TLS
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2013

P. Wouters, Ed.
Red Hat
H. Tschofenig, Ed.
Nokia Siemens Networks
J. Gilmore

S. Weiler
SPARTA, Inc.
T. Kivinen
AuthenTec
October 22, 2012

Out-of-Band Public Key Validation for Transport Layer Security (TLS)
draft-ietf-tls-oob-pubkey-06.txt

Abstract

This document specifies a new certificate type for exchanging raw public keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) for use with out-of-band public key validation. Currently, TLS authentication can only occur via X.509-based Public Key Infrastructure (PKI) or OpenPGP certificates. By specifying a minimum resource for raw public key exchange, implementations can use alternative public key validation methods.

One such alternative public key valiation method is offered by the DNS-Based Authentication of Named Entities (DANE) together with DNS Security. Another alternative is to utilize pre-configured keys, as is the case with sensors and other embedded devices. The usage of raw public keys, instead of X.509-based certificates, leads to a smaller code footprint.

This document introduces the support for raw public keys in TLS.

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 April 25, 2013.

Table of Contents

1. Introduction

Traditionally, TLS server public keys are obtained in PKIX containers in-band using the TLS handshake and validated using trust anchors based on a [PKIX] certification authority (CA). This method can add a complicated trust relationship that is difficult to validate. Examples of such complexity can be seen in [Defeating-SSL].

Alternative methods are available that allow a TLS client to obtain the TLS server public key:

- The TLS server public key is obtained from a DNSSEC secured resource records using DANE [RFC6698].

- The TLS server public key is obtained from a [PKIX] certificate chain from an Lightweight Directory Access Protocol (LDAP) [LDAP] server.

- The TLS client and server public key is provisioned into the operating system firmware image, and updated via software updates.

Some smart objects use the UDP-based Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] to interact with a Web server to upload sensor data at a regular intervals, such as temperature readings. CoAP [I-D.ietf-core-coap] can utilize DTLS for securing the client-to-server communication. As part of the manufacturing process, the embeded device may be configured with the address and the public key of a dedicated CoAP server, as well as a public key for the client itself. The usage of X.509-based PKIX certificates [PKIX] may not

suit all smart object deployments and would therefore be an unneccesarry burden.

The Transport Layer Security (TLS) Protocol Version 1.2 [RFC5246] provides a framework for extensions to TLS as well as guidelines for designing such extensions. This document defines an extension to indicate the support for raw public keys.

2. Terminology

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 RFC 2119 [RFC2119].

3. New TLS Extension

This section describes the changes to the TLS handshake message

contents when raw public key certificates are to be used. Figure 3 illustrates the exchange of messages as described in the sub-sections below. The client and the server exchange the newly defined certificate_type extension to indicate their ability and desire to exchange raw public keys. These raw public keys, in the form of a SubjectPublicKeyInfo structure, are then carried inside the certificate payload. The SubjectPublicKeyInfo structure is defined in Section 4.1 of RFC 5280. Note that the SubjectPublicKeyInfo block does not only contain the raw keys, such as the public exponent and the modulus of an RSA public key, but also an algorithm identifier. The structure, as shown in Figure 1, is encoded in an ASN.1 format and therefore contains length information as well.

```
SubjectPublicKeyInfo  ::=  SEQUENCE  {
     algorithm            AlgorithmIdentifier,
     subjectPublicKey     BIT STRING  }
```

Figure 1: SubjectPublicKeyInfo ASN.1 Structure.

The algorithm identifiers are Object Identifiers (OIDs). RFC 3279 [RFC3279], for example, defines the following OIDs shown in Figure 2.

Key Type	Document	OID
RSA	Section 2.3.1 of RFC 3279	1.2.840.113549.1.1
Digital Signature Algorithm (DSS)	Section 2.3.2 of RFC 3279	1.2.840.10040.4.1
Elliptic Curve Digital Signature Algorithm (ECDSA)	Section 2.3.5 of RFC 3279	1.2.840.10045.2.1

Figure 2: Example Algorithm Identifiers.


```
    client_hello,
    certificate_type          ->

                              <-  server_hello,
                                  certificate_type,
                                  certificate,
                                  server_key_exchange,
                                  certificate_request,
                                  server_hello_done
    certificate,
    client_key_exchange,
    certificate_verify,
    change_cipher_spec,
    finished                  ->
```

```
                              <- change_cipher_spec,
                                 finished

Application Data        <------->     Application Data
```

Figure 3: Basic Raw Public Key TLS Exchange.

The "certificate_type" TLS extension carries a list of supported certificate types the client can send and receive, sorted by client preference. Two values are defined for each certificate types to differentiate whether a client or a server is able to process a certificate of a specific type or can also send it. This extension MUST be omitted if the client only supports X.509 certificates. The "extension_data" field of this extension contains a CertTypeExtension structure.

Note that the CertTypeExtension structure is being used both by the client and the server, even though the structure is only specified once in this document.

The structure of the CertTypeExtension is defined as follows:


```
enum { client, server } ClientOrServerExtension;

enum { X.509-Accept (0),
       X.509-Offer (1),
       RawPublicKey-Accept (2),
       RawPublicKey-Offer (3),
       (255)
      } CertificateType;
```

```
struct {
   select(ClientOrServerExtension)
       case client:
         CertificateType certificate_types<1..2^8-1>;
       case server:
         CertificateType certificate_type;
   }
} CertTypeExtension;
```

Figure 4: CertTypeExtension Structure.

The '-Offer' postfix indicates that a TLS entity is able to send the indicated certificate type to the other communication partner. The '-Accept' postfix indicates that a TLS entity is able to receive the indicated certificate type.

No new cipher suites are required to use raw public keys. All existing cipher suites that support a key exchange method compatible with the defined extension can be used.

4. TLS Handshake Extension

4.1. Client Hello

In order to indicate the support of out-of-band raw public keys, clients MUST include an extension of type "certificate_type" to the extended client hello message. The "certificate_type" TLS extension is assigned the value of [TBD] from the TLS ExtensionType registry. This value is used as the extension number for the extensions in both the client hello message and the server hello message. The hello extension mechanism is described in TLS 1.2 [RFC5246].

4.2. Server Hello

If the server receives a client hello that contains the "certificate_type" extension and chooses a cipher suite then two outcomes are possible. The server MUST either select a certificate

type from the CertificateType field in the extended client hello or

terminate the session with a fatal alert of type "unsupported_certificate".

The certificate type selected by the server is encoded in a CertTypeExtension structure, which is included in the extended server hello message using an extension of type "certificate_type". Servers that only support X.509 certificates MAY omit including the "certificate_type" extension in the extended server hello.

If the client supports the receiption of raw public keys and the server is able to provide such a raw public key then the TLS server MUST place the SubjectPublicKeyInfo structure into the Certificate payload. The public key MUST match the selected key exchange algorithm.

4.3. Certificate Request

The semantics of this message remain the same as in the TLS specification.

4.4. Other Handshake Messages

All the other handshake messages are identical to the TLS specification.

4.5. Client authentication

Client authentication by the TLS server is supported only through authentication of the received client SubjectPublicKeyInfo via an out-of-band method

5. Examples

Figure 5, Figure 6, and Figure 7 illustrate example exchanges.

The first example shows an exchange where the TLS client indicates its ability to receive raw public keys. This client is quite restricted since it is unable to process other certificate types sent by the server. It also does not have credentials it could send. The 'certificate_type' extension indicates this in [1]. When the TLS server receives the client hello it processes the certificate_type extension. Since it also has a raw public key it indicates in [2] that it had choosen to place the SubjectPublicKeyInfo structure into the Certificate payload [3]. The client uses this raw public key in the TLS handshake and an out-of-band technique, such as DANE, to verify its validity.

```
client_hello,
certificate_type=(RawPublicKey-Accept) -> // [1]

                          <-  server_hello,
                              certificate_type=(RawPublicKey-Offer), // [2]
                              certificate, // [3]
                              server_key_exchange,
                              server_hello_done

client_key_exchange,
change_cipher_spec,
finished                   ->

                          <- change_cipher_spec,
                             finished

Application Data        <------->     Application Data
```

Figure 5: Example with Raw Public Key provided by the TLS Server

In our second example the TLS client as well as the TLS server use raw public keys. This is a use case envisioned for smart object networking. The TLS client in this case is an embedded device that is configured with a raw public key for use with TLS and is also able to process raw public keys sent by the server. Therefore, it indicates these capabilities in the 'certificate_type' extension in [1]. As in the previously shown example the server fulfills the client's request, indicates this via the 'RawPublicKey-Offer'in the certificate_type payload, and provides a raw public key into the Certificate payload back to the client (see [3]). The TLS server, however, demands client authentication and therefore a certificate_request is added [4]. The certificate_type payload in [2] indicates that the TLS server accepts raw public keys. The TLS client, who has a raw public key pre-provisioned, returns it in the Certificate payload [5] to the server.

```
   client_hello,
   certificate_type=(RawPublicKey-Offer, RawPublicKey-Accept) -> // [1]

                            <-  server_hello,
                                certificate_type=(RawPublicKey-Offer,
                                            RawPublicKey-Accept) // [2]
                                certificate, // [3]
                                certificate_request, // [4]
                                server_key_exchange,
                                server_hello_done

   certificate, // [5]
   client_key_exchange,
   change_cipher_spec,
   finished                    ->

                            <- change_cipher_spec,
                               finished

   Application Data         <------->     Application Data
```

Figure 6: Example with Raw Public Key provided by the TLS Server and the Client

In our last example we illustrate a combination of raw public key and X.509 usage. The client uses a raw public key for client authentication but the server provides an X.509 certificate. This exchange starts with the client indicating its ability to process X.509 certificates provided by the server, and the ability to send raw public keys. The server provides the X.509 certificate in [3] with the indication present in [2]. For client authentication, however, the server indicates in [2] that it is able to support raw public keys and requests a certificate from the client in [4]. The TLS client provides a raw public key in [5] after receiving and processing the TLS server hello message.

```
   client_hello,
   certificate_type=(X.509-Accept, RawPublicKey-Offer) -> // [1]

                             <-  server_hello,
                                 certificate_type=(X.509-Offer,
                                      RawPublicKey-Accept), // [2]
                                 certificate, // [3]
                                 certificate_request, // [4]
                                 server_key_exchange,
                                 server_hello_done
   certificate, // [5]
   client_key_exchange,
   change_cipher_spec,
   finished                   ->

                             <- change_cipher_spec,
                                finished

   Application Data        <------->     Application Data
```

Figure 7: Hybrid Certificate Example

6. Security Considerations

The transmission of raw public keys, as described in this document, provides benefits by lowering the over-the-air transmission overhead since raw public keys are quite naturally smaller than an entire certificate. There are also advantages from a codesize point of view for parsing and processing these keys. The crytographic procedures for assocating the public key with the possession of a private key

also follows standard procedures.

The main security challenge is, however, how to associate the public key with a specific entity. This information will be needed to make authorization decisions. Without a secure binding, man-in-the-middle attacks may be the consequence. This document assumes that such binding can be made out-of-band and we list a few examples in Section 1. DANE [RFC6698] offers one such approach. If public keys are obtained using DANE, these public keys are authenticated via DNSSEC. Pre-configured keys is another out of band method for authenticating raw public keys. While pre-configured keys are not suitable for a generic Web-based e-commerce environment such keys are a reasonable approach for many smart object deployments where there is a close relationship between the software running on the device and the server-side communication endpoint. Regardless of the chosen mechanism for out-of-band public key validation an assessment of the

most suitable approach has to be made prior to the start of a deployment to ensure the security of the system.

7. IANA Considerations

This document defines a new TLS extension, "certificate_type", assigned a value of [TBD] from the TLS ExtensionType registry defined in [RFC5246]. This value is used as the extension number for the extensions in both the client hello message and the server hello message. The new extension type is used for certificate type negotiation.

The "certificate_type" extension contains an 8-bit CertificateType field, for which a new registry, named "TLS Certificate Types", is established in this document, to be maintained by IANA. The registry is segmented in the following way:

1. The values 0 - 3 are defined in Figure 4.

2. Values from 3 through 223 decimal inclusive are assigned via IETF Consensus [RFC5226].

3. Values from 224 decimal through 255 decimal inclusive are reserved for Private Use [RFC5226].

8. Acknowledgements

The feedback from the TLS working group meeting at IETF#81 has substantially shaped the document and we would like to thank the meeting participants for their input. The support for hashes of public keys has been moved to [I-D.ietf-tls-cached-info] after the discussions at the IETF#82 meeting and the feedback from Eric Rescorla.

We would like to thank the following persons for their review comments: Martin Rex, Bill Frantz, Zach Shelby, Carsten Bormann, Cullen Jennings, Rene Struik, Alper Yegin, Jim Schaad, Paul Hoffman, Robert Cragie, Nikos Mavrogiannopoulos, Phil Hunt, John Bradley, Klaus Hartke, Stefan Jucker, and James Manger.

9. References

9.1. Normative References

[PKIX] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

9.2. Informative References

[Defeating-SSL] Marlinspike, M., "New Tricks for Defeating SSL in Practice", February 2009, <http://www.blackhat.com/

presentations/bh-dc-09/Marlinspike/
BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>.

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-12 (work in progress), October 2012.

[I-D.ietf-tls-cached-info]
Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-13 (work in progress), September 2012.

[LDAP] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

Authors' Addresses

Paul Wouters (editor)
Red Hat

Email: paul@nohats.ca

Hannes Tschofenig (editor)
Nokia Siemens Networks

Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

John Gilmore
PO Box 170608
San Francisco, California 94117
USA

Phone: +1 415 221 6524
Email: gnu@toad.com
URI: https://www.toad.com/

Samuel Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com

Tero Kivinen
AuthenTec
Eerikinkatu 28
HELSINKI FI-00180
FI

Email: kivinen@iki.fi