NETWORK WORKING GROUP
Internet-Draft
Updates: 4279 (if approved)
Intended status: Standards Track
Expires: January 26, 2008

L. Zhu
G. Chander
Microsoft Corporation
J. Altman
Secure Endpoints Inc.
S. Santesson
Microsoft Corporation

July 25, 2007

Flexible Key Agreement for Transport Layer Security (FKA-TLS) draft-santesson-tls-gssapi-03

Status of this Memo

Copyright Notice

Abstract

This document defines extensions to RFC 4279, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", to enable dynamic key sharing in distributed environments using a Generic Security Service Application Program Interface (GSS-API) mechanism, and then

import that shared key as the "Pre-Shared Key" to complete the TLS handshake.

This is a modular approach to perform authentication and key exchange based on off-shelf libraries. And it obviates the need of pair-wise key sharing by enabling the use of the widely-deployed Kerberos alike trust infrastructures that are highly scalable and robust. Furthermore, conforming implementations can provide server authentication without the use of certificates.

Table of Contents

1. Introduction

[RFC4279] defines Transport Layer Security (TLS) based on pre-shared keys (PSK). This assumes a pair-wise key sharing scheme that is less scalable and more costly to manage in comparison with a trusted third party scheme such as Kerberos [RFC4120]. In addition, off-shelf GSS-API libraries that allow dynamic key sharing are not currently accessible to TLS applications. Lastly, [RFC4279] does not provide true mutual authentication against the server.

This document extends [RFC4279] to establish a shared key, and optionally provide client or server authentication, by using off-shelf GSS-API libraries, and the established shared key is then imported as "PSK" to [RFC4279]. No new key cipher suite is defined in this document.

As an example usage scenario, Kerberos [RFC4121] is a GSS-API mechanism that can be selected to establish a shared key between a client and a server based on either asymmetric keys [RFC4556] or symmetric keys [RFC4120]. By using the extensions defined in this document, a TLS connection is secured using the Kerberos version 5 mechanism exposed as a generic security service via GSS-API.

With regard to the previous work for the Kerberos support in TLS, [RFC2712] defines "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)" which has not been widely implemented due to violations of Kerberos Version 5 library abstraction layers, incompatible implementations from two major distributions (Sun Java and OpenSSL), and its lack of support for credential delegation. This document defines a generic extensible method that addresses the limitations associated with [RFC2712] and integrates Kerberos and TLS. Relying on [RFC4121] for Kerberos Version 5 support will significantly reduce the challenges associated with implementing this protocol as a replacement for [RFC2712].

2. Conventions 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].

3. Protocol Definition

In this protocol, the on-demand key exchange is implemented by encapsulating the GSS security context establishment within the TLS handshake messages when PSK cipher suites are requested in the

extended ClientHello message.

The gss_api TLS extension is defined according to [RFC3546]. The extension data carries GSS-API token within the TLS hello messages.

```
enum {
    gss_api(TBD), (65535)
} ExtensionType;
```

The client MUST NOT include a gss_api TLS extension if there is no PSK ciphersuite [RFC4279] included in the cipher_suites field of the client hello message.

Initially the client computes the gss_api TLS extension data by calling GSS_Init_sec_context() [RFC2743] to establish a security context. The TLS client MUST set the mutual_req_flag and identify the server by targ_name so that mutual authentication is performed in the course of context establishment. The extension_data from the client contains the output token of GSS_Init_sec_context().

If a GSS-API context cannot be established, the gss_api TLS extension MUST NOT be included in the client hello message and it is a matter of local policy on the client whether to continue or reject the TLS authentication as if the gss_api TLS extension is not supported.

If the mutual authentication is not available on the established GSS-API context, the PSK key exchange described in Section 2 of [RFC4279] MUST NOT be selected, and the DHE_PSK or RSA_PSK key exchange MUST be negotiated instead in order to authenticate the server.

Upon receipt of the gss_api TLS extension from the client, and if the server supports the gss_api TLS extension, the server calls GSS_Accept_sec_context() with the client GSS-API output token in the client's extension data as the input token. If GSS_Accept_sec_context() returns a token successfully, the server responds by including a gss_api TLS extension in the server hello message and places the output token in the extension_data. If GSS_Accept_sec_context() fails, it is a matter of local policy on the server whether to continue or reject the TLS authentication as if the gss_api TLS extension is not supported.

The server MUST ignore a TLS gss_api extension in the extended ClientHello if its selected CipherSuite is not a PSK CipherSuite [RFC4279], and the server MUST NOT include a gss_api TLS extension in the server hello message.

If after the exchange of extended ClientHello and extended ServerHello with the gss_api extension, at least one more additional

GSS token is required in order to complete the GSS security context establishment, the additional GSS-API token is encapsulated in a new TLS Handshake message called the token_transfer message.

```
enum {
    token_transfer(TBD), (255)
} HandshakeType;

struct {
    HandshakeType msg_type;    /* handshake type */
    uint24 length;             /* bytes in message */
    select (HandshakeType) {
        case token_transfer: /* NEW */
              TokenTransfer;
    } body;
} Handshake;

enum {
    gss_api_token(1), (255)
} TokenTransferType;

struct {
      TokenTransferType token_type; /* token type */
      opaque token<0..2^16-1>;
} TokenTransfer;
```

The TokenTransfer structure is filled out as follows:

- The token_type is gss_api_token.

- The token field contains the GSS-API context establishment tokens from the client and the server.

The client calls GSS_Init_sec_context() with the token in the TokenTransfer stucture from the server as the input token, and then places the output token, if any, into the TokenTransfer message and sends the handshake message to the server. The server calls GSS_Accept_sec_context() with the token in the TokenTransfer structure from the client as the input token, and then places the output token, if any, into the TokenTransfer message and sends the handshake message to the client.

This loop repeats until either the context fails to establish or the context is established successfully. To prevent an infinite loop, both the client and the server MUST have a policy to limit the maximum number of GSS-API context establishment calls for a given session. The recommended value is a total of five (5) calls including the GSS_Init_sec_context() and GSS_Accept_sec_context()

from both the client and server. Exceeding the maximum number of calls is to be treated as a GSS security context establishment failure. It is RECOMMENDED that the client and server enforce the same maximum number

If the GSS-API context fails to establish, it is a matter of local policy whether to continue or reject the TLS authentication as if the gss_api TLS extension is not supported.

When the last GSS-API context establishment token is sent by the client or when the GSS-API context fails to establish on the client side and the local policy allows the TLS authentication to proceed as if the TLS gss_api extension is not supported, the client sends an empty TokenTransfer handshake message.

If the GSS-API context fails to establish and local policy allows the TLS authentication continue as if the gss_api TLS extension is not supported, the server MAY send another ServerHello message in order to choose a different cipher suite. The client then MUST expect the second ServerHello message from the server before the session is established. The additional ServerHello message MUST only differ from the first ServerHello message in the choice of CipherSuite and it MUST NOT include a TLS gss_api extension. The second ServerHello MUST NOT be present if there is no TokenTransfer message.

If the client and the server establish a security context successfully, both the client and the server call GSS_Pseudo_random() [RFC4401] to compute a sufficiently long shared secret with the same value based on the negotiated cipher suite (see details below), and then proceed according to [RFC4279] using this shared secret value as the "PSK".

When the shared key is established using a GSS-API mechanism as described in this document, the identity of the server and the identity of the client MUST be obtained from the GSS security context. In this case, the PSK identity MUST be processed as follows:

- The PSK identity as defined in Section 5.1 of [RFC4279] MUST be specified as an empty string.

- If the server key exchange message is present, the PSK identity hint as defined in Section 5.2 of [RFC4279] MUST be empty, and it MUST be ignored by the client.

The input parameters to GSS_Pseudo_random() to compute the shared secret value MUST be provided as follows:

- The context is the handle to the GSS-API context established in the given session.
- The prf_key is GSS_C_PRF_KEY_FULL.
- The prf_in contains the UTF8 encoding of the string "GSS-API TLS PSK".
- The desired_output_len is 64. In other words, the output keying mastering size is 64 in bytes. Note that this is the maximum PSK length required to be supported by implementations conforming to [RFC4279].

The following text art summaries the protocol message flow.

Fig. 1. Message flow for a full handshake

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

There could be multiple TokenTransfer handshake messages, and the last TokenTransfer message, if present, is always sent from the client to the server and it can carry an empty token.

4. Choosing GSS-API Mechanisms

If more than one GSS-API mechanism is shared between the client and the server, it is RECOMMENDED to deploy a pseudo GSS-API mechanism such as [RFC4178] to choose a mutually preferred GSS-API mechanism.

When Kerberos is selected as the GSS-API mechanism, the extensions defined in [KRB-ANON] can perform server authentication without client authentication, thus provide the functional equivalence to the certificate-based TLS [RFC4346].

If the Kerberos client does not have access to the KDC but the server does, [IAKERB] can be chosen to tunnel the Kerberos authentication exchange within the TLS handshake messages.

5. Client Authentication

If the GSS-API mechanism in the gss_api TLS extension provides client authentication [RFC2743], the CertificateRequest, the client Certificate and the CertificateVerify handshake messages MUST NOT be present. This is illustrated in Appendix A.

6. Protecting GSS-API Authentication Data

GSS-API [RFC2743] provides security services to callers in a generic fashion, supportable with a range of underlying mechanisms and technologies and hence allowing source-level portability of applications to different environments. For example, Kerberos is a GSS-API mechanism defined in [RFC4121]. It is possible to design a GSS-API mechanism that can be used with FKA-TLS in order to, for example, provide client authentication, and is so weak that its GSS-API token MUST NOT be in clear text over the open network. A good example is a GSS-API mechanism that implements basic authentication. Although such mechanisms are unlikely to be standardized and will be encouraged in no circumstance, they exist for practical reasons. In addition, it is generally beneficial to provide privacy protection for mechanisms that send client identities in the clear.

In order to provide a standard way for protecting weak GSS-API data for use over FKA-TLS, TLSWrap is defined in this section as a pseudo GSS-API mechanism that wraps around the real GSS-API authentication context establishment tokens. This pseudo GSS-API mechanism does not provide per-message security. The real GSS-API mechanism protected by TLSWrap may provide per-message security after the context is established.

The syntax of the initial TLSWrap token follows the initialContextToken syntax defined in Section 3.1 of [RFC2743]. The TLSWrap pseudo mechanism is identified by the Object Identifier iso.org.dod.internet.security.mechanism.tls-wrap (1.3.6.1.5.5.16). Subsequent TLSWrap tokens MUST NOT be encapsulated in this GSS-API generic token framing.

TLSWrap encapsulates the TLS handshake and data protection in its context establishment tokens.

The innerContextToken [RFC2743] for the initial TLSWrap context token contains the ClientHello message encoded according to [RFC4346]. No PSK ciphersuite can be included in the client hello message. The targ_name is used by the client to identify the server and it follows the name forms defined in Section 4 of [PKU2U].

Upon receipt of the initial TLSWrap context token, the GSS-API server processes the client hello message. The output GSS-API context token for TLSWrap contains the ServerHello message and the ServerHelloDone potentially with the optional handshake messages in the order as defined in [RFC4346].

The GSS-API client then processes the server reply and returns the ClientKeyExchange message and the Finished message potentially with the optional handshake messages in the order as defined in [RFC4346]. The client places the real GSS-API authentication mechanism token as an application data record right after the TLS Finished message in the same GSS-API context token for TLSWrap. Because the real mechanism token is placed after the ChangeCipherSpec message, the GSS-API data for the real mechanism is encrypted. If the GSS-API server is not authenticated at this point of the TLS handshake for TLSWrap, the TLSWrap context establishment MUST fail and the real authentication mechanism token MUST not be returned.

The GSS-API server in turn processes the client reply and returns the TLS Finished message, the server places the reply token from the real authentication mechanism, if present, as an application data record.

If additional TLS messages are needed before the application data, these additional TLS messages are encapsulated in the context token of TLSWrap in the same manner how the client hello message and the server hello message are encapsulated as described above.

If additional tokens are required by the real authentication mechanism in order to establish the context, these tokens are placed as an application data record, encoded according to [RFC4346] and then returned as TLSWrap GSS-API context tokens, with one TLSWrap context token per each real mechanism context token. The real

mechanism context tokens are decrypted by TLSWrap and then supply to the real mechanism to complete the context establishment.

7. Security Considerations

As described in Section 3, when the shared key is established using a GSS-API mechanism as described in this document, the identity of the server MUST be obtained from the GSS security context and the identity of the client MUST be obtained from the GSS security context. Authentication methods such as GSS security context and X.509 certificate mixed MUST NOT conflict. Such confusion about the identity will interfere with the ability to properly determine the client's authorization privileges, thus potentially result in a security weakness.

When Kerberos as defined in [RFC4120] is used to establish the share key, it is vulnerable to offline dictionary attacks. The threat is mitigated by deploying Kerberos FAST [KRB-FAST].

Shared symmetric keys obtained from mutual calls to GSS_Pseudo_random() are not susceptible to off-line dictionary attacks in the same way that traditional pre-shared keys are. The strength of the generated keys are determined based upon the security properties of the selected GSS mechanism. Implementers MUST take into account the Security Considerations associated with the GSS mechanisms they decide to support.

8. Acknowledgements

Ari Medvinsky was one of the designers of the original TLS Kerberos version 5 CipherSuite and contributed to the first two revisions of this protocol specification.

Raghu Malpani provided insightful comments and was very helpful along the way.

Ryan Hurst contributed significantly to the use cases of FKA-TLS.

Love Hornquist Astrand, Nicolas Williams and Martin Rex provided helpful comments while reviewing early revisions of this document.

9. IANA Considerations

A new handshake message token_transfer is defined according to [RFC4346] and a new TLS extension called the gss_api extension is

defined according to [RFC3546]. The registry needs to be updated to include these new types.

This document defines the type of the transfer tokens in Section 3, a registry need to be setup and the allocation policy is "Specification Required".

10. References

10.1. Normative References

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

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.

10.2. Informative References

[IAKERB] Zhu, L., "Initial and Pass Through Authentication Using Kerberos V5 and the GSS-API", draft-zhu-ws-kerb-03.txt (work in progress), 2007.

[KRB-ANON] Zhu, L. and P. Leach, "Kerberos Anonymity Support", draft-ietf-krb-wg-anon-04.txt (work in progress), 2007.

[KRB-FAST] Zhu, L. and S. Hartman, "A Generalized Framework for Kerberos Pre-Authentication", draft-ietf-krb-wg-preauth-framework-06.txt (work in progress), 2007.

[PKU2U] Zhu, L., Altman, J., and A. Medvinsky, "Public Key Cryptography Based User-to-User Authentication - (PKU2U)", draft-zhu-pku2u-02.txt (work in progress), 2007.

[RFC2487] Hoffman, P., "SMTP Service Extension for Secure SMTP over TLS", RFC 2487, January 1999.

[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.

[RFC2712] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.

[RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 4402, February 2006.

[RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.

Appendix A. An FKA-TLS Example: Kerberos TLS

This section provides a non-normative description of the message flow when Kerberos Version 5 is used to established the shared secret according to [RFC4121] and that shared secret is then used to secure the TLS connection according to FKA-TLS defined in this document.

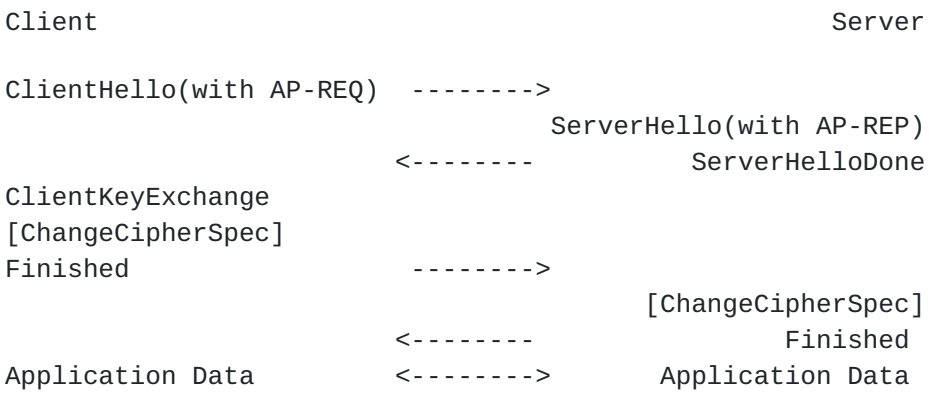

Fig. 2. Kerberos FKA-TLS example message flow

In this successful authentication sample, the TLS client sends the Kerberos AP-REQ [RFC4120] in the inital context token according to [RFC4121]. The initial GSS-API context token from the GSS-API client contains the Object Identifier that signifies the Kerberos mechanism and it is encapsulated in the gss_api TLS extension in the client hello message. The TLS client always requests mutual authentication, and the TLS server then sends a GSS-API context token that contains the AP-REP [RFC4120] according to [RFC4121]. The TLS server's GSS-API context token is encapsulated in the gss_api TLS extension in the server hello message. The GSS-API context is established at that point and both sides can derive the shared secret value according to [RFC4402].

In this example, the ServerKeyExchange handshake message is not needed and it is not present. And according to Section 5 none of the CertificateRequest, the client Certificate or the CertificateVerify handshake messages is present.

Appendix B. Additional Use Cases for FXA-TLS

TLS runs on layers beneath a wide range of application protocols such

as LDAP [RFC4510], SMTP [RFC2487], and XMPP [RFC3920] and above a reliable transport protocol. TLS can add security to any protocol that uses reliable connections (such as TCP). TLS is also increasingly being used as the standard method for protecting SIP [RFC3261] application signaling. TLS can provide authentication and encryption of the SIP signaling associated with VOIP (Voice over IP) and other SIP-based applications.

Today these applications use public key certificates to verify the identity of endpoints.

However, it is overwhelmingly complex to manage the assurance level of the certificates when deploying PKI and such complexity has gradually eroded the confidence for the PKI-based systems in general. In addition, the perceived overhead of deploying and managing certificates is fairly high. As a result, the industry badly needs the ability to secure TLS connections by leveraging the existing credential infrastructure. For many customers that means Kerberos. It is highly desirable to enable PKI-less deployments yet still offer strong authentication.

Having Kerberos/GSS-API in the layer above TLS means all TLS applications need to be changed in the protocol level. In many cases, such changes are not technically feasible. For example, [RFC4559] provides integration with Kerberos in the HTTP level. It suffers from a couple of drawbacks, most notably it only supports single-round-trip GSS-API mechanisms and it lacks of channel bindings to the underlying TLS connection which makes in unsuitable for deployment in situations where proxies exists. Furthermore, [RFC4559] lacks of session-based re-authentication (comparing with TLS). The root causes of these problems are inherent to the HTTP protocol and can't be fixed trivially.

Consequently, It is a better solution to integrate Kerberos/GSS-API in the TLS layer. Such integration allows the existing infrastructure work seamlessly with TLS for the products based on them in ways that were not practical to do before. For instance, an increasing number of client and server products support TLS natively, but many still lack support. As an alternative, users may wish to use standalone TLS products that rely on being able to obtain a TLS connection immediately, by simply connecting to a separate port reserved for the purpose. For example, by default the TCP port for HTTPS is 443, to distinguish it from HTTP on port 80. TLS can also be used to tunnel an entire network stack to create a VPN, as is the case with OpenVPN. Many vendors now marry TLS's encryption and authentication capabilities with authorization. There has also been substantial development since the late 1990s in creating client technology outside of the browser to enable support for client/server

applications. When compared against traditional IPSec VPN technologies, TLS has some inherent advantages in firewall and NAT traversal that make it easier to administer for large remote-access populations.

PSK-TLS as defined in [RFC4279] is a good start but this document finishes the job by making it more deployable. FKA-TLS also fixes the mutual-authentication problem in [RFC4279] in the cases where the PSK can be shared among services on the same host.

Authors' Addresses

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: lzhu@microsoft.com

Girish Chander
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
US

Email: gchander@microsoft.com

Jeffrey Altman
Secure Endpoints Inc.
255 W 94th St
New York, NY 10025
US

Email: jaltman@secure-endpoints.com

Stefan Santesson
Microsoft Corporation
Tuborg Boulevard 12
2900 Hellerup, WA
Denmark

Email: stefans@microsoft.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).