Day - 17 (Aug - 6)

WebRTC Section

Signaling

What is WebRTC Signaling?

When you create a WebRTC agent, it knows nothing about the other peer. It has no idea who it is going to connect with or what they are going to send! Signaling is the initial bootstrapping that makes a call possible. After these values are exchanged, the WebRTC agents can communicate directly with each other. Signaling messages are just text. The WebRTC agents don’t care how they are transported. They are commonly shared via Websockets, but that is not a requirement.

How does WebRTC signaling work?

WebRTC uses an existing protocol called the Session Description Protocol. Via this protocol, the two WebRTC Agents will share all the state required to establish a connection. The protocol itself is simple to read and understand. The complexity comes from understanding all the values that WebRTC populates it with. This protocol is not specific to WebRTC. We will learn the Session Description Protocol first without even talking about WebRTC. WebRTC only really takes advantage of a subset of the protocol, so we are only going to cover what we need. After we understand the protocol, we will move on to its applied usage in WebRTC.

What is the Session Description Protocol (SDP)?

The Session Description Protocol is defined in RFC 8866. It is a key/value protocol with a newline after each value. It will feel similar to an INI file. A Session Description contains zero or more Media Descriptions. Mentally you can model it as a Session Description that contains an array of Media Descriptions. A Media Description usually maps to a single stream of media. So if you wanted to describe a call with three video streams and two audio tracks you would have five Media Descriptions.

How to read the SDP

Every line in a Session Description will start with a single character, this is your key. It will then be followed by an equal sign. Everything after that equal sign is the value. After the value is complete, you will have a newline. The Session Description Protocol defines all the keys that are valid. You can only use letters for keys as defined in the protocol. These keys all have significant meaning, which will be explained later. Take this Session Description excerpt:

a=my-sdp-value

a=second-value

You have two lines. Each with the key a. The first line has the value my-sdp-value, the second line has the value second-value

WebRTC only uses some SDP keys

Not all key values defined by the Session Description Protocol are used by WebRTC. Only keys used in the JavaScript Session Establishment Protocol (JSEP), defined in RFC 8829, are important. The following seven keys are the only ones you need to understand right now:

• v - Version, should be equal to 0.
• o - Origin, contains a unique ID useful for renegotiations.
• s - Session Name, should be equal to -.
• t - Timing, should be equal to 0 0.
• m - Media Description (m= ...), described in detail below.
• a - Attribute, a free text field. This is the most common line in WebRTC. • c - Connection Data, should be equal to IN IP4 0.0.0.0.

Media Descriptions in a Session Description

A Session Description can contain an unlimited number of Media Descriptions.

A Media Description definition contains a list of formats. These formats map to RTP Payload Types. The actual codec is then defined by an Attribute with the value rtpmap in the Media Description. The importance of RTP and RTP Payload Types is discussed later in the Media chapter. Each Media Description can contain an unlimited number of attributes.

Take this Session Description excerpt as an example:

```
v=0
m=audio 4000 RTP/AVP 111
a=rtpmap:111 OPUS/48000/2
m=video 4000 RTP/AVP 96
a=rtpmap:96 VP8/90000
a=my-sdp-value
```

You have two Media Descriptions, one of type audio with fmt 111 and one of type video with the format 96. The first Media Description has only one attribute. This attribute maps the Payload Type 111 to Opus. The second Media Description has two attributes. The first attribute maps the Payload Type 96 to be VP8, and the second attribute is just my-sdp-value.

Full Example

The following brings all the concepts we have talked about together. These are all the features of the Session Description Protocol that WebRTC uses. If you can read this, you can read any WebRTC Session Description!

```
v=0
o=- 0 0 IN IP4 127.0.0.1
s=- c=IN IP4 127.0.0.1
t=0 0
m=audio 4000 RTP/AVP 111
a=rtpmap:111 OPUS/48000/2
m=video 4002 RTP/AVP 96
a=rtpmap:96 VP8/90000
```

How Session Description Protocol and WebRTC work together

The next piece of the puzzle is understanding how WebRTC uses the Session Description Protocol.

What are Offers and Answers?

WebRTC uses an offer/answer model. All this means is that one WebRTC Agent makes an “Offer” to start a call, and the other WebRTC Agents “Answers” if it is willing to accept what has been offered. This gives the answerer a chance to reject unsupported codecs in the Media Descriptions. This is how two peers can understand what formats they are willing to exchange.

Transceivers are for sending and receiving

Transceivers is a WebRTC specific concept that you will see in the API. What it is doing is exposing the “Media Description” to the JavaScript API. Each Media Description becomes a Transceiver. Every time you create a Transceiver a new Media Description is added to the local Session Description. Each Media Description in WebRTC will have a direction attribute. This allows a WebRTC Agent to declare “I am going to send you this codec, but I am not willing to accept anything back”. There are four valid values:

• send
• recv
• sendrecv
• inactive

SDP Values used by WebRTC

This is a list of some common attributes that you will see in a Session Description from a WebRTC Agent. Many of these values control the subsystems that we haven’t discussed yet.

#group :BUNDLE Bundling is an act of running multiple types of traffic over one connection. Some WebRTC implementations use a dedicated connection per media stream. Bundling should be preferred.

#fingerprint :sha-256 This is a hash of the certificate a peer is using for DTLS. After the DTLS handshake is completed, you compare this to the actual certificate to confirm you are communicating with whom you expect.

#setup : This controls the DTLS Agent behavior. This determines if it runs as a client or server after ICE has connected. The possible values are:
• setup:active - Run as DTLS Client.
• setup:passive - Run as DTLS Server.
• setup:actpass - Ask the other WebRTC Agent to choose.

#mid The “mid” attribute is used for identifying media streams within a session description.

#ice-ufrag This is the user fragment value for the ICE Agent. Used for the authentication of ICE Traffic.

#ice-pwd This is the password for the ICE Agent. Used for authentication of ICE Traffic.

#rtpmap This value is used to map a specific codec to an RTP Payload Type. Payload types are not static, so for every call the offerer decides the payload types for each codec.

#fmtp Defines additional values for one Payload Type. This is useful to communicate a specific video profile or encoder setting.

#candidate This is an ICE Candidate that comes from the ICE Agent. This is one possible address that the WebRTC Agent is available on. These are fully explained in the next chapter.

#ssrc A Synchronization Source (SSRC) defines a single media stream track. label is the ID for this individual stream. mslabel is the ID for a container that can have multiple streams inside it

Example of a WebRTC Session Description

The following is a complete Session Description generated by a WebRTC Client:

v=0
o=- 3546004397921447048 1596742744 IN IP4 0.0.0.0
s=- t=0 0
a=fingerprint:sha-
2560F:74:31:25:CB:A2:13:EC:28:6F:6D:2C:61:FF:5D:C2:BC:B9:DB:3D:98:14:8D:1
a=group:BUNDLE 0 1 m=audio 9 UDP/TLS/RTP/SAVPF 111 c=IN IP4 0.0.0.0 a=setup:active
a=mid:0
a=ice-ufrag:CsxzEWmoKpJyscFj
a=ice-pwd:mktpbhgREmjEwUFSIJyPINPUhgDqJlSd
a=rtcp-mux
a=rtcp-rsize
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10;useinbandfec=1
a=ssrc:350842737 cname:yvKPspsHcYcwGFTw
a=ssrc:350842737 msid:yvKPspsHcYcwGFTw DfQnKjQQuwceLFdV
a=ssrc:350842737 mslabel:yvKPspsHcYcwGFTw
a=ssrc:350842737 label:DfQnKjQQuwceLFdV
a=msid:yvKPspsHcYcwGFTw DfQnKjQQuwceLFdV
a=sendrecv
a=candidate:foundation 1 udp 2130706431 192.168.1.1 53165 typ host gener
a=candidate:foundation 2 udp 2130706431 192.168.1.1 53165 typ host gener
a=candidate:foundation 1 udp 1694498815 1.2.3.4 57336 typ srflx raddr 0.
a=candidate:foundation 2 udp 1694498815 1.2.3.4 57336 typ srflx raddr 0.
a=end-of-candidates
m=video 9 UDP/TLS/RTP/SAVPF 96
c=IN IP4 0.0.0.0 a=setup:active
a=mid:1
a=ice-ufrag:CsxzEWmoKpJyscFj
a=ice-pwd:mktpbhgREmjEwUFSIJyPINPUhgDqJlSd
a=rtcp-mux
a=rtcp-rsize
a=rtpmap:96 VP8/90000
a=ssrc:2180035812 cname:XHbOTNRFnLtesHwJ
a=ssrc:2180035812 msid:XHbOTNRFnLtesHwJ JgtwEhBWNEiOnhuW
a=ssrc:2180035812 mslabel:XHbOTNRFnLtesHwJ
a=ssrc:2180035812 label:JgtwEhBWNEiOnhuW
a=msid:XHbOTNRFnLtesHwJ JgtwEhBWNEiOnhuW
a=sendrecv

This is what we know from this message:

• We have two media sections, one audio and one video.
• Both of them are sendrecv transceivers. We are getting two streams, and we can send two back.
• We have ICE Candidates and Authentication details, so we can attempt to connect.
• We have a certificate fingerprint, so we can have a secure call.