RTCWEB Security Architecture
draft-ietf-rtcweb-security-arch-02

Abstract

   The Real-Time Communications on the Web (RTCWEB) working group is
   tasked with standardizing protocols for enabling real-time
   communications within user-agents using web technologies (e.g
   JavaScript). The major use cases for RTCWEB technology are real-time
   audio and/or video calls, Web conferencing, and direct data transfer.
   Unlike most conventional real-time systems (e.g., SIP-based soft
   phones) RTCWEB communications are directly controlled by some Web
   server, which poses new security challenges. For instance, a Web
   browser might expose a JavaScript API which allows a server to place
   a video call. Unrestricted access to such an API would allow any
   site which a user visited to "bug" a user's computer, capturing any
   activity which passed in front of their camera. [I-D.ietf-rtcweb-
   security] defines the RTCWEB threat model. This document defines an
   architecture which provides security within that threat model.

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 December 7, 2012.

Copyright Notice

Table of Contents

1. Introduction

The Real-Time Communications on the Web (RTCWEB) working group is
tasked with standardizing protocols for real-time communications
between Web browsers. The major use cases for RTCWEB technology are
real-time audio and/or video calls, Web conferencing, and direct data
transfer. Unlike most conventional real-time systems, (e.g., SIP-
based[RFC3261] soft phones) RTCWEB communications are directly
controlled by some Web server, as shown in Figure 1.

```
                       +----------------+
                       |                |
                       |   Web Server   |
                       |                |
                       +----------------+
                          ^          ^
                         /            \
                 HTTP   /              \   HTTP
                       /                \
                      /                  \
                     v                    v
                   JS API               JS API
             +-----------+        +-----------+
             |           | Media  |           |
             |  Browser  |<---------->|  Browser  |
             |           |        |           |
             +-----------+        +-----------+
```
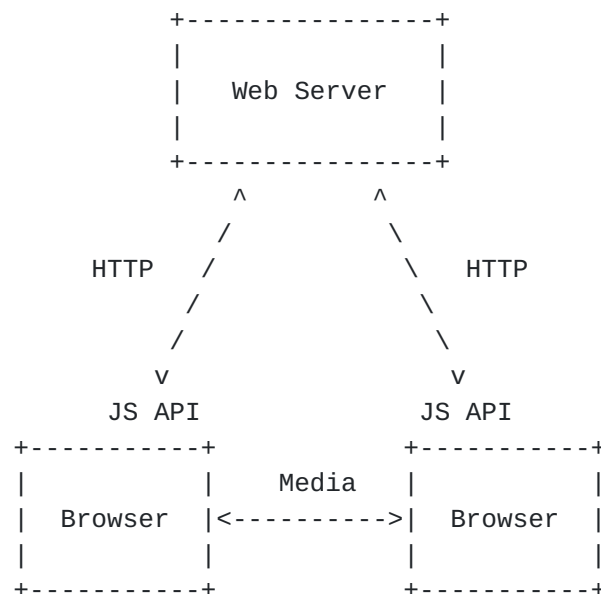
Figure 1: A simple RTCWEB system

This system presents a number of new security challenges, which are
analyzed in [I-D.ietf-rtcweb-security]. This document describes a
security architecture for RTCWEB which addresses the threats and
requirements described in that document.

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 RFC 2119 [RFC2119].

3. Trust Model

The basic assumption of this architecture is that network resources
exist in a hierarchy of trust, rooted in the browser, which serves as
the user's TRUSTED COMPUTING BASE (TCB). Any security property which

the user wishes to have enforced must be ultimately guaranteed by the browser (or transitively by some property the browser verifies). Conversely, if the browser is compromised, then no security guarantees are possible. Note that there are cases (e.g., Internet kiosks) where the user can't really trust the browser that much. In these cases, the level of security provided is limited by how much they trust the browser.

Optimally, we would not rely on trust in any entities other than the browser. However, this is unfortunately not possible if we wish to have a functional system. Other network elements fall into two categories: those which can be authenticated by the browser and thus are partly trusted--though to the minimum extent necessary--and those which cannot be authenticated and thus are untrusted. This is a natural extension of the end-to-end principle.

3.1. Authenticated Entities

There are two major classes of authenticated entities in the system:

o Calling services: Web sites whose origin we can verify (optimally via HTTPS).
o Other users: RTCWEB peers whose origin we can verify cryptographically (optimally via DTLS-SRTP).

Note that merely being authenticated does not make these entities trusted. For instance, just because we can verify that https://www.evil.org/ is owned by Dr. Evil does not mean that we can trust Dr. Evil to access our camera an microphone. However, it gives the user an opportunity to determine whether he wishes to trust Dr. Evil or not; after all, if he desires to contact Dr. Evil (perhaps to arrange for ransom payment), it's safe to temporarily give him access to the camera and microphone for the purpose of the call, but he doesn't want Dr. Evil to be able to access his camera and microphone other than during the call. The point here is that we must first identify other elements before we can determine whether and how much to trust them.

It's also worth noting that there are settings where authentication is non-cryptographic, such as other machines behind a firewall. Naturally, the level of trust one can have in identities verified in this way depends on how strong the topology enforcement is.

3.2. Unauthenticated Entities

Other than the above entities, we are not generally able to identify other network elements, thus we cannot trust them. This does not mean that it is not possible to have any interaction with them, but

it means that we must assume that they will behave maliciously and
design a system which is secure even if they do so.

[4](#). Overview

This section describes a typical RTCWeb session and shows how the
various security elements interact and what guarantees are provided
to the user. The example in this section is a "best case" scenario
in which we provide the maximal amount of user authentication and
media privacy with the minimal level of trust in the calling service.
Simpler versions with lower levels of security are also possible and
are noted in the text where applicable. It's also important to
recognize the tension between security (or performance) and privacy.
The example shown here is aimed towards settings where we are more
concerned about secure calling than about privacy, but as we shall
see, there are settings where one might wish to make different
tradeoffs--this architecture is still compatible with those settings.

For the purposes of this example, we assume the topology shown in the
figure below. This topology is derived from the topology shown in
Figure 1, but separates Alice and Bob's identities from the process
of signaling. Specifically, Alice and Bob have relationships with
some Identity Provider (IdP) that supports a protocol such OpenID or
BrowserID) that can be used to attest to their identity. This
separation isn't particularly important in "closed world" cases where
Alice and Bob are users on the same social network and have
identities based on that network. However, there are important
settings where that is not the case, such as federation (calls from
one network to another) and calling on untrusted sites, such as where
two users who have a relationship via a given social network want to
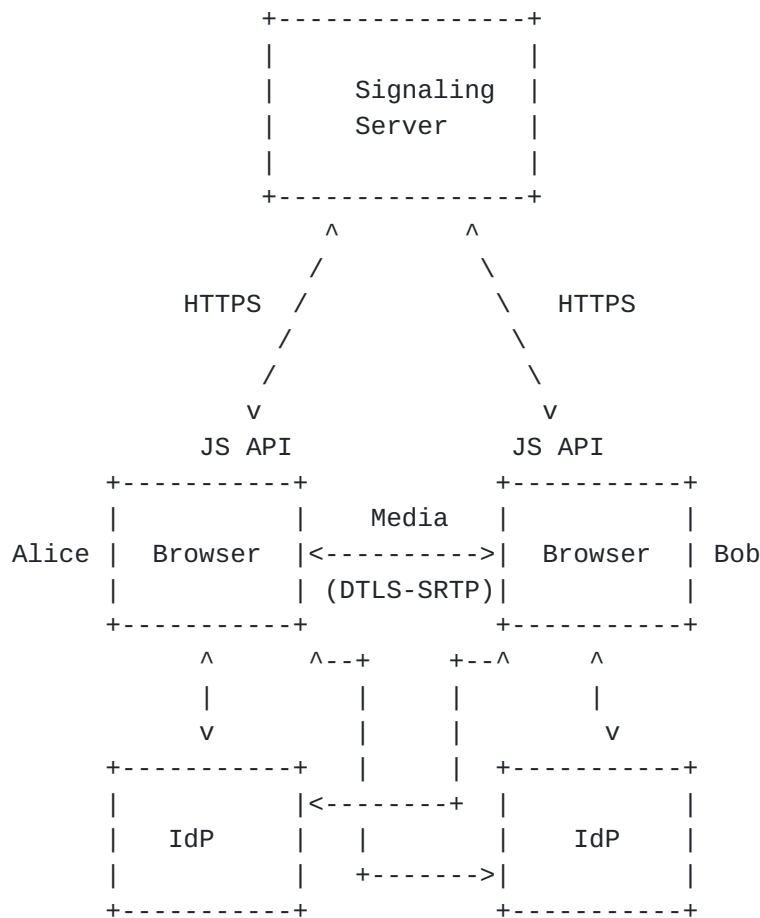call each other on another, untrusted, site, such as a poker site.

```
                        +----------------+
                        |                |
                        |    Signaling   |
                        |     Server     |
                        |                |
                        +----------------+
                            ^          ^
                           /            \
                   HTTPS  /              \   HTTPS
                         /                \
                        /                  \
                       v                    v
                    JS API               JS API
                +-----------+        +-----------+
                |           |  Media |           |
        Alice   |  Browser  |<---------->|  Browser  | Bob
                |           | (DTLS-SRTP)|           |
                +-----------+        +-----------+
                    ^       ^--+      +--^       ^
                    |       |    |    |          |
                    v       |    |    |          v
                +-----------+    |    |  +-----------+
                |           |<--------+  |           |
                |    IdP    |    |    |  |    IdP    |
                |           |    | +------->|           |
                +-----------+    | +------->|           +-----------+
```

                Figure 2: A call with IdP-based identity

[4.1](#). Initial Signaling

   Alice and Bob are both users of a common calling service; they both
   have approved the calling service to make calls (we defer the
   discussion of device access permissions till later). They are both
   connected to the calling service via HTTPS and so know the origin
   with some level of confidence. They also have accounts with some
   identity provider. This sort of identity service is becoming
   increasingly common in the Web environment in technologies such
   (BrowserID, Federated Google Login, Facebook Connect, OAuth, OpenID,
   WebFinger), and is often provided as a side effect service of your
   ordinary accounts with some service. In this example, we show Alice
   and Bob using a separate identity service, though they may actually
   be using the same identity service as calling service or have no
   identity service at all.

   Alice is logged onto the calling service and decides to call Bob. She
   can see from the calling service that he is online and the calling
   service presents a JS UI in the form of a button next to Bob's name

which says "Call". Alice clicks the button, which initiates a JS
callback that instantiates a PeerConnection object. This does not
require a security check: JS from any origin is allowed to get this
far.

Once the PeerConnection is created, the calling service JS needs to
set up some media. Because this is an audio/video call, it creates
two MediaStreams, one connected to an audio input and one connected
to a video input. At this point the first security check is
required: untrusted origins are not allowed to access the camera and
microphone. In this case, because Alice is a long-term user of the
calling service, she has made a permissions grant (i.e., a setting in
the browser) to allow the calling service to access her camera and
microphone any time it wants. The browser checks this setting when
the camera and microphone requests are made and thus allows them.

In the current W3C API, once some streams have been added, Alice's
browser + JS generates a signaling message The format of this data is
currently undefined. It may be a complete message as defined by ROAP
[I-D.jennings-rtcweb-signaling] or separate media description and
transport messages as defined in [I-D.ietf-rtcweb-jsep] or may be
assembled piecemeal by the JS. In either case, it will contain:

o Media channel information
o ICE candidates
o A fingerprint attribute binding the communication to Alice's
   public key [RFC5763]

[Note that it is currently unclear where JSEP will eventually put
this information, in the SDP or in the transport info.] Prior to
sending out the signaling message, the PeerConnection code contacts
the identity service and obtains an assertion binding Alice's
identity to her fingerprint. The exact details depend on the
identity service (though as discussed in
[I-D.rescorla-rtcweb-generic-idp] PeerConnection can be agnostic to
them), but for now it's easiest to think of as a BrowserID assertion.
The assertion may bind other information to the identity besides the
fingerprint, but at minimum it needs to bind the fingerprint.

This message is sent to the signaling server, e.g., by XMLHttpRequest
[XmlHttpRequest] or by WebSockets [RFC6455] The signaling server
processes the message from Alice's browser, determines that this is a
call to Bob and sends a signaling message to Bob's browser (again,
the format is currently undefined). The JS on Bob's browser
processes it, and alerts Bob to the incoming call and to Alice's
identity. In this case, Alice has provided an identity assertion and
so Bob's browser contacts Alice's identity provider (again, this is
done in a generic way so the browser has no specific knowledge of the

IdP) to verity the assertion. This allows the browser to display a
trusted element indicating that a call is coming in from Alice. If
Alice is in Bob's address book, then this interface might also
include her real name, a picture, etc. The calling site will also
provide some user interface element (e.g., a button) to allow Bob to
answer the call, though this is most likely not part of the trusted
UI.

If Bob agrees [I am ignoring early media for now], a PeerConnection
is instantiated with the message from Alice's side. Then, a similar
process occurs as on Alice's browser: Bob's browser verifies that
the calling service is approved, the media streams are created, and a
return signaling message containing media information, ICE
candidates, and a fingerprint is sent back to Alice via the signaling
service. If Bob has a relationship with an IdP, the message will
also come with an identity assertion.

At this point, Alice and Bob each know that the other party wants to
have a secure call with them. Based purely on the interface provided
by the signaling server, they know that the signaling server claims
that the call is from Alice to Bob. Because the far end sent an
identity assertion along with their message, they know that this is
verifiable from the IdP as well. Of course, the call works perfectly
well if either Alice or Bob doesn't have a relationship with an IdP;
they just get a lower level of assurance. Moreover, Alice might wish
to make an anonymous call through an anonymous calling site, in which
case she would of course just not provide any identity assertion and
the calling site would mask her identity from Bob.

4.2. Media Consent Verification

As described in ([I-D.ietf-rtcweb-security]; Section 4.2) This
proposal specifies that media consent verification be performed via
ICE. Thus, Alice and Bob perform ICE checks with each other. At the
completion of these checks, they are ready to send non-ICE data.

At this point, Alice knows that (a) Bob (assuming he is verified via
his IdP) or someone else who the signaling service is claiming is Bob
is willing to exchange traffic with her and (b) that either Bob is at
the IP address which she has verified via ICE or there is an attacker
who is on-path to that IP address detouring the traffic. Note that
it is not possible for an attacker who is on-path but not attached to
the signaling service to spoof these checks because they do not have
the ICE credentials. Bob's security guarantees with respect to Alice
are the converse of this.

4.3. DTLS Handshake

Once the ICE checks have completed [more specifically, once some ICE
checks have completed], Alice and Bob can set up a secure channel.
This is performed via DTLS [RFC4347] (for the data channel) and DTLS-
SRTP [RFC5763] for the media channel. Specifically, Alice and Bob
perform a DTLS handshake on every channel which has been established
by ICE. The total number of channels depends on the amount of
muxing; in the most likely case we are using both RTP/RTCP mux and
muxing multiple media streams on the same channel, in which case
there is only one DTLS handshake. Once the DTLS handshake has
completed, the keys are exported [RFC5705] and used to key SRTP for
the media channels.

At this point, Alice and Bob know that they share a set of secure
data and/or media channels with keys which are not known to any
third-party attacker. If Alice and Bob authenticated via their IdPs,
then they also know that the signaling service is not attacking them.
Even if they do not use an IdP, as long as they have minimal trust in
the signaling service not to perform a man-in-the-middle attack, they
know that their communications are secure against the signaling
service as well.

4.4. Communications and Consent Freshness

From a security perspective, everything from here on in is a little
anticlimactic: Alice and Bob exchange data protected by the keys
negotiated by DTLS. Because of the security guarantees discussed in
the previous sections, they know that the communications are
encrypted and authenticated.

The one remaining security property we need to establish is "consent
freshness", i.e., allowing Alice to verify that Bob is still prepared
to receive her communications. ICE specifies periodic STUN
keepalizes but only if media is not flowing. Because the consent
issue is more difficult here, we require RTCWeb implementations to
periodically send keepalives. If a keepalive fails and no new ICE
channels can be established, then the session is terminated.

5. Detailed Technical Description

5.1. Origin and Web Security Issues

The basic unit of permissions for RTCWEB is the origin [RFC6454].
Because the security of the origin depends on being able to
authenticate content from that origin, the origin can only be
securely established if data is transferred over HTTPS [RFC2818].

Thus, clients MUST treat HTTP and HTTPS origins as different
permissions domains. [Note: this follows directly from the origin
security model and is stated here merely for clarity.]

Many web browsers currently forbid by default any active mixed
content on HTTPS pages. I.e., when JS is loaded from an HTTP origin
onto an HTTPS page, an error is displayed and the content is not
executed unless the user overrides the error. Any browser which
enforces such a policy will also not permit access to RTCWEB
functionality from mixed content pages. It is RECOMMENDED that
browsers which allow active mixed content nevertheless disable RTCWEB
functionality in mixed content settings. [[OPEN ISSUE: Should this
be a 2119 MUST? It's not clear what set of conditions would make
this OK, other than that browser manufacturers have traditionally
been permissive here here.]] Note that it is possible for a page
which was not mixed content to become mixed content during the
duration of the call. Implementations MAY choose to terminate the
call or display a warning at that point, but it is also permissible
to ignore this condition. This is a deliberate implementation
complexity versus security tradeoff. [[OPEN ISSUE:: Should we be
more aggressive about this?]]

5.2. Device Permissions Model

Implementations MUST obtain explicit user consent prior to providing
access to the camera and/or microphone. Implementations MUST at
minimum support the following two permissions models:

o Requests for one-time camera/microphone access.
o Requests for permanent access.

In addition, they SHOULD support requests for access to a single
communicating peer. E.g., "Call customerservice@ford.com". Browsers
servicing such requests SHOULD clearly indicate that identity to the
user when asking for permission.

API Requirement: The API MUST provide a mechanism for the requesting
   JS to indicate which of these forms of permissions it is
   requesting. This allows the client to know what sort of user
   interface experience to provide. In particular, browsers might
   display a non-invasive door hanger ("some features of this site
   may not work..." when asking for long-term permissions) but a more
   invasive UI ("here is your own video") for single-call
   permissions. The API MAY grant weaker permissions than the JS
   asked for if the user chooses to authorize only those permissions,
   but if it intends to grant stronger ones it SHOULD display the
   appropriate UI for those permissions and MUST clearly indicate
   what permissions are being requested.

API Requirement: The API MUST provide a mechanism for the requesting
   JS to relinquish the ability to see or modify the media (e.g., via
   MediaStream.record()). Combined with secure authentication of the
   communicating peer, this allows a user to be sure that the calling
   site is not accessing or modifying their conversion.

UI Requirement: The UI MUST clearly indicate when the user's camera
   and microphone are in use. This indication MUST NOT be
   suppressable by the JS and MUST clearly indicate how to terminate
   a call, and provide a UI means to immediately stop camera/
   microphone input without the JS being able to prevent it.

UI Requirement: If the UI indication of camera/microphone use are
   displayed in the browser such that minimizing the browser window
   would hide the indication, or the JS creating an overlapping
   window would hide the indication, then the browser SHOULD stop
   camera and microphone input. [Note: this may not be necessary in
   systems that are non-windows-based but that have good
   notifications support, such as phones.]

Clients MAY permit the formation of data channels without any direct
user approval. Because sites can always tunnel data through the
server, further restrictions on the data channel do not provide any
additional security. (though see Section 5.3 for a related issue).

Implementations which support some form of direct user authentication
SHOULD also provide a policy by which a user can authorize calls only
to specific counterparties. Specifically, the implementation SHOULD
provide the following interfaces/controls:

o Allow future calls to this verified user.
o Allow future calls to any verified user who is in my system
   address book (this only works with address book integration, of
   course).

Implementations SHOULD also provide a different user interface
indication when calls are in progress to users whose identities are
directly verifiable. Section 5.5 provides more on this.

5.3. Communications Consent

Browser client implementations of RTCWEB MUST implement ICE. Server
gateway implementations which operate only at public IP addresses may
implement ICE-Lite.

Browser implementations MUST verify reachability via ICE prior to
sending any non-ICE packets to a given destination. Implementations
MUST NOT provide the ICE transaction ID to JavaScript during the

lifetime of the transaction (i.e., during the period when the ICE
stack would accept a new response for that transaction). [Note:
this document takes no position on the split between ICE in JS and
ICE in the browser. The above text is written the way it is for
editorial convenience and will be modified appropriately if the WG
decides on ICE in the JS.]

Implementations MUST send keepalives no less frequently than every 30
seconds regardless of whether traffic is flowing or not. If a
keepalive fails then the implementation MUST either attempt to find a
new valid path via ICE or terminate media for that ICE component.
Note that ICE [RFC5245]; Section 10 keepalives use STUN Binding
Indications which are one-way and therefore not sufficient. Instead,
the consent freshness mechanism [I-D.muthu-behave-consent-freshness]
MUST be used.

5.4. IP Location Privacy

A side effect of the default ICE behavior is that the peer learns
one's IP address, which leaks large amounts of location information,
especially for mobile devices. This has negative privacy
consequences in some circumstances. The following two API
requirements are intended to mitigate this issue:

API Requirement: The API MUST provide a mechanism to suppress ICE
   negotiation (though perhaps to allow candidate gathering) until
   the user has decided to answer the call [note: determining when
   the call has been answered is a question for the JS.] This
   enables a user to prevent a peer from learning their IP address if
   they elect not to answer a call and also from learning whether the
   user is online.

API Requirement: The API MUST provide a mechanism for the calling
   application to indicate that only TURN candidates are to be used.
   This prevents the peer from learning one's IP address at all. The
   API MUST provide a mechanism for the calling application to
   reconfigure an existing call to add non-TURN candidates. Taken
   together, these requirements allow ICE negotiation to start
   immediately on incoming call notification, thus reducing post-dial
   delay, but also to avoid disclosing the user's IP address until
   they have decided to answer.

5.5. Communications Security

Implementations MUST implement DTLS [RFC4347] and DTLS-SRTP
[RFC5763][RFC5764]. All data channels MUST be secured via DTLS.
DTLS-SRTP MUST be offered for every media channel and MUST be the
default; i.e., if an implementation receives an offer for DTLS-SRTP

and SDES, DTLS-SRTP MUST be selected. Media traffic MUST NOT be sent
over plain (unencrypted) RTP.

[OPEN ISSUE: What should the settings be here? MUST?]
Implementations MAY support SDES and RTP for media traffic for
backward compatibility purposes.

API Requirement: The API MUST provide a mechanism to indicate that a
   fresh DTLS key pair is to be generated for a specific call. This
   is intended to allow for unlinkability. Note that there are also
   settings where it is attractive to use the same keying material
   repeatedly, especially those with key continuity-based
   authentication.

API Requirement: The API MUST provide a mechanism to indicate that a
   fresh DTLS key pair is to be generated for a specific call. This
   is intended to allow for unlinkability.

API Requirement: When DTLS-SRTP is used, the API MUST NOT permit the
   JS to obtain the negotiated keying material. This requirement
   preserves the end-to-end security of the media.

UI Requirements: A user-oriented client MUST provide an
   "inspector" interface which allows the user to determine the
   security characteristics of the media. [largely derived from
   [I-D.kaufman-rtcweb-security-ui]
   The following properties SHOULD be displayed "up-front" in the
   browser chrome, i.e., without requiring the user to ask for them:

   * A client MUST provide a user interface through which a user may
      determine the security characteristics for currently-displayed
      audio and video stream(s)
   * A client MUST provide a user interface through which a user may
      determine the security characteristics for transmissions of
      their microphone audio and camera video.
   * The "security characteristics" MUST include an indication as to
      whether the cryptographic keys were delivered out-of-band (from
      a server) or were generated as a result of a pairwise
      negotiation.
   * If the far endpoint was directly verified, either via a third-
      party verifiable X.509 certificate or via a Web IdP mechanism
      (see Section 5.6) the "security characteristics" MUST include
      the verified information.
   The following properties are more likely to require some "drill-
   down" from the user:

    * The cryptographic algorithms in use (For example: "AES-CBC" or
       "Null Cipher".)
    * The "security characteristics" MUST indicate whether PFS is
       provided.
    * The "security characteristics" MUST include some mechanism to
       allow an out-of-band verification of the peer, such as a
       certificate fingerprint or an SAS.

5.6. Web-Based Peer Authentication

   In a number of cases, it is desirable for the endpoint (i.e., the
   browser) to be able to directly identity the endpoint on the other
   side without trusting only the signaling service to which they are
   connected. For instance, users may be making a call via a federated
   system where they wish to get direct authentication of the other
   side. Alternately, they may be making a call on a site which they
   minimally trust (such as a poker site) but to someone who has an
   identity on a site they do trust (such as a social network.)

   Recently, a number of Web-based identity technologies (OAuth,
   BrowserID, Facebook Connect), etc. have been developed. While the
   details vary, what these technologies share is that they have a Web-
   based (i.e., HTTP/HTTPS identity provider) which attests to your
   identity. For instance, if I have an account at example.org, I could
   use the example.org identity provider to prove to others that I was
   alice@example.org. The development of these technologies allows us
   to separate calling from identity provision: I could call you on
   Poker Galaxy but identify myself as alice@example.org.

   Whatever the underlying technology, the general principle is that the
   party which is being authenticated is NOT the signaling site but
   rather the user (and their browser). Similarly, the relying party is
   the browser and not the signaling site. Thus, the browser MUST
   securely generate the input to the IdP assertion process and MUST
   securely display the results of the verification process to the user
   in a way which cannot be imitated by the calling site.

   In order to make this work, we must standardize the following items:

   o The precise information from the signaling message that must be
      cryptographically bound to the user's identity. At minimum this
      MUST be the fingerprint, but we may choose to add other
      information as the signaling protocol firms up. This will be
      defined in a future version of this document.
   o The interface to the IdP. [I-D.rescorla-rtcweb-generic-idp]
      specifies a specific protocol mechanism which allows the use of
      any identity protocol without requiring specific further protocol

       support in the browser.
   o The JavaScript interfaces which the calling application can use to
      specify the IdP to use to generate assertions and to discover what
      assertions were received. These interfaces should be defined in
      the W3C document.

6. Security Considerations

   Much of the security analysis of this problem is contained in
   [I-D.ietf-rtcweb-security] or in the discussion of the particular
   issues above. In order to avoid repetition, this section focuses on
   (a) residual threats that are not addressed by this document and (b)
   threats produced by failure/misbehavior of one of the components in
   the system.

6.1. Communications Security

   While this document favors DTLS-SRTP, it permits a variety of
   communications security mechanisms and thus the level of
   communications security actually provided varies considerably. Any
   pair of implementations which have multiple security mechanisms in
   common are subject to being downgraded to the weakest of those common
   mechanisms by any attacker who can modify the signaling traffic. If
   communications are over HTTP, this means any on-path attacker. If
   communications are over HTTPS, this means the signaling server.
   Implementations which wish to avoid downgrade attack should only
   offer the strongest available mechanism, which is DTLS/DTLS-SRTP.
   Note that the implication of this choice will be that interop to non-
   DTLS-SRTP devices will need to happen through gateways.

   Even if only DTLS/DTLS-SRTP are used, the signaling server can
   potentially mount a man-in-the-middle attack unless implementations
   have some mechanism for independently verifying keys. The UI
   requirements in Section 5.5 are designed to provide such a mechanism
   for motivated/security conscious users, but are not suitable for
   general use. The identity service mechanisms in Section 5.6 are more
   suitable for general use. Note, however, that a malicious signaling
   service can strip off any such identity assertions, though it cannot
   forge new ones. Note that all of the third-party security mechanisms
   available (whether X.509 certificates or a third-party IdP) rely on
   the security of the third party--this is of course also true of your
   connection to the Web site itself. Users who wish to assure
   themselves of security against a malicious identity provider MUST
   verify peer credentials directly, e.g., by checking the peer's
   fingerprint against a value delivered out of band.

6.2. Privacy

   The requirements in this document are intended to allow:

   o Users to participate in calls without revealing their location.
   o Potential callees to avoid revealing their location and even
      presence status prior to agreeing to answer a call.

   However, these privacy protections come at a performance cost in
   terms of using TURN relays and, in the latter case, delaying ICE.
   Sites SHOULD make users aware of these tradeoffs.

   Note that the protections provided here assume a non-malicious
   calling service. As the calling service always knows the users
   status and (absent the use of a technology like Tor) their IP
   address, they can violate the users privacy at will. Users who wish
   privacy against the calling sites they are using must use separate
   privacy enhancing technologies such as Tor. Combined RTCWEB/Tor
   implementations SHOULD arrange to route the media as well as the
   signaling through Tor. [Currently this will produce very suboptimal
   performance.]

6.3. Denial of Service

   The consent mechanisms described in this document are intended to
   mitigate denial of service attacks in which an attacker uses clients
   to send large amounts of traffic to a victim without the consent of
   the victim. While these mechanisms are sufficient to protect victims
   who have not implemented RTCWEB at all, RTCWEB implementations need
   to be more careful.

   Consider the case of a call center which accepts calls via RTCWeb.
   An attacker proxies the call center's front-end and arranges for
   multiple clients to initiate calls to the call center. Note that
   this requires user consent in many cases but because the data channel
   does not need consent, he can use that directly. Since ICE will
   complete, browsers can then be induced to send large amounts of data
   to the victim call center if it supports the data channel at all.
   Preventing this attack requires that automated RTCWEB
   implementations implement sensible flow control and have the
   ability to triage out (i.e., stop responding to ICE probes on) calls
   which are behaving badly, and especially to be prepared to remotely
   throttle the data channel in the absence of plausible audio and video
   (which the attacker cannot control).

   Another related attack is for the signaling service to swap the ICE
   candidates for the audio and video streams, thus forcing a browser to
   send video to the sink that the other victim expects will contain

audio (perhaps it is only expecting audio!) potentially causing
overload. Muxing multiple media flows over a single transport makes
it harder to individually suppress a single flow by denying ICE
keepalives. Media-level (RTCP) mechanisms must be used in this case.

Yet another attack, suggested by Magnus Westerlund, is for the
attacker to cross-connect offers and answers as follows. It induces
the victim to make a call and then uses its control of other users
browsers to get them to attempt a call to someone. It then
translates their offers into apparent answers to the victim, which
looks like large-scale parallel forking. The victim still responds
to ICE responses and now the browsers all try to send media to the
victim. [[OPEN ISSUE: How do we address this?]]

[TODO: Should we have a mechanism for verifying total expected
bandwidth]

Note that attacks based on confusing one end or the other about
consent are possible primarily even in the face of the third-party
identity mechanism as long as major parts of the signaling messages
are not signed. On the other hand, signing the entire message
severely restricts the capabilities of the calling application, so
there are difficult tradeoffs here.

7. Acknowledgements

Bernard Aboba, Harald Alvestrand, Dan Druta, Cullen Jennings, Hadriel
Kaplan, Matthew Kaufman, Martin Thomson, Magnus Westerland.

8. References

8.1. Normative References

[I-D.ietf-rtcweb-security]
            Rescorla, E., "Security Considerations for RTC-Web",
            draft-ietf-rtcweb-security-02 (work in progress),
            March 2012.

[I-D.muthu-behave-consent-freshness]
            Perumal, M., Wing, D., and H. Kaplan, "STUN Usage for
            Consent Freshness",
            draft-muthu-behave-consent-freshness-00 (work in
            progress), March 2012.

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

   [RFC2818]   Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

   [RFC4347]   Rescorla, E. and N. Modadugu, "Datagram Transport Layer
               Security", RFC 4347, April 2006.

   [RFC5245]   Rosenberg, J., "Interactive Connectivity Establishment
               (ICE): A Protocol for Network Address Translator (NAT)
               Traversal for Offer/Answer Protocols", RFC 5245,
               April 2010.

   [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.

   [RFC5764]   McGrew, D. and E. Rescorla, "Datagram Transport Layer
               Security (DTLS) Extension to Establish Keys for the Secure
               Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.

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

8.2. Informative References

   [I-D.ietf-rtcweb-jsep]
               Uberti, J. and C. Jennings, "Javascript Session
               Establishment Protocol", draft-ietf-rtcweb-jsep-00 (work
               in progress), March 2012.

   [I-D.jennings-rtcweb-signaling]
               Jennings, C., Rosenberg, J., and R. Jesup, "RTCWeb Offer/
               Answer Protocol (ROAP)",
               draft-jennings-rtcweb-signaling-01 (work in progress),
               October 2011.

   [I-D.kaufman-rtcweb-security-ui]
               Kaufman, M., "Client Security User Interface Requirements
               for RTCWEB", draft-kaufman-rtcweb-security-ui-00 (work in
               progress), June 2011.

   [I-D.rescorla-rtcweb-generic-idp]
               Rescorla, E., "RTCWeb Generic Identity Provider
               Interface", draft-rescorla-rtcweb-generic-idp-00 (work in
               progress), January 2012.

   [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.

   [RFC5705] Rescorla, E., "Keying Material Exporters for Transport
              Layer Security (TLS)", RFC 5705, March 2010.

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

   [XmlHttpRequest]
              van Kesteren, A., "XMLHttpRequest Level 2".

Author's Address

   Eric Rescorla
   RTFM, Inc.
   2064 Edgewood Drive
   Palo Alto, CA 94303
   USA

   Phone: +1 650 678 2350
   Email: ekr@rtfm.com