Encrypted SNI: Privacy and Security

No Name

October 22, 2019

Contents

1 Introduction

The TLS Server Name Indication extension [**?**] is an increasingly important part of the TLS protocol. Modern TLS server deployments often offer multiple certificates behind a single IP address. The SNI helps servers choose which certificate to choose for a given connection. However, this extension also leaks the server name to any on-path observer. With recent pushes to encrypt DNS and protect names from such adversaries, the SNI extension is a privacy problem for clients. The Encrypted SNI extension [1] attempts to address this privacy problem by encrypting the SNI in transit. However, to date, asserting correctness and privacy properties of the protocol proved difficult.

Formal Analysis.

writeme

Our Contributions.

writeme

2 Encrypted SNI Overview

Encrypted SNI is a tool for hiding server names from network connections. There are several operational goals for Encrypted SNI [?], described below:

- Avoid widely-deployed shared secrets: One approach to the problem would be for all clients and servers to share a secret that encrypts (and decrypts) the SNI. However, any client in this set of trusted peers could then decrypt the SNI of others. Moreover, compromise of any node in possession of the secret puts all members at risk. Thus, ESNI requires public key encryption.
- Work with non-ESNI servers to avoid fallback: Without the need for fallback, ESNI is a simple ECIES-like protocol, wherein the SNI is encrypted under a public key of the service provider.
- Do not introduce extra round trips: Encrypting the SNI must not come at the cost of extra round trips. For example, one possible approach is to SNI-based certificate authentication at the application protocol layer, e.g., using HTTP/2 Secondary Certificates [?], after the TLS connection finishes and is authenticated with a "public name." While this may work, the extra latency cost may be prohibitvely expensive for certain clients.
- Forward secrecy: Ideally, SNI encryption would have some amount of forward secrecy. However, as SNI encryption cannot introduce additional round trips, forward secrecy is not possible using public key encryption primitives such as ECIES [?] or HPKE [?]
- Prevent SNI-based DoS attacks: A consequence of using public key encryption is that servers must perform a public key operation without having validated the client. HelloRetryRequests may help dampen the effects of DoS attacks, though these come at the cost of introducing additional complexity into the protocol. See Section 2.2 for more details.
- Mitigate replay attacks: Encrypted SNI values must not be replayable from one ClientHello to another, otherwise an attacker could use an ESNI value from a victim client message in its own ClientHello.
- Support shared and split mode: Client-facing servers which use the SNI to determine the target service may not be the entity which terminates the TLS connection. ESNI should therefore support proxies which route TLS connections to backend or origin services. This suggests two possible deployment models for ESNI, referred to as shared and split mode, shown in Figure **??**.

XXX: mode figure

2.1 Draft-04 Design Overview

XXX: include figure of the protocol

Figure 1: Early ESNI Client Reaction Attack

2.2 Security and Privacy Goals

ESNI assumes a standard active and on-path Dolev-Yao attacker that can arbitrarily drop, tamper, replay, and forward messages from clients. Fundamentally, a TLS handshake that negotiates ESNI should leak no more information than one which did not negotiate ESNI in the presence of this adversary. This means there are at least two necessary requirements for ESNI:

1. SNI agreement: A successful TLS handshake implies agreement on the SNI transmitted. This means, among other things, that the client authenticated the server's certificate using the SNI, and that both client and server share the same view of the SNI negotiated.

2. SNI privacy: A successful TLS handshake that negotiates ESNI does so without leaking any information about the underlying SNI. Moreover, the SNI is known only to the client and server (or any recipient of the private ESNI key). We do not require forward secrecy for the SNI encryption.

We may optionally want to hide the fact that ESNI was negotiated, as per the "do not stick out" goal. However, this is primarily only deployment concern. Furthermore, we may also want to hide the fact that a client offered ESNI in its handshake. This may be useful for clients that wish to GREASE [**?**] the extension.

2.3 Known Attacks

Early versions of ESNI did not achieve these goals. For example, the first version was vulnerable to a certificate-based client reaction attack shown in Figure 1. The core problem was that servers did not signal to clients whether or not they processed the ESNI extension. This allowed any MITM to complete the handshake – prior to authentication – on behalf of the server with a certificate of its choosing. Adding a nonce to the server's response prohibits this as clients can then check whether or not the nonce is correct.

Despite this fix, draft-04 of ESNI does not achieve the necessary requirements stated above. To show why, we illustrate some more attacks on the protocol.

Probing Attacks : Differences in service configurations can leak information.

HelloRetryRequest Mix and Match : Attacker-chosen key share, client-chosen SNI.

Server Reaction Attacks : Use ticket and server reaction for dictionary attack.

The number of edge cases makes it clear that a formal model of the protocol is needed.

3 Core Protocol

At its core, ESNI is a protocol between a client and server that works as described in Figure **??**. It aims to provide the following guarantees:

- Client: TLS handshake secret known by entity which has the private handshake key share (y), corresponding PSK, private ESNI decryption key, and ENSI nonce.
- Server: TLS handshake secret known by entity which has the private handshake key share (x), corresponding PSK, and ENSI nonce.
- Client and server both agree on the same TLS handshake secret and transcript.

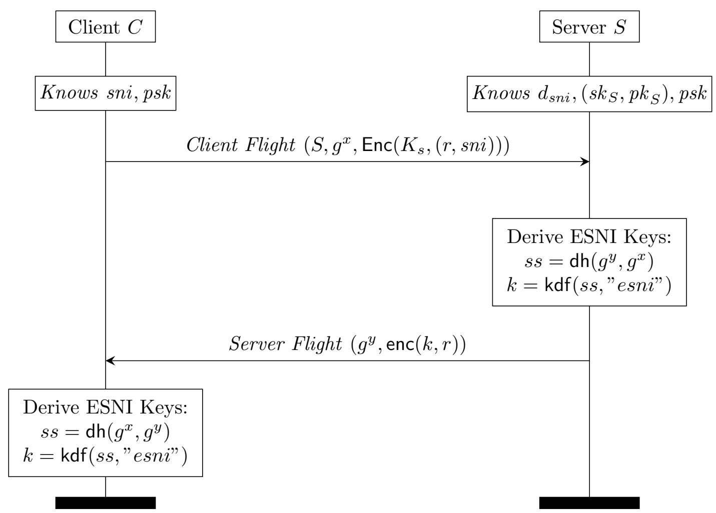

Figure 2: Simple ESNI Protocol without resumption or HelloRetryRequest support.

Moreover, these guarantees must hold for all TLS handshake patterns, including: normal handshakes, resumption handshakes, and HelloRetryRequest handshakes. Considering the full variant, there are five secrets that influence the handshake: private signing key, secret Diffie Hellman key shares, pre-shared key(s), ESNI decryption key (K_S), and the ESNI nonce. Informally, we require that these values endorse each other in order to avoid the attacks described in Section 2.3.

XXX: figure of secret relationship

For example, consider the ticket-based server reaction attack, wherein the ESNI nonce is not bound to a ClientHello PSK. This leads to a situation wherein a server receives a ClientHello with an attacker-controlled PSK meant for one

SNI, yet an ESNI carrying an encryption of a different SNI. If said server then *checks* that these SNIs for equality and reacts differently in response, the SNI value may leak. However, if these values are properly bound, then such a check will not possible yield a negative answer (for honestly generated ClientHello messages), and therefore the reaction attack vanishes.

As another example, consider the certificate-based client reaction attack.

finishme

3.1 Necessary Properties

Joint binding of these secrets yields the following properties:

- Backward binding: ESNI contents are bound to the entire ClientHello such that any modification is detectable by servers. ESNI contents are *backward bound* to a ClientHello if it is not possible to modify a CH in any way without causing an ESNI check to fail.

- Forward binding: TLS handshake secrets are bound to the ESNI contents such that knowledge of both ESNI secret(s) and one of the TLS key shares is needed to derive the handshake secrets. ESNI is *forward bound* if it is not possible to learn the TLS handshake secret without both the Diffie Hellman shared secret and ESNI shared secret.

As a consequence of forward binding and the need to interoperate with ESNI-incapable servers, we also require a signalling mechanism for clients to determine whether or not the ESNI contents were used to protect the handshake secret. (Trial decryption is always an option, though other more practical solutions exist.)

Beyond these critical properties, it must also be the case that observable information, which includes messages sent on the wire and the *actions* of clients and servers using ESNI, does not leak information about the SNI. Indeed, the reaction attack described above was due in part to such an information leak.[1] We must capture this notion of information indistinguishability for completeness. We do so via *message indistinguishability* and *action indistinguishability*. Informally, message indistinguishability means that all messages written on the wire do not vary based on SNI. Similarly, action indistinguishability means that all node behavior as observed by Adv does not vary based on SNI (or any SNI-influenced value used in the protocol).

add more rigorous definition of these

describe how message indistinguishability is a property of configuration, and give the functions discussed IRL

[1]Fortunately, joint binding resolved the problem by removing a branch in the way servers handle ClientHello messages.

3.2 Formal Model

In this section, we present a model for ESNI based on ProVerif [**?**, **?**]. ProVerif analyzes symbolic protocol models using processes to represent entities which communicate using messages sent over public channels. Processes can trigger security events representing attacks or critical steps of the target protocol, e.g., TLS connection establishment. Moreover, processes can save messages in lookup tables for use later on. This is useful for storing long-term keying material, such as ESNI and certificate private keys.

Our ESNI model accounts for backward and forward binding. It also accounts for action indistinguishability for client and server processes. It does not account for message indistinguishability, as this is something largely determined by configuration.

3.2.1 Long Term Secrets

XXX

3.2.2 Client ESNI Process

XXX

3.2.3 Server ESNI Process

XXX

4 Summary of Proposals

There are two proposals that conform to the core ESNI protocol described above. They are summarized in the following sections.

4.1 ESNI Proxy Transformation

XXX

4.2 ESNI PSK Binders and Key Schedule Injection

XXX

4.3 Comparison

XXX

References

[1] Eric Rescorla, Kazuho Oku, Nick Sullivan, and Christopher A. Wood. Encrypted Server Name Indication for TLS 1.3. Internet-Draft draft-ietf-tls-esni-04, Internet Engineering Task Force, July 2019. Work in Progress.