Internet Engineering Task Force (IETF)
Request for Comments: 5878
Updates: 5246
Category: Experimental
ISSN: 2070-1721
May 2010

M. Brown
RedPhone Security
R. Housley
Vigil Security

Transport Layer Security (TLS) Authorization Extensions

Abstract

This document specifies authorization extensions to the Transport Layer Security (TLS) Handshake Protocol. Extensions are 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, such as attribute certificates (ACs) or Security Assertion Markup Language (SAML) assertions, is exchanged in the supplemental data handshake message.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5878.

Copyright Notice

1. Introduction

The Transport Layer Security (TLS) protocol ([TLS1.0], [TLS1.1], [TLS1.2]) 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.

The format and content of the authorization information carried in these extensions are extensible. This document references Security Assertion Markup Language (SAML) assertion ([SAML1.1], [SAML2.0]) and X.509 attribute certificate (AC) [ATTRCERT] authorization formats, but other formats can be used. Future authorization extensions may include any opaque assertion that is digitally signed by a trusted issuer. Recognizing the similarity to certification path validation, this document recommends the use of TLS Alert messages related to certificate processing to report authorization information processing failures.

Straightforward binding of identification, authentication, and authorization information to an encrypted session 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 TLS 1.0, TLS 1.1, and TLS 1.2. These extensions observe the conventions defined for TLS extensions that were originally defined in [TLSEXT1] and revised in [TLSEXT2]; TLS extensions are now part of TLS 1.2 [TLS1.2]. TLS extensions use 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, TLS 1.1, and TLS 1.2. 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 set up; 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" 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.

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.

Successful session resumption uses the same authorization information as the original session.

```
   Client                                                 Server

   ClientHello (w/ extensions) -------->

                                    ServerHello (w/ extensions)
                                              SupplementalData*
                                                   Certificate*
                                             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

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 [TLS1.2], the extension format used in the extended client hello message and extended server hello message is repeated here for convenience:

```
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. 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:

```
enum {
  client_authz(7), server_authz(8), (65535)
} ExtensionType;
```

The authorization extensions are relevant when a session is initiated and on 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.

When a session is resumed, ClientHello is followed immediately by ChangeCipherSpec, which does not provide an opportunity for different authorization information can be exchanged. Successful session resumption MUST use the same authorization information as the original session.

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 is 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 5755 [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 5755 [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.

Implementations that support either x509_attr_cert_url or saml_assertion_url MUST support URLs that employ the http scheme [HTTP]. These implementations MUST confirm that the hash value computed on the fetched authorization matches the one received in the handshake. Mismatch of the hash values SHOULD be treated as though the authorization was not provided, which will result in a bad_certificate_hash_value alert (see Section 4). Implementations

MUST deny access if the authorization cannot be obtained from the provided URL, by sending a certificate_unobtainable alert (see Section 4).

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(16386), (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, and 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, and the following fictitious example of a single 5-octet SAML assertion illustrates its use:

```
17             # Handshake.msg_type == supplemental_data(23)
00 00 11       # Handshake.length = 17
00 00 0e       # length of SupplementalData.supp_data = 14
40 02          # SupplementalDataEntry.supp_data_type = 16386
00 0a          # SupplementalDataEntry.supp_data_length = 10
00 08          # length of AuthorizationData.authz_data_list = 8
01             # authz_format = saml_assertion(1)
00 05          # length of SAMLAssertion
aa aa aa aa aa # SAML assertion (fictitious: "aa aa aa aa aa")
```

3.3. AuthorizationData Type

The AuthorizationData structure carries 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.

The HashAlgorithm type is taken from [TLS1.2], which allows additional one-way hash functions to be registered in the IANA TLS HashAlgorithm registry in the future.

```
      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>;
         HashAlgorithm hash_alg;
         select (hash_alg) {
            case md5:    MD5Hash;
            case sha1:   SHA1Hash;
            case sha224: SHA224Hash;
            case sha256: SHA256Hash;
            case sha384: SHA384Hash;
            case sha512: SHA512Hash;
         } hash;
      } URLandHash;

      enum {
         none(0), md5(1), sha1(2), sha224(3), sha256(4), sha384(5),
         sha512(6), (255)
      } HashAlgorithm;
```

```
      opaque MD5Hash[16];

      opaque SHA1Hash[20];

      opaque SHA224Hash[28];

      opaque SHA256Hash[32];

      opaque SHA384Hash[48];

      opaque SHA512Hash[64];
```

3.3.1. X.509 Attribute Certificate

When X509AttrCert is used, the field contains an ASN.1 Distinguished Encoding Rules (DER)-encoded X.509 attribute certificate (AC) that follows the profile in RFC 5755 [ATTRCERT]. An AC is a structure similar to a public key certificate (PKC) [PKIX1]; the main difference is 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 either the 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 MUST contain well-formed XML [XML1.0] and MUST use either UTF-8 [UTF-8] or UTF-16 [UTF-16] character encoding. UTF-8 is the preferred character encoding. The XML text declaration MUST be followed by an <Assertion> element using the AssertionType complex type as defined in [SAML1.1] and [SAML2.0]. The XML text MUST also follow the rules of [XML1.0] for including the Byte Order Mark (BOM) in encoded entities. 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 6 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. When dereferencing these URLs, circular dependencies MUST be avoided. Avoiding TLS when dereferencing these URLs is one way to avoid circular dependencies. Therefore, clients using the HTTP scheme MUST NOT use these TLS extensions if UPGRADE in HTTP [UPGRADE] is used. For other schemes, similar care must be taken to avoid using these TLS extensions.

Implementations that support either x509_attr_cert_url or saml_assertion_url MUST support both SHA-1 [SHS] and SHA-256 [SHS] as one-way hash functions. Other one-way hash functions may also be supported. Additional one-way hash functions can be added to the IANA TLS HashAlgorithm registry in the future.

Implementations that support x509_attr_cert_url MUST support responses that employ the "application/pkix-attr-cert" Multipurpose Internet Mail Extension (MIME) media type as defined in [ACTYPE].

Implementations that support saml_assertion_url MUST support responses that employ the "application/samlassertion+xml" MIME type as defined in Appendix A of [SAMLBIND].

TLS authorizations SHOULD follow the additional guidance provided in Section 3.3 of [TLSEXT2] regarding client certificate URLs.

4. Alert Messages

This document specifies the reuse of TLS Alert messages related to public key certificate processing for any errors that arise during authorization processing, while preserving the AlertLevels as authoritatively defined in [TLS1.2] or [TLSEXT2]. All alerts used in authorization processing are fatal.

The following updated definitions for the Alert messages are used to describe errors that arise while processing authorizations. For ease of comparison, we reproduce the Alert message definition from Section 7.2 of [TLS1.2], augmented with two values defined in [TLSEXT2]:

```
      enum { warning(1), fatal(2), (255) } AlertLevel;

      enum {
          close_notify(0),
          unexpected_message(10),
          bad_record_mac(20),
          decryption_failed_RESERVED(21),
          record_overflow(22),
          decompression_failure(30),
          handshake_failure(40),
          no_certificate_RESERVED(41),
          bad_certificate(42),
          unsupported_certificate(43),
          certificate_revoked(44),
          certificate_expired(45),
          certificate_unknown(46),
          illegal_parameter(47),
          unknown_ca(48),
          access_denied(49),
          decode_error(50),
          decrypt_error(51),
          export_restriction_RESERVED(60),
          protocol_version(70),
          insufficient_security(71),
          internal_error(80),
          user_canceled(90),
          no_renegotiation(100),
          unsupported_extension(110),
          certificate_unobtainable(111),
          bad_certificate_hash_value(114),
          (255)
      } AlertDescription;

      struct {
          AlertLevel level;
          AlertDescription description;
      } Alert;
```

TLS processing of alerts includes some ambiguity because the message does not indicate which certificate in a certification path gave rise to the error. This problem is made slightly worse in this extended use of alerts, as the alert could be the result of an error in processing of either a certificate or an authorization. Implementations that support these extensions should be aware of this imprecision.

The AlertDescription values are used as follows to report errors in authorizations processing:

bad_certificate

In certificate processing, bad_certificate indicates that a certificate was corrupt, contained signatures that did not verify correctly, and so on. Similarly, in authorization processing, bad_certificate indicates that an authorization was corrupt, contained signatures that did not verify correctly, and so on. In authorization processing, bad_certificate can also indicate that the handshake established that an AuthzDataFormat was to be provided, but no AuthorizationData of the expected format was provided in SupplementalData.

unsupported_certificate

In certificate processing, unsupported_certificate indicates that a certificate was of an unsupported type. Similarly, in authorization processing, unsupported_certificate indicates that AuthorizationData uses a version or format unsupported by the implementation.

certificate_revoked

In certificate processing, certificate_revoked indicates that a certificate was revoked by its issuer. Similarly, in authorization processing, certificate_revoked indicates that authorization was revoked by its issuer, or a certificate that was needed to validate the signature on the authorization was revoked by its issuer.

certificate_expired

In certificate processing, certificate_expired indicates that a certificate has expired or is not currently valid. Similarly, in authorization processing, certificate_expired indicates that an authorization has expired or is not currently valid.

certificate_unknown

In certificate processing, certificate_unknown indicates that some other (unspecified) issue arose while processing the certificate, rendering it unacceptable. Similarly, in authorization processing, certificate_unknown indicates that processing of AuthorizationData failed because of other (unspecified) issues, including AuthzDataFormat parse errors.

unknown_ca

In certificate processing, unknown_ca indicates that a valid certification path or partial certification path was received, but the certificate was not accepted because the certification authority (CA) certificate could not be located or could not be matched with a known, trusted CA. Similarly, in authorization processing, unknown_ca indicates that the authorization issuer is not known and trusted.

access_denied
In certificate processing, access_denied indicates that a valid certificate was received, but when access control was applied, the sender decided not to proceed with negotiation. Similarly, in authorization processing, access_denied indicates that the authorization was not sufficient to grant access.

certificate_unobtainable
The client_certificate_url extension defined in RFC 4366 [TLSEXT2] specifies that download errors lead to a certificate_unobtainable alert. Similarly, in authorization processing, certificate_unobtainable indicates that a URL does not result in an authorization. While certificate processing does not require this alert to be fatal, this is a fatal alert in authorization processing.

bad_certificate_hash_value
In certificate processing, bad_certificate_hash_value indicates that a downloaded certificate does not match the expected hash. Similarly, in authorization processing, bad_certificate_hash_value indicates that a downloaded authorization does not match the expected hash.

5. IANA Considerations

This document defines two TLS extensions: client_authz(7) and server_authz(8). These extension type values are assigned from the TLS Extension Type registry defined in [TLSEXT2].

This document defines one TLS supplemental data type: authz_data(16386). 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 5226 [IANA] IETF Review. Values from the inclusive range 64-223 (decimal) are assigned via RFC 5226 Specification Required. Values from the inclusive range 224-255 (decimal) are reserved for RFC 5226 Private Use.

6. 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 cases where the authorization information itself is 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.

AuthzDataFormats that point to authorization data, such as x509_attr_cert_url and saml_assertion_url, rather than simply including the authorization data in the handshake, may be exploited by an attacker. Implementations that accept pointers to authorization data SHOULD adopt a policy of least privilege that limits the acceptable references that they will attempt to use. For more information, see Section 6.3 of [TLSEXT2].

```
    Client                                                   Server

    ClientHello (no extensions) -------->                            |0
                                        ServerHello (no extensions)  |0
                                                       Certificate*  |0
                                                 ServerKeyExchange*  |0
                                                CertificateRequest*  |0
                                <--------           ServerHelloDone  |0
    Certificate*                                                     |0
    ClientKeyExchange                                                |0
    CertificateVerify*                                               |0
    [ChangeCipherSpec]                                               |0
    Finished                    -------->                            |1
                                                 [ChangeCipherSpec]  |0
                                <--------                  Finished  |1
    ClientHello (w/ extensions) -------->                            |1
                                        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                    -------->                            |2
                                                 [ChangeCipherSpec]  |1
                                <--------                  Finished  |2
    Application Data            <------->          Application Data  |2
```

Figure 2. Double Handshake To Protect Authorization Data

7. Acknowledgement

The authors thank Scott Cantor for his assistance with the SAML assertion portion of the document.

8. References

8.1. Normative References

[ACTYPE] Housley, R., "The application/pkix-attr-cert Media Type for Attribute Certificates", RFC 5877, May 2010.

[ATTRCERT] Farrell, S., Housley, R., and S. Turner, "An Internet Attribute Certificate Profile for Authorization", RFC 5755, January 2010.

[HTTP] 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.

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

[PKIX1] 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.

[SAML1.1] OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 1.1 Specification Set", September 2003.

[SAML2.0] OASIS Security Services Technical Committee, "Security Assertion Markup Language (SAML) Version 2.0 Specification Set", March 2005.

[SAMLBIND] OASIS Security Services Technical Committee, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0", March 2005.

[SHS] National Institute of Standards and Technology (NIST), FIPS PUB 180-3, Secure Hash Standard (SHS), October 2008.

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

[TLS1.0] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[TLS1.1] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[TLS1.2] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[TLSEXT2] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[TLSSUPP] Santesson, S., "TLS Handshake Message for Supplemental Data", RFC 4680, October 2006.

[UPGRADE] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[UTF-8] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[UTF-16] Hoffman, P. and F. Yergeau, "UTF-16, an encoding of ISO 10646", RFC 2781, February 2000.

[XML1.0] Bray, T., J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", http://www.w3.org/TR/xml/, November 2008.

8.2. Informative References

[TLSEXT1] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

Authors’ Addresses

Mark Brown
RedPhone Security
1199 Falls View Court
Mendota Heights, MN 55118
USA
EMail: mark@redphonesecurity.com

Russell Housley
Vigil Security, LLC
918 Spring Knoll Drive
Herndon, VA 20170
USA
EMail: housley@vigilsec.com