A Project/Dissertation Review-1 Report

on

Serverless chat and Audio/Video Calling Android app

Submitted in partial fulfilment of the
requirement for the award of the degree of

B.tech/ CSE

Under The Supervision of
Mr. V Gokul Rajan
Professor

Submitted By -

Saumya Kumar (18SCSE1010138)

Ashish Kumar (18SCSE1010474)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
OCTOBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the thesis/project/dissertation, entitled **"Server less chat and audio/video calling android application"** in partial fulfilment of the requirements for the award of the Bachelor of Technology submitted in the School of Computing Science and Engineering of Galgotias University, Greater Noida, is an original work carried out during the period of November, 2021 to May 2022, under the supervision of Mr V Gokul Rajan Assistant Professor, Department of Computer Science and Engineering of School of Computing Science and Engineering , Galgotias University, Greater Noida

The matter presented in the thesis/project/dissertation has not been submitted by me/us for the award of any other degree of this or any other places.

Saumya Kumar – 18SCSE1010138

Ashish Kumar – 18SCSE1010474

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Mr V Gokul Rajan

Assistant Professor

Acknowledgement

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **Saumya Kumar – 18SCSE1010138** & **Ashish Kumar – 18SCSE1010474** has been held on _____ and his/her work is recommended for the award of Bachelor of Technology.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: November, 2013

Place: Greater Noida

Abstract

Chat applications have become one of the most important and popular applications on smartphones. It has the capability of exchange text messages, images and files which it cost free for the users to communicate with each other such as Skype.

Skype however, makes use of centralized server for user registration, login and buddy list. Indeed, this idea could be disastrous in the event of a compromise. In this Project we are developing a chat application that is based on pure peer-to-peer architecture that totally rid of centralized or third-party elements.

All messages must be protected. The aim of the project is to create a simple, easy-to-use, persistent real-time, serverless chat application that provides End-to-End security that let safely exchange private information with each other without worrying about data. The goal of this project is to simplify and demonstrate the power, security/confidentiality and persistency of serverless data exchanges in mobile phones.

Using the power of WebRTC, it is Google's open-source base for peer communication in the browser, using QUIC protocol which provides a way to establish data channels between 2 peers supporting any type of data, including real-time audio and video.

The future of WebRTC is associated with the emergence of technology in new markets. Furthermore, as a long as WebRTC is a W3C standard, anybody can influence its development, which implies great prospects.

Table of Contents

List of Tables

List of Figures

Chapter 1

1.1 Introduction

The aim of this project is to use WebRTC technologies to create a viable real-time messaging tool for users who want privacy and security. Due to the increase in internet bandwidth, which allows for high-speed streaming, video has become an essential medium for communication in recent years. Previously, the footage was recorded and broadcast in analogue format.

The advancement of computers and digital integrated circuits has resulted in the digitalization of video, which has resulted in a revolution in video transmission and compression. Streaming refers to the technique of transmitting material over the Internet by encoding it into a variety of decodable forms. The stream is considered a "Live" stream when it is broadcast while material is being generated.

Live Video Streaming is a type of video streaming in which an electronic message is transmitted in real time over a local area network (LAN) or the Internet so that the video and/or audio from the transmitter source can be heard and seen on the receiver side via personal computers, smart phones, and mobile devices, among other things. One-way communication (streaming) or two-way communication (video/audio chat or video conference) are examples of real-time media communication (audio, video) between various client devices.

Multiple endpoints broadcast and receive video and audio during live media video conferences. Media streaming, in particular, necessitates the installation of independent streaming servers, the installation of a suitable standalone application on the client side, and support for streaming protocols that regulate the transmission of streamed packets. In the case of conferencing and chatting, there is also a requirement for the mediation of a session manager between the clients and the implementation of the appropriate session protocols. In terms of real-time online communication, HTTP has remained the media streaming myth method till now. Receiving and processing media streamed over the web can only be accomplished by installing the appropriate third-party software (browser plug-in) to receive and process the material streamed from the server.

Finally, there is a need for media players that provide browser plugins that allow audio and video streams to be streamed over the internet. To integrate video conferencing with a flexible android-based app that goes beyond a simple face-to-face configuration, the video conference must also be conducted in an android app context. This means restrictions in terms of user interface (UI) layout (e.g., window transparency) and accessible video processing application programming interfaces as compared to native groupware programmes (APIs).

Web Real-Time Communication (WebRTC) is a set of communication protocols and APIs that allow web browsers to communicate in real time with one another. Browsers are now able to break away from the traditional client-server architecture thanks to WebRTC's P2P capabilities. The benefit of this transition is that WebRTC APIs are the same independent of the underlying browser, operating system, or other factors, and are available on a wide range of platforms, including mobile devices.

The project's goal is to create an audio/video chat android app that allows people from all over the world to participate in real-time secure chatting session. This suggested environment is in the security infrastructure that provides safe streaming to consumers in order to avoid security concerns.

Chapter 2

2.1 Literature Survey

Authors	Origin	Purpose	Type of source	Summary points
Ammar H. Ali, Ali Makki Sagheer (2017)	JSEIS	To understand the design of secure chatting application	Research Paper	The algorithms used for encrypting voice and image message is RC4 which is one of the fastest and secure encryption techniques, but can be cracked by RC4 attacks which uses vulnerabilities in RC4 key streams.[1]
Ms. Swara Pampatwar, Vinisha Kalyani, Prachi Shamdasani, Urja Ramwani (2020)	Annals of RSCB	To understand the implementation of multilayer of Data encryption and decryption used to transmit data in social networking application.	Research Paper	Data confidentiality, Authentication, Integrity, Non-repudiation, Access control, and Availability are the most imperative security services in the security criteria that ought to be considered in secure applications and frameworks. Not with standing, there is no arrangement for such security services in the mobile chat systems.[2]
Ben Feher, Lior Sidi, Asaf Shabtai, Rami Puzis (2016)	arXiv	To understand the security of WebRTC	Research Paper	WebRTC security measures can be divided into three main groups: Client, Server and Network, As WebRTC is designed as Peer-to-Peer communication interface, the server's duty is the mediation of clients Roles performed by the server are implemented using STURN and IdP. WebRTC specifications explicitly forbids clear RTP [3] and enforces using a secure encrypted version of it called SRTP.
K wok-Fai Ng, Man-Yan Ching, Yang Liu, Tao Cai, Li, Wu Chou	IJFCC	To understand the multi-Party Video Conference	Research Paper	When the number of participants increases, the bandwidth and CPU requirements have become a

(2014)		with WebRTC		serious issue in the push-based mesh network. Using P2P-MCU approach for multi-party video conferencing that efficiently supports both ordinary smart mobile phones and PCs. In this approach, a MCU module is integrated into the browser to mix and transcode the video & audio streams in real time. This solution reduces 64% CPU uses and 35% of bandwidth consumption for each participant compared to the mesh network.[4]
Mohamad Afendee, Abdullah Muhammed, Mustafa Man, (2015)	Journal of Computer Science	To understand the architecture of pure peer-to-peer application	Research Paper	We can create a self-secured individual user database without any involvement of third party. Security element was built within user registration, user login, message exchange and database storage. Can offer different authentication and encryption algorithm to be chosen by users. Whenever key size should be allowed to be negotiated manually by the uses.

Chapter 3

Working of Project

WebRTC consists of two distinct standards efforts and components:

1. Protocol specifications developed in the IETF RTCweb working group (WG).

2. JavaScript (ECMAScript) Application Interfaces (API) that control components on the client side, specifications are called WebRTC and are standardised in the W3C WebRTC working group.

While this is a somewhat artificial split, we nonetheless follow it here since the protocol/API split is understood by the WebRTC community and is followed by the extensive W3C documentation and the set of IETF Internet Drafts scheduled to become the protocol specification RFCs in due course.

1.1 WebRTC Protocols

The Web Server delivers content to the browser that allows the browser to initiate the data or communication channel to another browser. The Server controls the communication between the browsers via JavaScript modules by initiating the connection and providing all WebRTC parameters to the browsers. Of course, since the call setup is handled via JavaScript, this can include user interaction, but ultimately the web server controls which scripts are executed. The thinking behind WebRTC call setup has been to fully specify the media plane, but to leave the signalling plane up to the application as much as possible. The rationale is that different applications may prefer to use different protocols, such as the existing SIP or Jingle [2] call signalling protocols, or something custom to the particular application, perhaps for a novel use case. In any case, the crucial information that needs to be exchanged is the multimedia session description which specifies the necessary transport and media configuration information necessary to establish and maintain the media plane.

It is important to note that, except for Network Address Traversal (NAT), conferencing, or similar reasons, the media path goes directly between the browsers, and so, in order to achieve interoperability, it has to conform to the specifications of the WebRTC protocol suite. In contrast, the signalling path (“JavaScript over HTTP with TLS”) goes via servers that can modify the signals as needed.

According to the WebRTC protocol overview [3] the protocol part requires:

1. Connection management: Setting up connections, agreeing on data formats (codecs), changing data formats during the duration of a call.

2. Data transport: TCP, UDP and the means to set up connections between entities while meeting the agreed upon security requirements, as well as the functions for deciding when to send data: Congestion management, bandwidth estimation and so on.
3. Data framing: RTP (RFC 3550 [73]) and SRTP (RFC 3711 [8]) and other data formats that serve as containers, and their functions for data confidentiality and integrity.

4. Data formats: Codec specifications, format specifications and functionality specifications for the data passed between systems. Audio and video codecs, as well as formats for data and document sharing, belong in this category. In order to make use of data formats, a way to describe them that is a session description is needed.

Within this architecture, many different paths can be used for the signalling between one or more Web servers and the browsers. While SIP [6] and XMPP/Jingle [7, 8, and 9] have been most common protocols for this signalling path to date, customised JavaScript is expected to be the most common in future. However, even if SIP and Jingle are somewhat in the category of "legacy" capabilities all the vulnerabilities of those technologies will also affect the level of security achievable by WebRTC. The communication channels between a Web server and a browser are well known. They use HTTP URIs via HTTP/1.1 over TCP or HTTPs Uris HTTP/1.1 over TLS (TLS 1.0 or TLS 1.2). However, with the on-going development of HTTP/2.0 new combinations will become possible, for example accessing HTTP URIs via HTTP/2.0 over TLS. The content transported is HTML4 or HTML5.

1.1.1 Connection Management

Media session parameters are described using the offer/answer mode of the Session Description Protocol (SDP) [67, 38]. While there is some dis-satisfaction with this, at least amongst some of the web community, no alternatives are under consideration. So at least WebRTC version 1.0 (if versioning is the proper metaphor) is committed to SDP, but future versions may attempt to move away from SDP. The main objection to SDP is that its string representation is hard to manipulate, and it does need to be manipulated even in some quite simple use-cases.

This adds complexity to the application/signalling layer JS code. While this is a valid criticism, it is not clear that a non-SDP solution would really be easier. However, we ought to expect the complexity associated with SDP manipulation

to enable some exploits in the wild. SDP allows servers to convey the necessary information about the communication session endpoints, codecs and some of the security mechanisms required. SDP was originally conceived as a way to describe multicast sessions carried on the Mbone, an experimental IP Multi-Cast network. A complete view of the session requires information from both participants, and agreement on parameters between them. Thus in the general case an SDP description can include many attributes such as:

• Session name and purpose

• Time(s) the session is active

• The media comprising the session

• Information to receive those media (addresses, ports, formats and so on)

As resources necessary to participate in a session may be limited, some additional information may also be desirable:

• Information about the bandwidth required by the session

• Contact information for the person responsible for the session

• The type of media (video, audio, etc.)

• The transport protocol (RTP/UDP/IP, H.320, etc.)

• The format of the media (H.261 video, MPEG video, etc.)

For an IP multicast session, the following are also conveyed:

• Multicast address for media

• Transport Port for media

This addresses and port are the destination address and destination port of the multicast stream, whether being sent, received, or both.

• Remote address for media

• Transport port for contact address

The fact that SDP descriptions can be quite complex and can be bundled together and encompass kinds of session that are not expected to be common in WebRTC (e.g., recordings of TV broadcasts) and that SDP descriptions are new to the web, mean we ought focus on the security aspects of SDP to explore for potential vulnerabilities. The JavaScript Session Establishment Protocol (JSEP) allows for

browser control of SDP description handling. Essentially this interface provides a way to handle the offer/answer model in a way that suits browsers (e.g., considering page reloads) but can also work for other applications. JSEP mainly consists in methods for handling local and remote session descriptions as negotiated by some signalling mechanism, plus a way to interact with the ICE (RFC 5245) state machine.

1.1.2 Data transport

Another challenge for WebRTC lies in the fact that most browsers are either behind a firewall or within a locally managed network using Network Address Translation (NAT). Internet traffic from a typical internal IP network (using private IP address blocks such as 192.168/16 or 10/8) is funnelled via an external gateway (firewall and/or NAT unit) with a globally routable IPv4 address to the "outside". Frequently such firewalls block most unsolicited UDP traffic from outside and NAT makes the direct addressing of a peer browser more difficult. To handle such issues, WebRTC uses Interactive Connectivity Establishment (ICE).

ICE is a protocol for NAT traversal for (initially, UDP-based) multimedia sessions established with the offer/answer model. ICE makes use of the Session Traversal Utilities for NAT (STUN) protocol and its extension, Traversal Using Relay NAT (TURN). ICE can be used by any protocol utilizing the offer/answer model, including WebRTC. Consequently, WebRTC requires the implementation of TURN as defined in RFC 5766. If IPv6 traffic is supported, as will be common, RFC 6156 [10] must be supported as well as STUN, RFC 5389. All this allows connection either to the servers or between the browser machines and to establish a session. Since ICE, STUN and TURN are frequently confusing, we re-iterate their functions. STUN is a server operated service that allows nodes behind NAT boxes to find out their public IP addresses and thus assists in establishing connectivity. TURN goes a step further and provides a server through which packets may be exchanged, which enables two nodes that are both behind firewall/NAT boxes to communicate. Both STUN and TURN require a form of authentication between the possibly NATted node and the server. That authentication is based on shared secrets (i.e., currently, passwords), though a new IETF working group ("tram"3) has been formed recently to try improve on this.

And finally ICE is an algorithm used to establish transport layer connectivity between WebRTC endpoints. ICE essentially tells a node how to create a list of STUN and TURN candidate endpoint addresses and then how to prioritise those so that both sides of a communication arrive at common endpoints. For WebRTC the browser is the canonical ICE

endpoint. For the media channel between browsers, WebRTC implementations must also support SCTP over DTLS-SRTP. A very brief explanation of these protocols may be useful: The Stream Control Transmission Protocol (SCTP) is another transport layer, similar to the much better known TCP or UDP that has been selected for WebRTC. DTLS is the Datagram Transport Layer Security protocol, which securely establishes shared cryptographic keys between two hosts so that those keys can be used to secure datagrams. RTP is, of course, the Real Time Protocol, used for transferring media packets. And SRTP is a framework for securing RTP packets, which differs from other security protocols in that with such media; some lost packets can be accommodated without damage to the overall application. DTLS-SRTP as a combination means using DTLS for key establishment (the "handshake") and then using those keys with some SRTP security method.

1.1.3 Data framing and baseline security

For transport of media between the browsers, the Real-time Transport Protocol (RTP) [73] is used. RTP itself comprises two parts: the RTP data transfer protocol, and the RTP control protocol (RTCP). In order to support the video conferencing scenario, WebRTC needs support for multiple channels (or sources), thus the need for multiple synchronisation sources (SSRCs). WebRTC is notable in that it not only requires implementation of security via SRTP (RFC 3711) and specifically, DTLS-SRTP but also mandates that the secure flavour be the default to offer. This is not a requirement to use the secure flavour, but is quite close to one, and represents an "upgrade" on the usual IETF/W3C level of security requirement, which is to make security features Mandatory-To-Implement (MTI). Additionally, the SDES [11] method of keying SRTP has not been agreed as being MTI, despite some pressure for that from those most interested in conference calling. The reason to not want SDES is that that exposes the SRTP keying material to the JS code and hence the web server, whereas with DTLS-SRTP the keying material used to protect media is only (easily) available to the media endpoints (i.e., mainly browsers). Nonetheless SDES can still be implemented as an option, and probably will be by some at least. Whether or not browsers will support SDES is not yet clear, so any security analysis needs to consider the SDES option as well.

1.1.4 Data formats

WebRTC wants to allow each communications event to use the data formats that are best suited for that particular instance, where a format is supported by both sides of the connection. However, a minimum standard is greatly helpful in order to ensure that communication can be achieved. The current development is heavily in flux, mainly because of IPR issues around audio and video codecs. It seems the IETF Group has achieved some agreement around draft-ietf-rtcweb-audio-05 on the audio codecs that are mandatory to implement.

The choice fell on Opus as specified in RFC 6716 and on G.711 PCMA and PCMU with the payload format specified in section 4.5.14 of RFC 3551. During the second half of 2014, the RTCweb Group were consumed by controversy concerning the video codec, and did not manage to reach rough consensus on which codecs to make MTI. The group is somewhat equally divided over the use of H.264 from MPEG or VP8 as specified in RFC 6386 [12]. Both solutions have their merits and their burden of IPR problems. At the time of writing, the IETF WG have decided to shelve the video codec debate until September 2014 in the hope that market developments will produce a rough consensus for some MTI video codecs.

1.2 WebRTC APIs

The complex protocol machinery that was described in Section 1.1 will of course not be exposed to the developer. It serves as a framework for developers to build web applications into web pages using the HTML5 development platform. While the protocols provide the channels, the W3C WebRTC API efforts provide the necessary hooks and APIs to allow developers to use the system for applications that allow for real-time communication. The real innovative power of WebRTC becomes particularly apparent on mobile phones. The browser in a web application scenario does not have the same look and feel as our desktop browsers.

It is mainly a screen with functionality that is controlled by a browser running with little or no browser “chrome” (decoration) around it. One can try that experience by hitting the full screen button (typically F11) in a desktop browser. All surrounding window borders and decoration disappears and only the page is rendered. The HTML5 platform means that this apparently pared down environment provides all the capabilities needed for fully fledged application development. Real time communication may thus be only one part of a web application running on a mobile phone. To make the complexity manageable, we will take a step-by-step

approach and start with the browser's part in a very basic WebRTC scenario and then increase the complexity and the context.

1.2.1 The Browser interactions

As usual we start with the browser loading a web page that contains relevant information within some JS (or more properly, ECMAScript). The browser can now start a negotiation (via the web server) with another browser in order to establish a Browser-to-Browser connection. To do that it uses the channels given by WebRTC as explained in Section 1.1. While an SDP description doesn't tell the user agent how to exchange the session information, it contains the proposed setup for a session to come.

For WebRTC this includes requirements of all kinds like the presence of a microphone, the codecs to be used for the exchange, the STUN/TURN credentials required etc. Now both browsers identified by the server and controlled via the server signalling channel start to exchange rounds of offers and answers, each of which are processed via call-backs. The JS API defined by the W3C WebRTC specification describes the commands the browser can use to react to those call-backs and where to get the information to put in the response data of the call back dialog.

Once both browsers are in agreement over the transport, framing and content format to use, communication via the media path is opened between the browsers. It is important to bear in mind that the server retains full control over the communication, even if the packets flow from browser to browser directly. Before media transfer starts, the browsers need to complete the ICE negotiations, and any DTLS-SRTP exchanges. ICE [29] is how WebRTC browsers search for connectivity potentially in the face of firewalls and/or Network Address Translators (NATs) local to each browser.

These exchanges might take a number of round-trips, which can be problematic for so-called "early media," i.e., voice or video packets that are produced before the channel is ready. Handling early media with DTLS-SRTP is an on-going work item for the IETF rtcweb and other WGs4. As the audio or video communication starts, the browser has to provide the relevant information into the media path. The browser mediates the access that the media path has to the local system used. The browser API also can trigger another API called Media Capture.
This API is able to control several features of the browser machine, namely microphone and camera(s) to provide the audio and video content. In a realistic web application scenario, this goes way beyond the mere establishment of a communication. This also explains the significant

attention paid in the IETF WebRTC security drafts to topics like identity management, since the users' decisions about permissions for sharing their device's audio and video will be highly dependent on each user's perceptions of who is on the other end of a call and need to be independent for each user.

The JS contained in the page that controls the WebRTC sessions can also access and control many other resources that are important in a bilateral or multilateral audio or video communication. This includes things like accelerometer data, or ambient light information, and perhaps in future, addresses books, location, temperature and the many other sensors a smartphone may make available. It is therefore of prime interest to control how far information accessed by one part of a web application can be delivered to another part. Already today, there are social networks offering audio and video communication within their own context. But this is irretrievably entangled with their own identity management systems. Regularly, we hear that such social networking applications escape their limitations and send local data over the wires, see for example a Dutch report on WhatsApp. The WebRTC JS module specifications addressed by the W3C contains a set of object definitions and function calls for the following features:

• Browser-to-browser connections

• Browser-to-browser Data API

• Browser-to-browser Dual-Tone Multi-Frequency (DTMF) signalling

• Identity Provider Interaction

• Various extension points

The main items of note here are the Media Stream and PeerConnection objects and APIs. The former represents the stream(s) of media data involved in a call that are supported by both browsers, the negotiation of the details of which is one of the main complexities of WebRTC.

The latter (PeerConnection) provides methods for two browsers to communicate data dealing with the many problematic connectivity issues that can arise.
With these function calls a WebRTC application can be programmed using JS. This web application comes embedded in the HTML5 page from the server and can make network connections of all kinds within its security limitations that are mainly determined by the Same Origin Policy (SOP).

There is a potential discrepancy between what a web application is able to do and what a good web application ought to do. One of the hardest and most onerous achievements in the Device API Working Group was precisely to get to agreement on such Best Practices, which for WebRTC have yet to emerge. The Device API and WebRTC WGs have formed a task force to define the Media Capture and Streams capability that will mediate WebRTC access to devices and manipulation of data streams from those devices but this is still work in progress at the time of writing.

1.2.2 Core WebRTC Specifications

1. WebRTC 1.0: Real-Time Communication Between Browsers W3C Working Draft 10 September 2013

2. Media Capture and Streams W3C Working Draft 03 September 2013

3. Media Stream Capture Scenarios W3C Working Draft 06 March 2012

4. Media Stream Recording W3C Working Draft 05 February 2013

5. Media stream Image Capture W3C First Public Working Draft 09 July 2013

1.2.3 Other relevant Documents

1. Standards for Web Applications on Mobile: current state and roadmap

2. Device API Access Control Use Cases and Requirements W3C Working Group Note 17 March 2011

3. Device API Privacy Requirements W3C Working Group Note 29 June 2010

4. Web Application Privacy Best Practices W3C Working Group Note 03 July 2012

5. Web Intents W3C Working Group Note 23 May 2013

1.3 Implementations

The WebRTC project is an open-source project supported by Google, Mozilla and Opera. It reports implementation of current drafts in all the three browsers to a certain extent. For Google Chrome, both getUserMedia and PeerConnection are implemented and shipping in the desktop version of Chrome for Windows, Linux and Mac. These APIs do not require any flags or command line switches to use as they are now part of Chrome Stable. Mozilla Firefox has implemented getUserMedia, PeerConnection and DataChannels in the desktop version of Firefox for Windows, Linux and Mac. getUserMedia does not require any flags to use in Firefox 20 or later. It has not implemented the Recording API yet, but Mozilla plans to support it in future versions of Firefox.

It is already a common experience in W3C that once the specifications are more mature, the number of implementations will grow and the implementations will diversify. The fact that two of the main browsers with a market reach of over 50% are implementing WebRTC means that there is a high probability that the end user will get widespread access to the technology within a reasonable time.

1.4 Legalities

Telephony and, more generally, inter-personal messaging of all kinds hold a special place in our societies. Nearly all constitutions and charters of fundamental rights protect the communication channels we use. WebRTC thus has to respond to the criteria given in Article 7 of the Charter of Fundamental Rights of the European Union stating: “Everyone has the right to respect for his or her private and family life, home and communications”. These rights guaranteed in Article 7 correspond to those guaranteed by Article 8 of the ECHR.

To take account of developments in technology the word “correspondence” has been replaced by “communications”. The nuanced change in the wording means that it is conceivable that some jurisdictions might conclude that these rights to respect for communications privacy and security apply not just to ’traditional’ communications technology but also to newer technology such as WebRTC. This in turn might mean that one could apply the rich case law developed by the European Court of Human Rights (ECHR) when considering WebRTC. The Pervasive Monitoring (PM) many of us suspected and its confirmation by Edward Snowden show the significant impact that insecure real time communications can ultimately have. The ECHR has developed the rule that intervention “is legitimate and may become necessary in a democratic society ... for the protecting ... of the rights and freedoms of others.

It confers a positive obligation of the governmental actors to protect the rights of individuals". In its communication COM (2010)573 "Strategy for the effective implementation of the Charter of Fundamental Rights by the European Union", the European Commission sets ambitious objectives for its policy following the entry into force of the Lisbon Treaty. The Commission wants to make the fundamental rights provided for in the Charter of Fundamental Rights as effective as possible. This implies a duty for the European Commission to act in securing its Citizens' communications.

This report also tries to give hints as to what the actions of the European Commission could be. On 8 April 2014, the European Court of Justice (ECJ) declared the Data Retention Directive 2006/24/EC invalid. The Directive, specifying a 6 month long full retention of all traffic data, exceeded the limits imposed by compliance with the principle of proportionality in the light of Articles 7, 8 and 52(1) of the Charter of Fundamental Rights of the European Union. In this decision, the court hints at requirements for a lawful processing of logged data. There are limits to data retention as such, but the ECJ also erects further requirements concerning security of the resulting data.

• The court asked for access control to limit data access and use to what is strictly necessary in the light of the objective pursued.

• The court encourages governments to make time of data retention dependent on the category of data collected

• For future regulations in this area, the court requires the making of rules concerning the security of data retained by providers that are specific and adapted to (i) the vast quantity of data whose retention is required by that directive, (ii) the sensitive nature of that data and (iii) the risk of unlawful access to that data; rules which would serve, in particular, to govern the protection and security of the data in question in a clear and strict manner in order to ensure their full integrity and confidentiality.

• Data retention, furthermore, needs to take technical and organisational measures into account and must ensure the irreversible destruction of the data at the end of the retention period.

This ruling could have some impact on WebRTC – if it requires new national legislation in various European countries, then the extent to which WebRTC traffic is included or excluded in any future data retention laws may have a significant impact on deployments, though perhaps less of an impact on protocols. The "danger" being that requirements for data retention, if they include WebRTC call data or metadata, could act as a disincentive to many web sites from deploying WebRTC. However, it is too early to tell if such changes will or will not impact on WebRTC; for now, we merely note the risk here.

WebRTC Assets

In this chapter we begin the analysis of the WebRTC framework following the methodology from STREWS Web-platform Security Guide (Deliverable D1.1). This analysis is of course a work-in-progress and will continue to be such so long as the work of the IETF rtcweb, W3C webrtc working groups and of implementers and deployments is similarly a work-in-progress. However, it is useful to try doing the analysis in parallel with those developments. (That being one of the main hypotheses behind STREWS.) In this chapter (and overall in the remainder of the document) we emphasize the new aspects that WebRTC adds to the existing Web platform and do not for example, consider the generic Web platform security issues that were already analyzed previously.

2.1 Enumeration

We can identify the following assets related to WebRTC.

1. Browser - the end user's browser, which the end-user expects to be operating, safely, on behalf of the end-user and not, for example, working solely to the benefit of any web site accessed. This is a user asset and implements the W3C WebRTC APIs, as well as other normal browser functions. The browser is also as usual responsible for enforcing Web and WebRTC security controls, for example, the Same Origin Policy (SOP). For WebRTC the new APIs offered by the browser and how those interact with the SOP is one of our main concerns.

2. Client Machine - the end user's computer (or phone) "outside" the browser together with the potentially sensitive devices that the machine is fitted with (e.g., microphone, cam- era(s), file system and screen). This is relevant to the WebRTC as some features (in particular screen sharing) could impact on other applications running on the host, or on the host operating system itself. This is considered a user asset here, though of course could also be a corporate asset in another view. Our main interest here is how the new WebRTC functions provide potential new targets for attack via the Web.

3. Server Machine (aka Web application) - the application on the web server that sets up and manages the call. In this context, since the WebRTC is really a set of applications running on a web server, we mainly treat the web server as an application asset, but all the usual web server vulnerabilities remain relevant (e.g., direct access to storage etc.). Our main interest here is in how the web server could be attacked via the new WebRTC application. Note that there can be two web

servers involved in which case there signalling is needed between the web servers which is not standardised by WebRTC.

4. Client-side Application Code (WebRTC JS) - the JavaScript that manages the calls on the user's browser that is distributed from the web server. This includes JavaScript libraries that may be downloaded from the web server or a third party web server and that implement parts of the WebRTC JS functionality. The difference here is that these libraries are not usually under the control of the WebRTC JS author. This is an application asset.

5. Identify provider's infrastructure (IdP) - this is actually quite a complex asset, but for the purposes of this analysis can be regarded as a single service. Each user on a call may of course use a different IdP, and is likely to use the IdP for much more than WebRTC purposes. This is an infrastructure asset.

6. STUN/TURN server - the server(s) that operate the STUN/TURN services used for ICE. This is an infrastructure asset.

Figure 2.1: WebRTC Assets and Flows

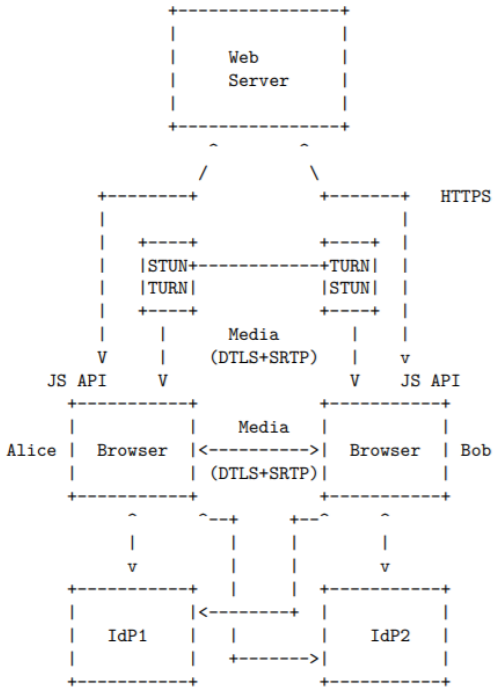
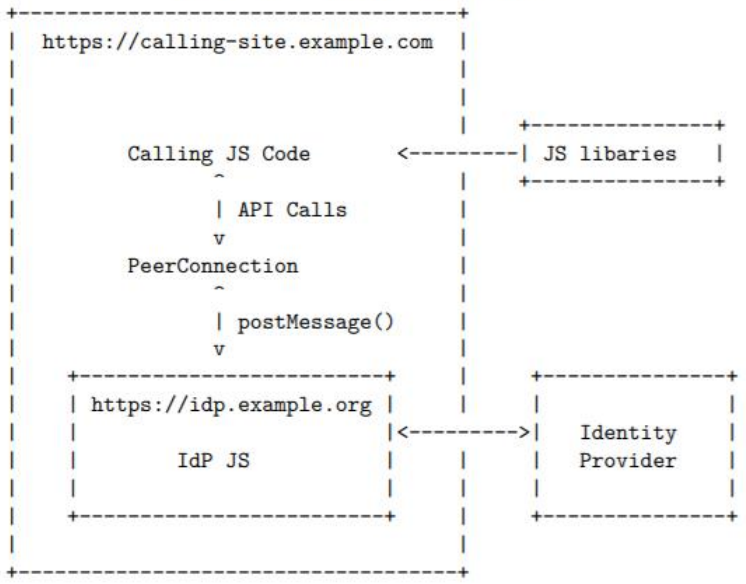


Figure 2.2: WebRTC inside the Browser



2.2 New WebRTC Interactions raise new avenues for Threats

Based on an architectural assessment of the WebRTC architecture sketched in Figure 2.1, the following security has been identified:

1. A browser can now attack another browser directly. This is new to the Web model that normally works in a client-server environment.

2. A Web server can attack a browser via another browser. If there are two web servers involved in a call, then each of those can cause "their" browser to attack the other party in the call. This is really consequence of the previous case in an indirect manifestation, as Alice's browser is executing code from Alice's web server whilst talking to Bob's browser.

3. If there are two web servers involved in a call, then each of those can attack the other via whatever signalling is in place between them.

4. The STUN/TURN servers (particularly if TURN is used) can attack the communications between the browsers in various ways. Note that the STUN/TURN server can be se- lected by, but will often not be operated by, the web server and can also be discovered algorithmically via ICE.

5. Browsers and web servers can attack STUN/TURN servers, for example to consume (unauthorized) bandwidth, or as a potential denial of service (DoS).

6. Web servers can monitor calling patterns and metadata.

7. STUN/TURN servers can monitor calling patterns and Metadata.

8. New technologies (new to the Web) such as SRTP, SDP, codecs etc. will have inevitable implementation bugs opening up attack vectors.

9. Call metadata which will be accumulated in logs of the various components will contain information about pairs of (or more) users that are not affiliated with the same web site. Previously, the Web essentially allowed accumulation of similar metadata only when two users used the same web site, absent which the logged data represented how a single user interacted with that and other web sites.

10. The semi-standardised API to allow interaction with the IdP provides a new way for attackers to attempt phishing attacks.

11. Assumptions previously made about voice communications being somewhat "out-of-band" of the Web are invalidated, especially on mobile phones. This is more of layer 8 security considerations, rather than technically part of WebRTC but significantly broader deployment of WebRTC compared to VoIP solutions may mean it becomes significant.

12. Screen sharing is a common conferencing tool that will be a new browser feature. In principle, while a screen is being shared, malicious JS could continually snapshot the image and infiltrate that to a command and control server once the user has once granted permission for screen sharing for that origin.

13. Although we don't examine it further here, the interactions between the web and legacy signalled communications (such as conference calling systems and the PSTN) could expose both to the other, resulting in damage.

2.3 New Threats by Asset

In contrast to the current Web platform, we do not yet necessarily have crisp names and definitions for a number of the threats below. As happened in the case of other new technologies, we expect it will take some time until the terminology for and etymology of the more important threats becomes well established.

In the following we generally will not repeat descriptions of threats that are already described in the STREWS Web-platform Security Guide.

2.3.1 Browser

We have two levels of vulnerabilities here. First there is the WebRTC JS itself that can be used to gain access to functionality of the Client Machine or browser that is supposed to be disallowed, or that might go beyond the users wishes.

The second level is a rogue web application that tries to escape the browser sandbox to access the operating system of the client machine or store malicious code on it so that an attacker can control the client machine from outside.

Escape Client-side Sandbox/Environment

The WebRTC JS can try to escape the sandbox the browser is providing and access local storage objects or browser functions. Ultimately, the loaded code can try to escape from the browser and install malicious code on the client side that can be used to take over the entire machine (root access, or the equivalent in the case of, for example a Chrome book machine).

In particular, if a browser implements screen sharing via a browser extension (and not via “normal” JS) then that extension will have greater power to; for example, ignore the SOP as compared with normal JS. At least one browser implementer (Chrome) at present plans to support screen sharing via this mechanism. Reportedly, this is being done in order to raise- the-bar of security, since extensions (known as plugins in other browsers) must be deliberately installed, however that also gives the extension (and hence possibly malicious code) enhanced powers, outside the sandbox. There is also the issue here that if different browsers implement screen sharing differently than different security behaviors will be seen in each.

New code with old bugs

As WebRTC requires significant new code to be included in the browser, and as implementers will find that code in various libraries, and as it is likely that none of those libraries will have been as tested to the extent that browsers are or in conjunction with all the browser (versions) that the library might be used with, we can predict a range of vulnerabilities will continue to be found in such code for some time to come. For example, CVE-2013-6631 is a use-after-free bug in a codec. Much such vulnerability can be expected.

Client-side Content Storage

Recently introduced capabilities in the browser allow a web application to store data at the client-side, within the browser. After a successful negotiation using the WebRTC API and the JSEP protocol and the SDP descriptions, a browser or the WebRTC JS may choose to store the result of the negotiation into local storage. Gaining access to this storage may allow an attacker to manipulate the stored descriptions, which typically includes high value information such as contact information for peers contacted via a particular origin.

This could be used to trick the user into communicating with the wrong peer, which is perhaps mostly a nuisance, but more seriously could leak the user's communications history. Any web site offering WebRTC functionality that has for example XSS vulnerability can offer a conduit for exploitation of this threat.

Probing Browser Capabilities

By making many attempts to connect to a browser with various different kinds of SDP proposals, an attacker can determine browser capabilities, preferences and other information about the user. Similarly, a badly designed WebRTC web application can turn a smartphone into a perfect tracking device. Inappropriate access to sensors and the measurements made by them are problems of the permission system that are analyzed in Chapter 3. Such access also creates new possibilities for an advanced fingerprinting system.

Permission Management Abuse

Most browsers discourage resetting such permissions because they only allow that to be done at a very coarse granularity (typically "forget all site data for this site"). Over time, for many users, this is likely to lead to a situation where the most popular sites have been granted permissions that would allow them to eavesdrop on the user. And for some of

those sites, for some duration, such eavesdropping will be done, either due to rogue employees, or related to some form of advertising, or via some exploit code of bug in a JS library used by the site.

New WebRTC Interstitials

Experience seems to show that any new security relevant interstitial dialogs will at some point cause problems. CVE-2014-1499 is an early demonstration of this, based on timing and such a dialog, with the former under JS control.

2.3.2 Client Machine

Access Camera or Microphone

WebRTC requires controlled access to the camera and/or microphone in order to originate data streams. Some permission systems, on smartphones in particular, are not granular enough. In Chapter3 we provide detailed analysis of possible attacks here. For example, a WebRTC application on an Android based system may ask for permission to activate the camera. But it does not control the circumstances and timing of camera activation. This means an attacker who gains control over the browser may gain control over camera, microphone and other sensors in a smartphone whenever the browser or other application is running. This turns the device into a perfect eavesdropping system that may also provide GPS coordinates.

Access Devices

Another very important aspect is accessing hardware sensors from the web application. This is mainly dealt with in the Device API Working Group in W3C. See also Section3regarding permissions to access those sensors. This covers also the most prominent fears around secret activation of camera and microphone. As an example, fingerprint sensor outputs, if raw data is accessible, could also be stolen and could allow other kinds of login.

Screen Sharing Snapshots

As previously noted, once screen sharing is enabled this enables many threats. For example, the ability to take a snapshot of the screen every 10ms say would defeat any second-factor authentication scheme that depended on selecting known images and schemes that specify subsets of password characters. Highly sensitive application data, such as bank balances or transfer details, or healthcare information could also be captured this way.

Call Recording

Some browsers may (via plugins, extensions or natively) provide call recording features. While recording ought to be known to both parties, this will not always be the case and many calls (both voice and video) will be recorded without all appropriate permissions having been acquired.

Caller-Callee Mismatch vs. Same Origin Policy

The basis for current browser security is the Same Origin Policy (SOP), and that is being extended for use with WebRTC. For example, browsers will be allowed to remember permission settings based on the SOP. This however does not match the caller/callee model that users might find more intuitive. For example, I may be much more likely to want to receive a call from my friend, rather than allow in inbound call that is setup by example.com. One could predict that such mismatches will cause some security and privacy issues as WebRTC develops.

2.3.3 Server Machine

Call Man in the Middle

The server machine does not, in the nominal case, have access to the content of the real-time communication. But as the Web server controls all the signaling, it can manipulate the data channel to go to the wrong calling party (i.e. not Bob) thus allowing for an easy man in the middle (MITM) attack. For example, setting up a call as a conference call. In any case, many calls will be between a browser and a server (e.g. customer support) and so can easily be recorded on the server side.

Call Recording

When the server is in the media path, unauthorized recordings may be made, as in the browser case.

Directory access

The Web application containing the real-time communication will have access to a directory of users of the server or other Web servers that offer real-time communication capabilities. This is not really gaining full access to a Server machine (as in root prompt), but a Web application could be manipulated on the client side to allow for bulk access to, or probing of, such directory information. A recent news report of the attempt by Facebook to acquire WhatsApp for $19 billion have been translated into cost per user or cost per address book. In the deal between Facebook and WhatsApp, newspapers talked about $45 per person. WhatsApp forces people to upload their entire address book including all the phone numbers. This means by acquiring WhatsApp, Facebook acquires a lot of user data. This results in the extremely high valuation of the proposed purchase.

In our context, it means that the directory information on the server is potentially very valuable to attackers, especially if one can download the entire directory. So while there is no abuse of Server-side Privileges in a classic sense, a Web application may be constructed or manipulated in a way to allow it to gain access to very valuable directory information. Such directory information might be a target of and exploitable via “normal” web attack such as SQL injection.

Given that we have also a distributed scenario with a second Web server, an attacker could gain access to more than one directory. If we have a world directory (as, for example, we have for DNS), access to one rogue Web application with unlimited access to all directories could turn out to be a major privacy breach that may even trigger legal action or data breach notification obligations. Further considerations concerning leaking information are found in Section7.3

Directly Access Server-side Storage

As WebRTC involves Browser-to-Browser communication, the WebRTC server only stores information about possible or potential connections between browsers. The most valuable information stored in the web server is probably the state of the connection.

Directly accessing the server-side storage will allow an attacker to find out about other potential participants of a communication and the current state of that

communication. But as the credentials are exchanged in the SDP exchanges between the browsers trying to establish a direct real time communication, the direct access to the server side storage is only done to access or influence the control channel that controls the signaling of the Browser-to-Browser communication.

Consume Server Resources

WebRTC calls involve some server resources, in particular those that involve a server-side callee (e.g., calls to a helpdesk). Clients or a botnet could use WebRTC as a significant DoS accelerator in some cases, for example if it were possible to direct video traffic from a large set of callers at an endpoint within the server infrastructure.

2.3.4 Client Side Application Code

Attack on Web Server

Because of the complexity of the WebRTC platform, it is highly probable that certain WebRTC providers will offer ready to insert JS code for pages that want to use their services. As the Web server only needs to provide the signaling, the amount of executable code on the server side is rather limited. Thus we can conclude here that running Attacker-controlled Server-side components to attack the Server machine is not very likely. This form of attack is more likely to be used against the client machine.

One can of course expect there to be bugs and possibly backdoors in such shared code, and vulnerabilities due to the complexity of WebRTC.

Attack on local Browser

Similar to the above, one can expect vulnerabilities in library code to create exploit opportunities against the caller's browser. This could come in the form of a library or piece of WebRTC code that exposes the browser to remote access, or that allows for additional fingerprinting of the browser (e.g. if a library calls home to its author) that could in turn be used for tracking or as a prelude to other attacks based on the fingerprint.

Attack on remote Browser

Similar to the above, one can expect vulnerabilities in library code to create exploit opportunities against the peer's browser.

2.3.5 Identity Provider

Phishing

The fact that the WebRTC APIs does not include a "special" IdP API means that it is not possible for browser "chrome" to be used to help users detect the differences between their interactions with real IdPs compared to with a fake site masquerading as an IdP. This therefore will not make phishing harder and may even make it easier, if WebRTC results in users becoming accustomed to entering their IdP credentials after visiting a web site and clicking on a link that sets up a WebRTC call.

Tracking

The ways in which WebRTC allows IdP's to control the duration of DTLS sessions for calls (via issuing credentials) means that an IdP that issues very short lived credentials would be in a good position to track a user's calls.

Authentication Credentials

As already laid out above, many of the use cases around WebRTC also include payment and subscription models. Those subscriptions are tied to credentials of all sorts that are more or less secure. An attacker who can get hold of such credentials thus gets a free ride on the subscription service while the victim will have to pay.

WebRTC further entrenches massive central IdPs

Web sites offering inter-user calling services will need some form of user identification. For some sites (e.g., bulletin boards) with existing nicknames or user naming, that may be relatively simple. However, for sites without such identification, WebRTC is likely to further entrench the use of massive centralized IdPs. Creating such large targets means that breaches they suffer have greater impact and also makes them more attractive to attack.

2.4 Threats against Content in Transit

Content in transit is always vulnerable, especially on a packet switched network where all middle boxes have access to the information unless the payload in the packets is encrypted. Note that even if the payload of the packets is encrypted, some headers necessarily remain visible to the boxes guiding the packets through the network. Unless one uses Onion Routing or some equivalent, source and destination are clearly visible. As Ed Felten noted: "Metadata can often be a proxy for content".

But this is a generic issue that is not special to WebRTC and is being addressed by the IETF in the future. For the analysis, we do not take this generic and inherent possibility of wiretapping and eavesdropping of the Internet into account. For WebRTC, eavesdropping of media content from the network should be greatly mitigated by the use of transport layer security for all network connections.

2.4.1 Inject and Manipulate Traffic

The exploit described here has apparently been used by the NSA in the "Quantum"[8] program. To trick targets into visiting a FoxAcid server, the NSA relies on its secret partnerships with US telecoms companies. As part of the Turmoil system, the NSA places secret servers, codenamed Quantum, at key places on the internet backbone. This placement ensures that they can react faster than other websites can. By exploiting that speed difference, these servers can impersonate a visited website to the target before the legitimate website can respond, thereby tricking the target's browser to visit a FoxAcid server.

From an attacker's point of view being able to control the content that is fed into the browsers API means it can control the entire communication between the two users via their browsers. This in turn means that control over the server side of a WebRTC communication allows for a range of manipulations of the data channel between the two communicating browsers.

As we have seen, by controlling the Web server and sending a malicious WebRTC application to the browser, the attacker gains access to most of the valuable information one can obtain in a WebRTC scenario. Not controlling the (well protected) web server, a MITM attack allows an attacker to remove the legitimate web application and to send malicious code instead. This makes it crucial for WebRTC that the Web server is trusted not to spawn malicious code and those TLS/HTTPs is used for the interaction between browser and Web server.

And this may not only happen in the relation between the Web server and the browser. In the more complex scenario of 1.2 with at least two distinct services talking to each other, the Web server to Web server communication can be also a target for manipulation of content in transit. The NSA used this to intercept traffic between data centers of Internet giants. While they had the traffic between browser and their servers encrypted, the traffic between their data centers wasn't and allowed clear access to everything.

2.4.2 Generate Traffic

Denial of service attacks is possible against the Web server providing the WebRTC service as well as against the browser. One could imagine that heavy SDP traffic from a botnet against a single browser can shut this machine down.

As WebRTC can use SIP and XMPP for the signaling, all the DDoS characteristics apply. As in politics, calling campaigns, mainly to governments or parliaments, are already current practice; it is worth noting that those will be greatly facilitated. A NGO could set up a page that uses a certain WebRTC service and then propagate the URI via social networks. Someone clicking on the link would automatically call a certain connection in parliaments or governments.

In a report from 2008 the risk of SIP DDoS is commented in a rather laconic way. It is seen as one way to abuse diagnostic tools to mute a service or to achieve a buffer overflow.

2.4.3 Authenticated Session

Use cases for WebRTC include telephony that the user has to pay for. Another use case talks about online video games that typically require some subscription and payment. The same applies to the video on demand services that require a subscription or a payment. Paid services then require authentication and credentials. Rather than leverage the IdP infrastructure, for some web sites the relevant credentials might be exchanged using SDP over the signaling channel where eavesdropping can happen. Thus an attacker can hijack a session and get a free ride on the cost of the victim whose credentials the attacker is using. The actual attack is mitigated by the use of TLS on the signaling channel as was described above.

2.4.4 Forge Request

It is clear that once the browser is on a page containing JS, this JS may mimic a WebRTC client to abuse the connection to test out some other targeted client machine. It can now, in the name of the hijacked browser request SDP information and thus explores the capabilities of the target. A forged request could also be used to escape subscription fees.

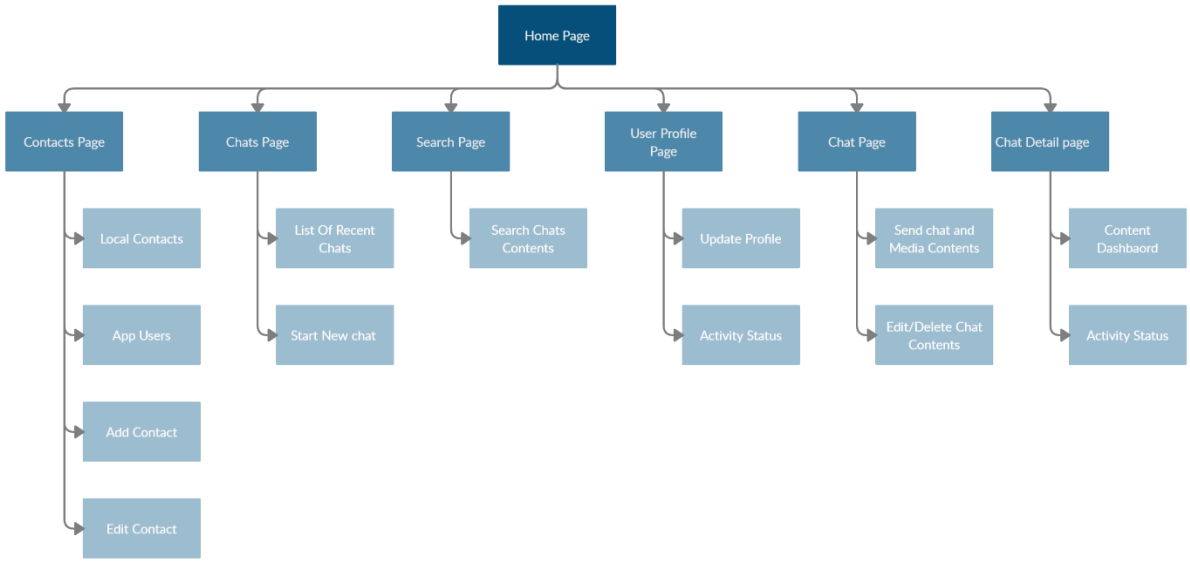
2.4.5 Personal Information

As said already many times, the biggest threats to the user's privacy via the network might be:

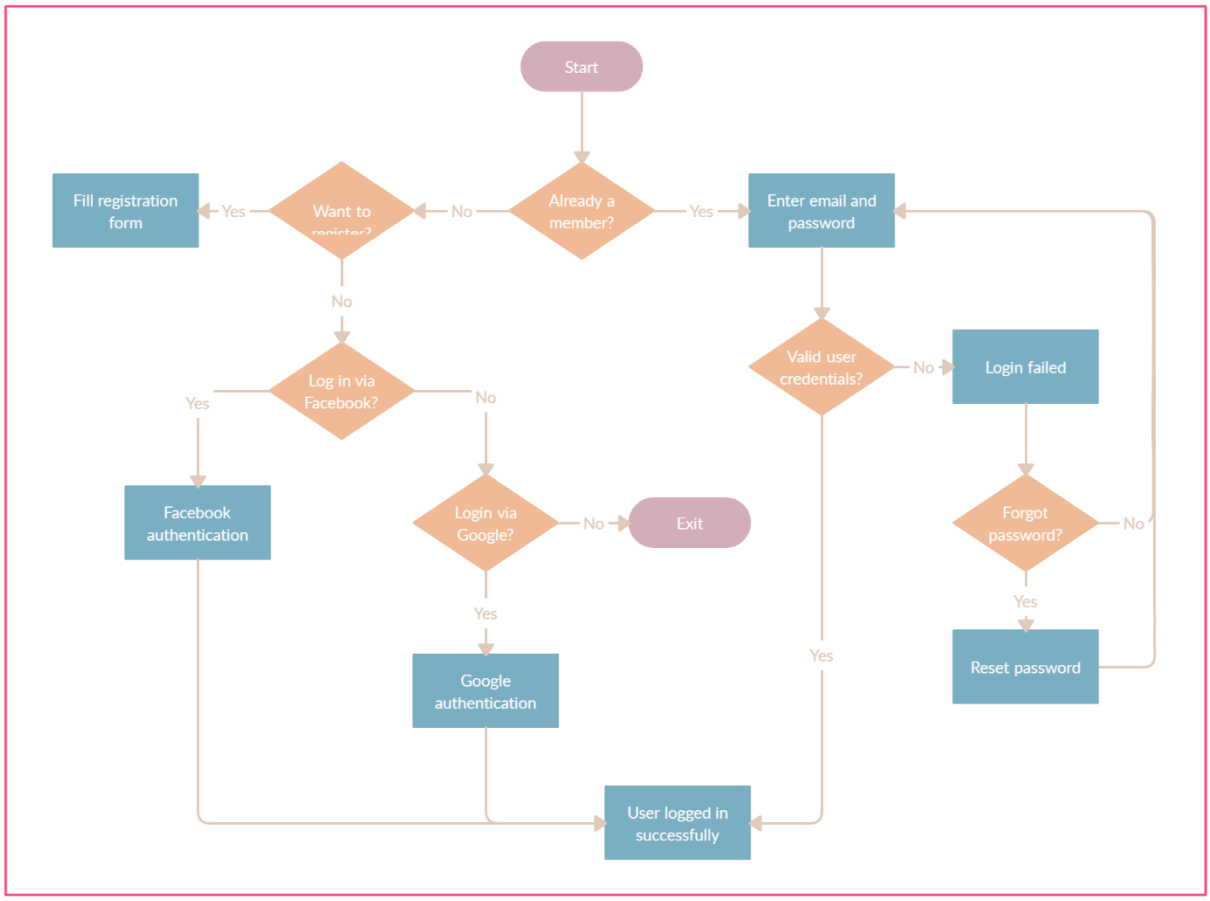
1. Eavesdropping on the browser-to-browser communication
2. bulk access and abuse of information from the WebRTC directory
3. profile building by the Web server

Diagrams

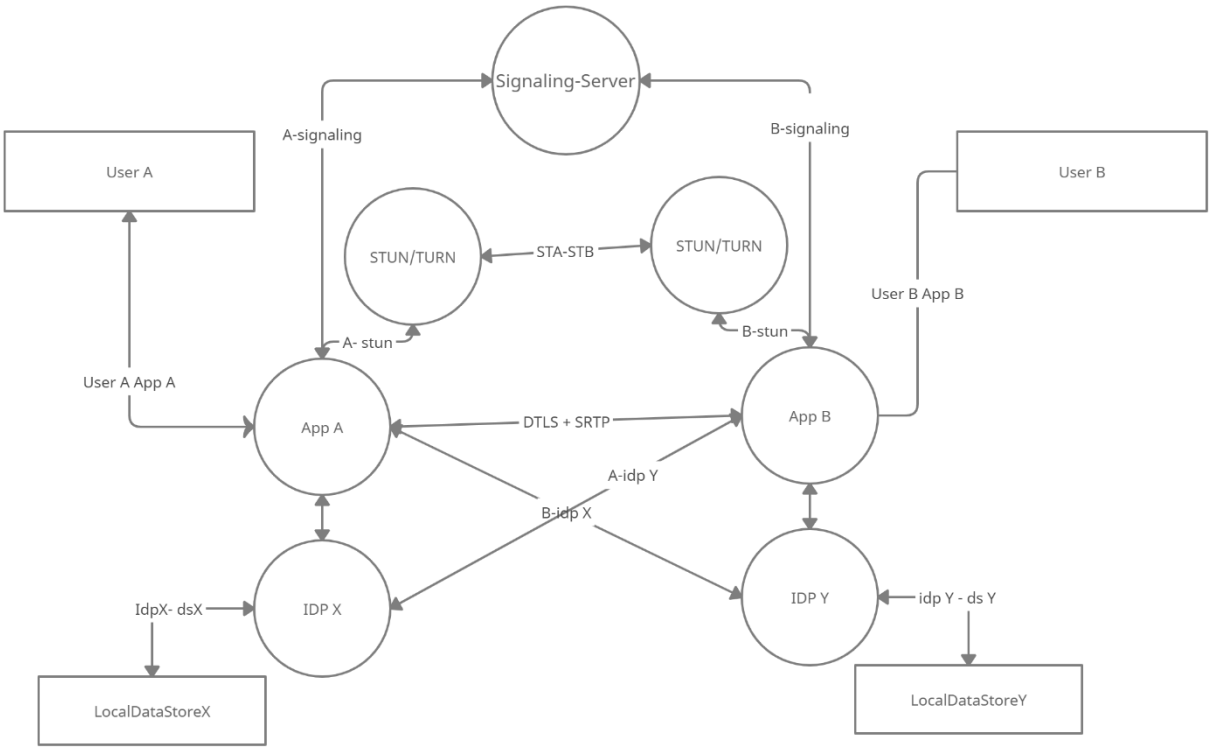
App Flow Chart



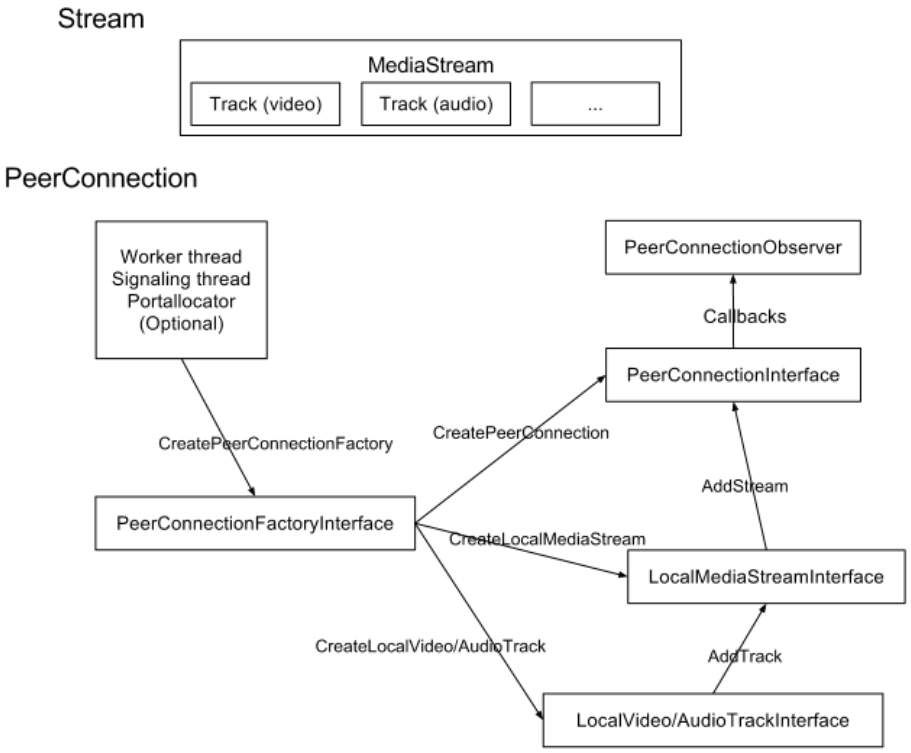
Authentication UML Diagram



Data Flow Diagram

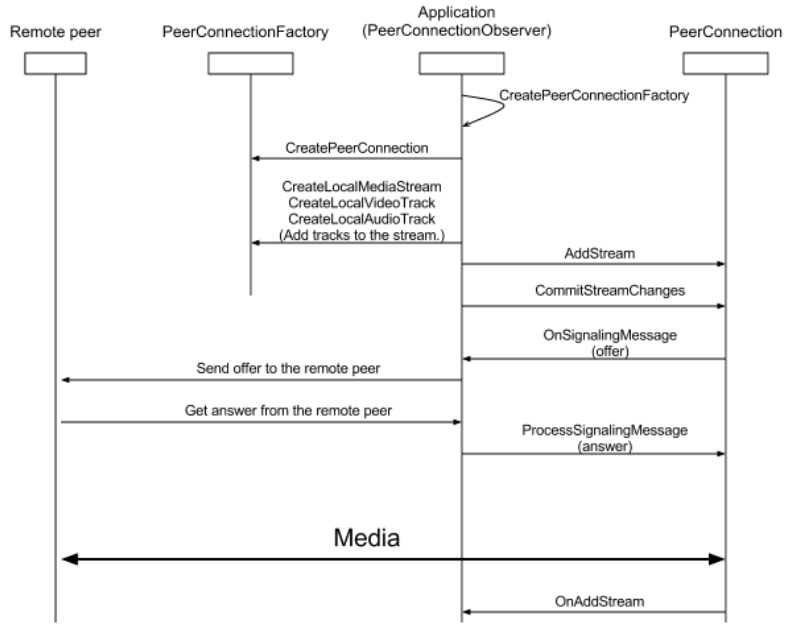


Block Diagram

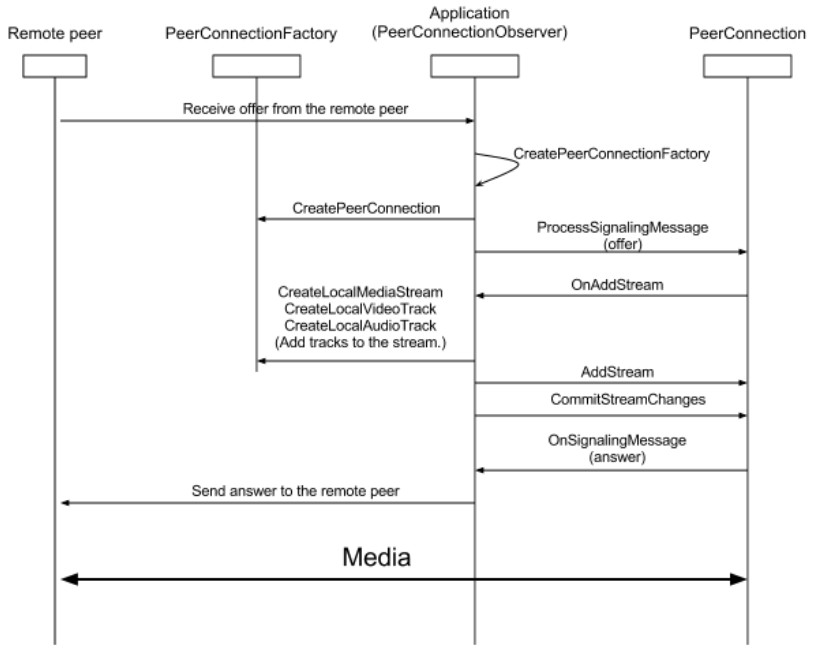


Sequence Diagram

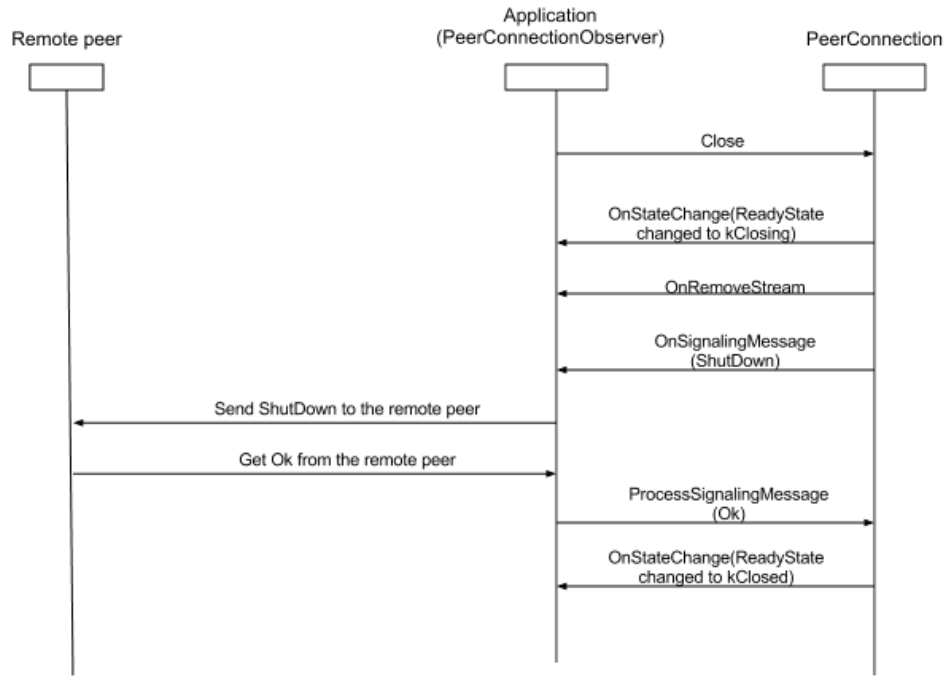
Set up a call



Receive a Call



Close Connection



Acronyms

B.Tech.	Bachelor of Technology
RTC	Real – Time Communication
WebRTC	Web Browser Real Time Communication
NAT	Network Address Translation
HTTPS	Hyper Text Transfer Protocol
STUN	Session Traversal Utilities for NAT
TURN	Traversal Using Relays around NAT
P2P	Peer-to-Peer
CPU	Central Processing Unit
MCU	Multiuser Conference

References

1. Ammar H. Ali, Ali Makki Sagheer (2017). Design of a secure android chatting application using end to end encryption, Journal of software engineering & intelligent systems.

2. Ms. Swara Pampatwar. Vinisha Kalyani, Prachi Shamdasani, Urja Ramwani (2020). Multi-layered Data Encryption/Decryption Chatting Application, Annals of R. S. C.

3. Ben Feher, Lior Sidi, Asaf Shabtai, Rami Puzis (2016). The Security of WebRTC, arXiv,

4. Kwok-Fai Ng, Man-Yan Ching, Yang Liu, Tao Cai, Li Li, Wu Chou (2014). A P2P-MCU Approach to Multi-Party Video Conference, International Journal of Future Computer and Communication.

5. Mohamed, M. A., Muhammed, A. & Man, M. (2015). A Secure Chat Application Based on Pure Peer-to-Peer Architecture. Journal of Computer Science, 11(5), 723-729.

6. J. Rosenberg and H. Schulzrinne. An Offer/Answer Model with Session Description Protocol (SDP). RFC 3264 (Proposed Standard), June 2002. Updated by RFC 6157.

7. P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Address Format. RFC 6122 (Proposed Standard), March 2011.

8. P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Core. RFC 6120 (Proposed Standard), March 2011.

9. P. Saint-Andre. Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence. RFC 6121 (Proposed Standard), March 2011.

10. G. Camarillo, O. Novo, and S. Perreault. Traversal Using Relays around NAT (TURN) Extension for IPv6. RFC 6156 (Proposed Standard), April 2011.

11. M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. The Secure Real-time Transport Protocol (SRTP). RFC 3711 (Proposed Standard), March 2004. Updated by RFCs 5506, 6904.

12. J. Bankoski, J. Koleszar, L. Quillio, J. Salonen, P. Wilkins, and Y. Xu. VP8 Data Format and Decoding Guide. RFC 6386 (Informational), November 2011.