MPTCP Working Group
Internet-Draft
Intended status: Experimental
Expires: April 30, 2015

O. Bonaventure
UCLouvain

October 27, 2014

MPTLS : Making TLS and Multipath TCP stronger together
draft-bonaventure-mptcp-tls-00

Abstract

Multipath TCP and the Transport Layer Security (TLS) include several techniques that improve the reliability and the security of data transfers. In this document we propose Multipath TLS (MPTLS), a tighter coupling between TLS and Multipath TCP that provides improved security and reliability in the presence of adversaries. MPTLS would the resilience of existing TLS applications to attacks. It could also serve as a basis for the TCP extension that is being discussed within the TCPINC working group to provide unauthenticated encryption and integrity protection of TCP streams.

Table of Contents

1. Introduction

Transport Layer Security (TLS) [RFC5246] is an application layer protocol that allows to encrypt and authenticate the data exchanged between applications running on different hosts. Various documents have analysed the security of the TLS protocol from different viewpoints [I-D.sheffer-uta-tls-attacks]. Over the years, various extensions to TLS have been developed and deployed. Besides attacks on the cryptographic algorithms or their implementations, TLS is also vulnerable to some forms of attacks that affect the underlying TCP protocol [RFC0793].

TCP, by default, does not include any cryptographic technique to authenticate/encrypt data. Three types of solutions have been proposed to improve the security of TCP. A first approach is to tune the TCP stack to prevent some packet injection attacks. Examples of this approach may be found in [RFC5927] or [RFC5961]. Another approach is to add authentication to the TCP protocol. The TCP MD5 option defined in [RFC2385] was the first example of such an

approach. The TCP-AO option defined in [RFC5925] and [RFC5926] is a more recent example. These two solutions were designed to protect long-lived TCP connections such as BGP sessions from packet injection attacks. They assume that a secret is shared among the communicating hosts. A third approach is to extend TCP to include the cryptographic techniques directly inside the TCP stack [I-D.bittau-tcp-crypt].

On the other hand, Multipath TCP [RFC6824] is a recent extension to TCP that allows to use several interfaces (or paths) to transmit the packets that belong to a single connection. It achieves this by managing several TCP connections (called subflows) for each Multipath TCP session. With Multipath TCP, the number of subflows associated to a given session may change dynamically. This is typically the case for mobile hosts that change of IP address, but this ability of Multipath TCP to adapt to changes in the underlying TCP subflows can also be used to improve the reaction to various packet injection attacks. Thanks to its ability to manage subflows, Multipath TCP can cope with various types of attacks and errors that affect the TCP stack and cannot easily be recovered at the application layer without implementing a session layer protocol.

In this document, we propose a high level design for Multipath TLS (MPTLS). Multipath TLS integrates Multipath TCP and TLS together to provide enhanced security for the applications. This integration can be beneficial for security sensitive applications that already rely on TLS. It could also address the requirements of the TCP extension being developed within the TCPINC working group.

This document is organised as follows. Section Section 2 describes the attacks that can affect TCP or TLS. Section Section 3 provides a high-level overview of the proposed architecture. Section Section 4 describes the required changes to the Multipath TCP protocol while section Section 5 explains the proposed modifications to the TLS protocol. Section Section 6 discusses the interactions with middleboxes and section Section 7 the security considerations.

2. Attacks considered

Any security protocol must be designed with the set of attacks that need to be prevented in mind. For this work, we consider three different types of attackers :

- a completely off-path attacker that cannot capture any of the packets exchanged by the communicating hosts but is able to inject spoofed packets. We call this attacker the off-path attacker.

- an attacker that sits on (at least one of) the paths used to exchange packets between the communicating hosts at the beginning of the connection but later moves away from this path. We assume that this attacker is able to capture the packets exchanged and send spoofed packets but cannot modify the packets sent by the communicating hosts. We call this attack the partially on-path attacker.

- an attacker that is always on the path between the communicating hosts. This attacker is able to capture the packets exchanged by the communicating hosts and modify them. We call this attacker the on-path attacker. This attacker could also be a middlebox that sits on one of the paths used by the communicating hosts.

The off-path attacker is the simplest type of attacker in our taxonomy. This attacker can inject spoofed packets inside existing TCP connections. To inject a packet so that it is accepted inside an existing TCP connection, the attacker needs to guess the IP addresses of the communicating hosts, the port numbers (one is usually well-known) and sequence numbers and acknowledgements that fit inside the receive window. Several techniques have been defined [RFC6528],[RFC6056], [RFC5961], [Bellovin] to cope with such attacks and some have been implemented [I-D.ietf-tcpm-tcp-security]. The off-path attacker can try to inject either packets containing data or control packets (i.e. packets carrying the RST or the FIN flags). For both control and data packet injection attacks, a successful attack results in the termination of the affected TCP connection.

Multipath TCP is by design less vulnerable than regular TCP to such attacks since it uses 64 bits data sequence numbers. It should be noted that the utilisation of TLS on a TCP connection does not increase its reaction against this form of attack. If the underlying TCP connection is reset due to an off-path attack, the TLS session is reset as well. The standard solution to cope with these off-path attacks is to authenticate the packets that are exchanged at either the TCP or the IP layer. The TCP-MD5 option, defined in [RFC2385], allows to authenticate the packets exchanged by the communicating hosts. This prevents the packet injection attacks discussed above since the receiving host can verify the validity of all received packets and easily reject those sent by the attacker. The recently proposed TCP-AO option [RFC5925] generalises this technique by allowing different types of hash functions and supporting key rollover techniques. Unfortunately, both TCP-MD5 and TCP-AO suffer from an important drawback. They assume that the communicating hosts have a shared secret. This shared secret is required because both techniques aim at authenticating all packets including the initial packets of the three-way handshake. While TCP-MD5 and TCP-AO can be used to secure the long-lived TCP connections used by BGP sessions

between Internet routers, they cannot be easily used to secure regular Internet traffic. Furthermore, combining TCP-AO with Multipath TCP would consume a large fraction of the TCP option space in the packets and would prevent the use of other important TCP extensions such as TCP-SACK. For these reasons, TCP-AP [RFC5925] is not a suitable solution.

The second type of attacker that we consider is the on-path attacker. This is the most powerful attacker. TCP by itself is not protected against such attackers that can modify the packets exchanged on a TCP connection. Some of the deployed middleboxes operate like on-path attackers since they modify the contents of TCP packets. Typical examples are the NAT devices that change IP addresses and port numbers. Application Level Gateway running on NATs sometimes also need to modify the packet payloads or TCP normalisers that re-segment TCP packets [Normaliser] are other examples. Furthermore, studies have also shown that some middleboxes may generate packets to terminate TCP connections for various reasons [RFC3360]. We need to distinguish two types of attacks from these on-path attackers of middleboxes :

- attacks that modify the packet payload without terminating the connection

- attacks where the middlebox terminates the affected TCP connection

The first type of attack is transparent for TCP. TCP does not detect the attack since the checksum has been modified by the attacker and delivers the modified payload. Multipath TCP, thanks to its DSS checksum, can detect a payload modification performed by a middlebox that understand TCP but not Multipath TCP. In this case, it falls back to regular TCP to preserve the connectivity between the communicating hosts. It should be noted that it would be possible to design a middlebox that modifies the payload of Multipath TCP packets in a way that cannot be detected by Multipath TCP (e.g. if the middlebox updates the DSS checksum after having modified the payload). If the middlebox decides to close the connection by using regular TCP RST or FIN flags, then Multipath TCP can react by reestablishing a new subflow (possibly via another path) to preserve the connection despite of the attack. The ability to manage several subflows is an important technique that allows Multipath TCP to react to attacks. While this reaction is effective against currently deployed TCP middleboxes, it is possible to design Multipath TCP aware middleboxes that inject specific Multipath TCP packets (e.g. by using the FAST_CLOSE option whose security relies on the keys exchanged during the initial handshake) to actively terminate Multipath TCP connections.

On the other hand, TLS uses cryptographic techniques that enable the secure of keys that allow to authenticate and encrypt the records exchanged over the (Multipath) TCP connection. With appropriate cryptographic techniques and a PKI that prevents MITM attacks, it is possible to negotiate keys through an on-path attacker without enabling this attacker to derive the security keys. These security keys can then be used to authenticate and encrypt the records that are exchanged. Unfortunately, while the current TLS specification and implementations verify the authenticity of the received records from the derived secret keys, they react to an authentication failure by releasing the underlying TCP connection and alerting the application. This implies that an on-path attack will result in a denial of service attack for TLS applications.

3. High-level architecture

At a high level, MPTLS integrated TLS and Multipath TCP together as shown in the figure below. The application interacts with the TLS implementation through the existing API without any change. This ensures the backward compatibility with existing applications.

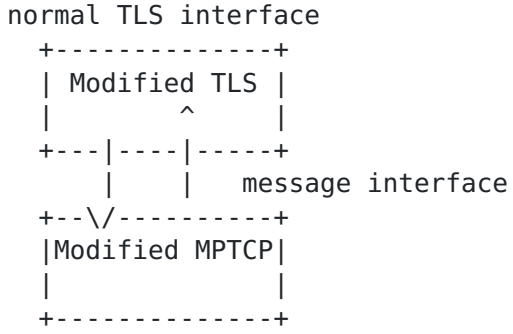


Figure 1: Simplified architecture

We modify the TLS sublayer and keep in this layer the following functions that are already part TLS :

- secure handshake and key negotiation
- transmission and reception of the TLS records
- encryption/decryption of the TLS records

The TLS techniques that allow to authenticate TLS records are moved to the modified Multipath TCP sublayer. For this, we leverage the recently proposed encrypt-then-mac technique [RFC7366]. This is motivated by two reasons. First, Multipath TCP already verifies the

received data by using its optional DSS checksum. We replace this checksum with a cryptographic MAC authentication that has strong security properties. Second, if Multipath TCP receives a TLS record with an invalid MAC it can simply discard the data and wait for its retransmission or perhaps terminate the affected TCP subflow and create a new one to retransmit the data. This is much better from a security viewpoint than the current approach used by TLS that terminates the TLS session as soon as a record with an invalid MAC has been received.

To enable Multipath TCP to correctly compute the MAC of each TLS record, we modify the interface between TLS and Multipath TCP. While regular TCP provides a bytestream service to TLS, our modified Multipath TCP provides a message mode service. With this service, Multipath TCP transports a sequence of TLS records. All these records are delivered in sequence (possibly after some retransmissions by the underlying layer) to the TLS sublayer at the receiver.

As explained earlier, TLS negotiates the keys that are used to encrypt and authenticate the TLS records. Multipath TCP also needs keys to authenticate the establishment of subflows. Instead of exchanging the Multipath TCP keys in clear as defined in [RFC6824], we leverage the technique proposed in [RFC5705] that allows to derive additional keys from the MasterSecret negotiated during the secure TLS key exchange. The required keys are then passed to Multipath TCP as proposed in [I-D.paasch-mptcp-ssl] to secure the establishment of new subflows and also authenticate the transported TLS records. A drawback of this approach is that the keys required to authenticate the establishment of subflows are only available at the end of the TLS key exchange. Thus, this key exchange can only be performed over the initial subflows.

We modify the service model of Multipath TCP when integrating it with TLS. Instead of providing a bytestream service, Multipath TCP provides an (authenticated) message-mode service. Each TLS record is sent as a single message by Multipath TCP. More precisely, the following workflow is used :

- the application generates a block of up to 2^14 bytes of plain text
- the TLS layer encrypts the plain text and adds the TLS record header
- the TLS layer passes the TLS record to Multipath TCP as a message

- Multipath TCP maps the entire TLS record via one DSS option onto a single subflow, computes the MAC and adds the DSN option
- the corresponding packets are reliably transported through the network and delivered to the Multipath TCP layer at the destination
- Once all packets have been correctly received at the destination, the MAC is verified. If the verification succeeds, the TLS record (without the authenticated MAC) is passed to the TLS layer that extracts the record header and decrypts the ciphertext to retrieve the plaintext and pass it to the application.

Additional details about the required modifications to TLS and Multipath TCP are discussed in sections Section 5 and Section 4.

4. Required modifications to Multipath TCP

Several modifications are required inside Multipath TCP to integrate it with TLS as proposed in the previous sections.

The high level solution described above requires some modifications to Multipath TCP to support the integration of TLS with Multipath TCP :

- Multipath TCP must be modified to support a message-mode service (limited to messages of up to 2^16 bytes) instead of the default bytestream service
- Multipath TCP must be able to utilize the keys generated by TLS to authenticate the messages through a MAC algorithm and the establishment of new subflows
- optionally an improved API to enable a better exchange of information between TLS and Multipath TCP

4.1. The three-way handshake

[RFC6824] uses the three way handshake to negotiate the utilisation of Multipath TCP through the utilisation of the MP_CAPABLE option and also to exchange the keys that are used to both identify the Multipath TCP connection and authenticate the additional subflows. Since in MPTLS the keys will be provided by TLS, there is no need to exchange keys during the three way handshake. However, the three-way handshake is also used to exchange the tokens that identify the Multipath TCP connection on each host and the initial data sequence numbers in both directions. To identify the Multipath TCP

connection, we place the Sender's token in the MP_CAPABLE option of the SYN and SYN+ACK segments.

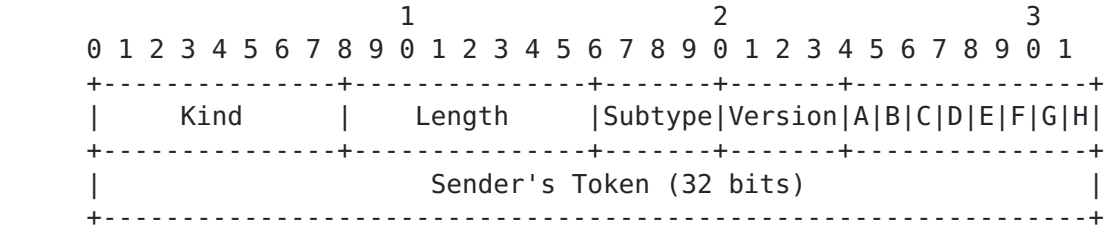


Figure 2: MP_CAPABLE option

This variant of the MP_CAPABLE option is shorter than the MP_CAPABLE option defined in [RFC6824]. The two could be distinguished by relying on the length of the MP_CAPABLE option or based on the version number or one of the bits of the MP_CAPABLE option.

The IDSN of each host should be generated by using the hash function associated with this version of Multipath TCP as IDSN=Hash(LocalToken||RemoteToken) where Local Token is the locally generated token and RemoteToken the token that has been generated by the remote host. An alternative could be to add a 32 bits random number in the MP_CAPABLE option and generate the IDSN as proposed in [I-D.paasch-mptcp-lowoverhead]. However, this solution uses 32 more bits of options in the SYN segment.

4.2. Provision of a message-mode service

The second important modification to Multipath TCP is to replace its bytestream service by a message-mode service that is targeted to the needs of the modified TLS sublayer. To distribute the data over different subflows, Multipath TCP relies on the Data Sequence Signal option [RFC6824] whose format is shown below :

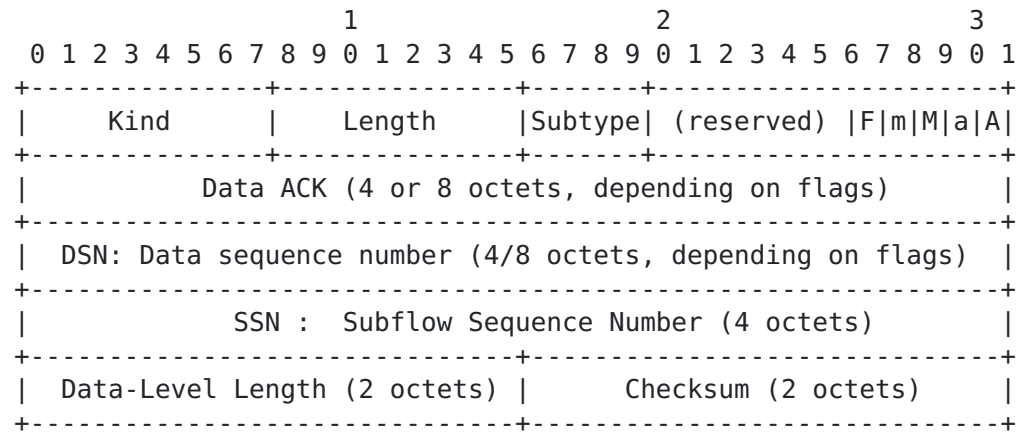


Figure 3: Data Sequence Signal option

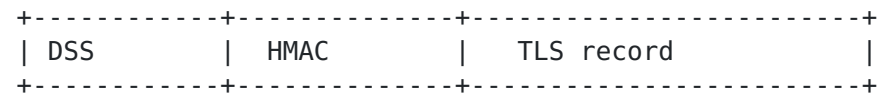
This option allows to map Length bytes from the bytestream of the Multipath TCP connection starting at data sequence number 'DSN' to the subflow sequence number 'SSN'. Since the length field is encoded as a two bytes long unsigned integer, it can be used to map up to 2^16 bytes. This is larger than the maximum size of the TLS plaintext encoded inside a record (2^14 bytes) [RFC5246]. Even with expansion, a TLS ciphertext will never become larger than 2^16 bytes. We cover each TLS record with a single mapping. This implies that a single TLS record can only be mapped onto one subflow. If the TLS session is interactive, then short TLS records will be used anyway. If the TLS session transports a large amount of data, sending entire records over each subflow should not impact the performance. We note that some deployments of TLS already dynamically adapt the length of the TLS records to the activity of the session [Grigorik].

We modify the semantics of the DSS option as follows. First, the Data-level length becomes the size of the TLS record that is transported. Given the constraints imposed by [RFC5246], this record will always be shorter than 2^16 bytes. Second, the DSS checksum is disabled and replaced by a MAC.

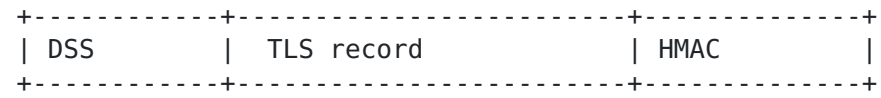
4.3. HMAC authentication

TLS supports a range of authentication techniques that can be negotiated during the TLS handshake. In this first version of the document, we assume that TLS has negotiated a keyed HMAC to authenticate the TLS record. Subsequent versions of this document will analyse other record authentication methods such as [RFC5116]. There are several possible design options to transport the computed HMAC.

At a high level, our objective is to transport the computed HMAC together with the mapped data. As explained in the previous subsection, this data will be transported on the same TCP subflow given that the TLS record is covered by one DSS mapping. These three blocks of information can be transported either as



or



Sending the HMAC before the TLS record could simplify the processing at the receiver, but would force the sender to compute the entire HMAC before transmitting the TLS record. On the other hand, by sending the HMAC after the TLS record, we could enable hardware accelerators to both encrypt the TLS record and compute the HMAC on the fly while transmitting the data.

At this stage, it is too early to opt for one encoding over the other, even if supporting both would create a too complex protocol.

The second design question is how to encode the HMAC. Again, several options are possible.

A first approach would be to rely on the TCP-AO option [RFC5925] and modify it so that it can authenticate all the data covered by the DSS mapping. However, using another (long) TCP option would consume space in the limited TCP option space. Furthermore, TCP-AO was designed with the assumption that each TCP-AO option covers a single TCP segment. Thus, there might be middleboxes that rely on this assumption to accept/reject packets. Modifying the TCP-AO option would likely result in difficulties with some middelboxes.

A second approach is to place the length of the HMAC inside the DSS option, e.g. as a single byte that follows the Data-Level length field. This allows to support various HMAC lengths, including truncated HMAC [RFC6066].

A third approach is to encode the HMAC as a variable length option using the same format as the TCP options, but transport this

information inside the DSS payload, before or after the TLS record that contains the real data. The variable-length TCP options are encoded in [RFC0793] as :

<Kind><Length><Option>

Where Kind and Length are encoded as a single byte. This limits the length of each option at 255 bytes, which is large enough to carry a HMAC. Option Kind=0 defined in [RFC0793] is a special option that does not contain a length information and is used to indicate the end of the option list. The HMAC option would always be transmitted before the TLS record and the null option would terminate the list of options.

The simplest solution appears to be the second approach and sending the HMAC before the TLS record. Since the TLS keys can change during the lifetime of a TLS sessions, MPTLS needs to indicate which key has been used to compute an HMAC. This problem can be solved by associating a key identifier to the keys that are passed by TLS to Multipath TCP and place this key identifier inside the DSS option. The figure below provides the structure of the DSS option used by MPTLS.

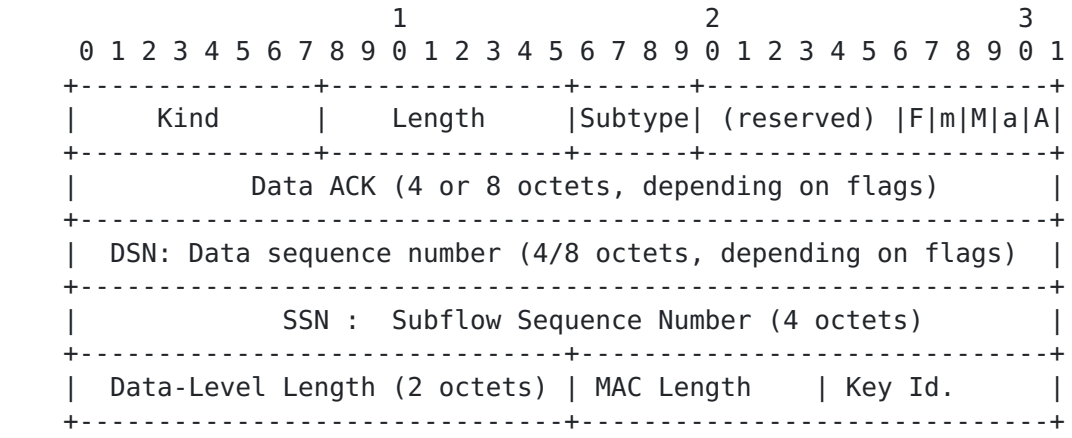


Figure 4: MPTLS Data Sequence Signal option

The semantics of the last two fields of this modified DSS option is the following.

o 'MAC Length' is an 8 bits unsigned integer that specifies the length of the HMAC that follows the DSS option, i.e. the HMAC starts at subflow sequence number 'SSN' and ends at 'SSN+MACLength-1'. A length of 0 indicates that no HMAC has been computed for the TLS record.

- 'Key Identifier' is an 8 bits unsigned integer that indicates the key that was used to compute the attached HMAC. When a TLS session starts, the key identifier is set to zero. It is incremented by one after each renegotiation of the keys. Using key identifiers allows an MPTLS implementation to verify the validate of TLS records that have been generated by using different keys.

The last point to be discussed about the HMAC and the modified DSS option is the data that is covered by the HMAC. MPTLS uses the HMAC to authenticate both the TLS record, i.e. the payload mapped by the DSS, and the DSS option itself. This implies that any modification to one of the fields of the DSS option or to the data part would be detected by the HMAC. It should be noted that in contrast with TCP-AO, MPTLS does not protect the TCP header fields. This choice is motivated by the fact that various middleboxes modify IP and TCP fields, notably the IP addresses, the TCP port numbers, the sequence and acknowledgement numbers. Including these fields in the MPTLS HMAC would prevent any communication through such middleboxes.

The HMAC should also be used to authenticate the following Multipath TCP options that may be exchanged during a Multipath TCP connection :

- ADD_ADDR2 : This option must be authenticated with a HMAC [I-D.ietf-mptcp-rfc6824bis]. If replay attacks inside a single connection are concern, then the option needs to be updated to include a valid data-level sequence number and a valid data level ack

- REMOVE_ADDR : This option must be authenticated with a HMAC. The above comment applies if replay attacks are a concern.

- MP_PRIO : This option must be authenticated with a HMAC. The above comment applies if replay attacks are a concern.

- MP_FASTCLOSE : This option must be authenticated with a HMAC. The current version defined in [RFC6824] includes the key negotiated during the three-way exchange. Placing the MAC key generated by TLS in this option would not be appropriate. A possible solution would be to place the Initial Data Sequence number inside the FAST_CLOSE option and authenticate this number via the HMAC.

- MP_FAIL : This option is used to perform a fallback to regular TCP when middlebox interference has been observed. When MPTCP is used together with TLS, this mechanism should never be triggered and a host should silently ignore any MP_FAIL option that it receives.

The processing of data on a receiving host is slightly modified compared to regular TCP. In Multipath TCP version 1, as defined in [RFC6824], two levels of acknowledgements are used. With MPTLS, data remains acknowledged at the subflow level as soon as it has been received. Selective acknowledgements [RFC2018] can be negotiated to provide additional information about out-of-sequence data at the subflow level.

With MPTLS, data can only be acknowledged at the Data-Sequence level (i.e. through the Data ack field of the DSS option) once it has been received in sequence and the HMAC that authenticates the received TLS record has been validated. If the HMAC authentication fails for one received TLS record, then the corresponding subflow should be terminated with a reason code [I-D.bonaventure-mptcp-rst] that indicates an authentication failure. The TLS record will then be retransmitted over another available subflow or a new subflow will be established to transmit this record. This mode of operation allows Multipath TCP to cope with various types of packet injection attacks without breaking the connection and affecting the TLS layer or the application.

5. Required modifications to TLS

Multipath TCP can be completely transparent to the application since it provides the same socket interface as regular TCP. On the other hand, TLS is a record oriented protocol. Data is encoded in records that are reliably exchanged over the underlying TCP connection. TLS defines two types of records :

- the control records that are used to exchange control information such as the secure handshake messages that negotiate the cryptographic parameters and keys

- the data records that transport encrypted and authenticated data

These two types of records have a variable length and are encoded by using a TLV format specified in [RFC5246]. The TLS protocol defines several types of data records depending on the type of encryption scheme that is used. Current TLS implementations apply a MAC-then-Encrypt approach to transmit data [RFC5246]. This implies that each data record, is first authenticated, e.g. by using a negotiated HMAC algorithm, then the authenticated record is encrypted. Several cryptographers have argued about several security problems with this approach there are ongoing discussions to use Encrypt-then-MAC for the data record [RFC7366]. This technique has better security properties and we build upon it to integrate TLS and Multipath TCP together.

At a high level, an MPTLS connection starts like a regular Multipath TCP connection. The Multipath TCP connection starts with a three-way handshake using the MP_CAPABLE option to negotiate the utilisation of Multipath TCP and exchange the tokens as explained in section Section 4.

Once the three-way handshake has finished, the bytestream is established and TLS can start its key negotiation. All the crypto mechanisms defined in [RFC5246] can be used to negotiate the crypto parameters and keys. In contrast with TCP-AO, this crypto negotiation is performed over an unprotected bytestream as when TLS is used over single-path TCP. In particular, no HMAC is included in the DSS option defined in the previous section.

TLS uses the client_write_MAC_key and server_write_MAC_key to authenticate the data records. We build upon the encrypt-then-mac principle [RFC7366] and place the encryption/decryption function in the TLS layer and the authentication function inside Multipath TCP. As in AEAD algorithms [RFC5116], we assume that two different keys are used for the encryption and the authentication. The encryption keys are generated and stored in the TLS layer. The authentication keys are generated in the TLS layer by using the PRF described in [RFC5246] or through the procedure defined in [RFC5705].

This part of the document describes in more details the modifications to TLS that are required to better integrate with Multipath TCP. The key issues that need to be discussed here are :

- the new format for the TLS records
- the derivation of the keys that are used by TLS and Multipath TCP
- the interactions between TLS and Multipath TCP

5.1. Modifying the TLS record

[RFC7366] defines a TLS record as being composed of :

```
 struct {
          ContentType type;
          ProtocolVersion version;
          uint16 length;
          GenericBlockCipher fragment;
          opaque MAC;
         } TLSCiphertext;
```

Although the length field of this record is encoded as a 16 bits integer, TLS limits the record size to 2^14 bytes at most. Since

MPTLS performs the authentication outside of the TLS record, we need to remove the opaque MAC from the structure of the TLS record. With regular TCP, this record format enables the receiver to retrieve the record boundaries and extract it from the bytestream. In MPTLS, this feature is not required since it provides a message mode service that delivers entire TLS records. For this reason, we also remove the length information from the TLS record. Note that this implies that it will be impossible for MPTLS to fallback to regular TCP if middlebox interference is detected. The modified TLS record becomes :

```
 struct {
          ContentType type;
          ProtocolVersion version;
          GenericBlockCipher fragment;
        } MPTCP_TLSCiphertext;
```

5.2. Key derivation

MPTLS needs to derive more keys than when it is used over regular TCP. Once the TLS handshake has succeeded, the crypto parameters and keys are known by the two communicating hosts. In TLS, [RFC5246], six keys are derived from the Master key :

- client_write_MAC_key[SecurityParameters.mac_key_length]
- server_write_MAC_key[SecurityParameters.mac_key_length]
- client_write_key[SecurityParameters.enc_key_length]
- server_write_key[SecurityParameters.enc_key_length]
- client_write_IV[SecurityParameters.fixed_iv_length]
- server_write_IV[SecurityParameters.fixed_iv_length]

The last two keys are only used with some specific crypto algorithms. We leverage [RFC5705] to derive two additional keys :

- client_write_MPTCP_key[SecurityParameters.mac_key_length]
- server_write_MPTCP_key[SecurityParameters.mac_key_length]

The *write_MAC_key and *_write_MPTCP_key keys are derived by TLS and immediately passed to the Multipath TCP sublayer by using a similar technique as [I-D.paasch-mptcp-ssl]. The *write_MAC_key keys are used to authenticate the TLS records while the *_write_MPTCP_key keys are used to authenticate the establishment of subflows.

It should be noted that the key derivation technique used by TLS requires a distinction between the active opener (i.e. the client) and the passive opener (i.e. the server). For most MPTLS connections it is easy to determine the role of each communicating host. However, in the case of a simultaneous establishment of the Multipath TCP connection, we need a tiebreak to determine which host acts as the client and which host acts as the server. We propose to use the Tokens and the random number exchanged in the MP_CAPABLE option. By convention, the client will be the host that proposed the smallest [token] in the MP_CAPABLE option and the server the other one. In the unlikely case that both hosts propose the same token, then we rely on the IP addresses of the two hosts. These addresses are unique and the client is the host that has the numerically smallest address. It should be noted that this solution does not work if there is a NAT on the path between the two communicating hosts that changes (one of) the IP addresses. However, it is unlikely that a simultaneous establishment of a TCP connection will be possible through a NAT anyway. Adding complexity to handle this unlikely scenario (same token pair and NAT) does not seem to be necessary.

6. Interactions with middleboxes

In this document, we assume that there are no middleboxes that modify, split or reassemble packets and/or options. A subsequent version of this document will discuss how such middleboxes could be handled.

7. Security considerations

Various TCP attacks have been documented in the literature. In this section, we discuss some of these attacks and analyze how they affect our proposed TLS above Multipath TCP. A more detailed version of this section will be provided in the next version of this document.

7.1. RST injection

A first attack is the injection of a RST segment in an existing TCP connection. Such RST segments can be injected by middleboxes as described [RFC3360]. There have also been documented attacks against very long-lived TCP connections (typically BGP sessions) where an attacker sends spoofed RST segments. Several techniques have been proposed to mitigate this attack [RFC4953].

If an off-path attacker sends a spoofed RST segment, it could terminate the corresponding TCP connection. For regular TCP, this attack would cause a denial of service. However, since our solution builds upon Multipath TCP, this attack is less severe. If an attacker is able to inject a RST segment on an established subflow,

this subflow will be terminated. This will not cause the termination of the Multipath TCP connection and thus will not affect the application. Multipath TCP can be configured to reestablish the subflow when a RST segment is received without a FAST_CLOSE.

An on-path attacker such as a middlebox as discussed in [RFC3360] can send a spoofed RST segment that would pass the mitigations described in [RFC4953]. However, such an attacker cannot know the keys that are used to authenticate the subflows and the TLS records. Thus, the only attack that such attacker could carry is inserting RST or FIN segments in one of the subflows. MPTLS will react to such attacks by reestablishing a new subflow.

7.2. Data injection

An attacker can also inject TCP segments containing an invalid TLS record in one of the TCP subflows. Given that the TLS record is protected by an HMAC computed with keys that are negotiated during the secure TLS handshake, the attacker, cannot inject a valid TLS record in the connection even if the attacker can predict the sequence numbers, acknowledgements, Data Sequence Number and Data acknowledgements that are verified by Multipath TCP. When a receiver detects an invalid HMAC, it discards the associated TLS record and terminates the associated TCP subflow. This will slowdown the data transfer but would not affect the reliability of the data transfer.

7.3. Fake TCP acknowledgements

An attacker can inject fake TCP acknowledgements at the subflow level but not that the DSS level given that the DSS-level acknowledgements are protected by the HMAC. Such an attack could force the retransmission of already transmitted data at the subflow level.

8. Conclusion

In this document, we have proposed the main design principles of MPTLS : a tighter integration between Multipath TCP and Transport Layer Security (TLS). By leveraging the best characteristics of each protocol, MPTLS provides better security and reliability than by considering Multipath TCP and TLS as isolated protocols.

MPTLS would be useful for two different types of applications. First, it would be beneficial for all the existing applications that rely on TLS, even if they are used on single-homed devices. Second, MPTLS could serve as a basis for the TCP extension that is being discussed within the TCPINC working group to provide unauthenticated encryption and integrity protection of TCP streams. These two use

cases will be discussed in more details in the next versions of this draft.

9. Acknowledgements

This work was partially supported by the FP7-Trilogy2 project.

10. References

10.1. Normative References

[I-D.ietf-mptcp-rfc6824bis]
Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-02 (work in progress), January 2014.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

[RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, September 2014.

10.2. Informative References

[Bellovin]
Bellovin, S., "Security problems in the TCP/IP protocol suite", SIGCOMM Comput. Commun. Rev. 19, 2 (April 1989), 32-48 , April 1989, <http://doi.acm.org/10.1145/378444.378449>.

[Grigorik]
Grigorik, I., "High Performance Browser Networking", O'Reilly , 2013, <http://chimera.labs.oreilly.com/books/1230000000545>.

[I-D.bittau-tcp-crypt]
Bittau, A., Boneh, D., Hamburg, M., Handley, M., Mazieres, D., and Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-bittau-tcp-crypt-04 (work in progress), February 2014.

[I-D.bonaventure-mptcp-rst]
Bonaventure, O., Paasch, C., and G. Detal, "Processing of RST segments by Multipath TCP", draft-bonaventure-mptcp-rst-00 (work in progress), July 2014.

[I-D.ietf-tcpm-tcp-security]
Gont, F., "Survey of Security Hardening Methods for Transmission Control Protocol (TCP) Implementations", draft-ietf-tcpm-tcp-security-03 (work in progress), March 2012.

[I-D.paasch-mptcp-lowoverhead]
Paasch, C. and O. Bonaventure, "MultiPath TCP Low Overhead", draft-paasch-mptcp-lowoverhead-00 (work in progress), October 2012.

[I-D.paasch-mptcp-ssl]
Paasch, C. and O. Bonaventure, "Securing the MultiPath TCP handshake with external keys", draft-paasch-mptcp-ssl-00 (work in progress), October 2012.

[I-D.sheffer-uta-tls-attacks]
Sheffer, Y., Holz, R., and P. Saint-Andre, "Summarizing Current Attacks on TLS and DTLS", draft-sheffer-uta-tls-attacks-00 (work in progress), February 2014.

[Normaliser]
Kreibich, C., Handley, M., and V. Paxson, "Network intrusion detection Evasion, traffic normalization, and end-to-end protocol semantics", Proc. USENIX Security Symposium , 2001.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC3360] Floyd, S., "Inappropriate TCP Resets Considered Harmful", BCP 60, RFC 3360, August 2002.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.

[RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.

Author's Address

Olivier Bonaventure
UCLouvain

Email: Olivier.Bonaventure@uclouvain.be