Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 12, 2019

C. Huitema
Private Octopus Inc.
D. Kaiser
University of Luxembourg

March 11, 2019

Private Discovery with TLS-ESNI
draft-huitema-dnssd-tls-privacy-01

Abstract

DNS-SD (DNS Service Discovery) normally discloses information about both the devices offering services and the devices requesting services. This information includes host names, network parameters, and possibly a further description of the corresponding service instance. Especially when mobile devices engage in DNS Service Discovery over Multicast DNS at a public hotspot, a serious privacy problem arises.

We propose to solve this problem by developing a private discovery profile for UDP based transports using TLS, such as DTLS and QUIC. The profile is based on using the Encrypted SNI extension. We also define a standalone private discovery service, that can be combined with arbitrary applications in the same way as DNS-SD.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

DNS-SD [RFC6763] over mDNS [RFC6762] enables configurationless service discovery in local networks. It is very convenient for users, but it requires the public exposure of the offering and requesting identities along with information about the offered and requested services. Parts of the published information can seriously breach the user's privacy. These privacy issues and potential solutions are discussed in [KW14a] and [KW14b].

When analyzing these scenarios in [I-D.ietf-dnssd-prireq], we find that the DNS-SD messages leak identifying information such as the instance name, the host name or service properties.

In this document, we propose a discovery solution that can replace DNS-SD in simple deployment scenarios, with the following characteristics:

1. We only target discovery of peers on the same local network, using multicast. We make no attempt at compatibility with the server based solutions such as DNSSD over Unicast DNS [RFC6763].

2. We do not attempt to keep the same formats as mDNS [RFC6762].

3. We assume a solution that can be integrated in an application, without dependency on system support.

The proposed solution aligns with TLS 1.3 [RFC8446], and specifically with transports of TLS over UDP, such as DTLS [I-D.ietf-tls-dtls13] or QUIC [I-D.ietf-quic-transport]. The solution uses SNI encryption [I-D.ietf-tls-esni].

The solution assumes that entities who want to be privately discovered maintain an asymmetric discovery key pair. The public component of that discovery key pair and the service name of the entity are provisioned to authorized discovery clients. In this document, we will refer to this public component as the "Discovery Key" of the server. When needed, we will refer to the private component of the key pair as the "Discovery Private Key".

1.1. Requirements

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. Discovery Service Using TLS and ESNI

In the proposed solution, the first packet of a TLS-based UDP transport is broadcast or multicast over the local network. This packet carries the TLS 1.3 ClientHello message, including an Encrypted SNI (ESNI) extension. The ESNI is encrypted with the Discovery Key of the requested service.

Services that are ready to be discovered listen on the broadcast or multicast address and check whether the received packets contain a TLS 1.3 ClientHello Message and an ESNI extension. If the extension can be decrypted with the Private Discovery Key of the local service, they set up a connection.

This mechanism only works with TLS based protocols operating over UDP, such as DTLS or QUIC.

2.1. Discovery Key

Private discovery relies on the privacy of the server Discovery Key, which is used to encrypt the target server name carried in the ESNI extension. Clients can only discover a server if they know the server's Discovery Key. Servers receiving a properly encrypted discovery request do not know the identity of the client issuing the request, but they know that the client belongs to a group authorized to perform the discovery.

In the ESNI based specification, the server's Discovery Key plays the same role as the ESNI Encryption Key of the client facing server, but a major difference is that the Discovery Key is kept private. According to standard ESNI, the client facing server publishes an ESNI encryption key in the DNS. In contrast, the Discovery Key MUST NOT be publicly available in the DNS.

The discovery key is passed to the peer in exactly the same format as ESNI:

```
    struct {
        uint8 checksum[4];
        KeyShareEntry keys<4..2^16-1>;
        CipherSuite cipher_suites<2..2^16-2>;
        uint16 padded_length;
        uint64 not_before;
        uint64 not_after;
        Extension extensions<0..2^16-1>;
    } ESNIKeys;
```

This document does not discuss how the Discovery Key is provisioned to authorized discovery clients.

The ESNI extension design assumes that several services are available through a single client facing server. These different services have different SNI values and length. ESNI pads these SNI to a padded length specified for the client facing server, ensuring that third parties cannot infer the identity of the service from the length of the extension. In our design, requests for multiple services are sent over multicast. If different services used different padding length, third parties could infer the service identity from the ESNI length. To prevent this information leakage, services participating in private discovery MUST set the padded length to exactly 128 bytes.

2.2. ESNI Extension for Discovery

The ESNI extension is defined in [I-D.ietf-tls-esni] as:

```
      struct {
          CipherSuite suite;
          KeyShareEntry key_share;
          opaque record_digest<0..2^16-1>;
          opaque encrypted_sni<0..2^16-1>;
      } ClientEncryptedSNI;
```

In standard ESNI usage, the "record_digest" identifies the ESNI Encryption Key. Clients using private discovery MUST leave the "record_digest" empty, and encode it as a zero-length binary string. The ESNI Encryption Key will be the Discovery Key of the target server.

The KeyShareEntry is set in accordance with the ESNI specification. It is combined with the server Discovery Key to encrypt the SNI. According to the ESNI specification, the encrypted structure contains:

```
      struct {
          ServerNameList sni;
          opaque zeros[ESNIKeys.padded_length - length(sni)];
      } PaddedServerNameList;

      struct {
          uint8 nonce[16];
          PaddedServerNameList realSNI;
      } ClientESNIInner;
```

Servers that receive the extension as part of private discovery attempt to decrypt the value using their Private Discovery Key. If the decryption succeeds, and if the decrypted SNI corresponds to the expected value, the server will accept the discovery request.

2.3. Integration with DTLS

The message flows of DTLS 1.3 [I-D.ietf-tls-dtls13] all start with the client sending a TLS ClientHello message. The following figure presents a simple DTLS flow using the ESNI extension for private discovery:

```
   Client                                             Server

   ClientHello                                              +----------+
    + key_share*                                            | Flight 1 |
    + ESNI                 -------->                        +----------+

                                               ServerHello
                                              + key_share*  +----------+
                                     {EncryptedExtensions}  | Flight 2 |
                                                    {ESNI}  +----------+
                                            {Certificate*}
                           <--------            {Finished}
                                       [Application Data*]

                                                            +----------+
                                                            | Flight 3 |
    {Finished}             -------->                        +----------+
    [Application Data]

                                                            +----------+
                           <--------                 [Ack]  | Flight 4 |
                                       [Application Data*]  +----------+

    [Application Data]     <------->   [Application Data]
```

The first flight consists of a single UDP packet. In classic DTLS, this would be sent to the IP address and UDP port chosen for the application. Instead, the client using private discovery MUST sent this to the "private discovery" multicast address defined in Section 6 and to the UDP port chosen for the application.

The multicast message sent by the client will be received by many servers. The servers using private discovery MUST verify that the ESNI extension is present. If it is present, each server attempts to decrypt the ESNI extension using the local private discovery key, as specified in Section 2.2. If the verification succeeds, the server proceeds with the connection, and sends the second flight of DTLS packets to the IP address and UDP port from which it received the client's first flight.

The client receiving the second flight of messages processes them as specified in DTLS 1.3 [I-D.ietf-tls-dtls13]. The client MUST verify that the ESNI extension is present, and matches the expected value as specified in Section 2.2. If the ESNI extension is absent or does not pass verification, the entire flight MUST be ignored. If the verification succeeds, the client remembers the IP address and UDP port of the server, and uses it for the reminder of the session.

A successful ESNI exchange demonstrates to the server that the client has knowledge of the server discovery key, and to the client that the server is in possession of the corresponding private discovery key. This is meant to restrict access to a subset of the client and server population, but does not replace the need for server authentication and optional client authentication as specified in TLS 1.3.

2.4. Integration with QUIC

The QUIC Transport uses TLS to negotiate encryption keys. The use of TLS in QUIC is specified in [I-D.ietf-quic-tls], and can be summarized as follow:

1. The QUIC connection starts with the client sending an Initial message, carrying a TLS ClientHello and its extensions.

2. The server replies with its own Initial message, carrying the server hello and establishing the "handshake" keys used to protect the reminder of the TLS 1.3 exchange.

3. Server and client exchange encrypted Handshake messages to verify that the session is properly established and to negotiate the encryption keys of the application data.

4. Server and Client exchange encrypted application messages until they decide to close the connection.

All messages are sent over UDP.

When using Private Discovery, the client adds an ESNI extension to the ClientHello sent in the Initial message. The ESNI extension is formatted a specified in Section 2.2. In classic QUIC, the Initial message would be sent in a UDP packet to the IP address and UDP port of the server. Instead, the client using private discovery MUST sent this to the "private discovery" multicast address defined in Section 6 and to the UDP port chosen for the application.

The multicast message sent by the client will be received by many servers. The servers using private discovery MUST verify that the ESNI extension is present. If it is present, each server attempts to decrypt the ESNI extension using the local private discovery key, as specified in Section 2.2. If the verification succeeds, the server proceeds with the connection, and sends the next QUIC packets to the IP address and UDP port from which it received the client's first flight.

The client receiving the second flight of messages processes them as specified in DTLS 1.3 [I-D.ietf-tls-dtls13]. The client MUST verify

that the ESNI extension is present, and matches the expected value as specified in Section 2.2. If the ESNI extension is absent or does not pass verification, the entire QUIC connection MUST be ignored. If the verification succeeds, the client remembers the IP address and UDP port of the server, and uses it for the reminder of the QUIC connection.

3. Private Discovery DNS Service

The mechanisms discussed in Section 2 assume that an application based on DTLS or QUIC is modified to enable private local discovery. This does not cover all services. Further services can be supported by a two-phase process in which each application is paired with an implementation of the private discovery service.

The private discovery service is an implementation of DNS over QUIC, as specified in [I-D.huitema-quic-dnsoquic], modified to also implement the Private Discovery over Quic defined in Section 2.4. The DNS implementation is solely for the purpose of providing a service equivalent to DNS-SD.

The Private Discovery DNS Service is run by the service that wants to be discovered. The name of the discovery service is the name of the service that needs to be discovered. Clients are provisioned with the associated Discovery Key. Clients discover the Private Discovery DNS Service, and can then use it to obtain the DNS records associated with the application service: SRV, TXT, A or AAAA records.

4. Optional Server Arrival Announce

The proposed design relies on active discovery of servers by the clients. When a client arrives on a new network and wishes to establish a connection to a server, it sends a multicast message that tries to reach that server. This designs has two potential issues:

> If the server is not present when a client tries to contact it, the client may have to repeat the connection attempts over time.
>
> If multiple clients connect to the same server, each client sends a multicast message, which can translate in heavy multicast load in some configurations.

The importance of these two issues is debatable. Many applications have a peer-to-peer nature, in which peers can be either clients or servers. When two peers are not present at the same time, the first peer to arrive will fail to establish a connection, but the second peer will succeed, without having to rely on time based repetitions. Similarly, if we move from device oriented discovery to application

oriented discovery, the number of client per server is likely to be very small, so that there will be little difference between "multicast per client" and "multicast per server".

However, there may be configurations where a "server arrival announce" message would result in fast discovery and fewer multicast messages.

4.1. Server Arrival Announce Specification

The server arrival announce message is a UDP packet sent to the Discovery Multicast Address reserved in Section 6 and port %lt;TBD>.

The format of the message will be defined in a next draft version, meeting the following requirements:

Multiplexing: should be easily demuxed from DTLS or QUIC traffic.

ESNI: should contain an ESNI extension, secured with the server's discovery key.

Replay: should contain a timestamp and the global scope IPv6 address of the server.

5. Security Considerations

The use of TLS 1.3 and the ESNI extension provides robust defenses against attacks. In particular, Private Discovery benefits from the defenses against dictionary attacks and replay attacks built in the ESNI design. Protections against a residual DOS attack is discussed in Section 5.1.

The privacy of the discovery relies on keeping the discovery key of the service secret. The consequences and partial mitigation of leaking the discovery key are discussed in Section 5.2.

Compromising of the server's private discovery key will allow an attacker to break the privacy of the discovery, as discussed in Section 5.3.

5.1. Denial of Service by Spoofed Response

Attackers may try to disrupt a private discovery handshake by sending a spoofed Server Hello (DTLS) or a spoofed Server Initial packet (QUIC). The client will reject these attempts after noticing that the encrypted extensions do not include a proper ESNI extension, containing the expected copy of the ESNI nonce.

Attackers will not succeed spoofing the server, but they could succeed in denying the connection if the fake response arrives before the response from the actual server, and if the implementation just gave up the attempt after failing to validate the first response that it received.

To defend against this attack, implementations SHOULD keep listening for responses and attempting validation until they receive at least one valid response from the expected server.

5.2. Discovery Key Compromise

The Discovery Key is known by all the authorized clients. If one of these clients is compromised, the privacy of the server will be compromised: attackers will be able to spoof the authorized client and discover whether the server is present on a local network. However, the leak can only be exploited in an active attack: the attacker must actively set up a connection with the target server.

The attack is mitigated when the server migrates to a different discovery key and restricts the availability of that key to non-compromised clients.

Exploiting a compromised discovery key normally requires that the attacker is present on the same link as the target. Attackers might try to work around that limitation by sending unicast packet targeted at plausible server locations. Servers participating in private discovery MUST NOT accept discovery requests arriving from off-link sources.

5.3. Private Discovery Key Compromise

The private component of the asymmetric key pair used for discovery MUST be kept secret by the server. If it is compromised, attackers can process discovery requests and verify that they can be decrypted with the server's private discovery key. They could also process logs of old discovery attempts.

The design provides two mitigations against the consequences of this failure:

- The discovery requests do not uniquely identify the client, and the attacker will only know that an attempt came from one of the authorized clients.

- The actual communications are protected by TLS, and inherit the forward secrecy properties of TLS 1.3.

[[TODO: consider specifying a way to rotate the discovery key, so as to mitigate the lack of forward secrecy. Maybe add that to ESNI.]]

5.4. Tracking by Replay

Suppose that an attacker has identified a client, and is capable of recording the multicast messages from that client. The attacker can then replay the message, triggering a response from the target server if present on the network.

The attacker will not be able to actually establish a connection with the server -- the TSL and ESNI designs protect against that. But it will be able to find out that the same server that responded to the client before responds now, which is a way to track the server.

The attacker can mount two variations of the attack: replay over time, and replay at different locations. In the current design, the main protection against that attack is the implementation of a "discovery window", so that servers only listen to multicast requests when they are "ready to be discovered".

[[TODO: consider adding a time stamp in an extension to ESNI.]]

[[TODO: consider adding the IPv6 address of the sender in an extension to ESNI.]]

[[TODO: special consideration for server arrival announce.]]

6. IANA Considerations

IANA is required to allocate the IPv6 multicast address FF02::<TBD> for use as "Private Discovery Multicast Address" described in this document.

6.1. Experimental use

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

Early experiments MAY use the address FF02::60DB:F6C5. This address is marked in the IANA registry as unassigned.

7. Acknowledgments

[[TODO]]

8. References

8.1. Normative References

[I-D.huitema-quic-dnsoquic]
Huitema, C., Shore, M., Mankin, A., Dickinson, S., and J. Iyengar, "Specification of DNS over Dedicated QUIC Connections", draft-huitema-quic-dnsoquic-06 (work in progress), March 2019.

[I-D.ietf-quic-tls]
Thomson, M. and S. Turner, "Using TLS to Secure QUIC", draft-ietf-quic-tls-18 (work in progress), January 2019.

[I-D.ietf-quic-transport]
Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-18 (work in progress), January 2019.

[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-dtls13-30 (work in progress), November 2018.

[I-D.ietf-tls-esni]
Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "Encrypted Server Name Indication for TLS 1.3", draft-ietf-tls-esni-02 (work in progress), October 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[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>.

8.2. Informative References

[I-D.ietf-dnssd-prireq]
Huitema, C., "DNS-SD Privacy and Security Requirements", draft-ietf-dnssd-prireq-00 (work in progress), September 2018.

[KW14a] Kaiser, D. and M. Waldvogel, "Adding Privacy to Multicast DNS Service Discovery", DOI 10.1109/TrustCom.2014.107, 2014, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7011331>.

[KW14b] Kaiser, D. and M. Waldvogel, "Efficient Privacy Preserving Multicast DNS Service Discovery", DOI 10.1109/HPCC.2014.141, 2014, <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7056899>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

Authors' Addresses

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

Email: huitema@huitema.net
URI: http://privateoctopus.com/

Daniel Kaiser
University of Luxembourg
6, avenue de la Fonte
Esch-sur-Alzette 4364
Luxembourg

Email: daniel.kaiser@uni.lu
URI: https://secan-lab.uni.lu/