INTERNET-DRAFT
Transport Layer Security Working Group
Draft-ietf-tls-passauth-00.txt

Daniel Simon
Microsoft Corp.

November 20, 1996
Expires: May 25, 1997

Addition of Shared Key Authentication to Transport Layer Security (TLS)

0. Status Of this Memo

This document is an Internet-Draft. 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.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

1. Abstract

This document presents a shared-key authentication mechanism for the TLS protocol. It is intended to allow TLS clients to authenticate using a secret key (such as a password) shared with either the server or a third-party authentication service. The security of the secret authentication key is augmented by its integration into the normal SSL/TLS server authentication/key exchange mechanism.

2. Introduction

Recent transport-layer security protocols for the Internet, such as SSL versions 2.0 and 3.0 [1, 2] and PCT version 1 [3], have effected challenge-response authentication using strictly public-key (asymmetric) cryptographic methods, with no use of out-of-band shared secrets. This choice has both benefits and drawbacks. The primary benefit is improved security: an asymmetric private key used for authentication is only stored in one location, and the out-of-band identification necessary for public key certification need only be reliable, not secret (as an out-of-band shared key exchange must be). In addition, the difficult task of out-of-band shared-key exchange in shared-key authentication systems

often leads implementers to resort to human-friendly shared keys (manually typed passwords, for instance), which may be vulnerable to discovery by brute force search or "social engineering".

However, shared-key authentication has certain advantages as well. These are, chiefly:

- Portability: Precisely because shared keys are often human-remembered passwords or passphrases, they can be transported from (trusted) machine to (trusted) machine with ease--unlike asymmetric private keys, which must be transported using some physical medium, such as a diskette or "smart card", to be available for use on any machine.

- Backward Compatibility: Shared-key authentication is in very wide use today, and the cost of conversion to its public-key counterpart may not be worth the extra security, to some installations.

- Established Practice: Shared-key authentication has been in use for quite a while, and a valuable body of tools, techniques and expertise has grown up around it. In contrast, public-key authentication is very new, its associated tools and methods are either untested or non-existent, and experience with possible implementation or operation pitfalls simply doesn't exist.

These reasons are particularly relevant when individual human users of a service are being authenticated over the Internet, and as a result, virtually all authentication of (human) clients of such services is currently performed using shared passwords. Typically, servers implementing one of the aforementioned transport-layer security protocols, and needing client authentication, simply accept secure (i.e., encrypted and server-authenticated) connections from each client, who then provides a password (or engages in a challenge-response authentication protocol based on a password) over the secure connection to authenticate to the server.

Unfortunately, such "secure" connections are often not secure enough to protect passwords, because of the various international legal restrictions that have been placed on the use of encryption. Obviously, secret keys such as passwords should not be sent over weakly encrypted connections. In fact, even a challenge-response protocol which never reveals the password is vulnerable, if a poorly chosen, guessable password is used; an attacker can obtain the (weakly protected) transcript of the challenge-response protocol, then attempt to guess the password, verifying each guess against the transcript.

However, it is possible to protect even badly-chosen passwords against such attacks by incorporating shared-key authentication into the transport-layer security protocol itself. These protocols already involve the exchange of long keys for message authentication, and those same keys can be used (without the legal restraints associated with encryption) to provide very strong protection for shared-key-based

challenge-response authentications, provided that the mechanism used cannot be diverted for use as a strong encryption method. This latter requirement makes it essential that the shared-key-based authentication occur at the protocol level, rather than above it (as is normally the case today), so that the implementation can carefully control use of the long authentication key.

3. Protocol Additions

Starting from SSL version 3.0 notation and formats, the following three new HandshakeTypes are added, and included in the Handshake message definition:

shared_keys(30), shared_key_request(31), shared_key_verify(32)

A new CipherSuite is also included, to allow the client to signal support for shared-key authentication to the server:

TLS_AUTH_SHARED_KEY = {x01, x01};

The client's inclusion of this CipherSuite is independent of other listed CipherSuites, and simply indicates to the server the client's support for shared-key authentication.

3.1 SharedKeys message

The SharedKeys message has the following structure:

```
struct {
     DistinguishedName auth_services_client<1..65535>;
} SharedKeys;
```

This optional message may be sent by the client immediately following the ClientHello message; in fact, if sent, it is actually enclosed within the ClientHello message, immediately following the last defined field of the ClientHello message. (For forward compatibility reasons, the SSL 3.0 ClientHello message is allowed to contain data beyond its defined fields, and because there is no ClientHelloDone message, the server cannot know that an extra message follows the ClientHello unless it is actually included in the ClientHello message itself. A server that does not support shared-key authentication will simply ignore the extra data in the ClientHello message.) Although enclosed within the ClientHello, the SharedKeys message retains the normal structure and headers of a Handshake message.

The SharedKeys message contains a list of distinguished names of authentication services to which the client is willing to authenticate. This list need not be exhaustive; if the server cannot find an acceptable authentication service from the list in the SharedKeys message, then the server is free to reply with a list of acceptable

services in a subsequent SharedKeyRequest message.

In cases where pass-through authentication is used, this message allows clients to be able to notify servers in advance of one or more authentication services sharing a key with the client, so that the server need only fetch (or use up) a challenge from a single service for that client. This message may also be useful in non-pass-through situations; for example, the client may share several keys with the server, associated with identities on different systems (corresponding to different "authentication services" residing on the same server). If a server receives a SharedKeys message, then any subsequent SharedKeyRequest message can contain a single authentication service selected from the client's list.

Note that sending a SharedKeys message does not in itself normally reveal significant information about the client's as-yet-unspecified identity or identities. However, if information about the set of authentication services supported by a particular client is at all sensitive, then the client should not send this message.

3.2 SharedKeyRequest message

The SharedKeyRequest message has the following structure:

```
struct {
         DistinguishedName auth_service_name;
         opaque display_string<0..65535>;
         opaque challenge<0..255>;
} AuthService;

struct {
     AuthService auth_services_server<1..65535>;
} SharedKeyRequest;
```

This optional message may be sent immediately following the server's first set of consecutive messsages, which includes the ServerHello and (possibly) the Certificate, CertificateRequest and ServerKeyExchange messages, but before the ServerHelloDone message. The auth_services_server field contains a list of distinguished names of shared-key authentication services by which the client can authenticate. The challenge field accompanying each authentication service name contains an optional extra authentication challenge, in case the server needs to obtain one from an authentication service for pass-through authentication. If none is required, then it would simply be an empty (zero-length) field. Similarly, the display_string field may contain information to be used (displayed to the user, for example) during authentication, if needed; its interpretation is left to the implementation.

3.3 SharedKeyVerify message

The SharedKeyVerify message is sent in response to a SharedKeyRequest message from the server, at the same point at which a CertificateVerify message would be sent in response to a CertificateRequest message. (If both a CertificateRequest and a SharedKeyRequest are sent by the server, then the client may respond with either a CertificateVerify message or a SharedKeyVerify message. Only one of the two messages is ever sent in the same handshake, however.) The SharedKeyVerify message has the following structure:

```
struct {
     AuthService auth_service;
     opaque identity<1..65535>;
     opaque shared_key_response<1..255>;
} SharedKeyVerify;
```

The value of auth_service must be identical to one of the AuthService values on the list in SharedKeyRequest.auth_services_server. If the client does not share a key with any of the authentication services listed in the SharedKeyRequest message (and cannot supply a certificate matching the requirements specified in the accompanying CertificateRequest message, if one was sent), then the client returns a "no certificate" alert message (in its normal place in the protocol).

The format of the identity field is left to the implementation, and must be inferable from the accompanying value of auth_service. The value of shared_key_response is defined as

```
SharedKeyVerify.shared_key_response
     hash (auth_write_secret + pad_2 +
            hash (auth_write_secret + pad_1 + hash (handshake_messages)
                  + SharedKeyVerify.auth_service.auth_service_name
                  + SharedKeyVerify.auth_service.display_string
                  + SharedKeyVerify.auth_service.challenge
                  + SharedKeyVerify.identity + shared_key) )
```

Here "+" denotes concatenation. The hash function used (hash) is taken from the pending cipher spec. The client_auth_write_secret and server_auth_write_secret values are obtained by extending the key_block by CipherSpec.hash_size bytes beyond the server_write_key (or the server_write_IV, if it is derived from key_block as well), and using this extended portion as the client_auth_write_secret value. (Only the client_auth_write_secret is used, since only the client ever sends a SharedKeyVerify message.) The value of handshake_messages is the concatenation of all handshake messages from the first one sent up to (but not including) the shared_key_verify message. The pad_1 and pad_2 values correspond to the ones used for MAC computation in the application_data message. The fields from the SharedKeyVerify message are input with their length prefixes included.

4. Normal Authentication

A shared-key-based client authentication may proceed as follows: the client includes the TLS_AUTH_SHARED_KEY CipherSuite in its list of CipherSuites in its ClientHello message. It also may or may not send a SharedKeys message along with the ClientHello message, listing the authentication services with which the client shared a key for authentication purposes. In any event, the server sends a SharedKeyRequest handshake message following the ServerHello and accompanying messages containing a list of names of one or more authentication services; if a SharedKeys message was sent, then this list will contain a single choice from the client's SharedKeys message. The client, on receiving the SharedKeyRequest message, selects an authentication service from the server's list (if more than one is offered) and constructs the appropriate authentication response as described above, sending it back, along with its identity and choice of authentication service, in a SharedKeyVerify handshake message. The server itself also constructs the correct authentication response using the known shared key, and checks it against the one provided by the client. The authentication is successful if the two match exactly. Note that if the shared key is password-based, then it would typically be derived from the password using a one-way cryptographic hash function, rather than being the password itself, so that the original password need not be remembered by anyone but the client.

5. Pass-through Authentication

In some circumstances, it is preferable for shared keys to be stored in one place (a central, well-protected site, for instance) while servers that actually communicate with clients are elsewhere (possibly widely distributed, but maintaining secure connections to the central shared-key server). One of the advantages of the shared-key authentication method proposed here is that it allows "pass-through" authentication by a third party, if the server accepting the public-key key exchange and the server sharing the key with the client happen to be different. (The use of a separately derived authentication key in the response computation makes this possible.)

Pass-through authentication might work as follows: The server would either collect random challenges in advance from its authentication services, or request them as needed. (If the client sends a SharedKeys message, then the server can select an authentication service from the client's list, and obtain a challenge from that service alone.) Assuming that the client indicates support for shared-key authentication by including the TLS_AUTH_SHARED_KEY CipherSuite in its list, the server would then send a list of one or more authentication services and associated challenges in a SharedKeyRequest message. The client would then select an authentication service (if more than one is offered), compute the correct authentication response using the above proposed formula, and send it to the server in a SharedKeyVerify message.

The server, on receiving a response from a client, would pass it through to the authentication service, along with the values necessary to recalculate it: the client_auth_write_key, the hash of all the handshake messages and the identity field from the certificate verify message. The authentication service would then use the values provided, along with the secret key it shares with the client and the challenge it supplied, to reconstruct the correct value of the response. If this value exactly matches the one provided by the server, then the authentication would succeed; otherwise it would fail.

References

[1] K. Hickman and T. Elgamal, "The SSL Protocol", Internet Draft <draft-hickman-netscape-ssl-01.txt> (deleted), February 1995.

[2] A. Freier, P. Karlton and P. Kocher, "The SSL Protocol Version 3.0", Internet Draft <draft-freier-ssl-version3-01.txt>, March 1996.

[3] J. Benaloh, B. Lampson, D. Simon, T. Spies and B. Yee, The PCT Protocol", Internet Draft <draft-benaloh-pct-00.txt>, November 1995.

Author's Address

Daniel Simon <dansimon@microsoft.com>

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
Phone: (206) 936-6711
Fax: (206) 936-7329

Draft-ietf-tls-passauth-00.txt
November 20, 1996
Expires: May 25, 1997