Internet Engineering Task Force C. Paasch
Internet-Draft Apple, Inc.
Intended status: Experimental A. Ford
Expires: November 28, 2016 Pexip
May 27, 2016

# Application Layer Authentication for MPTCP
draft-paasch-mptcp-application-authentication-00

## Abstract

Multipath TCP (MPTCP), described in [3], is an extension to TCP to provide the ability to simultaneously use multiple paths between hosts.

MPTCP currently specifies a single authentication mechanism, using keys that are initially exchanged in the clear. There are application-layer protocols that may have better information as to the identity of the parties and so is able to better provide keying material that could be used for the authentication of future subflows.

This document specifies "application layer authentication" for Multipath TCP, an alternatively negotiated keying mechanism for MPTCP.

Copyright Notice



Table of Contents



# 1. Introduction

The MPTCP handshake serves multiple purposes. First, hosts discover their peer's support of MPTCP. Second, each host announces a key that will be tied to this MPTCP session. The key also serves multiple purposes. First, the derivate of the key is being used as a token-identifier for the MPTCP connection. This derivate is a truncated hash of the key. Second, another truncated hash of the key serves as the initial data sequence number. And third, the key itself is used as an authenticator to prove that the host behind the IP-address used to establish new subflows is indeed the one that participated in the handshake of the initial subflow.

In the following we explain the shortcomings of this exchange and how they impact the deployment of MPTCP.

### 1.1. Key in plaintext

The key-exchange happens during the handshake of the initial subflow. RFC 6824 specifies that this exchange happens in plaintext. As has been noted in RFC 7430, an eavesdropper on the initial handshake is thus able to learn the keys used in this MPTCP session. This allows him to generate the session's tokens and data sequence numbers, enabling him to effectively hijack the MPTCP session by creating a subflow with a different IP-address. The attacker will be able to generate a valid HMAC as he has full knowledge of the keys of this MPTCP session.

To enhance MPTCP's security, it would be beneficial to not reveal MPTCP's keys in plaintext on the wire.

### 1.2. Token generation

The token is a truncation of the 32 most significant bits of the SHA-1 of the key. The key must be a random number of sufficient entropy to be used as part of the authentication mechanism, and thus a host has no control over the token as it is generating the key for the MPTCP-session. This has some implications on the deployability of MPTCP, outlined hereafter.

#### 1.2.1. Hash collision

Due to the nature of the token-generation, the 32-bit token might collide with another already existing MPTCP session. While a 32-bit token collision should be very rare on client devices, a busy server (with potentially tens of millions of active MPTCP connections) will have a very high probability of a token collision.

Upon such a collision, the server needs to generate a new cryptographically secure 64-bit key, and derive the token through a SHA-1 computation upon which he finally can verify the uniqueness of the token. If a collision happened again, the server has to start anew. This process imposes a computation overhead and complexity upon the server and impacts the scalability compared to regular TCP. Allowing a server to generate a token in such a way that uniqueness can be achieved easily would be beneficial for the scalability and deployment of MPTCP.

#### 1.2.2. Derive information from the token

As the token is a truncated hash of the key, it is entirely of a random nature. As has been shown in [5], this brings several deployment challenges in large server farms. In particular, the layer-4 load balancers in front of this server farm need to maintain MPTCP-specific state in order to map a token to the server.

The token can be looked at as a route-identifier, as it allows the server to associate the incoming SYN+MP_JOIN with an existing MPTCP-session. However, the random nature of the token does not allow a load balancer in the middle to do the same without having to maintain MPTCP-specific state.

If the token can be generated in such a way that it carries the required routing information in such a way that it can be deciphered by all the trusted parties in the server farm deployment, large-scale deployment of MPTCP would be simplified.

In the following we suggest an alternative handshake that allows MPTCP to increase its security by leveraging an external key-exchange and thus benefit from the security provided by protocols like TLS. As a side-effect of this approach, the token also can be exchanged in a more flexible way, addressing the above identified issues with the token generation.

## 2. Proposed Technical Changes

### 2.1. MP_CAPABLE Changes

To resolve the issues identified in the previous section, this proposal separates the key handling for security (i.e. the method for protecting new subflow exchanges) from the token exchange. This means that:

- o Key exchange is handled in the application layer

- o Meaning can be exchanged in the token, and a custom generation method can be used, as it is decoupled from keying material

This specification allocates the 'G' bit from the flags of MP_CAPABLE as an alternative security mechanism - "handled by application layer". In this case, the MP_CAPABLE exchange will send and receive tokens rather than keys.

When the 'G' bit is set to 1, this implies support for this new mechanism, and the MP_CAPABLE exchange will operate as follows. The tokens take the place of the keys in the MP_CAPABLE exchange, but otherwise the exchange remains very similar. This exchange still maintains support for stateless servers. Note that this now means that tokens are 64 bits in length.

```
                         1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +---------------+---------------+-------+-------+---------------+
    |     Kind      |    Length     |Subtype|Version|A|B|C|D|E|F|G|H|
    +---------------+---------------+-------+-------+---------------+
    |                   Option Sender's Token (64 bits)             |
    |                      (if option Length > 4)                   |
    |                                                               |
    +---------------------------------------------------------------+
    |                  Option Receiver's Token (64 bits)            |
    |                      (if option Length > 12)                  |
    |                                                               |
    +-------------------------------+-------------------------------+
    |  Data-Level Length (16 bits)  |  Checksum (16 bits, optional) |
    +-------------------------------+-------------------------------+
```

Figure 1: Proposed Multipath Capable (MP_CAPABLE) Option

The MP_CAPABLE option is carried on the SYN, SYN/ACK, and ACK packets that start the first subflow of an MPTCP connection, as well as the first packet that carries data, if the initiator wishes to send first. The data carried by each option is as follows, where A = initiator and B = listener.

- SYN (A->B): only the first four octets (Length = 4).

- SYN/ACK (B->A): B's token for this connection (Length = 12).

- ACK (no data) (A->B): A's token followed by B's token (Length = 20).

- ACK (with first data) (A->B): A's key followed by B's key followed by Data-Level Length, and optional Checksum (Length = 22 or 24).

The contents of the option is determined by the SYN and ACK flags of the packet, along with the option's length field. For the diagram shown in Figure 1, "sender" and "receiver" refer to the sender or receiver of the TCP packet (which can be either host).

If the sender of the initial SYN supports both SHA-1 (as specified in [3]) and application-layer, it can set both G and H bits to "1". The sender of the SYN/ACK can then make a decision as to which mode to support, and selects only one of those bits in the SYN/ACK.

## 2.2. MP_JOIN Changes

The MP_JOIN exchange remains almost the same:

```
          Host A                                  Host B
   ------------------------                     ----------
   Address A1    Address A2                     Address B1
   ----------    ----------                     ----------
        |             |                              |
        |             |   SYN + MP_JOIN(Token-B, R-A)  |
        |             |------------------------------->|
        |             |<-------------------------------|
        |             | SYN/ACK + MP_JOIN(HMAC-B, R-B) |
        |             |                                |
        |             |     ACK + MP_JOIN(HMAC-A)      |
        |             |------------------------------->|
        |             |<-------------------------------|
        |             |             ACK                |

   HMAC-A = HMAC(Key=(Key-A+Key-B), Msg=(R-A+R-B))
   HMAC-B = HMAC(Key=(Key-B+Key-A), Msg=(R-B+R-A))
```

Figure 2: Example Use of MP_JOIN

However, the token presented is now 64 bits. The key used in the HMAC exchange here is provided by the application layer. Otherwise, there are no other changes to the handshake. Note, however, that an MP_JOIN message cannot be sent until the application layer protocol has determined that the key exchange has completed.

Depending on the key-exchange protocol that is in use at the application layer, it may be that the client already knows the key, while the server is not yet aware of it. In that case the server might receive SYN+MP_JOIN with a valid token, but the MPTCP-state on the server has not yet been populated with the key. The server must silently drop in that case the SYN+MP_JOIN. The client will retransmit its SYN+MP_JOIN and eventually the application on the server will have populated the MPTCP-state with the key.

## 2.3. Data Sequence Number Changes

The Initial Data Sequence Number for each host involved in an MPTCP connection is, by [3], derived from the SHA-1 hash of the key. If application-layer authentication is selected, the IDSN MUST instead be derived from the most-significant 64 bits of the SHA-1 hash of the token.

### 2.4. MP_FASTCLOSE Changes

MP_FASTCLOSE is the other method that uses the key in [3]. Given there is no knowledge as to a potential key's sensitivity, it can no longer be said that a key should be sent here. Instead, a truncation of the 64 most-significant bits of the SHA-1 hash [4] of the key should be used.

## 3. Security Considerations

This draft is proposing a mechanism that would allow an application-layer protocol to provide security, rather than relying on a cleartext exchange of the keys. As such, this document itself does not introduce any additional security concerns, but provides a mechanism by which additional security could be added to the MPTCP handshake, depending on the authentication method used at the application layer.

## 4. IANA Considerations

This document would update the "MPTCP Handshake Algorithms" sub-registry under the "Transmission Control Protocol (TCP) Parameters" registry, based on the flags in MP_CAPABLE, to add the following algorithm:

| Flag Bit | Meaning | Reference |
|---|---|---|
| G | Application-layer Authentication | This document |

Table 1: MPTCP Handshake Algorithms

## 5. References

### 5.1. Normative References

[1] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[3] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-05 (work in progress), January 2016.

[4] National Institute of Science and Technology, "Secure Hash Standard", Federal Information Processing Standard (FIPS) 180-3, October 2008, <http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf>.

## 5.2. Informative References

[5] Paasch, C., Greenway, G., and A. Ford, "Multipath TCP behind Layer-4 loadbalancers", draft-paasch-mptcp-loadbalancer-00 (work in progress), September 2015.

## Authors' Addresses

Christoph Paasch
Apple, Inc.
Cupertino
US

EMail: cpaasch@apple.com

Alan Ford
Pexip

EMail: alan.ford@gmail.com