DOTS
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

T. Reddy
D. Wing
Cisco
M. Boucadair
Orange
K. Nishizuka
NTT Communications
L. Xia
Huawei

October 27, 2016

Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel

draft-reddy-dots-data-channel-01

Abstract

The document specifies a Distributed Denial-of-Service Open Threat Signaling (DOTS) data channel used for bulk exchange of data not easily or appropriately communicated through the DOTS signal channel under attack conditions. This is a companion document to the DOTS signal channel specification.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 30, 2017.

Table of Contents

1. Introduction 2
2. Notational Conventions and Terminology 4
3. DOTS Data Channel 4
3.1. DOTS Provisioning 5
3.1.1. Create Identifiers 5
3.1.2. Delete Identifier 7
3.1.3. Retrieving Installed Identifiers 8
3.2. Filtering Rules 9
3.2.1. Install Filtering Rules 10
3.2.2. Remove Filtering Rules 12
3.2.3. Retrieving Installed Filtering Rules 13
4. IANA Considerations 14
5. Security Considerations 14
6. Acknowledgements 15
7. References 15
7.1. Normative References 15
7.2. Informative References 16
Authors' Addresses 17

1. Introduction

A distributed denial-of-service (DDoS) attack is an attempt to make machines or network resources unavailable to their intended users. In most cases, sufficient scale can be achieved by compromising enough end-hosts and using those infected hosts to perpetrate and amplify the attack. The victim in this attack can be an application server, a client, a router, a firewall, or an entire network.

DDoS Open Threat Signaling (DOTS) defines two channels: signal and data channels [I-D.ietf-dots-architecture] (Figure 1). The DOTS signal channel used to convey that a network is under a DDOS attack to an upstream DOTS server so that appropriate mitigation actions are undertaken on the suspect traffic is further elaborated in [I-D.reddy-dots-signal-channel]. The DOTS data channel is used for infrequent bulk data exchange between DOTS agents in the aim to significantly augment attack response coordination.

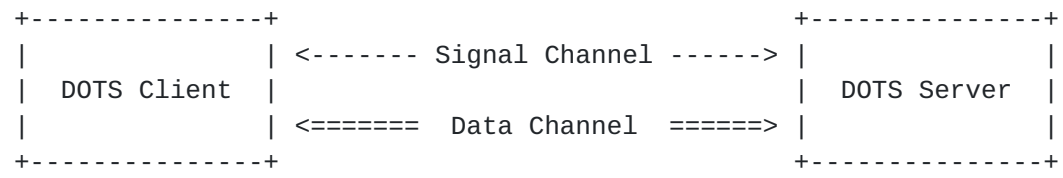

Figure 1: DOTS Channels

Section 2 of [I-D.ietf-dots-architecture] identifies that the DOTS data channel is used to perform the tasks listed below:

- Filter management, which enables a DOTS client to install or remove traffic filters dropping or rate-limiting unwanted traffic and permitting white-listed traffic. Sample use cases for populating black- or white-list filtering rules are detailed hereafter:

  A. If a network resource (DOTS client) detects a potential DDoS attack from a set of IP addresses, the DOTS client informs its servicing router (DOTS gateway) of all suspect IP addresses that need to be blocked or black-listed for further investigation. The DOTS client could also specify a list of protocols and ports in the black-list rule. That DOTS gateway in-turn propagates the black-listed IP addresses to the DOTS server which will undertake appropriate action so that traffic from these IP addresses to the target network (specified by the DOTS client) is blocked.

  B. An enterprise network has partner sites from which only legitimate traffic arrives and the enterprise network wants to ensure that the traffic from these sites is not penalized during DDOS attacks. The DOTS client uses DOTS data channel to convey the white-listed IP addresses or prefixes of the partner sites to its DOTS server. The DOTS server uses this information to white-list flows from such IP addresses or prefixes reaching the enterprise network.

- Creating identifiers, such as names or aliases, for resources for which mitigation may be requested:

  A. The DOTS client may submit to the DOTS server a collection of prefixes it wants to refer to by alias when requesting mitigation, to which the server would respond with a success status and the new prefix group alias, or an error status and message in the event the DOTS client's data channel request failed (see requirement OP-006 in [I-D.ietf-dots-requirements] and Section 2 in [I-D.ietf-dots-architecture]).

2. Notational Conventions and Terminology

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].

The reader should be familiar with the terms defined in [I-D.ietf-dots-architecture].

3. DOTS Data Channel

The DOTS data channel is intended to be used for bulk data exchanges between DOTS agents. Unlike the signal channel, which must operate nominally even when confronted with despite signal degradation due to packet loss, the data channel is not expected to be constructed to deal with attack conditions.

As the primary function of the data channel is data exchange, a reliable transport is required in order for DOTS agents to detect data delivery success or failure. Constrained Application Protocol (CoAP) [RFC7252] over TLS [RFC5246] over TCP is used for DOTS data channel (Figure 2). COAP was designed according to the REST architecture, and thus exhibits functionality similar to that of HTTP, it is quite straightforward to map from CoAP to HTTP and from HTTP to CoAP.

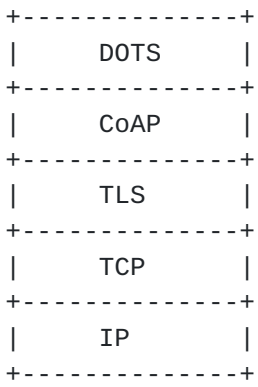

Figure 2: Abstract Layering of DOTS data channel over CoAP over TLS

JSON [RFC7159] payloads is used to convey DOTS signal channel session configuration, filtering rules as well as data channel specific payload messages that convey request parameters and response information such as errors. All data channel URIs defined in this document, and in subsequent documents, MUST NOT have a URI containing "/DOTS-signal".

A single DOTS data channel between DOTS agents can be used to exchange multiple requests and multiple responses. To reduce DOTS client and DOTS server workload, DOTS client SHOULD re-use the TLS session.

3.1. DOTS Provisioning

A DOTS client registers itself to its DOTS server(s) in order to set up DOTS related configuration and policy information exchange between the two DOTS agents.

3.1.1. Create Identifiers

A POST request is used to create identifiers, such as names or aliases, for resources for which a mitigation may be requested. Such identifiers may then be used in subsequent DOTS signal channel exchanges to refer more efficiently to the resources under attack (Figure 3).

```
  Header: POST (Code=0.02)
  Uri-Host: "host"
  Uri-Path: ".well-known"
  Uri-Path: "version"
  Uri-Path: "DOTS-data-channel"
  Uri-Path: "identifier"
  Content-Format: "application/json"
  {
     "policy-id": "integer",
     "id": { "alias-name" : [
                               "traffic-protocol": "string",
                               "destination-protocol-port": "string",
                               "destination-ip": "string",
                             ],
              "alias-name" :  [
                               "FQDN": "string",
                             ],
              "alias-name" :  [
                                "URI": "string",
                             ],
            }
              "alias-name" :  [
                                "E.164": "string",
                             ]
            }
   }
```

Figure 3: POST to create identifiers

The header fields are described below:

policy-id: Identifier of the policy represented using an integer. This identifier MUST be unique for each policy bound to the DOTS client, i.e., the policy-id needs to be unique relative to the active policies with the DOTS server. This identifier MUST be generated by the client. This document does not make any assumption about how this identifier is generated. This is a mandatory attribute.

alias-name: Name of the alias. This is a mandatory attribute.

traffic-protocol: Valid protocol values include tcp, udp, sctp, and dccp. Protocol values are separated by commas (e.g., "tcp, udp"). This is an optional attribute.

destination-protocol-port: The destination port number. Ports are separated by commas and port number range (using "-"). For TCP, UDP, SCTP, or DCCP: the destination range of ports (e.g., 80-8080). This information is useful to avoid disturbing a group of customers when address sharing is in use [RFC6269]. This is an optional attribute.

destination-ip: The destination IP address or prefix. IP addresses and prefixes are separated by commas. Prefixes are represented using CIDR [RFC4632] notation. This is an optional attribute.

FQDN: Fully Qualified Domain Name, is the full name of a system, rather than just its hostname. For example, "venera" is a hostname, and "venera.isi.edu" is an FQDN. This is an optional attribute.

URI: Uniform Resource Identifier (URI). This is an optional attribute.

E.164: E.164 number. This is an optional attribute.

In the POST request at least one of the attributes traffic-protocol or destination-protocol-port or destination-ip or FQDN or URI or E.164 MUST be present.

Figure 4 shows a POST request to create alias called "https1" for HTTPS servers with IP addresses 2002:db8:6401::1 and 2002:db8:6401::2 listening on port 443.

```
Header: POST (Code=0.02)
Uri-Host: "www.example.com"
Uri-Path: ".well-known"
Uri-Path: "v1"
Uri-Path: "DOTS-data-channel"
Uri-Path: "identifier"
Content-Format: "application/json"
{
 "policy-id": 123321333242,
 "id": { "Server1" : [
                     "traffic-protocol": "tcp",
                     "destination-protocol-port": "443",
                     "destination-ip": "2002:db8:6401::1,
                                        2002:db8:6401::2",
                     ]
       }
 }
```

Figure 4: POST to create identifiers

The DOTS server indicates the result of processing the POST request using CoAP response codes. CoAP 2.xx codes are success, CoAP 4.xx codes are some sort of invalid requests and 5.xx codes are returned if the DOTS server has erred or it is incapable of accepting the alias. Response code 2.01 (Created) will be returned in the response if the DOTS server has accepted the alias. If the request is missing one or more mandatory attributes, then 4.00 (Bad Request) will be returned in the response or if the request contains invalid or unknown parameters then 4.02 (Invalid query) will be returned in the response. The CoAP response will include the JSON body received in the request.

3.1.2. Delete Identifier

A DELETE request is used to delete an identifier maintained by a DOTS server (Figure 5).

```
Header: DELETE (Code=0.04)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Uri-Path: "identifier"
Content-Format: "application/json"
 {
   "policy-id": "number"
 }
```

Figure 5: DELETE identifier

If the DOTS server does not find the policy number conveyed in the DELETE request in its policy state data, then it responds with a 4.04 (Not Found) error response code. The DOTS server successfully acknowledges a DOTS client's request to remove the identifier using 2.02 (Deleted) response code.

3.1.3. Retrieving Installed Identifiers

A GET request is used to retrieve the set of installed identifiers from a DOTS server.

Figure 6 shows how to retrieve all the identifiers that were instantiated by the DOTS client while Figure 7 shows how to retrieve a specific identifier.

```
Header: GET (Code=0.01)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Uri-Path: "identifier"
```

Figure 6: GET to retrieve all the installed identifiers

```
Header: GET (Code=0.01)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Uri-Path: "identifier"
Uri-Path: "policy-id value"
```

Figure 7: GET to retrieve the specific identifier

Figure 8 shows response for all identifiers on the DOTS server.

```
{
  "policy-data":[
    {
      "policy-id": 1234534333242
      "id": { "Server1" : [
                           "traffic-protocol": "tcp",
                           "destination-protocol-port": "443",
                           "destination-ip": "2002:db8:6401::1,
                                              2002:db8:6401::2",
                          ]
            }
    },
    {
      "policy-id": 1233213344443
      "id": { "Server2" : [
                            "traffic-protocol": "tcp",
                            "destination-protocol-port": "80",
                            "destination-ip": "2002:db8:6401::10,
                                               2002:db8:6401::20",
                         ]
            }
    }
  ]
}
```

Figure 8: Response body

If the DOTS server does not find the policy number conveyed in the GET request in its policy state data, then it responds with a 4.04 (Not Found) error response code.

3.2. Filtering Rules

One of the possible arrangements for a DOTS client to signal filtering rules to a DOTS server via the DOTS gateway is discussed below:

The DOTS data channel conveys the filtering rules to the DOTS gateway. The DOTS gateway validates if the DOTS client is authorized to signal the filtering rules and if the client is authorized propagates the rules to the DOTS server. Likewise, the DOTS server validates if the DOTS gateway is authorized to signal the filtering rules. To create or purge filters, the DOTS client sends CoAP requests to the DOTS gateway. The DOTS gateway validates the rules and proxies the requests containing the filtering rules to a DOTS server. When the DOTS gateway receives the associated CoAP response from the DOTS server, it propagates the response back to the DOTS client.

The following APIs define means for a DOTS client to configure filtering rules on a DOTS server.

3.2.1. Install Filtering Rules

A POST request is used to push filtering rules to a DOTS server (Figure 9).

```
  Header: POST (Code=0.02)
  Uri-Host: "host"
  Uri-Path: ".well-known"
  Uri-Path: "version"
  Uri-Path: "DOTS-data-channel"
  Uri-Path: "filter"
  Content-Format: "application/json"
  {
     "policy-id": "integer",
     "traffic-protocol": "string",
     "source-protocol-port": "string",
     "destination-protocol-port": "string",
     "destination-ip": "string",
     "source-ip": "string",
     "lifetime": "number",
     "dscp": "string",
     "traffic-rate" : "number"
   }
```

Figure 9: POST to install filtering rules

The header fields are described below:

policy-id: An identifier of the policy represented as an integer. This identifier MUST be unique for each policy bound to the DOTS client, i.e., the policy-id needs to be unique relative to the active policies with the DOTS server. This identifier MUST be generated by the client. This document does not make any assumption about how this identifier is generated. This is a mandatory attribute.

traffic-protocol: Valid protocol values include tcp, udp, sctp, and dccp. Protocol values are separated by commas (e.g., "tcp, udp"). This is an optional attribute.

source-protocol-port: The source port number. Ports are separated by commas and port number range (using "-"). For TCP, UDP, SCTP, or DCCP: the source range of ports (e.g., 1024-65535). This is an optional attribute.

destination-protocol-port: The destination port number. Ports are separated by commas and port number range (using "-"). For TCP, UDP, SCTP, or DCCP: the destination range of ports (e.g., 443-443). This information is useful to avoid disturbing a group of customers when address sharing is in use [RFC6269]. This is an optional attribute.

destination-ip: The destination IP address or prefix. IP addresses and prefixes are separated by commas. Prefixes are represented using CIDR notation. This is an optional attribute.

source-ip: The source IP addresses or prefix. IP addresses and prefixes are separated by commas. Prefixes are represented using CIDR notation. This is an optional attribute.

lifetime: Lifetime of the rule in seconds. Upon the expiry of this lifetime, and if the request is not refreshed, this particular rule is removed. The rule can be refreshed by sending the same message again. The default lifetime of the rule is 60 minutes -- this value was chosen to be long enough so that refreshing is not typically a burden on the DOTS client, while expiring the rule where the client has unexpectedly quit in a timely manner. A lifetime of zero indicates indefinite lifetime for the rule. The server MUST always indicate the actual lifetime in the response. This is an optional attribute.

dscp: Differentiated services code point (DSCP) value in the IP header of a packet. This is an optional attribute.

traffic-rate: This is the allowed traffic rate in bytes per second indicated in IEEE floating point [IEEE.754.1985] format. The value 0 indicates all traffic for the particular flow to be discarded. This is a mandatory attribute.

In the POST request at least one of the attributes traffic-protocol or source-protocol-port or destination-protocol-port or destination-ip or source-ip MUST be present. The relative order of two rules is determined by comparing their respective policy identifiers. The rule with higher numeric policy identifier value has higher precedence (and thus will match before) than the rule with lower numeric policy identifier value.

Figure 10 shows a POST request to block traffic from an attacker using 2001:db8:abcd:3f01::/64 IPv6 prefix to a network resource reachable at IP address 2002:db8:6401::1 to operate a server on TCP port 443.

```
Header: POST (Code=0.02)
Uri-Host: "www.example.com"
Uri-Path: ".well-known"
Uri-Path: "v1"
Uri-Path: "DOTS-data-channel"

Content-Format: "application/json"
 {
   "policy-id": 123321333242,
   "traffic-protocol": "tcp",
   "source-protocol-port": "0-65535",
   "destination-protocol-port": "443",
   "destination-ip": "2001:db8:abcd:3f01::/64",
   "source-ip": "2002:db8:6401::1",
   "lifetime": 1800,
   "traffic-rate": 0
 }
```

Figure 10: POST to Install filterng rules

The DOTS server indicates the result of processing the POST request using CoAP response codes. CoAP 2.xx codes are success, CoAP 4.xx codes are some sort of invalid request and 5.xx codes are returned if the DOTS server has erred or is incapable of configuring the filtering rules. Response code 2.01 (Created) will be returned in the response if the DOTS server has accepted the filtering rules. If the request is missing one or more mandatory attributes then 4.00 (Bad Request) will be returned in the response or if the request contains invalid or unknown parameters then 4.02 (Invalid query) will be returned in the response. The CoAP response will include the JSON body received in the request.

3.2.2. Remove Filtering Rules

A DELETE request is used to delete filtering rules from a DOTS server (Figure 11).

```
Header: DELETE (Code=0.04)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Content-Format: "application/json"
 {
   "policy-id": "number"
 }
```

Figure 11: DELETE to remove the filtering rules

If the DOTS server does not find the policy number conveyed in the DELETE request in its policy state data, then it responds with a 4.04 (Not Found) error response code. The DOTS server successfully acknowledges a DOTS client's request to withdraw the filtering rules using 2.02 (Deleted) response code, and removes the filtering rules as soon as possible.

3.2.3. Retrieving Installed Filtering Rules

The DOTS client periodically queries the DOTS server to check the counters for installed filtering rules. A GET request is used to retrieve filtering rules from a DOTS server.

Figure 12 shows how to retrieve all the filtering rules programmed by the DOTS client while Figure 13 shows how to retrieve specific filtering rules programmed by the DOTS client.

```
Header: GET (Code=0.01)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Uri-Path: "list"
```

Figure 12: GET to retrieve the filtering rules (1)

```
Header: GET (Code=0.01)
Uri-Host: "host"
Uri-Path: ".well-known"
Uri-Path: "version"
Uri-Path: "DOTS-data-channel"
Uri-Path: "policy-id value"
```

Figure 13: GET to retrieve the filtering rules (2)

Figure 14 shows response for all active policies on the DOTS server.

```
{
  "policy-data":[
    {
      "policy-id":123321333242,
      "traffic-protocol": "tcp",
      "source-protocol-port": "0-65535",
      "destination-protocol-port": "443",
      "destination-ip": "2001:db8:abcd:3f01::/64",
      "source-ip": "2002:db8:6401::1",
      "lifetime": 1800,
      "traffic-rate": 0,
      "match-count": 689324,
    },
    {
      "policy-id":123321333242,
      "traffic-protocol": "udp",
      "source-protocol-port": "0-65535",
      "destination-protocol-port": "53",
      "destination-ip": "2001:db8:abcd:3f01::/64",
      "source-ip": "2002:db8:6401::2",
      "lifetime": 1800,
      "traffic-rate": 0,
      "match-count": 6666,
    }
  ]
}
```

Figure 14: Response body

If the DOTS server does not find the policy number conveyed in the GET request in its policy state data, then it responds with a 4.04 (Not Found) error response code.

4. IANA Considerations

TODO

[TBD: DOTS WG will probably have to do something similar to https://tools.ietf.org/html/rfc7519#section-10, create JSON DOTS claim registry and register the JSON attributes defined in this specification].

5. Security Considerations

Authenticated encryption MUST be used for data confidentiality and message integrity. TLS based on client certificate MUST be used for mutual authentication. The interaction between the DOTS agents requires Transport Layer Security (TLS) with a cipher suite offering

confidentiality protection and the guidance given in [RFC7525] MUST be followed to avoid attacks on TLS.

An attacker may be able to inject RST packets, bogus application segments, etc., regardless of whether TLS authentication is used. Because the application data is TLS protected, this will not result in the application receiving bogus data, but it will constitute a DoS on the connection. This attack can be countered by using TCP-AO [RFC5925]. If TCP-AO is used, then any bogus packets injected by an attacker will be rejected by the TCP-AO integrity check and therefore will never reach the TLS layer.

Special care should be taken in order to ensure that the activation of the proposed mechanism won't have an impact on the stability of the network (including connectivity and services delivered over that network).

Involved functional elements in the cooperation system must establish exchange instructions and notification over a secure and authenticated channel. Adequate filters can be enforced to avoid that nodes outside a trusted domain can inject request such as deleting filtering rules. Nevertheless, attacks can be initiated from within the trusted domain if an entity has been corrupted. Adequate means to monitor trusted nodes should also be enabled.

6. Acknowledgements

Thanks to Christian Jacquenet, Roland Dobbins, Andrew Mortensen, Roman Danyliw, and Gilbert Clark for the discussion and comments.

7. References

7.1. Normative References

[I-D.ietf-dots-architecture]
Mortensen, A., Andreasen, F., Reddy, T., christopher_gray3@cable.comcast.com, c., Compton, R., and N. Teague, "Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture", draft-ietf-dots-architecture-00 (work in progress), July 2016.

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <http://www.rfc-editor.org/info/rfc5925>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

7.2. Informative References

[I-D.ietf-dots-requirements]
Mortensen, A., Moskowitz, R., and T. Reddy, "Distributed Denial of Service (DDoS) Open Threat Signaling Requirements", draft-ietf-dots-requirements-02 (work in progress), July 2016.

[I-D.reddy-dots-signal-channel]
Reddy, T., Boucadair, M., Wing, D., and P. Patil, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Signal Channel", draft-reddy-dots-signal-channel-01 (work in progress), September 2016.

[IEEE.754.1985]
Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", August 1985.

[RFC4632] Fuller, V. and T. Li, "Classless Inter-domain Routing (CIDR): The Internet Address Assignment and Aggregation Plan", BCP 122, RFC 4632, DOI 10.17487/RFC4632, August 2006, <http://www.rfc-editor.org/info/rfc4632>.

[RFC6269] Ford, M., Ed., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, DOI 10.17487/RFC6269, June 2011, <http://www.rfc-editor.org/info/rfc6269>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

Authors' Addresses

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

Mohamed Boucadair
Orange
Rennes 35000
France

Email: mohamed.boucadair@orange.com

Kaname Nishizuka
NTT Communications
GranPark 16F 3-4-1 Shibaura, Minato-ku
Tokyo 108-8118
Japan

Email: kaname@nttv6.jp

Liang Xia
Huawei
101 Software Avenue, Yuhuatai District
Nanjing, Jiangsu 210012
China

Email: frank.xialiang@huawei.com