Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: January 7, 2016

A. Johnston
Avaya
P. Zimmermann
J. Callas
Silent Circle
T. Cross
OfficeTone
J. Yoakum
Avaya

July 6, 2015

Using ZRTP to Secure WebRTC
draft-johnston-rtcweb-zrtp-02

Abstract

WebRTC, Web Real-Time Communications, is a set of protocols and APIs used to enable web developers to add real-time communications into their web pages and applications with a few lines of JavaScript. WebRTC media flows are encrypted and authenticated by SRTP, the Secure Real-time Transport Protocol while the key agreement is provided by DTLS-SRTP, Datagram Transport Layer Security for Secure Real-time Transport Protocol. However, without some third party identity service or certificate authority, WebRTC media flows have no protection against a man-in-the-middle (MitM) attack. ZRTP, Media Path Key Agreement for Unicast Secure RTP, RFC 6189, does provide protection against MitM attackers using key continuity augmented with a Short Authentication String (SAS). This specification describes how ZRTP can be used over the WebRTC data channel to provide MitM protection for WebRTC media flows keyed using DTLS-SRTP. This provides users protection against MitM attackers without requiring browsers to support ZRTP or users to download a plugin or extension to implement ZRTP.

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 January 7, 2016.

Table of Contents

1. Introduction

WebRTC, Web Real-Time Communications, adds real-time, interactive voice and video media capabilities to browsers [I-D.ietf-rtcweb-overview] without a plugin or download, and allows web developers to access this functionality using JavaScript API calls [WebRTC-API]. For a complete description of WebRTC protocols and APIs see [WebRTC-Book]. In addition, WebRTC supports the establishment of a peer-to-peer data channel between browsers [I-D.ietf-rtcweb-data-channel]. This document describes how ZRTP, Media Path Key Agreement for Unicast Secure RTP, [RFC6189], can be used over the WebRTC data channel to secure voice and video sessions established using WebRTC.

The security of voice and video media sessions established using WebRTC is described in [I-D.ietf-rtcweb-security]. All media

sessions utilize SRTP encryption and authentication, which relies on DTLS-SRTP for key management. DTLS-SRTP can utilize TLS modes offering perfect forward secrecy (PFS), but relies on the exchange of fingerprints for protection against Man-in-the-Middle (MitM) attacks [RFC5763]. A mechanism for utilizing trusted third parties, known as Identity Providers, to authenticate the fingerprint is also described. ZRTP always offers perfect forward secrecy, and protects against MitM attacks with key continuity, Short Authentication Strings (SAS), and optionally and additionally, with long-term signing keys or shared secrets. For subsequent calls between the same ZRTP endpoints, a hash of previous keying material is mixed in when generating the current keying material. In addition, the SAS can be used to confirm the absence of a MitM attack over the entire lifetime of the key continuity (going both backwards and forwards in time). Both parties in the communication must have ZRTP software, which performs a DH key agreement and are capable of storing a cache of previous shared secrets and rendering the SAS to the users. The human users then have the option to compare the SAS's to see if they match to confirm the absence of a MitM attacker. This could be done by verbally reading aloud the string (which can be two words or four hex characters), or otherwise exchanging them. If the SAS values match, then there is no MitM attacker. ZRTP is signaling channel and protocol independent, and does not rely on ANY third party services for authentication (though it can optionally and additionally leverage a public key infrastructure (PKI)). As such, ZRTP has been used with SIP, Jingle, and proprietary signaled VoIP systems. There are a number of open source ZRTP stacks and commercial implementations and products. For the reasons why ZRTP is a good fit for WebRTC, see [I-D.johnston-rtcweb-media-privacy].

ZRTP is not currently built into the browser like DTLS-SRTP. However, this doesn't mean that ZRTP cannot be used with WebRTC. ZRTP can be implemented in JavaScript and run over the WebRTC data channel between the browsers. The format and message flow can be identical to RFC 6189, with the exception that instead of ZRTP running on UDP, it runs on top of SCTP/DTLS/UDP. A small change in the policy usage of the ZRTP auxsecret provides MitM protection for media sessions established by WebRTC between the browsers.

This allows the ZRTP SAS to be used to authenticate WebRTC media sessions for WebRTC applications that include ZRTP JavaScript. Also, since the ZRTP data channel can be used to authenticate all WebRTC Peer Connections between a pair of browsers, a ZRTP WebRTC application could be used to authenticate and protect other WebRTC sessions that do not even use ZRTP. For example, users of a particular WebRTC service which claims to offer end-to-end media privacy could use a ZRTP-enabled WebRTC application in another tab or

window to verify that assertion or audit the service and protect against MitM attacks.

1.1. 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].

2. ZRTP over a WebRTC Data Channel

In the base ZRTP protocol [RFC6189], ZRTP uses UDP transport, multiplexed over the same port as the media session that it is keying. ZRTP over a WebRTC data channel means that ZRTP messages are sent over the SCTP/DTLS/UDP protocol stack. It is RECOMMENDED that SCTP reliability be used so that the ZRTP timer and retransmissions in Section 6 of [RFC6189] are not needed. The state machine is identical, with the exchange beginning with the Hello and ending with the ConfACK. The ZRTP Hello Hash MAY be exchanged over the WebRTC signaling channel. The ZID MAY be statelessly generated by hashing the DTLS-SRTP fingerprint of the browser. Also, the ZRTP cache of previous shared secrets can be stored in a number of ways, including indexed database, HTML5 file system, or even as a cookie.

In order to provide protection against a MitM attack of WebRTC media sessions, ZRTP needs to:

- Verify that both browsers see the same local and remote fingerprint used by DTLS-SRTP. This is accomplished by always including the DTLS-SRTP fingerprints in the ZRTP auxsecret.

- Verify that there is no MitM attack against ZRTP. This is accomplished by the various mechanisms ZRTP provides, including key continuity and human users comparing the SAS.

The ZRTP auxsecret is defined in Section 4.3 of [RFC6189]. This specification defines the following new policies relating to the usage of auxsecret when ZRTP is used to secure DTLS-SRTP media sessions.

The auxsecret MUST be used. The auxsecret is truncated to the negotiated hash length (defined in Section 4.5.1 of [RFC6189]) of:

```
auxsecret = hash(initiator's DTLS-SRTP fingerprint ||
                 responder's DTLS-SRTP fingerprint ||
                 original_auxsecret)
```

The original_auxsecret is any auxsecret value that would otherwise have been used with ZRTP, or the null string if no such value exists as will ordinarily be the case.

Note that this auxsecret is actually not a secret, since the fingerprints are hashes of known public keys used by the browsers. This does not affect the security of ZRTP.

If the auxsecrets of the initiator and responder do not match, this MUST be treated as a MitM attack. This is to protect against the case where the DTLS-SRTP session has an MitM attacker but the ZRTP session does not. Note that this can be done as soon as the DHPart1 and DHPart2 messages have been exchanged and can be done automatically without calculating or comparing the SAS.

Any failure in the ZRTP exchange MUST be treated as a MitM attack.

Detection of a MitM attack MUST result in the closure of the DTLS-SRTP sessions and alerting the browser users.

If the users successfully compare the SAS strings, it means that neither the DTLS nor the ZRTP sessions have MitM attackers. Any media sessions which were established using this same pair of local and remote fingerprints also do not have MitM attackers, regardless of which browser tab or window they are present in.

This specification requires DTLS to use a Forward Secrecy (FS) mode. If a FS mode is not available, the DTLS connection MUST fail.

3. IANA Considerations

This memo includes no request to IANA.

4. Security Considerations

For the security analysis of this approach, consider a pair of browsers, used by Alice and Bob which have established at a minimum a voice media session and a ZRTP data channel. There are two possibilities:

- Both the media and data run over the same DTLS connection, or
- The media and data run over separate DTLS connections.

As such, an attacker could choose to attack any combination of these connections and the DTLS and/or ZRTP protocols. However, note that since ZRTP runs on top of DTLS, it is not possible to MitM ZRTP without first launching a MitM attack on the DTLS connection over

which it runs. In the following analysis, "attacking the media channel" means a MitM attack launched against the DTLS session used to establish the voice media session, and "attacking the data channel" means a MitM attack against ZRTP and the DTLS session over which ZRTP runs.

Given these two possibilities, the attacker could choose to attack:

- Both the media and data channel,
- Just the media channel,
- Just the data channel, or
- Neither media or data channel.

These will be considered in turn. Note that a MitM attack launched against DTLS-SRTP will result in the remote fingerprint as seen by each browser to be that of the attacker instead of the other browser.

If the MitM attacks both the media and the data channel, the SAS as computed by each browser will be different, and the users can detect this by verbally comparing the SAS. Additionally, if the users have communicated before without a MitM attacker, the presence of the MitM will create a break in key continuity and the users will be alerted that they should verify the SAS.

If the MitM attacks just the media channel, after the exchange of DHPart1 and DHPart2 messages, the different fingerprints will be detected by checking the hashed auxsecret values and discovering that they do not match. The MitM attack is immediately and automatically detected.

If the MitM attacks just the data channel, the SAS as computed by each browser will be different as two independent DH exchanges occurred. If the users have spoken before, the MitM will cause a break in key continuity. In any case, the MitM will be definitively detected by comparing ZRTP's SAS. Note that it doesn't make much sense for the MitM to attack just the data channel, but this could happen.

If the MitM attacks neither the media nor the data channel, the auxsecrets will match, the SAS as computed by each browser will be the same, and key continuity will be maintained. As a result, both the ZRTP and media session are free of MitM attackers.

Note that only in one scenario does this approach rely on the users comparing the SAS -- and even there, the users would likely be

protected by key continuity even if the SAS were not manually checked. Also, note that all these attacks rely on the attacker being able to insert herself in the path as a MitM. For the scenario in which the media channel and data channel use different DTLS connections, it could be potentially difficult for the attacker to insert herself as a MitM in the data channel as it could take a complete different route over the Internet from the media channel. For example, the data channel used by ZRTP could be deliberately routed over a different IP connection or via a TURN server forcing a different path that may not accessible to the attacker.

In summary, this approach can be thought of as having three distinct layers. The first layer is the DTLS session, which protects against passive attacks but has no protection against a MitM attack without a third party service. The next layer is the ZRTP session, which allows the fingerprints to be exchanged and compared. A fingerprint mismatch allows a MitM attack on DTLS to be detected. The third layer is ZRTP and its protections against a MitM: short authentication strings, key continuity, and optional SAS signing with a PKI. These protections are cumulative -- even over time. Because of key continuity, a single comparison of the SAS guarantees that no MitM has attacked past sessions and cannot attack future sessions. And even if the SAS is not compared, key continuity ensures that for a MitM attacker to remain undetected, she must attack each session between the users without exception.

5. Implementation Status

Note to RFC Editor: Please remove this entire section prior to publication, including the reference to RFC 6982.

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".

An implementation of ZRTP over the data channel was developed for the TADHack 2015 Hackathon [TADHACK]. The preliminary implementation of the ZRTP JavaScript library, zrtp4js, was developed by Werner Dittman, using the Stanford JavaScript Crypto Library [SJCL] while the JavaScript application was developed by Dan Burnett, Alan Johnston, and Mahak Patel. A video is available at [TADHACK-ZRTP]. The next section shows an example of a ZRTP message from that implementation.

6. Appendix A: ZRTP JSON Encoding

For ZRTP running over the data channel between two browsers, a JavaScript Object Notation (JSON) encoding for ZRTP messages will simplify the JavaScript parsing and allow the "WebRTC String" PPID to be used over the data channel [I-D.ietf-rtcweb-data-channel]. Below is an example of an ZRTP DHPart1 message.

```
{
        "_data": {
                "_zrtpId":20570,
                "_length":0,
                "_messageType":"DHPart1",
                "_pv":"{\"x\":[9238189,9730715,15220722,
                        188609,9862612,13314338,
                        6663378,7562888,1865421,
                        11918668,16604863,755888,
                        14892086,11391512,6379092,210972],
                        \"y\":[14957537,2388710,8838444,
                        7786690,12191288,4191233,
                        10790862,1805965,14966182,
                        2627552,14359332,2309769,
                        8590080,6006833,7363525,1740185]}",
                "_aux":"[1622940278,772233674,93017422,
                        -1125510277,-2095646614,801724212,
                        -663866781,800173874]"
        }
}
```

7. Informative References

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-14 (work in progress), June 2015.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.

[I-D.johnston-rtcweb-media-privacy]
Johnston, A. and P. Zimmermann, "RTCWEB Media Privacy", draft-johnston-rtcweb-media-privacy-00 (work in progress), May 2011.

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

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC6189] Zimmermann, P., Johnston, A., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, April 2011.

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

[SJCL] "Stanford JavaScript Crypto Library", SJCL https://bitwiseshiftleft.github.io/sjcl/, 2015, <https://bitwiseshiftleft.github.io/sjcl/>.

[TADHACK] "Telecom Application Developer Hackathon", TADHack 2015 http://www.tadhack.com/2015/, 2015, <http://www.tadhack.com/2015/>.

[TADHACK-ZRTP]
"Telecom Application Developer Hackathon Remote Entry: WebRTC Security", TADHack 2015 Remote Entry https://www.youtube.com/watch?v=MOR2AYuRZ48, 2015, <https://www.youtube.com/watch?v=MOR2AYuRZ48>.

[WebRTC-API]
Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", W3C Working Draft http://www.w3.org/TR/webrtc/, 2013, <http://www.w3.org/TR/2012/WD-webrtc-20120821/>.

[WebRTC-Book]
Johnston, A. and D. Burnett, "WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web", 3rd Edition, Digital Codex LLC, 2014, <http://webrtcbook.com>.

Authors' Addresses

Alan Johnston
Avaya
St. Louis, MO
USA

Email: alan.b.johnston@gmail.com

Phil Zimmermann
Silent Circle
Santa Cruz, CA
USA

Email: prz@mit.edu

Jon Callas
Silent Circle

Email: jon@callas.org

Travis Cross
OfficeTone

Email: tc@traviscross.com

John Yoakum
Avaya
Cary, NC
USA

Email: yoakum@avaya.com