Network Working Group   M. Bagnulo
Internet-Draft   UC3M
Intended status: Standards Track   February 12, 2014
Expires: September 3, 2014

Secure MPTCP
draft-bagnulo-mptcp-secure-00

Abstract

This memo contains some initial thoughts about how to secure MPTCP. As currently defined, MPTCP provides basic security features to protect the MPTCP signaling and the data flows unprotected. In this note, we explore the possible use to tcpcrypt to provide enhanced security to MPTCP.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Multi-path TCP (MPTCP) [RFC6824] defines the extensions to TCP that allow transmitting data over multiple paths in a single TCP connection. This is achieved by opening multiple subflows within the same TCP connection. Each subflow is associated to a different address/port pair. As currently defined, MPTCP provides basic security for the signaling used to establish the subflows. A threat analysis for MPTCP is presented in [RFC6181] and a residual threat analysis is presented in [I-D.bagnulo-mptcp-attacks]. From these analysis we can extract that MPTCP as currently defined is vulnerable to attackers that can eavesdrop the initial connection establishment exchange and also to attackers that can intercept any subflow establishment exchange. In addition, MPTCP does not provide any protection to the data stream (other than splitting the data stream over multiple paths), as this was a non goal of the MPTCP design. In [I-D.bagnulo-mptcp-attacks] it is concluded that if a more secure version of MPTCP should be pursued, the path to follow would be to protect the data stream rather than trying to provide additional security to the signaling. The reader is referred to the aforementioned reference for additional insight why this is the case. The goal of this document is provide initial considerations about how to provide enhanced security to MPTCP by securing the data stream.

In this note, we analyze the use of tcpcryp [I-D.bittau-tcp-crypt] to secure MPTCP. tcpcrypt defines extensions to TCP so opportunistically encrypt the data stream of a TCP connection. By using tcpcrypt in MPTCP, we would be able to provide enhanced security to MPTCP. We note however, that the resulting solution would still be vulnerable to Man-in-the-Middle attacks during the initial key negotiation. However, the attacker in this case must be active and must remain located along the path during the whole lifetime of the connection.

We call SMTCP to the integration of MPTCP with tcpcryp. This would provide stronger security of MPTCP, both for the signaling and for the data. Since all the MPTCP signaling will be protected by tcpcrypt (i.e. encrypted and its integrity protected) there is no need for the existent MPTCP security mechanisms when used with tcpcrypt. This means that there is no need to negotiate a the current MPTCP key and that the HMAC protection provided by the MPTCP protocol is not needed (except for backward compatibility issues). All the protection will be achieved with the tcpcrypt extensions.

2. Initial SMTCP connection

Suppose both A and B are SMTCP capable. Suppose A has both IPA1 and IPA2. Suppose A initiates a SMTCP connection with B. The exchange would look like as follows:

A -> B: SYN + MP_CAPABLE (including A's key (ka) and C bit set) + CRYPT/Hello This contains 15 bytes of options (the motivation for including both the MP_CAPABLE and the CRYPT option is for backward compatibility, see Section 5). SYN packets usually carry as well MSS (4 bytes), SACK (2 bytes) and Window-Scale (3 bytes). Negotiating timestamps would be 10 more bytes. As options has a maximum length of 40 bytes, this would be compatible with all the mentioned options.

B -> A: SYN/ACK + MP_CAPABLE (including B's key (kb) and the C bit set) + CRYPT/PKCONF (with pub-cipher-list) Assuming we have 2 algorithms in the list, this is 10 bytes, making a total of 22 bytes. This packet also usually carries the same options than the SYN packet, so in this case, not all the mentioned additional options would fit.

A -> B: INIT (3 bytes of options) plus crypto data in the payload (The MP_CAPABLE option is needed since the keys are generated by tcpcrypt)

B -> A: INIT (3 bytes of options) plus crypto data in the payload

During the 3-way handshake MPTCP generates the following values:

The key: session to key to protect the signaling (one per each side)

The Token: used as connection identifier (one per each side)

The IDSN: Initial Data Seq Number (one per each side)

The idea would be to derive them from the tcpcrypt values.

o The keys: tcpcrypt generates 4 keys, kec, kac, kes and kas. These will be used to secure MPTCP, as discussed later on. for the purposes of MPTCP signaling, the keys that will be used are the authentication keys, so the keys for MPTCP are kac and kas.

o The tokens: tcpcrypt generates a Session ID, which is the full length of a hash output. The thing is that tcpcrypt generates a single SID for both endpoints, while MPTCP generates one per endpoint. In addition, MPTCP needed to generate a pair of ISDNs.

We could then generate the MPTCP values out of the tcpcrypt values as follows

Key A= kac

Key B= Kas

Token A= 32 most significant bits of (hash(ka))

Token B= 32 most significant bits of (hash(kb))

IDSN A= 64 least significant bits of (hash(kac + SID))

IDSN B= 64 least significant bits of (hash(kas + SID))

DISCUSSION: We need to exchange ka in the MP_CAPABLE message because of backwards compatibility issues (see Section 5), so there is no easy way around it. However, the reason to exchange kb in the MP_CAPABLE is because we need to be able to generate Token B in a way that is unique in host B. This seems an overkill (exchanging a 64 bit key to achieve a 32 bit unique token). It could be possible to define a new MP_CAPABLE option that would exchange a 32 bit long token directly. We also need to understand the security implications of exchanging the token in the clear. Another alternative would be to generate the token as a hash of kas and the SID and if it happens to clash, restart the connection. Tis may make sense depending how likely is this to happen. TODO: work out the other flavors of tcpcrypt connection initiation, including the use of cached keys

3. Adding new subflows

The options for this is whether to treat a new subflow as a new tcpcrypt connection or not, the implication being that a new tcpcrypt connection uses a different shared secret and hence different keys (even though not new public key operations are needed). Probably this is not the way to go. All the flows of a MPTCP connection should be part of the same tcpcrypt session. (the other option would

imply that there are different keys for the same MPTCP session, which may be cumbersome?)

So, one simple way of doing this would be to simply use the existent MPTCP exchange to add a new subflow with the MPTCP security measures. This implies sending an MP_JOIN containing the receiver's token and a random number which will be responded with another MP_JOIN and the final JOIN message. The tcpcrypt keys are used instead of the regular MPTCP keys.

(similar approach can be used for the ADD_ADDR option, which is secure using an HMAC using the tcpcryp derived key)

A alternative approach would be to drop completely the MPTCP security mechanisms and use the tcpcryp MAC option to secure the MPTCP signaling. This implies that the tcpcryp MAC option would need also to protect the MPTCP MP_JOIN option

4. Exchanging data

Once the keys and the other values have been negotiated, data can flow. All data in the MPTCP connection will be encrypted with tcpcrypt keys and its integrity protected using the tcpcrypt MAC option. This adds 22 bytes of options (assuming 160 bits long hash).

Question: do we need to have a 160 bit long hash or can we live with less?

Now, MPTCP includes the DSS option in order to synchronize the data sequence number with the sequence numbers of the subflows. The DSS option max length is 28 bytes. The results it that the MAC option plus the DSS option are 50 bytes, which is a problem.

The good news is that the DSS does not need to be sent in every segment and that 28 bytes is the maximum length.

Currently the DSS option includes information both about DSN mapping to subflow seq number and data ack. In order to limit the size of the option, one option is to prevent that both Data Ack and DSN to subflow seq numbers mappings are sent in the same option. This would result that when Data acks are sent, the DSS option has a maximum of 12 bytes and when DSN to subflow seq number mapping are sent, the max length is 20 bytes. This is still 2 bytes too long. There are two ways we can shrink this. One option is to prevent the use of Checksum when tcpcrypt is used. checksum is optional, so this could be done. Moreover, it makes sense to do this, because all the information protected in this checksum is protected by the tcpcrypt MAC option. this results that the DSS option is now 18 bytes, which

with the tcpcrypt MAC option will make up to 40 bytes of tcp options. The other possible way to shrink this is to use the 4 bytes seq numbers rather than the 8 ones. This would reduce the DSS option to 14 bytes for the DSN to subflow seq number mapping and to 8 bytes in the case of Data acks.

The DSS option will be sent in the clear i.e. not encrypted by tcpcrypt. The MAC option must cover the DSS option (correct?). This implies that we need to add to the MAC data structure the DSS option.

Question: how often the DSS needs to be sent? I mean, if we send the DSS and the MAC options, we will be using all the TCP option room, so, not SACK can be sent, which is bad.

5. Backward compatibility

There will be the following 5 types of node:

MPTCP nodes: supports MPTCP as defined in RFC6824 or RFC6824bis but does not support tcpcrypt,

tcpcryp nodes: supports tcpcrypt but does not support MPTCP,

SMTCP capable nodes: support MPTCP and tcpcrypt and the use of tcpcrypt to secure MPTCP,

legacy nodes: dont support neither tcpcrypt nor MPTCP,

MPTCP/tcpcrypt nodes: supports both tcpcrypt and MPTCP but does not support the use of tcpcrypt to protect MPTCP.

The expected behavior is as following:

a. SMTCP contacts a SMTCP node, SMTCP should be used

b. SMTCP contacts a MPTCP node, MPTCP should be used

c. SMTCP contacts a tcpcrypt node, tcpcrypt should be used

d. SMTCP contacts a MPTCP/tcpcrypt node, not sure what should happen... should MPTCP and tcpcrypt be used in a non integrated fashion? (not sure if there is enough space in the TCP options for this...)

e. SMTCP contacts a legacy node, TCP should be used.

In order to achieve, we use the following approach. In the initial SYN of the initial 3-way handshake, both the CRYPT/Hello option (3

bytes) plus the MP_CAPABLE option including the initiator's key (12 bytes) should be sent. This allows supporting b), c) and e) i.e. the receiver can discard either of the two options or both of them resulting in each of the mentioned cases.

In order to support case a) (and to distinguish it from case d), we need to signal it in an explicit way. I guess the easiest way is to use one of the flags C to H in the MP_CAPABLE message. Let's assume it is the C(rypt) flag. If the C flag is set and the CRYPT/Hello option is present, this means SMTCP (i.e. use tcpcrypt to protect MPTCP signaling and data).

6. Concluding remarks

One main challenge in order to use tcpcrypt to secure MPTCP is the option space. There is little room for TCP options and this approach would consume most of it, which would prevent the use of other options like SACK. One way to address this would be that tcpcrypt is changed to send the MAC as part of the data stream rather than an option. As tcpcrypt is being discussed, this can be an option.

A second issue to consider is how this would work with TSO. Currently MPTCP is compatible with TSO and it would be important that SMPTCP is also compatible.

Another comment is that it would be possible to secure MPTCP using something like TLS opportunistically and transparently to the application. This is TBD as an alternative approach.

7. Security Considerations

This whole document is about securing MPTCP. In future versions of the document, this section could include a residual threat analysis.

8. IANA Considerations

TBD

9. Acknowledgements

The authors thank ...

10. References

10.1. Normative References

[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-03 (work in progress), September 2012.

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

10.2. Informative References

[I-D.bagnulo-mptcp-attacks]
Bagnulo, M., Paasch, C., Gont, F., Bonaventure, O., and C. Raiciu, "Analysis of MPTCP residual threats and possible fixes", draft-bagnulo-mptcp-attacks-01 (work in progress), October 2013.

[RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, March 2011.

Author's Address

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es