Real-time communications with WebRTC

Lieven Desmet – iMinds-DistriNet, KU Leuven
Lieven.Desmet@cs.kuleuven.be
About me: Lieven Desmet

- Research manager at University of Leuven
  - (Web) Application Security

- Active participation in OWASP
  - Board member of the OWASP Belgium Chapter
  - Co-organizer of the OWASP AppSec EU 2015 Conference

- Program director at SecAppDev
  - Yearly week-long training on Secure Application Development
Overview

- WebRTC in a nutshell
- Communication protocols
- WebRTC JavaScript APIs
- Security & Privacy in WebRTC
- Wrap-up
WebRTC IN A NUTSHELL
Audio/video communication ... in the browser

Lync
Hangouts
AODBE CONNECT
WebEx
GoToMeeting
WhatsApp
skype
FaceTime
Source: WebRTC: A conversation Between Chrome and Firefox (by Mozilla Hacks) – http://youtu.be/MsAWR_rJ5n8
• Peer-to-peer browser connection
• Audio and media streams
• Fully JavaScript empowered
WebRTC architecture (simplified)
Various WebRTC deployments

2-party video conferencing

Federated signaling setup

Helpdesk call

Bridged to SIP/Jingle/… infrastructure
Multiple peer topologies

Peer to Peer connection

Star network

Mesh network

Multipoint Control Unit (MCU)
COMMUNICATION PROTOCOLS
Signaling path

- Signaling path between WebRTC end-points
- Signaling server(s)
  - Loads client-side context (JavaScript code)
  - Mediates control messages and meta-data between end-points
- Signaling protocol is undefined in WebRTC
  - Up to the application to deploy one!
Media path

- Secure peer-to-peer connection between browsers
  - Media streams (video/audio)
  - Data channels

- DTLS: Datagram Transport Layer Security
- SRTP: Secure Real-time Transport Protocol
  - Encryption, message authentication and integrity
Setting up the media path

1. Exchange of media parameters
   - SDP: Session description protocol

2. Exchange of network parameters
   - UDP hole punching
   - STUN: Session description protocol
   - TURN: Traversal Using Relays around NAT
   - ICE: Interactive Connectivity Establishment

Technologies borrowed from SIP
SDP: Session description protocol

- Initialization parameters for streaming media
  - Session announcement
  - Session invitation
  - Parameter negotiation (multimedia types, codecs, …)
- SDP offer and SDP answer
SDP example

v=0
o=- 20518 0 IN IP4 0.0.0.0
v=0
a=msid-semantic:WMS ma
a=group:BUNDLE audio
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8
c=IN IP4 24.23.204.141
a=mid:audio
a=msid:ma ta
a=rtp-mux
a=rtpmap:109 opus/48000/2
a=ptime:60
a=rtpmap:0 PCMU/8000
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=sendrecv
a=setup:actpass
a=ice-ufrag:074c6550
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host
a=candidate:1 1 UDP 1685987071 24.23.204.141 64678 typ srflx raddr 192.168.1.4 rport 54609
a=candidate:1 2 UDP 1685987071 24.23.204.141 64678 typ srflx raddr 192.168.1.4 rport 54609
a=rtcp-fb:109 nack
a=ssrc:12345 cname:EocUG1f0fcg/yvY7
a=rtcp-rsize
a=ice-options:trickle

Source: SDP Offer taken from “SDP for the WebRTC” (IETF Internet Draft)
UDP hole punching

- Enables connectivity between peers across NAT(s)

Diagram:
- Public Server
  - NAT / Firewall
  - Browser A
  - Browser B

Tags: #WebRTC #Security
When to use STUN/TURN/ICE?

- **STUN**
  - To collect your local network setup (local IPs, local subnets, NAT configuration, …)

- **TURN**
  - To relay your media connection if peer-to-peer fails

- **ICE**
  - Bundles all STUN/TURN information for exchange via the signaling channel
STUN: Session Traversal Utilities for NAT

- Discover your public IP address
- Determine whether your browser sits behind a NAT
- Retrieve the UDP port that NAT has allocated for external communication
TURN: Traversal Using Relays around NAT

- Used if STUN does not work
- TURN server relays traffic between 2 NAT’ed peers
- IP and port get allocated on STUN for sending or receiving a stream
ICE: Interactive Connectivity Establishment

- Gathering info (STUN, TURN, …)
- Offering and answering ICE candidates between peers
- Probe candidates in order of priority
  - Until ICE candidate pair works
WebRTC architecture (less simplified)

Browser A

HTTPS

Web Server (Signaling server)

HTTPS

Browser B

STUN TURN

ICE/STUN/TURN

DTLS+SRTP

#WebRTC #Security
Identity provision

➢ To authenticate the endpoint with an Identity Provider (IdP)
  ✗ Code of IdP gets loaded in a JavaScript realm
  ✗ Interaction between client-side code and IdP via Web Messaging (aka postMessage)
Loading the Identity Provider

Browser

1. Application-specific JavaScript code
2. WebRTC JavaScript APIs
3. IdP Proxy

Web Server (Signaling server)

Identity Provider

HTTPS

#WebRTC #Security
WebRTC architecture (full)
WEBRTC JAVASCRIPT APIS
Fully JavaScript empowered…

- Purpose of the WebRTC JavaScript APIs
  - Handle Audio/Video streams
  - Both local and remote

- Initialize the browser’s P2P capabilities
- Obtain all necessary information, so that the remote party can connect
  - SDP offer
  - ICE candidates
Setting up a RTCPeerConnection

```javascript
// overcome temporary browser differences 😊
RTCPeerConnection = window.RTCPeerConnection ||
window.mozRTCPeerConnection || window.webkitRTCPeerConnection;

// configuration of STUN, TURN, …
// can also be derived automatically by the browser
var configuration = {
  "iceServers": [{"url": "stun:stun.example.org"}]
};

// Creating the Connection object and add a handler for incoming streams
peerConnection = new RTCPeerConnection(configuration);
```
Handling SDP offers and answers

```javascript
// create a SDP offer on negotiation
peerConnection.onnegotiationneeded = function () {
    peerConnection.createOffer(function (offer) {
        // set it as the Local SDP description and send the offer to the other peer
        return peerConnection.setLocalDescription(offer, function () {
            signalingChannel.send(JSON.stringify({ "sdp": peerConnection.localDescription }));
        })
    })
}

signalingChannel.on('message', function (evt) {
    if (message.sdp) {
        var desc = new RTCSessionDescription(message.sdp);
        // if we get an offer, we need to reply with an answer
        peerConnection.setRemoteDescription(desc, function () {
            return peerConnection.createAnswer(function (answer) {
                return peerConnection.setLocalDescription(answer, function () {
                    signalingChannel.send(JSON.stringify({ "sdp": peerConnection.localDescription }));
                })
            })
        })
    }
});
```
// send any ice candidates to the other peer
peerConnection.onicecandidate = function (evt) {
    if (evt.candidate) {
        signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
    }
};

// receive and process remote ICE candidates
signalingChannel.on(msg)', function (evt) {
    if (msg.candidate) {
        peerConnection.addIceCandidate(new RTCIceCandidate(msg.candidate));
    }
});

Exchange of information “how” to connect.
Capturing a video stream

```javascript
// overcome temporary browser differences
navigator.getUserMedia = navigator.getUserMedia ||
navigator.webkitGetUserMedia || navigator.mozGetUserMedia;

// request a UserMedia Stream and use it on the RTCPeerConnection
navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {
    if(stream){
        video1.src = URL.createObjectURL(stream);
        peerConnection.addStream(stream);
    }
} logError);
```

Asks the user for permission
Setting up Identity provision

```
// setting up the identity provider
// [ this can also be done by the browser ]
// commented out example: also provide optional protocol and username
peerConnection.setIdentityProvider("example.com", "default", "alice@example.com");

peerConnection.setIdentityProvider("example.com");

// possible interaction with the IdP proxy
// this is done implicitly by the PeerConnection
peerConnection.getIdentityAssertion();

peerConnection.onpeeridentity = function(e) {
  console.log("IdP= " + e.target.peerIdentity.idp + " identity=" + e.target.peerIdentity.name);
};
```

Happens behind the scenes
WebRTC assessment by EU-funded project STREWS
Joint effort of SAP, W3C, TCD, and KU Leuven

SECURITY & PRIVACY OF WEBRTC
Attacker models in scope

- Network attacker
  - Can observe (and tamper) network traffic

- Malicious web server
  - Sits inline on the signaling path (signaling server)
  - Controls the client-side JavaScript code

- Malicious 3\textsuperscript{rd} party JavaScript
  - Can be injected (XSS) or included
  - Runs in same context as WebRTC JS code
Overview of the security analysis

- **General observations**
  - Cross p2p attacker surface
  - WebRTC permission model

- **Security impact on WebRTC-enabled websites**
  - Potential meta-data leaks
  - Eavesdropping the connection
  - Endpoint authenticity

- **Security impact on websites without WebRTC**
  - Network info leaks
  - Giving up online privacy
GENERAL OBSERVATIONS

#1 CROSS P2P ATTACK SURFACE
Increased attack surface

[Diagram showing increased attack surface in WebRTC security context]
GENERAL OBSERVATIONS

#2 WEBRTC PERMISSION MODEL
Permission model / UI feedback

For which operation is user consent required?

- Camera? ✔
- Microphone? ✔
- Getting network characteristics (ICE)? ❌
- Setting up a peer-to-peer connection? ❌
- Sending your audio/video to a remote peer? ❌
- Sharing your screen? ❌ ✔
- Selecting an appropriate Identity Provider? ❌
- Verifying your endpoint’s identity? ❌
WebRTC-ENABLED WEBSITES

#3 POTENTIAL META-DATA LEAKS
Meta-data leakage:
Trace that communication has happened
Leaking the fact that communication has happened between entities:

- Signaling server
- STUN/TURN server (*)
- IdP server (*)
- 3rd party JavaScript provider

(*) Possibly by aggregating data from both end-points
WebRTC-ENABLED WEBSITES
#4 NETWORK ATTACKERS EAVESDROPPING THE CONNECTION
Eavesdropping on the connection

Browser A ➔ MiTM ➔ Browser B

1. MiTM

Web Server (Signaling server)

Identity Provider X

Identity Provider Y

STUN TURN

STUN TURN

3rd party JavaScript

3rd party JavaScript

#WebRTC #Security
DTLS-RSTP to the rescue

- DTLS provides
  - Encryption
  - Message authenticity
  - Message integrity

- Endpoint’s certificate fingerprint is stored as part of the SDP
WebRTC-ENABLED WEBSITES
#5 SIGNALING COMPONENTS
EAVESDROPPING THE CONNECTION
Setting up the media channel

1. Browser A requests STUN TURN from 3rd party JavaScript
2. SDP is sent from Key Store to Browser A
3. SDP is sent from Browser A to Identity Provider X
4. SDP is sent from Identity Provider X to Web Server (Signaling server)
5. SDP is sent from Web Server (Signaling server) to Identity Provider Y
6. SDP is sent from Identity Provider Y to Web Server (Signaling server)
7. SDP is sent from Web Server (Signaling server) to Browser B
8. SDP is sent from Browser B to 3rd party JavaScript

A MitM attack occurs at step 8, potentially stealing the SDP data.
Eavesdropping on the connection

1. MiTM
2. Reroute Stream

Browser A
- Identity Provider X
- Web Server (Signaling server)
- STUN
- TURN

Browser B
- Identity Provider Y
- Web Server (Signaling server)
- STUN
- TURN

3rd party JavaScript

3rd party JavaScript
Eavesdropping on the connection

- Set up the connection to a MiTM
  - By modifying the SDP information
- Reroute the stream
  - By cloning the media stream in JavaScript

Can be achieved by:
- Malicious 3rd party JavaScript (included or via XSS)
- Malicious signaling server
WebRTC-ENABLED WEBSITES
#6 ENDPOINT AUTHENTICITY
Setting up the media channel with the IdP

1. Browser A sends STUNTURN request to Identity Provider X
2. Identity Provider X generates SDP and sends it to Browser A
3. Browser A sends STUNTURN request to Identity Provider Y
4. Identity Provider Y generates SDP and sends it to Browser A
5. Web Server (Signaling server) receives SDP from Browser A
6. Web Server (Signaling server) sends SDP to Browser B
7. Browser B sends SDP to Web Server (Signaling server)

Signature does not match
IDP setups

IDP = Signaling Server

- Browser chooses IDP
- Signaling server chooses IDP

User chooses own IDP

- Browser chooses IDP
- Signaling server chooses IDP
WEBSITES WITHOUT WebRTC
#7 LEAK OF LOCAL NETWORK INFO
Leaking local network info (ICE)

Your network IP is:
172.16.193.251 or perhaps fd7f:fca3:76f1:272d:6c2a:5116.53.38.165

Make the locals proud.
WEBSITES WITHOUT WebRTC

#8 HOLY GRAIL OF ...

... GIVING UP ONLINE PRIVACY
Maliciously triggering the IdP

- IdP is not allowed to have any user interaction
- If user is logged in:
  - Authenticate based on cookies/localstorage/…
- Else:
  - Fail authentication
  - Return authentication URL to JS signaling code
- Up to signaling code to load authentication URL

IdPs will automatically provide identity assertions for \(<\text{username}>@<\text{idp-domain}>\)
Say goodbye to online privacy...

1. Malicious JS code can trigger automatic identity assertions
2. Online identity can be extracted from the identity assertion
3. Malicious JavaScript code can iterate over set of popular identity providers
4. All retrieved identities can linked to each other
WRAP-UP
Take home message

- WebRTC increases the attack surface
- WebRTC permission model is very liberal
  - Your browser has become a peer-to-peer tool without needing your consent
- DTLS-SRTP without an IdP does not authenticate endpoints
  - Use an identity provider to assert the identity of your remote party
  - The IdP integration may be subject to change in the near future
Take home message (2)

- JavaScript running in your application have full control over the WebRTC functionality
  - To eavesdrop on the media streams
  - To get network and identity information
- Limit trust in 3rd party JS running in your origin
- Use best-practices to protect against XSS

- Embrace the new browser capabilities!
Relevant sources

- Large security assessment of relevant specifications
  - Joint work with IETF, W3C and SAP
  - https://www.strews.eu/results/91-d12

- Identifying open issues and security challenges for WebRTC
  - Special Issue of IEEE Internet Computing, nov/dec 2014
Client-side Web Security Handbook

- Provides an up-to-date overview of
  - State-of-the-art in web security
  - State-of-practice on the Web
  - Recent research and standardization activities
  - Security best practices per category
