Security Implementation in Media Streaming Applications using Open Network Adapter

Vijaya Patne  
Department of Information Science & Engineering  
RVCE Bangalore

Chethana R Murthy  
Department of Information Science & Engineering  
RVCE Bangalore

Abstract

Media has been a very important medium for entertainment and communications and the captured media was transmitted in analog form. Media providers do not want their end users to store and duplicate the streamed media because the end user can freely distribute the streamed media without any control from the source. Hence while dealing with media streaming, replay protection and integrity protection are the most important factors. The main aim of this paper is to implement the concept of WebRTC to stream the media between the participating end points which is a powerful tool used to incorporate RTC capabilities into browsers and mobile applications. The aim is to develop a secure media stream from an end point that flows through the Open Network Adapter to the Avaya Media Server (AMS) and is hosted by an application on the Engagement Development Platform. The Open Network Adapter with Avaya Fabric Attach is capable of securing the required flow.

Keywords: Engagement Development Platform, HTTP, Media Streaming, Open Network Adaptor, Session Initiation Protocol WebRTC

I. INTRODUCTION

Media Streaming sends audio or video content in compressed form over the internet without saving to the hard drive and playing immediately. The audio or video stream of data is sent in continuous form using media streaming so that the stream of data can be played as soon as the user receives it, without having to wait a user to download a file and play it. Just like a downloaded file, a user can pause, re-play the streamed media. Media streaming has following advantages [1]:

- Instant viewing of streamed media
- Users do not have to wait to download streamed file, reducing long download times
- Users can avoid wastage of memory space on hard drive
- Use specific bandwidths

The Open Networking Adapter provides few devices. The ONA can be used as an interface between devices which has Ethernet enabled port and a private cloud. Using ONA it is easy to connect the end devices like a medical device to a network. Using Fabric Attach and ONA automated provisioning is achieved. [2].The network needs to be protected from potential threats emanating from these end-points; a compromised IoT device should not be the launch point for a network- wide assault. End-point devices are associated with an Open Networking Adapter that provides dynamic, automated connectivity. The ONA-based solution delivers the required mobility, and based on the device identity, allows security services to be customized. This also allows the network presence of individual devices to be tracked, and for all services and security policies to follow the device if and as it moves within prescribed tolerances. If the ONA becomes non-compliant with policy, the device can be reset or even disabled, isolating it from the environment and thus neutralizing the risk of a threat originating from misuse or misappropriation.

ONAs enable users to connect end devices as and when required.

Fabric Attach and Fabric Connect are key building blocks of the Avaya SDN architecture. Fabric Attach can be deployed in two ways:

- In the access layer(s) of any network.
- In the access layer(s) of an Avaya Fabric Connect network.

Fabric Attach enables auto-attach of access layer and edge devices and the automatic creation of VLAN based network services. All network infrastructures supporting Fabric Attach is able to dynamically create and configure the required services right up to the legacy core network infrastructure. When Avaya Identity Engines policy server is in place, it can be used to authenticate and authorize both network devices and users, then create the VLAN based services to automatically connect the user or end device with the appropriate policy and permissions [3].

II. LITERATURE SURVEY

In video conferencing it is important to make sure that multiple participant are connected to the conference and they all can see each other. Meanwhile, it is also to make sure that these entire participants get access to all the required resources in an efficient manner. To ensure this, every conferencing application has to go through the steps shown in Figure 1 [4].
A. Conference Setup

Increase in the number of participants in video conference makes the video conferencing scenarios complex and poses lot of network challenges which will not always provide a fixed quality of service. A set of protocols are being used to address these complex issues. Media compression schemes and network topologies are used to maintain the quality of service. Two standards used to guarantee quality of service are Internet Engineering Task Force (IETF) and International Telecommunication Union (ITU) [5].

B. Session Creation/Teardown

Before starting with video conferencing the participating entities must be connected to each other. This is called as session establishment. Sometimes it is preferred that the participant must be connected to a server before starting with video conference. Whenever the participants wish to leave the conference established, they must be removed from the conference being created and thus deleting the session created. This is called session teardown. For this a lightweight internet friendly protocol; a Session Initiation Protocol is used. In other words this protocol can be used for session creation, session modification and session deletion as shown in Figure 2.

C. Media Transport

Media flow between ends points starts only on successful media handshake and matched media capability sets are exchanged. IP based networks uses Real Time Transport Protocol which is an application layer protocol and is most widely deployed protocol for media stream transports. Use of RTP for media exchange is recommended by both SIP architecture and the H.323 protocol suite. RTP is deployed over UDP to maintain steady throughput by avoiding unnecessary retransmissions of lost or delayed packets and is independent of the underlying transport layer protocol. Media stream order and synchronized playback of real time data is maintained by RTP using timestamps and sequence number. Statistics of real time session like condition of the link and the measures needed to improve the quality of the real time session is achieved using RTP Control Protocol [6]. The purpose of this protocol is to exchange information about real time packets between the communicating devices.

1) Inter-Stream Synchronization

Real-time Transport Protocol for video conferences transmits audio and video packets as separate media streams. Both the audio and video streams flow independently in the network, this is shown in Figure 3.
Since audio and video streams flow independently in the network, the packets may reach receiver at different delays. On the other hand video conferencing application might have separate pipelines for processing video and audio streams. Hence it is important to re-synchronize both audio and video packets at the receiver before the stream is played [7]. This re-synchronization can be done using timing information inside RTC headers of the independent media streams to a common sender based reference timestamp.

III. DESIGN OF MEDIA STREAMING APPLICATIONS

Any web application available now can empower a rich video chatting background with shared information exchanges using few number of JavaScript code.

A. Audio and Video Engines

Empowering a rich remotely coordinating knowledge in a browser needs the browser have the capability to get to the framework equipment to catch both sound and video; this doesn’t need custom drivers or third party plug-ins [8].

In any case, crude sound and video streams are additionally not adequate all alone: every stream should be handled to improve its quality and the bit rate of the output should conform to the latency and consistently fluctuating transfer speed between the customers. The process is reversed when the data is received on the receiving end. The customer decodes the streams progressively and has the capacity to change in accordance with system jitter and latency delays. Catching and handling sound and video is an unpredictable issue. WebRTC adds video and audio engines to the browser which is shown in Figure 4, which deals with processing signals [9].

B. System Architecture of Media Streaming Applications using ONA

The ONA is designed to address enterprise deployments that require seamless connectivity between business end-points and Fabric Connect-based services; segmentation of traffic and granular control of flows deliver previously unheard of functionality. Equally, service provider solutions such as distributed video surveillance and cloud-hosted CPE can be addressed by leveraging
the agility and flexibility delivered by OVS [10]. The openness and off-the-shelf nature of both the hardware and software makes Avaya ONA a very versatile networking component. Given that ONA is based on the Open vSwitch platform, any and all evolutions in OVS functionality and be equally applied through the environment, be that for a Hypervisor in the Data Center or a business end-point attaching via ONA [11].

Architecture of a secure media streaming application is shown in Figure 5. The Avaya Media Server provides advanced multimedia processing features to a huge range of applications and products. Making use of the latest available open standards for media processing and media control, the software which is highly scalable deploys on standard server hardware, Linux or Windows operating systems running.

C. Acquiring Audio and Video with getUserMedia

A set of new JavaScript APIs that empower the application to demand sound and video streams from the stage, and an arrangement of APIs to control and process the gained media streams are defined by media capture and stream WC specification. MediaStream is the essential interface that empowers every one of this usefulness.

The MediaStream object comprises of one or more individual tracks as shown in Figure 6. Tracks inside MediaStream are synchronized with each other. The information source can be a physical gadget, for example, a webcam or a neighborhood or remote document from the client's hard drive or a remote system peer [12]. The yield of a MediaStream can be sent to one or more destinations: a nearby video or sound component, JavaScript code for post-preparing, or a remote companion. A MediaStream object speaks to an ongoing media stream and permits the application code to acquire information, control singular tracks, and determine yields. All the sound and video preparing, for example, commotion cancelation, evening out, picture upgrade, and more are naturally taken care of by the sound and video motors. Be that as it may, the elements of the obtained media stream are compelled by the abilities of the information source: a mouthpiece can emanate just a sound stream, and a few webcams can deliver higher-determination video streams than others. Subsequently, while asking for media streams in the program, the getUserMedia() API permits us to indicate a rundown of obligatory and discretionary imperatives to coordinate the necessities of the application.

D. Engagement Development Platform

EDP is an integrated platform for customer and collaboration engagement applications. The EDP works in conjunction with any system or device which enables developers and enterprises to build and deploy customer engagement and collaboration
applications within days without advanced communication development skills. Moreover, the Avaya Engagement platform is a single integrated environment providing capabilities that extend across both Contact Center space and the Unified Communications space to build applications in such a way that reflects customer requirements and real business processes. Since Avaya Engagement platform is an integrated environment, this provides a set of developer enablement capabilities, exposing all the important enterprise collaboration capabilities in a single developer experience. As shown in Figure VII, a user can fire request to the application making a REST call. Upon loading the page it displays the user interface of the application, where the user can enter his handle and meeting id he wishes to join.

A join request is sent to the conferencing application to locate the avaya media server and creates a media session which can be accessed only by valid users. In other words, the devices which are connected to open network adapter can only make use of this newly created session. New conferences can be created on the session being created using the session id and a media server sends an answer SDP to the browser. A client sends a request SDP to the application when requesting to join media session. This part of the application forms the core of the snap-in as the title suggests. A Session Description Protocol contains few additional details like the conference Id and name of the end-user/person wishing to join the conference and return an appropriate answer SDP back to the browser Client or the Raspberry Pi device [13].

**IV. IMPLEMENTATION**

**A. Algorithm for Media Applications Streaming**

1) **Connecting to the Avaya Media Server (AMS)**
The application needs to connect to the server only once; this was implemented using a singleton class, thus using only the instances after the first connection.

2) **Storing the Participant or Client Data**
The data relating to the client received by the application is stored for future reference and each user is identified by a unique key generated after the first access to the application. The client is identified using this unique key at a later point in time. This data is stored using in Hash Map, within a synchronized class, to eliminate any data discrepancies.

3) **Creation of a conference and addition of participants**
Based on the Conference Id sent from the client, the application checks for the status and either creates a new conference or adds the client to the existing conference accordingly. Each client is associated with a media session and once the session is created by the AMS, the session is added to the conference. The details regarding the participants like the media session, user name within a conference are dynamically added and maintained as a list. The media configuration is done while creating a conference, for example the audio and video parameters are set for the entire conference while creating the conference itself.

4) **Answer SDP**
Once the client is connected to the server and has a valid session, an answer SDP is requested and received from the server based on the offer SDP sent by the client, this answer is then sent back to the Client.

5) **Session management**
Irrespective of the Conference, each client is attributed to a unique session. Based on the session details, differentiation between audio-video and only audio calls can be done. The sessions are terminated after a fixed period of inactive time; each session is monitored using its unique timer [14].
6) Termination of individual participants and deletion of conference
The participants in a conference can terminate themselves or leave the conference individually. Once the participant chooses to leave the conference, the required data is sent from the client side. Based on the unique ID sent, the session attributed to the client is first unjointed from the conference and then ended subsequently all data relating to the Client is reset. If a conference is found to have no participants or no valid sessions running the conference is removed and is ended automatically, all relevant data regarding the conference that is stored in either Hash Maps or Array Lists are reset or deleted [15].

1) User enters Name, Handle and conference id on user interface
2) Application connects to Avaya Media Server
3) Application then stores the client or user’s data
4) Join request and Offer SDP sent from the client
5) Start Media Session and add listener
6) Creation of a conference and addition of participants
7) Once a valid session established, client receives Answer SDP from server
8) Session management, after a fixed period of inactive time sessions are terminated and it goes to step 11.
9) When client sends terminate request with unique and conference id, it goes to Step 2 and Step 10
10) End Media session and remove listener
11) Termination of individual participants and deletion of conference.

V. EXPERIMENTAL RESULTS

The graphical user interface of client web page is shown in Figure 8.

![Conference Service Test](image)

Fig. 8: Client Web Page

The client web page consists of two parameters, display name and meeting id. Display name is any string value which is given as the name to the user joining the conference and Meeting Id is a unique integer value which identifies the conference a user wish to join. Figure 9 shows client 1’s web page. Here the client enters her display name and meeting Id. The meeting Id entered should be same for all the participants who wish to join this conference. After the two parameters are entered the client checks enable video button. This is to enable video conference among the participants. The user finally clicks on join button to send join request to the streaming server.

![Conference Service Test](image)

Fig. 9: Client1 sending join request
Figure 10 shows client2 sending join request. Client2 fills up the parameters like client1.

He enters his display name and meeting Id. The meeting id entered should be same as the meeting id entered by client1, in this case 1234.

VI. CONCLUSION & FUTURE ENHANCEMENT

Security for the media stream from the possible security attacks is necessary and this has been accomplished in the current work using an open network adapter for a conferencing application. The test results have shown that the media stream has been protected against the security attacks using the powerful features of Avaya’s open network adapter, security and auto-attach functionality.

Current implementation of WebRTC creates a service profile which is dependent upon health care application. The service profile creation on ONA is done manually using a health care application. Bandwidth of the media stream cannot be changed dynamically i.e., a user cannot switch between bandwidths. The user is only allowed to choose the quality of stream while he sends a request SDP.

The conferencing application implemented above should allow users to switch between the different bandwidth networks. This implementation will help user to choose the media quality which suits their requirement. Another enhancement to be made is to push the security profile to ONA without using a health care application. This can be done by using one of the REST API’s available in java. Addition of the above mentioned features to the conference application implemented using WebRTC will give more reliability, efficiency and improved performance.

REFERENCES