ippm F. Brockners
Internet-Draft Cisco
Intended status: Informational S. Bhandari
Expires: July 29, 2021 Thoughtspot
T. Mizrahi
Huawei
January 25, 2021

Integrity of In-situ OAM Data Fields
draft-brockners-ippm-ioam-data-integrity-00

Abstract

In-situ Operations, Administration, and Maintenance (IOAM) records operational and telemetry information in the packet while the packet traverses a path between two points in the network. This document is to assist the IPPM WG in designing a solution for those deployments where the integrity of IOAM data fields is a concern. This document proposes several methods to ensure the integrity of IOAM data fields.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

"In-situ" Operations, Administration, and Maintenance (IOAM) records OAM information within the packet while the packet traverses a particular network domain. The term "in-situ" refers to the fact that the OAM data is added to the data packets rather than is being sent within packets specifically dedicated to OAM. IOAM is to complement mechanisms such as Ping, Traceroute, or other active probing mechanisms. In terms of "active" or "passive" OAM, "in-situ"

OAM can be considered a hybrid OAM type. "In-situ" mechanisms do not require extra packets to be sent. IOAM adds information to the already available data packets and therefore cannot be considered passive. In terms of the classification given in [RFC7799] IOAM could be portrayed as Hybrid Type 1. IOAM mechanisms can be leveraged where mechanisms using e.g. ICMP do not apply or do not offer the desired results, such as proving that a certain traffic flow takes a pre-defined path, SLA verification for the live data traffic, detailed statistics on traffic distribution paths in networks that distribute traffic across multiple paths, or scenarios in which probe traffic is potentially handled differently from regular data traffic by the network devices.

The current [I-D.ietf-ippm-ioam-data] assumes that IOAM is deployed in specific network domains, where an operator has means to select, monitor, and control the access to all the networking devices, making the domain a trusted network. As such, IOAM tracing data is carried in the packets in clear and there are no protections against any node or middlebox tampering with the data. As a consequence, IOAM tracing data collected in an untrusted or semi-trusted environments cannot be trusted for critical operational decisions. Any rogue or unauthorized change to IOAM data fields in a user packet cannot be detected.

Recent discussions following the IETF last call on [I-D.ietf-ippm-ioam-data] revealed that there might be uses of IOAM where integrity protection of IOAM data fields is at least desirable, knowing that IOAM data fields integrity protection would incur extra effort in the data path of a device processing IOAM data fields. As such, the following additional considerations and requirements are to be taken into account in addition to addressing the problem of detectability of any integrity breach of the IOAM trace data collected:

1. IOAM trace data is processed by the data plane, hence viability of any method to prove integrity of the IOAM trace data must be feasible at data plane processing/forwarding rates (IOAM data might be applied to all traffic a router forwards).

2. IOAM trace data is carried within data packets. Additional space required to prove integrity of the data needs to be optimal, i.e. should not exceed the MTU or have adverse affect on packet processing.

3. Replay protection of older IOAM trace data should be possible. Without replay protection a rogue node can present the old IOAM trace data masking any ongoing network issues/activity making the IOAM trace data collection useless.

This document is to assist the IPPM working group in designing and specifying a solution for those deployments where the integrity of IOAM data fields is a concern. This document proposes several methods to achieve integrity protection for IOAM data fields.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Abbreviations used in this document:

Geneve: Generic Network Virtualization Encapsulation [I-D.ietf-nvo3-geneve]

GRE Generic Routing Encapsulation

IOAM: In-situ Operations, Administration, and Maintenance

MTU: Maximum Transmit Unit

NSH: Network Service Header [RFC8300]

OAM: Operations, Administration, and Maintenance

POT: Proof of Transit

SFC: Service Function Chain

3. Threat Analysis

This section presents a threat analysis of integrity-related threats in the context of IOAM. The threats that are discussed are assumed to be independent of the lower layer protocols; it is assumed that threats at other layers are handled by security mechanisms that are deployed at these layers.

This document is focused on integrity protection for IOAM data fields. Thus the threat analysis includes threats that are related to or result from compromising the integrity of IOAM data fields. Other security aspects such as confidentiality are not within the scope of this document.

Throughout the analysis there is a distinction between on-path and off-path attackers. As discussed in [I-D.ietf-detnet-security], on-path attackers are located in a position that allows interception and

modification of in-flight protocol packets, whereas off-path attackers can only attack by generating protocol packets.

The analysis also includes the impact of each of the threats. Generally speaking, the impact of a successful attack on an OAM protocol [RFC7276] is a false illusion of nonexistent failures or preventing the detection of actual ones; in both cases, the attack may result in denial of service (DoS). Furthermore, creating the false illusion of a nonexistent issue may trigger unnecessary processing in some of the IOAM nodes along the path, and may cause more IOAM-related data to be exported to the management plane than is conventionally necessary. Beyond these general impacts, threat-specific impacts are discussed in each of the subsections below.

3.1. Modification: IOAM Data Fields

Threat

An attacker can maliciously modify the IOAM data fields of in-transit packets. The modification can either be applied to all packets or selectively applied to a subset of the en route packets. This threat is applicable to on-path attackers.

Impact

By systematically modifying the IOAM data fields of some or all of the in-transit packets an attacker can create a false picture of the paths in the network, the existence of faulty nodes and their location, and the network performance.

3.2. Modification: IOAM Option-Type Headers

Threat

An on-path attacker can modify IOAM data fields in one or more of the IOAM Option-Type headers in order to change or disrupt the behavior of nodes processing IOAM data fields along the path.

Impact

Changing the header of IOAM Option-Types may have several implications. An attacker can maliciously increase the processing overhead in nodes that process IOAM data fields and increase the on-the-wire overhead of IOAM data fields, for example by modifying the IOAM-Trace-Type field in the IOAM Trace-option header. An attacker can also prevent some of the nodes that process IOAM data fields from incorporating IOAM data fields by modifying the RemainingLen field.

3.3. Injection: IOAM Data Fields

Threat

An attacker can inject packets with IOAM Option-Types and IOAM data fields. This threat is applicable to both on-path and off-path attackers.

Impact

This attack and it impacts are similar to Section 3.1.

3.4. Injection: IOAM Option-Type Headers

Threat

An attacker can inject packets with IOAM Option-Type headers, thus manipulating other nodes that process IOAM data fields in the network. This threat is applicable to both on-path and off-path attackers.

Impact

This attack and it impacts are similar to Section 3.2.

3.5. Replay

Threat

An attacker can replay packets with IOAM data fields. Specifically, an attacker may replay a previously transmitted IOAM Option-Type with a new data packet, thus attaching old IOAM data fields to a fresh user packet. This threat is applicable to both on-path and off-path attackers.

Impact

As with previous threats, this threat may create a false image of a nonexistent failure, or may overload nodes which process IOAM data fields with unnecessary processing.

3.6. Management and Exporting

Threat

Attacks that compromise the integrity of IOAM data fields can be applied at the management plane, e.g., by manipulating network management packets. Furthermore, the integrity of IOAM data

fields that are exported to a receiving entity can also be compromised. Management plane attacks are not within the scope of this document; the network management protocol is expected to include inherent security capabilities. The integrity of exported data is also not within the scope of this document. It is expected that the specification of the export format will discuss the relevant security aspects.

Impact

Malicious manipulation of the management protocol can cause nodes that process IOAM data fields to malfunction, to be overloaded, or to incorporate unnecessary IOAM data fields into user packets. The impact of compromising the integrity of exported IOAM data fields is similar to the impacts of previous threats that were described in this section.

3.7. Delay

Threat

An on-path attacker may delay some or all of the in-transit packets that include IOAM data fields in order to create the false illusion of congestion. Delay attacks are well known in the context of deterministic networks [I-D.ietf-detnet-security] and synchronization [RFC7384], and may be somewhat mitigated in these environments by using redundant paths in a way that is resilient to an attack along one of the paths. This approach does not address the threat in the context of IOAM, as it does not meet the requirement to measure a specific path or to detect a problem along the path. It is noted that this threat is not within the scope of the threats that are mitigated in the scope of this document.

Impact

Since IOAM can be applied to a fraction of the traffic, an attacker can detect and delay only the packets that include IOAM data fields, thus preventing the authenticity of delay and load measurements.

3.8. Threat Summary

Threat	In scope	Out of scope
Modification: IOAM Data Fields	+	
Modification: IOAM Option-Type Headers	+	
Injection: IOAM Data Fields	+	
Injection: IOAM Option-Type Headers	+	
Replay	+	
Management and Exporting		+
Delay		+

Figure 1: Threat Analysis Summary

4. Methods of providing integrity to IOAM data fields

This section outlines four different methods that are to provide integrity protection of IOAM data fields. As noted earlier, this document is to support the IPPM working group in designing and specifying a method for protecting the integrity of IOAM data fields. It isn't expected that all four methods would be chosen for a solution specification.

The discussion of the different methods focuses on protecting the integrity of IOAM trace data fields, though the outlined methods are not limited to protecting IOAM trace data fields only. The methods could be applied to other IOAM Option-Types, such as the E2E Option-Type.

IOAM trace data can be embedded in a variety of protocols. There are specific drafts that cover the encapsulation of IOAM data into different protocols, like IPv6 [I-D.ietf-ippm-ioam-ipv6-options], NSH [I-D.ietf-sfc-ioam-nsh], Geneve [I-D.brockners-ippm-ioam-geneve], etc.

The IOAM Option-Types for tracing (Pre-allocated Trace-Option and Incremental Trace-Option) organize the collected data in an array, the "node data list". See [I-D.ietf-ippm-ioam-data] for further details).

The basic idea is to introduce a new "signed node-data hash field" added by each node along with the node data to prove the integrity of the node data inserted.

The following sections describe different methods of how such a "signed node-data field" could be used and populated. The methods assume an IOAM-Domain containing IOAM-encapsulating nodes, IOAM-decapsulating nodes and IOAM-transit nodes. In addition, it is assumed that traffic also traverses a Validator node, which verifies the integrity of the IOAM data fields. In a typical deployment, the IOAM-decapsulating node would also serve as the Validator. The setup also includes a network management entity/controller which handles key distributions to the network nodes and also serves as a receiver for validation results provided by the Validator. Protocols and procedures for the exchange of keys and validation results between the network management entity/controller and the nodes are outside the scope of this document.

4.1. Method 1: Using asymmetric keys for signing node trace data

Method 1 uses asymmetric keys for signing node trace data. This is the procedure to be followed by each node:

1. Each IOAM capable node creates a key pair and shares the public key with the controller, the Validator and the network management system responsible for using the IOAM trace information in the network domain. The detailed mechanisms how keys are exchanged between nodes are outside the scope of this document. For optimal performance, use of algorithms like BLS [BLS] or ED25519 [EdDSA25519] are suggested, resulting in fast signing for small keys and limited overhead (see below for an overhead calculation).

2. Each node data list [x] field is extended with an additional "signed node-data" field: node_data_sign[x]. Node_data_sign includes a signature using the private key of the node over the hash of node data list[x] of the node and the previous node's node data sign node_data_sign[x-1]. This couples the signature of the current field to the earlier field and creates a chain of trust. This way of chaining the node data signatures provides protection against replay of a previous node trace of a specific node.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        node_data_sign [x]                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                        node data list [x]                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3. The IOAM encapsulating node (the node that inserts IOAM data fields into the packet) will add a seed in its node data list that is used in its node_data_sign. So the first IOAM node inserting the IOAM trace data will add node_data_sign over a "seed" || [hash of node data of first node]. The seed can be included as a field in first node data or the seed can be the trailer of the IOAM Trace-Option.

4. The validating node - will use the public key of each node to validate the signed node data elements in the same way the node Trace signatures were created, i.e. it'll repeat the individual operations of the IOAM nodes traversed and will compare the result to the last node's node_data-sign value. If the two values match, the IOAM data was not tampered with.

4.1.1. Overhead consideration for Method 1

Assuming e.g Ed25519, the public keys would have a size of 256 bits / 32 bytes, and as such signatures would be 512 bits / 64 bytes wide. node_data_sign[x] would consume 64 bytes per hop. Note that depending on the deployment, weaker keys might well apply, given that the provided integrity check is an online method, i.e. packets are verified as they arrive. This allows an attacker only a short time-window.

4.2. Method 2: Using symmetric keys for signing node trace data

The same procedure as Method 1 can be followed by using a MAC (Message Authentication Code) algorithm for node signature. This involves distributing a secret key to the individual IOAM nodes and the Validator. Steps 1 to 4 of Method 1 apply in a similar way, the only difference is that symmetric keys are used. As such, each node data list [x] field is extended with an additional "signed node-data" field: node_data_sign[x]. The size of the node_data_sign[x] field depends on the cryptographic message authentication code used.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                        node_data_sign [x]                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                                |
|                        node data list [x]                      |
|                                                                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.2.1. Overhead consideration for Method 2

Different types of cryptographic message authentication codes could be chosen, such as HMAC-SHA256 or Poly1305-AES.

HMAC-SHA256 would take a secret key of any size and provide a 32 byte authenticator. Consequently, node_data_sign[x] would consume 32 bytes per hop.

Poly1305-AES would use a 32 bytes secret key and provide a 16 byte authenticator. Consequently, node_data_sign[x] would consume 16 bytes per hop.

4.3. Method 3: Space optimized symmetric key based signing of trace data

Methods 1 and 2 add a node_data_sign field at every IOAM node the packet traverses. While feasible for network domains with only a few IOAM enabled hops, the number of bytes consumed in case of larger networks might not be acceptable. For those deployments, an approach with a single fixed sized signature field could apply.

Method 3 enhances the IOAM Trace-Option header to carry a "Trace Signature" field.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Namespace-ID           |NodeLen  | Flags | RemainingLen|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               IOAM-Trace-Type                 |  Reserved     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Trace Signature                          ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Method 3 assumes that symmetric keys have been distributed to the respective nodes as well as the Validator (the Validator receives all the keys). The details of the mechanisms of how keys are distributed are outside the scope of this document. The "Trace Signature" field is populated as follows:

1. The first node creates a seed and sign/HMAC over the hash of its node_data_list[x], the seed and its symmetric key. The seed can be included as a field in first node data or the seed can be the trailer to the trace option. The resulting HMAC/signature is included in the Trace Signature field.

2. Subsequent nodes will update the Trace Signature field by creating a signature/HMAC of data where the data is [Trace Signature || its node_data_list[x] hash] with its symmetric key.

3. The Validator will iteratively recreate the Trace Signature over the node data trace fields collected and matches the Trace Signature field to validate the trace data integrity.

4.3.1. Overhead consideration for Method 3

Much like method 2, the Trace Signature would consume 16 or 32 bytes - though with method 3, the Trace Signature is only carried once for the entire packet.

4.4. Method 4: Dynamic symmetric keys based signing of trace data

This method builds on top of Method 3 leverages Post-quantum Secure Pre-shared key distribution for deriving a dynamic symmetric key for every packet or a set of packets. The method utilizes the dynamic keys to provide for replay protection and does not require a seed to be added to the trace data to protect from replays because a private key is derived for each packet. The method relies on a local service that generates common Key/KeyID pairs for the participating Node and Validator (see the figure below). This common key generator uses ratcheting cryptography to generate the next secret while forgetting about the previous one. A unique ID is paired with each secret generated. Given the same seed secret as input parameter, two implementations of the common key generator will generate the exact same key and associated ID. The common key generator can be queried for the next key or for a specific key ID.

The figure below illustrates the concept:

```
         Validator                                         Node
             |                                              |
             |                                              |
       Generate McEliece                                    |
       public/private key-pair                              |
             |                                              |
             |<---Establ. classic secure connection---------|
             |               (e.g. TLS)                     |
             |---Send public key over secure connection---->|
             |                                              |
             |                             Generate random secret seed
             |                               and encrypt w/ Validator
             |                                        public key
             |                                              |
             |<--Send encrypted seed over secure connection-|
             |                                              |
      Decrypt secret seed sent from Node                    |
         using Validator's private key                      |
             |                                              |
             (-- Common secret seed established between   --)
             (--       Node and Validator                 --)
             |                                              |
             |                             Generate Node's KeyID pair
             |                             based on common secret seed
             |                                              |
             |           Use Node's key to update Trace Signature field
             |           in trace option header. Include Node's KeyID
             |                           in the extended node data.
             |                                              |
             (--        Packet reaches Validator          --)
             |                                              |
       Get Node's key using Node's KeyID                    |
       present in extended node data.                       |
       Validate Trace Signature using Node's key.           |
```

The main steps of method 4 are:

1. Each node will establish a common secret seed establishment using McEliece [McEliece] with the Validator.

2. Each node will then use the seed to generate a symmetric key per packet and use it in updating the Trace Signature field in the IOAM Trace-Option header over its node data hash. The node data is extended to include the KeyID of the dynamic key generated.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            KeyID [x]                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                        node data list [x]                     |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3. The Validator will validate the Trace Signature by deducing the key for each node using the KeyID.

The detailed mechanisms how keys and seeds are exchanged between nodes are outside the scope of this document.

4.4.1. Overhead consideration for Method 4

Like with method 3, the Trace Signature is only carried once for the entire packet and could be 32 bytes total. In addition, the KeyID needs to be added on a per hop basis. For sizing the Key ID, similar considerations like those for proof-of-transit packet random numbers apply - i.e. it depends on the packet rates of quickly keys are consumed. E.g. assuming a packet rate of 100Gbps and a KeyID space of 64 bits / 8 bytes, the system would need to be re-keyed after 3100 years (see also [I-D.ietf-sfc-proof-of-transit]). If frequent re-keying is feasible, 32 bits for KeyID might well be feasible.

5. IANA Considerations

This document is to support the IPPM working group to design and specify a solution for protecting the integrity of IOAM data fields. It does not include any requests to IANA.

6. Security Considerations

This section will be completed in a future revision of this document.

7. Acknowledgements

The authors would like to thank Santhosh N, Rakesh Kandula, Saiprasad Muchala, Greg Mirsky, Benjamin Kaduk and Martin Duke for their comments and advice.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

8.2. Informative References

[BLS] "BLS (Boneh-Lynn-Shacham) digital signature", 2021, <https://en.wikipedia.org/wiki/BLS_digital_signature>.

[EdDSA25519] "Edwards-curve Digital Signature Algorithm (EdDSA)", 2021, <https://en.wikipedia.org/wiki/EdDSA#Ed25519>.

[I-D.brockners-ippm-ioam-geneve] Brockners, F., Bhandari, S., Govindan, V., Pignataro, C., Nainar, N., Gredler, H., Leddy, J., Youell, S., Mizrahi, T., Lapukhov, P., Gafni, B., Kfir, A., and M. Spiegel, "Geneve encapsulation for In-situ OAM Data", draft-brockners-ippm-ioam-geneve-05 (work in progress), November 2020.

[I-D.ietf-detnet-security] Grossman, E., Mizrahi, T., and A. Hacker, "Deterministic Networking (DetNet) Security Considerations", draft-ietf-detnet-security-13 (work in progress), December 2020.

[I-D.ietf-ippm-ioam-data] Brockners, F., Bhandari, S., and T. Mizrahi, "Data Fields for In-situ OAM", draft-ietf-ippm-ioam-data-11 (work in progress), November 2020.

[I-D.ietf-ippm-ioam-ipv6-options] Bhandari, S., Brockners, F., Pignataro, C., Gredler, H., Leddy, J., Youell, S., Mizrahi, T., Kfir, A., Gafni, B., Lapukhov, P., Spiegel, M., Krishnan, S., Asati, R., and M. Smith, "In-situ OAM IPv6 Options", draft-ietf-ippm-ioam-ipv6-options-04 (work in progress), November 2020.

[I-D.ietf-nvo3-geneve]
Gross, J., Ganga, I., and T. Sridhar, "Geneve: Generic Network Virtualization Encapsulation", draft-ietf-nvo3-geneve-16 (work in progress), March 2020.

[I-D.ietf-sfc-ioam-nsh]
Brockners, F. and S. Bhandari, "Network Service Header (NSH) Encapsulation for In-situ OAM (IOAM) Data", draft-ietf-sfc-ioam-nsh-05 (work in progress), December 2020.

[I-D.ietf-sfc-proof-of-transit]
Brockners, F., Bhandari, S., Mizrahi, T., Dara, S., and S. Youell, "Proof of Transit", draft-ietf-sfc-proof-of-transit-08 (work in progress), November 2020.

[McEliece]
McEliece, R., "A Public-Key Cryptosystem Based on Algebraic Coding Theory", 1978, <https://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF>.

[RFC7276] Mizrahi, T., Sprecher, N., Bellagamba, E., and Y. Weingarten, "An Overview of Operations, Administration, and Maintenance (OAM) Tools", RFC 7276, DOI 10.17487/RFC7276, June 2014, <https://www.rfc-editor.org/info/rfc7276>.

[RFC7384] Mizrahi, T., "Security Requirements of Time Protocols in Packet Switched Networks", RFC 7384, DOI 10.17487/RFC7384, October 2014, <https://www.rfc-editor.org/info/rfc7384>.

[RFC7799] Morton, A., "Active and Passive Metrics and Methods (with Hybrid Types In-Between)", RFC 7799, DOI 10.17487/RFC7799, May 2016, <https://www.rfc-editor.org/info/rfc7799>.

[RFC8300] Quinn, P., Ed., Elzur, U., Ed., and C. Pignataro, Ed., "Network Service Header (NSH)", RFC 8300, DOI 10.17487/RFC8300, January 2018, <https://www.rfc-editor.org/info/rfc8300>.

Authors' Addresses

Frank Brockners
Cisco Systems, Inc.
Hansaallee 249, 3rd Floor
DUESSELDORF, NORDRHEIN-WESTFALEN 40549
Germany

Email: fbrockne@cisco.com

Shwetha Bhandari
Thoughtspot
3rd Floor, Indiqube Orion, 24th Main Rd, Garden Layout, HSR Layout
Bangalore, KARNATAKA 560 102
India

Email: shwetha.bhandari@thoughtspot.com

Tal Mizrahi
Huawei
8-2 Matam
Haifa 3190501
Israel

Email: tal.mizrahi.phd@gmail.com