XMPP Working Group M. Miller
Internet-Draft Cisco Systems, Inc.
Intended status: Standards Track P. Saint-Andre
Expires: July 30, 2015 &yet
January 26, 2015

PKIX over Secure HTTP (POSH)
draft-ietf-xmpp-posh-03

Abstract

Experience has shown that it is extremely difficult to deploy proper PKIX certificates for TLS in multi-tenanted environments. As a result, domains hosted in such environments often deploy applications using certificates that identify the hosting service, not the hosted domain. Such deployments force end users and peer services to accept a certificate with an improper identifier, resulting in obvious security implications. This document defines two methods that make it easier to deploy certificates for proper server identity checking in non-HTTP application protocols. While these methods developed for use in the Extensible Messaging and Presence Protocol (XMPP) as a Domain Name Association (DNA) prooftype, they might also be usable in other non-HTTP application protocols.

Status of This Memo

This Internet-Draft will expire on July 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

We begin with a thought experiment.

Imagine that you work on the operations team of a hosting company that provides the "foo" service (or email or instant messaging or social networking service) for ten thousand different customer organizations. Each customer wants their service to be identified by the customer's domain name (e.g., bar.example.com), not the hosting company's domain name (e.g., hosting.example.net).

In order to properly secure each customer's "foo" service via Transport Layer Security (TLS) [RFC5246], you need to obtain PKIX certificates [RFC5280] containing identifiers such as bar.example.com, as explained in the "CertID" specification [RFC6125]. Unfortunately, you can't obtain such certificates because:

- Certification authorities won't issue such certificates to you because you work for the hosting company, not the customer organization.

- Customers won't obtain such certificates and then give them (plus the associated private keys) to you because their legal department is worried about liability.

- You don't want to install such certificates (plus the associated private keys) on your servers anyway because your legal department is worried about liability, too.

- Even if your legal department is happy, this still means managing one certificate for each customer across the infrastructure, contributing to a large administrative load.

Given your inability to deploy public keys / certificates containing the right identifiers, your back-up approach has always been to use a certificate containing hosting.example.net as the identifier. However, more and more customers and end users are complaining about warning messages in user agents and the inherent security issues involved with taking a "leap of faith" to accept the identity mismatch between what [RFC6125] calls the Source Domain (bar.example.com) and the Delegated Domain (hosting.example.net).

This situation is both insecure and unsustainable. You have investigated the possibility of using DNS Security [RFC4033] and DNS-Based Authentication of Named Entities (DANE) [RFC6698] to solve the problem. However, your customers and your operations team have told you that it will be several years before they will be able to deploy DNSSEC and DANE for all of your customers (because of tooling updates, slow deployment of DNSSEC at some top-level domains, etc.). The product managers in your company are pushing you to find a method that can be deployed more quickly to overcome the lack of proper server identity checking for your hosted customers.

One possible approach that your team has investigated is to ask each customer to provide the public key / certificate for the "foo" service at a special HTTPS URL on their website ("https://bar.example.com/.well-known/posh.foo.json" is one possibility). This could be a public key that you generate for the customer, but because the customer hosts it via HTTPS, any user agent can find that public key and check it against the public key you provide during TLS negotiation for the "foo" service (as one added benefit, the customer never needs to hand you a private key). Alternatively, the customer can redirect requests for that special HTTPS URL to an HTTPS URL at your own website, thus making it explicit that they have delegated the "foo" service to you.

The approach sketched out above, called POSH ("PKIX Over Secure HTTP"), is explained in the remainder of this document. While this approach was developed for use in the Extensible Messaging and Presence Protocol (XMPP) as a prooftype for Domain Name Associations (DNA) [I-D.ietf-xmpp-dna], it can be applied to any non-HTTP application protocol.

2. Terminology

This document inherits security terminology from [RFC5280]. The terms "Source Domain", "Delegated Domain", "Derived Domain", and "Reference Identifier" are used as defined in the "CertID" specification [RFC6125].

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

3. Obtaining Verification Materials

Server identity checking (see [RFC6125]) involves three different aspects:

1. A proof of the POSH server's identity (in PKIX, this takes the form of a PKIX end-entity certificate [RFC5280]).

2. Rules for checking the certificate (which vary by application protocol, although [RFC6125] attempts to harmonize those rules).

3. The materials that a POSH client uses to verify the POSH server's identity or check the POSH server's proof (in PKIX, this takes the form of chaining the end-entity certificate back to a trusted root and performing all validity checks as described in [RFC5280], [RFC6125], and the relevant application protocol specification).

When POSH is used, the first two aspects remain the same: the POSH server proves it identity by presenting a PKIX certificate [RFC5280] and the certificate is checked according to the rules defined in the appropriate application protocol specification (such as [RFC6120] for XMPP). However, the POSH client obtains the materials it will use to verify the server's proof by retrieving a JSON document [RFC7159] containing hashes of the PKIX certificate over HTTPS ([RFC7230] and [RFC2818]) from a well-known URI [RFC5785] at the Source Domain. (This means that the POSH client needs to verify the certificate of the HTTPS service at the Source Domain in order to securely "bootstrap" into the use of POSH; specifically, the rules of

[RFC2818] apply to this "bootstrapping" step to provide a secure basis for all subsequent POSH processing.)

The process for retrieving a PKIX certificate over secure HTTP is as follows.

1. The POSH client performs an HTTPS GET request at the Source Domain to the path "/.well-known/posh.{servicedesc}.json". The value of "{servicedesc}" is application-specific; see Section 8 of this document for more details. For example, if the application protocol is some hypothetical "foo" service, then "{servicedesc}" could be "foo"; thus if an application client were to use POSH to verify an application server for the Source Domain "bar.example.com", the HTTPS GET request would be as follows:

   ```
   GET /.well-known/posh.foo.json HTTP/1.1
   Host: bar.example.com
   ```

2. The Source Domain HTTPS server responds in one of three ways:

   * If it possesses PKIX certificate information for the requested path, it responds as detailed in Section 3.1.

   * If it has a reference to where the PKIX certificate information can be obtained, it responds as detailed in Section 3.2.

   * If it does not have any PKIX certificate information or a reference to such information for the requested path, it responds with an HTTP client error status code (e.g., 404).

3.1. Source Domain Possesses PKIX Certificate Information

If the Source Domain HTTPS server possesses the certificate information, it responds to the HTTPS GET request with a success status code and the message body set to a JSON document [RFC7159]; the document is a JSON object which MUST have the following:

- A "fingerprints" field whose value is a JSON array of fingerprint descriptors.

- An "expires" field whose value is a JSON number specifying the number of seconds after which the POSH client ought to consider the key information to be stale (further explained under Section 6).

Each included fingerprint descriptor is a JSON object, where each member name is the textual name of a hash function (as listed in [HASH-NAMES]) and its associated value is the base 64 encoded fingerprint hash generated using the named hash function (where the encoding adheres to the definition in Section 4 of [RFC4648] and where the padding bits are set to zero). Each fingerprint descriptor MUST possess at least one named hash function.

The fingerprint hash for a given hash algorithm is generated by performing the named hash function over the DER encoding of the PKIX X.509 certifiate; for example, a "sha-1" fingerprint is generated by performing the SHA-1 hash function over the DER encoding of the PKIX certificate.

The following example illustrates the usage described above.

Example Content Response

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 134

{
  "fingerprints": [
    {
      "sha-1":"UpjRI/A3afKE8/AIeTZ5o1dECTY=",
      "sha-256":"4/mggdlVx8A3pvHAWW5sD+qJyMtUHgiRuPjVC48N0XQ="
    }
  ],
  "expires": 604800
}
```

The "expires" value is a hint regarding the expiration of the keying materials. It MUST be a non-negative integer. If no "expires" field is included or its value is equal to 0, a POSH client SHOULD consider these verification materials invalid. See Section 6 for how to reconcile this "expires" field with the reference's "expires" field.

3.2. Source Domain References PKIX Certificate

If the Source Domain HTTPS server has a reference to the certificate information, it responds to the HTTPS GET request with a success status code and message body set to a JSON document. The document is a JSON object which MUST contain the following:

- A "url" field whose value is a JSON string specifying the HTTPS URL where POSH clients can obtain the actual certificate information.

- An "expires" field whose value is a JSON number specifying the number of seconds after which the POSH client ought to consider the delegation to be stale (further explained under Section 6).

Example Reference Response

```
HTTP/1.1 200 Ok
Content-Type: application/json
Content-Length: 79

{
  "url":"https://hosting.example.net/.well-known/posh.foo.json",
  "expires":86400
}
```

The client performs an HTTPS GET request for the URL specified in the "url" field value. The HTTPS server for the URL to which the client has been redirected responds to the request with a JSON document containing fingerprints as described in Section 3.1. The content retrieved from the "url" location MUST NOT itself be a reference (i.e., containing a "url" field instead of a "fingerprints" field), in order to prevent circular delegations.

> Note: The JSON document returned by the Source Domain HTTPS server MUST contain either a reference or a fingerprints document, but MUST NOT contain both.
>
> Note: See Section 9 for discussion about HTTPS redirects.

The "expires" value is a hint regarding the expiration of the Source Domain's delegation of service to the Delegated Domain. It MUST be a non-negative integer. If no "expires" field is included or its value is equal to 0, a POSH client SHOULD consider the delegation invalid. See Section 6 for guidelines about reconciling this "expires" field with the "expires" field of the fingerprints document.

3.3. Performing Verification

The POSH client compares the PKIX information obtained from the POSH server against each fingerprint descriptor object in the POSH results, until a match is found using the hash functions that the client suports, or until the collection of POSH verification materials is exhausted. If none of the fingerprint descriptor objects match the POSH server PKIX information, the POSH client

SHOULD reject the connection (however, the POSH client might still accept the connection if other verification schemes are successful).

4. Secure Delegation

The delegation from the Source Domain to the Delegated Domain can be considered secure if the credentials offered by the POSH server match the verification materials possessed by the client, regardless of how those materials are obtained.

5. Order of Operations

In order for the POSH client to perform verification of Reference Identifiers without potentially compromising data, POSH processes MUST be complete before any application-layer data is exchanged for the Source Domain. In cases where the POSH client initiates an application-layer connection, the client SHOULD perform all POSH retrievals before initiating a connection (naturally this is not possible in cases where the POSH client receives an application-layer connection). For application protocols that use DNS SRV (including queries for TLSA records in concert with SRV records as described in [I-D.ietf-dane-srv]), the POSH processes ideally ought to be done in parallel with resolving the SRV records and the addresses of any targets, similar to the "happy eyeballs" approach for IPv4 and IPv6 [RFC6555].

The following diagram illustrates the possession flow:

```
   Client                    Domain                    Server
   ------                    ------                    ------
     |                          |                          |
     |      Request POSH        |                          |
     |------------------------->|                          |
     |                          |                          |
     | Return POSH fingerprints |                          |
     |<-------------------------|                          |
     |                          |                          |
     |                  Service TLS Handshake              |
     |<===================================================>|
     |                          |                          |
     |                     Service Data                    |
     |<===================================================>|
     |                          |                          |
```

Figure 1: Order of Events for Possession Flow

While the following diagram illustrates the reference flow:

```
   Client                    Domain                    Server
   ------                    ------                    ------
     |                          |                          |
     |      Request POSH        |                          |
     |------------------------->|                          |
     |                          |                          |
     |     Return POSH url      |                          |
     |<-------------------------|                          |
     |                          |                          |
     |                      Request POSH                   |
     |---------------------------------------------------->|
     |                          |                          |
     |                Return POSH fingerprints             |
     |<----------------------------------------------------|
     |                          |                          |
     |                 Service TLS Handshake               |
     |<===================================================>|
     |                          |                          |
     |                     Service Data                    |
     |<===================================================>|
     |                          |                          |
```

Figure 2: Order of Events for Reference Flow

6. Caching Results

The POSH client MUST NOT cache results (reference or fingerprints) indefinitely. If the Source Domain returns a reference, the POSH client MUST use the lower of the two "expires" values when determining how long to cache results (i.e., if the reference "expires" value is lower than the fingerprints "expires" value, honor the reference "expires" value). Once the POSH client considers the results stale, it needs to perform the entire POSH process again starting with the HTTPS GET request to the Source Domain. The POSH client MAY use a lower value than any provided in the "expires" field(s), or not cache results at all.

The POSH client SHOULD NOT rely on HTTP caching mechanisms, instead using the expiration hints provided in the POSH reference document or fingerprints documents. To that end, the HTTPS servers for Source Domains and Derived Domains SHOULD specify a 'Cache-Control' header indicating a very short duration (e.g., max-age=60) or "no-cache" to indicate that the response (redirect, reference, or content) is not appropriate to cache at the HTTP layer.

7. Alternates and Roll-over

To indicate alternate PKIX certificates (such as when an existing certificate will soon expire), the returned fingerprints document MAY contain multiple fingerprint descriptors. The fingerprints SHOULD be ordered with the most relevant certificate first as determined by the application service operator (e.g., the renewed certificate), followed by the next most relevant certificate (e.g., the certificate soonest to expire). Here is an example:

```
{
  "fingerprints": [
    {
      "sha-1":"UpjRI/A3afKE8/AIeTZ5o1dECTY=",
      "sha-256":"4/mggdlVx8A3pvHAWW5sD+qJyMtUHgiRuPjVC48N0XQ"
    },
    {
      "sha-1":"T29tGO9d7kxbfWnUaac8+5+ICLM=",
      "sha-256":"otyLADSKjRDjVpj8X7/hmCAD5C7Qe+PedcmYV7cUncE="
    }
  ],
  "expires": 806400
}
```

Rolling over from one hosting provider to another is best handled by updating the relevant SRV records, not primarily by updating the POSH files themselves.

8. IANA Considerations

This document registers a well-known URI [RFC5785] for protocols that use POSH. The completed template follows.

URI suffix: posh.

Change controller: IETF

Specification document: [[this document]]

Related information: Because the "posh." string is merely a prefix, protocols that use POSH need to register particular URIs that are prefixed with the "posh." string.

Note that the registered URI is "posh." (with a trailing dot). This is merely a prefix to be placed at the front of well-known URIs [RFC5785] registered by protocols that use POSH, which themselves are

responsible for the relevant registrations with the IANA. The URIs registered by such protocols SHOULD match the URI template [RFC6570] path "/.well-known/posh.{servicedesc}.json"; that is, begin with "posh." and end with ".json" (indicating a media type of application/json [RFC7159]).

For POSH-using protocols that rely on DNS SRV records [RFC2782], the "{servicedesc}" part of the well-known URI SHOULD be "{service}.{proto}", where the "{service}" is the DNS SRV "Service" prepended by the underscore character "_" and the "{proto}" is the DNS SRV "Proto" also prepended by the underscore character "_". As an example, the well-known URI for XMPP server-to-server connections would be "posh._xmpp-server._tcp.json" since XMPP [RFC6120] registers a service name of "xmpp-server" and uses TCP as the underlying transport protocol.

For other POSH-using protocols, the "{servicedesc}" part of the well-known URI can be any unique string or identifier for the protocol, which might be a service name registered with the IANA in accordance with [RFC6335] or which might be an unregistered name. As an example, the well-known URI for the mythical "foo" service could be "posh.foo.json".

Note: As explained in [RFC5785], the IANA registration policy [RFC5226] for well-known URIs is Specification Required.

9. Security Considerations

This document supplements but does not supersede the security considerations provided in specifications for application protocols that decide to use POSH (e.g., [RFC6120] and [RFC6125] for XMPP). Specifically, the security of requests and responses sent via HTTPS depends on checking the identity of the HTTP server in accordance with [RFC2818]. Additionally, the security of POSH can benefit from other HTTP hardening protocols, such as HSTS [RFC6797] and key pinning [I-D.ietf-websec-key-pinning], especially if the POSH client shares some information with a common HTTPS implementation (e.g., platform-default web browser).

Note well that POSH is used by a POSH client to obtain the public key of a POSH server to which it might connect for a particular application protocol such as IMAP or XMPP. POSH does not enable a hosted domain to transfer private keys to a hosting service via HTTPS. POSH also does not enable a POSH server to engage in certificate enrollment with a certification authority via HTTPS, as is done in Enrollment over Secure Transport [RFC7030].

A web server at the Source Domain might redirect an HTTPS request to another URL. The location provided in the redirect response MUST specify an HTTPS URL. Source domains SHOULD use only temporary redirect mechanisms, such as HTTP status codes 302 (Found) and 307 (Temporary Redirect). Clients MAY treat any redirect as temporary, ignoring the specific semantics for 301 (Moved Permanently) and 308 (Permanent Redirect) [RFC7238]. To protect against circular references, clients MUST NOT follow an infinite number of redirects. It is RECOMMENDED that clients follow no more than 10 redirects, although applications or implementations can require that fewer redirects be followed.

Hash function agility is an important quality to ensure secure operations in the face of attacks against the fingerprints obtained within verification materials. Because POSH verification materials are relatively short-lived compared to long-lived credentials such as PKIX end-entity certificates (at least as typically deployed), entities that deploy POSH are advised to swap out POSH files if the hash functions in use are found to be subject to realistic attacks.

10. References

10.1. Normative References

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

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

[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 (CRL) Profile", RFC 5280, May 2008.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

[RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

10.2. Informative References

[I-D.ietf-dane-srv]
Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA Records with SRV Records", draft-ietf-dane-srv-06 (work in progress), June 2014.

[I-D.ietf-websec-key-pinning]
Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", draft-ietf-websec-key-pinning-14 (work in progress), June 2014.

[I-D.ietf-xmpp-dna]
Saint-Andre, P. and M. Miller, "Domain Name Associations (DNA) in the Extensible Messaging and Presence Protocol (XMPP)", draft-ietf-xmpp-dna-05 (work in progress), February 2014.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

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

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

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

[RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, November 2012.

[RFC7030] Pritikin, M., Yee, P., and D. Harkins, "Enrollment over Secure Transport", RFC 7030, October 2013.

[RFC7238] Reschke, J., "The Hypertext Transfer Protocol Status Code 308 (Permanent Redirect)", RFC 7238, June 2014.

[HASH-NAMES] "Hash Function Textual Names", <http://www.iana.org/assignments/hash-function-text-names/hash-function-text-names.xhtml>.

Appendix A. Acknowledgements

Many thanks to Thijs Alkemade, Philipp Hancke, Joe Hildebrand, and Tobias Markmann for their implementation feedback. Thanks also to Dave Cridland, Chris Newton, Max Pritikin, and Joe Salowey for their input on the specification.

Authors' Addresses

Matthew Miller
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO 80202
USA

Email: mamille2@cisco.com

Peter Saint-Andre
&yet

Email: peter@andyet.com
URI: https://andyet.com/