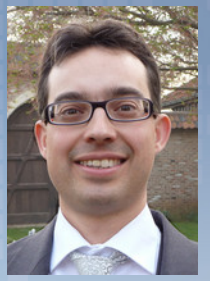
WebRTC, Or How Secure Is P2P Browser Communication?

Lieven Desmet – iMinds-DistriNet, KU Leuven
Martin Johns – SAP AG

OWASP AppSecEU 15
Amsterdam, The Netherlands

About us: Lieven Desmet



@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

OWASP AppSecEU 15
Amsterdam, The Netherlands

About us: Martin Johns

@datenkeller

- Research Expert at SAP AG
  - Leader of the web application security team
- Board member of the OWASP Germany Chapter
- Speaker at international security conferences
  - OWASP AppSec series, BlackHat, CCS, PacSec, HackInTheBox, RSA Europe, CCC, ...

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

GoToMeeting by CITRIX

Hangouts

ADOBE CONNECT

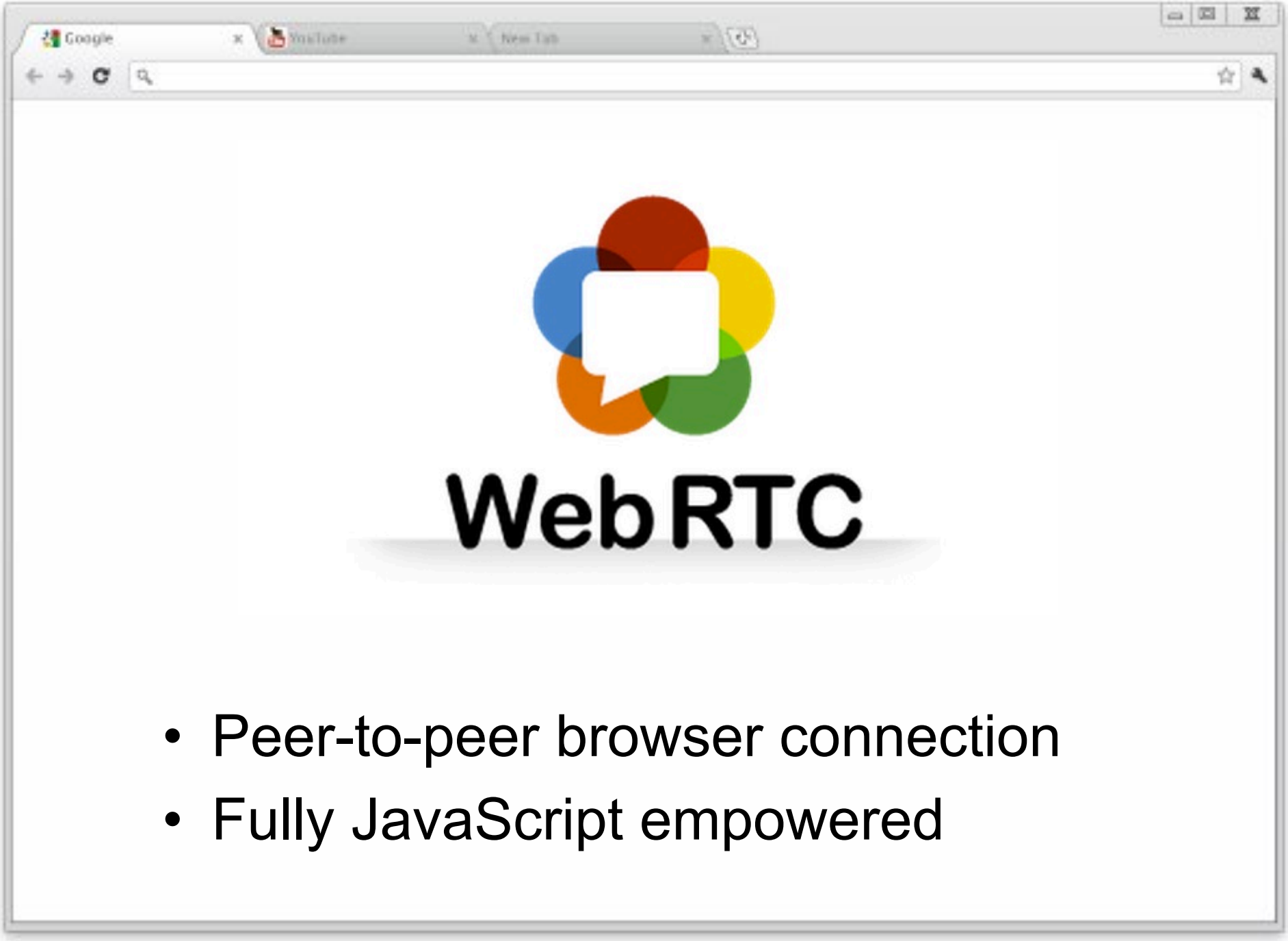
skype

cisco WebEx

FaceTime

WhatsApp

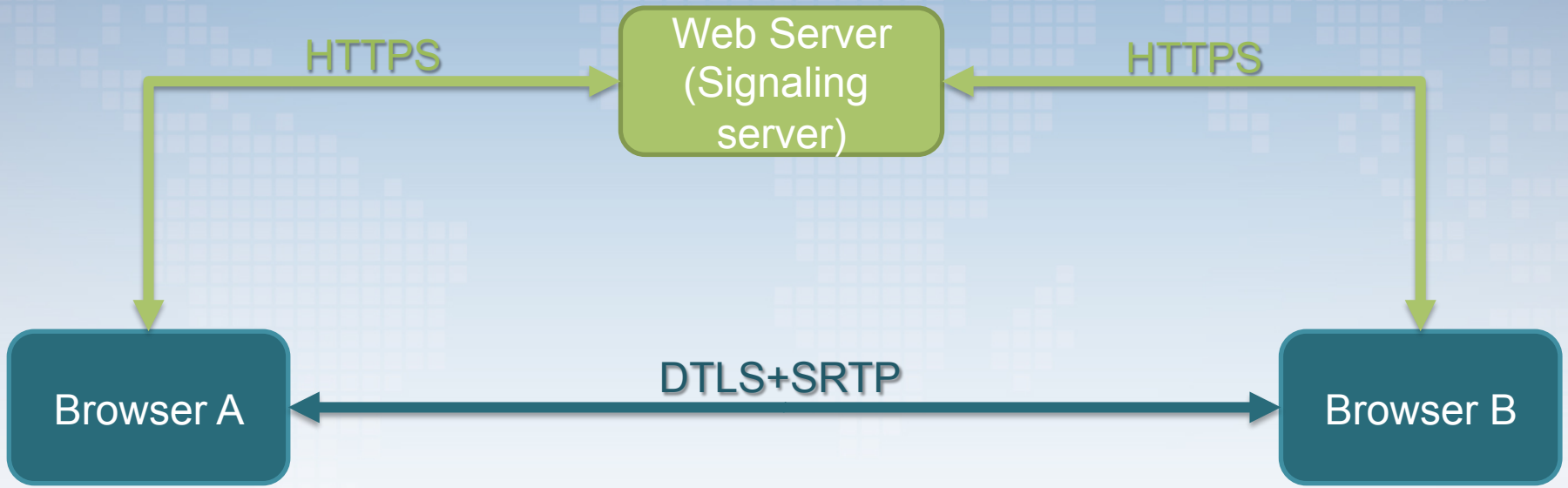
OWASP AppSecEU 15
Amsterdam, The Netherlands



- Peer-to-peer browser connection
- Fully JavaScript empowered

Source: WebRTC: A conversation Between Chrome and Firefox (by Mozilla Hacks) – http://youtu.be/MsAWR_rJ5n8

WebRTC architecture (simplified)

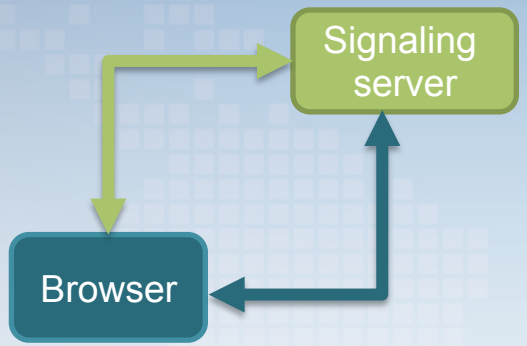


IETF®

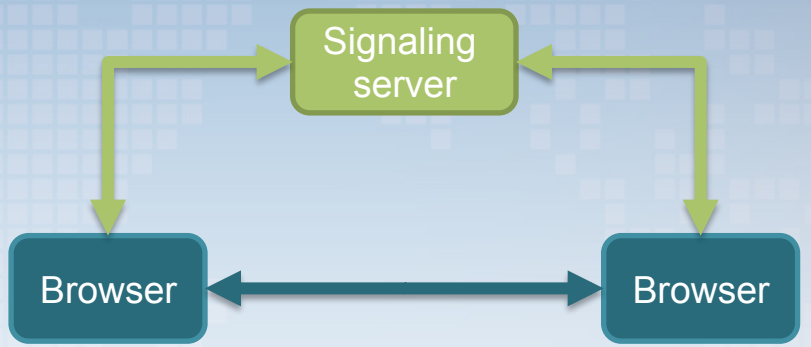
W3C®

OWASP AppSecEU 15
Amsterdam, The Netherlands

Various WebRTC deployments

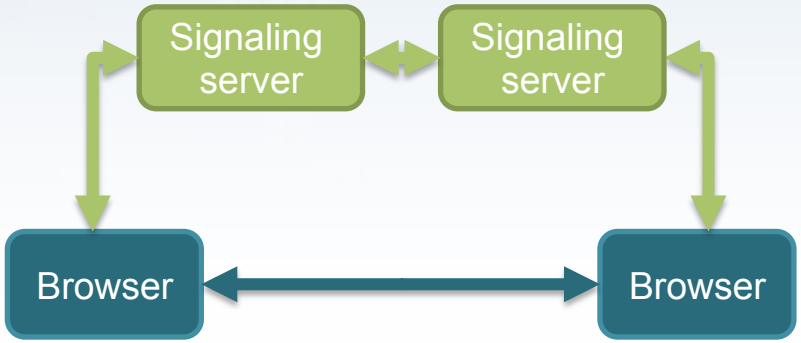


Helpdesk call

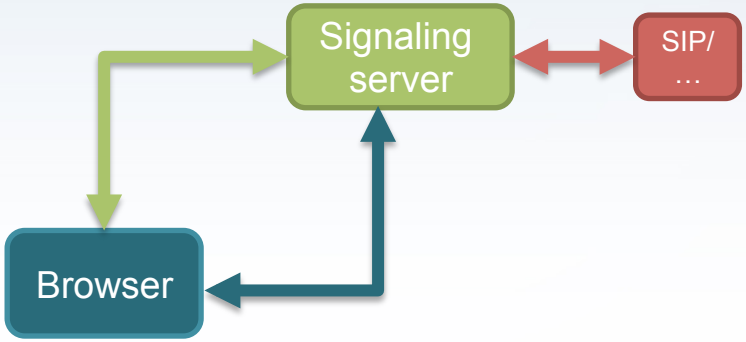


2-party video conferencing

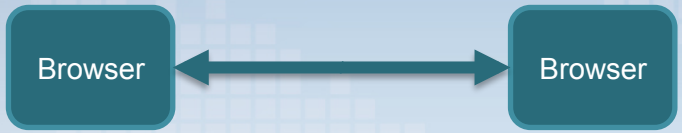
Federated signaling setup



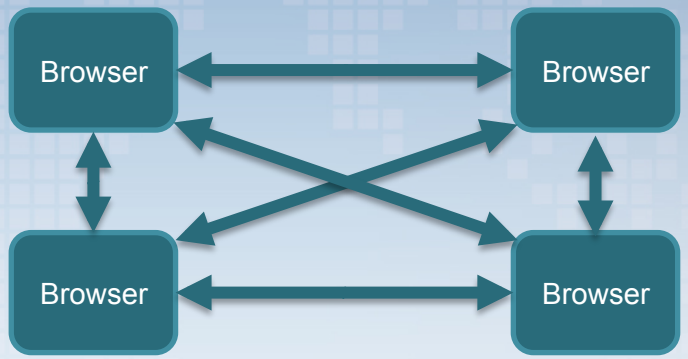
Bridged to SIP/Jingle/… infrastructure



Multiple peer topologies

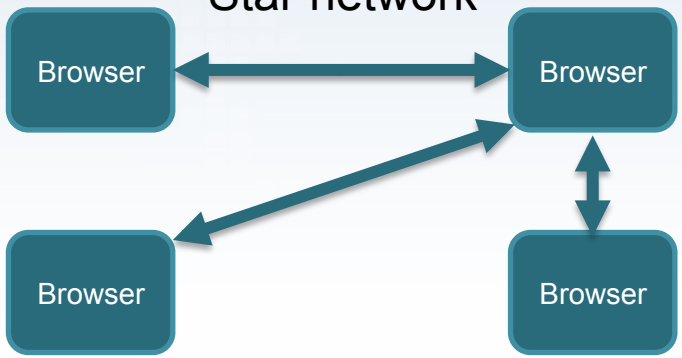


Peer to Peer connection

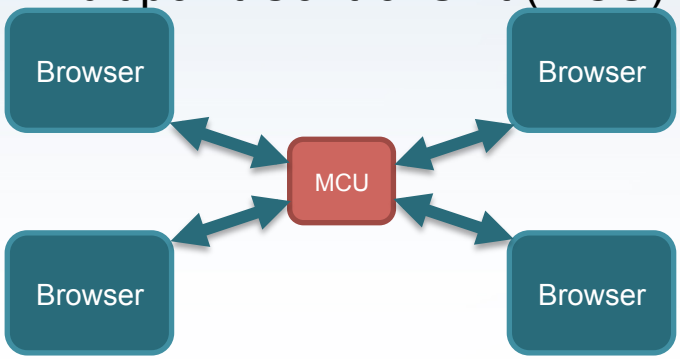


Mesh network

Star 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

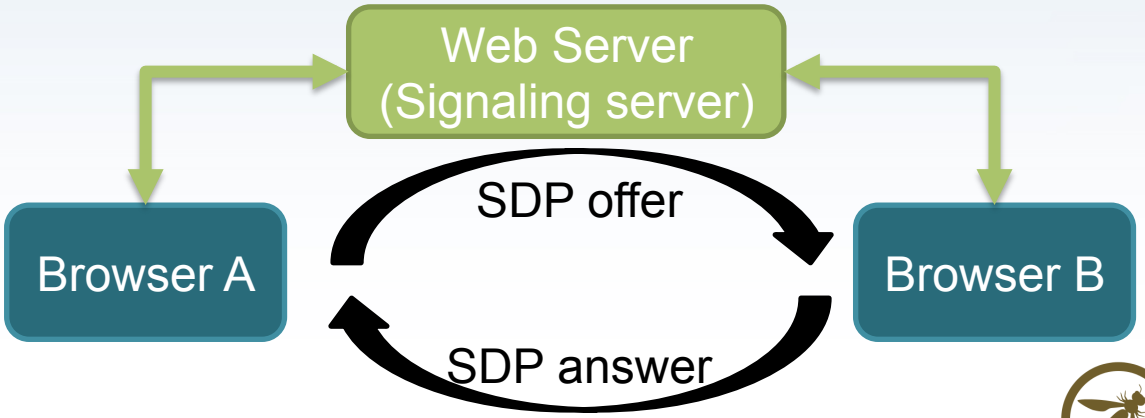
1. Exchange of media parameters
   - SDP: Session description protocol

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

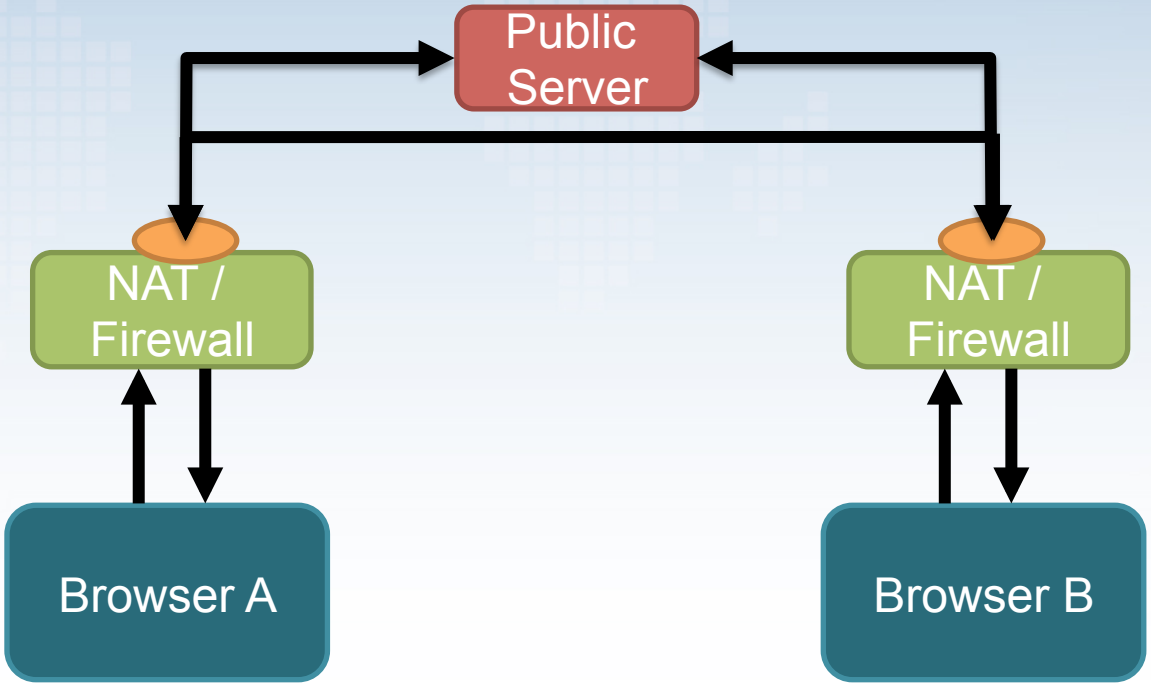
```
v=0
o=- 20518 0 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS ma
a=group:BUNDLE audio
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8
c=IN IP4 24.23.204.141
a=mid:audio
a=msid:ma ta
a=rtcp-mux
a=rtcp:54609 IN IP4
24.23.204.141
a=rtpmap:109 opus/48000/2
a=ptime:60
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
…
```

```
…
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=sendrecv
a=setup:actpass
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:
1f:66:79:a8:07
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)

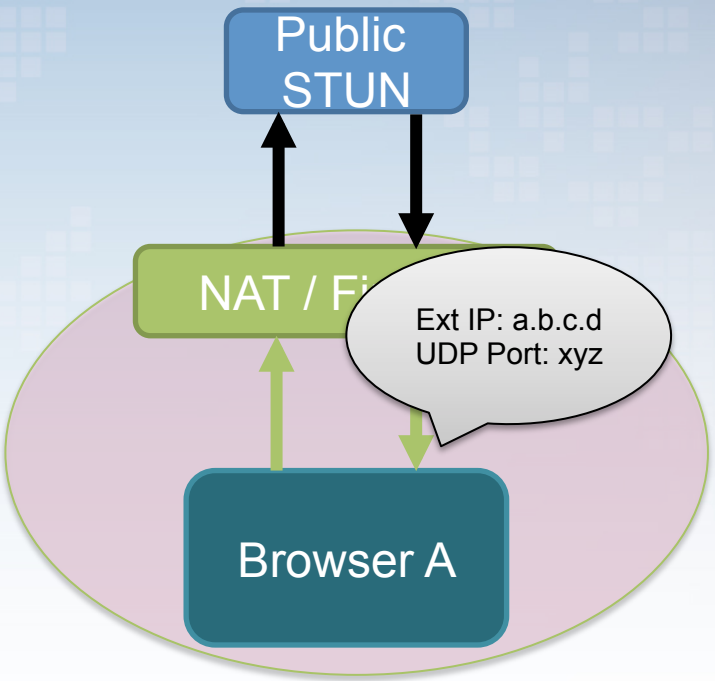


When to use STUN/TURN/ICE?

- 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

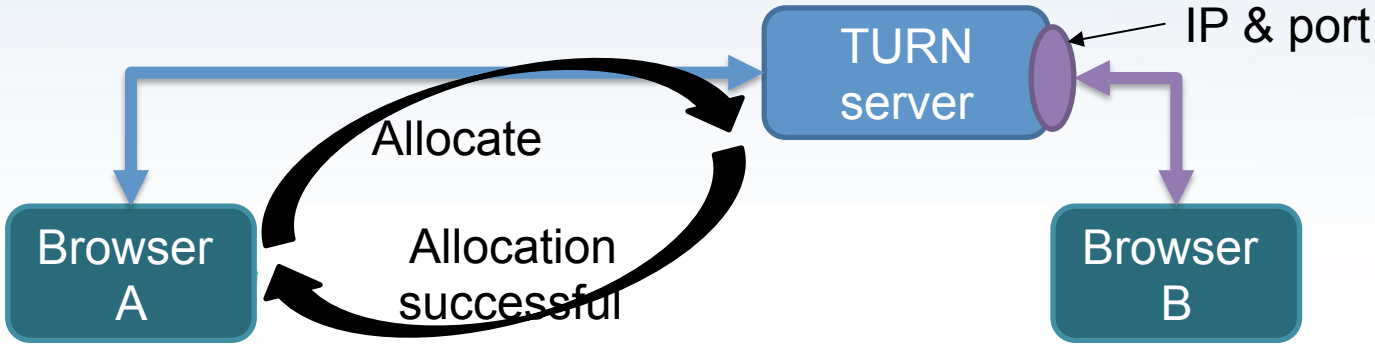
STUN: Session Traversal Utilities for NAT

- Discover your public IP address
- Determine whether your browser sits behind a NAT
- Retrieve the UDP port that NAT has allocated for external communication



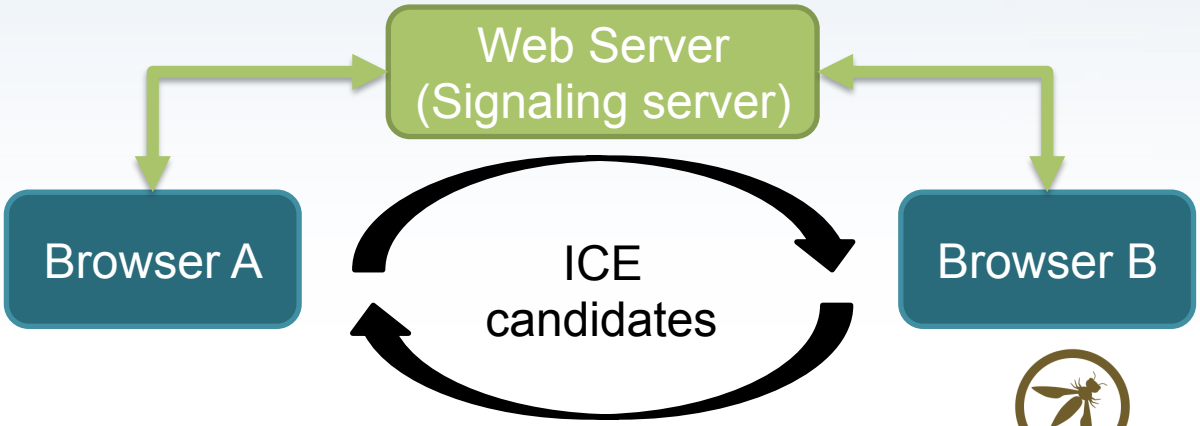
TURN: Traversal Using Relays around NAT

- 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

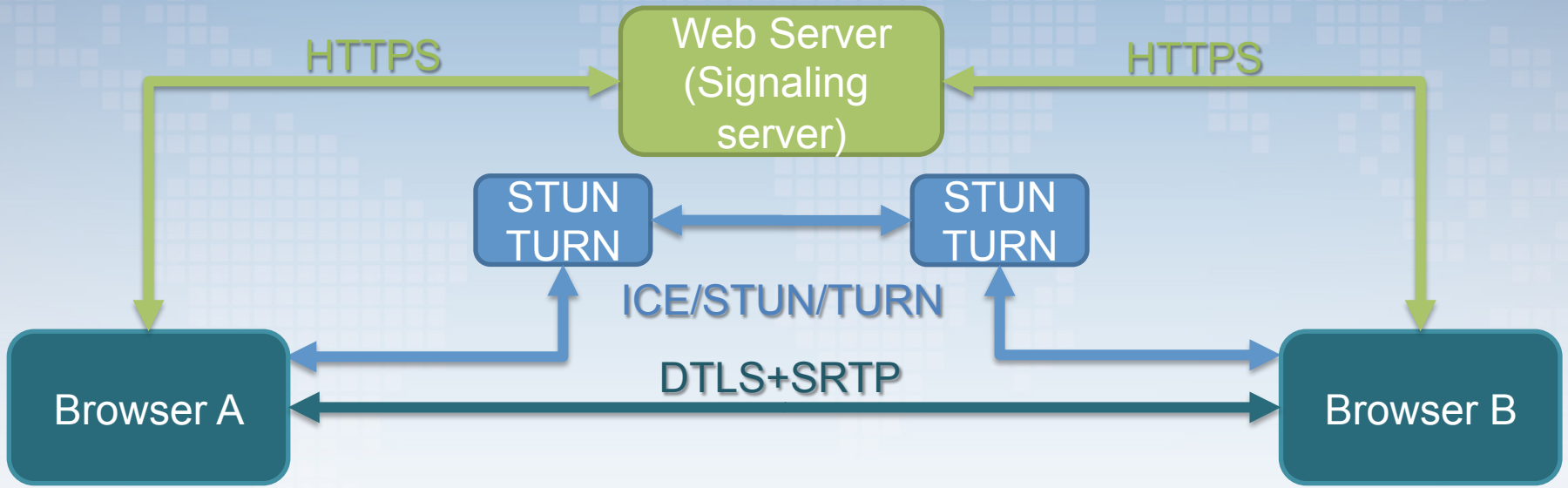


ICE: Interactive Connectivity Establishment

- 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)



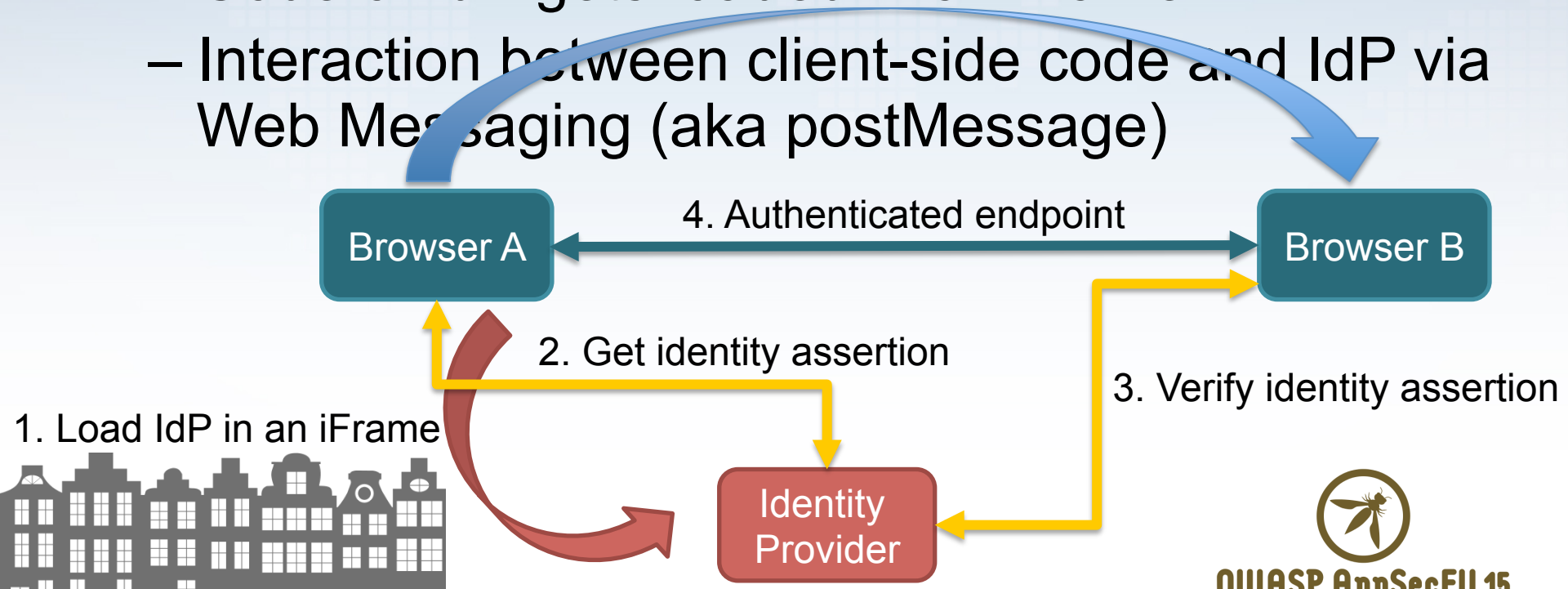
IETF®

W3C®

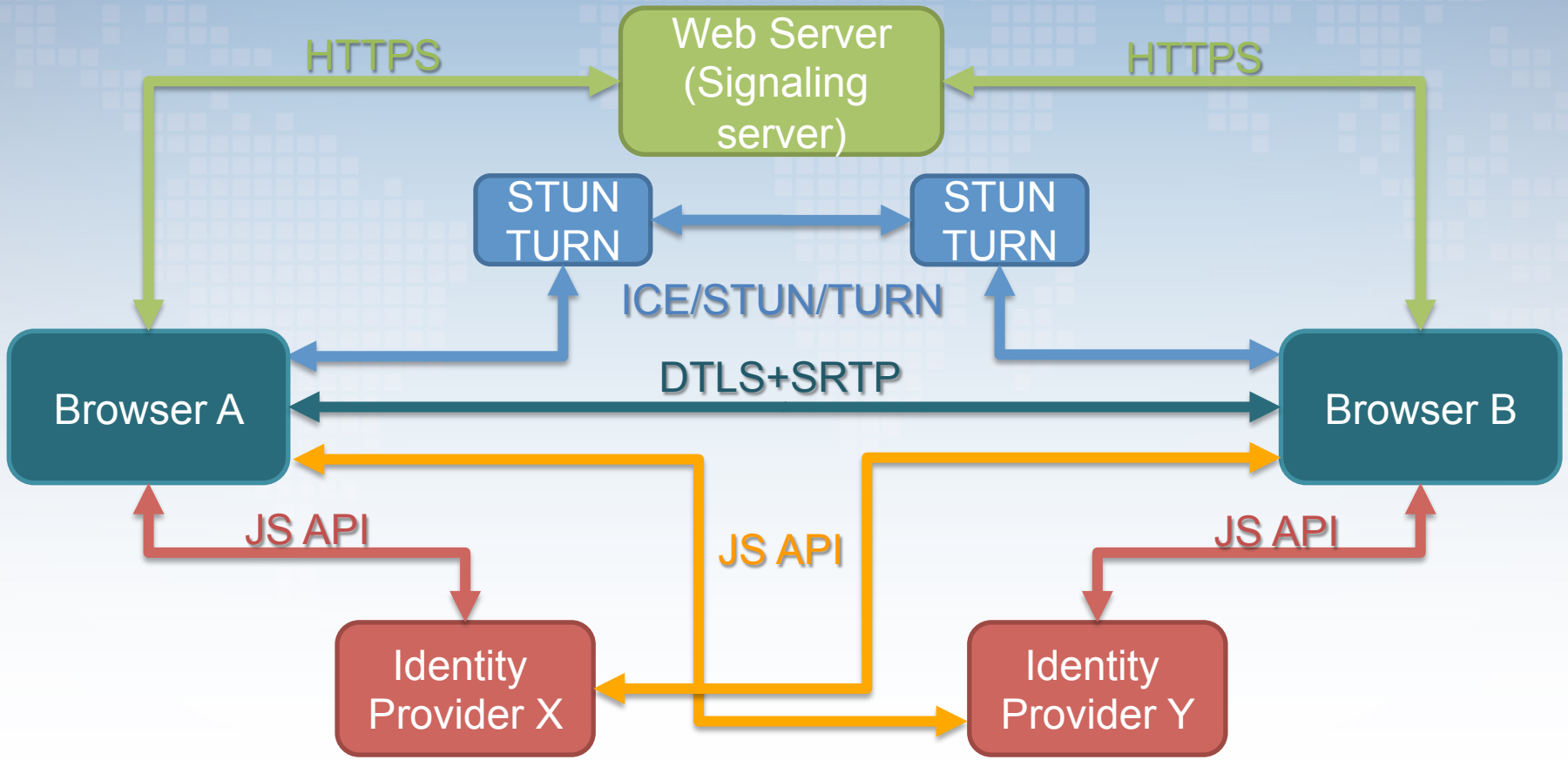
OWASP AppSecEU 15
Amsterdam, The Netherlands

Identity provision

- 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)



WebRTC architecture (full)



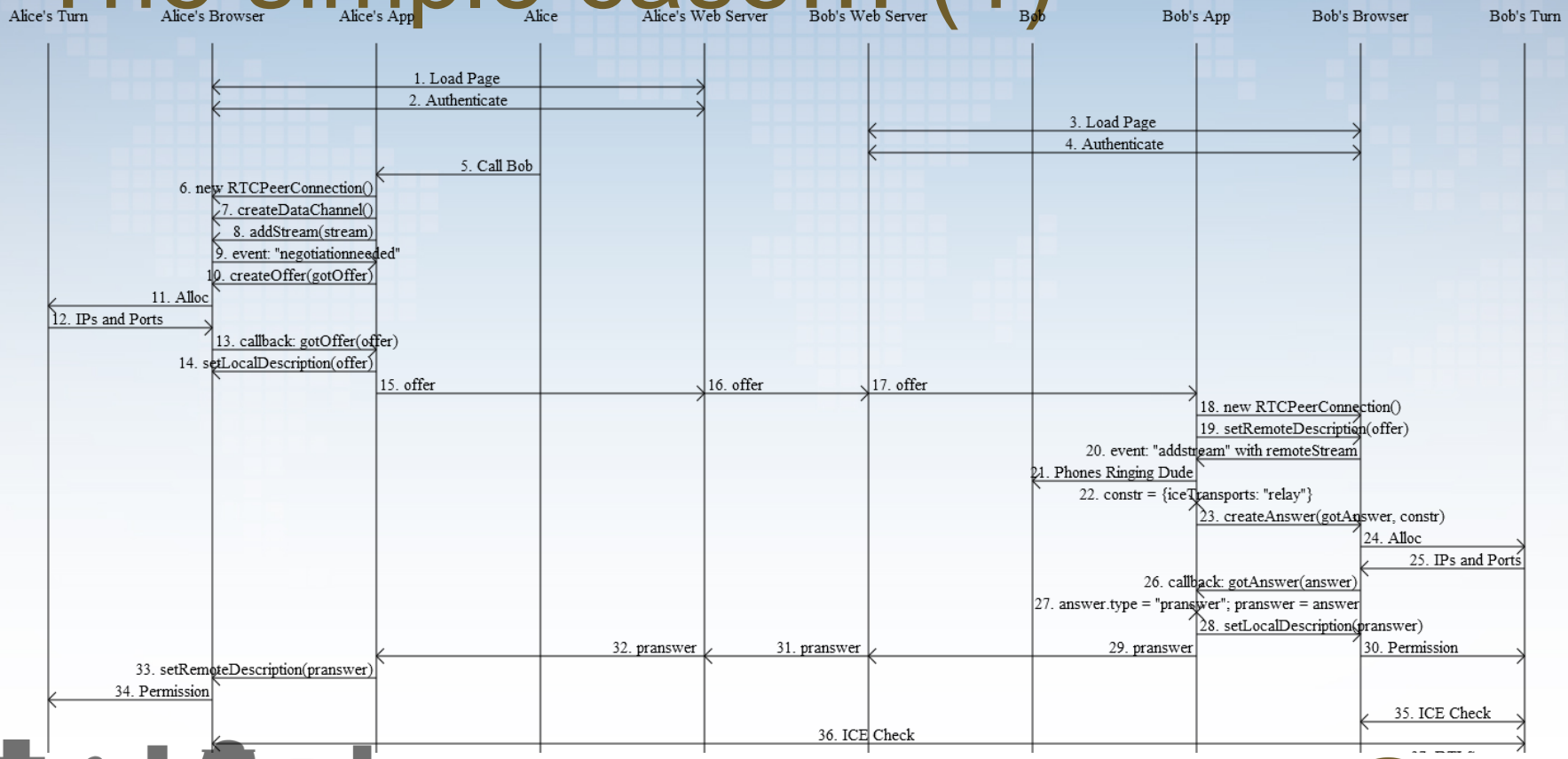
IETF®

W3C®

OWASP AppSecEU 15
Amsterdam, The Netherlands

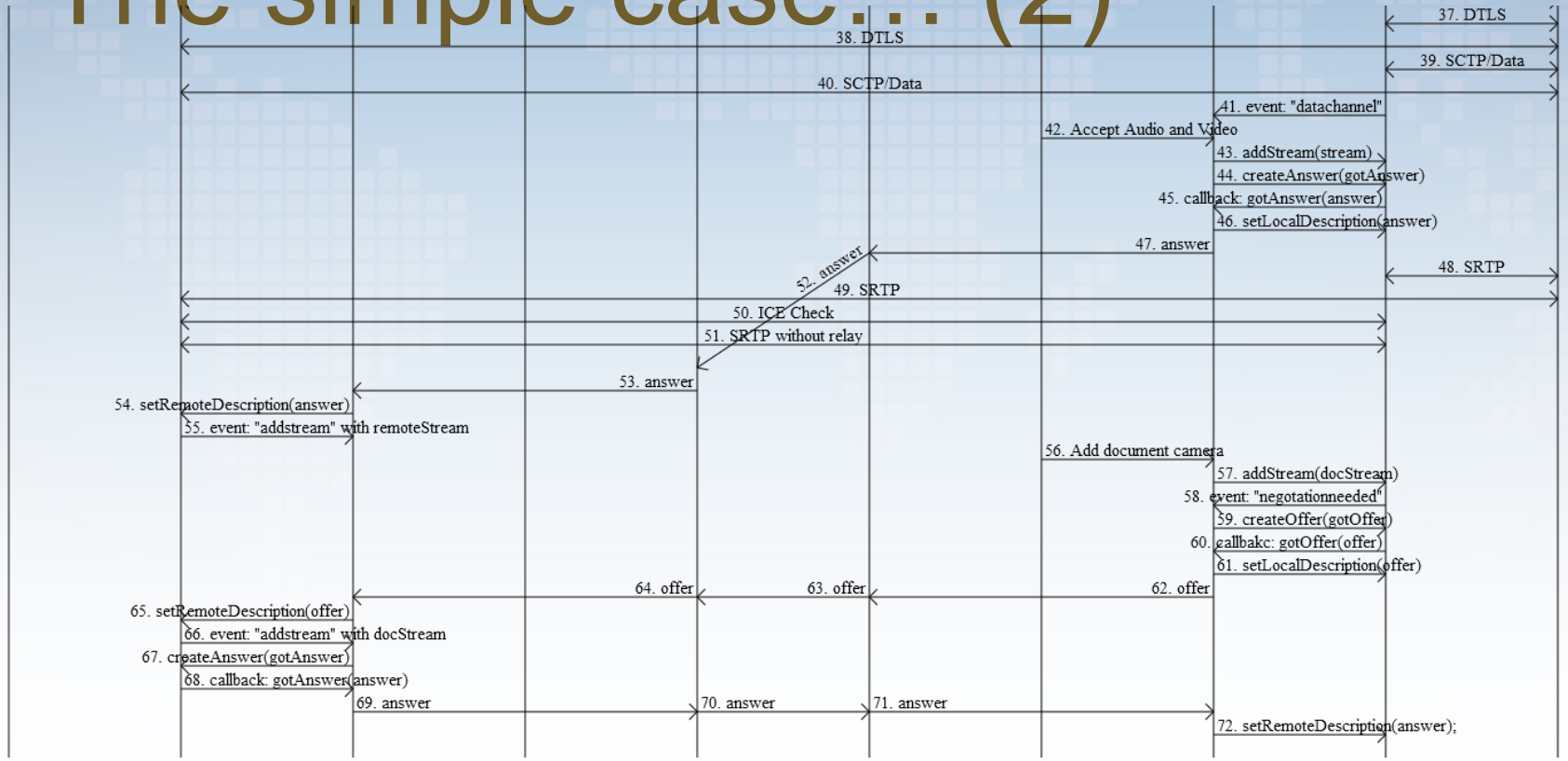
WEBRTC JAVASCRIPT APIS

To give you an idea of the complexity: The simple case… (1)



Source: 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)



Source: Taken from "WebRTC 1.0: Real-time Communication Between Browsers" (W3C Editor's Draft)

Fully JavaScript empowered…

- Purpose of the WebRTC JavaScript APIs
  - Handle A/V
    - 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

```
// 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


```
// 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(
            signalingChannel.send(
```

```
signalingChan
    if(message.s
        var desc =
        // if we get
        peerConne
            return pee
                return peerConnection.setLocalDescription(answer, function () {
                    signalingChannel.send(JSON.stringify({ "sdp":
                        peerConnection.localDescription }));
```

Exchange of information that a connection shall happen.

Application-specific signaling!

Handling ICE Candidates


```
// send any ice candidates to the other peer
peerConnection.onicecandidate = function (evt) {
    if (evt.candidate) {
        signalingChannel.send(JSON.stringif                    vt.candidate }));
    }
};
```

Exchange of information "how" to connect.

```
// receive and pro
signalingChannel
    if(message.can
        peerConnection.addIceCandidate(new RTCIceCandidate(message.candidate));
    }
});
```

Capturing a video stream

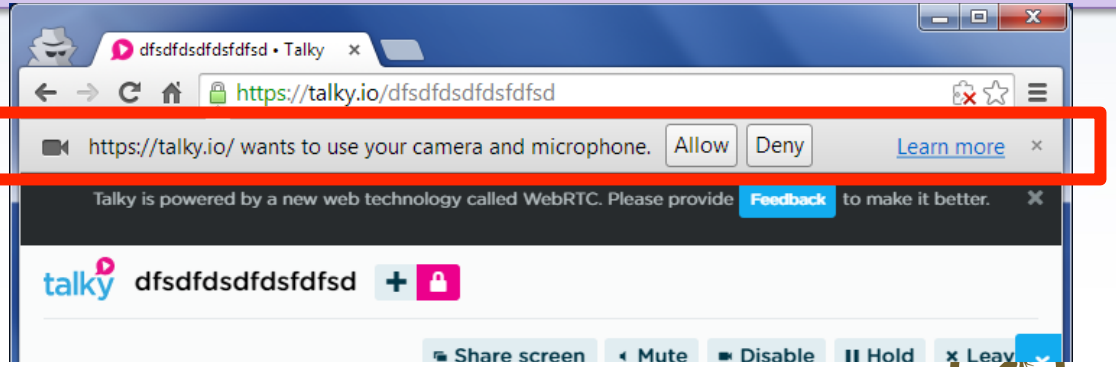
Browser A

Web Server

Browser B

```
// 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 a data channel

```
// 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) { … };
```

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

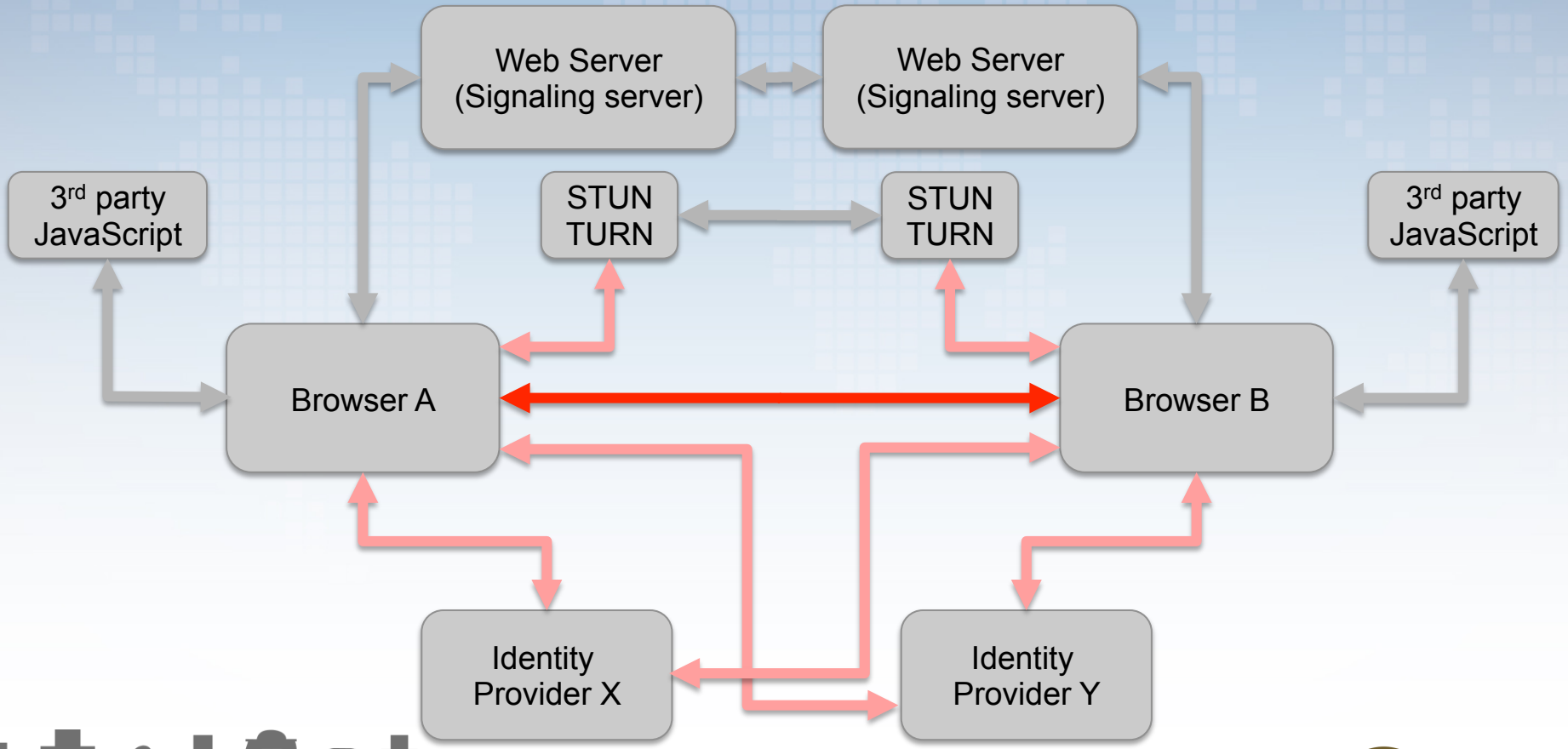
OWASP AppSecEU 15
Amsterdam, The Netherlands

SECURITY & PRIVACY OF WEBRTC

OWASP AppSecEU 15
Amsterdam, The Netherlands

#1 INCREASED ATTACK SURFACE

Increased attack surface



#2 WEBRTC PERMISSION MODEL

OWASP AppSecEU 15
Amsterdam, The Netherlands

Permission model / UI feedback

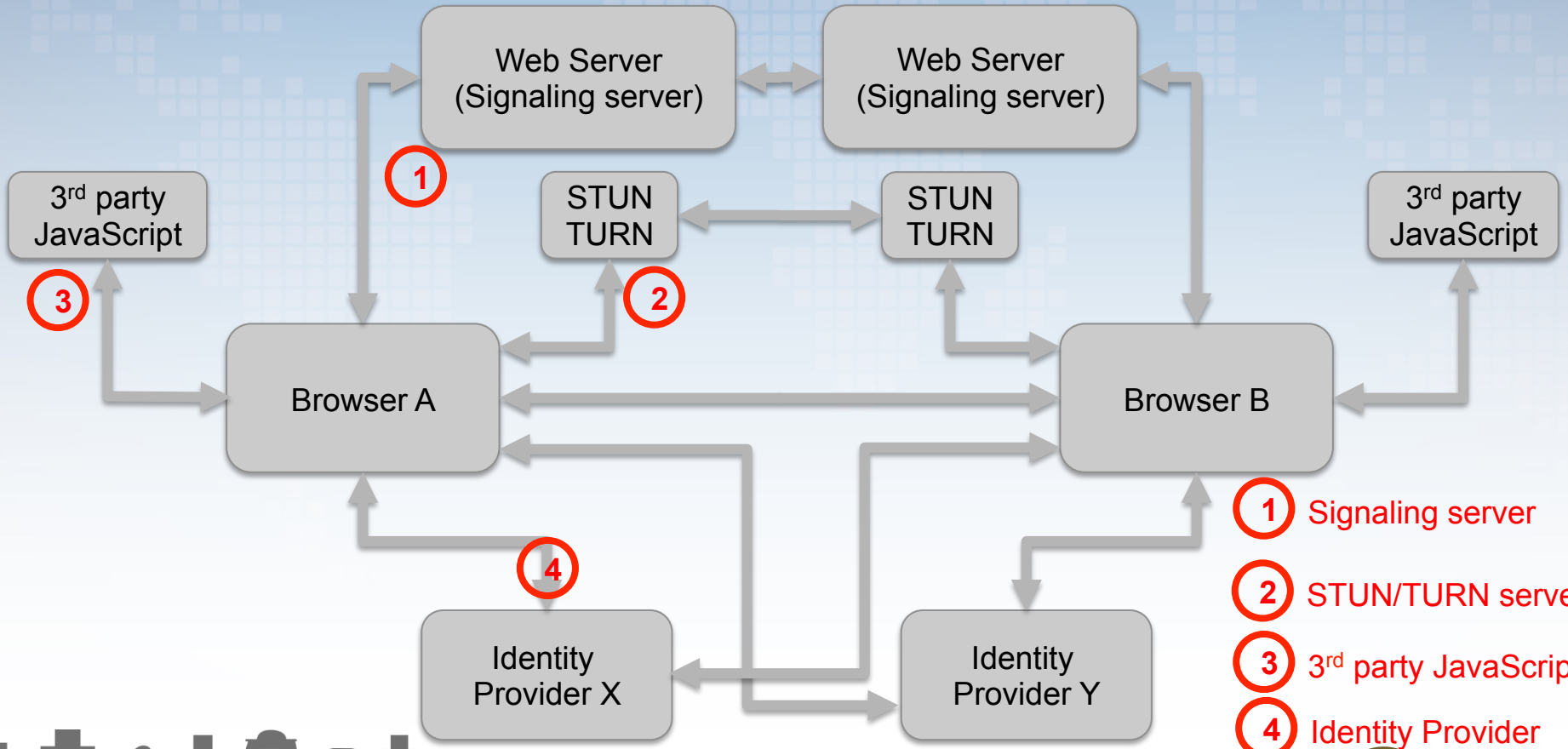
- For which operation, user consent is required?
  - Camera? ✓
  - Microphone? ✓
  - Getting network characteristics (ICE)? ✗
  - Setting up a peer-to-peer connection? ✗
  - Sending your audio/video to a remote peer? ✗
  - Sharing your screen? ✗ ✓
  - Selecting an appropriate Identity Provider? ✗
  - Veryfying your endpoint's identity? ✗

Potential issues due to permission model

- Multiple streams of your camera been sent to different remote endpoints
- 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 META-DATA LEAKS

Meta-data leakage: Trace that communication has happened



OWASP AppSecEU 15
Amsterdam, The Netherlands

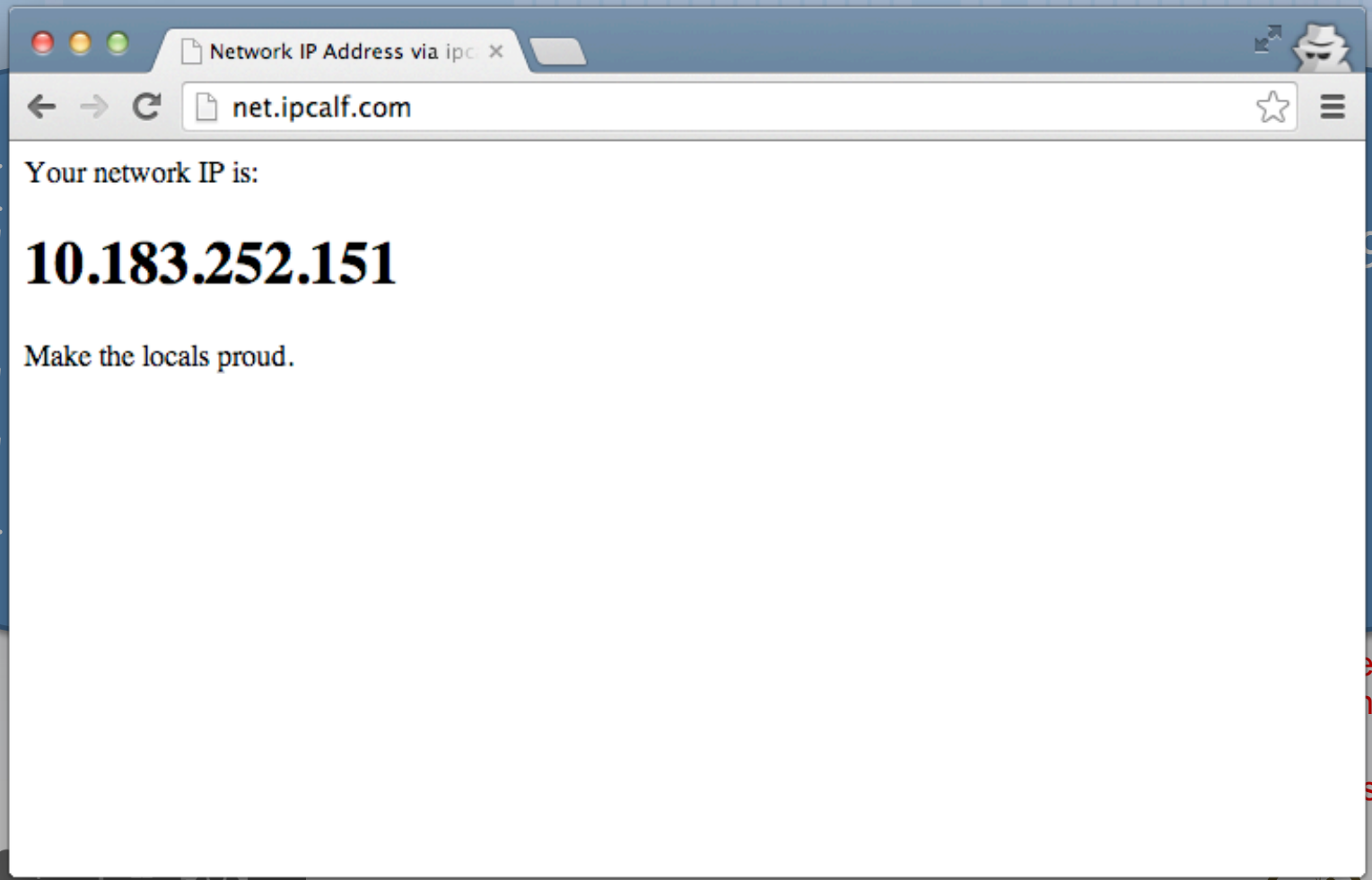
Meta-data leakage

- Leaking the fact that communication has happened between entities:
  - Signaling server
  - STUN/TURN server (*)
  - IdP server (*)
  - 3rd party JavaScript server

(*) Possibly by aggregating data from both end-points

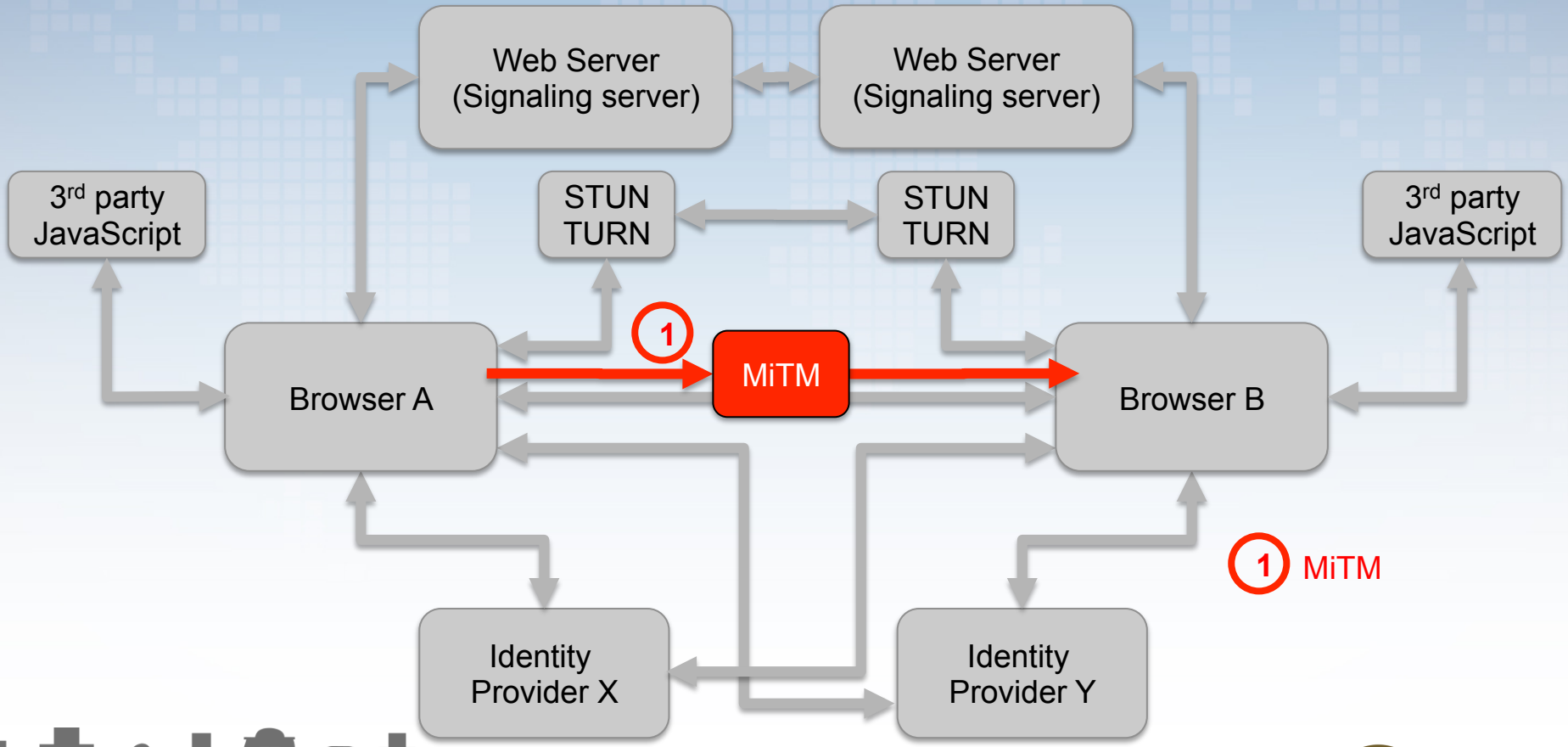
#4 LEAK OF LOCAL NETWORK INFO

Leaking local network info (ICE)



#5 NETWORK ATTACKERS EAVESDROPPING THE CONNECTION

Eavesdropping on the connection



DTLS-RSTP to the rescue

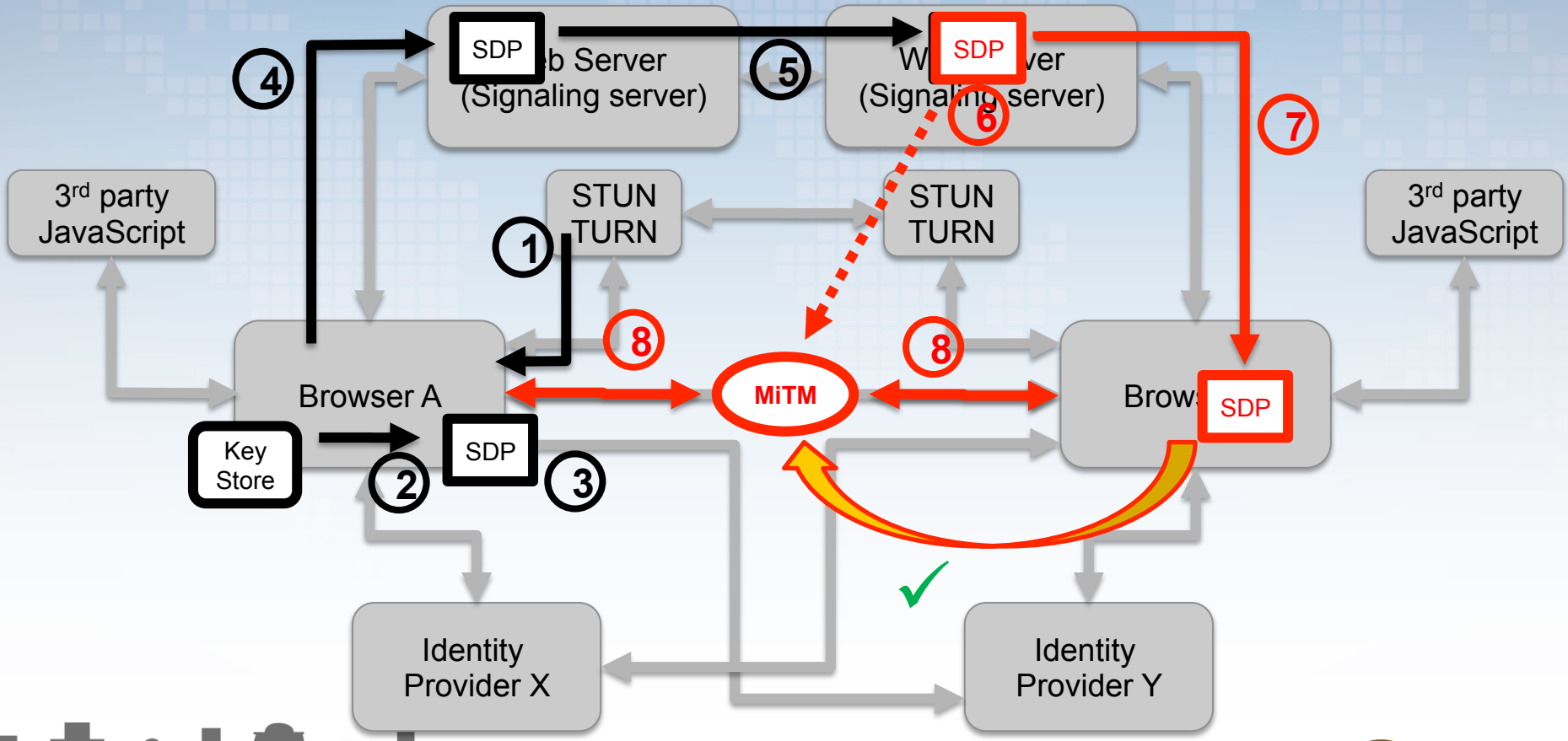
- DTLS provides
  - Encryption
  - Message authenticity
  - Message integrity

- Endpoint's certificate fingerprint is stored as part of the SDP

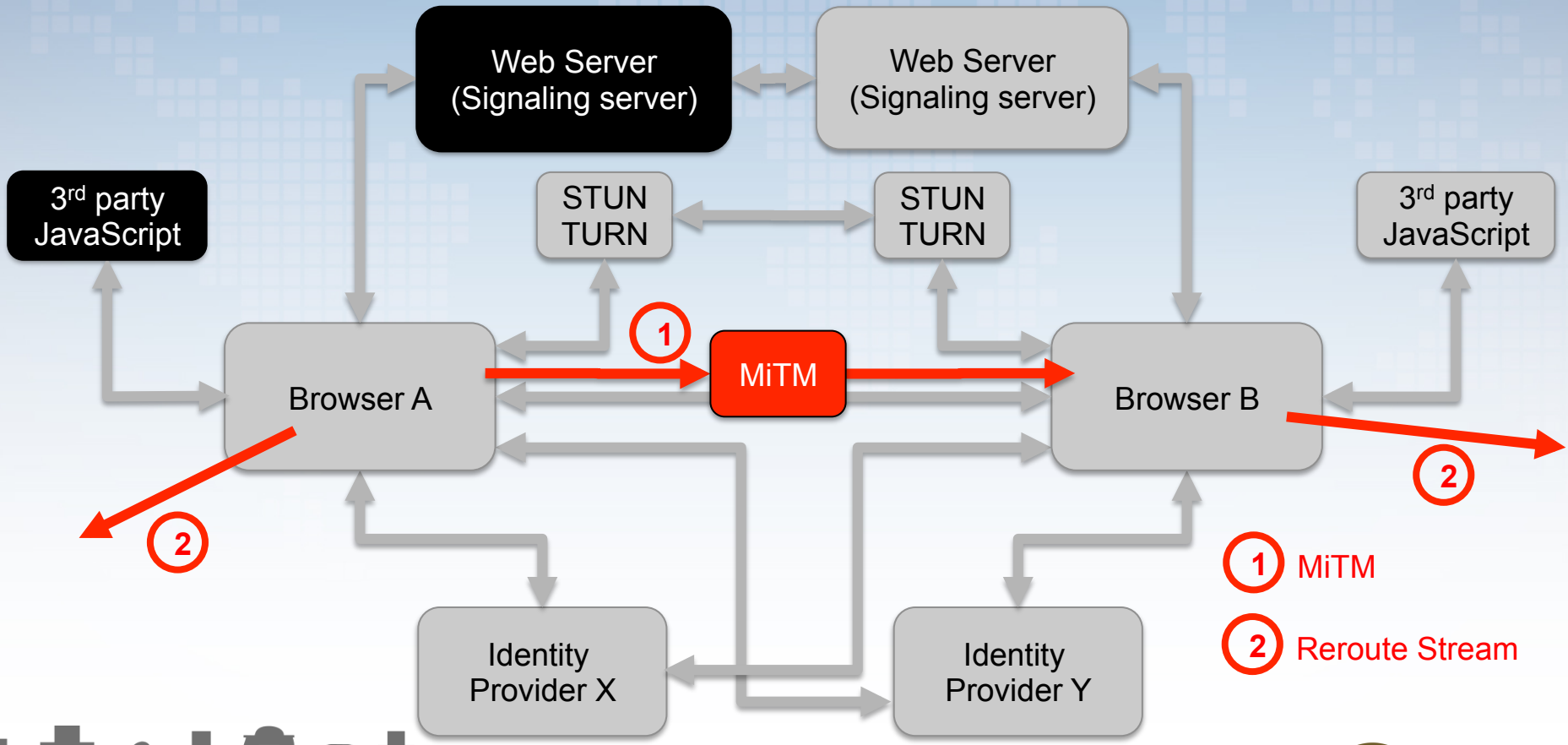
#6 SIGNALING COMPONENTS EAVESDROPPING THE CONNECTION

OWASP AppSecEU 15
Amsterdam, The Netherlands

Setting up the media channel



Eavesdropping on the connection



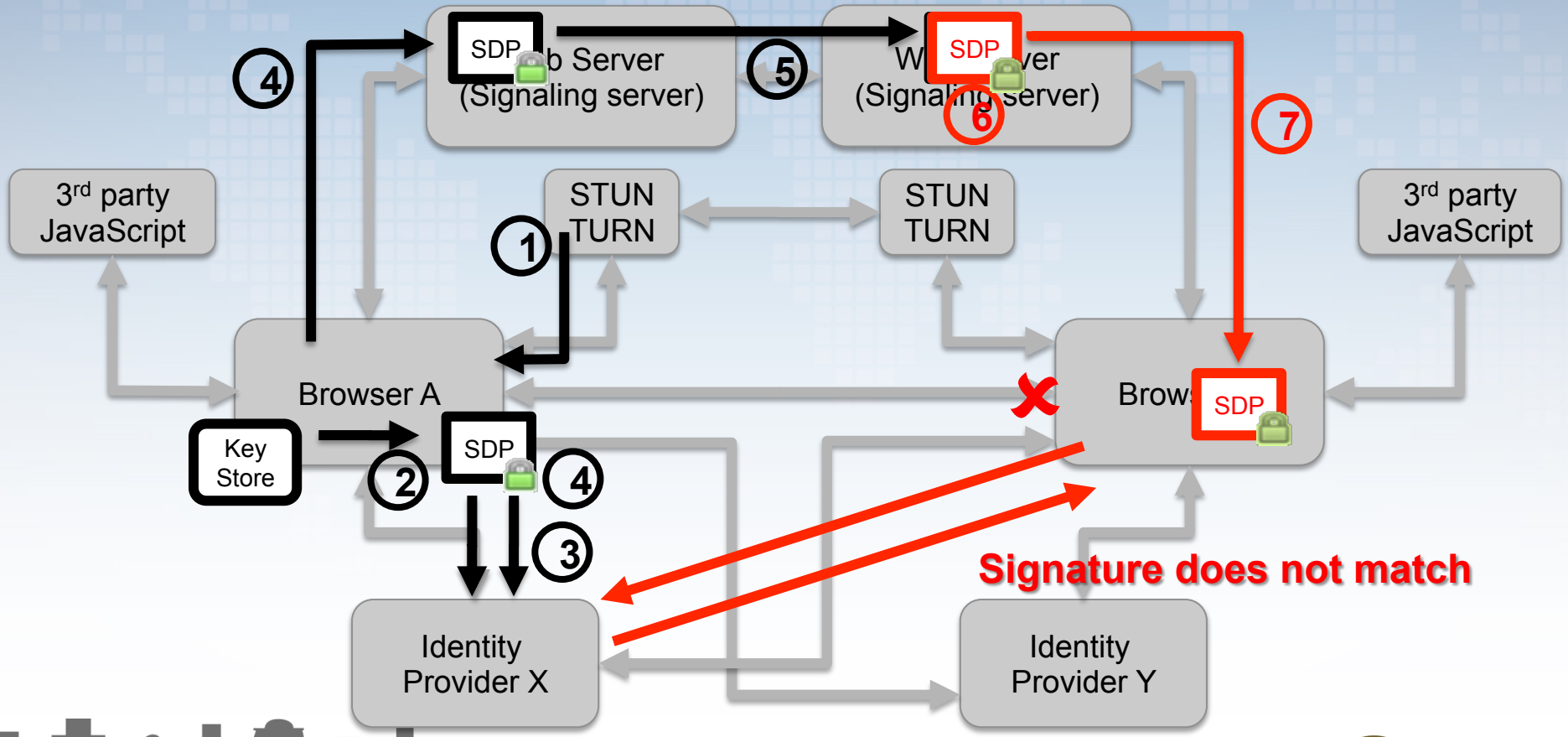
Eavesdropping on the connection

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

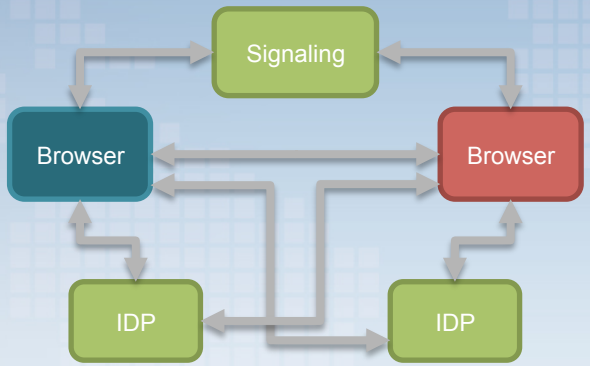
- Can be achieved by:
  - Malicious 3rd party JavaScript (included or via XSS)
  - Malicious signaling server

#7 ENDPOINT AUTHENTICITY

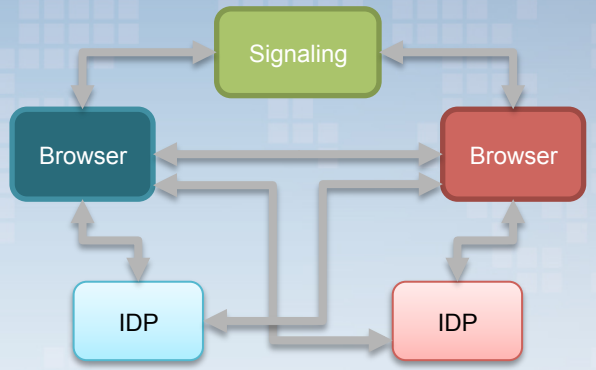
Setting up the media channel with the IdP



IDP setups

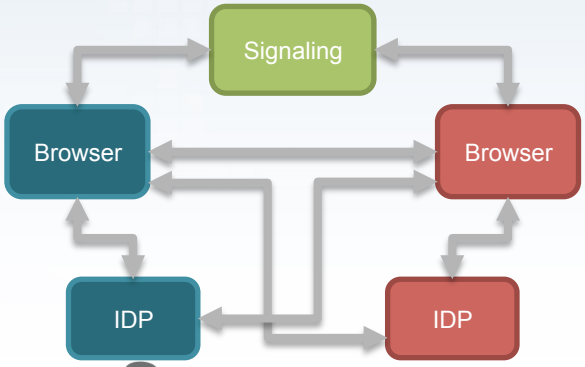


IDP = Signaling Server

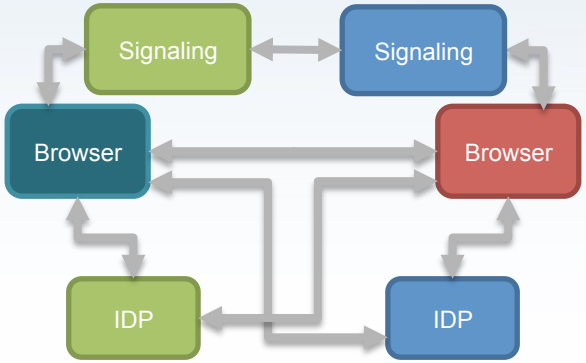


User chooses own IDP

Browser chooses IDP



Signaling server chooses IDP



OWASP AppSecEU 15
Amsterdam, The Netherlands

WRAP-UP

OWASP AppSecEU 15
Amsterdam, The Netherlands

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
- JavaScript running in your application have full control over your WebRTC communication
  - Limit trust in 3rd party JS running in your origin
  - Use best-practices to protect against XSS
- DTLS-SRTP does not authenticate endpoints
  - Use an identity provider to assert the identity of your remote party

- Embrace the new browser capabilities!

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

- Identifying open issues and security challenges for WebRTC
  - Special Issue of IEEE Internet Computing, nov/dec 2014
  - http://www.computer.org/csdl/mags/ic/2014/06/index.html