Segment Routing over IPv6 (SRv6) Proof of Transit

draft-iannone-spring-srv6-pot-00

L. Iannone, A. Fressancourt

IETF 119 – Brisbane
Path Verification Problem Statement
(or should we say segment verification...)

- Path Validation via Proof of Transit makes sure the path we chose is the actual path the traffic travels
- How to make sure that DESIRED path equals ACTUAL path?

- Growing interest Proof of Transit (POT) for:
  - Service Function Chaining (SFC)
  - Workload identity
  - Traffic path compliance
  - uRPF validation
  - Segment Routing Security
1. Generate HMAC using key pre-shared with egress
   - Algorithm can be SHA1 or SHA256 according to RFC 8754
2. Generate random Nonce
3. Repeatedly encrypt HMAC using Nonce and pre-shared keys starting from last segment
POT in Practice: Middle Segments Operations

• For each packet containing PoT TLV:
  1. Retrieve pre-shared key
  2. Decrypt HMAC once
  3. Forward Packet to next segment
For each packet containing PoT TLV:
1. Retrieve pre-shared key
2. Decrypt HMAC once
3. Compute packet HMAC
   • If newly compute HMAC == HMAC obtained from the packet after decryption then forward; else discard.
Recursive Cryptography as POT

1. **Packet creation:**

   Given the set of keys
   \[
   [k_1, k_2, k_3, k_4, k_D, k_{HSD}]
   \]
   Identified by a *Key Set ID* and a randomly generated value *N*,
   the following Path Verification Factor (PVF) is computed:
   \[
   PVF_{HSD} = HMAC_{k_{HSD}}(SID-list, \ldots)
   \]
   \[
   PVF_1 = Enc_{k_1}(Enc_{k_2}(Enc_{k_3}(Enc_{k_4}(Enc_{k_D}(PVF_{HSD}), N), N), N), N), N)
   \]
   Then along with the SRH the following is sent:
   \[
   M = Key Set ID | N | PVF_1
   \]

2. **Packet processing**

   Each intermediate node *i* receives a packet
   \[
   M = Key Set ID | N | PVF_{i-1}
   \]
   and computes:
   \[
   PVF_i = Dec_{k_i}(PVF_{i-1}, N)
   \]
   Then it sends:
   \[
   M = Key Set ID | N | PVF_i
   \]

3. **Packet verification:**

   The destination receives a packet:
   \[
   M = Key Set ID | N | PVF_{N_{i-1}}
   \]
   and computes:
   \[
   PVF_{HSD} = Dec_{k_D}(PVF_{D-1}, N)
   \]
   To verify the HMAC, it computes:
   \[
   PVF_{HSD}' = HMAC_{k_{HSD}}(SID-list, \ldots)
   \]
   If \(PVF_{HSD} = PVF_{HSD}'\), then SID-list and the path has been verified.
Proof of Transit Segment Routing Header TLV

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Type</td>
<td>Length</td>
<td>Reserved (8 bits)</td>
<td>Nonce Length (8 bits)</td>
</tr>
</tbody>
</table>

- **Key Set ID (32 bits)**
- **Nonce (variable length)**
- **Encrypted HMAC (variable length)**

- **Improvements compared to SRv6 HMAC:**
  - Sequential encryption and decryption of the HMAC → verification that the routers listed in the SID list have been crossed

- **Drawbacks compared to SRv6 HMAC:**
  - Intermediate routers are unable to verify the SID list’s integrity.
Linux Kernel Implementation

- WSL Kernel 6.1.21.1
  - Header (TLV) Operations
  - Crypto: AES/XTS
  - Sysctl API
  - CLI Configuration
    - Support in iproute2 (v6.6.0)
Conclusion

• Initial evaluation quite promising
  • Cost in terms of delay similar to Segment-by-Segment HMAC
    • (in the specific setup)

• Happy to discuss more and show you a demo!