Network Working Group
Internet-Draft
Intended status: Informational
Expires: November 21, 2018

C. Huitema
Private Octopus Inc.
E. Rescorla
RTFM, Inc.

May 20, 2018

Issues and Requirements for SNI Encryption in TLS
draft-ietf-tls-sni-encryption-03

Abstract

This draft describes the general problem of encryption of the Server Name Identification (SNI) parameter. The proposed solutions hide a Hidden Service behind a Fronting Service, only disclosing the SNI of the Fronting Service to external observers. The draft lists known attacks against SNI encryption, discusses the current "co-tenancy fronting" solution, and presents requirements for future TLS layer solutions.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Historically, adversaries have been able to monitor the use of web services through three channels: looking at DNS requests, looking at IP addresses in packet headers, and looking at the data stream between user and services. These channels are getting progressively closed. A growing fraction of Internet communication is encrypted, mostly using Transport Layer Security (TLS) [RFC5246]. Progressive deployment of solutions like DNS in TLS [RFC7858] mitigates the disclosure of DNS information. More and more services are colocated on multiplexed servers, loosening the relation between IP address and web service. However, multiplexed servers rely on the Service Name Information (SNI) to direct TLS connections to the appropriate service implementation. This protocol element is transmitted in

clear text. As the other methods of monitoring get blocked, monitoring focuses on the clear text SNI. The purpose of SNI encryption is to prevent that.

In the past, there have been multiple attempts at defining SNI encryption. These attempts have generally floundered, because the simple designs fail to mitigate several of the attacks listed in Section 3. In the absence of a TLS level solution, the most popular approach to SNI privacy is HTTP level fronting, which we discuss in Section 4.

2. History of the TLS SNI extension

The SNI extension was standardized in 2003 in [RFC3546] to facilitate management of "colocation servers", in which a multiple services shared the same IP address. A typical example would be mutiple web sites served by the same web server. The SNI extension carries the name of a specific server, enabling the TLS connection to be established with the desired server context. The current SNI extension specification can be found in [RFC6066].

The SNI specification allowed for different types of server names, but only the "hostname" variant was standardized and deployed. In that variant, the SNI extension carries the domain name of the target server. The SNI extension is carried in clear text in the TLS "Client Hello" message.

2.1. Unanticipated usage of SNI information

The SNI was defined to facilitate management of servers, but the developer of middleboxes soon found out that they could take advantage of the information. Many examples of such usage are reviewed in [I-D.mm-wg-effect-encrypt]. They include:

- Censorship of specific sites by "national firewalls",

- Content filtering by ISP blocking specific web sites in order to implement "parental controls", or to prevent access to fraudulent web sites, such as used for phishing,

- ISP assigning different QOS profiles to target services,

- Enterprise firewalls blocking web sites not deemed appropriate for work,

- Enterprise firewalls exempting specific web sites from MITM inspection, such as healthcare or financial sites for which inspection would intrude with the privacy of employees.

The SNI is probably also included in the general collection of metadata by pervasive surveillance actors.

2.2. SNI encryption timeliness

The clear-text transmission of the SNI was not flagged as a problem in the security consideration sections of [RFC3546], [RFC4366], or [RFC6066]. These specifications did not anticipate the abuses described in Section 2.1. One reason may be that, when these RFCs were written, the SNI information was available through a variety of other means.

Many deployments still allocate different IP addresses to different services, so that different services can be identified by their IP addresses. However, content distribution networks (CDN) commonly serve a large number of services through a small number of addresses.

The SNI carries the domain name of the server, which is also sent as part of the DNS queries. Most of the SNI usage described in Section 2.1 could also be implemented by monitoring DNS traffic or controlling DNS usage. But this is changing with the advent of DNS resolvers providing services like DNS over TLS [RFC7858] or DNS over HTTPS [I-D.ietf-doh-dns-over-https].

The common name component of the server certificate generally exposes the same name as the SNI. In TLS versions 1.0 [RFC2246], 1.1 [RFC4346], and 1.2 [RFC5246], the server send their certificate in clear text, ensuring that there would be limited benefits in hiding the SNI. But the transmission of the server certificate is protected in TLS 1.3 [I-D.ietf-tls-tls13].

The decoupling of IP addresses and server names, the deployment of DNS privacy, and the protection of server certificates transmissions all contribute to user privacy. Encrypting the SNI now will complete this push for privacy and make it much harder to censor specific internet services.

2.3. End-to-end alternatives

Deploying SNI encryption will help thwarting most the "unanticipated" SNI usages described in Section 2.1, including censorship and pervasive surveillance. It will also thwart functions that are sometimes described as legitimate. Most of these functions can however be realized by other means. For example, some DNS service providers offer customers the provision to "opt in" filtering services for parental control and phishing protection. Per stream QoS can be provided by a combination of packet marking and end to end agreements. Enterprises can deploy monitoring software to control

usage of the enterprises computers. As SNI encryption becomes common, we can expect more deployment of such "end to end" solutions.

3. Security and Privacy Requirements for SNI Encryption

Over the past years, there have been multiple proposals to add an SNI encryption option in TLS. Many of these proposals appeared promising, but were rejected after security reviews pointed plausible attacks. In this section, we collect a list of these known attacks.

3.1. Mitigate Replay Attacks

The simplest SNI encryption designs replace in the initial TLS exchange the clear text SNI with an encrypted value, using a key known to the multiplexed server. Regardless of the encryption used, these designs can be broken by a simple replay attack, which works as follow:

1- The user starts a TLS connection to the multiplexed server, including an encrypted SNI value.

2- The adversary observes the exchange and copies the encrypted SNI parameter.

3- The adversary starts its own connection to the multiplexed server, including in its connection parameters the encrypted SNI copied from the observed exchange.

4- The multiplexed server establishes the connection to the protected service, thus revealing the identity of the service.

One of the goals of SNI encryption is to prevent adversaries from knowing which Hidden Service the client is using. Successful replay attacks breaks that goal by allowing adversaries to discover that service.

3.2. Avoid Widely Shared Secrets

It is easy to think of simple schemes in which the SNI is encrypted or hashed using a shared secret. This symmetric key must be known by the multiplexed server, and by every users of the protected services. Such schemes are thus very fragile, since the compromise of a single user would compromise the entire set of users and protected services.

3.3. Prevent SNI-based Denial of Service Attacks

Encrypting the SNI may create extra load for the multiplexed server. Adversaries may mount denial of service attacks by generating random encrypted SNI values and forcing the multiplexed server to spend resources in useless decryption attempts.

It may be argued that this is not an important DOS avenue, as regular TLS connection attempts also require the server to perform a number of cryptographic operations. However, in many cases, the SNI decryption will have to be performed by a front end component with limited resources, while the TLS operations are performed by the component dedicated to their respective services. SNI based DOS attacks could target the front end component.

3.4. Do not stick out

In some designs, handshakes using SNI encryption can be easily differentiated from "regular" handshakes. For example, some designs require specific extensions in the Client Hello packets, or specific values of the clear text SNI parameter. If adversaries can easily detect the use of SNI encryption, they could block it, or they could flag the users of SNI encryption for special treatment.

In the future, it might be possible to assume that a large fraction of TLS handshakes use SNI encryption. If that was the case, the detection of SNI encryption would be a lesser concern. However, we have to assume that in the near future, only a small fraction of TLS connections will use SNI encryption.

3.5. Forward Secrecy

The general concerns about forward secrecy apply to SNI encryption just as well as to regular TLS sessions. For example, some proposed designs rely on a public key of the multiplexed server to define the SNI encryption key. If the corresponding private key was compromised, the adversaries would be able to process archival records of past connections, and retrieve the protected SNI used in these connections. These designs failed to maintain forward secrecy of SNI encryption.

3.6. Proper Security Context

We can design solutions in which the multiplexed server or a fronting service act as a relay to reach the protected service. Some of those solutions involve just one TLS handshake between the client and the multiplexed server, or between the client and the fronting service.

The master secret is verified by verifying a certificate provided by either of these entities, but not by the protected service.

These solutions expose the client to a Man-In-The-Middle attack by the multiplexed server or by the fronting service. Even if the client has some reasonable trust in these services, the possibility of MITM attack is troubling.

The multiplexed server or the fronting services could be pressured by adversaries. By design, they could be forced to deny access to the protected service, or to divulge which client accessed it. But if MITM is possible, the adversaries would also be able to pressure them into intercepting or spoofing the communications between client and protected service.

3.7. Fronting Server Spoofing

Adversaries could mount an attack by spoofing the Fronting Service. A spoofed Fronting Service could act as a "honeypot" for users of hidden services. At a minimum, the fake server could record the IP addresses of these users. If the SNI encryption solution places too much trust on the fronting server, the fake server could also serve fake content of its own choosing, including various forms of malware.

There are two main channels by which adversaries can conduct this attack. Adversaries can simply try to mislead users into believing that the honeypot is a valid Fronting Server, especially if that information is carried by word of mouth or in unprotected DNS records. Adversaries can also attempt to hijack the traffic to the regular Fronting Server, using for example spoofed DNS responses or spoofed IP level routing, combined with a spoofed certificate.

3.8. Supporting multiple protocols

The SNI encryption requirement do not stop with HTTP over TLS. Multiple other applications currently use TLS, including for example SMTP [RFC5246], DNS [RFC7858], or XMPP [RFC7590]. These applications too will benefit of SNI encryption. HTTP only methods like those described in Section 4.1 would not apply there. In fact, even for the HTTPS case, the HTTPS tunneling service described in Section 4.1 is compatible with HTTP 1.0 and HTTP 1.1, but interacts awkwardly with the multiple streams feature of HTTP 2.0 [RFC7540]. This points to the need of an application agnostic solution, that would be implemented fully in the TLS layer.

3.8.1. Hiding the Application Layer Protocol Negotiation

The Application Layer Protocol Negotiation (ALPN) parameters of TLS allow implementations to negotiate the application layer protocol used on a given connection. TLS provides the ALPN values in clear text during the initial handshake. While exposing the ALPN does not create the same privacy issues as exposing the SNI, there is still a risk. For example, some networks may attempt to block applications that they do not understand, or that they wish users would not use.

In a sense, ALPN filtering could be very similar to the filtering of specific port numbers exposed in some network. This filtering by ports has given rise to evasion tactics in which various protocols are tunneled over HTTP in order to use open ports 80 or 443. Filtering by ALPN would probably beget the same responses, in which the applications just move over HTTP, and only the HTTP ALPN values are used. Applications would not need to do that if the ALPN was hidden in the same way as the SNI.

It is thus desirable that SNI Encryption mechanisms be also able hide the ALPN.

3.8.2. Support other transports than HTTP

The TLS handshake is also used over other transports such as UDP with both DTLS [I-D.ietf-tls-dtls13] and QUIC [I-D.ietf-quic-tls]. The requirement to encrypt the SNI apply just as well for these transports as for TLS over TCP.

This points to a requirement for SNI Encryption mechanisms to also be applicable to non-TCP transports such as DTLS or QUIC.

3.9. Fail to fronting

It is easy to imagine designs in which the client sends some client hello extension that points to a secret shared by client and hidden server. If that secret is incorporated into the handshake secret, the exchange will only succeeds if the connection truly ends at the hidden server. The exchange will fail if the extension is stripped by an MITM, and the exchange will also fail if an adversary replays the extension in a Client Hello.

The problem with that approach is clear. Adversaries that replay the extension can test whether the client truly wanted to access the fronting server, or was simply using that fronting server as an access gateway to something else. The adversaries will not know what hidden service the client was trying to reach, but they can guess.

They can also start directly interrogate the user, or other unpleasant alternatives.

When designing SNI encryption schemes, we have to take into account attacks that strip parameters from the Client Hello, or replay attacks. In both cases, the desired behavior is to fall back to a connection with the fronting server, so there is no visble difference between a regular connection to that server and an attempt to reach the hidden server.

4. HTTP Co-Tenancy Fronting

In the absence of TLS level SNI encryption, many sites rely on an "HTTP Co-Tenancy" solution. The TLS connection is established with the fronting server, and HTTP requests are then sent over that connection to the hidden service. For example, the TLS SNI could be set to "fronting.example.com", the fronting server, and HTTP requests sent over that connection could be directed to "hidden.example.com/some-content", accessing the hidden service. This solution works well in practice when the fronting server and the hidden server are 'co-tenant" of the same multiplexed server.

The HTTP fronting solution can be deployed without modification to the TLS protocol, and does not require using any specific version of TLS. There are however a few issues regarding discovery, client implementations, trust, and applicability:

- o The client has to discover that the hidden service can be accessed through the fronting server.

- o The client browser's has to be directed to access the hidden service through the fronting service.

- o Since the TLS connection is established with the fronting service, the client has no proof that the content does in fact come from the hidden service. The solution does thus not mitigate the context sharing issues described in Section 3.6.

- o Since this is an HTTP level solution, it would not protected non HTTP protocols such as DNS over TLS [RFC7858] or IMAP over TLS [RFC2595].

The discovery issue is common to pretty much every SNI encryption solution. The browser issue may be solved by developing a browser extension that support HTTP Fronting, and manages the list of fronting services associated with the hidden services that the client uses. The multi-protocol issue can be mitigated by using implementation of other applications over HTTP, such as for example

DNS over HTTPS [I-D.hoffman-dns-over-https]. The trust issue, however, requires specific developments.

4.1. HTTPS Tunnels

The HTTP Fronting solution places a lot of trust in the Fronting Server. This required trust can be reduced by tunnelling HTTPS in HTTPS, which effectively treats the Fronting Server as an HTTP Proxy. In this solution, the client establishes a TLS connection to the Fronting Server, and then issues an HTTP Connect request to the Hidden Server. This will establish an end-to-end HTTPS over TLS connection between the client and the Hidden Server, mitigating the issues described in Section 3.6.

The HTTPS in HTTPS solution requires double encryption of every packet. It also requires that the fronting server decrypts and relay messages to the hidden server. Both of these requirements make the implementation onerous.

4.2. Delegation Control

Clients would see their privacy compromised if they contacted the wrong fronting server to access the hidden service, since this wrong server could disclose their access to adversaries. This requires a controlled way to indicate which fronting ferver is acceptable by the hidden service.

This problem is both similar and different from the "fronting server spoofing" attack described in Section 3.7. Here, the spoofing would be performed by distributing fake advice, such as "to reach example hidden.example.com, use fake.example.com as a fronting server", when "fake.example.com" is under the control of an adversary.

In practice, this attack is well mitigated when the hidden service is accessed through a specialized application. The name of the fronting server can then be programmed in the code of the application. But the attack is much harder to mitigate when the hidden service has to be accessed through general purpose web browsers. The browsers will need a mechanism to obtain the fronting server indication in a secure way.

There are several proposed solutions to this problem, such as creating a special form of certificate to codify the relation between fronting and hidden server, or obtaining the relation between hidden and fronting service through the DNS, possibly using DNSSEC to avoid spoofing.

We can observe that content distribution network have a similar requirement. They need to convince the client that "www.example.com" can be accessed through the seemingly unrelated "cdn-node-xyz.example.net". Most CDN have deployed DNS-based solutions to this problem.

5. Security Considerations

Replacing clear text SNI transmission by an encrypted variant will improve the privacy and reliability of TLS connections, but the design of proper SNI encryption solutions is difficult. This document does not present the design of a solution, but provide guidelines for evaluating proposed solutions.

This document lists a number of attacks against SNI encryption in Section 3, and also in Section 4.2, and presents a list of requirements to mitigate these attacks. The current HTTP based solutions described in Section 4 only meet some of these requirements. In practice, it may well be that no solution can meet every requirement, and that practical solutions will have to make some compromises.

In particular, the requirement to not stick out presented in Section 3.4 may have to be lifted, especially if for proposed solutions that could quickly reach large scale deployments.

6. IANA Considerations

This draft does not require any IANA action.

7. Acknowledgements

A large part of this draft originates in discussion of SNI encryption on the TLS WG mailing list, including comments after the tunneling approach was first proposed in a message to that list: <https://mailarchive.ietf.org/arch/msg/tls/tXvdcqnogZgqmdfCugrV8M90Ftw>.

Thanks to Daniel Kahn Gillmor for a pretty detailed review of the initial draft.

8. References

8.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

8.2. Informative References

[I-D.hoffman-dns-over-https]
Hoffman, P. and P. McManus, "DNS Queries over HTTPS", draft-hoffman-dns-over-https-01 (work in progress), June 2017.

[I-D.ietf-doh-dns-over-https]
Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DOH)", draft-ietf-doh-dns-over-https-08 (work in progress), May 2018.

[I-D.ietf-quic-tls]
Thomson, M. and S. Turner, "Using Transport Layer Security (TLS) to Secure QUIC", draft-ietf-quic-tls-11 (work in progress), April 2018.

[I-D.ietf-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-ietf-tls-dtls13-26 (work in progress), March 2018.

[I-D.mm-wg-effect-encrypt]
Moriarty, K. and A. Morton, "Effects of Pervasive Encryption on Operators", draft-mm-wg-effect-encrypt-25 (work in progress), March 2018.

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, DOI 10.17487/RFC2246, January 1999, <https://www.rfc-editor.org/info/rfc2246>.

[RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, DOI 10.17487/RFC2595, June 1999, <https://www.rfc-editor.org/info/rfc2595>.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, DOI 10.17487/RFC3546, June 2003, <https://www.rfc-editor.org/info/rfc3546>.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, DOI 10.17487/RFC4346, April 2006, <https://www.rfc-editor.org/info/rfc4346>.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, DOI 10.17487/RFC4366, April 2006, <https://www.rfc-editor.org/info/rfc4366>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7590] Saint-Andre, P. and T. Alkemade, "Use of Transport Layer Security (TLS) in the Extensible Messaging and Presence Protocol (XMPP)", RFC 7590, DOI 10.17487/RFC7590, June 2015, <https://www.rfc-editor.org/info/rfc7590>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

Authors' Addresses

Christian Huitema
Private Octopus Inc.
Friday Harbor WA 98250
U.S.A

Email: huitema@huitema.net

Eric Rescorla
RTFM, Inc.
U.S.A

Email: ekr@rtfm.com