Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

I. Miers
M. Green
Johns Hopkins University
E. Rescorla
Mozilla

February 14, 2014

Short Authentication Strings for TLS
draft-miers-tls-sas-00

Abstract

TLS and DTLS connections generally rely on a PKI, a shared secret, or endpoint fingerprints for endpoint authentication. This document describes an authentication mechanism which instead generates a "short authentication string" (SAS) as an emergent property of the connection. The SAS can then be verified via an external channel in order to authenticate the connection.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

TLS [TLS12] and DTLS connections generally rely on a PKI, a shared secret, or endpoint fingerprints for endpoint authentication. This document describes an authentication mechanism which instead generates a "short authentication string" (SAS) as an emergent property of the connection. The SAS can then be verified via an external channel in order to authenticate the connection.

While a fingerprint can be used for authentication (and is used in SSH), it is too long to be conveniently read and compared by two users. If a predictable subset of the fingerprint is compared (e.g., the first or last bits) an attacker can create a fingerprint which just matches that subset. The mechanism described by this document is based on fingerprints but compares a small number of bits derived from the fingerprint and randomness generated by both endpoints, thus requiring an attacker to match the entire fingerprint (which is too long to be feasible) in order to produce a low probability of detection. In order to compute the SAS, the endpoints run a "coin flip" protocol to generate a short shared bitstring which is not under the control of either endpoint. The bitstring is then used, along with the TLS fingerprint, to derive a set of bits that are mapped to a SAS.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Background: Coin-Flipping Protocols

The general pattern of a coin-flipping protocol is shown below:

```
        Alice             Bob

        H(R_a) ------------->
        <----------------- R_b
        R_a ---------------->
```

Alice and Bob each generate a large random number R_a and R_b. Alice (who speaks first) computes a commitment to R_a by hashing R_a and sends it to Bob. Bob then sends R_b to Alice and finally then Alice reveals R_a to Bob. Bob then verifies that R_a matches the hash Alice sent in the first message and then each side computes the shared value S = R_a XOR R_b.

4. Protocol Definition

4.1. Coin Flipping

We map the coin flipping messages to TLS using a TLS extension and a new handshake message, as shown below.

```
      Client                                               Server

      ClientHello + SASXtn        -------->

                                                ServerHello + SASXtn
                                                        Certificate*
                                                  ServerKeyExchange*
                                                 CertificateRequest*
                                  <--------          ServerHelloDone
      Certificate*
      ClientKeyExchange
      CertificateVerify*
      SASShare
      [ChangeCipherSpec]
      Finished                    -------->                 SASShare
                                                  [ChangeCipherSpec]
                                  <--------                 Finished
      Application Data            <------->         Application Data
```

Each side generates a 512-bit cryptographically random value R_client and R_server.

The SASXtn is defined as follows:

```
          struct {
              opaque digest<255>;
          } SASExtension;
```

The client's SASXtn is a zero-length value indicating the client's desire to do the SAS handshake. The server's SASXtn is a digest of R_server using the Hash defined for the Finished message in Section 7.4.9 of [TLS12].

The SASShare structure is defined as follows:

```
          struct {
              opaque share[64];
          } SASShare;
```

"share" is the raw byte value of R_client or R_server, as appropriate.

4.2. Computing the raw SAS bits

The SAS bits are computed as follows:

1. If you are the client, verify that the server's R_server matches their SASExtension value. If not, abort the handshake with error handshake_failure
2. Compute R_shared = R_client ^ R_server
3. If both endpoints have certificate fingerprints compute fingerprints=fingerprint_server | fingerprint_client. If only the server has a fingerprint, compute fingerprints=fingerprint_server.
4. Compute SAS_bits as HASH(fingerprints||R_shared) using the Hash defined for the Finished message in Section 7.4.9 of [TLS12]

4.3. Computing the SAS String

The application should map the first 15<n<len(fingerprints) bits of SAS_bits to some set of words or symbols defined for the application. One option is the PGP word list. For a spoken language agnostic solution, symbols could be use.

5. Security Considerations

Implementations MUST use fresh, random R_client and R_server values for each TLS handshake.

Implementations MUST ensure their share of the coin flip remains secret until after the TLS session key is established.

Applications SHOULD abort if the SAS strings do not match.

Applications SHOULD abort after multiple failed TLS handshakes and notify the user. Failure to do so will allow an attacker multiple attempts to guess a SAS. They will succeed after a few thousand attempts.

6. Normative References

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

[TLS12] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

Authors' Addresses

Ian Miers
Johns Hopkins University

Email: imiers@cs.jhu.edu

Matthew Green
Johns Hopkins University

Email: mgreen@cs.jhu.edu

Eric Rescorla
Mozilla
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com