Session Initiation Protocol (SIP) Header Parameter for Debugging
draft-dawes-sipping-debug-03

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.

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 February 7, 2011.

Copyright Notice

Copyright (c) 2010 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 (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Requirements Language ......................................... 3
3. Motivating Scenario ............................................ 3
4. Signalling for Example Scenario ............................... 4
   4.1. General .................................................. 4
   4.2. Originating Session ..................................... 4
   4.3. Terminating Sessions ..................................... 9
5. Avoiding Configuring all Entities on the Signalling Path ....... 9
   5.1. General .................................................. 9
   5.2. Originating Sessions ..................................... 10
   5.3. Terminating Sessions ..................................... 10
6. Multiple Simultaneous Events .................................. 10
7. debug Parameter in SIP Requests ................................ 12
   7.1. Forked Requests .......................................... 12
   7.2. Back-to-Back User Agents ................................ 12
8. debug Parameter in SIP Responses ................................ 12
9. Multiple Service Providers .................................... 12
   9.1. General .................................................. 12
10. Configuration for Multiple AORs ............................... 13
11. Retrieving Debugging Logs ..................................... 13
12. Security Considerations ....................................... 13
   12.1. Trust Domain ........................................... 13
   12.2. Security Threats ....................................... 13
   12.3. Security Mechanisms .................................... 14
13. Formal Syntax .................................................. 14
   13.1. debug Header Field Parameter Syntax ................... 14
14. XML Schema for Debug Configuration ............................ 14
15. References ..................................................... 17
   15.1. Normative References ..................................... 17
   15.2. Informative References ................................... 18
Appendix A. Additional Stuff ...................................... 18
Author’s Address .................................................... 18
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 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, the first proxy the terminal connects to, and 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. 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 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.

4. Signalling for Example Scenario

4.1. 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.

4.2. Originating Session

The first step is to provide Alice’s SIP client with 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 only be subscribed to 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.
Figure 1: Prompting Client to Retrieve Debugging Configuration

The XML document returned to Alice’s terminal contains the debugging configuration shown below. 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.

```
<session id="u01">
  <start-trigger>
    <to>bob@barcorp.com</to>
    <debug-id>1A346D</debug-id>
  </start-trigger>
  <stop-trigger>
    <time-period>T0H2M0S</time-period>
  </stop-trigger>
</session>
```

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.

```
<session id="u01">
  <start-trigger>
    <to>bob@barcorp.com</to>
    <debug-id>1A346D</debug-id>
  </start-trigger>
  <stop-trigger>
    <time-period>T0H2M0S</time-period>
  </stop-trigger>
</session>
```

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
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.
Figure 4: Example of Debugging
4.3. 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.

```xml
<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.

5. Avoiding Configuring all Entities on the Signalling Path

5.1. 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.

5.2. 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.

5.3. 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.

6. 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. 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.
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
   Session-ID:
   ;debug="1A346E"

<------------------------
(4) 501 Not Implemented
   Session-ID:
   debug="1A346E"
   Dialog has ended, so
   proxy stops
   logging

<------------------------
   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.

7. debug Parameter in SIP Requests

7.1. 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.

7.2. 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.

8. 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.

9. Multiple Service Providers

9.1. 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).
10. 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.

11. 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.

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.

12. Security Considerations

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

12.1. 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.

12.2. 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.

12.3. Security Mechanisms

Security considerations are very similar to those in
draft-ietf-sipping-config-framework
[I-D.ietf-sipping-config-framework], so the same mechanisms can be
used to secure debugging configuration and logged signalling.

13. 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.

13.1. debug Header Field Parameter Syntax

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

    debug = "debug" EQUAL 6*6HEXDIG

14. XML Schema for Debug Configuration

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

<?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. -->

  <!-- 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:simpleType>
  </xs:attribute>
<xs:restriction base="xs:string">
  <xs:enumeration value="full"/>
  <xs:enumeration value="partial"/>
</xs:restriction>
</xs:simpleType>
</xs:attribute>

<!-- 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>

<xs:element name="start-trigger">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="from" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="to" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="icsi" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="iari" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="stop-trigger">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="time" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="time-period" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="reason" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="control">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="interface"/>
      <xs:element ref="depth" minOccurs="0" maxOccurs="1"/>
      <xs:element ref="debug-id" minOccurs="0" maxOccurs="1"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>

Figure 8: XML schema for debugging configuration

15. References

15.1. Normative References

[I-D.kaplan-dispatch-session-id]
Kaplan, H., "A Session Identifier for the Session Initiation Protocol (SIP)",
draft-kaplan-dispatch-session-id-02 (work in progress),
July 2010.

15.2. Informative References

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


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