Real-Time communication with WebRTC

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

SecAppDev Leuven 2015 (27/02/2015, Leuven)
About myself: Lieven Desmet

- Research manager at KU 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
iMinds-DistriNet, KU Leuven

- Headcount:
  - 10 professors
  - 65 researchers

- Research Domains
  - Secure Software
  - Distributed Software

- Academic and industrial collaboration in 30+ national and European projects

https://distrinet.cs.kuleuven.be
Relevant sources

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

- Identifying open issues and security challenges for WebRTC
  - Special Issue of IEEE Internet Computing, nov/dec 2014
WebRTC ?
Real-Time communication on the Web
Overview

- Introduction to WebRTC
- WebRTC JavaScript APIs
- WebRTC deployments
- Overview of attack vectors
- Wrap-up
Introduction to WebRTC
WebRTC architecture

- **Web Server (Signaling server)**
  - HTTPS connection
  - STUN/TURN signaling
  - ICE/STUN/TURN signaling

- **Browser A**
  - HTTPS connection
  - DLTS+SRTP media stream
  - JS API
  - Identity Provider X

- **Browser B**
  - HTTPS connection
  - DLTS+SRTP media stream
  - JS API
  - Identity Provider Y

- **Identity Provider X**
- **Identity Provider Y**
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

- SDP
- UDP hole punching
- STUN
- TURN
- ICE
SDP: Session description protocol

- Initialization parameters for streaming media
  - Session announcement
  - Session invitation
  - Parameter negotiation (multimedia types, codecs, …)

- SDP offer and SDP answer
v=0
o=- 20518 0 IN IP4 0.0.0.0
s=-
t=0 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=rtcp-mux
a=rtcp:54609 IN IP4 24.23.204.141
a=rtpmap:109 opus/48000/2
a=ptime:60
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000

...
UDP hole punching

- Enables connectivity between peers across NAT(s)
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 work
To authenticate the endpoint, an Identity Provider (IdP) can be used

- Code of IdP gets loaded in an iframe
- Interaction between client-side code and IdP via Web Messaging (aka postMessage)

Browser A

Load IdP in an iframe

Get identity assertion

Authenticated endpoint

Identity Provider

Browser B

Verify identity assertion
WebRTC JavaScript APIs
To give you an idea of the complexity: The simple case... (1)

Taken from “WebRTC 1.0: Real-time Communication Between Browsers“ (W3C Editor’s Draft)
To give you an idea of the complexity: The simple case... (2)

Taken from “WebRTC 1.0: Real-time Communication Between Browsers” (W3C Editor’s Draft)
Capturing a video stream

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

// request a UserMedia Stream and use it on the local page and 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
// 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);

peerConnection.onaddstream = function (evt) {
    video2.src = URL.createObjectURL(evt.stream);
};
Handling SDP offers and answers

// 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 }));
                })
            });
        })
    })
});
Handling ICE Candidates

// 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('message', function (evt) {
    if (message.candidate) {
        peerConnection.addIceCandidate(new RTCIceCandidate(message.candidate));
    }
});
// setting up a data channel
var dataChannel = peerConnection.createDataChannel("myLabel", dataChannelOptions);

dataChannel.onerror = function (error) { … };

dataChannel.onmessage = function (error) { … };

dataChannel.onopen = function (error) { … };

dataChannel.onclose = function (error) { … };
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 deployments
Various WebRTC deployments

Helpdesk call

Federated signaling setup

Bridged to SIP/Jingle/… infrastructure
Multiple peer topologies

Peer to Peer connection

Star network

Mesh network

Multipoint Control Unit (MCU)
**IDP setups**

-IDP = Signaling Server

-User chooses own IDP

-Browser chooses IDP

-Signaling server chooses IDP
Overview of attack vectors
Attack overview

- Classical web attacks still apply
- WebRTC permission model
- Potential confidentiality leaks
- Endpoint authenticity
- Attacking the underlying infrastructure
#1 Classical web attacks still apply
Client-side web attacks still apply
Classical attacks in WebRTC setup

- Attacks such as XSS also apply in a WebRTC setup
  - New attack surface added
  - New capabilities can be achieved
  - Hard to trace back to its origin

- Attacks can cross origins and browsers, via peer-to-peer connection
Don’t forget the server-side web attacks

1. Browser-to-Signaling
2. Signaling-to-Signaling
3. Signaling-to-Signaling

1. 3rd party JavaScript
2. STUN TURN
3. STUN TURN

Browser A
Identity Provider X

Browser B
Identity Provider Y

Web Server (Signaling server)

3rd party JavaScript
#2 WebRTC permission model
Permission model

- For which operation, user consent is 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? ✗
Potential issues due to permission model

- Multiple streams of your camera been sent to different parties
- Phishing opportunities for IdP authentication
- ICE fingerprinting
- Screen sharing via extension

Verification of endpoint’s authenticity depends on:
- Signaling server setting up IdP authentication and verification
- Browser setting up selection of IdP
- Browser displaying IdP verification
#3 Potential confidentiality leaks
Meta-data leakage: Trace that communication has happened

1. Web Server (Signaling server) → STUN TURN
2. STUN TURN → Web Server (Signaling server)
3. 3rd party JavaScript → Browser A → Identity Provider X
4. Identity Provider Y → 3rd party JavaScript

1. Signaling server
2. STUN/TURN server
3. 3rd party JavaScript
4. Identity Provider
Eavesdropping on the connection

3rd party JavaScript

Browser A
STUN TURN
Identity Provider X

MiTM

Web Server (Signaling server)

Browser B
STUN TURN
Identity Provider Y

3rd party JavaScript

1 MiTM

2 Reroute Stream
#4 Endpoint authenticity
Setting up the media channel without IdP

1. Web Server (Signaling server) sends SDP to Browser A
2. Browser A sends SDP to Key Store
3. Key Store sends SDP to Identity Provider X
4. Browser A requests STUN TURN
5. Web Server (Signaling server) sends SDP to Browser B
6. Browser B requests STUN TURN
7. 3rd party JavaScript requests SDP from Browser B
8. MiTM intercepts and modifies the SDP messages.
Setting up the media channel with the IdP

1. Browser A
   - Key Store
   - SDP
   - STUN
   - TURN

2. Identity Provider X
   - SDP

3. Identity Provider Y
   - SDP
   - Signature does not match

4. Web Server (Signaling server)
   - SDP

5. Web Server (Signaling server)
   - SDP

6. 3rd party JavaScript

7. 3rd party JavaScript

3rd party JavaScript

SDP

Signatures do not match
#5 Attacking the underlying infrastructure
There are many other things to consider...

- STUN/TURN server share symmetric secret for authentication
  - Steal bandwidth of other tenants
  - Trigger DDoS attacks on/via STUN/TURN

![Diagram showing STUN/TURN and Web Server interactions](image)
There are many other things to consider…

- Attack legacy components such as SIP/VOIP/Jingle/…
Wrap-up
Flexibility vs security?

- Many deployment variation make it hard to assess the end-to-end security

- Very open specifications
  - Signaling path is not specified
  - Limited specification on how to interact with the IdP
  - Responsibility for IdP settings is unspecified
    - Could be the responsibility of the end-user, browser, signaling server
    - No obligation to use IdPs (i.e. unauthenticated endpoints by default)
    - Unspecified how identity management needs to be visualized
Minimal permission model enforced

- Only permission for capture of video/audio is currently needed
- A PeerConnection can be setup without user intervention
  - A user does not know on what connection a stream is used
  - Streams can be cloned over multiple connections
- PeerConnections can be set up without authenticating endpoints
- Websites can trigger ICE fingerprints
WebRTC: security impact on applications?

- Even if your application is not using WebRTC, new attack vectors and attacker capabilities may apply
  - Browser-to-browser attacks, …
  - ICE fingerprints
  - Screen sharing to circumvent other security mechanisms
  - …
Peer-to-peer provides more privacy?

- Peer-to-peer could provide more confidentiality in user communication

- But:
  - Possible collection of meta-data at signaling server, STUN/TURN server, IdP provider, …
  - Possible confidentiality leaks in communication channel
    - MiTM attacks in unauthenticated endpoints
    - Streams being cloned and rerouted
Taking home message

- Limit trust in third-party libraries running in your origin
  - If possible, isolate the WebRTC functionality in a separate origin (e.g. subdomain)

- Use an identity provider to authenticate the end-points

- Use best-practices in protecting your application
  - Don’t forget the browser-to-browser, signaling-to-signaling communication

- Be careful with screen sharing extension

- Embrace the new browser capabilities!
Real-Time communication with WebRTC

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

SecAppDev Leuven 2015 (27/02/2015, Leuven)