Internet-Draft
Updates: 4346 (once approved)
June 2006
Expires: December 2006

Transport Layer Security (TLS) Authorization Extensions
<draft-housley-tls-authz-extns-07.txt>

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Copyright Notice

Copyright (C) The Internet Society (2006). All Rights Reserved.

Abstract

This document specifies authorization extensions to the Transport Layer Security (TLS) Handshake Protocol. Extensions carried in the client and server hello messages to confirm that both parties support the desired authorization data types. Then, if supported by both the client and the server, authorization information is exchanged in the supplemental data handshake message.
1. Introduction

Transport Layer Security (TLS) protocol [TLS1.0][TLS1.1] is being used in an increasing variety of operational environments, including ones that were not envisioned at the time of the original design for TLS. The extensions introduced in this document are designed to enable TLS to operate in environments where authorization information needs to be exchanged between the client and the server before any protected data is exchanged.

The use of these TLS authorization extensions is especially attractive when more than one application protocol can make use of the same authorization information. Straightforward binding of identification, authentication, and authorization information is possible when all of these are handled within TLS. If each application requires unique authorization information, then it might best be carried within the TLS-protected application protocol. However, care must be taken to ensure appropriate bindings when identification, authentication, and authorization information are handled at different protocol layers.

This document describes authorization extensions for the TLS Handshake Protocol in both TLS 1.0 and TLS 1.1. These extensions observe the conventions defined for TLS Extensions [TLSEXT] that make use of the general extension mechanisms for the client hello message and the server hello message. The extensions described in this document confirm that both the client and the server support the desired authorization data types. Then, if supported, authorization information is exchanged in the supplemental data handshake message [TLSSUPP].

The authorization extensions may be used in conjunction with TLS 1.0 and TLS 1.1. The extensions are designed to be backwards compatible, meaning that the Handshake Protocol Supplemental Data messages will only contain authorization information of a particular type if the client indicates support for them in the client hello message and the server indicates support for them in the server hello message.

Clients typically know the context of the TLS session that is being setup, thus the client can use the authorization extensions when they are needed. Servers must accept extended client hello messages, even if the server does not "understand" the all of the listed extensions. However, the server will not indicate support for these "not understood" extensions. Then, clients may reject communications with servers that do not support the authorization extensions.
1.1. Conventions

The syntax for the authorization messages is defined using the TLS Presentation Language, which is specified in Section 4 of [TLS1.0].

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 [STDWORDS].

1.2. Overview

Figure 1 illustrates the placement of the authorization extensions and supplemental data messages in the full TLS handshake.

```
Client                                                   Server
ClientHello (w/ extensions) --------> ServerHello (w/ extensions)
                             SupplementalData*                      Certificate*
                             Certificate*                               ServerKeyExchange*
                             ServerKeyExchange*                          CertificateRequest*
                                                     <-------- ServerHelloDone
SupplementalData*
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                    -------->                      [ChangeCipherSpec]
                             Finished
Application Data            <--------                  Application Data
```

* Indicates optional or situation-dependent messages that are not always sent.

[] Indicates that ChangeCipherSpec is an independent TLS Protocol content type; it is not actually a TLS handshake message.

Figure 1. Authorization data exchange in full TLS handshake

The ClientHello message includes an indication of the client authorization data formats that are supported and an indication of the server authorization data formats that are supported. The
ServerHello message contains similar indications, but any authorization data formats that are not supported by the server are not included. Both the client and the server MUST indicate support for the authorization data types. If the list of mutually supported authorization data formats is empty, then the ServerHello message MUST NOT carry the affected extension at all.

2. Authorization Extension Types

The general extension mechanisms enable clients and servers to negotiate whether to use specific extensions, and how to use specific extensions. As specified in [TLSEXT], the extension format used in the extended client hello message and extended server hello message is repeated here for convenience:

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

The extension_type identifies a particular extension type, and the extension_data contains information specific to the particular extension type.

As specified in [TLSEXT], for all extension types, the extension type MUST NOT appear in the extended server hello message unless the same extension type appeared in the corresponding client hello message. Clients MUST abort the handshake if they receive an extension type in the extended server hello message that they did not request in the associated extended client hello message.

When multiple extensions of different types are present in the extended client hello message or the extended server hello message, the extensions can appear in any order, but there MUST NOT be more than one extension of the same type.

This document specifies the use of two new extension types: client_authz and server_authz. These extension types are described in Section 2.1 and Section 2.2, respectively. This specification adds two new types to ExtensionType:

```c
enum {
    client_authz(TBD), server_authz(TBD), (65535)
} ExtensionType;
```

The authorization extensions are relevant when a session is initiated and any subsequent session resumption. However, a client that requests resumption of a session does not know whether the server
will have all of the context necessary to accept this request, and
therefore the client SHOULD send an extended client hello message
that includes the extension types associated with the authorization
extensions. This way, if the resumption request is denied, then the
authorization extensions will be negotiated as normal.

2.1. The client_authz Extension Type

Clients MUST include the client_authz extension type in the extended
client hello message to indicate their desire to send authorization
data to the server. The extension_data field indicates the format of
the authorization data that will be sent in the supplemental data
handshake message. The syntax of the client_authz extension_data
field is described in Section 2.3.

Servers that receive an extended client hello message containing the
client_authz extension MUST respond with the same client_authz
extension in the extended server hello message if the server is
willing to receive authorization data in the indicated format. Any
unacceptable formats must be removed from the list provided by the
client. The client_authz extension MUST be omitted from the extended
server hello message if the server is not willing to receive
authorization data in any of the indicated formats.

2.2. The server_authz Extension Type

Clients MUST include the server_authz extension type in the extended
client hello message to indicate their desire to receive
authorization data from the server. The extension_data field
indicates the format of the authorization data that will be sent in
the supplemental data handshake message. The syntax of the
server_authz extension_data field as described in Section 2.3.

Servers that receive an extended client hello message containing the
server_authz extension MUST respond with the same server_authz
extension in the extended server hello message if the server is
willing to provide authorization data in the requested format. Any
unacceptable formats must be removed from the list provided by the
client. The server_authz extension MUST be omitted from the extended
server hello message if the server is not able to provide
authorization data in any of the indicated formats.
2.3. AuthzDataFormat Type

The AuthzDataFormat type is used in both the client_authz and the server_authz extensions. It indicates the format of the authorization data that will be transferred. The AuthzDataFormats type definition is:

```
enum {
    x509_attr_cert(0),
    saml_assertion(1),
    x509_attr_cert_url(2),
    saml_assertion_url(3),
    (255)
} AuthzDataFormat;
```

```
AuthzDataFormats authz_format_list<1..2^8-1>;
```

When the x509_attr_cert value is present, the authorization data is an X.509 Attribute Certificate (AC) that conforms to the profile in RFC 3281 [ATTRCERT].

When the saml_assertion value is present, the authorization data is an assertion composed using the Security Assertion Markup Language (SAML) [SAML1.1][SAML2.0].

When the x509_attr_cert_url value is present, the authorization data is an X.509 AC that conforms to the profile in RFC 3281 [ATTRCERT]; however, the AC is fetched with the supplied URL. A one-way hash value is provided to ensure that the intended AC is obtained.

When the saml_assertion_url value is present, the authorization data is a SAML Assertion; however, the SAML Assertion is fetched with the supplied URL. A one-way hash value is provided to ensure that the intended SAML Assertion is obtained.

3. Supplemental Data Handshake Message Usage

As shown in Figure 1, supplemental data can be exchanged in two places in the handshake protocol. The client_authz extension determines what authorization data formats are acceptable for transfer from the client to the server, and the server_authz extension determines what authorization data formats are acceptable for transfer from the server to the client. In both cases, the syntax specified in [TLSSUPP] is used along with the authz_data type defined in this document.
enum {
    authz_data(TBD), (65535)
} SupplementalDataType;

struct {
    SupplementalDataType supplemental_data_type;
    select(SupplementalDataType) {
        case authz_data: AuthorizationData;
    }
} SupplementalData;

3.1. Client Authorization Data

The SupplementalData message sent from the client to the server contains authorization data associated with the TLS client. Following the principle of least privilege, the client ought to send the minimal set of authorization information necessary to accomplish the task at hand. That is, only those authorizations that are expected to be required by the server in order to gain access to the needed server resources ought to be included. The format of the authorization data depends on the format negotiated in the client_authz hello message extension. The AuthorizationData structure is described in Section 3.3.

In some systems, clients present authorization information to the server, and then the server provides new authorization information. This type of transaction is not supported by SupplementalData messages. In cases where the client intends to request the TLS server to perform authorization translation or expansion services, such translation services ought to occur within the ApplicationData messages, not within the TLS Handshake protocol.

3.2. Server Authorization Data

The SupplementalData message sent from the server to the client contains authorization data associated with the TLS server. This authorization information is expected to include statements about the server’s qualifications, reputation, accreditation, and so on. Wherever possible, authorizations that can be misappropriated for fraudulent use ought to be avoided. The format of the authorization data depends on the format negotiated in the server_authz hello message extensions. The AuthorizationData structure is described in Section 3.3.
3.3. AuthorizationData Type

The AuthorizationData structure carried authorization information for either the client or the server. The AuthzDataFormat specified in Section 2.3 for use in the hello extensions is also used in this structure.

All of the entries in the authz_data_list MUST employ authorization data formats that were negotiated in the relevant hello message extension.

```
struct{
    AuthorizationDataEntry authz_data_list<1..2^16-1>;
} AuthorizationData;

struct {
    AuthzDataFormat authz_format;
    select (AuthzDataFormat) {
        case x509_attr_cert:         X509AttrCert;
        case saml_assertion:         SAMLAssertion;
        case x509_attr_cert_url:     URLandHash;
        case saml_assertion_url:     URLandHash;
    }
} AuthorizationDataEntry;

enum {
    x509_attr_cert(0),
    saml_assertion(1),
    x509_attr_cert_url(2),
    saml_assertion_url(3),
    (255)
} AuthzDataFormat;

opaque X509AttrCert<1..2^16-1>;

opaque SAMLAssertion<1..2^16-1>;

struct {
    opaque url<1..2^16-1>;
    HashType hash_type;
    select (hash_type) {
        case sha1:   SHA1Hash;
        case sha256: SHA256Hash;
    } hash;
} URLandHash;
```
enum {
    sha1(0), sha256(1), (255)
} HashType;

opaque SHA1Hash[20];

opaque SHA256Hash[32];

3.3.1. X.509 Attribute Certificate

When X509AttrCert is used, the field contains an ASN.1 DER-encoded X.509 Attribute Certificate (AC) that follows the profile in RFC 3281 [ATTRCERT]. An AC is a structure similar to a public key certificate (PKC) [PKIX1]; the main difference being that the AC contains no public key. An AC may contain attributes that specify group membership, role, security clearance, or other authorization information associated with the AC holder.

When making an authorization decision based on an AC, proper linkage between the AC holder and the public key certificate that is transferred in the TLS Certificate message is needed. The AC holder field provides this linkage. The holder field is a SEQUENCE allowing three different (optional) syntaxes: baseCertificateID, entityName and objectDigestInfo. In the TLS authorization context, the holder field MUST use the either baseCertificateID or entityName. In the baseCertificateID case, the baseCertificateID field MUST match the issuer and serialNumber fields in the certificate. In the entityName case, the entityName MUST be the same as the subject field in the certificate or one of the subjectAltName extension values in the certificate. Note that [PKIX1] mandates that the subjectAltName extension be present if the subject field contains an empty distinguished name.

3.3.2. SAML Assertion

When SAMLAssertion is used, the field contains an XML-encoded <Assertion> element using the AssertionType complex type as defined in [SAML1.1][SAML2.0]. SAML is an XML-based framework for exchanging security information. This security information is expressed in the form of assertions about subjects, where a subject is either human or computer with an identity. In this context, the SAML assertions are most likely to convey authentication or attribute statements to be used as input to authorization policy governing whether subjects are allowed to access certain resources. Assertions are issued by SAML authorities.

When making an authorization decision based on a SAML assertion, proper linkage between the SAML assertion and the public key
certificate that is transferred in the TLS Certificate message may be needed. A "Holder of Key" subject confirmation method in the SAML assertion can provide this linkage. In other scenarios, it may be acceptable to use alternate confirmation methods that do not provide a strong binding, such as a bearer mechanism. SAML assertion recipients MUST decide which subject confirmation methods are acceptable; such decisions MAY be specific to the SAML assertion contents and the TLS session context.

There is no general requirement that the subject of the SAML assertion correspond directly to the subject of the certificate. They may represent the same or different entities. When they are different, SAML also provides a mechanism by which the certificate subject can be identified separately from the subject in the SAML assertion subject confirmation method.

Since the SAML assertion is being provided at a part of the TLS Handshake that is unencrypted, an eavesdropper could replay the same SAML assertion when they establish their own TLS session. This is especially important when a bearer mechanism is employed, the recipient of the SAML assertion assumes that the sender is an acceptable attesting entity for the SAML assertion. Some constraints may be included to limit the context where the bearer mechanism will be accepted. For example, the period of time that the SAML assertion can be short-lived (often minutes), the source address can be constrained, or the destination endpoint can be identified. Also, bearer assertions are often checked against a cache of SAML assertion unique identifiers that were recently received in order to detect replay. This is an appropriate countermeasure if the bearer assertion is intended to be used just once. Section 5 provides a way to protect authorization information when necessary.

3.3.3. URL and Hash

Since the X.509 AC and SAML assertion can be large, alternatives provide a URL to obtain the ASN.1 DER-encoded X.509 AC or SAML Assertion. To ensure that the intended object is obtained, a one-way hash value of the object is also included. Integrity of this one-way hash value is provided by the TLS Finished message.

Implementations that support either x509_attr_cert_url or saml_assertion_url MUST support URLs that employ the http scheme. Other schemes may also be supported; however, to avoid circular dependencies, supported schemes SHOULD NOT themselves make use of TLS, such as the https scheme.

Implementations that support either x509_attr_cert_url or saml_assertion_url MUST support both SHA-1 [SHA1] and SHA-256 [SHA2]
as one-way hash functions. Other one-way hash functions may also be supported. Additional one-way hash functions can be registered in the future using the procedures in section 3.

4. IANA Considerations

This document defines a two TLS extensions: client_authz(TBD) and server_authz(TBD). These extension type values are assigned from the TLS Extension Type registry defined in [TLSEXT].

This document defines one TLS supplemental data type: authz_data(TBD). This supplemental data type is assigned from the TLS Supplemental Data Type registry defined in [TLSSUPP].

This document establishes a new registry, to be maintained by IANA, for TLS Authorization Data Formats. The first four entries in the registry are x509_attr_cert(0), saml_assertion(1), x509_attr_cert_url(2), and saml_assertion_url(3). TLS Authorization Data Format identifiers with values in the inclusive range 0-63 (decimal) are assigned via RFC 2434 [IANA] Standards Action. Values from the inclusive range 64-223 (decimal) are assigned via RFC 2434 Specification Required. Values from the inclusive range 224-255 (decimal) are reserved for RFC 2434 Private Use.

This document establishes a new registry, to be maintained by IANA, for TLS Hash Types. The first two entries in the registry are sha1(0) and sha256(1). TLS Hash Type identifiers with values in the inclusive range 0-158 (decimal) are assigned via RFC 2434 [IANA] Standards Action. Values from the inclusive range 159-223 (decimal) are assigned via RFC 2434 Specification Required. Values from the inclusive range 224-255 (decimal) are reserved for RFC 2434 Private Use.

5. Security Considerations

A TLS server can support more than one application, and each application may include several features, each of which requires separate authorization checks. This is the reason that more than one piece of authorization information can be provided.

A TLS server that requires different authorization information for different applications or different application features may find that a client has provided sufficient authorization information to grant access to a subset of these offerings. In this situation the TLS Handshake protocol will complete successfully; however, the server must ensure that the client will only be able to use the appropriate applications and application features. That is, the TLS server must deny access to the applications and application features
for which authorization has not been confirmed.

In many cases, the authorization information is itself sensitive. The double handshake technique can be used to provide protection for the authorization information. Figure 2 illustrates the double handshake, where the initial handshake does not include any authorization extensions, but it does result in protected communications. Then, a second handshake that includes the authorization information is performed using the protected communications. In Figure 2, the number on the right side indicates the amount of protection for the TLS message on that line. A zero (0) indicates that there is no communication protection; a one (1) indicates that protection is provided by the first TLS session; and a two (2) indicates that protection is provided by both TLS sessions.

The placement of the SupplementalData message in the TLS Handshake results in the server providing its authorization information before the client is authenticated. In many situations, servers will not want to provide authorization information until the client is authenticated. The double handshake illustrated in Figure 2 provides a technique to ensure that the parties are mutually authenticated before either party provides authorization information.

The use of bearer SAML assertions allows an eavesdropper or a man-in-the-middle to capture the SAML assertion and try to reuse it in another context. The constraints discussed in Section 3.3.2 might be effective against an eavesdropper, but they are less likely to be effective against a man-in-the-middle. Authentication of both parties in the TLS session, which involves the use of client authentication, will prevent an undetected man-in-the-middle, and the use of the double handshake illustrated in Figure 2 will prevent the disclosure of the bearer SAML assertion to any party other than the TLS peer.

6. Acknowledgement

The authors thank Scott Cantor for his assistance with the SAML Assertion portion of the document.
Client Hello (no extensions) --------> 
    ServerHello (no extensions) 0
    Certificate* 0
    ServerKeyExchange* 0
    CertificateRequest* 0
    <-------- ServerHelloDone 0

Certificate* 0
ClientKeyExchange 0
CertificateVerify* 0
[ChangeCipherSpec] 0
Finished --------> [ChangeCipherSpec] 1
<-------- Finished 1

ClientHello (w/ extensions) --------> 
    ServerHello (w/ extensions) 1
    SupplementalData (w/ authz data)* 1
    Certificate* 1
    ServerKeyExchange* 1
    CertificateRequest* 1
    <-------- ServerHelloDone 1

SupplementalData (w/ authz data)* 1
Certificate* 1
ClientKeyExchange 1
CertificateVerify* 1
[ChangeCipherSpec] 1
Finished --------> [ChangeCipherSpec] 2
<-------- Finished 2

Application Data <-------- Application Data 2

Figure 2. Double Handshake to Protect Authorization Data

7. Normative References


Author’s Address

Mark Brown
RedPhone Security
2019 Palace Avenue
Saint Paul, MN 55105
USA
mark <at> redphonesecurity <dot> com

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
housley <at> vigilsec <dot> com

Full Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
The IETF takes no position regarding the validity or scope of any
Intellectual Property Rights or other rights that might be claimed to
pertain to the implementation or use of the technology described in
this document or the extent to which any license under such rights
might or might not be available; nor does it represent that it has
made any independent effort to identify any such rights. Information
on the procedures with respect to rights in RFC documents can be
found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any
assurances of licenses to be made available, or the result of an
attempt made to obtain a general license or permission for the use of
such proprietary rights by implementers or users of this
specification can be obtained from the IETF on-line IPR repository at

The IETF invites any interested party to bring to its attention any
copyrights, patents or patent applications, or other proprietary
rights that may cover technology that may be required to implement
this standard. Please address the information to the IETF at
ietf-ipr@ietf.org.