A WebRTC Extension to Allow Identity Negotiation at Runtime

Kevin Corre, Simon Bécot, Olivier Barais, Gerson Sunyé

A WebRTC Extension to Allow Identity Negotiation at Runtime

Kevin Corre[1,3], Simon Bécot[1], Olivier Barais[2], and Gerson Sunyé[2]

[1] Orange Labs, Cesson-Sevigne, France, @orange.com
[2] INRIA, Rennes, France @inria.fr
[3] IRISA, Rennes, France @irisa.fr

Abstract. In this paper we describe our implementation of the WebRTC identity architecture. We adapt OpenID Connect servers to support WebRTC peer to peer authentication and detail the issues and solutions found in the process. We observe that although WebRTC allows for the exchange of identity assertion between peers, users lack feedback and control over the other party authentication. To allow identity negotiation during a WebRTC communication setup, we propose an extension to the Session Description Protocol. Our implementation demonstrates current limitations with respect to the current WebRTC specification.

1 Introduction

In business communications, especially with previously unknown party, when dealing with high value information, or when having privacy concerns, ensuring the identity of the other party and the confidentiality of the communication is of the greatest importance. Usually Communication Services (CS) require and manage user authentication, secure call signaling, and may transmit the actual communication in the case of telecom operators. Trust between users and CSs is thus required in order to ensure the safety of the conversation. Telecom operators rely on a Circle of Trust model to allow CS interoperability. Joining the Circle of Trust ensures an *implicit initial level of trust* [1] to its members. On the contrary, Web CSs are usually organized in a silo model: to communicate together, two users must be using the same CS. Large identity federation are difficult to build and present complex issues [2]. As a result users calling outside of their enterprise's domain are often stranded in a silo situation, with the fallback option of using self-asserted identities.

WebRTC [3] is a Web API, specified by the W3C and the IETF, which supports Peer-to-Peer (P2P) audio-video calling and data sharing. It allows real-time communications between browsers, without additional plugins. The IETF-draft by Rescorla et al. [4], now part of the specification, offers a mechanism for explicit authentication, decoupling the identity and signaling functions.

It is expected to see the emergence of numerous WebRTC-enabled Web sites, resulting in a large number of security configurations. Even more if services and applications start being interoperable with each others. Designing new P2P

The final publication is available at Springer via
http://dx.doi.org/10.1007/978-3-319-60131-1_27.

service architecture enabling dynamic trusted relationships among distributed applications is currently explored within the Matrix[4] specification or within the reThink project[5]. However, in such kinds of distributed, dynamic and heterogeneous architecture, it may become difficult for users to understand if their communications are secured enough.

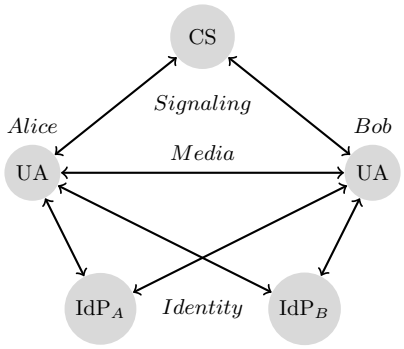
One of the challenge is then to build a trust and security model for communication services, capable of returning a single metric encompassing different aspects of the communication's security configuration. It would help users in making trust decision, and in return allow negotiation to raise trust in the communication's security.

In this paper we report and discuss on our implementation of the WebRTC identity specification. We first present the WebRTC identity specification in Section 2. In Section 3 we detail our implementation, and in particular the change we did to two OpenID Connect (OIDC) [5] server implementations to support WebRTC user to user authentication. A report of a similar work for the SAML protocol was already published in 2015 [6], but our work is to our knowledge the first implementation of a WebRTC IdP Proxy for OIDC. The WebRTC identity architecture gives an initial level of trust in the other party's identity. However, we show that users need more information and control in order to trust Identity Providers (IdP) used in the session establishment. In order to solve this issue, in Section 4, we propose a WebRTC extension to negotiate over the other-party authentication during a call setup. We implemented the negotiation, demonstrates it to just decorate the current protocol and remain compatible with existing user-agent and report on the limitations imposed by the current WebRTC specification. After discussing possible future work in Section 5, we conclude in Section 6.

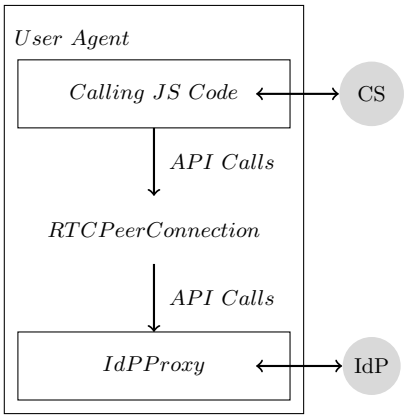
2 WebRTC Overview

WebRTC communication setup can be decomposed into three different paths as shown in Figure 1a. The Signaling path initializes the communication between Alice and Bob User-Agents (UA) through one or more Communication Service (CS) servers. On this path, Alice and Bob negotiate communication parameters by exchanging Session Description Protocol (SDP) [6] call offer and answer messages in order to setup the Media path. The Media path is a peer-to-peer encrypted connection between Alice and Bob used to exchange audio, video, or data. Optionally, one or two Identity paths can be used to generate and verify Identity Assertions. Identity Assertions bind the identity of the authenticated user to the media session, and are included in SDP signaling messages. Their role is to assert that media path encryption keys are used by users that are authenticated by IdPs.

[4] https://matrix.org
[5] https://rethink-project.eu/
[6] https://www.terena.org/activities/tf-webrtc/meeting2/slides/20150519-webrtc-identity.pdf



(a) A call with IdP-based identity and a single communication service provider. The Signaling path join Alice and Bob through a CS. The Media path is the P2P session. Each Identity path allows a user to authenticate to the other user through an IdP.



(b) API calls are the RTCPeerConnection's setIdentityProvider function, and the generateAssertion and validateAssertion functions, offered by the IdP Proxy to the RTCPeerConnection. The protocol used between the IdP Proxy and the IdP is unspecified.

Fig. 1: WebRTC Call Setup

The WebRTC security architecture [4] specifies the IdP Proxy component. This component serves as an interface between the WebRTC RTCPeerConnection object and an IdP. The IdP Proxy is supposed to be available at a standardized location on each compatible IdP domain. Before making a SDP call offer or answer, the RTCPeerConnection calls the IdP Proxy to generate an assertion covering a set of keys. Once the IdP authenticated the user, an identity assertion is returned. This assertion is then included in the SDP message with the IdP Proxy origin, i. e., the IdP domain's URL. This allows users to discover an IdP Proxy location without prior knowledge or relationship with the other party's IdP. On receiving a SDP call offer or answer containing an Identity Assertion and the associated IdP domain-name, the User-Agent downloads the IdP Proxy from the specified location. It then calls its verifyAssertion function. If successful, the IdP Proxy returns the key fingerprint and the user identifier, for instance an email address. This bounds the key used on the media channel to the identity authenticated by the verifying IdP and ensures that no MitM attack is being set. The WebRTC Identity Architecture is shown in Figure 1b.

Richard L. Barnes and Martin Thomson gives a clear overview of the WebRTC security architecture in their 2014 IEEE article [7]. They describe the trust model on which the specification is based and explain why the CS may be considered as an adversary in this trust model. Their paper gives a good understanding, on how authenticated WebRTC communication are protected against network and signaling attack. Considering the CS to be untrusted, and at the same time relying on the same CS to offer protection through the IdP Proxy

mechanism, may be considered quite restrictive. Besides, the question of trust in the other party IdP is only addressed from the perspective of verifying the IdP's origin. The strength of the authentication process is not considered, and it is left to the user to decide whether to trust the received identity by using an address book.

3 Implementing the WebRTC Identity Specification

To implement the WebRTC identity specification, we develop a simple WebRTC service[7] offering communication for two users in a single room. The session signaling is done over WebSockets and through the communication server. A client code in Javascript manages the call session. Tests are conducted on Firefox version 50.1.0.

We implement IdP Proxies in conformance with the WebRTC specification. These proxies are added to two OIDC servers: a reference implementation by Nat Sakimura[8] and an implementation in NodeJS[9].

3.1 IdP Proxy implementation

In its simplest form, our IdP Proxy uses a simple REST API on the IdP server, matching the *generateAssertion* and *validateAssertion* functions. The GET request to the *generateAssertion* endpoint use the session cookie to prove that the user has been authenticated. The content parameter is passed in the request. If the user does not have an active session, the IdP login URL is returned by the IdP Proxy. A successful login following this URL is signaled to the CS client page by a "LOGINDONE" message from the login page. The *generateAssertion* returns an opaque token as a string. This token can then be used on the *validateAssertion* endpoint to retrieve the user identity and the associated content parameter, i. e. the media key fingerprint declared by the user.

The resulting IdP Proxy is a small piece of Javascript that can easily be deployed. And could even be provided by CS if an interoperability need arises.

3.2 IdP Proxy with OpenID Connect

To map the WebRTC Identity architecture to our OIDC server implementations, we consider the IdP Proxy to be a OIDC client using the *implicit flow*. Our implementation however requires additional modifications to the specification. As the *implicit flow* is used, and the URL fragment response is inaccessible from a Fetch API request[10], we request the ID Token to be returned in the response body, using the `response_mode` parameter. Additionally, we introduce

[7] https://github.com/Sparika/ACOR_SDP
[8] https://github.com/reTHINK-project/dev-IdPServer-phpOIDC
[9] https://github.com/reTHINK-project/dev-IdPServer
[10] https://developer.mozilla.org/en/docs/Web/API/Fetch_API

a new `rtcsdp` claim, to have the media key fingerprint included in the ID Token. Finally, the `redirect_uri` of the request is set to a particular page, sending the "LOGINDONE" message to the CS client page.

OpenID Connect is based on OAuth2, an authorization protocol. As such the concept of user consent is central to the protocol. However, in the user-to-user authentication use case, it is not clear who is the intended *audience* for the ID Token. As our IdP Proxy acts as the OIDC client, it is the effective *audience*. In practice, the identity assertion, i. e. the ID Token, is visible to each CS on the signaling path, and is ultimately verified by the peer's user-agent. The user should thus be asked for consent to share the ID Token to any of these actors. However, these are not all known in advance, and by design, any actor could instantiate an IdP Proxy to validate and decode the identity assertion.

Actually, without appropriate protection, a web page could look for user's identity assertion without triggering any warning. This could reveal user identities, or at least existing user accounts on IdPs. For instance, given a list of WebRTC compatible IdPs, running the following script would reveal identity assertion from unprotected IdPs with active sessions.

```
IdPArray.forEach(function(idp){
    var pc = new RTCPeerConnection()
    pc.setIdentityProvider(idp.domain, idp.proxy)
    pc.getIdentityAssertion()
    .then(res => {alert('Got your ID token: '+res)})
})
```

This is a major issue, and implementors should take appropriate measure to protect their interfaces against such vulnerabilities. Protections could consist in requesting user consent for every request, limiting the access to known web page origin, or managing user consent and authorization based on web page origin.

In conclusion, although OIDC offers an existing framework to generate and validate identity assertion, several modifications to existing implementations are still required in order to support WebRTC IdP Proxy peer-to-peer authentication. The resulting solution is more complex than a simple cookie based IdP Proxy, but would also easily handle authorization flow to check user consent.

4 Going Further: Trust Issue

Although the WebRTC identity specification allows to bind the media session to a validated peer identity, it does not offer a clear measure of trust to the user. In particular, a user may ask himself the following questions: should I trust my peer's IdP and what is the strength of my peer's authentication? In order to require a particular trust level, it would be necessary to act on the communication setup, i. e. conduct negotiation on the peer authentication process.

4.1 SDP negotiation

Two parameters could be negotiated: the other-party IdP origin and the authentication level [8]. To convey these requests we define two parameters. The Authentication Class Request (ACR): `List<ACRValue>`, a list, ordered by preference, of accepted authentication class value. And the Origin Request (OR): `List<Origin>`, a list, ordered by preference, of accepted IdP's origins.

WebRTC conveys identity assertions as SDP session level attributes [4]. Extension to the attribute are possible, but none are defined by the specification. Due to the identity attribute grammar, identity assertion would be mandatory to negotiate identity parameters.

We instead propose to define a new type of SDP session-level attribute to negotiate these identity parameters. The Authentication Class and Origin Request (ACOR) SDP attribute defines a list of accepted authentication class and IdP domain for the other peer identity.

- `a=acor:LIST<ACRValue> ; List<Origin>`

In SDP, a negotiation is a sequence of offer and answer exchanges, with an offer always followed by an answer. To accept the requested ACOR attribute, a peer must thus reply a SDP message with a compatible identity assertion.

4.2 Implementation

We implemented our solution[11] to negotiate identity parameters over SDP exchange. SDP messages are returned by the createOffer and createAnswer functions offered by the PeerConnection object. Once generated, the client code appends an ACOR attribute to the generated SDP offer or answer. The SDP is then sent to the other peer's client. On receiving a message, the client code reads the requested ACOR attribute. It also verifies that the received peer-identity follows the ACOR it previously requested.

The resulting negotiation solution is implemented in under 100 Javascript code lines, for a very simple client. Renegotiation allows both clients to make new offer once the session has been established. For instance, this allows to ask to an anonymous user to authenticate itself. We however identify some important limitations, mostly due to the specifications.

The *generateAssertion* function from the IdP Proxy has for parameters *contents*, *origin*, and *usernameHint*. Request for a particular authentication class to the IdP is not defined. We use the *usernameHint* parameters to pass ACR parameters to the IdP. The IdP is modified accordingly to understand this parameter. However, the browser generated WebRTC *IdentityValidationResult* do not represent the ACR. It is thus impossible for the validating IdP Proxy to return a certified ACR value to the client. The client could directly read the identity assertion contained by the SDP message, but this solution would loose the benefits of the WebRTC identity abstraction.

[11] Available with sequence diagrams at https://github.com/Sparika/ACOR_SDP

It also appears impossible to change identity at call runtime or use multiples identities simultaneously. The WebRTC specification states that if the PeerConnection object has *"previously authenticated the identity of the peer [...], then this also establishes a target peer identity. The target peer identity cannot be changed once set"* [3]. Our tests demonstrate that modifying the remote peer identity effectively close the connection. Once a first identity has been set, it cannot be changed. If this is an issue and if several IdPs are available, the client should wait to receive an ACOR request from the peer before setting an IdP.

In the end, we are able to establish two anonymous sessions and then request the other peer to authenticate with a particular identity domain. We are however unable to control the strength of the authentication. In addition, our modified SDP messages are effectively ignored by other services and by the user-agent. As a result, interoperability with other services should not be compromised by this new attribute.

5 Discussion and Future Work

Identity parameters negotiation We evaluate the possibility to deploy negotiation over ACR and IdP's origin with current WebRTC specifications. Our conclusion shows that it is not possible to request ACR to IdP Proxy when calling the *generateAssertion* function. As a result, the specification would need to be updated to support ACR negotiation. In particular: the *generateAssertion* function could be extended to accept additional parameters, and the *IdentityValidationResult* could be left open to extensions.

Recommendation source We implement our identity parameters negotiation solution on the CS side as it was the simplest solution. As identity negotiation is most useful in scenarios of inter-operable communication services, such services could be acting as the identity recommendation source. This may seems to contradict the WebRTC trust model with untrusted CS. However, in the interoperable scenario, we may want to relax the trust model and consider that a CS may be trusted by its own user. In this situation a CS could be well-suited to provide recommendation and evaluation of the other-peer's authentication.

Formalizing the trust model We are working on a model of the full WebRTC communication setup, from a security configuration point of view. We use security configuration inputs from the three communication setup paths described in Figure 1a. Knowing and negotiating the other peer's authentication strength and IdP, as presented in Section 4, is a first step towards building an explicit trust model for the identity path. User-agents could use this model to display a security indicator for the overall WebRTC communication.

6 Conclusion

Decoupling authentication from the signaling process by providing an explicit identity assertion to the user is an interesting new paradigm. Especially true for the design of interoperable communication services on the web.

In this paper, we described our implementation of the WebRTC identity architecture and the issues we encountered in our OpenID Connect implementation. Although WebRTC identity specification provides users with an initial trust level in their peer's authentication, it does not offer a clear measure of trust. To solve this issue we propose an extension to WebRTC standard based on a new SDP attribute for users to negotiate over the authentication of the other party during the signaling process. Our implementation highlights that this solution could be readily used by communication services, but it also demonstrates actual limit of the current specifications in some scenarios, such as a multiple identities for a user, e. g. personal and professional identities.

Users are not best fitted to analyze at runtime a complex security configuration, such as a WebRTC setup configuration. They need help from a recommendation source that would conduct the negotiation and notify them of the current trust and security level. This recommendation source would use actual session parameters at runtime to evaluate trust in the communication. In our future work we will build such trust model.

Acknowledgment

This work has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 645342, project reTHINK.

References

1. L. Boursas and V. A. Danciu, "Dynamic inter-organizational cooperation setup in circle-of-trust environments," in *NOMS 2008-2008 IEEE Network Operations and Management Symposium*. IEEE, 2008, pp. 113–120.
2. A. Jøsang, J. Fabre, B. Hay, J. Dalziel, and S. Pope, "Trust requirements in identity management," in *Proceedings of the 2005 Australasian workshop on Grid computing and e-research-Volume 44*. Australian Computer Society, Inc., 2005, pp. 99–108.
3. C. Jennings, A. Narayanan, B. Aboba, A. Bergkvist, and D. Burnett, "WebRTC 1.0: Real-time communication between browsers," W3C, Working Draft, Mar. 2017.
4. E. Rescorla, "WebRTC Security Architecture," IETF Secretariat, Internet-Draft draft-ietf-rtcweb-security-arch-12, June 2016.
5. N. Sakimura, J. Bradley, M. Jones, B. de Medeiros, and C. Mortimore, "OpenID connect core 1.0," The OpenID Foundation, OpenID Specification, 2014, http://openid.net/specs/openid-connect-core-1_0.html.
6. M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," Network Working Group, RFC 4566, July 2006.
7. R. L. Barnes and M. Thomson, "Browser-to-browser security assurances for WebRTC," *IEEE Internet Computing*, vol. 18, no. 6, pp. 11–17, 2014.
8. "ISO/IEC 29115:2013 - Information technology – Security techniques – Entity authentication assurance framework."