Multipath TCP Security Issues: Current Solution Space

Alan Ford <alan.ford@roke.co.uk>

MPTCP Interim – 2010-12-14

Why we need security

- Say a host is multi-addressed, and wants to open a new subflow from its second address to the same destination
- When a new TCP SYN comes in from this second address, how can the receiver know and verify to which MPTCP connection it belongs?
- A multi-addressed host can also signal its additional addresses to the peer, which then initiates connection attempts

Threat Model

- *draft-ietf-mptcp-threat-06*
- Most threats relate to hijacking:
  - On-path and off-path attackers
  - Live and time-shifted attacks
  - Leading to creating new subflows and injecting or intercepting data, potentially also closing existing subflows
- Essentially, we want to create a solution that is *no worse than TCP today*
- However, not all factors translate sufficiently well, so an appropriate goal is: *ensure the hosts communicating in the new subflow setup are the same as the hosts in the initial connection setup*

Scope

- How to ensure the hosts in a new subflow setup are the same as the hosts in the initial connection setup?
- Man-in-the-middle (i.e. on-path attacker able to change packets) is not possible to solve
- In regular TCP, if a passive listener has timely information, it may be able to inject spoofed packets, but cannot hijack the connection
  - Ideally we want to keep the same here
- Multiple subflows brings with it additional interfaces to a connection
  - This shouldn't weaken the security, but may increase available locations for listeners

Simple (-01) Proposal

- Each end has a 32-bit token for the connection
- Tokens used as authenticators
  - Seen in every subflow SYN exchange
  - Once you know one, you can glean the other
- Initial Data Sequence Number set at MP_CAPABLE handshake
- DSNs used as blind attack security

Simple (-01) Proposal

```
         Host A                               Host B
------------------------             ------------------------
Address A1    Address A2             Address B1    Address B2
----------    ----------             ----------    ----------
    |             |                      |             |
    |     SYN + MP_CAPABLE(Token A)      |             |
    |----------------------------------->|             |
    |<-----------------------------------|             |
    |   SYN/ACK + MP_CAPABLE(Token B)    |             |
    |             |                      |             |
    |             |       SYN + MP_JOIN(Token B)       |
    |             |----------------------------------->|
    |             |<-----------------------------------|
    |             |     SYN/ACK + MP_JOIN(Token A)     |
    |             |                      |             |
```

But this has weaknesses

- We had concerns about attackers being able to join a connection with a time-shifted attack
- Listening to any subflow setup will give tokens
- If one token is known, the other can be gleaned from a subflow SYN exchange
- Knowledge of a token is the only restriction to setting up a subflow
  - Potential to inject/glean information
  - Potential DoS attack, but probably no worse than traditional TCP state exhaustion

What protection do we have?

- The Data Sequence Number would be required to be in-window for the path to be used (as for TCP blind attacks today)
- But to ensure this, we must prevent this information from being leaked
  - Therefore we cannot send data to newly initiated subflow until it has verified it knows DSN through transmitting data
  - This causes problems for unidirectional flows, unless we could rely on duplicate DATA_ACKs
- Also, there are liveness tests for REMOVE_ADDR
  - But this may be a threat in address loss scenarios!
- We started looking at a hash-based solution as an alternative…

Hash-based (-02) Proposal

- Connection setup (MP_CAPABLE) exchanges keys:
  - SYN A->B: Option carries (Key-A)
  - SYN/ACK B->A: Option carries (Key-B)
  - ACK A->B: Options carry (Key-A, Key-B)
- Initial DSNs created from hashes
  - E.g. IDSN-A = H(Token-A); Token-A = H(Key-A)
- New subflows (MP_JOIN) uses hash of Key as Token for Connection ID, plus Random Number (for replay protection), and HMACs this data using the Keys (keys never again seen in the clear):
  - SYN A->B: Option carries (Token-B, R-A)
  - SYN/ACK B->A: Option carries (Token-A, R-B)
  - ACK A->B: Payload carries:
    HMAC(Key=Key-A|Key-B, Message=R-A|R-B)
  - ACK B->A: Payload carries:
    HMAC(Key=Key-B|Key-A, Message=R-B|R-A)

Current (-02) Proposal

```
         Host A                                      Host B
------------------------                    ------------------------
Address A1    Address A2                    Address B1    Address B2
----------    ----------                    ----------    ----------
    |             |                             |             |
    |         SYN + MP_CAPABLE(Key-A)           |             |
    |------------------------------------------>|             |
    |<------------------------------------------|             |
    |       SYN/ACK + MP_CAPABLE(Key-B)         |             |
    |             |                             |             |
    |      ACK + MP_CAPABLE(Key-A, Key-B)       |             |
    |------------------------------------------>|             |
    |             |                             |             |
    |             |        SYN + MP_JOIN(Token-B, R-A)        |
    |             |------------------------------------------>|
    |             |<------------------------------------------|
    |             |      SYN/ACK + MP_JOIN(Token-A, R-B)      |
    |             |                             |             |
    |             |  HMAC(Key=(Key-A+Key-B), Msg=(R-A+R-B))   |
    |             |------------------------------------------>|
    |             |<------------------------------------------|
    |             |  HMAC(Key=(Key-B+Key-A), Msg=(R-B+R-A))   |
    |             |                             |             |
```

First Question

- Is the new hash-based approach adding anything?
- We prevent anyone listening to any subflow setup from being able to set up another subflow – *except if they listen to the initial (MP_CAPABLE) connection handshake*
- Is this a relevant improvement?
  - Note that the DSN protection issues still remain, since there is still a potential weak point
  - The MP_JOIN handshake is extended to four messages

Hash Chains Proposal

- A hash chain removes the threat against the initial key exchange, but adds computational complexity
- A hash chain is based on a random number, *R*:
  H[0] = H(*R*); H[1] = H(H[0]); H[2] = H(H[1]); ...; H[n] = H(H[n-1])
- A host generates up to H[n] at the start and declares this value
- It then sends the preceding entry in the chain (i.e. H[n-1]) as a verification token
- A hash chain entry can only be used once
- Need signal to extend hash chain if you run out

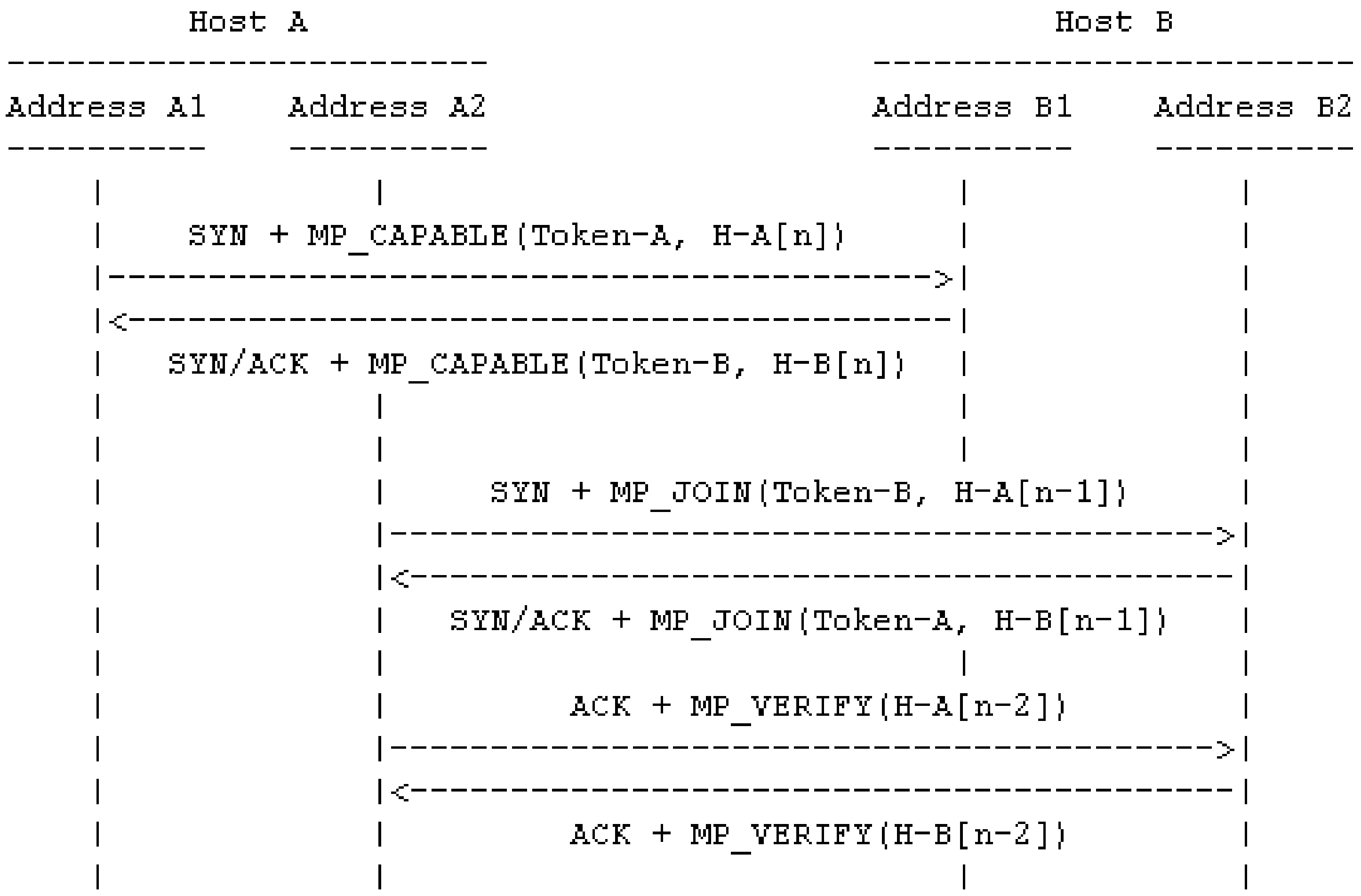
Applying Hash Chains to MPTCP

```
         Host A                                     Host B
------------------------                   ------------------------
Address A1    Address A2                   Address B1    Address B2
----------    ----------                   ----------    ----------
     |             |                            |             |
     |   SYN + MP_CAPABLE(Token-A, H-A[n])      |             |
     |----------------------------------------->|             |
     |<-----------------------------------------|             |
     |  SYN/ACK + MP_CAPABLE(Token-B, H-B[n])   |             |
     |             |                            |             |
     |             |                            |             |
     |             |    SYN + MP_JOIN(Token-B, H-A[n-1])      |
     |             |----------------------------------------->|
     |             |<-----------------------------------------|
     |             |  SYN/ACK + MP_JOIN(Token-A, H-B[n-1])    |
     |             |                            |             |
```

Remaining Threats

- This now ensures that even if an attacker sees the initial handshake, he cannot use that information to attack via MP_JOIN
- But there is a potential threat of a listener who is able to upgrade to an active attacker by racing a fresh hash chain entry, and winning
  - i.e. Attacker observes genuine MP_JOIN and replays it himself, before the genuine one gets to the receiver
- This can be solved by requiring a second entry from the hash chain after subflow establishment

Hash Chains #2



```
         Host A                                     Host B
------------------------                   ------------------------
Address A1    Address A2                   Address B1    Address B2
----------    ----------                   ----------    ----------
    |             |                            |             |
    |    SYN + MP_CAPABLE(Token-A, H-A[n])     |             |
    |----------------------------------------->|             |
    |<-----------------------------------------|             |
    |   SYN/ACK + MP_CAPABLE(Token-B, H-B[n])  |             |
    |             |                            |             |
    |             |                            |             |
    |             |     SYN + MP_JOIN(Token-B, H-A[n-1])     |
    |             |----------------------------------------->|
    |             |<-----------------------------------------|
    |             |   SYN/ACK + MP_JOIN(Token-A, H-B[n-1])   |
    |             |                            |             |
    |             |         ACK + MP_VERIFY(H-A[n-2])        |
    |             |----------------------------------------->|
    |             |<-----------------------------------------|
    |             |         ACK + MP_VERIFY(H-B[n-2])        |
    |             |                            |             |
```

So is this an improvement?

- Hash chains remove the threat from listeners at all stages of the connection
- But require computational overhead to calculate the hash chains
  - Especially if you need to do this before the subflow is properly established
  - Not currently SYN-cookie-friendly
- Also need new signalling for hash chain extension and initial verification
  - Plus we still need a four-way handshake, although MP_VERIFY could be in options along with data
- Are there other possibilities for replay attacks?

Summary of Properties

- Tokens (-01)
  - Stops blind attacks
- HMAC (-02)
  - Protect against attackers intercepting MP_JOIN
  - Susceptible to listener on MP_CAPABLE
- Hash Chains
  - Protect against interception of MP_CAPABLE & MP_JOIN
  - But there remains a racing attack of intercepting and copying a hash from genuine MP_JOIN
- Hash Chains #2
  - Additional hash chain entry removes the above threat, at the cost of more signalling and using more hash entries

Comparisons

- Assuming we need something better than tokens & DSN protection
- Accept we’ll always be vulnerable to MITM
- HMAC protects subflow setup against listeners who hasn’t seen the connection initialisation
- Hash Chains also protects against listeners who have seen the connection initialisation, at the expense of introducing racing threats
- Racing threats can be alleviated through the use of two hash entries
- But hash chains are also computationally expensive and complex, and don’t have stateless handshaking

So what other solutions are there?

- Remembering we are only trying to be *no worse than TCP*
  - Anything extra is a bonus, but not at the expense of MPTCP development or deployability
- If applications care about security, they will use TLS, which remains secure with MPTCP

- Indeed, should we be spending our time on this at all?
  - Charter specifies we should work on a "basic security model" – but what is the appropriate scope for this?

Where next?

- Option 1: Keep with HMAC proposal in draft
- Option 2: Move to hash chains proposal
- Option 3: Fall back to basic tokens approach, leaving extensions out of scope for now
- Option 4: Something else?

What still needs doing?

- Irrespective of option chosen:
  - Algorithm Agility
    - ~4 bits in the MP_CAPABLE option?
  - If appropriate, SYN-cookie like behaviour
    - Already there for HMAC, trivial for tokens, still unclear for hash chains

- *Anything else?*