Internet Engineering Task Force (IETF)
Request for Comments: 8744
Category: Informational
ISSN: 2070-1721

C. Huitema
Private Octopus Inc.
July 2020

Issues and Requirements for Server Name Identification (SNI) Encryption in TLS

Abstract

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

In practice, it may well be that no solution can meet every requirement and that practical solutions will have to make some compromises.

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 primary 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) [RFC8446]. Progressive deployment of solutions like DNS over TLS [RFC7858] and DNS over HTTPS [RFC8484] mitigates the disclosure of DNS information. More and more services are colocated on multiplexed servers, loosening the relation between IP address and web service. For example, in virtual hosting solutions, multiple services can be hosted as co-tenants on the same server, and the IP address and port do not uniquely identify a service. In cloud or Content Delivery Network (CDN) solutions, a given platform hosts the services or servers of a lot of organizations, and looking up what netblock an IP address belongs to reveals little. However, multiplexed servers rely on the Server Name Information (SNI) TLS extension [RFC6066] to direct connections to the appropriate service implementation. This protocol element is transmitted in cleartext. As the other methods of monitoring get blocked, monitoring focuses on the cleartext SNI. The purpose of SNI encryption is to prevent that and aid privacy.

Replacing cleartext 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. 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 for web services is HTTP-level fronting, which we discuss in Section 4.

This document does not present the design of a solution but provides guidelines for evaluating proposed solutions. (The review of HTTP-level solutions in Section 4 is not an endorsement of these solutions.) The need for related work on the encryption of the Application-Layer Protocol Negotiation (ALPN) parameters of TLS is discussed in Section 3.7.1.

2. History of the TLS SNI Extension

The SNI extension was specified in 2003 in [RFC3546] to facilitate management of "colocation servers", in which multiple services shared the same IP address. A typical example would be multiple websites 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, though only the "hostname" variant was specified and deployed. In that variant, the SNI extension carries the domain name of the target server. The SNI extension is carried in cleartext in the TLS "ClientHello" message.

2.1. Unanticipated Usage of SNI Information

The SNI was defined to facilitate management of servers, but the developers of middleboxes found out that they could take advantage of the information. Many examples of such usage are reviewed in [RFC8404]. Other examples came out during discussions of this document. They include:

- Filtering or censoring specific services for a variety of reasons

- Content filtering by network operators or ISPs blocking specific websites, for example, to implement parental controls or to prevent access to phishing or other fraudulent websites

- ISP assigning different QoS profiles to target services

- Firewalls within enterprise networks blocking websites not deemed appropriate for work

- Firewalls within enterprise networks exempting specific websites from man-in-the-middle (MITM) inspection, such as healthcare or financial sites for which inspection would intrude on the privacy of employees

The SNI is probably also included in the general collection of metadata by pervasive surveillance actors [RFC7258], for example, to identify services used by surveillance targets.

2.2. SNI Encryption Timeliness

The cleartext transmission of the SNI was not flagged as a problem in the Security Considerations sections of [RFC3546], [RFC4366], or [RFC6066]. These specifications did not anticipate the alternative usage 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, such as tracking IP addresses, DNS names, or server certificates.

Many deployments still allocate different IP addresses to different services, so that different services can be identified by their IP addresses. However, CDNs commonly serve a large number of services through a comparatively 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 [RFC8484].

The subjectAltName extension of type dNSName of the server certificate (or in its absence, the common name component) exposes the same name as the SNI. In TLS versions 1.0 [RFC2246], 1.1 [RFC4346], and 1.2 [RFC5246], servers send certificates in cleartext, ensuring that there would be limited benefits in hiding the SNI. However, in TLS 1.3 [RFC8446], server certificates are encrypted in transit. Note that encryption alone is insufficient to protect server certificates; see Section 3.1 for details.

The decoupling of IP addresses and server names, deployment of DNS privacy, and protection of server certificate transmissions all contribute to user privacy in the face of an RFC 7258-style adversary [RFC7258]. Encrypting the SNI complements this push for privacy and makes it harder to censor or otherwise provide differential treatment to specific Internet services.

2.3. End-to-End Alternatives

Deploying SNI encryption helps thwart most of the unanticipated SNI usages, including censorship and pervasive surveillance, but it also will break or reduce the efficacy of the operational practices and techniques implemented in middleboxes, as described in Section 2.1. Most of these functions can, however, be realized by other means. For example, some DNS service providers offer customers the provision to "opt in" to filtering services for parental control and phishing protection. Per-stream QoS could be provided by a combination of packet marking and end-to-end agreements. As SNI encryption becomes common, we can expect more deployment of such "end-to-end" solutions.

At the time of this writing, enterprises have the option of installing a firewall performing SNI filtering to prevent connections to certain websites. With SNI encryption, this becomes ineffective. Obviously, managers could block usage of SNI encryption in enterprise computers, but this wide-scale blocking would diminish the privacy protection of traffic leaving the enterprise, which may not be desirable. Enterprise managers could rely instead on filtering software and management software deployed on the enterprise's computers.

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. A review of the TLS mailing list archives shows that many of these proposals appeared promising but were rejected after security reviews identified plausible attacks. In this section, we collect a list of these known attacks.

3.1. Mitigate Cut-and-Paste Attacks

The simplest SNI encryption designs replace the cleartext SNI in the initial TLS exchange 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 cut-and-paste attack, which works as follows:

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, which sends its certificate, 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 cut-and-paste attacks break 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 user 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 (DoS) 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 avenue for DoS attacks, 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 ClientHello packets or specific values of the cleartext 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 were 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.

This requirement to not stick out may be difficult to meet in practice, as noted in Section 5.

3.5. Maintain Forward Secrecy

TLS 1.3 [RFC8446] is designed to provide forward secrecy, so that (for example) keys used in past sessions will not be compromised even if the private key of the server is compromised. The general concerns about forward secrecy apply to SNI encryption as well. 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 should be compromised, the adversaries would be able to process archival records of past connections and retrieve the protected SNI used in these connections. These designs fail to maintain forward secrecy of SNI encryption.

3.6. Enable Multi-party Security Contexts

We can design solutions in which a fronting service acts as a relay to reach the protected service. Some of those solutions involve just one TLS handshake between the client and the fronting service. The master secret is verified by verifying a certificate provided by the fronting service but not by the protected service. These solutions expose the client to a MITM attack by the fronting service. Even if the client has some reasonable trust in this service, the possibility of a MITM attack is troubling.

There are other classes of solutions in which the master secret is verified by verifying a certificate provided by the protected service. These solutions offer more protection against a MITM attack by the fronting service. The downside is that the client will not verify the identity of the fronting service, which enables fronting server spoofing attacks, such as the "honeypot" attack discussed below. Overall, end-to-end TLS to the protected service is preferable, but it is important to also provide a way to authenticate the fronting service.

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

Adversaries could also 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.

3.7. Support Multiple Protocols

The SNI encryption requirement does not stop with HTTP over TLS. Multiple other applications currently use TLS, including, for example, SMTP [RFC3207], DNS [RFC7858], IMAP [RFC8314], and the Extensible Messaging and Presence Protocol (XMPP) [RFC7590]. These applications, too, will benefit from 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 [RFC7540]. This points to the need for an application-agnostic solution, which would be implemented fully in the TLS layer.

3.7.1. 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 cleartext 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 networks. 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 were hidden in the same way as the SNI.

In addition to hiding the SNI, it is thus desirable to also hide the ALPN. Of course, this implies engineering trade-offs. Using the same technique for hiding the ALPN and encrypting the SNI may result in excess complexity. It might be preferable to encrypt these independently.

3.7.2. Supporting Other Transports than TCP

The TLS handshake is also used over other transports, such as UDP with both DTLS [DTLS-1.3] and QUIC [QUIC]. The requirement to encrypt the SNI applies 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.

4. HTTP Co-tenancy Fronting

In the absence of TLS-level SNI encryption, many sites rely on an "HTTP co-tenancy" solution, often referred to as "domain fronting" [DOMFRONT]. 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" (accessing the hidden service). This solution works well in practice when the fronting server and the hidden server are "co-tenants" of the same multiplexed server.

The HTTP domain 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:

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

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

- Since the TLS connection is established with the fronting service, the client has no cryptographic proof that the content does, in fact, come from the hidden service. Thus, the solution does not mitigate the context sharing issues described in Section 3.6. Note that this is already the case for co-tenanted sites.

- Since this is an HTTP-level solution, it does not protect non-HTTP protocols, as discussed in Section 3.7.

The discovery issue is common to most SNI encryption solutions. The browser issue was solved in [DOMFRONT] by implementing domain fronting as a pluggable transport for the Tor browser. The multi-protocol issue can be mitigated by implementing other applications over HTTP, for example, DNS over HTTPS [RFC8484]. The trust issue, however, requires specific developments.

4.1. HTTPS Tunnels

The HTTP domain fronting solution places a lot of trust in the fronting server. This required trust can be reduced by tunneling 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 decrypt 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 server is acceptable by the hidden service.

This problem is similar to the "word of mouth" variant of the "fronting server spoofing" attack described in Section 3.6. The spoofing would be performed by distributing fake advice, such as "to reach 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 harder to mitigate when the hidden service has to be accessed through general-purpose web browsers.

There are several proposed solutions to this problem, such as creating a special form of certificate to codify the relation between the fronting and hidden server or obtaining the relation between the hidden and fronting service through the DNS, possibly using DNSSEC, to avoid spoofing. The experiment described in [DOMFRONT] solved the issue by integrating with the Lantern Internet circumvention tool.

We can observe that CDNs 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 CDNs have deployed DNS-based solutions to this problem. However, the CDN often holds the authoritative certificate of the origin. There is, simultaneously, verification of a relationship between the origin and the CDN (through the certificate) and a risk that the CDN can spoof the content from the origin.

4.3. Related Work

The ORIGIN frame defined for HTTP/2 [RFC8336] can be used to flag content provided by the hidden server. Secondary certificate authentication [HTTP2-SEC-CERTS] can be used to manage authentication of hidden server content or to perform client authentication before accessing hidden content.

5. Security Considerations

This document lists a number of attacks against SNI encryption in Sections 3 and 4.2 and presents a list of requirements to mitigate these attacks. 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 for proposed solutions that could quickly reach large-scale deployments.

Replacing cleartext SNI transmission by an encrypted variant will break or reduce the efficacy of the operational practices and techniques implemented in middleboxes, as described in Section 2.1. As explained in Section 2.3, alternative solutions will have to be developed.

6. IANA Considerations

This document has no IANA actions.

7. Informative References

[DOMFRONT] Fifield, D., Lan, C., Hynes, R., Wegmann, P., and V. Paxson, "Blocking-resistant communication through domain fronting", DOI 10.1515/popets-2015-0009, 2015, <https://www.bamsoftware.com/papers/fronting/>.

[DTLS-1.3] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-38, 29 May 2020, <https://tools.ietf.org/html/draft-ietf-tls-dtls13-38>.

[HTTP2-SEC-CERTS] Bishop, M., Sullivan, N., and M. Thomson, "Secondary Certificate Authentication in HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2-secondary-certs-06, 14 May 2020, <https://tools.ietf.org/html/draft-ietf-httpbis-http2-secondary-certs-06>.

[QUIC] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-29, 9 June 2020, <https://tools.ietf.org/html/draft-ietf-quic-tls-29>.

[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>.

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, DOI 10.17487/RFC3207, February 2002, <https://www.rfc-editor.org/info/rfc3207>.

[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>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

[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>.

[RFC8314] Moore, K. and C. Newman, "Cleartext Considered Obsolete: Use of Transport Layer Security (TLS) for Email Submission and Access", RFC 8314, DOI 10.17487/RFC8314, January 2018, <https://www.rfc-editor.org/info/rfc8314>.

[RFC8336] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <https://www.rfc-editor.org/info/rfc8336>.

[RFC8404] Moriarty, K., Ed. and A. Morton, Ed., "Effects of Pervasive Encryption on Operators", RFC 8404, DOI 10.17487/RFC8404, July 2018, <https://www.rfc-editor.org/info/rfc8404>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8484] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DoH)", RFC 8484, DOI 10.17487/RFC8484, October 2018, <https://www.rfc-editor.org/info/rfc8484>.

Acknowledgements

A large part of this document originated 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 Eric Rescorla for his multiple suggestions, reviews, and edits to the successive draft versions of this document.

Thanks to Daniel Kahn Gillmor for a pretty detailed review of the initial draft of this document. Thanks to Bernard Aboba, Mike Bishop, Alissa Cooper, Roman Danyliw, Stephen Farrell, Warren Kumari, Mirja Kuelewind, Barry Leiba, Martin Rex, Adam Roach, Meral Shirazipour, Martin Thomson, Eric Vyncke, and employees of the UK National Cyber Security Centre for their reviews. Thanks to Chris Wood, Ben Kaduk, and Sean Turner for helping move this document toward publication.

Author's Address

Christian Huitema
Private Octopus Inc.
Friday Harbor, WA 98250
United States of America

Email: huitema@huitema.net