Alternative Security Solutions for MPTCP

Christoph Paasch<br>Christoph.paasch@uclouvain.be<br>IP Networking Lab - UCLouvain

draft-paasch-mptcp-lowoverhead-00
draft-paasch-mptcp-ssl-00
Motivation

- Need for low-overhead MPTCP handshake
  - Data centers
  - Non security-critical traffic

draft-paasch-mptcp-lowoverhead-00
Goals:

- Reduce chances of token collision
- Avoid extensive HMAC-calculations
- Allow stateless SYN+JOIN handling

Non-goal:

- Authenticate new subflows
Low overhead initial handshake

• Purposes of the initial handshake:
  ○ Detect whether the peer supports MPTCP
  ○ Exchange a connection identifier (Token)
  ○ Agree on the Initial Data Sequence Number
Low overhead initial handshake

Generate unique TokenA
Generate RandomA

Store TokenB
IDSN-A = RandA || TokA
IDSN-B = RandB || TokB

Generate unique TokenB
Generate RandomB

Store TokenA
IDSN-B = RandB || TokB
IDSN-A = RandA || TokA
Low overhead initial handshake

Generating the token:

- Could be a simple counter
- Better: block-cipher (RC5) of the counter plus a local secret

Benefits:
- Guarantees a high probability of uniqueness
- No need to compute a SHA-1
Low overhead additional subflows
Low overhead additional subflows

Benefits:

- No need to generate an HMAC
- Allows stateless handling of SYN+JOIN
Performance evaluation of MPTCP low overhead
SYN/ACK with MP_CAPABLE

Time measured
SYN/ACK with MP_CAPABLE
SYN/ACK with MP_CAPABLE
SYN/ACK with MP_CAPABLE
Random Number Generation

- MPTCP v0 generates random numbers:
  - Key (64-bits)
  - Random Number for MP_JOIN (32-bits)

- Generating these random numbers accounts for ~20% of the time
Random Number Generation

- **RFC 6258: Generating the TCP sequence number:**
  - ISN = MD5(5-tuple + secret)

- **MPTCP Proposal:**
  - Key = Hash(5-tuple + secret + counter)
  - Join-Nonce = Hash(5-tuple + secret + seqNo)
SYN/ACK with MP_CAPABLE
SYN/ACK with MPJOIN

Time measured
SYN/ACK with MP_JOIN
SYN/ACK with MP_JOIN
SYN/ACK with MP_JOIN
Impact on "real" traffic

- 250 simultaneous HTTP-requests
- File-size: 1KB
- Increasing number of MPTCP-subflows

- Measuring Requests/Second
- Baseline: Regular MPTCP
Impact on "real" traffic
Securing the MultiPath TCP handshake with external keys
Motivation

- MPTCP v0 sends the keys in clear
  - Attacker who sees the initial handshake can hijack an MPTCP session
  - Integrate application-level security into MPTCP to benefit from the app-security

draft-paasch-mptcp-ssl-00
Goals:

- Do not send MPTCP's key in clear
- Application-level protocols already do negotiate a key (cfr. SSL)

We should use these keys!

- Extend the socket-API to allow keys from the application
SSL initial handshake

setsockopt(MPTCP_ENABLE_APP_KEY)

setsockopt(MPTCP_ENABLE_APP_KEY)

SYN
MP_CAPABLE
B-bit set

SYN+ACK
MP_CAPABLE
B-bit set
SSL initial handshake

SSL exchange

Hash(Shared Key)

setsockopt(MPTCP_KEY)

Hash(Shared Key)

setsockopt(MPTCP_KEY)
SSL additional subflow

MPTCP-Key = Hash (Shared Key)

SYN

MP_JOIN

Token B, Rand A

SYN+ACK

MP_JOIN

HMac B, Rand B

ACK

MP_JOIN

Token B, Rand A, HMac A

HMac B = HMAC (Key, Rand A || Rand B)

HMac A = HMAC (Key, Rand B || Rand A)
JOIN-flooding Attack

MPTCP connection

Token A

Token B

SYN
Src-Port: 4245

MP_JOIN
Token B, Rand X3

Join State-Table
Stateless JOIN-handling

- Prevents JOIN-flooding attacks
- Based on the mechanism of TCP SYN-Cookies
Stateless JOIN

Possible thanks to the modified JOIN-format:

MP_JOIN

HMAC - truncated to 128-bits
Random A
Token B

Server must generate his random number in a verifiable fashion:
Rand B = Hash(5-tuple + secret)
Conclusion

MPTCP low overhead
  ● Performance improvement

MPTCP external keys
  ● Keys are no more exchanged in clear