TLS Extensions for Protecting SNI
draft-kazuho-protected-sni-00

Abstract

This memo introduces TLS extensions and a DNS Resource Record Type that can be used to protect attackers from obtaining the value of the Server Name Indication extension being transmitted over a Transport Layer Security (TLS) version 1.3 handshake.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 20, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
1. Introduction

As discussed in SNI Encryption in TLS Through Tunneling [I-D.huitema-tls-sni-encryption], it is becoming important to protect from observers the name of the server a client is attempting access. However, Transport Layer Security (TLS) version 1.3 [I-D.ietf-tls-tls13] transmits the Server Name Indication extension (RFC6066, section 3) unencrypted.

This memo defines the TLS-Bootstrap DNS Resource Record and two TLS extensions: the Encrypted SNI Extension, the Semi-Static Key Share Extension, that when being used together provides a way to transmit the server name in an encrypted form.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Protocol Overview

In the proposed scheme, the server operator publishes its X.509 certificate [RFC5280] chain and a semi-static (EC)DH key using the TLS-Bootstrap DNS Resource Record.
When a client tries to access the server, it queries the DNS resolver for the TLS-Bootstrap DNS Resource Record in addition to the IP address of the server. The two queries can be issued simultaneously.

Once the client obtains the address of the server and also the TLS-Bootstrap DNS Resource Record, the client connects to the server and starts a TLS handshake. Instead of sending the Server Name Indication extension, the client will send the server name using the Encrypted SNI Extension, which is encrypted using a symmetric key derived from the result of the (EC)DH key exchange, the two (EC)DH keys being the one embedded in the TLS-Bootstrap DNS Resource Record and the other included in the KeyShare extension of the ClientHello message.

3. TLS-Bootstrap DNS Resource Record

The DNS Resource Record Type is used to convey the server certificate chain and (EC)DH public keys associated to a hostname.

The structure of the record type is:

```c
struct {
    KeyShareEntry key_share;
    opaque cookie<0..2^16-1>;
} SemiStaticKeyShareEntry;

struct {
    CertificateEntry certificate_list<0..2^24-1>;
    SemiStaticKeyShareEntry semi_static_shares<1..2^16-1>;
    CipherSuite cipher_suites<2..2^16-2>;
    SignatureScheme signature_algorithm;
    opaque signature<0..2^16-1>;
} TLSBootstrapRecord;
```

- `key_share` A (EC)DH public key and its type.
- `cookie` An identifier associated to the key_share. The value is transmitted from the client to the server during the TLS handshake so that the server can identify which key share as been used.
- `certificate_list` The certificate chain of the server certificate along with extensions to verify the validity of the certificate (e.g., OCSP Status extensions ([RFC6066], [RFC6961])).
- `semi_static_shares` list of key_shares that the server offers to the client
signature. The signature is a digital signature using the algorithm specified in the signature_algorithm field. The digital signature covering from the beginning of the structure to the end of the signature_algorithm field.

The set of semi-static (EC)DH keys included in the DNS Resource Record MUST be a common value between the server names that are served by the server. For example, if a server hosts three server names: example.com, example.org, example.net, the keys that are published using the DNS Resource Record will be identical for the three server names.

4. Changes to ClientHello

When a client attempts to connect to a server, it at first queries the DNS resolver to obtain the TLS-Bootstrap DNS Resource Record as well as the IP address of the server. The two DNS queries can be issued simultaneously.

Once the client obtains the IP address of the server and also the TLS-Bootstrap DNS Resource Record, it MUST verify the certificate chain and the signature of the TLS-Bootstrap DNS Resource record. After a successful verification, the client will connect to the server and start a TLS 1.3 handshake, by sending a ClientHello handshake message with the following changes.

- The "key-share" extension MUST include exactly one KeyShareEntry. The algorithm of the KeyShareEntry MUST be one among the semi-static key shares offered by the server through the TLS-Bootstrap DNS Resource Record.

- The "cipher_suite" field MUST include exactly one cipher-suite. It should be one among the cipher-suites offered by the server through the TLS-Bootstrap DNS Resource Record.

- The Server Name Indication Extension MUST NOT be used.

- The Semi-Static Key Share Extension and the EncryptedSNI Extension MUST be used.

A client can use the Cached Information Extension [RFC7924] in hope that the server will try to send the certificates that are identical to the ones that are found in the TLS-Bootstrap DNS Resource Record, and that instead of sending the certificate, the server will use the extension to just send the hash values of the certificate.
4.1. Semi-Static Key Share Extension

The extension identifies the semi-static (EC)DH key that was being
selected by the client.

struct {
    select (Handshake.msg_type) {
        case client_hello:
            opaque cookie<0..2^16-1>;
        case encrypted_extensions:
            Empty;
    }
} SemiStaticKeyShare;

A server MUST abort the handshake with a "unknown-semi-static-key"
alert if it find an unknown or an invalid cookie in the extension. A
server MUST send an empty Semi-Static Key Share Extension in the
EncryptedExtensions handshake message, when the extension appeared in
the ClientHello handshake message.

4.2. Encrypted SNI Extension

The extension contains the Server Name Indication Extension encrypted
using a shared key derived from the (EC)DH key exchange.

struct {
    ServerName server_name;
    opaque padding<0..2^8-1>;
} PlaintextEncryptedSNI;

struct {
    select (Handshake.msg_type) {
        case client_hello:
            opaque encrypted_payload<0..2^16-1>;
        case encrypted_extensions:
            Empty;
    }
} EncryptedSNI;

server_name  the raw (un-encrypted) value of the Server Name
              Indication Extension

cipher_suite The AEAD algorithm and the hash algorithm to encrypted
              the encrypted_payload

encrypted_payload Contains a PlaintextEncryptedSNI structure
              encrypted using the only cipher-suite specified by the
              "cipher_suites" field of the ClientHello message.
The key that is being used for protecting the encrypted_payload is calculated as follows by using the HKDF functions [RFC5869], whereas the semi_static_master_key being calculated by applying HKDF-Extract to the result of the (EC)DH key exchange with an empty salt.

HKDF-Expand-Label(semi_static_master_key, "encrypted-sni", ",
Hash.length)

A client MUST pad the PlaintextEncryptedSNI structure so that the length of the server name cannot be observed.

A server MUST abort the handshake with a "decode_error" alert if it sees an Encrypted SNI Extension but not the Semi-Static Key Share Extension. A server MUST abort the handshake with a "decrypt_error" alert if it fails to decrypt the encrypted SNI.

Once the server successfully decrypts the EncryptedSNI extension, it will use the value of the contained ServerName extension and continue the handshake.

A server MUST send an empty EncryptedSNI extension using the EncryptedExtension handshake message to indicate that it has seen the extension in ClientHello.

If a client observes an EncryptedExtension handshake message with a Semi-Static Key Share Extension but without a Encrypted SNI extension in response to a ClientHello message containing an Encrypted SNI extension, it MUST abort the handshake by sending a "handshake_failure" alert.

5. Things to Consider

5.1. Downgrade Attack Protection

Attackers might want to block the transmission of a TLS-Bootstrap Record to prevent a server name from being sent in an encrypted form. To mitigate such attacks, we should consider extending HTTP Strict Transport Security [RFC6797] so that the servers can enforce the client’s use of the Encrypted SNI extension.

5.2. Encrypting Other Extensions

We might want to refactor the proposed method to send an arbitrary number of extensions protected within a ClientHello message, rather than just encrypting the Server Name Indication extension. Doing so opens up the possibility of protected more types of extensions such as the Application-Layer Protocol Negotiation Extension [RFC7301].
it would be possible to use the key to invoke a 0-RTT handshake even when resumption is impossible.

5.3. Mitigating Short-term Performance Impact

Today, most DNS requests / responses are sent using UDP in cleartext. That fact has two impacts on the proposed method.

Firstly, transmitting a TLS-Bootstrap Record will require multiple roundtrips, since the record is unlikely to fit in a single packet. Secondly, the merit of hiding the server name that appears in the ClientHello somewhat diminishes by the fact that the same value might be transmitted unencrypted in the DNS query.

Considering the facts, in might make sense to define the transfer of the certificate and the signature as an optional feature, so that the only mandatory field that needs to be conveyed in the TLS-Bootstrap Record will be the (EC)DH public key and the associated cookie.

It is likely that such a reduced format will fit into a single UDP packet. Hence the performance degradation will be negligible considering the fact that the query for the TLS-Bootstrap Record can happen simultaneously with the address resolution.

The downside of the approach will be that the server name would not be protected from an active attacker. However, until DNS queries start getting transmitted over an encrypted channel, the risk of such attacks might be very low, considering the fact that server names can be found unencrypted in the DNS packets anyways.

After DNS over secure transport becomes popular, we can start embedding certificates and digital signatures in the TLS-Bootstrap Record to prevent active attacks, as well as start using the semi-static EC(DH) key for other purposes.

6. Security Considerations

By using the value of the cookie, servers MUST detect and reject the use of outdated TLS-Bootstrap DNS Resource Records. Otherwise, an attacker might be able to inject an old record to force the peers to agree on using a key-share or a cipher-suite that has turned out to be vulnerable after the record was published on the authoritative server.

The injection of malformed or outdated TLS-Bootstrap DNS Resource Record can be used as an attack vector to cause denial-of-service attacks, since misuse of such records by a client ends up in a TLS handshake failure. However, it could be argued that injection of a
wrong A record will essentially have the same effect in terms of
denial-of-service attacks. In other words, use of a DNS record to
transmit TLS handshake parameters does not make us more prone to
attacks.

7. IANA Considerations

The TLS ExtensionType Registry will be updated to contain the
codepoints for the Semi-Static Key Share Extension Type and the
Encrypted SNI Extension type.

The TLS Alert Registry will be updated to contain the "unknown-semi-
static-key" alert.

The DNS Parameters Registry will be updated to contain the codepoint
for the TLS-Bootstrap Resource Record Type.

8. Acknowledgements

Thank you to Subodh Iyengar for suggesting that the certificate chain
and the signature could be optional fields of a TLS-Bootstrap record.

9. References

9.1. Normative References

Version 1.3", draft-ietf-tls-tls13-21 (work in progress),

[RFC1035] Mockapetris, P., "Domain names - implementation and
specification", STD 13, RFC 1035, DOI 10.17487/RFC1035,

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119,
DOI 10.17487/RFC2119, March 1997,

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
Housley, R., and W. Polk, "Internet X.509 Public Key
Infrastructure Certificate and Certificate Revocation List
9.2. Informative References

[I-D.huitema-tls-sni-encryption]


Author’s Address

Kazuho Oku

Email: kazuhooku@gmail.com