Session Initiation Protocol (SIP) Header Parameter for Debugging
draft-dawes-dispatch-debug-00

Abstract

Networks that use SIP to start and stop sessions between their users will frequently be upgraded with software and hardware changes. Users will similarly frequently change their client software and the way they use the network. In order to allow troubleshooting and regression testing, it is useful to provide debugging as part of the network fabric. This draft describes an event package that provides debugging configuration to SIP entities and a SIP private header that triggers logging of SIP signalling and identifies logs at multiple SIP entities as belonging to a single end-to-end session.

A list of related IETF drafts and non-IETF specifications is included to help develop this topic.

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 August 23, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
1. Introduction

If users experience problems with setting up sessions using SIP, their service provider needs to find out why by examining the SIP signalling. This draft defines an event package to configure SIP entities with conditions for starting and stopping logging of SIP signalling a SIP header field that allows a service provider to link signalling logged at various SIP entities in order to troubleshoot session setup.

The skeleton of the debugging procedure is as follows:

- The user’s terminal is prompted to enrol to debug configuration, supplied from a debug event package
- The first proxy the terminal connects to, at the edge of the network, either is already primed to log any signalling that is identified for debug, because it is permanently enrolled to receive debug configuration for all users, or is prompted to enrol in the same way as the terminal.
- The user’s terminal receives configuration, in the form of an XML file, that indicates to the terminal when it should start and stop logging signalling.
- When user’s terminal sends a SIP request that matches the pre-configured criteria for logging, logging starts at the user’s terminal, at the first proxy the terminal connects to, and at any other SIP entity within the trust domain that receives the request.
- Subsequent responses and requests in the same dialog are logged.
- Logging stops, either because the dialog has ended or because a ‘stop’ event, defined in the debug configuration, occurred.
- The user’s terminal, the proxy, and any other SIP entity that has logged signalling sends its logs to the debug server.

2. Requirements Language

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 RFC 2119 [RFC2119].
3. Related Drafts and Specifications

The following drafts are related to troubleshooting SIP calls.

- I-D.draft-kaplan-dispatch-session-id [I-D.kaplan-dispatch-session-id]
- I-D.worley-references [I-D.worley-references]
- I-D.kaithal-dispatch-sip-log-information [I-D.kaithal-dispatch-sip-log-information]
- I-D.jones-ipmc-session-id-reqts [I-D.jones-ipmc-session-id-reqts]
- I-D.jones-ipmc-session-id [I-D.jones-ipmc-session-id]

The following non-IETF specifications include requirements for troubleshooting and testing SIP calls.

- TS 32.422, Telecommunication management; Subscriber and equipment trace; Trace control and configuration management [3GPP.32.422] includes requirements for trace control and configuration for 3GPP networks, including the SIP-based IP Multimedia Subsystem (IMS).

- OMA Service Provider Environment Requirements Candidate Version 1.0 - 14 June 2005 [OMAOPSEV1] contains the Open Mobile Alliance requirements for regression testing after a network change or fix and for service troubleshooting.

The following RFC describes the standardization collaboration between 3GPP and IETF

- RFC3113 [RFC3113]

4. Motivating Scenario

Alice has a SIP client on her laptop, which she has been using for some time to make video calls to work colleagues inside her company, FooCorp, including making video calls and sending pager-mode messages. Last week, her company became able to contact staff working for its principal customer BarCorp, which recently installed a SIP-based network. Today, she tried to set up a call to Bob at BarCorp who uses an audio-only SIP phone, but the call failed and Alice does not know why. Also, she tried sending an instant message to her friend Carol, also working at BarCorp, and her terminal displayed 'message failed'. She contacts those who manage the SIP network within FooCorp, e.g. by phone or e-mail, to ask them to
investigate the problem.

This draft discusses the properties of a solution for debugging such a scenario, and outlines one possible solution.

5. Signalling for Example Scenario

5.1. General

The network administrators at FooCorp are first interested in whether the problem is within FooCorp or BarCorp. They would like to log the SIP signalling from Alice’s client to the edge of the FooCorp network. In order to do this, Alice’s client, the SIP entity at the border between FooCorp and BarCorp, and all of the SIP entities in between must log signalling for both the audio call and the instant message. The network administrators can then examine the logs to determine the cause of the problem.

5.2. Configuring Entities for Debugging

Before any debugging can be done, Alice’s SIP client needs configuration information that instructs the SIP client when to log SIP signalling. All debug configuration information at FooCorp is hosted on a single logical debug server, debug.foocorp.com, which hosts an event package that provides configuration using SUBSCRIBE and NOTIFY methods. Usually, SIP clients are not subscribed to this event package, since debugging is rarely used. Because debugging is rare, the debug event package should be subscribed to only when required, which is achieved by triggering subscription when Alice refreshes her registration. The administrators cause Alice to re-register by notifying her UA that its subscription has expired. When Alice’s UA re-registers, a session-ID header field with a debug parameter is included in the 200 OK response to the REGISTER request. This debug parameter causes Alice’s UA to subscribe to Alice’s debug event package at the debug server, which returns an XML document containing her debugging configuration. Typically, the Expires header field in the SUBSCRIBE request will have a 0 (zero) value because debugging is usually a one-off activity. Other than the NOTIFY request that triggers Alice’s SIP client to re-register and subscribe to the debug event package, signalling is as for the standardized framework for supplying configuration to a SIP client described in RFC 6080 [RFC6080].
### Figure 1: Prompting Client to Retrieve Debugging Configuration

<table>
<thead>
<tr>
<th>Step</th>
<th>Message and Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NOTIFY (Alice’s registration terminated)</td>
</tr>
<tr>
<td>2</td>
<td>REGISTER (Alice re-registers)</td>
</tr>
<tr>
<td>3</td>
<td>REGISTER</td>
</tr>
<tr>
<td>4</td>
<td>200 OK session-ID:;debug</td>
</tr>
<tr>
<td>5</td>
<td>200 OK session-ID:;debug</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
</tr>
<tr>
<td>7</td>
<td>SUBSCRIBE Event: debug</td>
</tr>
<tr>
<td>8</td>
<td>200 OK</td>
</tr>
<tr>
<td>9</td>
<td>NOTIFY (debug configuration in body)</td>
</tr>
<tr>
<td>10</td>
<td>200 OK</td>
</tr>
</tbody>
</table>

5.3. Originating Session

The XML document returned to Alice’s terminal contains the debugging configuration shown below. The schema for this XML file is described in detail in Section 16. This configuration instructs the terminal when to start logging, when to stop, and a value for the debug parameter added to the session-ID header field.
The start-trigger element instructs Alice’s terminal to begin to log signalling for any SIP request that contains bob@barcorp.com in the To: header field. The stop-trigger element instructs Alice’s terminal end logging signalling after a time period of two minutes. Alice’s terminal adds a debug parameter to the session-ID header field in all logged SIP requests, and the debug-control element contains the value that Alice’s terminal will assign to the debug parameter.

Proxy p1.foocorp.com is supplied with similar configuration, shown below, with one important difference, that the debug parameter value is part of the start trigger, thereby ensuring that the session from Alice is logged, not simply any request sent to Bob.

For all entities, debug configuration is used for a single dialog and then discarded, which means that once Alice’s UA has started the dialog with Bob, the debug configuration shown above is not re-used for any subsequent dialogs. The scope of logging is the dialog for which logging started, logging is not done of any other dialog that was in progress or is started while logging the dialog with Bob.

The FooCorp network is organized such that all SIP clients route requests through the first SIP proxy they connect to, and their
registrar, by using the path: and Service-Route: header fields. Other SIP proxies may also be on the signalling path.

The debugging configuration causes Alice’s UA and the first SIP proxy connected to Alice’s terminal to log SIP signalling the next time she sends an INVITE request to bob@barcorp.com. Alice retries calling Bob and signalling is logged for two minutes. Later examination of these logs shows that although requests and responses are correctly exchanged with Bob, Alice’s SIP client is not accepting audio-only sessions and is sending BYE immediately. This problem had not come to light previously as all calls within Alice’s company are video calls.

The outline call flow below illustrates how debugging works. Signalling logged at Alice’s UA and the Proxy shows that requests and responses are successfully exchanged, but Alice’s UA will not set up an audio-only session and sends BYE immediately.
Alice                  Proxy                    Bob
| (1) INVITE            |                       | (2) INVITE            |
| m = audio            |                       | debug value and From:|
| m = video            |                       | match debugging config|
| From:alice at atlanta.com |                   | so proxy starts       |
| Sessin-ID: ;debug="A076D1" |             | logging               |
| Alice’s UA starts logging |                 |---------------------->|

------------------------>

(3) 200 OK
m = audio
<---------------------->

(4) 200 OK
<--------------------->

(5) ACK
------------------------>

(6) ACK
------------------------>

(7) BYE
------------------------>

(8) BYE
------------------------>

(9) 200 OK
<---------------------->
Dialog has ended so
Proxy stops logging

(10) 200 OK
<--------------------->
Dialog has ended, so
Alice’s UA stops
logging

Figure 4: Example of Debugging
5.4. Terminating Sessions

Logging of a terminating session should start at the SIP proxy at the incoming edge of a network. For example, Bob has been told by Alice that her calls are not getting through and therefore asks the BarCorp network administrators to check any incoming calls from Alice. The proxy at the edge of the BarCorp network is provided with the configuration below to log any incoming calls from Alice. The <control> element contains the value for the debug header field parameter that the proxy will insert.

```
<session id="p01">
  <start-trigger>
    <to>bob@barcorp.com</to>
    <from>alice@foocorp.com</from>
  </start-trigger>
  <stop-trigger>
    <time-period>T0H2M0S</time-period>
  </stop-trigger>
  <control>
    <debug-id>2B346D</debug-id>
  </control>
</session>
```

Figure 5: Minimal Debugging Configuration for Proxy

When Alice calls Bob, the proxy at the edge of the BarCorp network begins logging and inserts a debug header field parameter with the value 2B346D taken from the configuration data.

6. Providing Configuration Data to the Terminal and Network

The configuration data required to trigger debugging in a network entity is an XML document, and this document must be delivered to network entities. RFC 6080 [RFC6080] describes a method for providing such configuration data, in this case of profile type "User Profile" as defined in clause 3.3 of RFC 6080 [RFC6080].

7. Avoiding Configuring all Entities on the Signalling Path

7.1. General

It is desirable to minimize the need for SIP entities to enrol for debug configuration for two reasons. Firstly, each enrollment
results in state maintained in the entity that enrolls and in the
debug server. Secondly, the path through proxies of a SIP request
cannot always be predicted, therefore an indication in the signalling
itself that this signalling should be logged is needed.

The requirements above can be met by one proxy policing the debug
header field parameter on behalf of all other proxies downstream.
Two cases are possible, a session originated at a terminal, and a
session that enters a network which will be terminated at a terminal
attached to that network.

7.2. Originating Sessions

Both the terminal and the proxy that it connects to at the edge of
the FooCorp network are configured with debug data. Since the
terminal is outside the trust domain, the edge proxy checks the debug
header field parameter inserted by the terminal, if any, against the
debug configuration data it has been supplied for that terminal. If
debug parameter should not have been inserted by the terminal, or
contains an incorrect value, the proxy removes it. If the SIP
request has no debug header field parameter but matches the debug
configuration data in the proxy, the proxy inserts a debug parameter
with the configured value.

7.3. Terminating Sessions

The SIP registrar for the address of record being debugged and the
terminating user’s UA are provided with debug configuration. The SIP
request passes through this registrar on its way to the terminating
UA and the registrar inserts a debug header field parameter. SIP
entities in the same trust domain and downstream of the registrar can
trust that the presence of the debug parameter indicates that they
should log that SIP request or response. The terminating user’s UA
is outside the trust domain and therefore requires its own
configuration data.

8. Multiple Simultaneous Events

At the same time as looking into the problem with calling Bob, the
administrators at FooCorp also want to find out why the message sent
to Carol caused an error display on Alice’s terminal. In order to do
this, they add the configuration below to the debug event package
hosted on the debug server. Each configuration fragment enclosed by
the tags <session id=> and </session> applies debugging to a single
SIP session, and in the same way that the number of simultaneous SIP
sessions is not restricted there is no restriction on how many
sessions are being simultaneously debugged. The configuration is a
new session that has a different id attribute to the previous session. This configuration is supplied to the terminal, and the terminal adds it to the session with id="u01" for debugging the problem with calling Bob.

```xml
<session id="u02">
  <start-trigger>
    <to>carol@barcorp.com</to>
  </start-trigger>
  <stop-trigger>
    <time-period>T0H2M0S</time-period>
  </stop-trigger>
  <debug-control>
    <debug-id>1A346E</debug-id>
  </debug-control>
</session>
```

Figure 6: Debugging Configuration for Instant Message

Alice then re-sends a message request to Carol and the call flow below is recorded.

```
Alice                    Proxy                     Carol
| (1) MESSAGE            |                         |
| From:alice@foocorp.com|                         |
| Session-ID:            |                         |
| ;debug=1A346E          |                         |
| Alice’s UA starts logging |-----------------------> |
| (2) MESSAGE            | debug value and To:    |
|                         | match debugging config |
|                         | so proxy starts        |
|                         | logging                |
|------------------------>|                         |
| (3) 501 Not Implemented| Dialog has ended, so   |
| Session-ID:            | proxy stops logging    |
| ;debug="1A346E"       |<------------------------|
|-----------------------|                         |
| Dialog has ended, so   |                         |
| Alice’s UA stops       |                         |
| logging                |                         |
```
Figure 7: Example of Debugging a MESSAGE Request

The signalling flow shows that Carol’s SIP UA is not able to process MESSAGE requests. In fact, Carol has an audio-only black phone. Logging for the MESSAGE request sent to Carol and the INVITE request sent to Bob happens simultaneously.

9. debug Parameter in SIP Requests

9.1. Forked Requests

Since forked requests are part of the same intention of the user to communicate, the debug header field parameter is copied unchanged from a single SIP request into all SIP requests that result from the forking.

9.2. Back-to-Back User Agents

Since requests generated by a B2BUA as a result of an incoming request that is being debugged are part of the same intention of the user to communicate, the debug header field parameter is copied unchanged from a SIP request into all new outgoing SIP requests that a B2BUA generates as a result of the incoming SIP request that contained the parameter.

10. debug Parameter in SIP Responses

The debug header field parameter is copied unchanged from a single SIP request into all responses, provisional and final, to that SIP request.

11. Multiple Service Providers

11.1. General

Foocorp is able to check signalling in its own network, but not in the network of Barcorp. Two solutions are possible, either entities in Barcorp are allowed to retrieve debugging configuration by sending a SUBSCRIBE request to the debug server in Foocorp, or Foocorp asks Barcorp to setup similar debugging in its own network to investigate why the MESSAGE request to Carol is failing. The debugging configuration in Barcorp would consist of logging signalling for requests that are incoming to Carol (i.e., with carol@barcorp.com in the From: header field).
12. Configuration for Multiple AORs

Any entity may subscribe to a URI that identifies a group of AORs. If multiple NOTIFY requests carry configuration information about the same AOR then the most recent configuration document is used. It might be that a new NOTIFY request adds a session to existing configuration for an AOR and otherwise leaves its existing configuration untouched.

13. Retrieving Debugging Logs

When logging finishes, either because the stop trigger event occurred, or because the dialog being logged has ended, the SIP entity sends logged signalling in the body of a PUBLISH request sent to the debug event server. If this PUBLISH request will cross a trust domain boundary, it MUST use authentication, integrity protection, and privacy protection. Logged signalling in the body of the PUBLISH request will typically be text such as MIME type text/plain of SIP requests and responses and their bodies. In order to correlate logged signalling, it might be useful to separate out the dialog ID as described in RFC 3261 [RFC3261] clause 12, the debug parameter value, and the Max Forwards: header field.

The debug event server reconstructs the flow of signalling using the dialog identity (Call-ID: header field and the tags in the To: and From: header fields) and the CSeq: and Max-Forwards: header fields.

14. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

14.1. Trust Domain

Since a non-empty header field parameter value may cause a SIP entity to log the SIP header and body of a request or response, the debug parameter must be removed at a trust domain boundary. If BarCorp is outside the trust domain of FooCorp, then BarCorp will not receive the debug parameter. However, the SIP entity at the edge of the BarCorp network can attempt to subscribe to the debug configuration for alice@foocorp.com and use this configuration to cause logging in the BarCorp network.
14.2. Security Threats

The identity carried by the debug header parameter is not sensitive information, although it will sometimes indicate that a particular device is experiencing problems. If the value in the header is maliciously changed, this will disrupt troubleshooting.

The presence of a debug header field parameter will cause some SIP entities to log signalling. Therefore, this header field must be removed at the earliest opportunity if it has been incorrectly inserted.

Debug configuration affects the operation of a terminal, therefore it must be supplied by an authorized server to an authorized terminal, it must not be altered in transit, and it must not be readable by an unauthorized third party.

Logged signalling is privacy-sensitive data, therefore it must be passed to an authorized server, it must not be altered in transit, and it must not be readable by an unauthorized third party.

14.3. Security Mechanisms

Security considerations are very similar to those in RFC 6080 [RFC6080], so the same mechanisms can be used to secure debugging configuration and logged signalling.

15. Formal Syntax

All of the mechanisms specified in this document are described in both prose and an augmented Backus-Naur Form (BNF) defined in RFC 2234 [RFC2234]. Further, several BNF definitions are inherited from SIP and are not repeated here. Implementors need to be familiar with the notation and contents of SIP RFC 3261 [RFC3261] and RFC 2234 [RFC2234] to understand this document.

15.1. debug Header Field Parameter Syntax

The syntax for the debug header field parameter is described as follows:

```
debug = "debug" EQUAL 6*6HEXDIG
```
16. XML Schema for Debug Configuration

Configuration for debugging is supplied as an XML document according to the schema in Figure 8.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:debuginfo"
xmlns:tns="urn:ietf:params:xml:ns:debuginfo"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <!-- This import brings in the XML language attribute xml:lang-->
schemaLocation="http://www.w3.org/2001/03/xml.xsd"/>
  <!-- debuginfo is the root element in debug configuration
     debuginfo contains one or more debugconfig elements, where one
     debugconfig element exists per address of record. -->
</xs:schema>
```

<!-- definition of simple elements -->
<xs:element name="time" type="xs:time"/>
<xs:element name="from" type="xs:string"/>
<xs:element name="to" type="xs:string"/>
<xs:element name="method" type="xs:string"/>
<xs:element name="icsi" type="xs:string"/>
<xs:element name="iari" type="xs:string"/>
<xs:element name="time-period" type="xs:duration"/>
<xs:element name="interface" type="xs:string"/>
<xs:element name="debug-id" type="xs:hexBinary"/>

<!-- definition of simple elements with restrictions -->
<xs:element name="reason">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="dialog_established"/>
      <xs:enumeration value="session_end"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

<xs:element name="depth">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="maximum"/>
      <xs:enumeration value="minimum"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
<!-- definition of attributes -->
<xs:attribute name="version" type="xs:nonNegativeInteger"/>
<xs:attribute name="aor" type="xs:string" minOccurs="1"
maxOccurs="1"/>
<xs:attribute name="id" type="xs:string" use="required"/>

<!-- definition of attributes with restrictions -->
<xs:attribute name="state">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="full"/>
      <xs:enumeration value="partial"/>
    </xs:restriction>
  </xs:simpleType>
</xs:attribute>

<!-- definition of complex elements -->
<!-- definition of complex elements -->
<xs:element name="debuginfo">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="debugconfig" minOccurs="0" maxOccurs="unbounded"/>
      <xs:any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="version" use="required"/>
    <xs:attribute ref="state" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="debugconfig">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="session" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="aor" use="required"/>
  </xs:complexType>
</xs:element>

<xs:element name="session">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="start-trigger"/>
      <xs:element ref="stop-trigger"/>
      <xs:element ref="control"/>
    </xs:sequence>
    <xs:attribute ref="id" use="required"/>
  </xs:complexType>
</xs:element>
Figure 8: XML schema for debugging configuration

17. References

17.1. Normative References

[I-D.kaplan-dispatch-session-id]
Kaplan, H., "A Session Identifier for the Session
17.2. Informative References

[3GPP.32.422]
3GPP, "Telecommunication management; Subscriber and equipment trace; Trace control and configuration management", 3GPP TS 32.422 10.6.0, December 2011.

[I-D.ietf-sipping-config-framework]

[I-D.jones-ipmc-session-id]

[I-D.jones-ipmc-session-id-reqts]

[I-D.kaithal-dispatch-sip-log-information]

[I-D.worley-references]

[OMAOPSEV1]
Open Mobile Alliance, "OMA Service Provider Environment Requirements Candidate Version 1.0 – 14 June 2005", 2005,


Appendix A. Additional Stuff

This becomes an Appendix.

Author’s Address

Peter Dawes
Vodafone Group
The Connection
Newbury, Berkshire  RG14 2FN
UK

Phone: +44 7717 275009
Email: peter.dawes@vodafone.com
End-to-End Session Identification in IP-Based Multimedia Communication Networks
draft-jones-ipmc-session-id-03.txt

Abstract

This document describes an end-to-end Session Identifier for use in IP-based Multimedia Communication systems that enables endpoints, intermediate devices, and management systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on July 30, 2012.
1. Introduction

IP-based multimedia communication systems like SIP [1] and H.323 [2] have the concept of a "call identifier" that is globally unique. The identifier is intended to represents an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, billing, session tracking, and so forth.

Unfortunately, there are a number of factors that contribute to the fact that the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. Perhaps most significant is the fact that the syntax for the call identifier in SIP and H.323 is different between the two protocols. This important fact makes it impossible for call identifiers to be exchanged end-to-end when a network utilizes one or more session protocols.

Another reason why the current call identifiers are not suitable to identify the session end-to-end is that in real-world deployments
devices like session border controllers often change the values as the session signaling passes through. This is true even when a single session protocol is employed and not a byproduct of protocol interworking.

This draft presents a new identifier, referred to as the Session Identifier or "Session ID", and associated syntax intended to overcome the issues that exist with the currently defined call identifiers. The proposal in this document attempts to comply with the requirements specified in [5].

2. Conventions used in this document

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 RFC 2119 [3].

3. Session Identifier Requirements

Requirements for the end-to-end Session Identifier can be found in a separate memo titled "Requirements for an End-to-End Session Identification in IP-Based Multimedia Communication Networks" [5].

4. Session Identifier Usage

The Session Identifier is intended to uniquely identify a communication session end-to-end. This document does not specify how the Session Identifier is to be used, but merely defines the identifier in such a way as to enable it to be used for situations encountered in real-world deployments of IP-based multimedia communication systems, including:

* End-to-end identification of a communication session

* Association of session signaling and media flows, made possible by including the session identifier in media-related messages (e.g., RSVP [7] or RTCP [8])

* Identification of devices taking part in the same multipoint conference

* Tracking sessions transferred from one endpoint to another

* Identification of recorded sessions

* Logging for the purposes of accounting, billing, debugging, etc.
5. Constructing the Session Identifier

The Session Identifier is comprised of two UUIDs [4] that are concatenated together, with each UUID created by the endpoints participating in the session. The first endpoint in the session will create a UUID and transmit that to the second endpoint. Likewise, the second endpoint will create a UUID and transmit that to the first endpoint. Each endpoint will then concatenate the two UUIDs to form the Session Identifier.

Note that it does not matter which endpoint might be considered the originating or terminating endpoint. For the purposes of session identification, it is only important that each endpoint create a UUID and transmit that value to the remote endpoint. Intermediaries such as session border controllers MUST NOT change any Session Identifier component received from an endpoint in a session.

What is also important is the order in which the UUIDs are concatenated together. To ensure that concatenation is performed consistently, a binary comparison is performed on the two UUIDs starting with the most significant byte. The UUID with the higher binary value is placed after the UUID with the lower binary value. Consider the following example.

Endpoint 1 produces this UUID: 0xaeffa652b22911df-a81f12313a006823

Endpoint 2 produces this UUID: 0xbe11afc8b22911df86c412313a006823

The resulting Session Identifier would be:
0xaeffa652b22911df-a81f12313a006823be11afc8b22911df86c412313a006823

In the above example, the UUIDs are presented as a string of hexadecimal characters that correspond to the binary values comprising the UUID as shown in the table at the end of Section 4.1.2 of RFC 4122 [4].

6. Transmitting the Session Identifier in SIP

Each session initiated or accepted MUST have a locally generated UUID associated with the session. This value MUST remain unchanged throughout the duration of the session and MUST persist even when the session is redirected (e.g., via a 3xx response) or transferred (e.g., via REFER [6]).

A SIP user agent MUST convey its Session Identifier UUID in all transmitted messages. To do this, each transmitted message MUST include the following header:

    Session-ID-UUID: aeffa652-b229-11df-a81f-12313a006823
In the above example, the UUID is presented in string form with hyphens inserted as shown in the UUID ABNF syntax shown in Section 3 of [4]. Note that the namespace-related syntax "urn:uuid:" is NOT present in the Session-ID-UUID header.

The formal Session-ID-UUID header syntax is:

```
Session-ID-UUID = "Session-ID-UUID" HCOLON UUID
```

The actual Session Identifier is derived, as described in the previous section, by concatenating the locally generated UUID value and the UUID value received from the remote endpoint.

Intermediaries that wish to utilize the Session Identifier must take note of the UUIDs transmitted in each direction between endpoints. Intermediaries MUST NOT alter the UUIDs. If performing interworking between SIP and another session protocol, the intermediary MUST convert the Session-ID-UUID header as necessary so that it preserves the value of the UUID.

Developers should understand that a session MAY be transferred at any point and without any explicit signaling. This is not uncommon for back-to-back user agents that provide various call control functions. When the session is transferred, joined, or merged, perhaps a new INVITE message might be received bearing a new Session-ID-UUID value. When a new UUID is received, the endpoint MUST compute a new Session Identifier value, as the session has in fact changed. The endpoint MUST NOT generate a new UUID in response, however.

7. Associating Endpoints in a Multipoint Conference

Multipoint Control Units (MCUs) group two or more sessions into a single multipoint conference. Each session that is grouped into a conference SHOULD utilize the same UUID from the MCU to each of the endpoints in the conference. In so doing, each individual session in the conference will have a unique Session Identifier (since each endpoint will create a unique UUID of its own), but will also have one UUID in common with all other participants in the conference.

Intermediary devices, such as proxies or session border controllers, or network diagnostics equipment might assume that when they see two or more sessions with different Session Identifiers, but with one UUID in common, that the sessions are part of the same conference.

Note, however, that this assumption is true only if the sessions are operating in parallel. If A tries to establish a session with B and B redirects the session to C, each of A, B, and C will share at least one UUID in common (i.e., the UUID created by A). Likewise, if B transfers the session between A and B to C, A will retain its UUID
and it will appear that A, B, and C are in a single conference. This is a desirable behavior, even though it may not be precise. It is assumed that any device that might wish to utilize this information would also recognize that a session is redirected or transferred.

8. Correlating Media Flows with Sessions

As mentioned previously, it may be desirable to insert the Session Identifier into media-related packets, such as RSVP messages or RTCP packets. In so doing, it is possible for network elements to

1. correlate session signaling with media flows,
2. associate multiple media flows with a single session, and
3. associate multiple media flows from multiple devices that are part of a single conference

Notwithstanding the foregoing, the use of the Session Identifier for purposes other than end-to-end session identification is outside the scope of this document.

9. Security Considerations

TBD

10. IANA Considerations

There are no IANA considerations associated with this document.

11. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

12. References

12.1. Normative References


12.2. Informative References


Author’s Addresses

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA
Phone: +1 919 476 2048
Email: paulej@packetizer.com
IM: xmpp:paulej@packetizer.com

Chris Pearce
Cisco Systems, Inc.
2300 East President George Bush Highway
Richardson, TX 75082
USA
Phone: +1 972 813 5123
Email: chrep@cisco.com
IM: xmpp:chrep@cisco.com

James Polk
Cisco Systems, Inc.
3913 Treemont Circle
Colleyville, Texas, USA
Phone: +1 817 271 3552
Email: jmpolk@cisco.com
IM: xmpp:jmpolk@cisco.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA
Phone: +1 919 392 3266
Email: gsalguei@cisco.com
IM: xmpp:gsalguei@cisco.com
Abstract

This document specifies the requirements for an end-to-end session identifier in IP-based multimedia communication networks. This identifier would enable endpoints, intermediate devices, and management and monitoring systems to identify a session end-to-end, associate multiple endpoints with a given multipoint conference, track communication sessions when they are redirected, and associate one or more media flows with a given communication session.

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 July 30, 2012.
1. Introduction

IP-based multimedia communication systems like SIP [1] and H.323 [2] have the concept of a "call identifier" that is globally unique. The identifier is intended to represent an end-to-end communication session from the originating device to the terminating device. Such an identifier is useful for troubleshooting, billing, session tracking, and so forth.

Unfortunately, there are a number of factors that contribute to the fact that the current call identifiers defined in SIP and H.323 are not suitable for end-to-end session identification. Perhaps most significant is the fact that the syntax for the call identifier in SIP and H.323 is different between the two protocols. This important fact makes it impossible for call identifiers to be exchanged end-to-end when a network utilizes one or more session protocols.
Another reason why the current call identifiers are not suitable to identify the session end-to-end is that in real-world deployments devices like session border controllers often change the values as the session signaling passes through. This is true even when a single session protocol is employed and not a byproduct of protocol interworking.

Lastly, identifiers that might have been used to identify a session end-to-end fail to meet that need when sessions are manipulated through supplementary service interactions. For example, when a session is transferred or if a PBX joins two communication sessions together locally, the end-to-end properties of currently-defined identifiers are lost.

This draft specifies the requirements for an end-to-end session identifier. With this draft, the authors would like to encourage discussion and progress work in the dispatch working group or working group as designated by the IETF, the outcome of which will be a new RFC that defines a session ID in conformance with these requirements.

2. Terminology

SIP defines additional terms used in this document that are specific to the SIP domain such as "proxy"; "registrar"; "redirect server"; "user agent server" or "UAS"; "user agent client" or "UAC"; "back-to-back user agent" or "B2BUA"; "dialog"; "transaction"; "server transaction".

In this document, the word "session" refers to a "communication session" that may exist between two SIP user agents or that might pass through one or more intermediary devices, including B2BUAs or SIP Proxies.

The term "end-to-end" in this document means the communication session from the point of origin, passing through any number of intermediaries, to the ultimate point of termination. It is recognized that legacy devices may not support the "end-to-end" session identifier, though an identifier might be created by an intermediary when it is absent from the session signaling.

3. Requirements for the End-to-End Session Identifier

REQ1: It must be possible for an administrator or an external device which monitors the SIP-traffic to use the identifier to identify a set of dialogs which have a relationship with each other, such that they represent the same SIP session, with as high a probability as possible.
REQ2: It must be possible to identify the end-to-end session when a session is transferred or if two different sessions are joined together via an intermediary (e.g., a PBX).

REQ3: It must be possible to identify all sessions participating in a multipoint or multi-party conference by observing the end-to-end session identifiers of each session.

REQ4: It must be possible to pass the identifier unchanged through SIP B2BUAs or other intermediaries.

REQ5: The identifier must not reveal any information related to any SIP device or domain identity, including IP Address, port, hostname, domain name, username, Address-of-Record, MAC address, IP address family, transport type, etc.

REQ6: The identifier must not reveal to the receiver of it that the Call-ID, tags, or any other SIP header or body portion have been changed by middleboxes, with as high a probability as possible.

REQ7: It must be possible to identify SIP traffic with an end-to-end session identifier from and to end devices that do not support this new identifier, such as by allowing an intermediary to inject an identifier into the session signaling.

REQ8: The identifier should be unique in time and space, similar to the Call-ID.

REQ9: The identifier should be constructed in such a way as to make it suitable for transmission in SIP, H.323, RSVP [3], and RTCP [4].

4. Session Identifier Use Cases

The Session Identifier is intended to uniquely identify a communication session end-to-end. This document does not specify how the Session Identifier is to be used, but merely defines the identifier in such a way as to enable it to be used for situations encountered in real-world deployments of IP-based multimedia communication systems, including:

* End-to-end identification of a communication session

* Association of session signaling and media flows, made possible by including the session identifier in media-related messages (e.g., RSVP or RTCP)

* Identification of devices taking part in the same multipoint conference
* Tracking sessions transferred from one endpoint to another

* Facilitate the recording of SIP sessions and correlating those sessions

* Logging for the purposes of accounting, billing, debugging, communication tracking (such as for security purposes in case of theft of service), etc.

5. Related Work in other Standards Organizations

5.1. Coordination with the ITU-T

IP multimedia networks are often comprised of a mix of session protocols like SIP and H.323. A benefit of the Session Identifier is that it uniquely identifies a communication session end-to-end across session protocol boundaries. Therefore, the need for coordinated standardization activities across Standards Development Organizations (SDOs) is imperative.

To facilitate this, a parallel effort is underway in the ITU-T to introduce the Session Identifier for the H.323 protocol. The ITU-T SG16 has approved contribution C.552 [5] as a work item with the intent that it be a coordinated and synchronized effort between the ITU-T and the IETF.

5.2. Requirements within 3GPP

3GPP identified in their Release 9 the need for a Session Identifier for O&M purposes to correlate flows in an end-to-end communication session. TS24.229 (IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP)) [6] points to the fact that the Session Identifier can be used to correlate SIP messages belonging to the same session. In the case where signaling passes through SIP entities like B2BUAs, the end-to-end session identifier indicates that these dialogs belong to the same end-to-end SIP communication session.

6. Security Considerations

An end-to-end identifier, if not properly constructed, could provide information that would allow one to identify the individual, device, or domain initiating or terminating a communication session. In adherence with REQ5, the solution produced in accordance with these requirements MUST NOT provide any information that allow one to identify a person, device, or domain. This means that information elements such as the MAC address or IP address MUST NOT be used when constructing the end-to-end session identifier.
7. IANA Considerations

There are no IANA considerations associated with this document.

8. Acknowledgments

The authors would like to acknowledge Chris Pearce for his contribution and collaboration in developing this document.

This document was prepared using 2-Word-v2.0.template.dot.

9. References

9.1. Normative References


9.2. Informative References


[6] 3GPP, "IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3", 3GPP TS 24.229 10.3.0, April 2011.
Author’s Addresses

Roland Jesske
Deutsche Telekom NP
64295 Darmstadt
Heinrich-Hertz-Str. 3-7
Germany

Phone: +49 6151 628 2766
Email: R.Jesske@telekom.de

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 476 2048
Email: paulej@packetizer.com
IM: xmpp:paulej@packetizer.com

Hadriel Kaplan
Acme Packet
71 Third Ave.
Burlington, MA 01803, USA

Email: hkaplan@acmepacket.com

Laura Liess
Deutsche Telekom NP
64295 Darmstadt
Heinrich-Hertz-Str. 3-7
Germany

Phone: +49 6151 268 2761
Email: laura.liess.dt@gmail.com

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com
James Polk
Cisco Systems, Inc.
3913 Treemont Circle
Colleyville, Texas,
USA

Phone: +1 817 271 3552
Email: jmpolk@cisco.com
IM: xmpp:jmpolk@cisco.com

Parthasarathi Ravindran
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka 560103
India

E-Mail: partr@cisco.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 392 3266
Email: gsalguei@cisco.com
IM: xmpp:gsalguei@cisco.com
Session Initiation Protocol (SIP) Extension for logging and debugging.
draft-kaithal-dispatch-sip-log-information-00

Abstract

The current mechanisms to debug issues in SIP network are not very
efficient. It requires to enable debugging logs across different
devices, recreate the problem and then collect the logs. The idea is
to provide a solution to automatically enable relevant logs (SIP
messages and any other debugging logs meaningful to SIP devices), and
also to indicate where the logs are to be collected or stored. The
enabling of logs will happen at all the SIP devices (upstream or
downstream). This will help to get the logs from all the SIP devices
in a Common logging format (CLF). The solution extends SIP to
provide the infrastructure to enable logging for upstream and
downstream devices with each server deciding how much troubleshooting
information it wants to log - with freedom to simply ignore requests
if required. This document specifies a new header called "Log-Me"
Header in all the SIP messages.

Status of this Memo

This Internet-Draft is submitted to IETF 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 15, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the
document authors. All rights reserved.
1. Introduction

Session Initiation Protocol (SIP) is gaining popularity in the VoIP Network. Many deployments, currently, have deployed SIP for their VoIP network. Because of the huge deployments, isolating the problem in the network and troubleshooting it becomes very difficult. Also, If the problem happens in high traffic condition, or happens intermittently, collecting the right set of debugging logs is very difficult for further fault analysis.

There is need for an effective troubleshooting mechanisms embedded in the signalling so that logs from all the devices can be collected for that particular call and stored in a common location for troubleshooting.

2. 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]. This document only uses these key words when referencing normative statements in existing RFCs.

3. Background

Troubleshooting in VOIP network is challenging as the signaling and media follow different path and Different equipment.

Scenario

UA1----B2BUA1-----P1-------P2-------B2BUA2------UA2

UA1 wishes to make a call to UA2, However, due to network Topology, UA1 has to go via B2BUA1, Proxy 1 (P1), Proxy2 (P2) and B2BUA2 to reach UA2. Likewise, all requests between UA2 and UA1 must also traverse through the same path.

UA1 makes a SIP call to UA2. This call has some issue. In order to debug the issue, logs need to be enabled at all the SIP devices in the network. It involves a lot of time and effort collect the logs and troubleshoot it.

The "Log-Me" extension header field allows all the SIP devices (upstream and downstream) to start collecting desired logs, store it in to a common location and available for debugging. It may store the data in Common Logging Format so that It can be fed to a device
which make the troubleshooting faster moreover the logs are in a
standard format and easier to troubleshoot. This header element’s
function is to invoke tracing and troubleshooting functions at each
"hop" in the signaling path for the call. Additionally, it includes
a unique "tag" to allow the information to be associated with the
particular call for proper correlation.

4. Applicability Statement

The Log-me mechanism is applicable to all the SIP entity in the
network which includes UAS, UAC, Proxy, B2BUA etc.

1) Each entity is free to simply ignore request, if it is not
interested in log collection, or busy with other activities.

2) Each entity can add its own "Log-Me" header and provide the path
for the log collection.

3) Each entity is free to log any information which is useful for
troubleshooting.

4) If an entity does not understand the header then it can simply
ignore it. However, it should not remove the header, as removing stops
the possibility for logging at the next hop.

5) If SIP signalling is secure then logging MUST be secure.

6) If B2BUA gets a "Log-Me" header then it MUST NOT remove it. It
MAY modify or append "Log-Me" header.

5. Log-me Header Field Definition and Syntax
Log-Me = "Log-Me" HCOLON log-value *(COMMA log-value)
log-value =  log-type  1*(SEMI log-params)
log-type =  "mailto" / "http"/ "syslog" / "tftp" /
           "ftp"/"sftp" / "local" / other-type
log-params = log-maddr /log-uri/
             log-username/log-password/log-tag
log-maddr= "maddr" EQUALS host
log-uri = "uri" EQUALS userinfo
userinfo = user "@" host
log-username = "username" EQUALS user
user = 1* ( unreserved / escaped / user-unreserved )
log-password= "password" EQUALS*( unreserved / escaped / "=" / "+" / "$" / "," )
log-tag = tag EQUALS token
other-type = token

Example :
Log-Me:mailto;uri=akaithal@cisco.com;tag=sdfrgf43
Log-Me:syslog;maddr=9.45.45.34;username=akaithal;
password=addfere2;tag=sdfdsfe
Log-Me:local;tag=sdfdsfe

local means it should store the data locally.

Support for the Log-me header field MAY be indicated by a UA by including the option-tag "log" in a Supported header field.

Log-me header field value MUST consist of exactly one log-type. If the log-type is mailto then it should have a log-uri (i.e. an email address). For any other log-type log-username.log-password and log-maddr is MUST. One entity can put more than one Log-me header in multiple line or separated by comma. In case there are more than one Log-me header then it is the intermediate end point’s decision to put the log in all places mentioned, or choose any one of them. If log-uri and log-username are present then user portion of log-uri and log-username MUST be same.

This is an optional header and can be built only in a SIP request and response. The header can also be inserted by any intermediate entity in the network.
<table>
<thead>
<tr>
<th>Header field</th>
<th>where</th>
<th>proxy</th>
<th>ACK</th>
<th>BYE</th>
<th>CAN</th>
<th>INV</th>
<th>OPT</th>
<th>REG</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-Me</td>
<td>R</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>1xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>2xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>3xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>4xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>5xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
<tr>
<td>6xx</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>-</td>
<td>o</td>
<td>o</td>
<td>o</td>
<td>o</td>
</tr>
</tbody>
</table>

If a SIP REFER message is sent to an endpoint and contains the "Log-Me" header, not only is the REFER method itself traced, but any call initiated via the REFER mechanism is also traced. If "Log-Me" header is present in the request then entity should at least log SIP messages and any other relevant information which is required for debugging.

6. UAC Behaviour

If UAC supports this extension then it should add "log" tag in the supported header in the request or response. In case UAC wants the call to be logged then it MUST add a Log-me header with the details to enable logging.

UAC receives response with the "Log-Me" header then it starts collecting the data from that response on words till the end of the call. It logs messages which is mandatory along with other logging information which is essential for troubleshooting from that device point of view.

7. UAS Behaviour

UAS receives request with the "Log-Me" header then it starts
collecting the data from that request on words till the end of the call. It logs messages which is mandatory along with other logging information which is essential for troubleshooting from that device point of view.

If UAS supports this extension the it should add debug tag in the supported header in the request or response. In case UAS wants the call to be logged then it MUST add a "Log-Me" header with the details in any of the responses to enable logging.

8. Proxy Behaviour

1) If proxy supports "log" extension and it gets "Log-Me" header in any of the request or response then it SHOULD process it and collect relevant logs. Additionally it can do one of the following
   a) Forward the received "Log-Me" header as is.
   b) Add additional "Log-Me" header with details.
   c) Add its own "Log-Me" header and removing the received "Log-Me" header.

The above action are applicable for the forked requests as well.

2) If proxy does not support "log" extension and it receives "Log-Me" header then it MUST NOT remove it from the requests or responses.

9. Security Considerations

There are security consideration with this header as password is exposed.

1) It is RECOMMENDED to have calls over TLS to send "Log-Me" header.

2) It is RECOMMENDED for Intermediate devices to remove "Log-Me" header if the next hop is not TLS. Alternatively it MAY modify "Log-Me" header local log type.

10. IANA Considerations

There is no IANA consideration for this draft.
11. Normative References


Authors’ Addresses

Adarsh Kaithal
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka  560103
India

Email: akaithal@cisco.com

Sandeep K S
Cisco Systems, Inc.
Cessna Business Park,
Kadabeesanahalli Village, Varthur Hobli,
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka  560103
India

Email: sandks@cisco.com