SIPCORE Working Group I. Baz Castillo
Internet-Draft J. Millan Villegas
Intended status: Standards Track Versatica
Expires: June 2, 2014 V. Pascual
Quobis
November 29, 2013

The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)
draft-ietf-sipcore-sip-websocket-10

Abstract

The WebSocket protocol enables two-way realtime communication between clients and servers in web-based applications. This document specifies a WebSocket sub-protocol as a reliable transport mechanism between SIP (Session Initiation Protocol) entities to enable usage of SIP in web-oriented deployments.

Table of Contents

1. Introduction

The WebSocket [RFC6455] protocol enables message exchange between clients and servers on top of a persistent TCP connection (optionally secured with TLS [RFC5246]). The initial protocol handshake makes use of HTTP [RFC2616] semantics, allowing the WebSocket protocol to reuse existing HTTP infrastructure.

Modern web browsers include a WebSocket client stack complying with the WebSocket API [WS-API] as specified by the W3C. It is expected that other client applications (those running in personal computers and devices such as smartphones) will also make a WebSocket client stack available. The specification in this document enables usage of SIP in these scenarios.

This specification defines a WebSocket sub-protocol (as defined in section 1.9 in [RFC6455]) for transporting SIP messages between a WebSocket client and server, a reliable and message-boundary preserving transport for SIP, DNS NAPTR [RFC3403] service values and procedures for SIP entities implementing the WebSocket transport. Media transport is out of the scope of this document.

Section 3 in this specification relaxes the requirement in [RFC3261] by which the SIP server transport MUST add a "received" parameter in the top Via header in certain circumstances.

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].

2.1. Definitions

SIP WebSocket Client: A SIP entity capable of opening outbound connections to WebSocket servers and communicating using the WebSocket SIP sub-protocol as defined by this document.

SIP WebSocket Server: A SIP entity capable of listening for inbound connections from WebSocket clients and communicating using the WebSocket SIP sub-protocol as defined by this document.

3. The WebSocket Protocol

The WebSocket protocol [RFC6455] is a transport layer on top of TCP (optionally secured with TLS [RFC5246]) in which both client and

server exchange message units in both directions. The protocol defines a connection handshake, WebSocket sub-protocol and extensions negotiation, a frame format for sending application and control data, a masking mechanism, and status codes for indicating disconnection causes.

The WebSocket connection handshake is based on HTTP [RFC2616] and utilizes the HTTP GET method with an "Upgrade" request. This is sent by the client and then answered by the server (if the negotiation succeeded) with an HTTP 101 status code. Once the handshake is completed the connection upgrades from HTTP to the WebSocket protocol. This handshake procedure is designed to reuse the existing HTTP infrastructure. During the connection handshake, client and server agree on the application protocol to use on top of the WebSocket transport. Such application protocol (also known as a "WebSocket sub-protocol") defines the format and semantics of the messages exchanged by the endpoints. This could be a custom protocol or a standardized one (as the WebSocket SIP sub-protocol defined in this document). Once the HTTP 101 response is processed both client and server reuse the underlying TCP connection for sending WebSocket messages and control frames to each other. Unlike plain HTTP, this connection is persistent and can be used for multiple message exchanges.

WebSocket defines message units to be used by applications for the exchange of data, so it provides a message boundary-preserving transport layer. These message units can contain either UTF-8 text or binary data, and can be split into multiple WebSocket text/binary transport frames as needed by the WebSocket stack.

> The WebSocket API [WS-API] for web browsers only defines callbacks to be invoked upon receipt of an entire message unit, regardless of whether it was received in a single Websocket frame or split across multiple frames.

4. The WebSocket SIP Sub-Protocol

The term WebSocket sub-protocol refers to an application-level protocol layered on top of a WebSocket connection. This document specifies the WebSocket SIP sub-protocol for carrying SIP requests and responses through a WebSocket connection.

4.1. Handshake

The SIP WebSocket Client and SIP WebSocket Server negotiate usage of the WebSocket SIP sub-protocol during the WebSocket handshake procedure as defined in section 1.3 of [RFC6455]. The Client MUST

include the value "sip" in the Sec-WebSocket-Protocol header in its handshake request. The 101 reply from the Server MUST contain "sip" in its corresponding Sec-WebSocket-Protocol header.

The WebSocket Client initiates a WebSocket connection when attempting to send a SIP request (unless there is an already established WebSocket connection for sending the SIP request). In case there is no HTTP 101 response during the WebSocket handshake it is considered a transaction error as per [RFC3261] section 8.1.3.1 "Transaction Layer Errors".

Below is an example of a WebSocket handshake in which the Client requests the WebSocket SIP sub-protocol support from the Server:

```
GET / HTTP/1.1
Host: sip-ws.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: http://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13
```

The handshake response from the Server accepting the WebSocket SIP sub-protocol would look as follows:

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip
```

Once the negotiation has been completed, the WebSocket connection is established and can be used for the transport of SIP requests and responses. Messages other than SIP requests and responses MUST NOT be transmitted over this connection.

4.2. SIP Encoding

WebSocket messages can be transported in either UTF-8 text frames or binary frames. SIP [RFC3261] allows both text and binary bodies in SIP requests and responses. Therefore SIP WebSocket Clients and SIP WebSocket Servers MUST accept both text and binary frames.

If there is at least one non-UTF-8 symbol in the whole SIP message (including headers and body) then the whole message MUST be sent within a WebSocket binary message. Given the nature of JavaScript and the WebSocket API it is RECOMMENDED to use UTF-8 encoding (or

ASCII which is a subset of UTF-8) for SIP messages carried over a WebSocket connection.

5. SIP WebSocket Transport

WebSocket [RFC6455] is a reliable protocol and therefore the SIP WebSocket sub-protocol defined by this document is a reliable SIP transport. Thus, client and server transactions using WebSocket for transport MUST follow the procedures and timer values for reliable transports as defined in [RFC3261].

Each SIP message MUST be carried within a single WebSocket message, and a WebSocket message MUST NOT contain more than one SIP message. Because the WebSocket transport preserves message boundaries, the use of the Content-Length header in SIP messages is not necessary when they are transported using the WebSocket sub-protocol.

This simplifies parsing of SIP messages for both clients and servers. There is no need to establish message boundaries using Content-Length headers between messages. Other SIP transports, such as UDP and SCTP [RFC4168] also provide this benefit.

5.1. Via Transport Parameter

Via header fields in SIP messages carry a transport protocol identifier. This document defines the value "WS" to be used for requests over plain WebSocket connections and "WSS" for requests over secure WebSocket connections (in which the WebSocket connection is established using TLS [RFC5246] with TCP transport).

The updated augmented BNF (Backus-Naur Form) [RFC5234] for this parameter is the following (the original BNF for this parameter can be found in [RFC3261], which was then updated by [RFC4168]):

```
transport  =/  "WS" / "WSS"
```

5.2. SIP URI Transport Parameter

This document defines the value "ws" as the transport parameter value for a SIP URI [RFC3986] to be contacted using the SIP WebSocket sub-protocol as transport.

The updated augmented BNF (Backus-Naur Form) for this parameter is the following (the original BNF for this parameter can be found in [RFC3261]):

```
transport-param  =/  "transport=" "ws"
```

5.3. Via received Parameter

[RFC3261] section 18.2.1 "Receiving Requests" states the following:

> When the server transport receives a request over any transport, it MUST examine the value of the "sent-by" parameter in the top Via header field value. If the host portion of the "sent-by" field contains a domain name, or if it contains an IP address that differs from the packet source address, the server MUST add a "received" parameter to that Via header field value. This parameter MUST contain the source address from which the packet was received.

The requirement of adding the "received" parameter does not fit well into the WebSocket protocol design. The WebSocket connection handshake reuses existing HTTP infrastructure in which there could be an unknown number of HTTP proxies and/or TCP load balancers between the SIP WebSocket Client and Server, so the source address the server would write into the Via "received" parameter would be the address of the HTTP/TCP intermediary in front of it. This could reveal sensitive information about the internal topology of the Server's network to the Client.

Given the fact that SIP responses can only be sent over the existing WebSocket connection, the Via "received" parameter is of little use. Therefore, in order to allow hiding possible sensitive information about the SIP WebSocket Server's network, this document updates [RFC3261] section 18.2.1 by stating:

> When a SIP WebSocket Server receives a request it MAY decide not to add a "received" parameter to the top Via header. Therefore SIP WebSocket Clients MUST accept responses without such a parameter in the top Via header regardless of whether the Via "sent-by" field contains a domain name.

5.4. SIP Transport Implementation Requirements

[RFC3261] section 18 "Transport" states the following:

> All SIP elements MUST implement UDP and TCP. SIP elements MAY implement other protocols.

The specification of this transport enables SIP to be used as a session establishment protocol in scenarios where none of other transport protocols defined for SIP can be used. Since some environments do not enable SIP elements to use UDP and TCP as SIP transport protocols, a SIP element acting as a SIP WebSocket Client is not mandated to implement support of UDP and TCP.

5.5. Locating a SIP Server

[RFC3263] specifies the procedures which should be followed by SIP entities for locating SIP servers. This specification defines the NAPTR service value "SIP+D2W" for SIP WebSocket Servers that support plain WebSocket connections and "SIPS+D2W" for SIP WebSocket Servers that support secure WebSocket connections.

> At the time this document was written, DNS NAPTR/SRV queries could not be performed by commonly available WebSocket client stacks (in JavaScript engines and web browsers).

In the absence of DNS SRV resource records or an explicit port, the default port for a SIP URI using the "sip" scheme and the "ws" transport parameter is 80, and the default port for a SIP URI using the "sips" scheme and the "ws" transport parameter is 443.

6. Connection Keep-Alive

SIP WebSocket Clients and Servers may keep their WebSocket connections open by sending periodic WebSocket "Ping" frames as described in [RFC6455] section 5.5.2.

> The WebSocket API [WS-API] does not provide a mechanism for applications running in a web browser to control whether or not periodic WebSocket "Ping" frames are sent to the server. The implementation of such a keep-alive feature is the decision of each web browser manufacturer and may also depend on the configuration of the web browser.

The indication and use of the CRLF NAT keep-alive mechanism defined for SIP connection-oriented transports in [RFC5626] section 3.5.1 or [RFC6223] are, of course, usable over the transport defined in this specification.

7. Authentication

This section describes how authentication is achieved through the requirements in [RFC6455], [RFC6265], [RFC2617] and [RFC3261].

WebSocket protocol [RFC6455] does not define an authentication mechanism, instead it exposes the following text in section 10.5 "WebSocket Client Authentication":

> This protocol doesn't prescribe any particular way that servers can authenticate clients during the WebSocket handshake. The

WebSocket server can use any client authentication mechanism available to a generic HTTP server, such as cookies, HTTP authentication, or TLS authentication.

The following list exposes mandatory to implement and optional mechanisms for SIP WebSocket Clients and Servers in order to get interoperability at WebSocket authentication level:

- A SIP WebSocket Client MUST be ready to add a session Cookie when it runs in a web browser (or behaves like a browser navigating a website) and has previously retrieved a session Cookie from the web server whose URL domain matches the domain in the WebSocket URI. This mechanism is defined by [RFC6265].

- A SIP WebSocket Client MUST be ready to be challenged with HTTP 401 status code by the SIP WebSocket Server when performing the WebSocket handshake as stated in [RFC2617].

- A SIP WebSocket Client MAY use TLS client authentication (when in a secure WebSocket connection) as an optional authentication mechanism.

  Note however that TLS client authentication in WebSocket protocol is governed by the rules of HTTP protocol rather than the rules of SIP protocol.

- A SIP WebSocket Server MUST be ready to read session Cookies when present in the WebSocket handshake request, and use such a Cookie value for determining whether the WebSocket connection has been initiated by a HTTP client navigating a website in the same domain (or subdomain) as the SIP WebSocket Server.

- A SIP WebSocket Server SHOULD be able to reject a WebSocket handshake request with HTTP 401 status code by providing a Basic/Digest challenge as defined for HTTP protocol.

Regardless of whether the SIP WebSocket Server requires authentication during the WebSocket handshake or not, authentication MAY be requested at SIP protocol level.

Some authentication use cases are exposed in Appendix A.

8. Examples

8.1. Registration

```
Alice    (SIP WSS)    proxy.example.com
|                             |
|HTTP GET (WS handshake) F1   |
|---------------------------->|
|101 Switching Protocols F2   |
|<----------------------------|
|                             |
|REGISTER F3                  |
|---------------------------->|
|200 OK F4                    |
|<----------------------------|
|                             |
```

Alice loads a web page using her web browser and retrieves JavaScript code implementing the WebSocket SIP sub-protocol defined in this document. The JavaScript code (a SIP WebSocket Client) establishes a secure WebSocket connection with a SIP proxy/registrar (a SIP WebSocket Server) at proxy.example.com. Upon WebSocket connection, Alice constructs and sends a SIP REGISTER request including Outbound and GRUU support. Since the JavaScript stack in a browser has no way to determine the local address from which the WebSocket connection was made, this implementation uses a random ".invalid" domain name for the Via header sent-by parameter and for the hostport of the URI in the Contact header (see Appendix B.1).

Message details (authentication and SDP bodies are omitted for simplicity):

```
F1 HTTP GET (WS handshake)  Alice -> proxy.example.com (TLS)

GET / HTTP/1.1
Host: proxy.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Origin: https://www.example.com
Sec-WebSocket-Protocol: sip
Sec-WebSocket-Version: 13


F2 101 Switching Protocols  proxy.example.com -> Alice (TLS)

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
```

```
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: sip
```

F3 REGISTER Alice -> proxy.example.com (transport WSS)

```
REGISTER sip:proxy.example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Max-Forwards: 70
Supported: path, outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
  ;reg-id=1
  ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
```

F4 200 OK proxy.example.com -> Alice (transport WSS)

```
SIP/2.0 200 OK
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKasudf
From: sip:alice@example.com;tag=65bnmj.34asd
To: sip:alice@example.com;tag=12isjljn8
Call-ID: aiuy7k9njasd
CSeq: 1 REGISTER
Supported: outbound, gruu
Contact: <sip:alice@df7jal23ls0d.invalid;transport=ws>
  ;reg-id=1
  ;+sip.instance="<urn:uuid:f81-7dec-14a06cf1>"
  ;pub-gruu="sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1"
  ;temp-gruu="sip:87ash54=3dd.98a@example.com;gr"
  ;expires=3600
```

8.2. INVITE Dialog through a Proxy

```
   Alice    (SIP WSS)    proxy.example.com    (SIP UDP)       Bob
   |                             |                             |
   |INVITE F1                    |                             |
   |---------------------------->|                             |
   |100 Trying F2                |                             |
   |<----------------------------|                             |
   |                             |INVITE F3                    |
   |                             |---------------------------->|
   |                             |200 OK F4                    |
   |                             |<----------------------------|
   |200 OK F5                    |                             |
   |<----------------------------|                             |
   |                             |                             |
   |ACK F6                       |                             |
   |---------------------------->|                             |
   |                             |ACK F7                       |
   |                             |---------------------------->|
   |                             |                             |
   |                 Bidirectional RTP Media                   |
   |<=========================================================>|
   |                             |                             |
   |                             |BYE F8                       |
   |                             |<----------------------------|
   |BYE F9                       |                             |
   |<----------------------------|                             |
   |200 OK F10                   |                             |
   |---------------------------->|                             |
   |                             |200 OK F11                   |
   |                             |---------------------------->|
   |                             |                             |
```

In the same scenario Alice places a call to Bob's AoR (Address Of Record). The SIP WebSocket Server at proxy.example.com acts as a SIP proxy, routing the INVITE to Bob's contact address (which happens to be using SIP transported over UDP). Bob answers the call and then terminates it.

Message details (authentication and SDP bodies are omitted for simplicity):

F1 INVITE Alice -> proxy.example.com (transport WSS)

```
INVITE sip:bob@example.com SIP/2.0
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
```

```
CSeq: 1 INVITE
Max-Forwards: 70
Supported: path, outbound, gruu
Route: <sip:proxy.example.com:443;transport=ws;lr>
Contact: <sip:alice@example.com
 ;gr=urn:uuid:f81-7dec-14a06cf1;ob>
Content-Type: application/sdp
```

F2 100 Trying proxy.example.com -> Alice (transport WSS)

```
SIP/2.0 100 Trying
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
```

F3 INVITE proxy.example.com -> Bob (transport UDP)

```
INVITE sip:bob@203.0.113.22:5060 SIP/2.0
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKhjhjqw32c
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com
Call-ID: asidkj3ss
CSeq: 1 INVITE
Max-Forwards: 69
Supported: path, outbound, gruu
Contact: <sip:alice@example.com
  ;gr=urn:uuid:f81-7dec-14a06cf1;ob>
Content-Type: application/sdp
```

F4 200 OK Bob -> proxy.example.com (transport UDP)

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKhjhjqw32c
  ;received=192.0.2.10
Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
Record-Route: <sip:proxy.example.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:alice@example.com;tag=asdyka899
To: sip:bob@example.com;tag=bmqkjhsd
Call-ID: asidkj3ss
```

```
   CSeq: 1 INVITE
   Contact: <sip:bob@203.0.113.22:5060;transport=udp>
   Content-Type: application/sdp


   F5 200 OK  proxy.example.com -> Alice (transport WSS)

   SIP/2.0 200 OK
   Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bK56sdasks
   Record-Route: <sip:proxy.example.com;transport=udp;lr>,
     <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
   From: sip:alice@example.com;tag=asdyka899
   To: sip:bob@example.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 INVITE
   Contact: <sip:bob@203.0.113.22:5060;transport=udp>
   Content-Type: application/sdp


   F6 ACK  Alice -> proxy.example.com (transport WSS)

   ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
   Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKhgqqp090
   Route: <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>,
     <sip:proxy.example.com;transport=udp;lr>,
   From: sip:alice@example.com;tag=asdyka899
   To: sip:bob@example.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 ACK
   Max-Forwards: 70


   F7 ACK  proxy.example.com -> Bob (transport UDP)

   ACK sip:bob@203.0.113.22:5060;transport=udp SIP/2.0
   Via: SIP/2.0/UDP proxy.example.com;branch=z9hG4bKhwpoc80zzx
   Via: SIP/2.0/WSS df7jal23ls0d.invalid;branch=z9hG4bKhgqqp090
   From: sip:alice@example.com;tag=asdyka899
   To: sip:bob@example.com;tag=bmqkjhsd
   Call-ID: asidkj3ss
   CSeq: 1 ACK
   Max-Forwards: 69


   F8 BYE  Bob -> proxy.example.com (transport UDP)

   BYE sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1;ob SIP/2.0
   Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
```

```
Route: <sip:proxy.example.com;transport=udp;lr>,
  <sip:h7kjh12s@proxy.example.com:443;transport=ws;lr>
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 70
```

F9 BYE proxy.example.com -> Alice (transport WSS)

```
BYE sip:alice@example.com;gr=urn:uuid:f81-7dec-14a06cf1;ob SIP/2.0
Via: SIP/2.0/WSS proxy.example.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
Max-Forwards: 69
```

F10 200 OK Alice -> proxy.example.com (transport WSS)

```
SIP/2.0 200 OK
Via: SIP/2.0/WSS proxy.example.com:443;branch=z9hG4bKmma01m3r5
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
```

F11 200 OK proxy.example.com -> Bob (transport UDP)

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 203.0.113.22;branch=z9hG4bKbiuiansd001
From: sip:bob@example.com;tag=bmqkjhsd
To: sip:alice@example.com;tag=asdyka899
Call-ID: asidkj3ss
CSeq: 1201 BYE
```

9. Security Considerations

9.1. Secure WebSocket Connection

It is RECOMMENDED that the SIP traffic transported over a WebSocket communication be protected by using a secure WebSocket connection

(using TLS [RFC5246] over TCP).

When establishing a connection using SIP over secure WebSocket transport, the client MUST authenticate the server using the server's certificate according to the WebSocket validation procedure in [RFC6455].

> Server operators should note that this authentication procedure is different from the procedure for SIP Domain Certificates defined in [RFC5922]. Certificates that are appropriate for SIP over TLS over TCP will probably not be appropriate for SIP over secure WebSocket connections.

9.2. Usage of SIPS Scheme

The SIPS scheme in a SIP URI dictates that the entire request path to the target be secure. If such a path includes a WebSocket connection it MUST be a secure WebSocket connection.

10. IANA Considerations

RFC Editor Note: Please set the RFC number assigned for this document in the sub-sections below and remove this note.

10.1. Registration of the WebSocket SIP Sub-Protocol

This specification requests IANA to register the WebSocket SIP sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: sip

Subprotocol Common Name: WebSocket Transport for SIP (Session Initiation Protocol)

Subprotocol Definition: TBD: this document

10.2. Registration of new NAPTR Service Field Values

This document defines two new NAPTR service field values (SIP+D2W and SIPS+D2W) and requests IANA to register these values under the "Registry for the Session Initiation Protocol (SIP) NAPTR Resource Record Services Field". The resulting entries are as follows:

Services Field	Protocol	Reference
SIP+D2W	WS	TBD: this document

SIPS+D2W	WS	TBD: this document

10.3. SIP/SIPS URI Parameters Sub-Registry

This specification requests IANA to add a reference to this document under the "SIP/SIPS URI Parameters" Sub-Registry within the "Session Initiation Protocol (SIP) Parameters" Registry:

Parameter Name	Predefined Values	Reference
transport	Yes	[RFC3261][TBD: this document]

10.4. Header Fields Sub-Registry

This specification requests IANA to add a reference to this document under the "Header Fields" Sub-Registry within the "Session Initiation Protocol (SIP) Parameters" Registry:

Header Name	compact	Reference
Via	v	[RFC3261][TBD: this document]

10.5. Header Field Parameters and Parameter Values Sub-Registry

This specification requests IANA to add a reference to this document under the "Header Field Parameters and Parameter Values" Sub-Registry within the "Session Initiation Protocol (SIP) Parameters" Registry:

Header Field	Parameter Name	Predefined Values	Reference
Via	received	No	[RFC3261][TBD: this document]

10.6. SIP Transport Sub-Registry

This document adds a new registry, "SIP Transport", to the "Session Initiation Protocol (SIP) Parameters" Registry. Its format and initial values are as shown in the following table:

Transport	Reference
UDP	[RFC 3261]
TCP	[RFC 3261]
TLS	[RFC 3261]
SCTP	[RFC 3261], [RFC 4168]
TLS-SCTP	[RFC 4168]
WS	[TBD: this document]
WSS	[TBD: this document]

The policy for registration of values in this registry is "Standards Action", as that term is defined by [RFC5226].

11. Acknowledgements

Special thanks to the following people who participated in discussions on the SIPCORE and RTCWEB WG mailing lists and contributed ideas and/or provided detailed reviews (the list is likely to be incomplete): Hadriel Kaplan, Paul Kyzivat, Robert Sparks, Adam Roach, Ranjit Avasarala, Xavier Marjou, Nataraju A. B., Martin Vopatek, Alexey Melnikov, Alan Johnston, Christer Holmberg, Salvatore Loreto, Kevin P. Fleming, Suresh Krishnan, Yaron Sheffer, Richard Barnes, Barry Leiba, Stephen Farrell, Ted Lemon, Benoit Claise, Pete Resnick, Binod, Saul Ibarra Corretge.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263,

June 2002.

[RFC3403] Mealling, M., "Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database", RFC 3403, October 2002.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

12.2. Informative References

[RFC2606] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", BCP 32, RFC 2606, June 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3327] Willis, D. and B. Hoeneisen, "Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts", RFC 3327, December 2002.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4168] Rosenberg, J., Schulzrinne, H., and G. Camarillo, "The Stream Control Transmission Protocol (SCTP) as a Transport for the Session Initiation Protocol (SIP)", RFC 4168, October 2005.

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

[RFC5627] Rosenberg, J., "Obtaining and Using Globally Routable User Agent URIs (GRUUs) in the Session Initiation Protocol (SIP)", RFC 5627, October 2009.

[RFC5922] Gurbani, V., Lawrence, S., and A. Jeffrey, "Domain Certificates in the Session Initiation Protocol (SIP)", RFC 5922, June 2010.

[RFC6223] Holmberg, C., "Indication of Support for Keep-Alive", RFC 6223, April 2011.

[WS-API] W3C and I. Hickson, Ed., "The WebSocket API", April 2013.

Appendix A. Authentication Use Cases

Sections below briefly describe some SIP over WebSocket scenarios in which authentication take place in different ways.

A.1. Just SIP Authentication

SIP PBX model A implements the SIP WebSocket transport defined by this specification. Its implementation is 100% website agnostic as it does not share information with the web server providing the HTML code to browsers, meaning that the SIP WebSocket Server (here the PBX model A) has no knowledge about web login activity within the website.

In this simple scenario, the SIP WebSocket Server does not inspect fields in the WebSocket handshake HTTP GET request such as the request URL, the Origin header value, the Host header value or the Cookie header value (if present). However some of those fields could be inspected for a minimal validation (i.e. PBX model A could require that the Origin header value contains a specific URL so just users navigating such a website would be able to establish a WebSocket connection with PBX model A).

Once the WebSocket connection has been established, SIP authentication is requested by PBX model A for each SIP request coming over that connection. Therefore SIP WebSocket Clients must be provisioned with their corresponding SIP password.

A.2. Just Web Authentication

A SIP-to-PSTN provider offers telephony service for clients logged into its website. The provider does not want to expose SIP passwords into the web for security/privacy reasons.

Once the user is logged into the web, the web server provides him with a SIP identity (SIP URI) and a session temporary token string (along with the SIP WebSocket Client JavaScript application and SIP settings). The web server stores the SIP identity and session token into a database.

The web application adds the SIP identity and session token as URL query parameters in the WebSocket handshake request and attempts the connection. The SIP WebSocket Server inspects the handshake request and validates that the session token matches the value stored in the database for the given SIP identity. In case the value matches, the WebSocket connection gets "authenticated" for that SIP identity. The SIP WebSocket Client can then register and make calls. The SIP WebSocket Server would however verify that the identity in those SIP requests (i.e. the From URI value) matches the SIP identity the WebSocket connection is associated to (otherwise the SIP request is rejected).

When the user performs logout action in the web, the web server removes the SIP identity and session token tuple from the database and notifies it to the SIP WebSocket Server which revokes and closes the WebSocket connection.

No SIP authentication takes place in this scenario.

A.3. Cookie Based Authentication

Apache web server comes with a new module mod_sip_websocket. The web server is configured to listen in port 80 for both HTTP common requests and WebSocket handshake requests. Therefore both the web server and the SIP WebSocket Server are co-located within the same host and same domain.

Once the user is logged into the web, he is provided with the SIP WebSocket Client JavaScript application and SIP settings. The HTTP 200 response after the login procedure also contains a session Cookie [RFC6265]. The web application attempts then a WebSocket connection against the same URL/domain of the website and thus, the session Cookie is automatically added by the browser into the WebSocket handshake request (as the WebSocket protocol [RFC6455] states).

The web server inspects the Cookie value (as it would do for a common HTTP request containing a session Cookie, so login procedure is not required again). If the Cookie is valid the WebSocket connection is authorized and, as in the previous use case, the connection is also associated with a specific SIP identity which must be satisfied by every SIP request coming over that connection.

No SIP authentication takes place in this scenario but just common Cookie usage as widely deployed in the WWW.

Appendix B. Implementation Guidelines

Let us assume a scenario in which the users access with their web browsers (probably behind NAT) an application provided by a server on an intranet, login by entering their user identifier and credentials, and retrieve a JavaScript application (along with the HTML) implementing a SIP WebSocket Client.

Such a SIP stack connects to a given SIP WebSocket Server (an outbound SIP proxy which also implements classic SIP transports such as UDP and TCP). The HTTP GET method request sent by the web browser for the WebSocket handshake includes a Cookie [RFC6265] header with the value previously provided by the server after the successful login procedure. The Cookie value is then inspected by the WebSocket server to authorize the connection. Once the WebSocket connection is established, the SIP WebSocket Client performs a SIP registration to a SIP registrar server that is reachable through the proxy. After registration, the SIP WebSocket Client and Server exchange SIP messages as would normally be expected.

This scenario is quite similar to ones in which SIP UAs behind NATs connect to a proxy and must reuse the same TCP connection for incoming requests (because they are not directly reachable by the proxy otherwise). In both cases, the SIP UAs are only reachable through the proxy they are connected to.

The SIP Outbound extension [RFC5626] seems an appropriate solution for this scenario. Therefore these SIP WebSocket Clients and the SIP registrar implement both the Outbound and Path [RFC3327] extensions, and the SIP proxy acts as an Outbound Edge Proxy (as defined in [RFC5626] section 3.4).

SIP WebSocket Clients in this scenario receive incoming SIP requests via the SIP WebSocket Server they are connected to. Therefore, in some call transfer cases the usage of GRUU [RFC5627] (which should be implemented in both the SIP WebSocket Clients and SIP registrar) is valuable.

> If a REFER request is sent to a third SIP user agent including the Contact URI of a SIP WebSocket Client as the target in its Refer-To header field, such a URI will be reachable by the third SIP UA only if it is a globally routable URI. GRUU (Globally Routable User Agent URI) is a solution for those scenarios, and would cause the incoming request from the third SIP user agent to

be sent to the SIP registrar, which would route the request to the SIP WebSocket Client via the Outbound Edge Proxy.

B.1. SIP WebSocket Client Considerations

The JavaScript stack in web browsers does not have the ability to discover the local transport address used for originating WebSocket connections. A SIP WebSocket client running in such an environment can construct a domain name consisting of a random token followed by the ".invalid" top-level domain name, as stated in [RFC2606], and uses it within its Via and Contact headers.

The Contact URI provided by SIP UAs requesting (and receiving) Outbound support is not used for routing requests to those UAs, thus it is safe to set a random domain in the Contact URI hostport.

Both the Outbound and GRUU specifications require a SIP UA to include a Uniform Resource Name (URN) in a "+sip.instance" parameter of the Contact header they include their SIP REGISTER requests. The client device is responsible for generating or collecting a suitable value for this purpose.

In web browsers it is difficult to generate or collect a suitable value to be used as a URN value from the browser itself. This scenario suggests that value is generated according to [RFC5626] section 4.1 by the web application running in the browser the first time it loads the JavaScript SIP stack code, and then it is stored as a Cookie within the browser.

B.2. SIP WebSocket Server Considerations

The SIP WebSocket Server in this scenario behaves as a SIP Outbound Edge Proxy, which involves support for Outbound [RFC5626] and Path [RFC3327].

The proxy performs Loose Routing and remains in the path of dialogs as specified in [RFC3261]. If it did not do this, in-dialog requests would fail since SIP WebSocket Clients make use of their SIP WebSocket Server in order to send and receive SIP messages.

Authors' Addresses

Inaki Baz Castillo
Versatica
Barakaldo, Basque Country
Spain

Email: ibc@aliax.net

Jose Luis Millan Villegas
Versatica
Bilbao, Basque Country
Spain

Email: jmillan@aliax.net

Victor Pascual
Quobis
Spain

Email: victor.pascual@quobis.com