TLS
Internet-Draft
Intended status: Standards Track
Expires: January 17, 2013

P. Wouters
Red Hat
J. Gilmore
S. Weiler
SPARTA, Inc.
T. Kivinen
AuthenTec
H. Tschofenig
Nokia Siemens Networks

July 16, 2012

Out-of-Band Public Key Validation for Transport Layer Security
draft-ietf-tls-oob-pubkey-04.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.

material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

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 [I-D.ietf-dane-protocol].
- 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] does 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 Extensions

In order to indicate the support for multiple certificate types two

new extensions are defined by this specification with the following semantic:

cert-send: The certificate payload in this message contains a certificate of the type indicated by this extension.

cert-receive: By including this extension an entity indicates that it is able to recieve and process the indicated certificate types. This list is sorted by preference.

```
enum { X.509(0), RawPublicKey(1), (255) } CertType;

CertType cert-receive <1..2^8-1>;

CertType cert-send;
```

Figure 1: New TLS Extension Structures

No new cipher suites are required for use with raw public keys. All existing cipher suites that support a key exchange method compatible with the key in the certificate can be used in combination with raw public key certificate types.

4. TLS Handshake Extension

This section describes the semantic of the 'cert-send' and the 'cert-receive' extensions for the different handshake messages.

4.1. Client Hello

To allow a TLS client to indicate that it is able to receive a certificate of a specific type it MAY include the 'cert-receive' extension in the client hello message. To indicate the ability to process a raw public key by the server the TLS client MUST include the 'cert-receive' with the value one (1) (indicating "RawPublicKey") in the list of supported certificate types. If a TLS client only supports X.509 certificates it MAY include this extension to indicate support for it.

Future documents may define additional certificate types that require addition values to be registered.

Note: 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.

4.2. Server Hello

If the server receives a client hello that contains the 'cert-receive' extension then two outcomes are possible. The server MUST either select a certificate type from client-provided list or terminate the session with a fatal alert of type "unsupported_certificate". In the former case the procedure in Section 4.4 MUST be followed.

4.3. Certificate Request

The Certificate Request payload sent by the TLS server to the TLS client MUST be accompanied by a 'cert-receive' extension, which indicates to the TLS client the certificate type the server supports.

4.4. Certificate Payload

Certificate payloads MUST be accompanied by a 'cert-send' extension, which indicates the certificate format found in the Certificate payload itself.

The list of supported certificate types to choose from MUST have been obtained via the 'cert-receive' extension. This ensures that a Certificate payload only contains a certificate type that is also supported by the recipient.

When the 'RawPublicKey' certificate type is selected then the SubjectPublicKeyInfo structure MUST be placed into the Certificate payload. The type of the asymmetric key MUST match the selected key exchange algorithm.

4.5. Other TLS Messages

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

5. Examples

Figure 2, Figure 3, and Figure 4 illustrate example message exchanges.

The first example shows an exchange where the TLS client indicates its ability to process two certificate types, namely raw public keys and X.509 certificates via the 'cert-receive' extension (see [1]). When the TLS server receives the client hello it processes the cert-

receive extension and 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 (see [3]). The client uses this raw public key in the TLS handshake and an out-of-band technique, such as DANE, to verify its validatity.

```
client_hello,
cert-receive=(RawPublicKey, X.509) -> // [1]

                          <-  server_hello,
                              cert-send=RawPublicKey, // [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 2: Example with Raw Public Key provided by the TLS Server

In our second example the TLS client and 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 only supports raw public keys and therefore it indicates this capability via the 'cert-receive' extension in [1]. As in the previously shown example the server fulfills the client's request and provides a raw public key into the Certificate payload back to the client (see [2] and [3]). The TLS server, however, demands client authentication and for this reason a Certificate_Request payload is added [4], which comes with an indication of the supported certificate types by the server, see [5]. The TLS client, who has a raw public key pre-provisioned, returns it in the Certificate payload [7] to the server with the indication about its content [6].

Figure 3: 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. The server provides the X.509 certificate using that format in [3] with the indication present in [2]. For client authentication, however, the server indicates in [5] that it is able to support raw public keys as well as X.509 certificates. The TLS client provides a raw public key in [7] and the indication in [6].

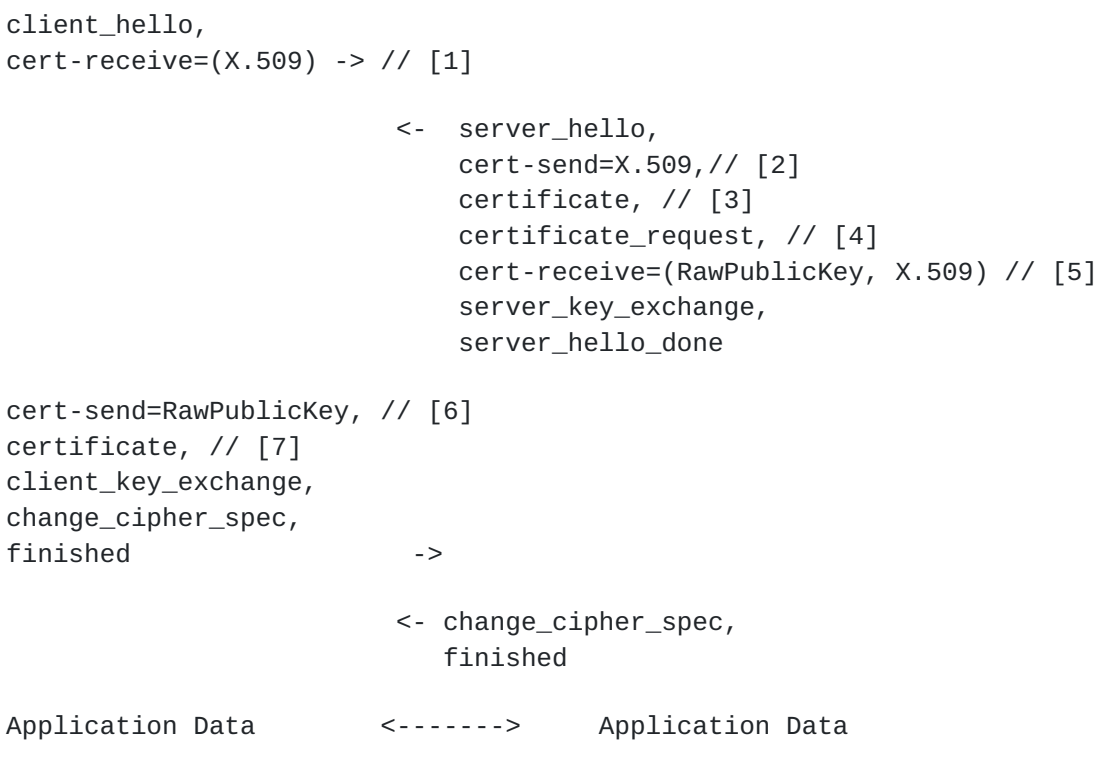

Figure 4: Hybrid Certificate Example

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 [I-D.ietf-dane-protocol] 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 two new TLS extension, 'cert-send' and 'cert-receive', and their values need to be added to the TLS ExtensionType registry created by RFC 5246 [RFC5246].

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

1. The value (0) is defined in this document.

2. Values from 2 through 223 decimal inclusive are assigned using the 'Specification Required' policy defined in RFC 5226 [RFC5226].

3. Values from 224 decimal through 255 decimal inclusive are reserved for 'Private Use', see [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, 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-10 (work in progress), June 2012.

[I-D.ietf-dane-protocol] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", draft-ietf-dane-protocol-23 (work in progress), June 2012.

[I-D.ietf-tls-cached-info] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-11 (work in progress), December 2011.

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

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

[RFC6091] Mavrogiannopoulos, N. and D. Gillmor, "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication", RFC 6091, February 2011.

Authors' Addresses

Paul Wouters
Red Hat

Email: paul@nohats.ca

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

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

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