MyWebRTC, a Free Do-It-Yourself Kit for Secure Real-Time Internet-Communication

Tolkiehn G., Lebedev M., Makariti A.
Business Computing
TH Wildau
Wildau, Germany

Abstract— **myWebRTC provides a basis for easy-to-use, cross-platform, low-cost, secure, privately owned solutions for real-time communication over the web. This may be particularly useful in confidential healthcare communication, but as well in many other branches and applications. WebRTC-based services may comprise chat, video/audio-calls, file-transfer, desktop-sharing and conferencing. Main features and implementation details of the prototype are outlined. All sources of the solution are available on GitHub. A demo-implementation is publicly available online. The authors encourage derivation of individual do-it-yourself systems as well as development of IT-service-oriented business models providing professional solutions to enterprises and authorities plus extensions for interoperability with other solutions.**

Keywords— WebRTC; open source; secure real-time communication; business model; node.js

I. INTRODUCTION AND MOTIVATION

At the latest after the shock of the revelation of the mass-surveillance by the NSA in 2013, everybody worries about massive data collections by third parties. Massive data collection is performed, however, not only by national intelligence agencies, but also by Internet-Access- and Over-the-Top service providers as part of their business models. Today therefore, everybody is alarmed about possible industrial espionage and other breaches of confidentiality in internet communication.

As one consequence, enterprises and public authorities tend to ban real time web services. This, however, reduces user-friendliness and interferes with trends like "Bring Your Own Device" and Homeworking.

In this setting and after more than 25 years of video communication solutions with the ITU-T and IETF families of standards as well as with several more or less proprietary solutions and taking into account the vast number of users that Skype, WhatsApp and other real-time services have gained, the initiatives of Google [1], the World Wide Web Consortium (W3C) [2] and the Internet Engineering Task Force (IETF) [3] to implement secure browser-based peer-o-peer real time communication over the web appeared attractive. In particular the concepts of standardized, browser based peer-to-peer media data transfer including a variety of standard codecs and strong end-to-end encryption (DTLS-SRTP [4]) seemed promising.

Also the options for proven technology and standards for the signaling plane (including handling of clients behind firewalls), the ease of implementation using HTML5 and JavaScript with a set of APIs and the future perspective of support by all relevant browsers (presently Mozilla Firefox, Google Chrome and Opera, Microsoft and Apple soon to come) seemed favorable.

Our interest in WebRTC was further encouraged by a medium sized German enterprise, Estos GmbH, who already in 2013 expressed their interest in cooperation on the subject plus by the ever-growing importance of teleworking in many fields, including e-health and university teaching.

II. REQUIREMENTS, AIMS AND METHOD OF OPERATION

Having had a look at a number of web services using WebRTC available already in 2013 (e.g. bistri.me, apprtc.appspot.com, palava.tv, go.estos.de, projectansible [5]) and regarding the perspective of Skype also moving to WebRTC, we thought about an alternative approach. After discussion we defined a number of requirements for a license-free open source WebRTC-based solution aiming at usage in enterprises, administrations and also universities as on-premise systems owned by the users' own organization.

The requirements were defined in the style of agile development (epics and user stories). A Scrum-like approach appeared favorable for our work, although the organizational working conditions in a students' project are not ideal (part-time work with only 20% of full-time and only during the lecture periods with ten weeks of semester break). However, there was affirmative experience from earlier projects. One main advantage of agile methods is the flexibility in adapting technical innovation, moving targets, uncertainty, and technical difficulties. The Scrum roles concept of product-owner, team and scrum-master is also beneficial for largely self-determined development as appropriate for semi-professional master course students.

Regarding the limitations in time (two semesters) and developers' work-hours budget for development (480 hours of graduate students) in combination with the necessary learning curve, it was obvious, that we could not expect to be able to provide a fully functional solution fit for productive operation.

978-1-5090-0234-4/15/$31.00 2015 IEEE

The development goal was therefore restricted to a working demonstrator solution with only some of the appreciable features, lacking sophisticated usability, but interesting enough for demonstration of the idea and presentation on the CeBIT fair in Hannover, in March, 2015, that is, within exactly one year after the start of work.

The product-backlog thus contained epics like

- "As a responsible manager, I want to provide inexpensive, platform-independent and secure real time communication for the members of my organization and their communication partners"
- "As a user, I want to easily set up and close secure real-time connections or conferences with my business contacts on devices of my choice"
- "As a system administrator I want a slim, transparent, standards-based system for easy and secure implementation, operation and administration"

In the process of decomposition of the epics into user stories (some of them in relation to special activities, like remote medical care or tele-teaching) and tasks, a number of choices had to be made. Our priorities for the communication options supported by WebRTC were defined in the order

- Chat,
- Video/Audio Calls
- File-Transfer,
- Desktop-Sharing
- Multi-Conferencing

In parallel, a number of architectural and procedural choices had to be made. Our strategy here was to be on the one hand pragmatic and efficient, but at the same time to remain as open as possible for alternative architectural choices. We decided to use node.js as webserver, PHP and MySQL for the user management. Not being familiar with node.js we chose not to use PHP on node.js, but to implement an additional Apache server, which is probably not the most effective implementation.

For the system platform there is the choice between different Microsoft and Linux versions as all components used are platform-independent. We chose Microsoft Server 2008 for the reference implementation.

As our hardware platform we chose a stand-alone desktop PC system in our lab with remote access as opposed to using a virtual machine operated by the IT-department.

For the WebRTC server application software (HTML5 and server side JavaScript) we used and adapted several code examples from the publication of Johnston and Burnett [6], which proved to be very helpful also in understanding the APIs.

User management and administration (see also second epic above) is not covered by the WebRTC standardization. In productive systems, an adaptation to or import of parts of the user-administration already present in every organization will probably be the method of choice. But this was defined to be out scope for our project. Here, this function was intentionally kept simple and the development of solutions by experienced teams is strongly encouraged.

Also, solutions for interoperability with other real-time communication systems seem very interesting, but were not yet considered in this first approach.

Most WebRTC-based business models so far offer free basic communication services but do nontransparent commercial evaluation and sales of user meta-data (like e.g. google Hangouts) or even user content (like Skype). They may additionally charge fees for gateway connection services or premium communication services.

The business model approach for myWebRTC is different. It does not rely on commercialization of any kind of user data. Business models for myWebRTC may instead base upon paid or "do-it-yourself" system integration, provision and system-administration or -operation services. Solutions may consist of license-free open source software alone or in combination or connection with licensed software components.

III. FUNCTIONALITY OF THE DEMO-SYSTEM

The demonstrator system implemented for public use contains the functions "chat" and "video/audio-communication". File-transfer was not implemented due to lack of time and lower priority. Desktop-sharing was not implemented due to an irritating turn in the concept of google (they surprisingly announced to use a plug-in for this feature in Chrome). We did not want to go for that. In fact the whole community objected to this obvious violation of the overall WebRTC strategy (no client software, no plug-ins, no add-ons). After some time google dropped the idea again, but at that point we could not resume work on this feature.

User-management is done by a MySQL database using PHP. The data base contains the standing data of the registered users and a number of state variables representing the current status of the user (like online/offline). Out of development performance reasons, the team decided against using PHP in node.js and for a second web server (Apache) for this task.

The basic concept here is, to create one web page per registered user and to open this page on login, say of user A. User A's status is thereby changed to "online" and the browser will ask permission of the user to access camera and microphone. If declined or not present, only chat will be available. A will see the list of other registered users and their online-status. This feature was specified imitating the well-recognized GUI of Skype. User A may then attempt to call one of the users with status "online", say B, by clicking on the "call" button.

Signaling of call attempts is done by polling the data base. On receiving a call attempt, user B will be signaled by a sound and an input field popping up, where he may choose to accept or decline the connection request. There still are, however, some flaws in this mechanism of our demo solution, so that up

to now, neither the users' online status nor call attempts are properly notified.

Thus, for experiencing the chat and video-call functions, users are up to now restricted to a more primitive approach using the button "Demo Version", where the user management is not engaged.

Here, the browser will ask permission to access camera and microphone. The users may then enter an arbitrary code into the input field "key". If the codes entered by two users at approximately the same time (timeout is 100 seconds) match, both will be asked to accept the connection and on acceptance be connected. For another connection, a new code has to be used. Old codes will be marked "used" for a day.

This very simple form of connecting works on any device using the specified browsers. There is, however, one exception for iPad. Here, video/audio with Chrome or Firefox apps will not work, the reason being that iPad so far does not permit the browser-Apps of the competitors to access the device's camera or microphone data. These browser apps will therefore not ask for such permission. So far however, this exception is not handled as refusal of audio and video access by the user, so that on iPad chat will not be available either.

Another restriction, related to NAPT of mobile ISPs will be mentioned below. Although this primitive functionality is hardly useful for everyday work, connection is so surprisingly universal, simple and effective for people familiar with other video-call systems, that most visitors of our CeBIT exhibit were deeply impressed, in particular about the fact that they could not only connect to one of our laptops in the exhibit, but also between two visitors.

The sources of myWebRTC have been available on GitHub [7] for public free use since May 2015.

IV. Clients behind firewalls and NAT-routers

While media data are preferably passed directly from browser to browser once a session is established, WebRTC clients need an intermediate server for metadata signaling and session management. Here WebRTC allows for different signaling protocols like e.g. SIP or Jingle as outlined in the JavaScript Session Establishment Protocol JSEP [8]. One important issue for web-based peer-to-peer communication today, well known already from IP-telephony, is that still most clients use IPv4 and the majority of client systems reside in local area networks behind firewalls and routers doing Network Address Translation (NAT). These systems cannot be accessed using their local IP-addresses because these addresses are non-routable. The WebRTC standard comprises the option of using Interactive Connectivity Establishment (ICE [9]) servers to overcome this complication.

Here, two different functions are used. One is, to use a server which supports Session Traversal Utilities for NAT (STUN [10]). Such a server will on request deliver the routable IP-address and port number by which the client accesses the internet. These external client data then may be transferred to the session management subsystem for connection establishment and used for peer-to-peer transmission of media data.

A private STUN server may easily be provided by the owner of a myWebRTC-system. It may run on the same machine as node.js and user administration. Still, for our demo system, we used one of the publicly available STUN servers.

Sometimes, however, this simple procedure will not be effective to allow connection setup and peer-to-peer data transmission, because firewalls do not allow pinholing at all or for extended time periods or out of other reasons. In such cases a server for Traversal Using Relay NAT (TURN [11]) may be employed technically avoiding peer-to-peer communication. Also a TURN server may be implemented and employed by the owner of a myWebRTC solution, but is not included.

While there are also public TURN servers available which automatically offer TURN for relaying media data in case the STUN approach alone fails, we did not include one of these in the myWebRTC code either. The reason is that relaying media data through an external server constitutes a severe security hazard and also an additional central point of failure.

The drawback of this precaution is that connection establishment will fail in some cases. On WLAN or wired internet this scarcely happened (depending on the firewall used for the client), but clients using mobile carriers for internet access will regularly fail to connect without using a relay server. For relaying however, sufficient internet bandwidth and also system performance for all relayed connections has to be provided.

However, even the use of only a public STUN server discloses all of the metadata of your communications to the operator of this server. Therefore, our recommendation for future implementations of myWebRTC-like solutions is to set up your own servers for ICE, including relaying, in a controlled environment in addition to the servers for WebRTC and user administration.

V. Conclusion and perspective

We think that our demo application may be helpful for organizations to establish their own experience and implement their own custom solution for secure communication (do-it-yourself approach) within a clear range of development and test effort.

It may also encourage IT-companies to establish new offers of solutions (not services) for secure real time communication based on free open source components, but not necessarily being completely open source and free of licenses. One such product development is presently under way in a German small or medium sized enterprise (SME).

myWebRTC as an open source starting point seems also well entitled for further development of both additional functionality and better design and quality. Such developments may also be performed in co-operations of enterprises with IT-specialists and students from universities as presently being established in our IHSITOP [12] project.

Except debugging and redesigning the user management and finding suitable ways to connect to existing user management systems, two issues of particular interest for myWebRTC will certainly be the accomplishment of interoperability with

- other WebRTC systems
- telephony and other real-time communication systems and services already established in the market.

A number of different ideas have already been discussed in this area [13], [14]. Several German SMEs have so far expressed their interest in co-operation on these issues.

Another area of general concern and future research, which, however, cannot be outlined here in detail, will obviously be security issues of WebRTC solutions in general and in particular of the relatively new node.js technology.

WebRTC technology appears well prepared to impede confidentiality breaches of the media streams. However, it does not provide measures against other general security issues, like unwanted collection of meta-data by third parties, hacking or Denial-of-Service (DoS) attacks are another story.

Unlike in private or general business communication, there are real time applications like in health-care or in search and rescue, where communication breakdowns may cause safety risks. Service availability will therefore be of special importance in these fields, not only for commercial reasons. Reliable Internet access with sufficient quality of service and prevention of DoS and other attacks on service availability will be issues of special interest in these application fields.

ACKNOWLEDGEMENT

The authors would like to thank Susanne Koczoh for her technical and administrational support during the whole project.

References

[1] http://webrtc.org

[2] http://www.w3.org/TR/webrtc/

[3] https://tools.ietf.org/html/draft-ietf-rtcweb-overview-14

[4] IETF RFC 5764: https://tools.ietf.org/html/rfc5764

[5] In 2015 renamed to https://www.circuit.com

[6] Johnston, A.B., Burnett D.C., WebRTC: APIs and Protocols of the HTML5 Real-Time Web, Digital Codex LLC, St. Louis, 2014.

[7] http://github.com/mywebrtc/code

[8] http://tools.ietf.org/html/draft-ietf-rtcweb-jsep-11

[9] IETF RFC 2545: https://tools.ietf.org/html/rfc5245

[10] IETF RFC 5389: https://tools.ietf.org/html/rfc5389

[11] IETF RFC 5766: https://tools.ietf.org/html/rfc5766

[12] https://ostpcen.wordpress.com/tempus-project-ihsitop/

[13] http://www.webrtcworld.com/topics/from-the-experts/articles/383883-sip-universal-restricted-with-webrtc.htm

[14] http://matrix.org/blog/wp-content/uploads/2014/11/2014-11-03-Matrix_Missing-Link_IOT.pdf