Dispatch Working Group
Internet-Draft
Updates: 4975, 4976 (if approved)
Intended status: Standards Track
Expires: June 16, 2014

P. Dunkley
G. Llewellyn
Crocodile RCS Ltd
V. Pascual
A. Roman
Quobis
G. Salgueiro
Cisco

December 13, 2013

The WebSocket Protocol as a Transport for the Message Session Relay Protocol (MSRP)
draft-pd-dispatch-msrp-websocket-04

Abstract

The WebSocket protocol enables two-way real-time communication between clients and servers. This document specifies a new WebSocket sub-protocol as a reliable transport mechanism between MSRP (Message Session Relay Protocol) clients and relays to enable usage of MSRP in new scenarios. This document normatively updates RFC 4975 and RFC 4976.

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 smart-phones) will also make a WebSocket client stack available. The specification in this document enables usage of MSRP in these scenarios.

This specification defines a new WebSocket sub-protocol (as defined in section 1.9 in [RFC6455]) for transporting MSRP messages between a WebSocket client and MSRP relay [RFC4976] containing a WebSocket server, a new transport for MSRP, and procedures for MSRP clients and relays implementing the WebSocket transport.

MSRP over WebSocket is well suited for MSRP interactions between clients and servers. Common use cases for MSRP over WebSocket include:

- Human-to-machine messaging
- Client-to-server data exchange (for example, application control signalling)
- Human-to-human messaging where local policy requires authentication and/or logging

2. Terminology

All diagrams, examples, and notes in this specification are non-normative, as are all sections explicitly marked non-normative. Everything else in this specification is normative.

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

MSRP WebSocket Client: An MSRP entity capable of opening outbound connections to MSRP relays which are WebSocket servers and communicating using the WebSocket MSRP sub-protocol as defined by this document.

MSRP WebSocket Server: An MSRP entity (specifically an MSRP relay [RFC4976]) capable of listening for inbound connections from WebSocket clients and communicating using the WebSocket MSRP sub-protocol as defined by this document.

3. The WebSocket Protocol

This section is non-normative.

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 (such as the WebSocket MSRP 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 MSRP 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 MSRP sub-protocol for carrying MSRP requests and responses through a WebSocket connection.

4.1. Handshake

The MSRP WebSocket Client and MSRP WebSocket Server negotiate usage of the WebSocket MSRP sub-protocol during the WebSocket handshake procedure as defined in section 1.3 of [RFC6455]. The Client MUST include the value "msrp" in the Sec-WebSocket-Protocol header in its handshake request. The 101 reply from the Server MUST contain "msrp" in its corresponding Sec-WebSocket-Protocol header.

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

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

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

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

Once the negotiation has been completed, the WebSocket connection is established and can be used for the transport of MSRP requests and responses. The WebSocket messages transmitted over this connection MUST conform to the negotiated WebSocket sub-protocol.

4.2. MSRP Encoding

WebSocket messages can be transported in either UTF-8 text frames or binary frames. MSRP [RFC4975] allows both text and binary bodies in MSRP requests. Therefore MSRP WebSocket Clients and Servers MUST accept both text and binary frames.

5. MSRP WebSocket Transport

5.1. General

WebSocket clients cannot receive WebSocket connections initiated by other WebSocket clients or WebSocket servers. This means that it is impossible for an MSRP client to communicate directly with other MSRP clients. Therefore, all MSRP over WebSocket messages MUST be routed via an MSRP WebSocket Server.

MSRP WebSocket Servers can be used to route MSRP messages between MSRP WebSocket Clients, and between MSRP WebSocket Clients and "normal" MSRP clients and relays.

Each MSRP chunk MUST be carried within a single WebSocket message, and a WebSocket message MUST NOT contain more than one MSRP chunk.

> This simplifies parsing of MSRP messages for both clients and servers. When large messages are sent MSRP chunking (as defined in section 5.1 of [RFC4975]) MUST be used to split the message into several smaller MSRP chunks.

5.2. Updates to RFC 4975

5.2.1. MSRP URI Transport Parameter

This document defines the value "ws" as the transport parameter value for an MSRP URI [RFC3986] to be contacted using the MSRP WebSocket sub-protocol as 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 [RFC4975]):

```
transport  =  "tcp" / "ws" / 1*ALPHANUM
```

5.2.2. SDP Transport Protocol

This document does not define a new SDP transport protocol for MSRP over WebSockets. As all MSRP over WebSocket messages MUST be routed via an MSRP WebSocket Server, it is acceptable for an MSRP WebSocket Client to specify the "TCP/MSRP" or "TCP/TLS/MSRP" protocols in the SDP m-line - that being the protocol used by non-WebSocket clients and between MSRP relays ([RFC4975] section 8.1).

The "ws" transport parameter will appear in the endpoint URI in SDP "path" attribute ([RFC4975] section 8.2); compliant MSRP implementation should already allow unrecognised transports, provided that they do not have to establish a direct connection to the resource described by the URI.

5.3. Updates to RFC 4976

5.3.1. AUTH Request Authentication

The MSRP relay specification [RFC4976] states that AUTH requests MUST be authenticated. This document modifies this requirement to state that all connections between MSRP clients and relays MUST be authenticated. In the case of the MSRP WebSocket Clients there are two possible authentication mechanisms:

1. HTTP Digest authentication in AUTH (as per [RFC4976]).

2. Cookie-based or HTTP Digest authentication in the WebSocket Handshake (see Section 7).

5.3.2. Use of TLS

The MSRP relay specification [RFC4976] mandates the use of TLS between MSRP clients and MSRP relays, and specifies the mechanisms that must be used for TLS authentication. This document downgrades the MUSTs with respect to TLS to SHOULDs when using the MSRP WebSocket sub-protocol as transport. Connections between MSRP WebSocket Clients and Servers SHOULD use secure WebSocket connections, but MAY use insecure WebSocket connections. As the secure WebSocket connections are negotiated between the client's WebSocket stack and the WebSocket server, an MSRP WebSocket Client may have no knowledge of, or control over, the mechanisms used for TLS authentication.

6. Connection Keep-alive

This section is non-normative.

It is RECOMMENDED that MSRP WebSocket Clients and Servers 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.

A future WebSocket protocol extension providing a similar keep alive mechanism could also be used.

When MSRP WebSocket Clients or Servers cannot use WebSocket "Ping" frames to keep connections open an MSRP implementation MAY use bodiless SEND requests as described in [RFC4975] section 7.1.

7. Authentication

This section is non-normative.

Prior to sending MSRP requests, an MSRP WebSocket Client connects to an MSRP WebSocket Server and performs the connection handshake. As described in Section 3 the handshake procedure involves a HTTP GET method request from the Client and a response from the Server including an HTTP 101 status code.

In order to authorize the WebSocket connection, the MSRP WebSocket Server MAY inspect any HTTP headers present (for example, Cookie [RFC6265], Host [RFC2616], or Origin [RFC6454]) in the HTTP GET request. For many web applications the value of such a Cookie is provided by the web server once the user has authenticated themselves to the web server, which could be done by many existing mechanisms. As an alternative method, the MSRP WebSocket Server could request HTTP authentication by replying to the Client's GET method request with a HTTP 401 status code. The WebSocket protocol [RFC6455] covers this usage in section 4.1:

> If the status code received from the server is not 101, the WebSocket client stack handles the response per HTTP [RFC2616] procedures, in particular the client might perform authentication if it receives 401 status code.

If the HTTP GET request contains an Origin header the MSRP WebSocket Server SHOULD indicate Cross-Origin Resource Sharing [CORS] by adding an Access-Control-Allow-Origin header to the 101 response.

Regardless of whether the MSRP WebSocket Server requires authentication during the WebSocket handshake, authentication MAY be requested at MSRP protocol level. Therefore an MSRP WebSocket Client SHOULD support HTTP Digest [RFC2617] authentication as stated in [RFC4976].

8. Examples

8.1. AUTH

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

Alice loads a web page using her web browser and retrieves JavaScript code implementing the WebSocket MSRP sub-protocol defined in this document. The JavaScript code (an MSRP WebSocket Client) establishes a secure WebSocket connection with an MSRP relay (an MSRP WebSocket Server) at a.example.com. Upon WebSocket connection, Alice constructs and sends an MSRP AUTH request. 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 hostpart of the From-Path URI (see Appendix A.1).

Message details (authentication is omitted for simplicity):

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

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


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

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


F3 AUTH  Alice -> a.example.com (transport WSS)

MSRP 49fi AUTH
To-Path: msrps://alice@a.example.com:443;ws
From-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
-------49fi$


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

MSRP 49fi 200 OK
To-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
From-Path: msrps://alice@a.example.com:443;ws
Use-Path: msrps://a.example.com:2855/jui787s2f;tcp
Expires: 900
-------49fi$
```

8.2. SDP Exchange (MSRP WebSocket Client to MSRP Client)

The following example shows SDP that could be included in a SIP message to set up an MSRP session between Alice and Bob where Alice uses a WebSocket MSRP relay.

Note that SDP does not permit line folding. A "\" in the examples shows a line continuation due to limitations in line length of this document. Neither the backslash nor the extra CRLF is included in the actual SDP.

Alice makes an offer with a path including the relay:

```
c=IN IP4 a.example.com
m=message 1234 TCP/TLS/MSRP *
a=accept-types:message/cpim text/plain text/html
a=path:msrps://a.example.com:2855/jui787s2f;tcp \
       msrps://df7jal23ls0d.invalid:2855/98cjs;ws
```

In this offer, Alice wishes to receive MSRP messages via the relay at a.example.com. She wants to use TLS as the transport for the MSRP session (beyond the relay). She can accept message/cpim, text/plain, and text/html message bodies in SEND requests.

Bob's answer to this offer could look like:

```
c=IN IP4 bob.example.com
m=message 1234 TCP/TLS/MSRP *
a=accept-types:message/cpim text/plain
a=path:msrps://bob.example.com:8145/foo;tcp
```

Here Bob wishes to receive the MSRP messages at bob.example.com. He can accept only message/cpim and text/plain message bodies in SEND requests and has rejected the text/html content offered by Alice. He does not need a relay to set up the MSRP session.

8.3. SEND (MSRP WebSocket Client to MSRP Client)

```
Alice    (MSRP WSS)     a.example.com      (MSRP TLS)     Bob
|                             |                             |
|SEND F1                      |                             |
|---------------------------->|                             |
|200 OK F2                    |                             |
|<----------------------------|                             |
|                             |SEND F3                      |
|                             |---------------------------->|
|                             |200 OK F4                    |
|                             |<----------------------------|
```

In the same scenario Alice sends an instant message to Bob (session details having been previously negotiated by some other mechanism - such as SDP [RFC4976]). The MSRP WebSocket Server at a.example.com acts as an MSRP relay, routing the message to Bob over TLS.

In this example Bob himself does not require an MSRP relay and has not authenticated with a.example.com but Alice does and has.

Message details (note that MSRP does not permit line folding. A "\" in the examples shows a line continuation due to limitations in line

length of this document. Neither the backslash nor the extra CRLF is included in the actual request or response):

```
F1 SEND  Alice -> a.example.com (transport WSS)

MSRP 6aef SEND
To-Path: msrps://a.example.com:2855/jui787s2f;tcp \
         msrps://bob.example.com:8145/foo;tcp
From-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
Success-Report: no
Byte-Range: 1-*/*
Message-ID: 87652
Content-Type: text/plain

Hi Bob, I'm about to send you file.mpeg
-------6aef$


F2 200 OK  a.example.com -> Alice (transport WSS)

MSRP 6aef 200 OK
To-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
From-Path: msrps://a.example.com:2855/jui787s2f;tcp
-------6aef$


F3 SEND  a.example.com -> Bob (transport TLS)

MSRP juh76 SEND
To-Path: msrps://bob.example.com:8145/foo;tcp
From-Path:  msrps://a.example.com:2855/jui787s2f;tcp \
            msrps://df7jal23ls0d.invalid:2855/98cjs;ws
Success-Report: no
Byte-Range: 1-*/*
Message-ID: 87652
Content-Type: text/plain

Hi Bob, I'm about to send you file.mpeg
-------juh76$


F4 200 OK  Bob -> a.example.com (transport TLS)

MSRP juh76 200 OK
To-Path: msrps://a.example.com:2855/jui787s2f;tcp
From-Path: msrps://bob.example.com:8145/foo;tcp
-------juh76$
```

8.4. SEND (MSRP Client to MSRP WebSocket Client)

```
Bob      (MSRP TLS)     a.example.com     (MSRP WSS)    Alice
|                             |                             |
|SEND F1                      |                             |
|---------------------------->|                             |
|200 OK F2                    |                             |
|<----------------------------|                             |
|                             |SEND F3                      |
|                             |---------------------------->|
|                             |200 OK F4                    |
|                             |<----------------------------|
```

In the same scenario Bob sends an instant message to Alice (session details having been previously negotiated by some other mechanism - such as SDP [RFC4976]). The MSRP WebSocket Server at a.example.com acts as an MSRP relay, routing the message to Alice over secure WebSocket.

In this example Bob himself does not require an MSRP relay and has not authenticated with a.example.com but Alice does and has.

Message details (note that MSRP does not permit line folding. A "\" in the examples shows a line continuation due to limitations in line length of this document. Neither the backslash nor the extra CRLF is included in the actual request or response):

```
F1 SEND  Bob -> a.example.com (transport TLS)

MSRP xght6 SEND
To-Path: msrps://a.example.com:2855/jui787s2f;tcp \
         msrps://df7jal23ls0d.invalid:2855/98cjs;ws
From-Path: msrps://bob.example.com:8145/foo;tcp
Success-Report: no
Byte-Range: 1-*/*
Message-ID: 87652
Content-Type: text/plain

Thanks for the file.
-------xght6$


F2 200 OK  a.example.com -> Bob (transport TLS)

MSRP xght6 200 OK
To-Path: msrps://bob.example.com:8145/foo;tcp
From-Path: msrps://a.example.com:2855/jui787s2f;tcp
-------xght6$


F3 SEND  a.example.com -> Alice (transport WSS)

MSRP yh67 SEND
To-Path:  msrps://df7jal23ls0d.invalid:2855/98cjs;ws
From-Path:  msrps://a.example.com:2855/jui787s2f;tcp \
            msrps://bob.example.com:8145/foo;tcp
Success-Report: no
Byte-Range: 1-*/*
Message-ID: 87652
Content-Type: text/plain

Hi Bob, I'm about to send you file.mpeg
-------yh67$


F4 200 OK  Bob -> a.example.com (transport TLS)

MSRP yh67 200 OK
To-Path:  msrps://a.example.com:2855/jui787s2f;tcp
From-Path:  msrps://df7jal23ls0d.invalid:2855/98cjs;ws
-------yh67$
```

8.5. SEND (MSRP WebSocket Client to MSRP WebSocket Client)

```
Alice    (MSRP WSS)     a.example.com     (MSRP WSS)    Carol
|                             |                             |
|SEND F1                      |                             |
|---------------------------->|                             |
|200 OK F2                    |                             |
|<----------------------------|                             |
|                             |SEND F3                      |
|                             |---------------------------->|
|                             |200 OK F4                    |
|                             |<----------------------------|
```

In the same scenario Alice sends an instant message to Carol (session details having been previously negotiated by some other mechanism - such as SDP [RFC4976]). The MSRP WebSocket Server at a.example.com acts as an MSRP relay, routing the message to Carol over secure WebSocket.

In this example both Alice and Carol are using MSRP WebSocket Clients and an MSRP WebSocket Server. This means that a.example.com will appear twice in the To-Path in F1. a.example.com can either handle this internally or loop the MSRP SEND request back to itself as if it were two, separate, MSRP relays.

Message details (note that MSRP does not permit line folding. A "\" in the examples shows a line continuation due to limitations in line length of this document. Neither the backslash nor the extra CRLF is included in the actual request or response):

```
F1 SEND  Alice -> a.example.com (transport WSS)

MSRP kjh6 SEND
To-Path: msrps://a.example.com:2855/jui787s2f;tcp \
         msrps://a.example.com:2855/iwnslt;tcp \
         msrps://jk9awp14vj8x.invalid:2855/76qwe;ws
From-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
Success-Report: no
Byte-Range: 1-*/*
Message-ID: 87652
Content-Type: text/plain

Carol, here is the file Bob sent me.
-------kjh6$


F2 200 OK  a.example.com -> Alice (transport WSS)

MSRP kjh6 200 OK
To-Path: msrps://df7jal23ls0d.invalid:2855/98cjs;ws
From-Path: msrps://a.example.com:2855/jui787s2f;tcp
-------kjh6$


F3 SEND  a.example.com -> Carol (transport WSS)

MSRP re58 SEND
To-Path: msrps://jk9awp14vj8x.invalid:2855/76qwe;ws
From-Path: msrps://a.example.com:2855/iwnslt;tcp \
           msrps://a.example.com:2855/jui787s2f;tcp \
           msrps://df7jal23ls0d.invalid/98cjs;ws
Success-Report: no
Byte-Range: 1-*/*
Content-Type: text/plain

Carol, here is the file Bob sent me.
-------re58$


F4 200 OK  Carol -> a.example.com (transport WSS)

MSRP re58 200 OK
To-Path: msrps://a.example.com:2855/iwnslt;tcp
From-Path: msrps://jk9awp14vj8x.invalid:2855/76qwe;ws
-------re58$
```

9. Implementation Status

Note to RFC Editor: Please remove this section and the reference to [RFC6982] before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

9.1. Kamailio SIP Server

Organization: Kamailio

Name: Kamailio v4.0.0 (4.0.0 http://www.kamailio.org/w/kamailio-v4-0-0-release-notes/)

Description: Kamailio (former OpenSER) is an Open Source SIP Server, able to handle thousands of call setups per second. (http://www.kamailio.org)

Level of maturity: Beta

Coverage: This module implements a WebSocket (RFC 6455) server and provides connection establishment (handshaking), management (including connection keep-alive), and framing for the SIP and MSRP WebSocket sub-protocols (draft-ietf-sipcore-sip-websocket and draft-pd-dispatch-msrp-websocket). The module supports WebSockets (ws) and secure WebSockets (wss).

Licensing: Open Source GPLv2

Contact: http://www.kamailio.org/w/contact-us/

URL: http://git.sip-router.org/cgi-bin/gitweb.cgi?p=kamailio;a=tree;f=modules/websocket;h=e75c6cd28493f812a955eeff9e64905aee01bcbf;hb=HEAD

http://git.sip-router.org/cgi-bin/gitweb.cgi?p=kamailio;a=tree;f=modules/msrp;h=0ffaeb57fb43a4d429680209262ad847f7ce6074;hb=HEAD

9.2. Crocodile MSRP

Organization: Crocodile RCS Ltd.

Name: Crocodile MSRP (https://github.com/crocodilertc/crocodile-msrp)

Description: Crocodile MSRP is a Javascript MSRP over WebSocket stack.

Level of maturity: Beta

Coverage: Open source client implementation of draft-pd-dispatch-msrp-websocket.

Licensing: Released under the MIT license (http://www.opensource.org/licenses/mit-license.php).

Contact: Gavin Llewellyn (gavin.llewellyn@crocodilertc.net)

URL: https://github.com/crocodilertc/crocodile-msrp

10. Security Considerations

10.1. Secure WebSocket Connection

It is RECOMMENDED that the MSRP traffic transported over a WebSocket communication be protected by using a secure WebSocket connection (using TLS [RFC5246] over TCP).

11. IANA Considerations

11.1. Registration of the WebSocket MSRP Sub-Protocol

This specification requests IANA to register the WebSocket MSRP sub-protocol in the registry of WebSocket sub-protocols with the following data:

Subprotocol Identifier: msrp

Subprotocol Common Name: WebSocket Transport for MSRP (Message Session Relay Protocol)

Subprotocol Definition: TBD, it should point to this document

12. Acknowledgements

Special thanks to Inaki Baz Castillo, Jose Luis Millan Villegas, and Victor Pascual, the authors of [I-D.ietf-sipcore-sip-websocket] which has inspired this draft.

Additional thanks to Inaki Baz Castillo who pointed out that "web-browser" shouldn't be used all the time as this specification should be valid for smartphones and apps other than browsers and suggested clarifications to the SDP handling for MSRP over WebSocket.

Special thanks to James Wyatt from Crocodile RCS Ltd for helping with the JavaScript MSRP over WebSockets prototyping.

Thanks to Saul Ibarra Corretge for suggesting that the existing MSRP keep alive mechanism may be used when WebSocket pings are not available.

13. References

13.1. Normative References

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

[RFC4975] Campbell, B., Mahy, R., and C. Jennings, "The Message Session Relay Protocol (MSRP)", RFC 4975, September 2007.

[RFC4976] Jennings, C., Mahy, R., and A. Roach, "Relay Extensions for the Message Sessions Relay Protocol (MSRP)", RFC 4976, September 2007.

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

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

13.2. Informative References

[CORS] W3C and A. van Kesteren, Ed., "Cross-Origin Resource Sharing", January 2013.

[I-D.ietf-sipcore-sip-websocket] Castillo, I., Millan, J., and V. Pascual, "The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP)", draft-ietf-sipcore-sip-websocket-09 (work in progress), June 2013.

[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.

[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.

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

[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.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

[WS-API] W3C and I. Hickson, Ed., "The WebSocket API", September 2012.

Appendix A. Implementation Guidelines

This section is non-normative.

A.1. MSRP 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. Therefore the MSRP WebSocket Client constructs 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 From-Path headers.

> The From-Path URI provided by MSRP clients which use an MSRP relay is not used for routing MSRP messages, thus it is safe to set a random domain in the hostpart of the From-Path URI.

Authors' Addresses

Peter Dunkley
Crocodile RCS Ltd
Forum 3, Parkway
Solent Business Park, Whiteley
Fareham PO15 7FH
United Kingdom

Email: peter.dunkley@crocodilertc.net

Gavin Llewellyn
Crocodile RCS Ltd
Forum 3, Parkway
Solent Business Park, Whiteley
Fareham PO15 7FH
United Kingdom

Email: gavin.llewellyn@crocodilertc.net

Victor Pascual
Quobis

Email: victor.pascual@quobis.com

Anton Roman
Quobis

Email: anton.roman@quobis.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
US

Email: gsalguei@cisco.com