. For the handshake, however, the web server has to supply the correct virtual site's certificate. A solution for this problem is the Server Name Indication (SNI) extension of TLS [11]. SNI allows a client to specify the hostname it is attempting to connect when the TLS negotiation starts, as shown in Figure 1. The hostname contains the fully qualified DNS hostname of the server. This makes a server able to present multiple certificates on a single IP address and to ultimately host multiple HTTPS sites using different certificates on the same IP address [2]. So, SNI is a useful complement to the concepts of virtual hosting, therefore a server is able to present the correct mapping between virtual hosts and the respective certificates [11].

In RFC 6066 [6], the behaviour of SNI is explained as follows: in order to provide any of the server names, clients may include an extension of type SNI in the (extended) Client Hello as shown in Figure 1. The "extension_data" field of this extension shall contain "ServerNameList", where the ServerNameList must at most contain one name. A server that receives a Client Hello containing the SNI extension uses the information contained in the extension to select an appropriate certificate to be returned to the client. Then, the server answers with an extended Server Hello, where the SNI extension should be empty.

B. Deployment and Support of SNI

Introduced in 2003, now SNI is widely used by browsers, servers and CAs. Only about 15 percent of the browsers and clients do not support the SNI extension yet, in particular, old versions of Internet Explorer running under Windows XP[2]. Based on a survey conducted by Netcraft in April 2012, Symantec had issued the highest number of active TLS certificates in the world[3]. In our context, Symantec supports securing multiple domains on single servers integrated with SNI. This means that there is global trend for using SNI and all new versions of web browsers and client-side applications must support it.

Most current server software, such as Apache or Microsoft IIS 8, already support the SNI extension [11]. On the browser side, all browsers released in the past 5 years use SNI, such as Mozilla Firefox, Internet Explorer, Safari or Google Chrome.

[2]http://www.psw-group.de/en/sni.html

[3]http://www.symantec.com/connect/blogs/why-symantec-ssl-certificates-are-most-trusted-ssl-certificates-world

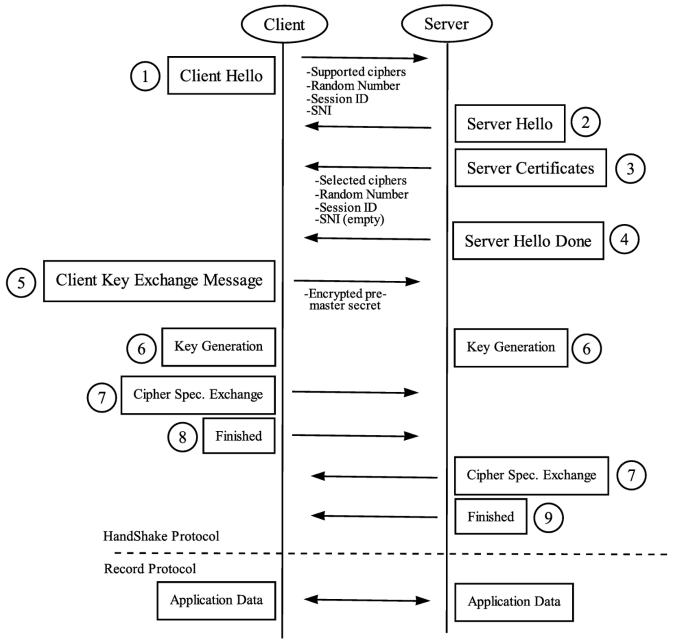

Fig. 1. TLS Handshake Protocol

C. HTTPS Traffic Filtering based on SNI

SNI filtering depends on checking the "server_name" value inside the SNI extension. As shown in Figure 2, the firewall inspects the SNI within the Client-Hello 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 Client-Hello message to pass toward the destination server, and further complete the TLS handshake.

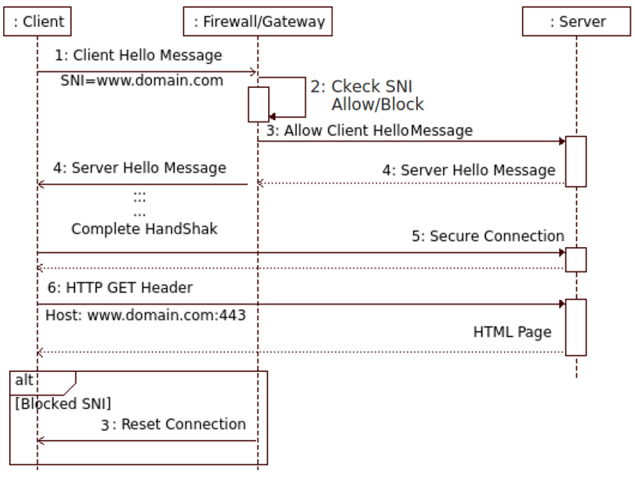

Fig. 2. SNI Filtering Sequence Diagram

Since SNI is a simply extracted value from Client-Hello messages, this is a lightweight process compared to statistical and website fingerprinting techniques to identify the source of

HTTPS traffic. It also preserves the privacy of users whose encrypted traffic is untouched. Many firewalls and Web Content Filtering solutions use SNI to identify and filter HTTPS traffic. For example, the Clavister[4] Web content filtering system supports both HTTP and HTTPS traffic, where HTTPS filtering is performed based on the SNI or on the Common Name (CN) in the server certificate. The Sphirewall[5] firewall system recently included SNI for HTTPS filtering with the 0.9.9.5 release (July,2013). It can determine the remote host used in a HTTPS web request by looking at the SNI parameter in the initial handshake. Sophos Unified Threat Management (UTM)[6] is another example of hardware and software network firewall system that recently (in UTM 9.2, released in September 2013) included the SNI filtering feature.

IV. Strategies to bypass SNI Filtering

The wide support and deployment of SNI makes it a prime choice to identify and filter HTTPS traffic nowadays. However, we will see in this section that such a SNI-based filtering is not reliable and we explain two strategies exploiting SNI weaknesses regarding (1) backward compatibility and (2) multiple services using a single certificate. These weaknesses can be used for circumventing firewalls relying on SNI to monitor and filter HTTPS traffic. We remind that the main principle of SNI filtering is checking the "server_name" value. If the domain is black-listed, the firewall will drop the Client-Hello message, aborting the TLS handshake.

A. Bypassing Based on Backward Compatibility

According to the related RFC [6], the SNI extension is designed to be backward compatible. This means that if the server does not recognize the SNI extension or the "server_name" value, it should still continue the handshake. The first bypassing strategy consists in sending a Client-Hello message without the SNI extension or with an alternative "server_name" value during the TLS handshake. In both cases, the firewall will not find the blacklisted name and the Client-Hello 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 customized an HTTPS Java client provided with the Oracle 8 JDK and added the ability to alter the SNI extension through TLS Socket configuration. We tested two different scenarios exploiting the backward compatibility.

The first scenario removes the SNI extension from the Client-Hello message. The TLS handshake is initialized with a Client-Hello message without SNI and then is sent an HTTP host header 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.

The second scenario inserts an alternative "server_name". 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 example 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, for example trying 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`

To better focus on the TLS exchange, we did not mention the prior DNS request, assuming that no DNS filtering is in place or that it has been bypassed using the specific DNS features of Escape (see section II-B).

B. Bypassing Based on Shared Server Certificate

One property of the certificate standard X.509 is an alternative name field, which can hold a set of domain names using the same certificate for 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 the SNI for non-banned websites sharing the same server certificate.

From the firewall side, the connection seams 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 for Google services, since many services share the same Google server certificate like Youtube or Google Maps. Assuming Youtube is restricted by an SNI filtering but Maps isn't, the bypassing technique works as follows:

- TLS Socket with domain name and port: `TARGET_HTTPS_SERVER= maps.google.com and TARGET_HTTPS_PORT=443;`
- Create SNI object with:
  `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, we make 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 maps server certificate, both web sites sharing the same infrastructure. Once the traffic is encrypted, the firewall can not detect Youtube traffic anymore based on SNI.

At the end of this section, we conclude that SNI is not really reliable for filtering HTTPS traffic. Users can easily

[4]https://www.clavister.com/
[5]http://www.sphirewall.net
[6]http://www.sophos.com/en-us.aspx

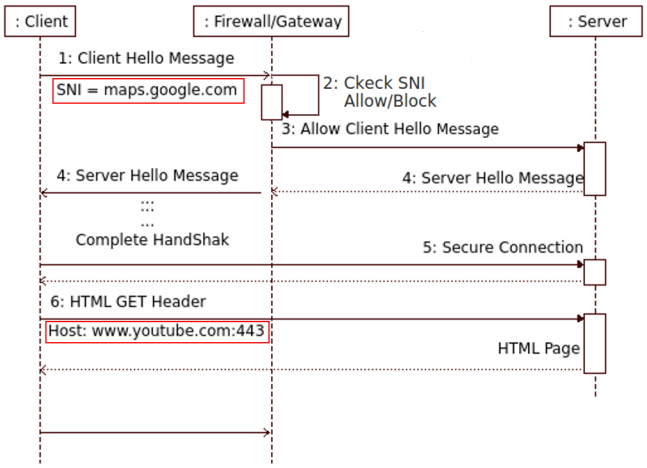

Fig. 3. Blocked Website Traffic Encapsulated in Another TLS Connection

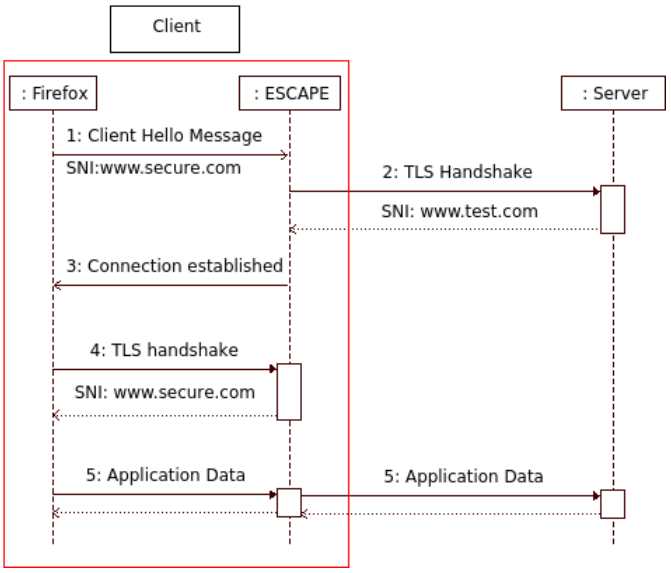

Fig. 4. Escape Plug-in Interactions

access prohibited resources by passing through the firewall and without even the need of a third party to circumvent the filtering system.

V. Implementation and Evaluation

A. Implementation of a Web Browser Plug-in

This section provides a direct application of the aforementioned results as an add-on named Escape[7] developed for the Firefox web browser. The choice of a web browser plug-in is motivated by the strong relation between our work and web browsing, the ease of use for 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[8] that intercepts TLS connections to perform a supplementary check on the certificates. It works as a local Man-In-The-Middle that intercepts 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. Figure 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 from Client-Hello packets.
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.
3) Escape notifies the Web browser that the handshake with the server is completed.
4) Escape presents its own certificate to the web browser and performs a new TLS handshake.
5) Web browser and web server communicates with secured TLS connections through Escape.

As presented in the previous section with our Java web browser, creating two TLS connections, one within the browser and another with the remote HTTPS server, is not mandatory at all to bypass SNI filtering. It is the technical solution we chose to get the control over the TLS handshake within the web browser in order to be able to modify Client-Hello packets and to access blocked websites.

B. Evaluation against SNI-based Firewalls

An evaluation environment was built to show the efficiency of our bypassing technique and of its implementation in Escape. A dedicated machine was configured with the following software firewalls that can filter HTTPS traffic based on SNI:

- 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[9] (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 SNI and Server certificate.
- IPFire[10] (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 Internet and the other connected to the local private network as shown in Figure5. The client machine with the web browser and Escape is directly connected to the firewall and has to pass through to access Internet. The client machine was configured and tested with Firefox 30 running on UBUNTU 12.04 and Windows 7. Firewalls are configured to reset the TLS connection for our list of blacklisted websites. We added to this list the top 20 of the most globally visited HTTPS websites (Google, Facebook, Youtube, Twitter, etc.) according to Alexa[11]. We mainly tested

[7] http://madynes.loria.fr/Research/Software
[8] http://convergence.io/
[9] https://www.untangle.com/
[10] http://www.ipfire.org/
[11] www.alexa.com

the SNI backward compatibility weakness (see section IV-A) for bypassing SNI filtering but also the shared server certificate issue (section IV-B) in the specific case of Google's websites.

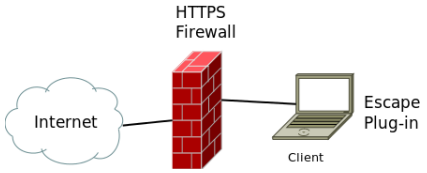

Fig. 5. SNI Filtering Evaluation Testbed

Table I shows the results of testing SNI filtering against the most visited HTTPS websites. The third column is the result of establishing a connection with a server without SNI (using the HTTPS Java client). The forth column is the result of sending an alternative SNI with Escape and the last column is the result of exploiting the shared certificate strategy for a subset of websites. In all cases the results show a positive evaluation of our bypassing techniques, at least when the SNI is removed or altered. The shared certificate has been only working so far for websites owned by Google but may work for other websites, depending on their shared server configuration.

TABLE I. LIST OF TOP 20 VISITED HTTPS WEBSITES TESTED FOR SNI FILTERING

No	Website	Without SNI	Alternative SNI	Shared Certificate
1	Google Search	Pass Filter	Pass Filter	Pass Filter by (mail.google.com)
2	Facebook	Pass Filter	Pass Filter	
3	Youtube	Pass Filter	Pass Filter	Pass Filter by (news.google.com)
4	Twitter	Pass Filter	Pass Filter	
5	LinkedIn	Pass Filter	Pass Filter	
6	Amazon	Pass Filter	Pass Filter	
7	Pinterest	Pass Filter	Pass Filter	
8	AdCash	Pass Filter	Pass Filter	
9	PayPal	Pass Filter	Pass Filter	
10	Blogger	Pass Filter	Pass Filter	Pass Filter by (mail.google.com)
11	NetFlix	Pass Filter	Pass Filter	
12	Flicker	Pass Filter	Pass Filter	
13	hootsuite	Pass Filter	Pass Filter	
14	Archive.org	Pass Filter	Pass Filter	
15	Soundcloud	Pass Filter	Pass Filter	
16	Github	Pass Filter	Pass Filter	
17	Chase	Pass Filter	Pass Filter	
18	usps.com	Pass Filter	Pass Filter	
19	AmiaricanExpress	Pass Filter	Pass Filter	
20	Dropbox	Pass Filter	Pass Filter	

The evaluation also shows that most of TLS servers implement backward compatibility following RFC6066. This explains why TLS servers always go further in the handshake process. For many years, SNI has not deployed widely in browsers and client-side systems and this give the opportunity to such systems accessing HTTPS services, but this behaviour is more questionable today.

VI. CONCLUSION

Internet users trust encrypted protocols such as HTTPS to prevent unauthorized viewing of their personal, financial and confidential information over the Web. Surveys show that websites are more and more being served over HTTPS. This motivates the development of new techniques to manage HTTPS traffic. In this paper we provide an in depth view of the latest technique for HTTPS traffic filtering that is based on the Server Name Indication (SNI) field of TLS and which has been recently implemented in many firewall solutions. We demonstrated why relying on this SNI extension for identifying and filtering HTTPS traffic is utterly flawed and can be easily cheated. We have shown that SNI has two weakness points, regarding (1) backward compatibility and (2) multiple services using a single certificate. We implemented our proof of concept exploiting SNI weaknesses inside a web browser plug-in for Firefox called "Escape" and that allows users to bypass such HTTPS filtering on their network. We successfully tested the plug-in against several firewalls and major Internet websites.

Our work has highlighted the fact that the management of HTTPS traffic deserves better solutions than the current analysis of SNI. Our future work will consist in researching more reliable ways to identify and manage HTTPS traffic, based on the learning and detection of specific traffic patterns of websites rather than on a single field.

REFERENCES

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

[2] 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. New York, NY, USA: ACM, 2013, pp. 291–304. [Online]. Available: http://doi.acm.org/10.1145/2504730.2504755

[3] M. Atighetchi, N. Soule, P. Pal, J. Loyall, A. Sinclair, and R. Grant, "Safe configuration of TLS connections," in *Communications and Network Security (CNS), 2013 IEEE Conference on*. IEEE, Oct 2013, pp. 415–422.

[4] Bloxx, "Protecting your network against risky SSL traffic," 2012. [Online]. Available: http://www.bloxx.com/downloads/bloxx_whitepaper_SSL_final.pdf

[5] J. Zittrain and J. G. Palfrey, "Internet filtering: The politics and mechanisms of control," *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)*, 2008.

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

[7] S. Valenti, D. Rossi, A. Dainotti, A. Pescap, A. Finamore, and M. Mellia, "Reviewing traffic classification," in *Data Traffic Monitoring and Analysis*, ser. Lecture Notes in Computer Science, E. Biersack, C. Callegari, and M. Matijasevic, Eds. Springer Berlin Heidelberg, 2013, vol. 7754, pp. 123–147.

[8] 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.

[9] 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.

[10] 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.

[11] 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.