Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2015

A. Popov
M. Nystroem
Microsoft Corp.
D. Balfanz, Ed.
A. Langley
Google Inc.

October 13, 2014

Token Binding over HTTP
draft-balfanz-https-token-binding-00

Abstract

This document describes a collection of mechanisms that allow HTTP servers to cryptographically bind authentication tokens (such as cookies and OAuth tokens) to a TLS [RFC5246] connection.

We describe both _first-party_ as well as _federated_ scenarios. In a first-party scenario, an HTTP server issues a security token (such as a cookie) to a client, and expects the client to send the security token back to the server at a later time in order to authenticate. Binding the token to the TLS connection between client and server protects the security token from theft, and ensures that the security token can only be used by the client that it was issued to.

Federated token bindings, on the other hand, allow servers to cryptographically bind security tokens to a TLS [RFC5246] connection that the client has with a _different_ server than the one issuing the token.

This Internet-Draft is a companion document to The Token Binding Protocol [DraftPopov]

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 16, 2015.

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. The Token-Binding Header 3
3. Federation Use Cases 4
3.1. Introduction 4
3.2. Overview 4
3.3. HTTP Redirects 5
3.4. Cross-Origin Resource Sharing 6
3.5. Negotiated Key Parameters 7
4. Security Considerations 7
4.1. Security Token Replay 7
4.2. Privacy Considerations 7
4.3. Triple Handshake Vulnerability in TLS 8
5. References 8
5.1. Normative References 8
5.2. Informative References 9
Authors' Addresses 9

1. Introduction

The Token Binding Protocol [DraftPopov] defines a Token Binding ID for a TLS connection between a client and a server. The Token Binding ID of a TLS connection is related to a private key that the client proves possession of to the server, and is long-lived (i.e., subsequent TLS connections between the same client and server have the same Token Binding ID). When issuing a security token (e.g. an HTTP cookie or an OAuth token) to a client, the server can include the Token Binding ID in the token, thus cryptographically binding the

token to TLS connections between that particular client and server, and inoculating the token against theft by attackers.

While the Token Binding Protocol [DraftPopov] defines a message format for establishing a Token Binding ID, it doesn't specify how this message is embedded in higher-level protocols. The purpose of this specification is to define how TokenBindingMessages are embedded in HTTP (both versions 1.1 [RFC2616] and 2 [I-D.ietf-httpbis-http2]). Note that TokenBindingMessages are only defined if the underlying transport uses TLS. This means that Token Binding over HTTP is only defined when the HTTP protocol is layered on top of TLS (commonly referred to as HTTPS).

HTTP clients establish a Token Binding ID with a server by including a special HTTP header in HTTP requests. The HTTP header value is a TokenBindingMessage.

TokenBindingMessages allow clients to establish multiple Token Binding IDs with the server, by including multiple TokenBinding structures in the TokenBindingMessage. By default, a client will establish a _provided_ Token Binding ID with the server, indicating a Token Binding ID that the client will persistently use with the server. Under certain conditions, the client can also include a _referred_ Token Binding ID in the TokenBindingMessage, indicating a Token Binding ID that the client is using with a _different_ server than the one that the TokenBindingMessage is sent to. This is useful in federation scenarios.

1.1. Requirements Language

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. The Token-Binding Header

Once a client and server have negotiated the Token Binding Protocol with HTTP/1.1 or HTTP/2 (see The Token Binding Protocol [DraftPopov]), clients MUST include the following header in their HTTP requests:

```
Token-Binding: EncodedTokenBindingMessage
```

The EncodedTokenBindingMessage is a web-safe Base64-encoding of the TokenBindingMessage as defined in the TokenBindingProtocol [DraftPopov].

The TokenBindingMessage MUST contain a TokenBinding with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the server that the HTTP request is sent to (clients use different Token Binding keys for different servers). The Token Binding ID established by this TokenBinding is called a _Provided Token Binding ID_

In HTTP/2, the client SHOULD use Header Compression [I-D.ietf-httpbis-header-compression] to avoid the overhead of repeating the same header in subsequent HTTP requests.

3. Federation Use Cases

3.1. Introduction

For privacy reasons, clients use different private keys to establish Provided Token Binding IDs with different servers. As a result, a server cannot bind a security token (such as an OAuth token or an OpenID Connect identity token) to a TLS connection that the client has with a different server. This is, however, a common requirement in federation scenarios: For example, an Identity Provider may wish to issue an identity token to a client and cryptographically bind that token to the TLS connection between the client and a Relying Party.

In this section we describe mechanisms to achieve this. The common idea among these mechanisms is that a server (called the _Token Consumer_ in this document) gives the client permission to reveal the Provided Token Binding ID that is used between the client and itself, to another server (called the _Token Provider_ in this document). Also common across the mechanisms is how the Token Binding ID is revealed to the Token Provider: The client uses the Token Binding Protocol [DraftPopov], and includes a TokenBinding structure in the Token-Binding HTTP header defined above. What differs between the various mechanisms is _how_ the Token Consumer grants the permission to reveal the Token Binding ID to the Token Provider.

3.2. Overview

In a Federated Sign-On protocol, an Identity Provider issues an identity token to a client, which sends the identity token to a Relying Party to authenticate itself. Examples of this include OpenID Connect (where the identity token is called "ID Token") and SAML (where the identity token is a SAML assertion).

To better protect the security of the identity token, the Identity Provider may wish to bind the identity token to the TLS connection

between the client and the Relying Party, thus ensuring that only said client can use the identity token: The Relying Party will compare the Token Binding ID in the identity token with the Token Binding ID of the TLS connection between it an the client.

This is an example of a federation scenario, which more generally can be described as follows:

- A Token Consumer causes the client to issue a token request to the Token Provider. The goal is for the client to obtain a token and then use it with the Token Consumer.
- The client delivers the token request to the Token Provider.
- The Token Provider issues the token. The token is issued for the specific Token Consumer who requested it (thus preventing malicious Token Consumers from using tokens with other Token Consumers). The token is, however, typically a bearer token, meaning that any client can use it with the Token Consumer, not just the client to which it was issued.
- Therefore, in the previous step, the Token Provider may want to include the Token Binding ID of the TLS connection between the client and the Token Consumer in the token.
- That Token Binding ID must therefore be communicated to the Token Provider along with the token request. Communicating a Token Binding ID involves proving possession of a private key and is described in the Token Binding Protocol [DraftPopov].

The client will perform this last operation (proving possession of a private key that corresponds to a Token Binding ID between the client and the Token Consumer while delivering the token request to the Token Provider) only if the Token Consumer permits the client to do so.

Below, we will enumerate a number of mechanisms available to Token Consumers to grant this permission.

3.3. HTTP Redirects

When a Token Consumer redirects the client to a Token Provider as a means to deliver the token request, it SHOULD include the following HTTP response header in its HTTP response:

```
Include-Referer-Token-Binding-ID: true
```

Including this response header signals to the client that it should reveal the Token Binding ID used between the client and the Token Consumer to the Token Provider. In the absence of this response header, the client will not disclose any information about the Token Binding used between the client and the Token Consumer to the Token Provider.

This header has only meaning if the HTTP status code is 302 or 301, and MUST be ignored by the client for any other status codes. If the client supports the Token Binding Protocol, and has negotiated the Token Binding Protocol with both the Token Consumer and the Token Provider, it already sends the following header to the Token Provider with each HTTP request (see above):

```
Token-Binding: EncodedTokenBindingMessage
```

The TokenBindingMessage SHOULD contain a TokenBinding with TokenBindingType referred_token_binding. If included, this TokenBinding MUST be signed with the Token Binding key used by the client for connections between itself and the Token Consumer (more specifically, the web origin that issued the Include-Referer-Token-Binding-ID response header). The Token Binding ID established by this TokenBinding is called a _Referred Token Binding ID_.

As described above, the TokenBindingMessage MUST additionally contain a Provided Token Binding ID, i.e., a TokenBinding structure with TokenBindingType provided_token_binding, which MUST be signed with the Token Binding key used by the client for connections between itself and the Token Privider (more specifically, the web origin that the token request sent to).

3.4. Cross-Origin Resource Sharing

When issuing an XML HTTP request across origins to a Token Provider, a Token Consumer can reveal its Token Binding ID through the withRefererTokenBindingID property of the XmlHttpRequest object. Example:

```
var xhr = new XMLHttpRequest();
xhr.withCredentials = true; // send cookies
xhr.withRefererTokenBindingID = true;
xhr.open(method, url, true);
```

The client SHOULD include the Token-Binding: header to the outgoing request as described above if:

- the withRefererTokenBindingID property of the XmlHttpRequest object is set to true, and
- the client has negotiated the Token Binding Protocol both with the web origin that issued the XmlHttpRequest, and the web origin to which the XmlHttpRequest is addressed.

3.5. Negotiated Key Parameters

The Token Binding Protocol [DraftPopov] allows the server and client to negotiate a signature algorithm used in the TokenBindingMessage. It is possible that the Token Binding ID used between the client and the Token Consumer, and the Token Binding ID used between the client and Token Provider, use different signature algorithms. The client MUST use the signature algorithm negotiated with the Token Consumer in the referred_token_binding TokenBinding of the TokenBindingMessage, even if that signature algorithm is different from the one negotiated with the origin that the header is sent to.

Token Providers SHOULD support all the SignatureAndHashAlgorithms specified in the Token Binding Protocol [DraftPopov]. If a token provider does not support the SignatureAndHashAlgorithm specified in the referred_token_binding TokenBinding in the TokenBindingMessage, it MUST issue an unbound token.

4. Security Considerations

4.1. Security Token Replay

The goal of the Federated Token Binding mechanisms is to prevent attackers from exporting and replaying tokens used in protocols between the client and Token Consumer, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by malware present in the client. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. The Token Binding private key is therefore a high-value asset and MUST be strongly protected, ideally by generating it in a hardware security module that prevents key export.

4.2. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. Unique Token Binding IDs MUST be generated for connections to different origins, so they cannot be used by cooperating servers to link user identities.

4.3. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens. For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

5. References

5.1. Normative References

[DraftPopov] Popov, A., "The Token Binding Protocol Version 1.0", 2014.

[I-D.ietf-httpbis-header-compression] Peon, R. and H. Ruellan, "HPACK - Header Compression for HTTP/2", draft-ietf-httpbis-header-compression-09 (work in progress), July 2014.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

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

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

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, July 2014.

5.2. Informative References

[I-D.ietf-httpbis-http2]
Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-14 (work in progress), July 2014.

[I-D.ietf-tls-session-hash]
Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-02 (work in progress), October 2014.

[TRIPLE-HS]
Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. IEEE Symposium on Security and Privacy", 2014.

Authors' Addresses

Andrei Popov
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz (editor)
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com

Network Working Group
Internet-Draft
Updates: 7001 (if approved)
Intended status: Standards Track
Expires: June 22, 2015

F. Martin, Ed.
LinkedIn

December 19, 2014

Authentication-Results Registration for TLS draft-martin-authentication-results-tls-03

Abstract

This memo updates the registry of properties in Authentication-Results: message header fields to allow relaying of the results of an email sent using STARTTLS [RFC3207] or not.

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 June 22, 2015.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirements Language 2
1.2. Discussion 2
2. Definitions 3
2.1. results meaning 3
2.2. properties 4
3. IANA Considerations 5
4. Security Considerations 7
5. References 7
5.1. Normative References 7
5.2. Informative References 8
Appendix A. Authentication-Results Examples 8
A.1. TLS Results 8
Author's Address 9

1. Introduction

STARTTLS [RFC3207] defines how to send an email over an SMTP [RFC5321] encrypted session between two mail servers.

This memo thus registers an additional reporting property allowing a TLS result to be relayed as an annotation in a message header.

1.1. Requirements Language

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].

1.2. Discussion

STARTTLS [RFC3207] defines how to send an email over an encrypted session between two mail servers, Message Transfer Agent (MTA), using the TLS [RFC5246] protocol.

Most of these exchanges are opportunistic, meaning a best effort is done to establish an encrypted message exchange regardless of the strength of the cipher or the validity of the certificates used. However, the results of this negotiation should be recorded in the message via the Authentication-Results header [RFC7001] to indicate to other message processing algorithms, including Messaging User Agents (MUA), how securely this message was transmitted from the MTA client to the MTA server.

The concept of authentication here is related to the presentation of a certificate which is verified valid by a set of trusted Certificate

Authorithies (CA), via DANE [RFC6698] or by local policy. This does not indicate that any string in the certificate is related to any string in the email.

The usage and usefulness of the Authentication-Results header is discussed in [RFC7001].

2. Definitions

This memo adds to the "Email Authentication Methods" registry, created by IANA upon publication of [RFC7001], the following:

- The method "tls"; and
- Associated with that method, the properties (reporting items) "cert.client", "cert.server", "cert.verif", "tls.v", "key.ciphersuite", "key.fingerprint", "key.length" and "key.strength".

2.1. results meaning

The "tls" method can have the following results:

none: the message was sent in clear.

pass: the message was sent encrypted and the client certificate was verified valid either using a trusted CA, via DANE [RFC6698] or via a local policy and host identity was verified.

selfsigned: the message was sent encrypted but the client certificate is self signed.

invalidhost: the message was sent encrypted and the client certificate is verified valid but the host identity is invalid.

fail: the message was sent encrypted but the client certificate is not valid. It is advised to use comments to indicate the nature of the problem like certifcate expired, not linked to a trusted CA,...

temperror: the message was sent encrypted and the server was not able to verify the client certificate at this time. This may indicate for instance that the server could not fetch the CRL.

permerror: the message was sent encrypted and the client certificate was not verified by the MTA server. MTA should always attempt to verify the client certificate.

2.2. properties

cert.client: the subject of the X.509 certificate used by the client to initiate TLS.

cert.server: the subject of the X.509 certificate used by the server to initiate TLS (optional).

cert.clientalt: the subject alternative name of the X.509 certificate used by the client to initiate TLS (optional).

cert.serveralt: the subject alternative name of the X.509 certificate used by the server to initiate TLS (optional).

cert.clientissuer: the issuer of the X.509 certificate used by the client to initiate TLS (optional).

cert.serverissuer: the issuer of the X.509 certificate used by the server to initiate TLS (optional).

cert.verif: the type of certification performed: CA, DANE [RFC6698], LOCAL (optional).

tls.v: the protocol version used to encrypt SSL2.0, SSL3.0, TLS1.0, TLS1.1,... (optional)

key.ciphersuite: the description of the TLS cipher suite used as defined in the IANA cipher suite registry.

key.fingerprint: the fingerprint of the key used (optional).

key.length: the length in bits of the key used (optional).

key.strength: as many SMTP TLS are opportunistic in nature this property is an arbitrary value set by the MTA server to indicate the strength of the encryption (optional).

While ciphers strength vary overtime, and key length in bits does not indicate a comparable strength between various cyphers, it may be difficult for all the processors of the authentication-results header to redo the analysis based on the cipher used and all to arrive to the same conclusion. It seems, therefore, best if the receiving MTA does that analysis and communicate it to the other layers. This is the purpose of the key.strength. For instance This value could be used by the MUA to indicate to the end user some quality of the encryption channel.

3. IANA Considerations

Per [IANA], the following items have been added to the "Email Authentication Methods" registry:

Method	Defined	ptype	property	value
tls	RFC 3207	cert	client	subject of client certificate section 4.1.2.6 of RFC 5280
tls	RFC 3207	cert	server	subject of server certificate section 4.1.2.6 of RFC 5280
tls	RFC 3207	cert	clientalt	alternate subject of client certificate section 4.2.1.6 of RFC 5280
tls	RFC 3207	cert	serveralt	alternate subject of server certificate section 4.2.1.6 of RFC 5280
tls	RFC 3207	cert	clientissuer	issuer of client certificate section 4.1.2.4 of RFC 5280
tls	RFC 3207	cert	serverissuer	issuer of server certificate section 4.1.2.4 of RFC 5280
tls	RFC 3207	cert	verif	CA DANE LOCAL

tls	RFC 3207	tls	v	protocol version description from RFC 5246
tls	RFC 3207	key	ciphersuite	IANA cipher suite registry description from RFC 5246
tls	RFC 3207	key	fingerprint	key fingerprint from RFC 5246
tls	RFC 3207	key	length	in bits
tls	RFC 3207	key	strength	low medium high

Also, the following items have been added to the "Email Authentication Result Names" registry:

Code	Existing/New	Defined In	Method	Meaning
none	existing	RFC 7001	tls (added)	this memo
pass	existing	RFC 7001	tls (added)	this memo
selfsigned	existing	RFC 7001	tls (added)	this memo
invalidhost	existing	RFC 7001	tls (added)	this memo
fail	existing	RFC 7001	tls (added)	this memo
temperror	existing	RFC 7001	tls (added)	this memo
permerror	existing	RFC 7001	tls (added)	this memo

4. Security Considerations

This memo creates a mechanism for relaying STARTTLS [RFC3207] results using the structure already defined by [RFC7001]. The Security Considerations sections of those documents should be consulted.

By this mechanism, some identifiers of the client certificates gets to live pass the receiving MTA. This is a change in the sender expectation on where the client certificate is used

5. References

5.1. Normative References

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

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, February 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[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.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 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.

[RFC7001] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 7001, September 2013.

5.2. Informative References

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

Appendix A. Authentication-Results Examples

This section presents an example of the use of this new header field to indicate TLS results.

A.1. TLS Results

A message that went over a successful TLS session:

```
Authentication-Results: mail-router.example.net;
  dkim=pass (good signature) header.d=newyork.example.com
    header.b=oINEO8hg;
  tls=pass (verified, expires 20140505)
    cert.verif=CA
    cert.client="CN=smtp.example.com,O=ACME,L=ToonTown,
      ST=CA,C=US"
    cert.clientalt="DNS:smtp.example.com, DNS:newyork.example.com"
    cert.clientissuer="C=US, O=AcmeCert Inc, CN=AcmeCert CA"
    key.ciphersuite=TLS_RSA_WITH_RC4_128_SHA
    tls.v=TLS1.0
    key.fingerprint="68:B3:29:DA:98:93:E3:40:99:C7:D8:
      AD:5C:B9:C9:40"
    key.length=128
    key.strength=MEDIUM;
Received: from newyork.example.com
  (newyork.example.com [192.0.2.250])
  by mail-router.example.net (8.11.6/8.11.6)
  for <recipient@example.net>
  with ESMTPS id i7PK0sH7021929;
  Fri, Feb 15 2002 17:19:22 -0800
DKIM-Signature: v=1; a=rsa-sha256; s=rashani;
  d=newyork.example.com;
  t=1188964191; c=relaxed/simple;
  h=From:Date:To:VBR-Info:Message-Id:Subject;
  bh=sEu28nfs9fuZGD/pSr7ANysbY3jtdaQ3Xv9xPQtS0m7=;
   b=oINEO8hgn/gnunsg ... 9n9ODSNFSDij3=
From: sender@newyork.example.com
Date: Fri, Feb 15 2002 16:54:30 -0800
To: meetings@example.net
Message-Id: <12345.abc@newyork.example.com>
Subject: here’s a sample
```

Author’s Address

Franck Martin (editor)
LinkedIn
Mountain View, CA
US

Email: fmartin@linkedin.com

Network Working Group
Internet-Draft
Updates: 1939, 3464, 3501, 5068, 6186 (if approved)
Intended status: Standards Track
Expires: February 17, 2015

K. Moore
Network Heretics
C. Newman
Oracle

August 16, 2014

Deployable Enhanced Email Privacy (DEEP)
draft-newman-email-deep-02.txt

Abstract

This specification defines a set of requirements and facilities designed to improve email privacy. This provides mechanisms intended to increase use of already deployed Transport Layer Security (TLS) technology, provide a model for mail user agents privacy assurance, and enable mail service providers to advertise improved TLS privacy facilities.

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 February 17, 2015.

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.

Table of Contents

1. Introduction 4
2. Conventions and Terminology Used in This Document 4
3. Mail Account Privacy Assurance Level 5
3.1. High Privacy Assurance 5
3.2. Certificate Pinning 6
3.3. Low Privacy Assurance 6
3.4. Other Privacy Assurance Levels 7
4. Implicit TLS 7
4.1. Implicit TLS for POP 7
4.2. Implicit TLS for IMAP 8
4.3. Implicit TLS for SMTP Submission 8
4.4. Implicit TLS Connection Closure for POP, IMAP and SMTP . 8
5. Email Security Upgrading Using Security Latches 9
5.1. Email Security Tags 9
5.2. Initial Set of Email Security Tags 10
5.3. Server DEEP Status 10
5.4. Email Security Tag Latch Failures 11
6. Recording TLS Cipher Suite in Received Header 11
7. Extensions for DEEP Status and Reporting 12
7.1. IMAP DEEP Extension 12
7.2. POP DEEP Extension 14
7.3. SMTP DEEP Extension 15
7.4. SMTP Error Extension 16
8. Use of SRV records in Establishing Configuration 16
9. Implementation Requirements 17
9.1. All Implementations (Client and Server) 17
9.1.1. Client Certificate Authentication 18
9.2. Mail Server Implementation Requirements 18
9.3. Mail User Agent Implementation Requirements 19
9.4. Non-configurable MUAs and nonstandard access protocols . 20
9.5. DEEP Compliance for Anti-Virus/Anti-Spam Software and Services 20
10. Mail Service Provider Requirements 20
10.1. Server Requirements 20
10.2. MSPs MUST provide Submission Servers 20
10.3. TLS Server Certificate Requirements 21
10.4. Recommended DNS records for mail protocol servers 21
10.4.1. MX records 21
10.4.2. SRV records 21
10.4.3. TLSA records 22
10.4.4. DNSSEC 22

10.5. MSP Server Monitoring 22
10.6. Advertisement of DEEP status 22
10.7. Require TLS 22
11. IANA Considerations 22
11.1. Security Tag Registry 22
11.2. Initial Set of Security Tags 23
11.3. POP3S Port Registration Update 25
11.4. IMAPS Port Registration Update 25
11.5. Submissions Port Registration 26
11.6. DEEP IMAP Capability 27
11.7. DEEP POP3 Capability 27
11.8. DEEP SMTP EHLO Keyword 27
11.9. SMTP Enhanced Status Code 27
11.10. MAIL Parameters Additional-registered-clauses Sub-Registry 28
12. Security Considerations 28
13. References 29
13.1. Normative References 29
13.2. Informative References 30
Appendix A. Design Considerations 31
Appendix B. Open Issues 32
Appendix C. Change Log 34
Appendix D. Acknowledgements 35
Authors' Addresses 35

1. Introduction

Software that provides email service via Internet Message Access Protocol (IMAP) [RFC3501], Post Office Protocol (POP) [RFC1939] and/or Simple Mail Transfer Protocol (SMTP) [RFC5321] usually has Transport Layer Security (TLS) [RFC5246] support but often does not use it in a way that maximizes end-user privacy. This specification proposes changes to email software and deployments intended to increase the use of TLS and record when that use occurs.

In brief, this memo now recommends that:

- MUAs associate a privacy assurance level with each mail account, and the default privacy level requires use of TLS with certificate validation for all TCP connections;

- TLS on a well-known port ("Implicit TLS") be supported for IMAP, POP, and SMTP Submission [RFC6409] for all electronic mail user agents (MUAs), servers, and service providers;

- MUAs and mail protocol servers cooperate (via mechanisms defined in this specification) to upgrade security/privacy feature use and record/indicate that usage appropriately.

Improved use of TLS with SMTP for message relaying is described in a separate document [I-D.ietf-dane-smtp-with-dane].

The recommendations in this memo do not replace the functionality of, and are not intended as a substitute for, end-to-end encryption of electronic mail.

This draft is subject to change. Implementation of this proposal is not recommended at this time. Please discuss this proposal on the ietf-uta mailing list.

2. Conventions and Terminology Used in This Document

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].

This specification expresses syntax using the Augmented Backus-Naur Form (ABNF) as described in [RFC5234], including the core rules in Appendix B and rules from [RFC5322].

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. If a single "C:" or "S:" label applies to

multiple lines, then the line breaks between those lines are for editorial clarity only and are not part of the actual protocol exchange.

3. Mail Account Privacy Assurance Level

The configuration necessary for a mail account includes an email address, connection information and authentication credentials for at least one mail access server (IMAP or POP) and at least one SMTP submission server. A mail user agent (MUA) typically supports one or more mail account configurations. MUAs compliant with this specification MUST associate a privacy assurance level with each mail account. MUAs MUST implement a high privacy level as described in the next section.

MUAs SHOULD continuously indicate to the user the privacy for an account's connections (e.g., via a lock icon, background colors and indications similar to those commonly used in web browsers for this purpose). Note that this could be higher than the level set at account configuration but never lower. If multiple active connections are associated with an account or view, the indication should match the privacy level provided by the least private connection.

Account configuration occurs when an MUA is first used to access a particular service, when a user wishes to access or submit mail through servers in addition to those specified or found during first use, or when a user explicitly requests to change account configuration parameters such as server names, user names, passwords, client certificates, etc. Account configuration can be entirely manual (entering server names explicitly) or partially automated via a mechanism such as DNS SRV records [RFC6186]. MUAs SHOULD use the high privacy assurance level as the default for newly configured accounts.

3.1. High Privacy Assurance

A mail account has a high privacy assurance when the following conditions are met on all TCP server connections associated with an account. This includes connections to POP, IMAP and SMTP submission servers as well as any other associated protocols defined now or in the future. Examples of protocols associated with a mail account include managesieve [RFC5804] and MTQP [RFC3887].

- TCP connections MUST attempt to negotiate TLS via either Implicit TLS Section 4 or STARTTLS.

- MUAs MUST implement [I-D.melnikov-email-tls-certs] and PKIX [RFC5280].

- MUAs MAY implement DANE [RFC6698].

- User agents MUST abort a TLS session if the TLS negotiation fails or the server's certificate or identity fails to verify. A user may reconfigure the account to lower the expected level of privacy if he/she chooses. Reduction of expected account privacy MUST NOT be done on a click-through basis.

The end user is part of the system that protects the user's privacy and security. As a result, it's critical not to present the end user with a simple action that reduces their privacy in response to certificate validation failure. An MUA which offers a user actions such as "connect anyway", "trust certificate for future connections" or "lower privacy assurance for this account" in response to certificate validation failure is not providing a high privacy assurance as defined in this section and thus does not comply with this document. Examples of acceptable actions to offer would be "work offline", "try again later", and "open service provider status web page".

3.2. Certificate Pinning

MUAs MAY implement certificate pinning as part of account setup, but MUST NOT offer this as an option in response to a failed certificate validation for an existing account. Certificate pinning occurs when the user agent saves a server certificate with the account settings and trusts that certificate for subsequent connections to that server. An MUA that allows certificate pinning MUST NOT allow a certificate pinned for one account to validate connections for other accounts.

A pinned certificate is subject to a man-in-the-middle attack at account setup time, and lacks a mechanism to revoke or securely refresh the certificate. Therefore use of a pinned certificate does not provide a high privacy assurance and an MUA MUST NOT indicate a high privacy level for an account or connection using a pinned certificate.

3.3. Low Privacy Assurance

MUAs MAY implement a low privacy assurance level for accounts. At this level, the MUA MUST attempt to negotiate TLS, but MAY ignore server certificate validation failures. MUAs MAY support use of connections without TLS, but if they do they SHOULD attempt TLS first if available and MUST implement code to reconnect without TLS if TLS

negotiation fails for reasons other than certificate validity.

Note that if the TLS certificate is not successfully validated as described in Section 3.1 or a version of SSL/TLS prior to TLS 1.0 is used, the client MUST NOT present a high privacy indication for the account or connection.

3.4. Other Privacy Assurance Levels

This specification is not intended to limit experimentation and innovation with respect to user privacy. As a result more privacy assurance levels are permitted. However, levels below the "low privacy assurance" described in the previous section are discouraged and implementers are cautioned that end users may be confused by too many privacy levels.

4. Implicit TLS

Previous standards for use of email protocols with TLS used the STARTTLS mechanism: [RFC2595], [RFC3207], and [RFC3501]. With STARTTLS, the client establishes a clear text application session and determines whether to issue a STARTTLS command based on server capabilities and client configuration. If the client issues a STARTTLS command, a TLS handshake follows that can upgrade the connection. While this mechanism has deployed, an alternate mechanism where TLS is negotiated immediately at connection start on a separate port (referred to in this document as "Implicit TLS") has deployed more successfully. To increase use of TLS, this specification recommends use of implicit TLS by new POP, IMAP and SMTP Submission software.

4.1. Implicit TLS for POP

When a TCP connection is established for the "pop3s" service (default port 995), a TLS handshake begins immediately. Clients MUST implement the certificate validation mechanism described in [I-D.melnikov-email-tls-certs]. Once the TLS session is established, POP3 [RFC1939] protocol messages are exchanged as TLS application data for the remainder of the TCP connection. After the server sends a +OK greeting, the server and client MUST enter AUTHORIZATION state, even if client credentials were supplied during the TLS handshake.

See Section 9.1.1 for additional information on client certificate authentication. See Section 11.3 for port registration information.

4.2. Implicit TLS for IMAP

When a TCP connection is established for the "imaps" service (default port 993), a TLS handshake begins immediately. Clients MUST implement the certificate validation mechanism described in [RFC3501] and SHOULD implement the certificate validation mechanism described in [I-D.melnikov-email-tls-certs]. Once the TLS session is established, IMAP [RFC3501] protocol messages are exchanged as TLS application data for the remainder of the TCP connection. If client credentials were provided during the TLS handshake that the server finds acceptable, the server MAY issue a PREAUTH greeting in which case both the server and client enter AUTHENTICATED state. If the server issues an OK greeting then both server and client enter NOT AUTHENTICATED state.

See Section 9.1.1 for additional information on client certificate authentication. See Section 11.4 for port registration information.

4.3. Implicit TLS for SMTP Submission

When a TCP connection is established for the "submissions" service (default port 465), a TLS handshake begins immediately. Clients MUST implement the certificate validation mechanism described in [I-D.melnikov-email-tls-certs]. Once a TLS session is established, message submission protocol data [RFC6409] is exchanged as TLS application data for the remainder of the TCP connection. (Note: the "submissions" service name is defined in section 10.3 of this document, and follows the usual convention that the name of a service layered on top of Implicit TLS consists of the name of the service as used without TLS, with an "s" appended.)

Note that the submissions port provides access to a Mail Submission Agent (MSA) as defined in [RFC6409] so requirements and recommendations for MSAs in that document apply to the submissions port, including the requirement to implement SMTP AUTH [RFC4954].

See Section 9.1.1 for additional information on client certificate authentication. See Section 11.5 for port registration information.

4.4. Implicit TLS Connection Closure for POP, IMAP and SMTP

When a client or server wishes to close the connection, it SHOULD initiate the exchange of TLS close alerts before TCP connection termination. The client MAY, after sending a TLS close alert, gracefully close the TCP connection without waiting for a TLS response from the server.

5. Email Security Upgrading Using Security Latches

Once an improved email security or privacy mechanism is deployed and ready for general use, it is desirable to continue using it for all future email service. For example, TLS is widely deployed in email software, but use of TLS is often not required. At the time this is written, deployed mail user agents (MUAs) [RFC5598] usually make a determination if TLS is available when an account is first configured and may require use of TLS with that account if and only if it was initially available. If the service provider makes TLS available after initial client configuration, many MUAs will not notice the change.

Alternatively, a security feature may be purely opportunistic and thus subject to downgrade attacks. For example, at the time this was written, most TLS stacks that support TLS 1.2 will fallback to TLS 1.0 without alerting the client of the reduced security. Thus a variety of active attacks could cause the loss of TLS 1.2 benefits. Only if client policy is upgraded to require TLS 1.2 can the client prevent all downgrade attacks. However, this sort of security policy upgrade will be ignored by most users unless it is automated.

This section describes a mechanism, called "security latches", which is designed to permit an MUA to recognize when a service provider has committed to provide certain server security features, and that it’s safe for the client to change its configuration for that account to require that such features be present in future sessions with that server. When an MUA implements both privacy assurance levels and security latches, then both the end-user and the service provider independently have the ability to improve the end-user’s privacy.

Note that security latches are a mechanism similar to HTTP Strict Transport Security (HSTS) [RFC6797] but are extensible.

5.1. Email Security Tags

Each security latch is given a name known as an email security tag. An email security tag is a short alphanumeric token that represents a security facility that can be used by an IMAP, POP or SMTP Submission session. When a server advertises a security tag it is making a commitment to support that security facility indefinitely and recommending that the client save that security tag with the account configuration and require that security feature for future connections to that server. When a security tag is saved by the client in this way, it is then considered latched. For the "tls10" and/or "tls12" tags, the client SHOULD refuse to connect to the server unless the appropriate level of TLS is successfully negotiated. If these tags are still advertised by the server after

negotiation, the client SHOULD latch these tags. Other security tags are latched if they are advertised by the server, TLS is active and the client successfully authenticates the server with the TLS session. Once a security tag is latched, all subsequent connections to that host require that security feature. For this privacy protection to work as desired clients MUST NOT offer a click-through-to-connect action when unable to achieve connection security matching the latched security tags.

An identifier for a security tag has the following formal syntax:

```
security-tag  =  ALPHA *63(ALPHA / DIGIT / "-" / "_")
```

5.2. Initial Set of Email Security Tags

This section describes an initial set of email security tags. The IANA Considerations Section 11 defines a registry so that more tags can be defined in the future. The initial set of tags are defined in Section 11.2 and include tls10, tls12, tls-cert and tls-dane-tlsa.

5.3. Server DEEP Status

Servers supporting this extension MUST advertise a DEEP status. This status includes a list of security-tags the server administrator has explicitly configured as recommended for use by end-users (the list MAY be empty), an optional https Uniform Resource Locator (URL) [RFC2818] that the client can save and subsequently resolve for the user in the event of a security connection problem, and the DEEP status can be extended by future updates to this specification. DEEP status has the following formal syntax:

```
EXTCHAR      = 0x20-21 / 0x23-2E / 0x30-3B / 0x3D-40
               / 0x5B-60 / 0x7B-7E
             ; printable characters excluding " \ < and ALPHA

deep-extend  = EXTCHAR *(EXTCHAR / ALPHA / "<")
             ; clients MUST ignore, for future extensibility

deep-status  = [deep-tag *(SP deep-tag)]

deep-tag     = deep-https / security-tag / deep-extend

deep-https   =  "<" <URI from RFC 3986 with https scheme> ">"
```

The syntax for a Uniform Resource Identifier (URI) is defined in [RFC3986]. Protocol extensions to advertise DEEP status are defined in Section 7.

If the client successfully negotiates TLS and authenticates the server (e.g., via tls-cert, tls-dane-tlsa or SCRAM-SHA1-PLUS with channel bindings [RFC5802]), then the client SHOULD record the server's DEEP status information in the account configuration with the server's hostname. Otherwise, the client SHOULD ignore the server-provided DEEP status except for the "tls10" and "tls12" security tags.

5.4. Email Security Tag Latch Failures

When a security tag latch has been set for connections from a client to a server and the property identified by that tag is no longer available, this results in a connection failure. An MUA SHOULD inform the user of a potential threat to their privacy and offer to resolve a previously-recorded DEEP status https URL if one is available. An MUA might suggest deleting the account and re-creating it as a cumbersome mechanism to reset the latches. MUAs are discouraged from offering a lightweight option to reset or ignore latches as this defeats the privacy benefit they provide to end users.

6. Recording TLS Cipher Suite in Received Header

The ESMTPS transmission type [RFC3848] provides trace information that can indicate TLS was used when transferring mail. However, TLS usage by itself is not a guarantee of privacy or security. The TLS cipher suite provides additional information about the level of privacy or security made available for a connection. This defines a new SMTP "tls" Received header additional-registered-clause that is used to record the TLS cipher suite that was negotiated for the connection. The value included in this additional clause SHOULD be the registered cipher suite name (e.g., TLS_DHE_RSA_WITH_AES_128_CBC_SHA) included in the TLS cipher suite registry. In the event the implementation does not know the name of the cipher suite (a situation that should be remedied promptly), a four-digit hexadecimal cipher suite identifier MAY be used. The ABNF for the field follows:

```
tls-cipher-clause  =  CFWS "tls" FWS tls-cipher

tls-cipher         =  tls-cipher-suite-name / tls-cipher-suite-hex

tls-cipher-name    =  ALPHA *(ALPHA / DIGIT / "_")
                      ; as registered in IANA cipher suite registry

tls-cipher-hex     =  "0x" 4HEXDIG
```

7. Extensions for DEEP Status and Reporting

This memo defines optional mechanisms for use by MUAs to communicate DEEP status to servers. One purpose of such mechanisms is to permit servers to determine which and how many clients have latched security facilities, and thus, to permit operators to be aware of potential impact to their users should support for such facilities be changed. For IMAP, the existing ID command is extended to provide this capability. For SMTP Submission, a new CLIENT command is defined. No similar mechanism is defined for POP in this version of the memo to keep POP simpler, but one may be added in the future if deemed necessary.

In addition, for each of IMAP, POP, and SMTP, a new DEEP capability is defined so the client can access the DEEP status.

7.1. IMAP DEEP Extension

When an IMAP server advertises the DEEP capability, that indicates the IMAP server implements IMAP4 ID [RFC2971] with additional field values defined here. This is grouped with the ID command because that is the existing IMAP mechanism for clients to report data for server logging, and provides a way for the server to report the DEEP status.

deep From server to client, the argument to this ID field is the server DEEP status. Servers MUST provide this information in response to an ID command.

latch From client to server, this is a space-separated list of security tags the client has latched for this server. Servers MAY record this information so administrators know the expected privacy level of the client and can thus act to avoid security latch failures (e.g., by renewing server certificates on time, etc).

latch-fail From client to server, a space-separated list including one or more security tag the client has latched that the client was unable to achieve. This allows clients to report errors to the server prior to terminating the connection to the server in the event an acceptable privacy level is unavailable.

security-tags From client to server, this is a space-separated list of security tags the client supports that are not latched.

tls Server-side IMAP proxies that accept TLS connections from clients and connect in-the-clear over a fully private secure network to the server SHOULD use this field to report the tls-cipher (syntax as defined in Section 6) to the server.

IMAP clients SHOULD use the IMAP ID command to report latch failures and determine the server DEEP status. Clients MAY use the ID command to report other latch or security tag information. IMAP servers MUST implement the ID command at least to report DEEP status to clients.

```
<client connected to port 993 and negotiated TLS successfully>
S: * OK [CAPABILITY IMAP4rev1 DEEP ID AUTH=PLAIN
        AUTH=SCRAM-SHA-1] hello
C: a001 ID ("name" "Demo Mail" "version" "1.5" "latch"
        "tls10 tls-cert" "security-tags" "tls12")
S: * ID ("name" "Demo Server" "version" "1.7" "deep-status"
         "<https://www.example.com/privacy-support.html>")
S: a001 OK ID completed
```

Example 1

This example shows a client that successfully negotiated TLS version 1.0 or later and verified the server's certificate as required by IMAP. The client supports TLS 1.2. However, even if the client successfully negotiated TLS 1.2, it will not latch that security tag automatically because the server did not advertise that tag. If the client successfully validated the server certificate, it will latch the provided URL.

```
<client connected to port 993 and negotiated TLS successfully>
S: * OK [CAPABILITY IMAP4rev1 DEEP ID AUTH=PLAIN
        AUTH=SCRAM-SHA-1] hello
C: a001 ID ("name" "Demo Mail" "version" "1.5" "latch-failure"
        "tls-cert")
S: * ID ("name" "Demo Server" "version" "1.7" "deep-status"
         "tls10 <https://www.example.com/privacy-support.html>")
S: a001 OK ID completed
C: a002 LOGOUT
```

Example 2

This example shows a client that negotiated TLS, but was unable to verify the server's certificate. The latch-failure informs the server of this problem, at which point the client can disconnect. If the client had previously latched a URI for privacy problems from this server, it could offer to resolve that URI. However, the deep-status in this exchange is ignored due to the latch failure.

```
<IMAP Proxy connected over private network on port 143, there is
a client connected to the proxy on port 993 that negotiated TLS>
S: * OK [CAPABILITY IMAP4rev1 DEEP ID AUTH=PLAIN
        AUTH=SCRAM-SHA-1] hello
C: a001 ID ("name" "Demo Mail" "version" "1.5" "latch"
        "tls10 tls-cert" "security-tags" "tls12"
        "tls" "TLS_RSA_WITH_AES_128_CBC_SHA")
S: * ID ("name" "Demo Server" "version" "1.7" "deep-status"
         "tls10 tls-cert <https://www.example.com/support.html>")
S: a001 OK ID completed
```

Example 3

This example shows the connection from an IMAP proxy to a back-end server. The client connected to the proxy and sent the ID command shown in example 1, and the proxy has added the "tls" item to the ID command so the back-end server can log the cipher suite that was used on the connection from the client.

7.2. POP DEEP Extension

POP servers supporting this specification MUST implement the POP3 extension mechanism [RFC2449]. POP servers MUST advertise the DEEP capability with an argument indicating the server's DEEP status.

```
<client connected to port 995 and negotiated TLS successfully>
S: +OK POP server ready
C: CAPA
S: +OK Capability list follows
S: TOP
S: SASL PLAIN SCRAM-SHA-1
S: RESP-CODES
S: PIPELINING
S: UIDL
S: DEEP tls10 tls12 <https://www.example.com/privacy-support.html>
S: .
```

Example

After verifying the TLS server certificate and issuing CAPA, the client can latch any or all of the DEEP status. If the client connects to this same server later and has a privacy failure, the client can direct the user's browser to the previously-latched URI where the service provider may provide advice to the end user.

7.3. SMTP DEEP Extension

SMTP Submission servers supporting this specification MUST implement the DEEP SMTP extension. The name of this extension is DEEP. The EHLO keyword value is DEEP and the deep-status ABNF is the syntax of the EHLO keyword parameters. This does not add parameters to the MAIL FROM or RCPT TO commands. This also adds a CLIENT command to SMTP which is used to report client information to the server. The formal syntax for the command follows:

```
deep-cmd             = "CLIENT" 1*(SP deep-parameter)

deep-parameter       = name / version / latch / latch-fail
                       / security-tags / tls / future-extension

name                 = "name=" esmtp-value

version              = "version=" esmtp-value

latch                = "latch=" security-tag *("," security-tag)

latch-fail           = "latch-fail=" security-tag
                       *("," security-tag)

security-tags        = "security-tags=" security-tag
                       *("," security-tag)

tls                  = "tls=" tls-cipher

future-extension     = esmtp-param

esmtp-param          = <as defined in RFC 5321>

esmtp-value          = <as defined in RFC 5321>
```

The CLIENT command parameters listed here have the same meaning as the parameters used in the IMAP DEEP extension (Section 7.1). The server responds to the CLIENT command with a "250" if the command has correct syntax and a "501" if the command has incorrect syntax.

```
<client connected to port 465 and negotiated TLS successfully>
S: 220 example.com Demo SMTP Submission Server
C: EHLO client.example.com
S: 250-example.com
S: 250-8BITMIME
S: 250-PIPELINING
S: 250-DSN
S: 250-AUTH PLAIN LOGIN
S: 250-DEEP tls10 tls-cert <https://www.example.com/status.html>
S: 250-BURL imap
S: 250 SIZE 0
C: CLIENT name=demo_submit version=1.5 latch=tls10,tls-cert
          security-tags=tls12
S: 250 OK
```

Example

7.4. SMTP Error Extension

Although this document focuses on SMTP Submission, it is possible to use security latches for SMTP transport as well. When MTA transport fails due to a security latch, the MTA MUST use the SMTP enhanced status code X.7.TBD. The SMTP notary response [RFC3464] for a security latch failure MUST include an additional "SMTP-Security-Latch" recipient-specific header field that includes a space-delimited list including one or more security latch that failed. The ABNF for this new field follows:

```
CFWS                =  <defined in RFC 5322>

FWS                 =  <defined in RFC 5322>

smtp-security-latch  =  "SMTP-Security-Latch:" CFWS
                        security-tag *(FWS security-tag)
```

8. Use of SRV records in Establishing Configuration

This section updates [RFC6186] by changing the preference rules and adding a new SRV service label _submissions._tcp to refer to Message Submission with implicit TLS.

User-configurable MUAs SHOULD support use of [RFC6186] for account setup. However, when using configuration information obtained by this method, MUAs SHOULD default to a high privacy assurance level, unless the user has explicitly requested reduced privacy. This will have the effect of causing the MUA to ignore advertised configurations which do not support TLS, even when those advertised

configurations have a higher priority than other advertised configurations.

When using [RFC6186] configuration information, Mail User Agents SHOULD NOT automatically establish new configurations that do not require TLS for all servers, unless there are no advertised configurations using TLS. If such a configuration is chosen, prior to attempting to authenticate to the server or use the server for message submission, the MUA SHOULD warn the user that traffic to that server will not be encrypted and that it will therefore likely be intercepted by unauthorized parties. The specific wording is to be determined by the implementation, but it should adequately capture the sense of risk given the widespread incidence of mass surveillance of email traffic.

When establishing a new configuration for connecting to an IMAP, POP, or SMTP Submission server, an MUA SHOULD NOT blindly trust SRV records unless they are signed by DNSSEC and have a valid signature. Instead, the MUA SHOULD warn the user that the DNS-advertised mechanism for connecting to the server is not authenticated, and request the user to manually verify the connection details by reference to his or her mail service provider’s documentation.

Similarly, an MUA MUST NOT consult SRV records to determine which servers to use on every connection attempt, unless those SRV records are signed by DNSSEC and have a valid signature. However, an MUA MAY consult SRV records from time to time to determine if an MSP’s server configuration has changed, and alert the user if it appears that this has happened. This can also serve as a means to encourage users to upgrade their configurations to require TLS if and when their MSPs support it.

9. Implementation Requirements

This section details requirements for implementations of electronic mail protocol clients and servers. A requirement for a client or server implementation to support a particular feature is not the same thing as a requirement that a client or server running a conforming implementation be configured to use that feature. Requirements for Mail Service Providers (MSPs) are distinct from requirements for protocol implementations, and are listed in a separate section.

9.1. All Implementations (Client and Server)

These requirements apply to MUAs as well as POP, IMAP and SMTP Submission servers.

- All implementations MUST be configurable to support implicit TLS using the TLS 1.2 protocol or later [RFC5246] including support for the mandatory-to-implement TLS 1.2 cipher suite TLS_RSA_WITH_AES_128_CBC_SHA.

- IMAP implementations MUST support the IMAP4rev1 mandatory-to-implement cipher suite TLS_RSA_WITH_RC4_128_MD5 for any connections made or received via IMAP although this MAY be disabled by default.

- All implementations MUST be configurable to require TLS before performing any operation other than capability discovery and STARTTLS.

9.1.1. Client Certificate Authentication

MUAs and mail servers MAY implement client certificate authentication on the implicit TLS port. Servers MUST NOT request a client certificate during the TLS handshake unless the server is configured to accept some client certificates as sufficient for authentication and the server has the ability to determine a mail server authorization identity matching such certificates. How to make this determination is presently implementation specific. Clients MUST NOT provide a client certificate during the TLS handshake unless the server requests one and the client has determined the certificate can be safely used with that specific server, OR the client has been explicitly configured by the user to use that particular certificate with that server. How to make this determination is presently implementation specific. If the server accepts the client's certificate as sufficient for authorization, it MUST enable the SASL EXTERNAL [RFC4422] mechanism. An IMAPS server MAY issue a PREAUTH greeting instead of enabling SASL EXTERNAL. A client supporting client certificate authentication with implicit TLS MUST implement the SASL EXTERNAL [RFC4422] mechanism using the appropriate authentication command (AUTH for POP3 [RFC5034], AUTH for SMTP Submission [RFC4954], AUTHENTICATE for IMAP [RFC3501]).

9.2. Mail Server Implementation Requirements

These requirements apply to servers that implement POP, IMAP or SMTP Submission.

- Servers MUST implement the DEEP extension described in Section 7

- IMAP and SMTP submission servers SHOULD implement and be configurable to support STARTTLS. This enables discovery of new TLS availability, and can increase usage of TLS by legacy clients.

- Servers MUST NOT advertise STARTTLS if it is unlikely to succeed based on server configuration (e.g., there is no server certificate installed).

- SMTP message submission servers that have negotiated TLS SHOULD add a Received header field to the message including the tls clause described in Section 6.

- Servers MUST be configurable to include the TLS cipher information in any connection or user logging or auditing facility they provide.

9.3. Mail User Agent Implementation Requirements

This section describes requirements on Mail User Agents (MUAs) using IMAP, POP, and/or Submission protocols. Note: Requirements pertaining to use of Submission servers are also applicable to use of SMTP servers (e.g., port 25) for mail submission.

- User agents SHOULD indicate at configuration time, the expected level of privacy based on appropriate security inputs such as which security latches are pre-set, the number of trust anchors, certificate validity, use of an extended validation certificate, TLS version supported, and TLS cipher suites supported by both server and client.

- MUAs SHOULD detect when STARTTLS and/or implicit TLS becomes available for a protocol and set the tls10 latch if the server advertises that latch.

- Whenever requested to establish any configuration that does not require both TLS and server certificate verification to talk to a server or account, an MUA SHOULD warn its user that his or her mail traffic (including password, if applicable) will be exposed to attackers, and give the user an opportunity to abort the connection prior to transmission of any such password or traffic.

- MUAs SHOULD implement the "tls12" security latch (the TLS library has to provide an API that controls permissible TLS versions and communicates the negotiated TLS protocol version to the application for this to be possible).

- See Section 3 for additional requirements.

9.4. Non-configurable MUAs and nonstandard access protocols

MUAs which are not configurable to use user-specified servers MUST implement TLS or similarly other strong encryption mechanism when communicating with their mail servers. This generally applies to MUAs that are pre-configured to operate with one or more specific services, whether or not supplied by the vendor of those services.

MUAs using protocols other than IMAP, POP, and Submission to communicate with mail servers, MUST implement TLS or other similarly robust encryption mechanism in conjunction with those protocols.

9.5. DEEP Compliance for Anti-Virus/Anti-Spam Software and Services

There are multiple ways to connect an Anti-Virus and/or Anti-Spam (AVAS) service to a mail server. Some mechanisms, such as the de-facto milter protocol do not impact DEEP. However, some services use an SMTP relay proxy that intercepts mail at the application layer to perform a scan and proxy to the real MTA. Deploying AVAS services in this way can cause many problems [RFC2979] including direct interference with DEEP and privacy reduction. An AVAS product or service is considered DEEP compliant if all IMAP, POP and SMTP-related software it includes is DEEP compliant and it advertises all security latches that the actual MTA advertises.

10. Mail Service Provider Requirements

This section details requirements for providers of IMAP, POP, and/or SMTP submission services, for providers who claim to conform to this specification.

10.1. Server Requirements

Mail Service Providers MUST use server implementations that conform to this specification.

10.2. MSPs MUST provide Submission Servers

This document updates the advice in [RFC5068] by making Implicit TLS on port 465 the preferred submission port.

Mail Service Providers that accept mail submissions from end-users using the Internet Protocol MUST provide one or more SMTP Submission servers for this purpose, separate from the SMTP servers used to process incoming mail. Those submission servers MUST be configured to support Implicit TLS on port 465 and SHOULD support STARTTLS if port 587 is used.

MSPs MAY also support submission of messages via one or more designated SMTP servers to facilitate compatibility with legacy MUAs.

Discussion: SMTP servers used to accept incoming mail or to relay mail are expected to accept mail in cleartext. This is incompatible with the purpose of this memo which is to encourage encryption of traffic between mail servers. There is no such requirement for mail submission servers to accept mail in cleartext or without authentication. For other reasons, use of separate SMTP submission servers has been best practice for many years.

10.3. TLS Server Certificate Requirements

MSPs MUST maintain valid server certificates for all servers. Those server certificates SHOULD present DNS-IDs and SRV-IDs conforming to [RFC6125] and which will be recognized by MUAs meeting the requirements of that specification. In addition, those server certificates MAY provide other DNS-IDs, SRV-IDs, or CN-IDs needed for compatibility with existing MUAs.

If a protocol server provides service for more than one mail domain, it MAY use a separate IP address for each domain and/or a server certificates that advertises multiple domains. This will generally be necessary unless and until it is acceptable to impose the constraint that the server and all clients support the Server Name Indication extension to TLS [RFC6066].

10.4. Recommended DNS records for mail protocol servers

This section discusses not only the DNS records that are recommended, but also implications of DNS records for server configuration and TLS server certificates.

10.4.1. MX records

It is recommended that MSPs advertise MX records for handling of inbound mail (instead of relying entirely on A or AAAA records), and that those MX records be signed using DNSSEC. This is mentioned here only for completeness, as handling of inbound mail is out of scope for this document.

10.4.2. SRV records

MSPs SHOULD advertise SRV records to aid MUAs in determination of proper configuration of servers, per the instructions in [RFC6186].

MSPs SHOULD advertise servers that support Implicit TLS in preference to those which support cleartext and/or STARTTLS operation.

10.4.3. TLSA records

MSPs SHOULD advertise TLSA records to provide an additional trust anchor for public keys used in TLS server certificates. However, TLSA records MUST NOT be advertised unless they are signed using DNSSEC.

10.4.4. DNSSEC

All DNS records advertised by an MSP as a means of aiding clients in communicating with the MSP's servers, SHOULD be signed using DNSSEC.

10.5. MSP Server Monitoring

MSPs SHOULD regularly and frequently monitor their various servers to make sure that: TLS server certificates remain valid and are not about to expire, TLSA records match the public keys advertised in server certificates, are signed using DNSSEC, server configurations are consistent with SRV advertisements, and DNSSEC signatures are valid and verifiable. Failure to detect expired certificates and DNS configuration errors in a timely fashion can result in significant loss of service for an MSP's users and a significant support burden for the MSP.

10.6. Advertisement of DEEP status

MSPs SHOULD advertise a DEEP status that includes tls10, tls-cert and an HTTPS URL that can be used to inform clients of service outages or problems impacting client privacy. Note that advertising tls-cert is a commitment to maintain and renew server certificates.

10.7. Require TLS

New servers and services SHOULD be configured to require TLS unless it's necessary to support legacy clients or existing client configurations.

11. IANA Considerations

11.1. Security Tag Registry

IANA shall create (has created) the registry "Email Security Tags". This registry is a single table and will use an expert review process [RFC5226]. Each registration will contain the following fields:

Name: The name of the security tag. This follows the security-tag ABNF.

Description: This describes the meaning of the security tag and the conditions under which the tag is latched.

Intended Usage: One of COMMON, LIMITED USE or OBSOLETE.

Reference: Optional reference to specification.

Submitter: The identify of the submitter or submitters.

Change Controller: The identity of the change controller for the registration. This will be "IESG" in case of registrations in IETF-produced documents.

The expert reviewer will verify the tag name follows the ABNF, and that the description field is clear, unambiguous, does not overlap existing deployed technology, does not create security or privacy problems and appropriately considers interoperability issues. Email security tags intended for LIMITED USE have a lower review bar (interoperability and overlap issues are less of a concern). The reviewer may approve a registration, reject for a stated reason or recommend the proposal have standards track review due to importance or difficult subtleties.

Standards-track registrations may be updated if the relevant standards are updated as a consequence of that action. Non-standards-track entries may be updated by the listed change controller. The entry’s name and submitter may not be changed. In exceptional cases, any aspect of any registered entity may be updated at the direction of the IESG (for example, to correct a conflict).

11.2. Initial Set of Security Tags

This document defines four initial security tags for the security tag registry as follows:

Name: tls10

Description: This indicates TLS version 1.0 [RFC2246] or later was negotiated successfully including negotiation of a strong encryption layer with a symmetric key of at least 128 bits. This tag does not indicate the server certificate was valid. This tag is latched if the client sees this tag in the advertised server DEEP status provided after successfully negotiating TLS version 1.0 or later.

Intended Usage: COMMON

Reference: RFC XXXX (this document once published)

Submitter: Authors of this document

Change Controller: IESG

Name: tls12

Description: This indicates TLS version 1.2 [RFC5246] or later was negotiated successfully including negotiation of a strong encryption layer with a symmetric key of at least 128 bits. This tag does not indicate the server certificate was valid. This tag is latched if the client sees this tag in the advertised server DEEP status provided after successfully negotiating TLS version 1.2 or later.

Intended Usage: COMMON

Reference: RFC XXXX (this document once published)

Submitter: Authors of this document

Change Controller: IESG

Name: tls-cert

Description: This tag indicates that TLS was successfully negotiated and the server certificate was successfully verified by the client using PKIX [RFC5280] and the server certificate identity was verified using the algorithm appropriate for the protocol (see Section 4). This tag is latched if the client sees this tag in the advertised server DEEP status after successfully negotiating TLS and verifying the certificate and server identity.

Intended Usage: COMMON

Reference: RFC XXXX (this document once published)

Submitter: Authors of this document

Change Controller: IESG

Name: tls-dane-tlsa

Description: This tag indicates that TLS was successfully negotiated and the server certificate was successfully verified by the client using the procedures described in [RFC6698] and the server certificate identity was verified using the algorithm appropriate for the protocol (see Section 4). This tag is latched if the client sees this tag in the advertised server DEEP status after successfully negotiating TLS and verifying the certificate and server identity.

Intended Usage: COMMON

Reference: RFC XXXX (this document once published)

Submitter: Authors of this document

Change Controller: IESG

11.3. POP3S Port Registration Update

IANA is asked to update the registration of the TCP well-known port 995 using the following template ([RFC6335]):

```
Service Name: pop3s
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: POP3 over TLS protocol
Reference: RFC XXXX (this document once published)
```

```
Service Name: pop3s
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: POP3 over TLS protocol
Reference: RFC XXXX (this document once published)
```

11.4. IMAPS Port Registration Update

IANA is asked to update the registration of the TCP well-known port 993 using the following template ([RFC6335]):

```
Service Name: imaps
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: IMAP over TLS protocol
```

Reference: RFC XXXX (this document once published)

Service Name: imaps
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: IMAP over TLS protocol
Reference: RFC XXXX (this document once published)

11.5. Submissions Port Registration

IANA is asked to assign an alternate usage of port 465 in addition to the current assignment using the following template ([RFC6335]):

Service Name: submissions
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: Message Submission over TLS protocol
Reference: RFC XXXX (this document once published)

Service Name: submissions
Transport Protocol: TCP
Assignee: IETF <iesg@ietf.org>
Contact: IESG <iesg@ietf.org>
Description: Message Submission over TLS protocol
Reference: RFC XXXX (this document once published)

This is a one time procedural exception to the rules in RFC 6335. This requires explicit IESG approval and does not set a precedent. Historically, port 465 was briefly registered as the "smtps" port. This registration made no sense as the SMTP transport MX infrastructure has no way to specify a port so port 25 is always used. As a result, the registration was revoked and was subsequently reassigned to a different service. In hindsight, the "smtps" registration should have been renamed or reserved rather than revoked. Unfortunately, some widely deployed mail software interpreted "smtps" as "submissions" [RFC6409] and used that port for email submission by default when an end-user requests security during account setup. If a new port is assigned for the submissions service, email software will either continue with unregistered use of port 465 (leaving the port registry inaccurate relative to de-facto practice and wasting a well-known port), or confusion between the de-facto and registered ports will cause harmful interoperability problems that will deter use of TLS for message submission. The authors believe both of these outcomes are less desirable than a wart in the registry documenting real-world usage of a port for two purposes. Although STARTTLS-on-port-587 has deployed, it has not

replaced deployed use of implicit TLS submission on port 465.

11.6. DEEP IMAP Capability

This document adds the DEEP capability to the IMAP capabilities registry. This is described in Section 7.1.

11.7. DEEP POP3 Capability

This document adds the DEEP capability to the POP3 capabilities registry.

CAPA Tag: DEEP

Arguments: deep-status

Added Commands: none

Standard Commands affected: none

Announced status / possible differences: both / may change after STLS

Commands Valid in States: N/A

Specification Reference: This document

Discussion: See Section 7.2.

11.8. DEEP SMTP EHLO Keyword

This document adds the DEEP EHLO Keyword to the SMTP Service Extension registry. This is described in Section 7.3.

11.9. SMTP Enhanced Status Code

This document adds the following entry to the "SMTP Enhanced Status Codes" registry created by [RFC5248].

Code: X.7.TBD (IANA, please assign the next available number)

Sample Text: Message Transport Failed due to missing required security.

Associated Basic Status Code: 450, 454, 550, 554

Description This code indicates an SMTP server was unable to forward a message to the next host necessary for delivery because it required a higher level of transport security or privacy than was available. The temporary form of this error is preferred in case the problem is caused by a temporary administrative error such as an expired server certificate.

Reference This document

Submitter C. Newman

Change Controller IESG

11.10. MAIL Parameters Additional-registered-clauses Sub-Registry

This document adds the following entry to the "Additional-registered-clauses" sub-registry of the "MAIL Parameters" registry, created by [RFC5321]:

Clause Name: tls

Description: Indicates the TLS cipher suite used for a transport connection.

Syntax Summary: See tls-cipher ABNF Section 6

Reference: This document.

12. Security Considerations

This entire document is about security considerations. In general, this is targeted to improve mail privacy and to mitigate threats external to the email system such as network-level snooping or interception; this is not intended to mitigate active attackers who have compromised service provider systems.

It could be argued that sharing the name and version of the client software with the server has privacy implications. Although providing this information is not required, it is encouraged so that mail service providers can more effectively inform end-users running old clients that they need to upgrade to protect their privacy, or know which clients to use in a test deployment prior to upgrading a server to have higher security requirements.

13. References

13.1. Normative References

[RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.

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

[RFC2449] Gellens, R., Newman, C., and L. Lundblade, "POP3 Extension Mechanism", RFC 2449, November 1998.

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

[RFC2971] Showalter, T., "IMAP4 ID extension", RFC 2971, October 2000.

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, February 2002.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC3464] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, January 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5034] Siemborski, R. and A. Menon-Sen, "The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism", RFC 5034, July 2007.

[RFC5068] Hutzler, C., Crocker, D., Resnick, P., Allman, E., and T. Finch, "Email Submission Operations: Access and Accountability Requirements", BCP 134, RFC 5068, November 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

[RFC5248] Hansen, T. and J. Klensin, "A Registry for SMTP Enhanced Mail System Status Codes", BCP 138, RFC 5248, June 2008.

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

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[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.

[RFC6186] Daboo, C., "Use of SRV Records for Locating Email Submission/Access Services", RFC 6186, March 2011.

[RFC6409] Gellens, R. and J. Klensin, "Message Submission for Mail", STD 72, RFC 6409, November 2011.

[I-D.melnikov-email-tls-certs] Melnikov, A., "Updated TLS Server Identity Check Procedure for Email Related Protocols", draft-melnikov-email-tls-certs-01 (work in progress), October 2013.

[I-D.ietf-dane-smtp-with-dane] Dukhovni, V. and W. Hardaker, "SMTP security via opportunistic DANE TLS", draft-ietf-dane-smtp-with-dane-02 (work in progress), October 2013.

13.2. Informative References

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC2595] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999.

[RFC2979] Freed, N., "Behavior of and Requirements for Internet Firewalls", RFC 2979, October 2000.

[RFC3848] Newman, C., "ESMTP and LMTP Transmission Types Registration", RFC 3848, July 2004.

[RFC3887] Hansen, T., "Message Tracking Query Protocol", RFC 3887, September 2004.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4954] Siemborski, R. and A. Melnikov, "SMTP Service Extension for Authentication", RFC 4954, July 2007.

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, July 2009.

[RFC5802] Newman, C., Menon-Sen, A., Melnikov, A., and N. Williams, "Salted Challenge Response Authentication Mechanism (SCRAM) SASL and GSS-API Mechanisms", RFC 5802, July 2010.

[RFC5804] Melnikov, A. and T. Martin, "A Protocol for Remotely Managing Sieve Scripts", RFC 5804, July 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 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.

[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.

Appendix A. Design Considerations

This section is not normative.

The first version of this was written independently from draft-moore-email-tls-00.txt; subsequent versions merge ideas from both drafts.

One author of this document was also the author of RFC 2595 that

became the standard for TLS usage with POP and IMAP, and the other author was perhaps the first to propose that idea. In hindsight both authors now believe that that approach was a mistake. At this point the authors believe that while anything that makes it easier to deploy TLS is good, the desirable end state is that these protocols always use TLS, leaving no need for a separate port for cleartext operation except to support legacy clients while they continue to be used. The separate port model for TLS is inherently simpler to implement, debug and deploy. It also enables a "generic TLS load-balancer" that accepts secure client connections for arbitrary foo-over-TLS protocols and forwards them to a server that may or may not support TLS. Such load-balancers cause many problems because they violate the end-to-end principle and the server loses the ability to log security-relevant information about the client unless the protocol is designed to forward that information (as this specification does for the cipher suite). However, they can result in TLS deployment where it would not otherwise happen which is a sufficiently important goal that it overrides the problems.

Although STARTTLS appears only slightly more complex than separate-port TLS, we again learned the lesson that complexity is the enemy of security in the form of the STARTTLS command injection vulnerability (CERT vulnerability ID #555316). Although there’s nothing inherently wrong with STARTTLS, the fact it resulted in a common implementation error (made independently by multiple implementers) suggests it is a less secure architecture than Implicit TLS.

Section 7 of RFC 2595 critiques the separate-port approach to TLS. The first bullet was a correct critique. There are proposals in the http community to address that, and use of SRV records as described in RFC 6186 resolves that critique for email. The second bullet is correct as well, but not very important because useful deployment of security layers other than TLS in email is small enough to be effectively irrelevant. The third bullet is incorrect because it misses the desirable option of "use and latch-on TLS if available". The fourth bullet may be correct, but is not a problem yet with current port consumption rates. The fundamental error was prioritizing a perceived better design based on a mostly valid critique over real-world deployability. But getting security and privacy facilities actually deployed is so important it should trump design purity considerations.

Appendix B. Open Issues

There are many open issues with this document. Here is an attempt to enumerate some of them:

- Port 465 is presently used for two purposes: for submissions by a large number of clients and service providers and for the "urd" protocol by one vendor. Actually documenting this current state is controversial as discussed in the IANA considerations section. However, there is no good alternative. Registering a new port for submissions when port 465 is widely used for that purpose already will just create interoperability problems. Registering a port that's only used if advertised by an SRV record (RFC 6186) would not create interoperability problems but would require all client and server deployments and software to change significantly which is contrary to the goal of promoting more TLS use. Encouraging use of STARTTLS on port 587 would not create interoperability problems, but is unlikely to have impact on current undocumented use of port 465 and makes the guidance in this document less consistent.

- Discussion of pinning certificates is new and may be inadequate. Suggestions to improve the text are welcome.

- This document should reference draft-ietf-uta-tls-bcp and possibly other guidance documents. Suggested text on where/how to reference this and possibly other TLS guidance (e.g., must staple). would be welcome.

- One author believes that the security latch model is complementary with draft-ietf-dane-smtp-with-dane-02 but hasn't thought about the issues in depth. We welcome feedback on this point.

- The three involved authors are willing to merge draft-melnikov-email-tls-certs into this document. However, this will take time so we are only willing to do so if there is rough consensus on the decision (so it's a one time action) and doing so will not significantly delay publication.

- It might make sense to split this in two or more documents if it's getting too long to evaluate in one IETF last call. In particular, it might make sense to put implementation requirements and service provider requirements in separate documents. The authors prefer to edit one document for now and defer discussion of splitting the document until all technical issues are resolved.

- The use of SRV records [RFC6186] for account setup or refresh is presently not secure from DNS active attacks unless DNSSEC is used. As this document is now focusing on MUA security/privacy, discussing how to do SRV record account setup or account refresh securely, probably using DANE, would be in scope for this document. It has been suggested that we add this.

- This document does not cover use of TLS with SMTP relay.

Appendix C. Change Log

Changes since -01:

- Updated abstract, introduction and document structure to focus more on mail user agent privacy assurance.
- Added email account privacy section, also moving section on account setup using SRV records to that section.
- Finished writing IANA considerations section
- Remove provisional concept and instead have server explicitly list security tags clients should latch.
- Added note that rules for the submissions port follow the same rules as those for the submit port.
- Reference and update advice in [RFC5068].
- Fixed typo in Client Certificate Authentication section.
- Removed tls-pfs security latch and all mention of perfect forward secrecy as it was controversial.
- Added reference to HSTS.

Changes since -00:

- Rewrote introduction to merge ideas from draft-moore-email-tls-00.
- Added Implicit TLS section, Account configuration section and IANA port registration updates based on draft-moore-email-tls-00.
- Add protocol details necessary to standardize implicit TLS for POP/IMAP/submission, using ideas from draft-melnikov-pop3-over-tls.
- Reduce initial set of security tags based on feedback.
- Add deep status concept to allow a window for software updates to be backed out before latches make that problematic, as well as to provide service providers with a mechanism they can use to assist customers in the event of a privacy failure.

- Add DNS SRV section from draft-moore-email-tls-00.
- Write most of the missing IANA considerations section.
- Rewrite most of implementation requirements section based more on draft-moore-email-tls-00. Remove new cipher requirements for now because those may be dealt with elsewhere.

Appendix D. Acknowledgements

Many thanks to Ned Freed for discussion of the initial latch concepts in this document. Thanks to Alexey Melnikov for draft-melnikov-pop3-over-tls-02, which was the basis of the POP3 implicit TLS text. Thanks to Dan Newman and Alexey Melnikov for review feedback. Thanks to Paul Hoffman for interesting feedback in initial conversations about this idea.

Authors' Addresses

Keith Moore
Network Heretics
PO Box 1934
Knoxville, TN 37901
US

Email: moore@network-heretics.com

Chris Newman
Oracle
440 E. Huntington Dr., Suite 400
Arcadia, CA 91006
US

Email: chris.newman@oracle.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 16, 2015

A. Popov, Ed.
M. Nystroem
Microsoft Corp.
D. Balfanz
A. Langley
Google Inc.

October 13, 2014

The Token Binding Protocol Version 1.0
draft-popov-token-binding-00

Abstract

This document specifies Version 1.0 of the Token Binding protocol. The Token Binding protocol allows client/server applications to create long-lived, uniquely identifiable TLS [RFC5246] bindings spanning multiple TLS sessions and connections. Applications are then enabled to cryptographically bind security tokens to the TLS layer, preventing token export and replay attacks. To protect privacy, the TLS Token Binding identifiers are only transmitted encrypted and can be reset by the user at any time.

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 16, 2015.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Requirements Language 3
2. Token Binding Protocol Overview 3
3. Negotiating the Token Binding Protocol and Key Parameters . . 4
4. Token Binding Protocol Message 5
5. Establishing a TLS Token Binding 7
6. TLS Token Binding ID Format 7
7. Security Token Validation 8
8. IANA Considerations 9
9. Security Considerations 11
9.1. Security Token Replay 11
9.2. Downgrade Attacks 11
9.3. Privacy Considerations 11
9.4. Triple Handshake Vulnerability in TLS 11
10. References 12
10.1. Normative References 12
10.2. Informative References 12
Authors' Addresses 13

1. Introduction

Web services generate various security tokens (e.g. HTTP cookies, OAuth tokens) for web applications to access protected resources. Any party in possession of such token gains access to the protected resource. Attackers export bearer tokens from the user's machine, present them to web services, and impersonate authenticated users. The idea of Token Binding is to prevent such attacks by cryptographically binding security tokens to the TLS layer.

A TLS Token Binding is established by the user agent generating a private-public key pair (possibly within a secure hardware module, such as TPM) per target server, and proving possession of the private key on every TLS connection to the target server. The proof of possession involves signing the tls_unique value [RFC5929] for the TLS connection with the private key. Such TLS Token Binding is identified by the corresponding public key. TLS Token Bindings are long-lived, i.e. they encompass multiple TLS connections and TLS sessions between a given client and server. To protect privacy, TLS

Token Binding identifiers are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies.

When issuing a security token to a client that supports TLS Token Binding, a server includes the client's TLS Token Binding ID in the token. Later on, when a client presents a security token containing a TLS Token Binding ID, the server makes sure the ID in the token matches the ID of the TLS Token Binding established with the client. In the case of a mismatch, the server discards the token.

In order to successfully export and replay a bound security token, the attacker needs to also be able to export the client's private key, which is hard to do in the case of the key generated in a secure hardware module.

1.1. Requirements Language

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. Token Binding Protocol Overview

The client and server use ALPN protocol IDs [RFC7301] to negotiate the use of the Token Binding protocol, in addition to the actual application protocol such as HTTP/1.1 [RFC2616] or HTTP/2 [I-D.ietf-httpbis-http2]. ALPN IDs are also used to negotiate the parameters (signature algorithm, length) of the Token Binding key. This negotiation does not require TLS protocol changes or additional round-trips.

The "IANA Considerations" section of this document defines an initial set of ALPN protocol IDs that allow the use of the Token Binding protocol with HTTP/1.1 and HTTP/2. The initial set of supported key parameters includes ECDSA with NIST P256 curve and 2048-bit RSA. New ALPN protocol IDs can be defined in the future to support Token Binding usage with other application protocols and key parameters.

The Token Binding protocol consists of one message sent by the client to the server, proving possession of one or more client-generated asymmetric keys. This message is only sent if the client and server agree on the use of the Token Binding protocol and the key parameters. The Token Binding message is sent with the application protocol data in TLS application_data records.

A server receiving the Token Binding message verifies that the key parameters in the message match the Token Binding parameters negotiated via ALPN, and then validates the signatures contained in

the Token Binding message. If either of these checks fails, the server terminates the connection, otherwise the TLS Token Binding is successfully established with the ID contained in the Token Binding message.

When a server supporting the Token Binding protocol receives a bound token, the server compares the TLS Token Binding ID in the security token with the TLS Token Binding ID established with the client. If the bound token came from a TLS connection without a Token Binding, or if the IDs don't match, the token is discarded.

This document describes the negotiation of the Token Binding protocol and key parameters, the format of the Token Binding protocol message, the process of establishing a TLS Token Binding, the format of the Token Binding ID, and the process of validating a security token. Token Binding over HTTP [HTTPSTB] explains how the Token Binding message is encapsulated within application protocol messages. [HTTPSTB] also describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party.

3. Negotiating the Token Binding Protocol and Key Parameters

The Token Binding protocol is used within TLS connections, in combination with an application protocol such as HTTP/1.1 or HTTP/2. The "IANA Considerations" section of this document defines a set of ALPN protocol IDs that combine application protocol and token binding key parameters:

- "h2_tb_p256" indicates support for HTTP/2 with Token Binding using ECDSA key and NIST P256 curve;
- "h2_tb_rsa2048" indicates support for HTTP/2 with Token Binding using 2048-bit RSA key;
- "http/1.1_tb_p256" indicates support for HTTP/1.1 with Token Binding using ECDSA key and NIST P256 curve;
- "http/1.1_tb_rsa2048" indicates support for HTTP/1.1 with Token Binding using 2048-bit RSA key.

The client advertises support of the Token Binding protocol by sending some of these IDs in the ALPN extension in the ClientHello. Application protocol IDs without Token Binding, such as "http/1.1" and "h2", can also be included for compatibility with the servers that do not support the Token Binding protocol.

The server indicates support of the Token Binding protocol by sending one of the above IDs in the ALPN extension in the ServerHello. The

server implements the protocol selection logic as described in section 3.2 "Protocol Selection" of [RFC7301], taking into account the application protocols and key parameters supported by the server.

4. Token Binding Protocol Message

The Token Binding message is sent by the client and proves possession of one or more private keys held by the client. This message MUST be sent if the client and server successfully negotiated the use of the Token Binding protocol via ALPN, and MUST NOT be sent otherwise. This message MUST be sent in the client's first application protocol message. This message MAY also be sent in subsequent application protocol messages, proving possession of other keys by the same client, to facilitate token binding between more than two communicating parties. Token Binding over HTTP [HTTPSTB] specifies the encapsulation of the Token Binding message in the application protocol messages, and the scenarios involving more than two communicating parties. The Token Binding message format is defined using TLS specification language, and reuses existing TLS structures and IANA registrations where possible:

```
enum {
    sha256(4), (255)
} HashAlgorithm;

enum {
    rsa(1), ecdsap256(3), (255)
} SignatureAlgorithm;

struct {
    HashAlgorithm hash;
    SignatureAlgorithm signature;
} SignatureAndHashAlgorithm;

struct {
    opaque modulus<1..2^16-1>;
    opaque publicexponent<1..2^8-1>;
} RSAPublicKey;

enum {
    secp256r1 (23), (0xFFFF)
} NamedCurve;

struct {
    opaque point <1..2^8-1>;
} ECPoint;
```

```
struct {
    NamedCurve namedcurve;
    ECPoint point;      // Uncompressed format
} ECDSAParams;

enum {
    provided_token_binding(0), referred_token_binding(1), (255)
} TokenBindingType;

struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;

enum {
    (255)           // No initial ExtensionType registrations
} ExtensionType;

struct {
    ExtensionType extension_type;
    opaque extension_data<0..2^16-1>;
} Extension;

struct {
    TokenBindingID tokenbindingid;
    opaque signature<0..2^16-1>;// Signature over hashed ("token binding", tls_u
nique)
    Extension extensions<0..2^16-1>;
} TokenBinding;

struct {
    TokenBinding tokenbindings<0..2^16-1>;
} TokenBindingMessage;
```

The Token Binding message consists of a series of TokenBinding structures containing the TokenBindingID, a signature over the hash of the NUL-terminated, ASCII label ("token binding") and the tls_unique, optionally followed by Extension structures. An implementation MUST ignore any unknown extensions. Initially, no extension types are defined. At least one TokenBinding MUST be included in the Token Binding message. The signature algorithm and key length used in the TokenBinding MUST match the parameters negotiated via ALPN. The client SHOULD generate and store Token Binding keys in a secure manner that prevents key export. In order

to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

5. Establishing a TLS Token Binding

Triple handshake vulnerability in the TLS protocol affects the security of the Token Binding protocol, as described in the "Security Considerations" section below. Therefore, the server MUST NOT negotiate the use of the Token Binding protocol unless the server also negotiates Extended Master Secret TLS extension [I-D.ietf-tls-session-hash].

The server MUST terminate the connection if the use of the Token Binding protocol has been successfully negotiated via ALPN within the TLS handshake, but the client's first application message does not contain the Token Binding message. The server MUST terminate the connection if the use of the Token Binding protocol was not negotiated, but the client sends the Token Binding message.

If the Token Binding type is "provided_token_binding", the server MUST verify that the signature algorithm (including elliptic curve in the case of ECDSA) and key length in the Token Binding message match those negotiated via ALPN. In the case of a mismatch, the server MUST terminate the connection. As described in [HTTPSTB], Token Bindings of type "referred_token_binding" may have different key parameters than those negotiated via ALPN.

If the Token Binding message does not contain at least one TokenBinding structure, or the signature contained in a TokenBinding structure is invalid, the server MUST terminate the connection. Otherwise, the TLS Token Binding is successfully established and its ID can be provided to the application for security token validation.

6. TLS Token Binding ID Format

The ID of the TLS Token Binding established as a result of Token Binding message processing is a binary representation of the following structure:

```
struct {
    TokenBindingType tokenbinding_type;
    SignatureAndHashAlgorithm algorithm;
    select (algorithm.signature) {
        case rsa: RSAPublicKey rsapubkey;
        case ecdsa: ECDSAParams ecdsaparams;
    }
} TokenBindingID;
```

TokenBindingID includes the type of the token binding and the key parameters negotiated via ALPN. This document defines two token binding types: provided_token_binding used to establish a Token Binding when connecting to a server, and referred_token_binding used when requesting tokens to be presented to a different server. Token Binding over HTTP [HTTPSTB] describes Token Binding between multiple communicating parties: User Agent, Identity Provider and Relying Party. TLS Token Binding ID can be obtained from the TokenBinding structure described in the "Token Binding Protocol Message" section of this document by discarding the signature and extensions. TLS Token Binding ID will be available at the application layer and used by the server to generate and verify bound tokens.

7. Security Token Validation

Security tokens can be bound to the TLS layer either by embedding the Token Binding ID in the token, or by maintaining a database mapping tokens to Token Binding IDs. The specific method of generating bound security tokens is application-defined and beyond the scope of this document.

Upon receipt of a security token, the server attempts to retrieve TLS Token Binding ID information from the token and from the TLS connection with the client. Application-provided policy determines whether to honor non-bound (bearer) tokens. If the token is bound and a TLS Token Binding has not been established for the client connection, the server MUST discard the token. If the TLS Token Binding ID for the token does not match the TLS Token Binding ID established for the client connection, the server MUST discard the token.

8. IANA Considerations

This document establishes a registry for Token Binding type identifiers entitled "Token Binding Types" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- Value: The octet value that identifies the Token Binding type (0-255).
- Description: The description of the Token Binding type.
- Specification: A reference to a specification that defines the Token Binding type.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding type.

An initial set of registrations for this registry follows:

Value: 0

Description: provided_token_binding

Specification: this document

Value: 1

Description: referred_token_binding

Specification: this document

This document establishes a registry for Token Binding extensions entitled "Token Binding Extensions" under the "Token Binding Protocol" heading.

Entries in this registry require the following fields:

- Value: The octet value that identifies the Token Binding extension (0-255).
- Description: The description of the Token Binding extension.

- Specification: A reference to a specification that defines the Token Binding extension.

This registry operates under the "Expert Review" policy as defined in [RFC5226]. The designated expert is advised to encourage the inclusion of a reference to a permanent and readily available specification that enables the creation of interoperable implementations using the identified Token Binding extension. This document creates no initial registrations in the "Token Binding Extensions" registry.

This document creates the following registrations for the identification of the Token Binding protocol in the "Application Layer Protocol Negotiation (ALPN) Protocol IDs" registry originally created in [RFC7301]:

Protocol: HTTP/2 with Token Binding using ECDSA key and NIST P256 curve

Identification Sequence: 0x68 0x32 0x5f 0x74 0x62 0x5f 0x70 0x32 0x35 0x36 ("h2_tb_p256")

Specification: this document

Protocol: HTTP/2 with Token Binding using 2048-bit RSA key

Identification Sequence: 0x68 0x32 0x5f 0x74 0x62 0x5f 0x72 0x73 0x61 0x32 0x30 0x34 0x38 ("h2_tb_rsa2048")

Specification: this document

Protocol: HTTP/1.1 with Token Binding using ECDSA key and NIST P256 curve

Identification Sequence: 0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 0x5f 0x74 0x62 0x5f 0x70 0x32 0x35 0x36 ("http/1.1_tb_p256")

Specification: this document

Protocol: HTTP/1.1 with Token Binding using 2048-bit RSA key

Identification Sequence: 0x68 0x74 0x74 0x70 0x2f 0x31 0x2e 0x31 0x5f 0x74 0x62 0x5f 0x72 0x73 0x61 0x32 0x30 0x34 0x38 ("http/1.1_tb_rsa2048")

Specification: this document

This document uses "TLS SignatureAlgorithm" and "TLS HashAlgorithm" registries originally created in [RFC5246], and "TLS NamedCurve" registry originally created in [RFC4492]. This document creates no new registrations in these registries.

9. Security Considerations

9.1. Security Token Replay

The goal of the Token Binding protocol is to prevent attackers from exporting and replaying security tokens, thereby impersonating legitimate users and gaining access to protected resources. Bound tokens can still be replayed by the malware present in the User Agent. In order to export the token to another machine and successfully replay it, the attacker also needs to export the corresponding private key. Token Binding private keys are therefore high-value assets and SHOULD be strongly protected, ideally by generating them in a hardware security module that prevents key export.

9.2. Downgrade Attacks

The Token Binding protocol is only used when negotiated via ALPN within the TLS handshake. TLS prevents active attackers from modifying the messages of the TLS handshake, therefore it is not possible for the attacker to remove or modify the ALPN IDs used to negotiate the Token Binding protocol and key parameters. The signature algorithm and key length used in the TokenBinding of type "provided_token_binding" MUST match the parameters negotiated via ALPN.

9.3. Privacy Considerations

The Token Binding protocol uses persistent, long-lived TLS Token Binding IDs. To protect privacy, TLS Token Binding IDs are never transmitted in clear text and can be reset by the user at any time, e.g. when clearing browser cookies. In order to prevent cooperating servers from linking user identities, different keys SHOULD be used by the client for connections to different servers, according to the token scoping rules of the application protocol.

9.4. Triple Handshake Vulnerability in TLS

The Token Binding protocol relies on the tls_unique value to associate a TLS connection with a TLS Token Binding. The triple handshake attack [TRIPLE-HS] is a known TLS protocol vulnerability allowing the attacker to synchronize tls_unique values between TLS connections. The attacker can then successfully replay bound tokens.

For this reason, the Token Binding protocol MUST NOT be negotiated unless the Extended Master Secret TLS extension [I-D.ietf-tls-session-hash] has also been negotiated.

10. References

10.1. Normative References

[HTTPSTB] Balfanz, D., Langley, A., Popov, A., and M. Nystroem, "Token Binding over HTTP", 2014.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, May 2006.

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

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

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, July 2014.

10.2. Informative References

[I-D.ietf-httpbis-http2] Belshe, M., Peon, R., and M. Thomson, "Hypertext Transfer Protocol version 2", draft-ietf-httpbis-http2-14 (work in progress), July 2014.

[I-D.ietf-tls-session-hash] Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", draft-ietf-tls-session-hash-02 (work in progress), October 2014.

[TRIPLE-HS]
Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Pironti, A., and P. Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS. IEEE Symposium on Security and Privacy", 2014.

Authors' Addresses

Andrei Popov (editor)
Microsoft Corp.
USA

Email: andreipo@microsoft.com

Magnus Nystroem
Microsoft Corp.
USA

Email: mnystrom@microsoft.com

Dirk Balfanz
Google Inc.
USA

Email: balfanz@google.com

Adam Langley
Google Inc.
USA

Email: agl@google.com