ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI SCIENZE

Corso di Laurea Triennale in Ingegneria e Scienze Informatiche

ANALISI DELLE PROBLEMATICHE DI SICUREZZA PER LO SVILUPPO DI APPLICAZIONI BASATE SU WEBRTC

Tesi di Laurea in Sistemi Multimediali

Relatore:
Prof.
STEFANO FERRETTI

Presentata da:
FRANCESCO DICARA

Sessione II

Anno Accademico 2015/2016

Indice

Capitolo 1
Introduzione

Dalla sua nascita il World Wide Web è diventato sempre più ricco di contenuti multimediali, prima audio e poi video (con l'esplosione determinata da Youtube) ma è un problema estremamente attuale distribuire questi contenuti attraverso la rete Internet garantendo prestazioni sempre maggiori e qualità sempre più alta. Si è inoltre sviluppata la possibilità di trasmettere contenuti in diretta sia per quanto riguarda le compagnie di telecomunicazioni (televisioni, radio e così via), sia per quanto riguarda il singolo utente che, grazie a servizi come Periscope, YouTube Live o Facebook permette a chiunque di trasmettere live ciò che desidera. Inoltre anche le videochiamate e le videoconferenze in tempo reale sono diventate un tema di grande investimento a partire dal famosissimo Skype fino ai più recenti prodotti di Google Hangout o Duo, soprattutto con il sempre maggior numero di smartphone ad alte prestazione venduti nel mondo. Proprio sull'onda di questi nuovi servizi è stato sviluppato WebRTC, una tecnologia estremamente interessante che nasce con l'intento di permettere una comunicazione peer-to-peer tra due utenti ma che, con il tempo, ha trovato sempre più ambiti di utilizzo grazie alla sua semplicità e flessibilità di applicazione.

Tutte queste funzionalità e allo stesso tempo immediatezza e facilità nello sviluppo pone però un grande problema per quanto riguarda la sicurezza e la privacy degli utenti che utilizzano questi servizi, spesso senza rendersi conto dei rischi a cui vanno incontro. Infatti, un'altra caratteristica che rende così interessante WebRTC, è la sua inclinazione alla sicurezza che lo ha contraddistinto fin dalla sua prima progettazione e che lo pone come standard de facto in tutte le future applicazioni web-based di questo tipo. Inoltre ha ottime

prospettive anche nel campo mobile: basti pensare che l'applicazione per smartphone Messenger di Facebook è basata su WebRTC e viene utilizzata da più di 800 milioni di utenti ogni mese [1].

Questa tesi si propone quindi di studiare tutti i meccanismi e i protocolli che l'architettura di WebRTC impone o propone per garantire la sicurezza nelle comunicazioni multimediali tra gli utenti: nel secondo capitolo vengono riassunte le principali definizioni e i principali problemi di sicurezza che si devono affrontare all'interno di internet e nello sviluppo di applicazioni web-based; nel terzo capitolo viene invece descritta l'architettura del funzionamento di WebRTC soffermandosi sulle principali API e protocolli utilizzati; nel quarto capitolo, in cui è presente il fulcro della tesi, è presente un analisi di come WebRTC affronta il tema della sicurezza e quali sono i suoi punti di forza e di debolezza e come potrebbero essere sfruttati; infine il quinto capitolo trae le considerazioni finali.

Capitolo 2
La sicurezza sulla rete

Questo capitolo si propone di effettuare una panoramica di come la rete Internet e soprattutto il web e le sue applicazioni necessitino di misure di sicurezza e perché sia un argomento di fondamentale importanza nel mondo odierno.

2.1 Perché è importante la sicurezza su Internet

La sicurezza informatica è un campo molto vasto che comprende principalmente la sicurezza dei dati, dei programmi e delle comunicazioni. Col passare degli anni ha acquisito sempre più valore e importanza poiché sempre più servizi e applicazioni della vita di tutti i giorni si sono spostate sui calcolatori e, con la diffusione di Internet, sul Web. Ogni azienda, anche estremamente piccola, ormai possiede un database tramite cui gestire i suoi prodotti o i suoi dipendenti, chiunque utilizza applicazioni che sfruttano Internet per rimanere in contatto con i propri conoscenti, tutti i servizi bancari e finanziari garantiscono l'accesso tramite la rete e così via. Non è difficile immaginare la quantità infinita di informazioni sensibili che viaggiano ogni giorno sulla rete considerando che gli utenti internet sono 3 miliardi e mezzo (la metà della popolazione mondiale) e circa il 40% del traffico avviene verso dispositivi mobili (generalmente i più utilizzati dal singolo utente piuttosto che da aziende) [2]. La maggior parte di queste informazioni inoltre è estremamente sensibile sia dal punto di vista della privacy che dell'importanza dei dati.

Il motivo per cui la sicurezza su Internet sia fondamentale è quindi molto chiaro ma, per capire perché non è un fatto scontata, bisogna fare un breve excursus storico: al momento

della sua creazione Internet era una piccola rete tra poche decine di calcolatori accessibili solo ad utenti di cui si conosceva per certo l'identità. Non era quindi necessario adottare misure di sicurezza poiché nessuno che non fosse autorizzato aveva modo di accedere alla rete: infatti, i protocolli di base che ancora oggi sono alla base della comunicazione non prevedono alcun tipo di criptazione dei dati né nessun'altra misura di sicurezza. Tutto ciò che prova a rendere sicura la rete è stato aggiunto e si è diffuso su larga scala soprattutto negli ultimi anni. Criptare i dati però non basta: anche un servizio malevolo potrebbe essere in grado di farlo; quindi l'obbiettivo dei protocolli sicuri odierni è cercare di raggiungere lo stesso obiettivo di fiducia reciproca tra gli utenti come al momento della nascita di Internet utilizzando, ad esempio, il meccanismo dei certificati.

2.2 Come funziona la rete Internet

Internet può essere definito come una rete che fornisce servizi e applicazioni accessibile al pubblico che si basa su un'architettura logica a livelli [3]. Secondo questa architettura ogni livello è in grado di comunicare solo con i livelli appena superiore e inferiore. Inoltre ogni livello prevede una serie di protocolli che possono trattare solo le informazioni riguardanti quel livello (per esempio il protocollo di rete può accedere solo agli indirizzi IP del destinatario e del mittente all'interno di un pacchetto). Questi livelli con i rispettivi protocolli formano quella che è definita pila protocollare: il modello di riferimento per la rete Internet è quello proposto dall'ISO a partire dal 1978 con il nome di Open Systems Interconnection (OSI) e prevede 7 livelli ma, come mostrato in figura 1, vi sono cinque livelli previsti nell'architettura attuale di Internet [4]:

- Application layer: è il livello a cui si trovano le applicazioni di rete e i loro protocolli. È proprio a questo livello che è possibile sviluppare più servizi con le caratteristiche necessarie (e come vedremo è principalmente a questo livello che troviamo i protocolli usati da WebRTC). Inoltre è il livello su cui è più facile lavorare per quanto riguarda la sicurezza.
- Transport layer: questo livello incapsula i contenuti che arrivano dal livello di applicazione e i due protocolli utilizzati nella quasi totalità dei casi sono UDP (User Datagram Protocol) e TCP (trasmission Control Protocol). Ognuno dei due protocolli garantisce caratteristiche diverse che influenzano sia

l'affidabilità che la velocità dei dati trasportati e allo stesso tempo anche la loro sicurezza.

- Internet Layer: il protocollo più usato a questo livello è IP (Internet Protocol) che permette la comunicazione logica fra host non direttamente connessi che tramite instradamento e indirizzamento riescono a scambiarsi dati.
- Link layer: permette di inoltrare i datagrammi del livello superiore ad host che si trovano tra l'origine e la destinazione.
- Physical layer: trasferisce i singoli bit di ogni pacchetto tra i nodi che compongono la rete.

Layer #	Layer Name	Protocol	Protocol Data Unit	Addressing
5	Application	HTTP, SMTP, etc…	Messages	n/a
4	Transport	TCP/UDP	Segments/ Datagrams	Port #s
3	Network or Internet	IP	Packets	IP Address
2	Data Link	Ethernet, Wi-Fi	Frames	MAC Address
1	Physical	10 Base T, 802.11	Bits	n/a

Figura 1 Tabella riepilogativa dei cinque livelli base dell'architettura su cui si basa la rete Internet.

2.3 Caratteristiche della sicurezza informatica

Ognuno dei cinque livelli della pila protocollare prevede differenti tecniche e protocolli con cui cercare di rendere sicuri i dati trasportati ma sono richieste alcune caratteristiche generali per garantire una comunicazione sicura.

L'idea che sta alla base è quella di ricreare l'ambiente in cui è nato Internet e quindi trovare un modo di fidarsi dell'identità della persona con cui si sta entrando in contatto. In questo caso con persona si intende anche un server, un router o qualunque altro soggetto presente nella rete e ovviamente lo stesso discorso vale anche nelle comunicazioni tra due server o due router.

Il "NIST Computer Security Handbook" definisce la computer security in questo modo: *"Computer security is the protection afforded to an automated information system in*

order to attain the applicable objectives of preserving the integrity, availability, and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).” [5]

Questa definizione introduce i concetti alla base di qualunque tecnologia di sicurezza informatica:

- Integrità del messaggio;
- Disponibilità;
- Riservatezza.

A queste tre principali va però sicuramente aggiunta l’autenticazione che garantisce alla base anche parte delle caratteristiche appena citate.

2.3.1 Integrità del messaggio

Con integrità del messaggio si intende il fatto che qualunque messaggio che viene inviato da punto a un altro della rete arrivi intatto, cioè esattamente come è stato mandato. Ovviamente siccome un messaggio non è altro che una sequenza di bit può succedere a qualunque livello della pila protocollare che per qualche motivo avvenga un errore durante la lettura del messaggio o durante l’inoltro di un router al nodo successivo. La maniera migliore di controllare l’integrità di un messaggio all’arrivo è aggiungere all’header del pacchetto un checksum.

Un checksum consiste in genere in una sequenza di bit associata al pacchetto. Le tecniche di checksum possono essere molto complesse e devono garantire un ottimo rapporto tra lunghezza del checksum stesso e garanzia dell’integrità dei dati. Una tecnica estremamente semplice può consistere nel sommare tutti i bit facenti parte di quel pacchetto: la somma viene eseguita nel calcolatore che invia il messaggio, il checksum viene salvato nell’header e quando viene ricevuto dal destinatario viene rieseguita la stessa operazione controllando che il valore contenuto nell’header del messaggio e quello appena calcolato coincidano. Anche con tecniche più complesse basate su algoritmi di hash crittografati, la meccanica di base rimane comunque la stessa [4].

Molti protocolli di rete a tutti i livelli utilizzano checksum per verificare l’integrità dei dati; tra questi si possono citare a livello di trasporto UDP, TCP o a livello di rete IPv4.

2.3.2 Disponibilità

Con disponibilità si intende che il contenuto in rete deve essere sempre disponibile a scapito di qualunque tipo di attacco: questa caratteristica si pone quindi l'obiettivo di evitare che azioni di disturbo possano compromettere la disponibilità del sistema. Questo ruolo è solitamente svolto da firewall e da sistemi per rilevare eventuali intrusioni.

Un firewall può essere definito come una collezione di componenti inserita tra due reti e deve essere sviluppato secondo tre principali obbiettivi [6]:

- Tutto il traffico dall'interno all'esterno e viceversa deve passare attraverso il firewall. In questo modo il firewall garantisce di essere un vero e proprio filtro per ogni pacchetto della rete in transito.
- Solo il traffico autorizzato secondo la policy locale deve essere lasciato passare e quindi ogni messaggio che non rispetta le regole imposte deve essere bloccato.
- Infine il firewall deve essere immune a essere attaccato dall'esterno e quindi deve basarsi su un sistema operativo sicuro ed essere sempre aggiornato.

Un firewall quindi può, ad esempio, chiudere determinate porte per evitare che alcuni servizi vengano utilizzati, oppure bloccare alcuni indirizzi IP o viceversa permettere solo ad alcuni di mandare pacchetti verso la rete che si sta tentando di proteggere o magari permettere di mandare pacchetti solo verso alcuni indirizzi IP.

Siccome è praticamente impossibile costruire una struttura completamente impenetrabile bisogna anche essere in grado di rilevare eventuali intrusioni al proprio sistema prima che sia troppo tardi. Per fare questo esistono tecniche di analisi del pacchetto che controllano in profondità il contenuto o che analizzano se la firma digitale allegata è originale o meno. Esistono anche sistemi in grado di accorgersi se il flusso in entrata o uscita è fuori dal normale (magari a causa di un attacco DoS) e avvertono l'amministratore del sistema.

2.3.3 Riservatezza

Quando si parla di riservatezza sulla rete ci si riferisce al fatto che solo il destinatario e il mittente del messaggio devono essere in grado di accedere liberamente al contenuto. La

riservatezza è indissolubilmente legata al tema della crittografia dei dati e tutti i protocolli che implementano funzioni per rendere i dati riservati utilizzano tecniche di cifratura.

I più noti algoritmi di cifratura sono di dominio pubblico ma quello che rende possibile la decifratura dei bit è la chiave privata che, invece, è differente per ogni sessione tra due soggetti. Esistono vari tipi di crittografia; quello più utilizzato su internet è definito a chiave pubblica e il più noto è RSA sviluppato da Ronald Rivest, Adi Shamir e Leonard Adleman per la prima volta nel 1977 [7]. Il funzionamento è piuttosto semplice ma molto interessante e fondamentale anche per capire successivamente su quali basi WebRTC fonda una parte della sua sicurezza: il destinatario del messaggio possiede due chiavi (una pubblica accessibile a chiunque e una privata che conosce soltanto lui), il mittente del messaggio quindi codifica i dati tramite un algoritmo standard e la chiave pubblica; una volta che il messaggio arriva al mittente, quest'ultimo potrà decifrarlo solo utilizzando la sua chiave privata; a questo punto bisognerà quindi evitare che la chiave privata possa essere rubata e che nessuno possa riuscire a mandare un messaggio fingendo di avere un'altra identità [4]. Queste tematiche sono legate anche alla firma digitale e al non ripudio dei dati inviati (cioè una volta che un soggetto invia un messaggio sulla rete non deve avere modo di negare che li ha inviati). Il protocollo attualmente più noto che utilizza questo meccanismo è TLS (Transport Layer Security, standardizzato nella RFC 4346), alla base di HTTPS di qualunque altra applicazione che faccia uso di TCP come protocollo a livello di trasporto. Esiste anche la versione per UDP, DTLS, che verrà approfondita parlando di WebRTC.

2.3.4 Autenticazione

L'ultima caratteristica per la sicurezza in rete riguarda l'autenticazione. Si tratta del processo con cui si può confermare l'identità di una persona che sta provando ad accedere a un sistema o di confermare l'autenticità di un messaggio [8]. L'autenticazione è necessariamente la prima fase all'inizio di una comunicazione e si basa sulla crittografia e l'integrità del messaggio. Non deve essere possibile per nessuno leggere le informazioni utilizzate per autenticarsi (anche se vengono intercettate) né tantomeno fingersi qualcun altro o alterare il messaggio iniziale.

Non è difficile immaginare quanto sia importante l'autenticazione all'interno di applicazioni come WebRTC e infatti sarà un tema molto importante nel capitolo 4.

2.4 Gli attacchi informatici

Basta partire dai dati sulla sicurezza in internet nel rapporto 2016 della Symantec per capire quanto ancora oggi la sicurezza in rete sia un tema che viene spesso sottovalutato: per esempio, nel 2015 sono state scoperte 54 nuove 0-day vulnerability (cioè falle nella sicurezza che venivano già utilizzate per attacchi o frodi informatiche) con un aumento di più del doppio rispetto all'anno precedente, mezzo milione di dati personali persi oppure rubati, tre quarti dei siti web presentano vulnerabilità e altri molti altri dati non fanno ancora sperare per i dovuti cambiamenti [9].

Un aspetto particolarmente interessante degli attacchi e delle frodi informatiche è che spesso partono da errori umani nel seguire protocolli o indicazioni o, semplicemente, dalla mancanza di informazione riguardo ai problemi. Un esempio molto famoso riguardo a un attacco che è stato possibile prima di tutto grazie a un errore umano è Stuxnet: si tratta di un worm sviluppato dal governo degli Stati Uniti per colpire le centrali nucleari iraniane; all'interno delle centrali sono presenti particolari macchinari (PLC) utilizzati per le centrifughe controllati da un computer. I computer sono disconnessi dalla rete Internet proprio come misura di sicurezza ma, tramite una chiavetta USB infetta, il virus è riuscito ad attaccare un primo calcolatore e, tramite la rete locale tra i computer delle centrali nucleare iraniane, ha poi infettato quasi tutte le altre. Inoltre, successivamente tramite alcuni componenti infetti, il virus si è diffuso in parecchi paesi del mondo diventando quasi incontrollabile. In questo caso l'errore umano è stato il fattore predominante nella diffusione di questo virus. Questo evento ha aperto le porte ha ulteriori discussioni e considerazioni di sicurezza anche in ambienti delicati e strategici a livello governativo come l'industria energetica [10]. Se è risultato così facile attaccare luoghi che dovrebbero essere tra i più protetti al mondo, quanto è facile attaccare un dispositivo alla portata di tutti?

Solitamente gli attacchi vengono divisi in due grandi gruppi: passivi e attivi [11]. I primi consistono nel monitorare le trasmissioni per ottenere le informazioni che vengono trasmesse secondo lo schema in figura 2. L'attaccante in questo caso potrebbe leggere il contenuto dei messaggi scambiati oppure solo analizzare il traffico perché, magari, risulta criptato. Gli attacchi passivi sono molto difficili da scoprire poiché il traffico tra i due soggetti che si stanno scambiando informazioni procede senza alcuna differenza dalla normalità.

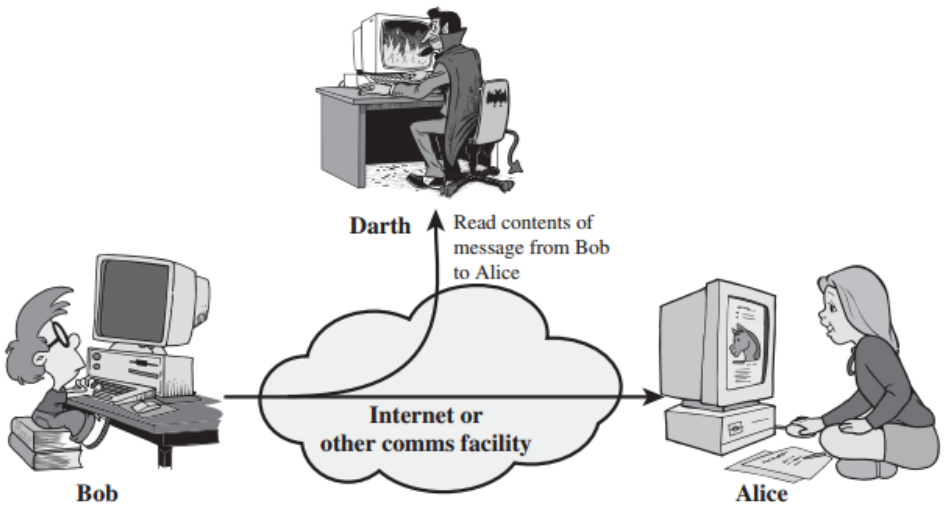

Figura 2 Schema esemplificativo di un attacco di tipo passivo [11]

Gli attacchi di tipo attivo invece consistono in qualche modifica del flusso dei dati o addirittura nella creazione di un flusso di dati con false credenziali o ancora nell'impedire che un soggetto sia raggiungibile o possa inviare informazioni come nel caso di attacchi DoS.

Nei prossimi paragrafi vengono analizzati alcuni tipi di attacco che, nel corso del capitolo 4, verranno ripresi legati alle possibili vulnerabilità di WebRTC e come la sua architettura sia studiata per cercare di limitarli.

2.4.1 Malware

I malicious software, o malware, sono programmi studiati con l'intento di essere inclusi o inseriti in un sistema con uno scopo malevolo. Esistono decine di tipologie di malware ed è al di fuori dello scopo di questa tesi approfondire in dettaglio ognuna di esse, quindi ne vengono presentate solo alcune con una breve descrizione [11].

- Virus: malaware che, quando eseguito, prova a replicarsi all'interno di un codice eseguibile; quando ha successo il codice viene definito infetto e il virus viene eseguito insieme ad esso.
- Worm: software che viene eseguito in maniera indipendente e si può propagare in altri dispositivi su una rete (l'esempio di Stuxnet).

- Trojan horse: software che viene considerato utile ma che all'interno contiene anche del codice malevolo in grado di sfruttare le autorizzazioni legittime date al software principale.
- Keylogger: software in grado di catturare e mandare all'attaccante tutte le lettere che vengono digitate tramite la tastiera del computer infetto.

Piuttosto che analizzarsi nello specifico uno ad uno è interessante capire come questi software possano essere introdotti all'interno del dispositivo che l'utente utilizza tramite la rete. Sicuramente la maniera più semplice è scaricando plug-in sconosciuti o scaricando software da fonti non attendibili. In questi casi infatti il software potrebbe essere stato manomesso o nascondere al suo interno un Trojan. Un altro modo può essere tramite un attacco che intercetta la comunicazione e scambia i pacchetti a cui siamo interessati con altri infetti. Queste tecniche sfruttano meccanismi che, essendo tutti attuabili sul web, possono interessare anche applicazioni sviluppate con WebRTC.

2.4.2 Denial of Service e Distributed Denial of Service

Quando ci si riferisce a un attacco DoS (Denial of Service) si parla di un tentativo di impedire ai legittimi utenti di un servizio di accedere al servizio stesso. Se l'attacco proviene da un unico host si parla di DoS, ma, ormai sempre più spesso, si parla di Distributed DoS (DDoS). Nel caso dei DDoS l'attacco proviene da un numero molto maggiore di host, spesso inconsapevoli.

Un'attacco DoS può essere eseguito sfruttando le vulnerabilità presenti in protocolli o applicazioni usati dalla vittima inviando dati malformati. Questi pacchetti portano la macchina a utilizzare in maniera eccessiva la CPU, a riavviare il dispositivo e in generale ad un rallentamento del sistema. Un altro modo per compiere un attacco di questo tipo è tramite un flooding attack: in questo caso l'obiettivo è sovraccaricare la macchina della vittima inviando una quantità più alta possibile di dati. Con gli attacchi di tipo flooding non è necessario che ci siano vulnerabilità nel sistema perché, semplicemente, risulta impossibile gestire una tale quantità di dati, anche solo a livello hardware. È proprio in attacchi basati sul flooding che vengono generalmente usati più host inconsapevoli generando un attacco DDoS. Infatti sarebbe molto facile individuare un attacco DoS di questo tipo poiché, se un unico utente richiede il consumo di così tanto traffico, può essere subito bloccato il suo indirizzo

IP. L'attaccante può usare svariati protocolli come UDP, TCP, ICMP continuando a ritrasmettere i dati. Un tipico esempio è basato sul flag SYN di TCP: questo flag è utilizzato generalmente per aprire una connessione con l'host destinatario nel three-way handshake tipico di TCP; quando viene richiesta una nuova connessione il server risponde con una richiesta SYN e se il client non risponde la connessione tra i due rimane aperta esponendo il server all'attacco. Questo tipo di attacchi è quasi impossibile da annullare poiché, per definizione, un servizio online deve essere sempre disponibile a qualunque utente ne faccia richiesta [12].

2.4.3 Man in the Middle attack e Hijacking

Il Man-in-the-middle attack si tratta di una categoria che comprende molti attacchi (anche molto vecchi) e che è stato usato contro un ampio spettro di protocolli. Un attacco MITM avviene quando l'attaccante si pone in mezzo alla connessione tra altri due soggetti. Può essere un attacco sia di tipo attivo, sia di tipo passivo e sfrutta svariate tecniche per rubare dati sensibili o fingere l'identità di qualcun altro. L'attacco MITM è estremamente semplice da effettuare su protocolli in chiaro (ad esempio HTTP) poiché basta intercettare il pacchetto per essere in grado di leggerne il contenuto. È però possibile eseguire questo tipo di attacco anche su connessioni cifrate falsificando un certificato o agendo sui DNS dell'utente.

Alcune delle tecniche utilizzate per compiere questo attacco sono il packet sniffing e l'IP spoofing.

- Il packet sniffing si basa su software in grado di leggere i pacchetti che attraversano il livello di rete. I packet sniffer sono software in grado di trovare pacchetti all'interno di Local Area Networks (LANs) e Wide Area Networks (WANs) e sono programmi molto utili per risolvere problemi legati alla rete, analizzare le performance di rete o per trovare eventuali intrusi all'interno della rete. Ma proprio per la loro capacità di "sniffare" i pacchetti possono essere utilizzati con scopi malevoli e quindi sottrarre password, dati di carte di credito e qualunque dato possa essere scambiato all'interno della rete. La soluzione migliore contro queste tecniche è la cifratura dei dati sensibili. Uno dei più famosi packet sniffer si tratta di WireShark. [13]

- L'IP spoofing consiste nel creare un pacchetto con le informazioni completamente scelte da chi lo crea: è possibile indicare indirizzo sorgente, contenuto, indirizzo del destinatario e una volta in rete verrà recapitato all'indirizzo IP scelto. Se non ci fosse alcuna misura di sicurezza all'interno di Internet sarebbe possibile creare pacchetti per raggiungere qualunque dispositivo di cui si conosca l'indirizzo IP e, inserendo all'interno del codice malevolo, mettere fuori uso il destinatario o entrare in possesso di quel dispositivo. Questo non è così semplice grazie ai processi di autenticazione di cui si è parlato in precedenza che permettono di avere qualche garanzia in più su chi abbia inviato il messaggio [4].

Grazie a queste tecniche è quindi possibile svolgere svariati attacchi e uno dei più interessanti sul web attuale (quindi interessante da analizzare per WebRTC) si tratta dell'Hijacking.

L'Hijacking in realtà si tratta di un vasto insieme di attacchi informatici in cui può rientrare anche l'attacco man-in-the-middle. Infatti si tratta di un "dirottamento" della connessione che può avvenire a vari livelli:

- IP Hijacking: consiste nel rubare indirizzi IP che appartengono ad altre reti. Si tratta di un attacco alle strutture che svolgono la funzione di routing in rete. In questo modo gli attaccanti sono in grado di reindirizzare il traffico verso altri siti. Per fare ciò bisogna essere in grado modificare le tabelle di routing all'interno degli autonomous system (AS) che sfruttano il Border Gateway Protocol (BGP) per scambiare tra di loro queste informazioni (per questo l'IP Hijakcing viene anche definito BGP Hijacking). Ogni AS, quindi, ha la possibilità di inviare i pacchetti verso determinati indirizzi IP in base all'area in cui si trova. La vulnerabilità di questo sistema sta nel fatto che questo protocollo implementa molto poco l'autenticazione e si basa sulla fiducia tra AS. Il problema è che un BGP router non può conoscere le routing policies dei suoi vicini e non può valutare la validità di un routing announcement. Questo tipo di dirottamente potrebbe anche avvenire in maniera involontaria a causa di un errore del router come è già successo in passato [14].
- Session Hijacking: siccome le connessioni HTTP sono stateless il server non è in grado di mantenere lo stato della connessione e per semplificare le azioni

degli utenti e garantire più possibilità di controllo a chi fornisce i servizi, sono stati introdotti i cookie. Questi oggetti non sono altro che stringhe che identificano l'utente e la sua sessione su un determinato sito web. È facilmente intuibile quindi che rubare un cookie equivale a riuscire ad entrare nella sessione al posto di un determinato utente e fingersi quest'ultimo. È proprio questo l'obiettivo del Session Hijacking. Per fare questo generalmente bisogna prima sniffare il pacchetto contenente il session ID e poi compiere richieste al server utilizzando lo stesso ID. Un'altra modalità consiste nel compiere un Cross-site script attack (XSS attack): in questo caso viene sostituito una parte del codice HTML con uno script in grado di svolgere azioni come reindirizzare l'utente verso una pagina web desiderata [15]. Nel 2008 un gruppo di ricerca testò i dieci principali servizi di webmail utilizzando il session Hijacking riuscendo ad hackerarli ed entrare quindi nelle caselle email delle vittime [16]

È quindi estremamente chiaro come nel web odierno sia necessario prestare estrema attenzione alle misure di sicurezza. Nel prossimo capitolo viene quindi introdotto WebRTC, una tecnologia che secondo le intenzioni degli sviluppatori vuole garantire, tra l'altro, anche importanti garanzie dal punto della sicurezza.

Capitolo 3
WebRTC

Web browser with Real-Time-Communication o WebRTC è una tecnologia open source sviluppata a partire dal 2011 da un consorzio di importanti attori del web come Google (con Chrome), Mozilla (con Firefox) e Opera. Al momento è sotto un processo di standardizzazione a livello di API da parte del World Wide Web Consortium (W3C) e a livello di protocolli da parte dell' Internet Engineering Task Force (IETF). Esistono infatti numerosi studi sull'architettura di questo framework sia un punto di vista delle funzionalità e possibili applicazioni sia della sicurezza.

L'idea da cui si è partiti per realizzare WebRTC è la semplificazione della comunicazione tra gli utenti utilizzando il Web e lo strumento alla base della navigazione Web: il browser. Infatti è nato per supportare la comunicazione browser-to-browser e creare una soluzione standard che unificasse le possibilità finora offerte dal mercato. A differenza delle soluzioni esistenti fino alla nascita di WebRTC che richiedono l'installazione di plugin o software esterni, le sue API vengono integrate direttamente all'interno dei browser poiché sono sviluppate in Javascript, il linguaggio che negli ultimi anni domina il Web ed è supportato da ogni browser [17].

L'uso principale con cui è nato questo progetto è la video chiamata tra due utenti ma grazie alle potenti funzioni disponibili si sono sviluppati velocemente decine di progetti che utilizzano i canali di comunicazione creati da WebRTC per P2P file sharing, comunicazione real-time di ogni tipo, videogiochi, video conferenze…

Un'altra caratteristica che lo sta rendendo così discusso e scelto per lo sviluppo di applicazioni è l'estrema importanza data alla sicurezza che viene implementata direttamente dalle API e che verrà discusso in dettaglio nel capitolo 4.

Nella figura 3 è possibile vedere uno schema riassuntivo del funzionamento di WebRTC che nei prossimi paragrafi verrà spiegato nel dettaglio.

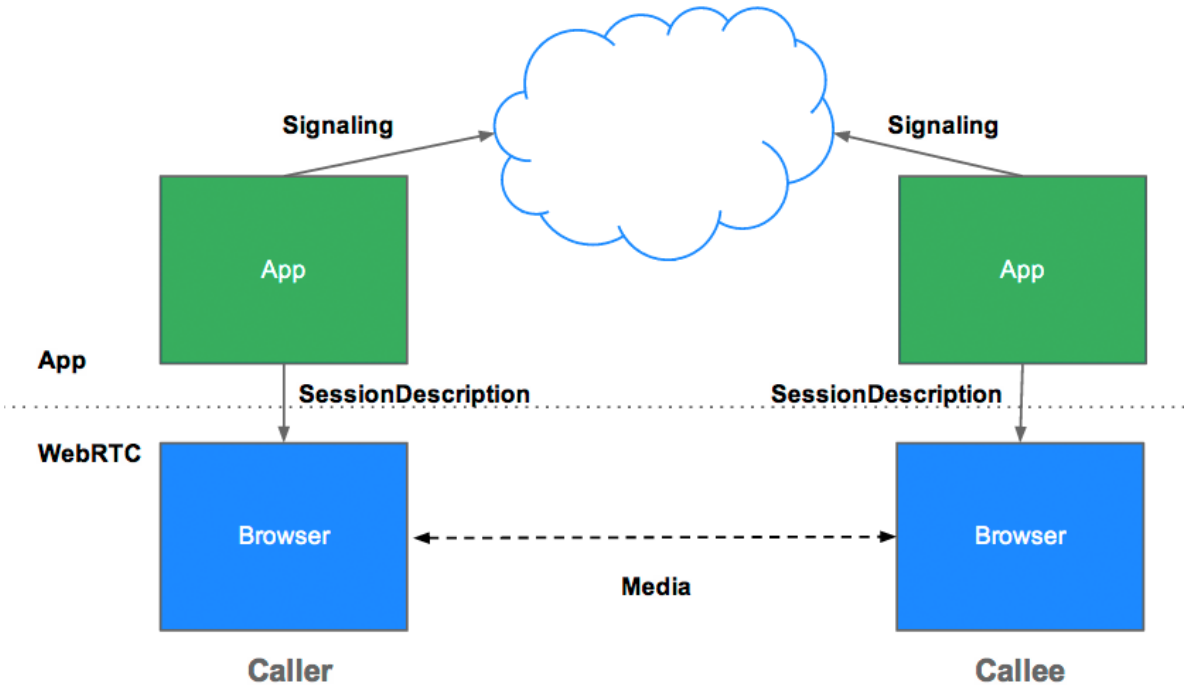

Figura 3 Schema riassuntivo del funzionamento di WebRTC. Il browser scarica la pagina web contenente l'applicazione in Javascript e HTML tramite HTTP(S), tramite le API messe a disposizione da WebRTC avviene il processo di signaling che permette al browser di iniziare lo scambio di dati direttamente con un altro browser senza più dover passare da alcun server centrale. [18]

3.1 UDP o TCP per WebRTC?

I protocolli di trasporto più utilizzati per qualunque applicazione web based sono TCP e UDP. Ma quali sono le differenze? E quale è meglio usare per lo streaming di contenuti multimediali e quindi per WebRTC?

TCP è un protocollo che tenta di rimediare alla mancanza di certezze che fornisce la rete per quanto riguarda la consegna dei pacchetti. È basato su algoritmi che rinviano i pacchetti finché non sono tutti giunti a destinazione, mantiene una coda dei pacchetti

arrivati e svolge un controllo di congestione della rete mandando quindi il flusso a differenti velocità in base al grado di congestione della rete in un determinato istante. Questo protocollo fornisce una grande affidabilità e permette una semplice gestione dei pacchetti che arrivano e che devono essere riprodotti poiché sono già accodati nel giusto ordine.

UDP è un protocollo molto più semplice che non prevede alcuna gestione della perdita dei pacchetti o della congestione di rete. Questo gli permette di avere meno ritardo nella consegna dei pacchetti ma allo stesso molti possono essere persi durante il loro percorso senza sapere se sono giunti a destinazione. Inoltre non vengono necessariamente accodati nell'ordine di partenza, bensì nell'ordine di arrivo: quindi nel caso in cui un pacchetto arrivi dopo che uno dei successivi nello stream è già stato riprodotto questo viene completamente perso.

La soluzione attualmente scelta per WebRTC è UDP per il poco ritardo che fornisce. In questo caso la gestione della perdita dei pacchetti viene svolta a livello applicazione tramite altri protocolli che verranno descritti e analizzati nei prossimi capitoli. Bisogna anche sottolineare che la pagina web su cui invece si trova l'applicazione è trasferita con HTTP(S) e quindi tramite TCP.

3.2 Signaling

La fase di signaling è la prima operazione che svolge WebRTC quando deve inizializzare una nuova conversazione. È in questa fase che vengono scambiate tra gli utenti le informazioni necessarie a creare un canale di comunicazione peer-to-peer.

Come specificato nella documentazione, WebRTC non prevede protocolli specifici per la signaling e gli sviluppatori possono scegliere qualunque protocollo duplex (che quindi permetta comunicazione bidirezionale) in base alle necessità dell'applicazione che stanno sviluppando. Viene però indicato che l'applicazione Javascript deve avere il controllo maggiore possibile nella fase di signaling, come specificato nel Javascript Session Establishment Protocol (JSEP) scritto proprio in relazione alla RTCPeerConnection API di WebRTC [19].

JSEP può essere utilizzato con molti protocolli di signaling, come ad esempio SDP che a sua volta può essere utilizzato insieme ad altri protocolli come RTP, RTSP o SIP.

SIP – Session Initiation Protocol [RFC3261] – è sicuramente il protocollo di signaling più diffuso (e che quindi trova più compatibilità anche in sistemi più vecchi). Si tratta di un protocollo leggero e aperto, funzionante sia su UDP che TCP, che fornisce meccanismi per notificare una chiamata in arrivo e di accordarsi sui codec che verranno utilizzati. Inoltre fornisce le tecniche necessarie per far determinare al chiamante l'indirizzo IP del chiamato permettendo una comunicazione diretta (proprio come richiesto da WebRTC). Infine può gestire la chiamata in corso aggiungendo flussi multimediali, cambiare codifica, invitare nuovi partecipanti, trasferire e mettere in attesa la chiamata. SIP è un protocollo out-of-band poiché agisce su socket diverse rispetto a quelle usate per lo scambio del flusso multimediale. Il funzionamento può essere descritto in pochi passaggi: ogni utente è registrato presso un dominio che provvede a tenere un database dei suoi utenti (secondo una modalità simile a SMTP con le email, ogni utente ha un nome del tipo utente@dominio.it); quando l'utente A vuole chiamare l'utente B, il primo manda un messaggio di invito che viene inoltrato da un server al ricevente che, se accetta la chiamata, invia un messaggio di risposta affermativo [4]. Nel capitolo 4 SIP verrà anche analizzato dal punto di vista della sicurezza.

Tramite il processo di signaling si crea quindi il canale condiviso tra i due peer ed è possibile scambiare tre tipi di informazione:

- Messaggi per il controllo della sessione, quindi inizializzare e chiudere la comunicazione oltre che riportare gli errori.
- Configurazione della rete, quindi dare informazione verso la rete esterna del proprio indirizzo IP e porta utilizzata.
- Le informazioni riguardanti i codec e le risoluzioni supportate dal browser in uso e dal browser dell'altro peer.

La fase di signaling è fondamentale poiché senza di essa non può essere inizializzata la RTCPeerConnection e quindi neanche le altre funzioni di WebRTC.

Come accennato precedentemente riguardo a JSEP, è necessario anche un protocollo che permetta agli utenti di scambiarsi le informazioni sulla sessione multimediale. In questo caso è stato scelto da WebRTC come standard il Session Description Protocol (SDP).

SDP è un protocollo in grado di descrivere una sessione multimediale con le informazioni riguardanti audio, video, i codec salvate in formato di testo. Lo scopo di SDP è quindi di annunciare l'inizio della sessione e la negoziazione dei parametri necessari. Il

protocollo prevede un meccanismo di offerta e risposta tra i due peer: in uno scenario di videochiamata possiamo immaginare come il peer chiamante invii un'offerta iniziale descrivendo il suo intento di iniziare una sessione di comunicazione multimediale, il ricevente a questo punto può declinare o accettare l'offerta facendo partire una risposta. In ogni momento della chiamata ognuno dei due partecipanti può aggiornare le informazioni della sessione generando un nuovo oggetto SDP [20].

Dal punto di vista implementativo i metodi appositamente sviluppati per rendere semplici questi procedimenti fanno parte della RTCPeerConnection. I principali sono questi: tramite *createOffer()* è possibile creare un offerta e *createAnswer()* genera un risposta; ognuno degli oggetti SDP creati viene salvato tramite *setLocalDescription()* e *setRemoteDescription()*, rispettivamente per quanto riguarda la propria descrizione e quella dell'altro utente [21]. Sempre con questi stessi metodi è possibile scambiarsi anche informazioni per l'autenticazione degli utenti, ma questa parte sarà studiata nel corso del capitolo 4.

Una volta scambiate sia le informazioni di rete che dei media può iniziare lo stream che viene gestito dalla RTCPeerConnection.

3.3 Le API di base

WebRTC si basa su tre API principali: MediaStream, RTCPeerConnection e RTCDataChannel. Vediamo più nel dettaglio cosa permette di fare ognuna di esse.

3.3.1 MediaStream

Prima dell'arrivo di HTML5 l'unico modo con cui si poteva accedere alle periferiche multimediali del proprio computer tramite il browser era con l'utilizzo di plugin come Flash o Silverlight: questi software però sono, prima di tutto, proprietari (e non open source come WebRTC). Inoltre sono stati spesso utilizzati come backdoor per compiere attacchi informatici; con HTML5 è stata introdotta la funzione *getUserMedia()* che, anche se con nomi leggermente diversi, è presente in tutti i browser attualmente più diffusi.

MediaStream è l'API di WebRTC che supporta il flusso audio e video e presenta due componenti principali: MediaStreamTrack e MediaStream.

Un oggetto MediaStreamTrack rappresenta un media di un solo tipo (quindi o audio o video) che proviene da una sorgente dell'utente. Un oggetto MediaStream invece può contenere zero o più oggetti MediaStreamTrack e possiede un input ed un output che rappresenta l'input e l'output combinato di tutte le tracce che ne fanno parte. L'output di un MediaStream può essere visualizzato tramite l'elemento <*video*> in HTML all'interno della pagina web, passato a una RTCPeerConnection per essere mandato come flusso a un altro peer, registrato su un file o anche tutte e tre allo stesso tempo. Lo stesso vale per un media solo audio che però viene visualizzato sulla pagina HTML tramite un elemento di tipo <*audio*>.

Il costruttore *MediaStream()*, quando invocato, compone un nuovo flusso di tracce esistenti e è possibile passargli un argomento (che può essere di tipo MediaStream oppure un vettore di MediaStreamTrack); ogni nuovo oggetto che viene creato ha un attributo che ne rappresenta l'ID. L'API prevede anche la possibilità di aggiungere ed eliminare tracce dall'oggetto MediaStream sia in maniera manuale sia in base ad eventi . Infine ogni oggetto MediaStreamTrack prevede alcune proprietà come *latency, sampleRate, volume, aspectRatio, frameRate, height, width* [22].

3.3.2 RTCPeerConnection

Questa componente di WebRTC è il fulcro della gestione della connessione tra la sessione locale e remota. Infatti ha il compito di creare una comunicazione stabile ed efficiente tra i peer utilizzando i codecs e i protocolli necessari.

Si tratta di una parte fondamentale delle applicazioni basate su WebRTC poiché permette di svolgere una grande mole di lavoro anche su reti inaffidabili come Internet gestendo i pacchetti persi, la cancellazione dell'eco, l'adattamento alla larghezza di banda, il buffering etc…Come è possibile notare dalla figura 4 svolge un ruolo centrale nel mettere in comunicazione le API e tutte le librerie che gestiscono audio, video e trasporto dei dati a livello più basso.

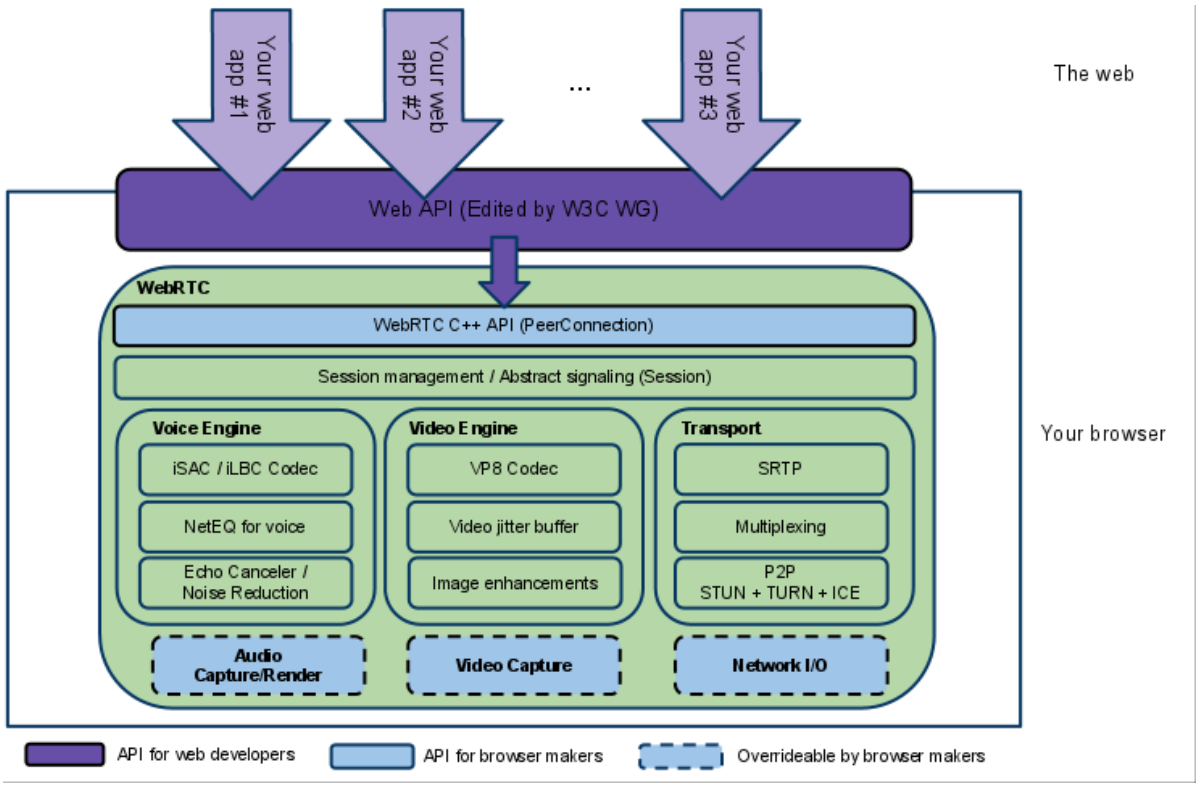

Figura 4 [18]

La maggior parte delle applicazioni che permettono lo scambio di contenuti multimediali in tempo reale utilizzano un server che si pone da intermediario tra due o più soggetti: ogni pacchetto mandato da un utente passa dal server che poi lo inoltra al soggetto destinatario; questa soluzione presenta moltissimi svantaggi sia dal punto di vista dei costi per i server sia dal punto di vista della privacy dell'utente. Uno degli scopi di WebRTC è proprio di semplificare e velocizzare la comunicazione eliminando il server multimediale e sostituendolo con la comunicazione diretta tra gli utenti (peer-to-peer). Nel mondo reale la comunicazione peer-to-peer tra due soggetti non è affatto scontata perché la rete internet ha una struttura complessa e tende a nascondere gli indirizzi IP degli utenti. La RTCPeerConnection possiede i metodi per aggirare queste problematiche e stabilire la connessione tra i due peer.

Il problema principale risulta essere quello del NAT. Il Network Address Translation permette di modificare gli indirizzi IP dei pacchetti che transitano su un sistema che agisce da router all'interno di una comunicazione tra due o più host. Questa tecnica permette di non associare univocamente un indirizzo IP ad ogni dispositivo ma di nasconderlo dietro a un indirizzo IP più generico che al suo interno prevede più dispositivi e quindi di risparmiare sul numero di indirizzi IP. È il router in cui è implementato il NAT che poi indirizza i

pacchetti nella sottorete che gestisce. Risulta chiaro quindi che mettere in comunicazione due peer tramite indirizzo IP non è scontato e sono necessarie tecniche per aggirare il NAT [23]. In generale qualunque protocollo che utilizza l'approccio offerta/risposta ha difficoltà attraverso i NAT poiché i pacchetti trasportano gli indirizzi IP e le porte necessarie a stabilire la connessione; inoltre, per diminuire i problemi di latenza e la perdita di pacchetti, stabiliscono un flusso diretto che è reso molto difficoltoso attraverso i NAT.

Prima di vedere come fa di preciso WebRTC a creare la connessione peer-to-peer, bisogna introdurre due protocolli utilizzati con i NAT: STUN e TURN.

STUN – Session Traversal Utilities for NAT – è descritto nella RFC5389 e può essere utilizzato da un dispositivo per scoprire l'indirizzo IP e la porta che gli vengono assegnate da un NAT o per controllare la connettività tra due punti terminali. Non si tratta di un protocollo che da solo è in grado di risolvere il problema del traversamento dei NAT ma può essere utilizzato come strumento per farlo. Si tratta di un protocollo client-server e prevede che il client invii una richiesta verso uno STUN server; quando questa richiesta giunge al server potrebbe aver attraversato uno o più NAT che modificano il pacchetto. A questo punto lo STUN server conosce l'indirizzo IP esterno alla rete protetta dal NAT e invia al client un pacchetto che al suo interno possiede proprio l'IP e la porta utilizzate; così il client viene a conoscenza del proprio indirizzo [24].

TURN – Traversal Using Relays around NAT – è un protocollo nato come estensione di STUN ed è definito dalla RFC5766. Infatti a volte risulta impossibile per due peer comunicare direttamente se si trovano dietro NAT nonostante l'utilizzo di STUN. TURN prevede quindi di essere utilizzato come nodo di scambio per permettere ai peer di scambiarsi pacchetti. È stato sviluppato per essere utilizzato come parte di ICE, di cui si tratta tra poco, ma può essere utilizzato anche indipendentemente. Tramite TURN un utente può quindi usare un server come relay e controllarne alcuni aspetti: una volta inviata la richiesta al server TURN viene assegnato al peer richiedente un IP e una porta da comunicare al peer con cui vuole scambiare dati e ogni pacchetto che passa attraverso quell'indirizzo e porta viene poi reindirizzato verso il peer richiedente [25]. Ovviamente un server TURN utilizzato in un'applicazione WebRTC potrebbe arrivare a gestire una mole consistente traffico e quindi un server di questo tipo è solitamente a pagamento, a differenza dei server STUN che di solito sono gratuiti (anche Google offre un servizio di serve STUN).

WebRTC basa la connessione sul framework ICE – Interactive Connectivity Establishment. Questo protocollo è descritto nella RFC5245 e può essere utilizzato da qualunque altro protocollo che utilizza il modello offerta/risposta e si appoggia ad UDP per il flusso multimediale. ICE funziona includendo una grande molteplicità di indirizzi IP e porte all'interno delle offerte e risposte SDP; questi IP e porte vengono poi testati per verificare se c'è la possibilità di stabilire una connessione P2P. Per trovare gli indirizzi IP e le porte viene utilizzato prima il protocollo STUN su UDP e poi, nel caso abbia fallito, il protocollo TURN. Il tipico funzionamento di ICE prevede due agenti che sono stati messi in contatto indirettamente tramite un processo di signaling e che si possono scambiare messaggi SDP. In questa fase ICE non ha idea se siano o meno presenti NAT nella rete che collega i due peer. Il compito di questo protocollo è quindi provare tutte le coppie possibili di IP e porte, che vengono attentamente ordinate secondo determinati algoritmi, finché non si trova quella funzionante. [26]. Esiste anche una versione di ICE per l'uso con TCP descritto nella RFC6544 ma non riguarda WebRTC [27].

In figura 5 si può vedere uno schema delle operazioni descritte svolte da ICE cercando di creare una connessione peer-to-peer.

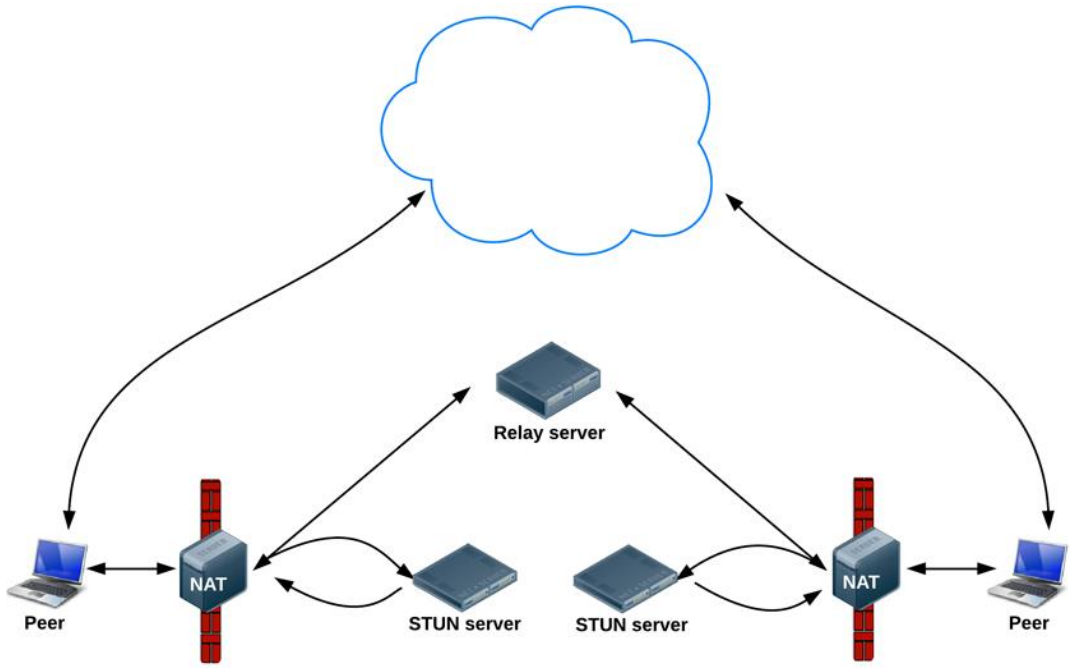

Figura 5 Funzionamento di ICE: i due peer si trovano nascosti dietro NAT e firewall e sono quindi impossibilitati a comunicare direttamente tra loro. Il primo tentativo di ICE per farli entrare avviene

attraverso i server STUN. Se per qualche motivo l'operazione fallisce si provvederà a creare una connesione tra il realy server basato su TURN [18].

Da un punto di vista implementativo la RTCPeerConnection riesce a nascondere tutti i complessi meccanismi che sono stati descritti utilizzando semplici metodi. L'interfaccia che spiega la struttura e i metodi della PeerConnection è descritta all'interno del protocollo JSEP [19].

Quando viene chiamato nel codice il metodo *new RTCPeerConnection()*, a cui è possibile passare i parametri di configurazioni contenenti le informazioni necessarie ad ICE, si crea un nuovo oggetto RTCPeerConnection che prevede lo stato della signaling, lo stato dell'ICE gathering e lo stato della connessione ICE. Inoltre un oggetto RTCPeerConnection prevede una coda delle operazioni che assicura che solo un'operazione asincrona nella coda sia eseguita concorrentemente. [21]

3.3.3 RTCDataChannel

Il Data Channel in WebRTC è stato sviluppato in un secondo momento diventando, però, un'API molto interessante per un sacco di applicazioni. Tramite RTCDataChannel è infatti possibile scambiarsi qualunque tipologia di dati con una latenza estremamente bassa e un alto throughput. In questo modo sono state sviluppati giochi, applicazioni di Desktop remoto, chat, trasferimento di file ecc. Tutto queste applicazioni risultano molto efficienti grazie alle caratteristiche proposte da WebRTC.

Il Data Channel riprende lo stesso modello delle WebSocket ma, invece, che basarsi su un'architettura client-server, utilizza la struttura peer-to-peer. Implica un flusso bidirezionale di dati e un campo di testo, definito label, che viene utilizzato per identificare cosa trasporta quel determinato Data Channel. Quindi un Data Channel è una coppia di un flusso in ingresso e di uno in uscita basati sul protocollo SCTP (Stream Control Transmission Protocol). Questo protocollo è molto importante nello scambio dei dati su UDP poichè aggiunge, tra gli altri, un controllo di congestione, supporta flussi unidirezionli multipli (ognuno con la propria consegna ordinata) e supporta anche la consegna di messaggi out-of-order ordinati. Nel capitolo 4 verrà approfondito come anche questo protocollo sia reso sicuro in WebRTC.

L’utilizzo del Data Channel da parte di un’applicazione prevede tre fasi: apertura del Data Channel, trasferimento dei dati e chiusura del Data Channel. [28]

- Apertura di un Data Channel: è possibile aprire un Data Channel sia con una in-band negotiation sia con una out-of-band negotiation. Nel primo caso è stato studiato un protocollo ad hoc per WebRTC, il WebRTC Data Channel Establishment Protocol (DCEP), che avviene al di sopra di SCPT. DCEP prevede un handshake a 2 fasi che accoppia un flusso di dati entrante e uno uscente, con lo stesso ID, all’interno di un singolo Data Channel bidirezionale [29]. Per quanto riguarda il secondo caso non è stato definito un protocollo specifico ma in può essere definito come qualunque metodo che porti a un accordo dei parametri necessari a creare il canale e alla creazione dello stesso. L’utente che vuole aprire un canale sceglie uno stream e l’applicazione è responsabile di impedire collisioni con altri flussi esistenti (non è possibile riutilizzare un flusso che è parte di un Data Channel esistente e l’applicazione deve conoscere quale stream è stato selezionato e le opzioni per mandarlo attraverso il canale) [28].
- Trasferimento dei dati su un Data Channel: tutti i dati che attraversano il canale devono essere mandati utilizzando il flusso specificato all’apertura. È specificato inoltre che in ogni messaggio SCTP deve essere inserito un solo messaggio da parte dell’applicazione. Sono inoltre definite alcune specifiche sulla sicurezza di cui si parlerà meglio più avanti ma che possono essere riassunte in breve come una stringente policy in caso di messaggi che contengono errori (il Data Channel viene chiuso). SCTP non supporta l’interleaving dei messaggi e quindi viene consigliato di utilizzare un’estensione aggiuntiva oppure di limitare la dimensione di ogni messaggio a 16kB in modo da ridurre il più possibile gli errori [28].
- Chiusura del Data Channel: quando un utente decide di chiudere il canale deve resettare il suo flusso uscente e quando anche l’altro peer riceve il flusso resettato deve eseguire la stessa operazione con il suo stream uscente. Una volta chiuso il canale gli stream possono essere riutilizzati per l’apertura di un nuovo Data Channel. Grazie a STCP vi è la certezza che tutti i messaggi siano comunque inviati e consegnati prima della chiusura del canale.

3.4 WebRTC è davvero peer-to-peer?

Più volte nel corso di questi capitoli ci si è riferiti a WebRTC come una tecnologia peer-to-peer ma questo è parzialmente vero. Infatti, come è stato spiegato, prevede che la fase di signaling sia attuata tramite server e che venga utilizzato ICE che, a sua volta, prevede l'utilizzo di server STUN o TURN.

In realtà queste sono solo le specifiche standard che sono state decise riguardo a WebRTC e che quindi sono quelle che si trovano nelle implementazioni di base nei browser. Ovviamente seguire le specifiche standard assicurano determinati livelli di efficienza, affidabilità e sicurezza, ma il punto di forza di WebRTC è anche la sua flessibilità grazie al fatto di essere open-source. Infatti non essendoci alcun protocollo predefinito per la fase di signaling sono state sviluppate molte possibilità che prevedono una topologia completamente P2P. Alcuni esempi sono basati su algoritmi di routing come Chord [30] che quindi rendono completamente libera da server l'applicazione. Oppure altri ancora utilizzano metodi di gossip per trovare e mettere in contatto i peer presenti nella rete; un esempio molto interessante di questo tipo è WebGC (Browser-Based Gossiping), una libreria che permette di sviluppare applicazioni basate su WebRTC utilizzando protocolli di Gossip.

In ogni caso lo scopo di questa tesi è di studiare l'applicazione da un punto di vista standardizzato e non tutti le singole estensioni e librerie studiate da singoli sviluppatori, ricercatori o aziende esterne poiché ognuna prevede le sue particolarità e, almeno in parte, i suoi protocolli.

Capitolo 4
WebRTC Security

WebRTC pone numerosi problemi di sicurezza, come qualunque applicazione basata sul web e che abbia come scopo primario la condivisione di contenuti sensibili, ma, proprio perché i problemi sono noti e studiati, WebRTC è stato sviluppato con l'idea della sicurezza interna all'applicazione stessa. Fin dal 2011 e durante tutto lo sviluppo delle API, una parte del gruppo che lavora a WebRTC è stato incaricato di analizzare i punti deboli e proporre le soluzioni più adeguate a evitare che malintenzionati possano sfruttare questa tecnologia a sfavore degli utenti o di chi fornisce il servizio o che possa essere utilizzata per attacchi di altro tipo. A capo degli studi riguardo la sicurezza in WebRTC vi è Eric Rescorla, esperto di sicurezza informatica e autore di importanti documenti riguardo TLS; con due Internet Draft principali, ancora work-in-progress, ha dettato le linee guida che vengono seguite nell'implementazione delle API standard di WebRTC.

Lo studio della sicurezza di WebRTC in questo capitolo segue un approccio bottom-up e verrà diviso in due parti principali: la parte che riguarda la rete (quindi i protocolli utilizzati e proposti dal working group) e la parte che riguarda il client e il browser (autenticazione, privacy, gestione dei permessi). Per ognuna di queste parti vengono descritti i problemi in gioco, le soluzioni adottate da WebRTC e i possibili attacchi che potrebbero essere fatti. Infatti, entrambe queste parti hanno un ruolo chiave nel proteggere l'utente dal furto di dati, di identità o nell'evitare che WebRTC venga utilizzato per compiere attacchi.

4.1 Rete

Sono già stati discussi, parlando delle meccaniche di signaling e scambio dati, alcuni dei protocolli utilizzati da WebRTC per mettere in comunicazione i peer. In questa parte verranno approfonditi in dettaglio dal punto di vista della sicurezza. Come si può vedere in figura 6, la pila protocollare standard di WebRTC prevede l'uso di diversi protocolli a livello applicazione, oltre al protocollo di signaling che può essere scelto dall'utente. Prima di cominciare, si noti che a livello trasporto o rete non esistono protocolli criptati poiché sono livelli necessari a inoltrare il pacchetto con le informazioni all'interno della rete e quindi devono poter essere leggibili da router, switch e server (per poter rispondere). Quindi tutta la parte riguardante la sicurezza si svolge necessariamente a livello applicazione. Infine bisogna sottolineare che l'utente, soprattutto se non esperto, può fare ben poco per evitare questi tipi di attacco poiché sfruttano principalmente falle di sicurezze presenti nei protocolli stessi.

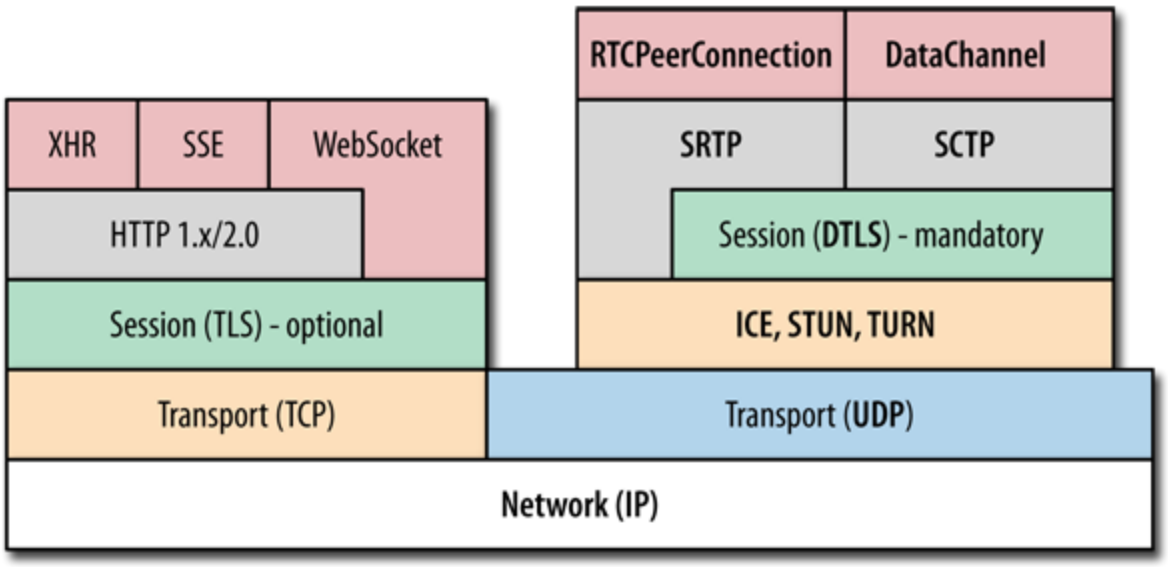

Figura 6 Pila protocollare per WebRTC. A sinistra i protocolli usati per scaricare la pagina web contenente l'applicazione, a destra i protocolli usati di WebRTC per scambiare i dati peer-to-peer. [31]

4.1.1 Fase di signaling

Siccome SIP è uno dei protocolli più utilizzati nel mondo della real time communication, anche nelle implementazioni di WebRTC, è il protocollo di signaling che

verrà analizzato. Il suo funzionamento è stato descritto al capitolo 3.2, ora vengono quindi presentate le sue vulnerabilità e i possibili attacchi.

SIP non ha al suo interno alcun meccanismo che prevede di criptare i messaggi che, se catturati, sono leggibili come semplici messaggi di testo e si presentano come quelli mostrati in tabella 1.

Tabella 1 Esempio di scambio di messaggi SIP tra due utenti: nella colonna a sinistra Alice invita Bob, in quella centrale Bob risponde e in quella a destra Bob chiude la chiamata [32].

INVITE sip:bob@192.168.20.5 SIP/2.0 Via: SIP/2.0/UDP 192.168.10.5:3456 Call-ID: a2e3a@192.168.10.5 From: sip:alice@192.168.10.5 To: sip:bob@192.168.20.5 Cseq 1 INVITE Content-type: application/sdp Content-Length: 98 \r\n v=0 o=mary 3123 121231 IP4 192.168.20.5 c=IN IP4 192.168.10.5 m=audio 49170 RTP/AVP 0	**SIP/2.0 200 OK** Via: SIP/2.0/UDP 192.168.10.5:3456 Call-ID: a2e3a@192.168.10.5 From: sip:alice@192.168.10.5 To: sip:bob@192.168.20.5 Cseq 1 INVITE **ACK sip:bob@192.168.20.5** **SIP/2.0** Via: SIP/2.0/UDP 192.168.10.5:3456 Call-ID: a2e3a@192.168.10.5 From: sip:alice@192.168.10.5 To: sip:bob@192.168.20.5 Cseq 1 ACK	**BYE sip:alice@192.168.20.5** **SIP/2.0** Via: SIP/2.0/UDP 192.168.20.5:3456 Call-ID: a2e3a@192.168.10.5 From: sip:bob@192.168.20.5 To: sip:alice@192.168.10.5 **Cseq 2 BYE** **SIP/2.0 200 OK** Via: SIP/2.0/UDP 192.168.10.5:3456 Call-ID: a2e3a@192.168.10.5 From: sip:bob@192.168.20.5 To: sip:alice@192.168.10.5 **Cseq 2 BYE**

- La prima idea che potrebbe venire in mente a un malintenzionato è di intercettare il messaggio di invito e cambiare il campo FROM (vedi messaggio INVITE in tabella 1) in maniera da reindirizzare la chiamata verso di lui. In questo caso si tratta di un attacco man-in-the-middle che potrebbe funzionare in questo modo [33]: l'attaccante compromette il server di signaling in maniera che l'offerta SDP creata con la PeerConnection venga ricevuta dal malintenzionato che, dopo averne creata una ad hoc, la manda al reale destinatario della conversazione. A questo punto è il computer dall'attaccante a negoziare lo scambio di offerta e risposta SDP e anche la connessione DTLS. All'interno del browser quindi i due utenti risultano correttamente collegati e autenticati anche se in realtà l'autenticazione è stata svolta dall'attaccante (vedi figura 7).

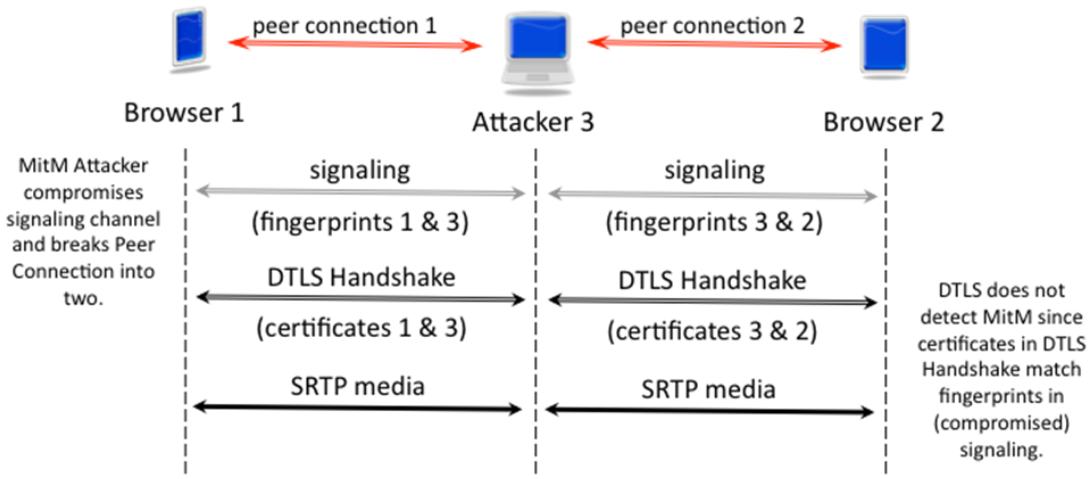

Figura 7 Attacco man-in-the-middle sul protocollo SIP in WebRTC [33]

- Un altro tipo attacco che potrebbe essere perpetrato è il Registration Hijacking. Come spiegato nel capitolo 3.2, ogni utente deve essere registrato a un dominio e per farlo deve mandare una REGISTER request al server dove vuole essere registrato. Questo tipo di attacco si basa proprio sulla richiesta di REGISTER e prevede due fasi: nella prima l'attaccante disabilita la registrazione dell'utente (ad esempio usando un attacco DoS oppure mandando decine di richieste fingendosi l'utente e quindi sovrascrivendo le richieste dell'utente legittimo); nella seconda invece manda la sua REGISTER request con l'indirizzo IP dell'utente originale modificato con il suo. A questo punto tutte le chiamate dirette all'utente verranno indirizzate all'attaccante [32].
- Un altro attacco potrebbe basarsi sulla session hijacking descritta nel capitolo 2.4.3, in questo caso il malintenzionato ruba il cookie che descrive la sessione per connettersi fingendosi un utente che si trova all'interno di una chiamata.

Gli attacchi descritti sono possibili per alcune vulnerabilità presenti in SIP. Prima di tutto non prevede alcun controllo sull'integrità del messaggio: questo permette di svolgere il Registration Hijacking poiché i pacchetti possono essere modificati a piacimento senza che l'applicazione abbia modo di accorgersene. Come già detto, non è un protocollo criptato ma esistono varie soluzione a questo problema: sono state sviluppate, infatti, implementazione di SIP che utilizza TLS (SIPS) oppure che scambia messaggi tramite WebSocket sicure. Utilizzando protocolli che rendono criptati i messaggi SIP vengono resi molto più difficili da compiere gli attacchi citati. Inoltre per quanto riguarda la session hijacking esistono algoritmi

che sfruttano l'indirizzo IP per creare i cookie: in questo modo, anche se rubati, rendono scaduta la sessione a chiunque si connetta con un indirizzo IP diverso da quello del cookie.

Infine consideriamo il caso in cui un oggetto JSON venga costruito in maniera difforme dallo standard e utilizzato per mandare in crash il server di signaling. Questo codice viene incapsulato in una normale richiesta e quando il server prova ad analizzare il messaggio che ha ricevuto produce un errore che, magari a causa di un bug, lo porta a bloccarsi e a non rispondere più alle richieste [34].

4.1.2 ICE

Come descritto nel capitolo 3.3.2 ICE viene utilizzato per superare il problema dei NAT. Anche un attaccante, però, potrebbe essere intenzionato a superare i firewall e i NAT per avere l'indirizzo IP diretto dell'utente, sia per attaccarlo direttamente sia per utilizzare un dispositivo in un attacco di tipo DDos.

- Il primo punto da analizzare è molto legato alla sicurezza del protocollo di signaling (vedi SIP nel paragrafo precedente). Infatti, se un attaccante è in grado di modificare o distruggere i messaggi SDP, potrebbe riuscire a compiere vari tipi di attacchi stile MITM. Come descritto nella RFC3264 [35], un altro attacco potrebbe essere quello di tipo replay: questo, a differenza di un attacco stile man-in-the-middle, può essere compiuto anche quando la comunicazione tra i due utenti è già terminata; infatti, consiste nel riuscire a replicare un vecchio messaggio di offerta SDP e ricreare una conversazione già avvenuta. Per evitare questo attacco è necessario che il protocollo preveda delle tecniche per riconoscere i messaggi di offerta e risposta già utilizzati in maniera da non accettarli.
- Nel funzionamento del protocollo ICE è presente una fase di connectivity check. Questa consiste nel controllare tutte le coppie di candidati (quindi di indirizzi IP e porte) fino a trovare quella più efficiente o fino al timeout impostato nel caso in cui non sia possibile trovare candidati funzionanti. Un attaccante potrebbe essere interessato a interferire in questa fase di connessione e può farlo in vari modi [26].

  - Primo modo è nel far credere che una coppia di candidati sia invalida anche se non lo è. Per compiere questo tipo di attacco deve riuscire a impedire la risposta positiva da parte di un utente (o ancora meglio impedire la risposta in maniera da farlo sembrare davvero irraggiungibile). Questo attacco risulta però molto difficile in ICE tramite STUN poiché quest'ultimo prevede un meccanismo di credenziali a breve termine.
  - Il secondo modo è far credere che una coppia di candidati sia valida anche se non lo è. In questo caso l'attaccante deve riuscire a creare una risposta falsa proveniente dall'utente che non sarebbe in grado di farla arrivare all'altro peer. Anche in questo caso il meccanismo di credenziali a breve termine di STUN rende tutto molto difficoltoso se svolto su un protocollo di signaling sicuro.
  - Il terzo modo è creare un falso peer reflexive candidate: in questo caso il flusso multimediale può essere utilizzato dall'attaccante. Per eseguire questo attacco è possibile come nei casi precedenti inserire falsi pacchetti per far credere che esista effettivamente un altro peer nella connessione oppure utilizzare il packet replay (quindi intercettare un pacchetto di controllo mandato da un peer verso un altro e replicarlo con un falso indirizzo IP e porta, bloccando la richiesta originale). Per compiere questo attacco è necessario che il falso peer riesca a identificarsi. Inoltre, se ai livelli superiori nella pila protocollare vengono utilizzati protocolli sicuri per il media stream (vedi SRTP), l'attaccante è impossibilitato a leggerli, ma potrebbe ancora distruggerli impedendo la comunicazione tra gli utenti.
- Esistono alcuni attacchi che potrebbero essere svolti attaccando i DNS o infettando direttamente i server STUN o TURN. Questa tipologia di attacchi risulta comunque estremamente complicata da gestire per l'attaccante ed è generalmente salvaguardata dall'utilizzo di DNSSEC come protocollo sicuro di risoluzione dei nomi in indirizzi IP.
- Cosa succede invece se l'attaccante è un peer autenticato all'interno della conversazione? In questi casi si parla di insider attack e nella RFC di ICE ne sono descritti due.

- The Voice Hammer Attack: consiste nel creare connessioni in maniera normale con altri peer, includendo però all'interno delle richieste anche l'indirizzo IP e la porta del target di attacco di DoS. ICE prevede un meccanismo per evitare attacchi di questo tipo: prima di mandare il traffico verso un terzo utente, esegue il connectivity check e, se il terzo utente non accetta la richiesta, non sarà portato a termine bloccando quindi sul nascere questo tipo di attacco.
- STUN Amplification Attack: l'idea è simile a quella di prima, cioè compiere un attacco DoS sfruttando un grande flusso di dati nei confronti di un target. In questo caso si sfruttano i pacchetti utilizzati da STUN per controllare la connettività e inserendo un grande numero di coppie candidate. Come detto, il target non risponderà mai alle richieste di connessione ma, siccome è presente un grande numero di coppie di indirizzi, continuerà a ricevere richieste saturando così la banda. La soluzione a questo attacco è limitare il numero di coppie di candidati accettate per ogni richiesta.

ICE è un protocollo studiato per essere molto sicuro e per questo è previsto nell'implementazione standard di WebRTC. Nei documenti che descrivono le misure di sicurezza di WebRTC ([36] e [37]), è indicato anche come il codice Javascript non debba in alcun modo avere accesso agli identificativi utilizzati per le sessioni ICE. Inoltre è necessario che chi sta ricevendo il traffico abbia un modo di confermare continuamente di volerlo ricevere. Per questo scopo è stata proposta la RFC7675 che prevede di utilizzare STUN [38]: dopo la connessione stabilita con ICE all'inizio, ogni 30 secondi viene richiesto tramite messaggi STUN nuovamente il consenso esplicito senza il quale non possono essere mandati dati multimediali. Come ulteriore misura di sicurezza ogni messaggio STUN prevede un nuovo transaction ID.

4.1.3 Datagram transport layer security: DTLS

DTLS si tratta del protocollo scelto da WebRTC per rendere sicure le comunicazioni sopra UDP ed è obbligatorio in ogni implementazione. La versione più recente è la 1.2 ed è descritto dalla RFC6347 [39] basata sulle stesse caratteristiche di TLS. Esattamente come TLS, quindi, prevede due livelli: il record layer e l'handshake. Il primo serve a garantire che

la connessione sia privata e affidabile tramite meccanismi per criptare i dati e per controllare l'integrità dei messaggi. Il secondo livello, invece, viene utilizzato per la fase di autenticazione tra il client e il server utilizzando la crittografia a chiave pubblica. Una volta conclusa anche la fase di handshake tutti i dati possono essere mandati in maniera sicura.

DTLS è un necessario adattamento di TLS per due principali motivi: il primo è che l'handshake di TLS prevede che i messaggi arrivino interi e nel giusto ordine, caratteristiche garantite da TCP ma in nessun modo da UDP. La seconda è che per decriptare i dati mandati tramite TLS sono necessari i dati inviati in precedenza e anche in questo caso UDP non garantisce nulla di simile e i pacchetti arrivano in ordine casuale. DTLS quindi implementa un modello basato su ACK e ritrasmissione e la possibilità di frammentare i pacchetti.

Utilizzando un software di packet sniffing come WireShark è possibile trovare un esempio dello scambio di pacchetti DTLS durante una sessione di WebRTC. In tabella 2 è mostrato un pacchetto DTLS di handshake.

Tabella 2 Struttura di un messaggio DTLS: viene specificato il tipo di messaggio (Handshake e più nello specifico Client Hello) ed è interessante notare soprattutto l'aggiunta dei campi legati alla sequenza e alla frammentazione assenti in UDP.

```
Datagram Transport Layer Security
DTLSV1.0 Record Layer: Handshake Protocol: Client Hello
     Content Type: Handshake (22)
     Version: DTLS 1.0 (0xfeff)
     Epoch: 0
     Sequence number: 2
     Length: 152
     Handshake Protocol: Client Hello
          Handshake Type: Client Hello (1)
          Length: 140
          Message Sequence: 0
          Fragment Offset: 0
          Fragment Length: 140
          Version: DTLS 1.2 (0xfefd)
          Random bits…
          Session ID Length: 0
          Cookie Length: 0
          Cipher Suites Length: 34
          Cipher Suites (17 suites)…
          Cookie…
```

Source	Destination	Protocol	Length	Info
64.95.96.18	192.168.254.88	DTLSv1.0	339	Client Hello
192.168.254.88	64.95.96.18	DTLSv1.0	725	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
192.168.254.88	64.95.96.18	DTLSv1.0	725	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
64.95.96.18	192.168.254.88	DTLSv1.0	1027	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Messa…
192.168.254.88	64.95.96.18	DTLSv1.0	133	Change Cipher Spec, Encrypted Handshake Message
64.95.96.18	192.168.254.88	DTLSv1.0	339	Client Hello
192.168.254.88	64.95.96.18	DTLSv1.0	727	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
192.168.254.88	64.95.96.18	DTLSv1.0	727	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
192.168.254.88	64.95.96.18	DTLSv1.0	727	Server Hello, Certificate, Server Key Exchange, Certificate Request, Server Hello Done
64.95.96.18	192.168.254.88	DTLSv1.0	1027	Certificate, Client Key Exchange, Certificate Verify, Change Cipher Spec, Encrypted Handshake Messa…
192.168.254.88	64.95.96.18	DTLSv1.0	133	Change Cipher Spec, Encrypted Handshake Message

Figura 8 Alcuni pacchetti DTLS scambiati durante una sessione di WebRTC sul sito opentokrtc.com catturati con WireShark. È possibile notare i messaggi di Hello tra il Client e il Server, lo scambio dei certificati e l'inzio dello scambio dei dati criptati.

In figura 8 viene mostrato una sequenza di un paio di handshake DTLS in un'applicazione WebRTC. Inizia con il primo messaggio mandato dal client (il Client Hello mostrato in tabella 2) a cui il server risponde mandando il suo certificato, la sua chiave e richiede il certificato al client che, a sua volta risponde inviando il suo certificato, la sua chiave e anche il protocollo usato per criptare i dati. Il server risponde per l'ultima volta confermando che tutto è andato a buon fine e può iniziare lo scambio dei dati protetto. Dalla figura 8 è anche possibile notare alcuni messaggi (ad esempio quelli di Server Hello) vengono divisi in più datagrammi poiché troppo lunghi. Risulta evidente come sia necessario implementare servizi per gestire la perdita e il riordino dei pacchetti ricevuti, siccome UDP non fornisce garanzie; questo è visibile nell'esempio in tabella 2 con i campi riguardo alla sequenza e alla frammentazione. Risulta molto difficile attaccare l'handshake DTLS perché, di fronte a qualunque imperfezione nei messaggi, la connessione viene subito interrotta.

In WebRTC per ogni canale di comunicazione che viene aperto tramite ICE viene creata una nuova sessione DTLS con un nuovo handshake e le nuove chiavi vengono utilizzate anche per criptare i pacchetti SRTP [36].

Come già spiegato, potrebbe capitare che non sia possibile utilizzare ICE tramite STUN server e quindi interviene un TURN server che agisce da relay. In questo caso cosa succede dal punto della sicurezza? Praticamente nulla perché, anche nel caso in cui il server TURN sia compromesso o addirittura controllato da un attaccante, l'unica cosa che può fare è leggere il livello di trasporto UDP del pacchetto che passa dal server grazie alla crittografia attuata da DTLS.

Nonostante il livello di sicurezza molto elevato garantito da DTLS, esistono alcuni attacchi possibili descritti nella RFC7457 [40]; sono tutti attacchi che possono essere compiuti indistintamente sia a TLS che DTLS.

- Il primo è SSL stripping: questo attacco consiste nel riuscire a trasformare una connessione HTTPS in una HTTP. Si tratta di un attacco man-in-the-middle e viene svolto facendo passare il traffico di un utente attraverso un server proxy gestito dall'attaccante. In questo modo la connessione criptata sarà tra l'attaccante e l'host della sorgente originaria, mentre tra l'attaccante e l'utente iniziale la connessione sarà stabilita tramite HTTP e quindi leggibile come messaggio di testo. Tramite questo attacco è possibile far credere ad un utente di essere connesso al servizio desiderato (come una banca o un servizio di videochiamate che sfrutta WebRTC) e, per esempio, rubare le credenziali di accesso.
- Un altro attacco è STARTTLS Command Injection Attack: questo attacco viene eseguito sfruttando il meccanismo definito Opportunistic TLS in cui messaggi di testo non criptati (ad esempio in protocolli come SMTP, FTP, XMPP, ma anche con SIP) vengono criptati fuori dall'applicazione solo in un secondo momento (e quindi non direttamente sul dispositivo che li sta utilizzando). La più famosa implementazione si chiama STARTTLS, da cui viene il nome dell'attacco. È facilmente intuibile quindi come la fase in cui l'attaccante ha la possibilità di agire è quella tra l'applicazione che invia dati non criptati e il momento in cui avviene la crittazione con TLS (o DTLS).
- Il Padding Oracle Attack, invece, sfrutta lo standard DTLS che prevede il meccanismo del MAC-then-encrypt che genera un codice MAC dal testo tramite una funzione hash e successivamente lo cripta. In questo modo per decriptare un messaggio il server deve controllare se il formato del messaggio è valido e invia una risposta in base all'esito del controllo. L'attaccante può quindi sfruttare queste risposte per capire se un messaggio è criptato correttamente e tramite determinate tecniche riuscire a decriptarlo. In realtà questo attacco al momento non ha quasi più riscontro grazie agli aggiornamenti più recenti delle implementazioni di DTLS.
- Un altro esempio è CRIME (Compression Ratio Info-leak Made Easy), un exploit che può essere usato quando DTLS viene usato in formato compresso [41].

Esistono infine alcuni attacchi che sfruttano vulnerabilità nella gestione e controllo dei certificati che dovrebbero garantire la sicurezza e affidabilità di un determinato sito Web, e

altri che, invece, si appoggiano a vulnerabilità nei server riuscendo a rubare le chiavi private utilizzate per criptare le comunicazioni con gli utenti.

WebRTC, comunque, prevede ancora una maggiore sicurezza salendo nella pila protocollare grazie all'utilizzo di altri protocolli criptati e che implementano varie contromisure agli attacchi informatici.

4.1.4 Secure Real Time Protocol: SRTP

RTP – Real-Time Protocol - è un protocollo descritto nella RFC3550 [42] che fornisce una struttura di pacchetto audio e video con al suo interno informazioni come il formato dei dati del pacchetto trasportato, il numero di sequenza e il timestamp. Queste sono utili poiché viene utilizzato in segmenti UDP, anche se le caratteristiche mancanti a UDP vengono fornite anche da DTLS in WebRTC. È un protocollo utilizzato soprattutto nello streaming live e nelle videochiamate o chiamate voce via internet. In WebRTC è necessario che sia sempre accompagnato da RTCP – Real-Time Control Protocol – che serve a trasmettere all'altro partecipante alla videochiamata un report con statistiche sui pacchetti inviati, ricevuti, persi, oltre a permettere un minimo controllo di identificazione. In questo modo si può avere un feedback delle informazioni e si è in grado di controllare la performance modificando la banda utilizzata in tempo reale. Siccome non prevede alcun meccanismo di crittazione, questo protocollo viene esplicitamente vietato nelle specifiche di WebRTC favorendo invece la sua versione sicura: SRTP (Secure RTP) [43]. Questo protocollo permette di criptare il flusso multimediale, di garantire l'integrità dei dati, di fornire un semplice meccanismo di autenticazione e una protezione con i replay attack. Alcune ricerche hanno dimostrato che l'utilizzo di SRTP non influisce in modo significativo sull'efficienza e sulla qualità delle conversazioni rispetto alla versione non sicura RTP, garantendo quindi la validità di questo protocollo [44]. In realtà ad oggi esistono numerose varianti ed estensione per RTP e per questo motivo per le applicazioni basate su WebRTC esistono delle specifiche apposite che danno delle indicazioni per implementare SRTP [45].

Proprio da questo documento si legge prima di tutto che WebRTC deve implementare un particolare profilo di RTP: si tratta di RTP/SAVPF, descritto dalla RFC5124 [46], che mette insieme la versione base di RTP, SRTP e RTP/AVPF (RTP Profile for RTCP-based feedback). I primi due sono già stati descritti brevemente; il terzo, invece, serve a rendere più

flessibile l'invio di pacchetti RTCP secondo un modello ad eventi, estendendo così le potenzialità base di RTCP che si basano solo sulle statistiche derivanti dalla larghezza di banda. Nello sviluppo di un'applicazione WebRTC rimane comunque possibile aggiungere tutte le estensioni necessarie di RTP: questo permette di creare più facilmente implementazioni per videoconferenza, di riparare pacchetti corrotti, di scegliere una differente qualità dell'immagine rinunciando a un frame rate più alto, e così via.

Dal punto di vista della sicurezza la parte più importante, che viene analizzata nel prossimo paragrafo, consiste sicuramente nello scambio della chiave per criptare la sessione RTP; si tratta di una fase estremamente delicata poiché, se le chiavi vengono intercettate da un attaccante, è possibile leggere i dati multimediali trasmessi. Altri problemi di sicurezza possono riguardare il Canonical Name (CNAME), utilizzato nei pacchetti RTCP per identificare l'utente: se il CNAME viene scelto per essere utilizzato a lungo termine, un attaccante potrebbe essere in grado di tracciare le chiamate di un determinato utente; nelle indicazioni per RTP con WebRTC viene infatti imposto l'utilizzo di CNAME validi solo per una sessione. Anche questo protocollo potrebbe essere utilizzato per attacchi DoS: un attaccante in grado di modificare gli intervalli con cui vengono mandati i messaggi RTCP potrebbe creare problemi nel controllo di congestione e quindi saturare tutta la banda disponibile oppure potrebbe saturare la banda proprio con messaggi RTCP; per modificare questi parametri è generalmente sufficiente aggiungere una linea di codice nei messaggi SDP che, come spiegato, vengono trasmessi già in fase di signaling. Un altro modo potrebbe essere fornire parametri RTCP differenti tra gli utenti di una conversazione portando quindi a un timeout che chiude la comunicazione. Infine una problematica di SRTP è che la parte criptata è solo il contenuto multimediale ma non l'header: in alcune estensioni utilizzate da RTP all'interno dell'header sono presenti anche alcuni informazioni sul livello dell'audio in quel pacchetto (per esempio in caso di conversazioni in conferenza è possibile inoltrare solo i pacchetti che hanno anche l'audio e non gli altri così da limitare il consumo della banda); ovviamente questo crea alcuni problemi dal punto di vista della privacy perché, se un attaccante fosse in grado di intercettare questi pacchetti, capirebbe se due utenti stanno parlando.

4.1.5 Gestione delle chiavi per STRP in WebRTC

In figura 6 si vede come per trasportare il flusso di dati multimediali creato tramite la RTCPeerConnection viene utilizzato il SRTP, di cui è stato appena discusso. Esistono però vari protocolli per scambiarsi le chiavi per criptare il flusso e per negoziare i parametri della sessione multimediale. Nei prossimi paragrafi verranno analizzati i principali cercando di capire quali vengono imposti, consigliati o vietati meno dalle specifiche di WebRTC e perché.

4.1.5.1 SDES: SDP Security Descriptions for Media Streams

Nei primi documenti riguardo WebRTC il protocollo scelto per scambiare tra i peer i parametri di sicurezza per criptare il flusso multimediale era SDES (SDP Security Descriptions for Media Streams), descritto nella RFC4568 [47]. Questo protocollo si basa su messaggi SDP, esattamente come avviene per lo scambio dei primi metadati quando si avvia una conversazione tramite WebRTC e, esattamente come gli altri messaggi SDP, sono messaggi di testo in chiaro, leggibili da chiunque intercetti il pacchetto. Anche la definizione del protocollo specifica che la sicurezza deve essere garantita dalla fase di signaling tramite autenticazione e crittazione dei messaggi. Il problema con WebRTC sorge proprio nella fase di signaling che può essere fatta utilizzando anche protocolli non sicuri. Inoltre, criptando in due fasi indipendenti il processo di signaling e il flusso multimediale SRTP, si corre il rischio che le due fasi siano svolte da due utenti diversi.

L'attacco più semplice che un malintenzionato potrebbe proporre in questo caso è un Replay Attack grazie alla facilità nel furto delle credenziali. Un altro attacco potrebbe essere svolto modificando i parametri di sessione definiti dal pacchetto SDES: un esempio è nel campo *UNENCRYPTED* che, se inserito, può portare a non rendere possibile decriptare il flusso SRTP a chi lo riceve oppure a mandare il flusso in chiaro, senza alcuna protezione.

4.1.5.2 ZRTP : Media Path Key Agreement for Unicast Secure RTP

Un altro protocollo che è possibile utilizzare è ZRTP: Media Path Key Agreement for Unicast Secure RTP nella RFC6189 [48]. Questo protocollo è studiato in primo luogo contro gli attacchi di tipo man-in-the-middle. Infatti prevede un meccanismo di autenticazione con i protocolli di signaling che garantiscono l'integrità dei messaggi end-to-end. Inoltre, non utilizza un classico meccanismo a chiavi pubbliche ma si basa sul protocollo per lo scambio di chiavi di Diffie-Hellman che permette di stabilire una chiave condivisa sicura senza lo scambio di informazioni specifiche. In questo modo è possibile lasciare gli utenti liberi dalle

certification authority (oltre al fatto che avrebbe un alto costo la gestione dei certificati a livello utente). Il protocollo ZRTP negozia le chiavi solo dopo la conclusione del protocollo di signaling e dopo i controlli di connettività di ICE e utilizza la stessa porta di RTP. È inoltre disegnato in maniera che abbia un header molto ben identificabile rispetto ai messaggi RTP o STUN così che non possa essere sfruttato da un attaccante per creare finti pacchetti. Ad ogni nuova sessione vengono creati random gli ZRTP ID che servono a identificare i due peer. Inoltre ZRTP, per sopperire alla mancanza di affidabilità di UDP, prevede due timer per la ritrasmissione dei pacchetti: il primo per il messaggio di Hello e il secondo per tutti i messaggi successivi. Per evitare la presenza di un attacco man-in-the-middle questo protocollo utilizza Short Authentication String (SAS): queste sono chiavi che devono essere adatte ad un utente umano (ad esempio viene visualizzata una scritta ad entrambi i peer che devono pronunciarla a voce, se le parole coincidono vuol dire che stanno utilizzando le stesse chiavi e quindi non è presente nessuno tra i due interlocutori); con questo meccanismo l'utente è in grado di autenticarsi facendosi riconoscere dall'altra persona. Infine, una volta avvenuta l'autenticazione, viene sfruttato il principio della key continuity. La key continuity è utilizzata da molti protocolli e consiste nel verificare se la chiave dell'utente cambia nel corso della conversazione. In questo caso il browser può accorgersene e mostrare un messaggio di avviso per l'utente.

In figura 9 viene mostrato come ZRTP provvede allo scambio delle chiavi: in questo caso è Bob che sta chiamando Alice e quindi manda il primo messaggio di Hello; Alice risponde confermando di aver ricevuto il "saluto" di Bob e risponde con il suo messaggio di Hello. Ogni messaggio di Hello trasporta varie informazioni come gli ZRTP ID, la versione di ZRTP e quali algoritmi utilizzare per criptare i dati. Lo scambio di messaggi descritto finora è la fase più delicata dal punto della vista della sicurezza poiché non esiste ancora una chiave condivisa tra i due soggetti e quindi non è criptata: quando si utilizza ZRTP, un protocollo di signaling sicuro è assolutamente necessario. Comunque, una volta negoziate queste informazioni, si inizia la seconda fase in cui, tramite un protocollo basato sullo scambio di chiavi Diffie-Hellman, i due utenti riescono a creare una connessione crittografata (la fase F6-F7 della figura 9). Infine, avviene lo scambio dei messaggi di conferma delle chiavi e la comunicazione può avere inizio in maniera sicura.

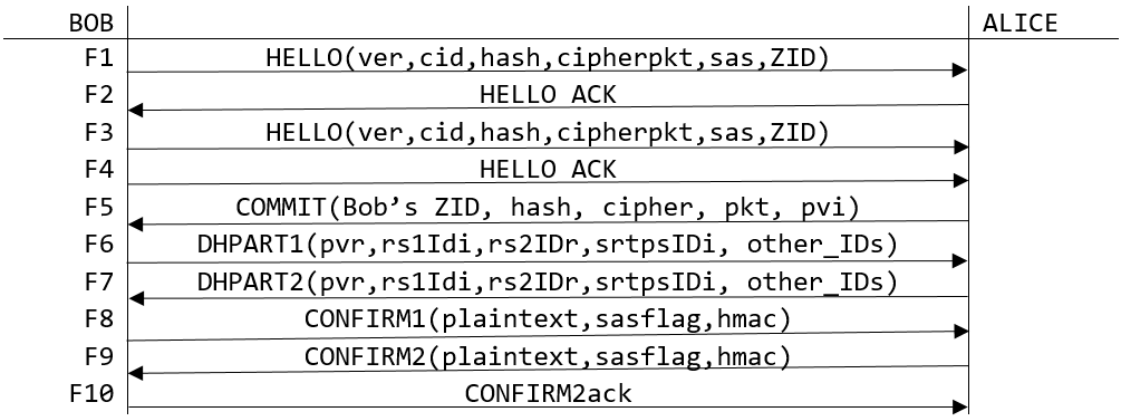

Figura 9 Scambio delle chiavi di una sessione SRTP utilizzando ZRTP

Nella realtà del Web e, quindi, in WebRTC sorgono però altri problemi. Riguardo alla key continuity bisogna considerare che spesso gli utenti accedono da più dispositivi allo stesso servizio (o anche solo da più browser) e questo crea problemi nella gestione della key continuity; si potrebbe quindi immaginare che il browser segnali continuamente la presenza di un problema nell'identità anche se in realtà si tratta della stessa persona che sta accedendo da due dispositivi differenti; il risultato finale è che l'utente finirebbe semplicemente con l'ignorare gli avvisi proposti dal browser. Allo stesso modo potrebbe essere facile indicare il nome dell'utente che si sta chiamando e che risulta autenticato, per esempio, presso il server di signaling; anche in questo modo però l'attaccante potrebbe riuscire a identificarsi con un nome molto simile a quello originale e l'utente potrebbe non dare importanza a piccole differenze. Per quanto riguarda la SAS, invece, potrebbero essere usati software per la modifica della voce riuscendo a impersonificare l'utente originale (anche se questo tipo di attacco risulta applicabile più nella teoria che nella pratica) [37].

Proprio per questi problemi l'architettura di WebRTC standard specifica che deve supportare l'utilizzo di DTLS-SRTP per rendere sicuro il flusso dei dati multimediali anche se potrebbe supportare anche altri protocolli per lo scambio delle chiavi [45]. Inoltre anche analisi che hanno coinvolto SDES, ZRTP e MIKEY (un altro protocollo utilizzato per lo scambio delle chiavi per SRTP) mostrano che sono protocolli che presentano molti punti deboli [49].

4.1.5.3 DTLS-SRTP

La RFC5764 è stata redatta per descrivere come DTLS debba essere utilizzato con SRTP [50]. Infatti, DTLS è un protocollo generico ed è stato necessario definire un protocollo specifico che mettesse assieme le garanzie di efficienza garantite da SRTP e di sicurezza proprie di DTLS. SRTP rimane il protocollo per criptare i dati, mentre DTLS viene utilizzato per lo scambio delle chiavi. La limitazione principale di DTLS-SRTP è che è studiato solo per le applicazioni con esattamente due partecipanti ma è comunque il più adatto nella maggior parte delle implementazioni di WebRTC (che è nato proprio con questo scopo). Il funzionamento prevede che per ogni flusso RTP avvenga un handshake DTLS che utilizza la stessa porta; una volta prodotta la chiave tramite DTLS, questa viene passata alle strutture dati di SRTP e da questo momento in poi i dati vengono criptati. È importante che per ogni coppia di porte tra i due peer venga stabilita una diversa sessione DTLS-SRTP. Ovviamente anche questo protocollo potrebbe essere soggetto ad attacchi informatici, ma risulta in assoluto il protocollo più sicuro per le tecnologie di VoIP e per questo WebRTC considera imperativo il suo supporto in tutte le applicazioni.

Come detto DTLS-SRTP è molto sicuro ed è universalmente accettato come sia necessario che venga supportato, mentre riguardo a SDES il dibattito è ancora parzialmente aperto, anche se esplicitamente vietato nell'architettura di WebRTC proposta al momento [36]. Basti pensare che ad oggi Google Chrome supporta entrambi i protocolli, mentre Mozilla Firefox implementa solamente DTLS-SRTP. Nell'agosto 2013, in un convegno alla IETF 87 di Berlino, Rescorla ha presentato un documento riguardo a SDES mettendo in luce i grossi problemi che comporta dal punto di vista della sicurezza [51]. Uno dei motivi principali che fanno tendere questo documento verso l'esclusione dell'utilizzo di SDES è l'incompetenza che a volte porta a costruire applicazione che lasciano in secondo piano l'aspetto della sicurezza per risparmiare sui costi. Secondo chi invece sostiene, almeno per alcuni casi, l'utilizzo di SDES è importante soprattutto per lasciare compatibilità tra chi utilizza WebRTC e chi invece utilizza sistemi più vecchi ed esistenti da diverso tempo. In questo caso Rescorla prevede di utilizzare i gateway tramite cui passano i dati come stazioni per recriptare i dati che hanno utilizzato SDES, come mostrato nella figura 10

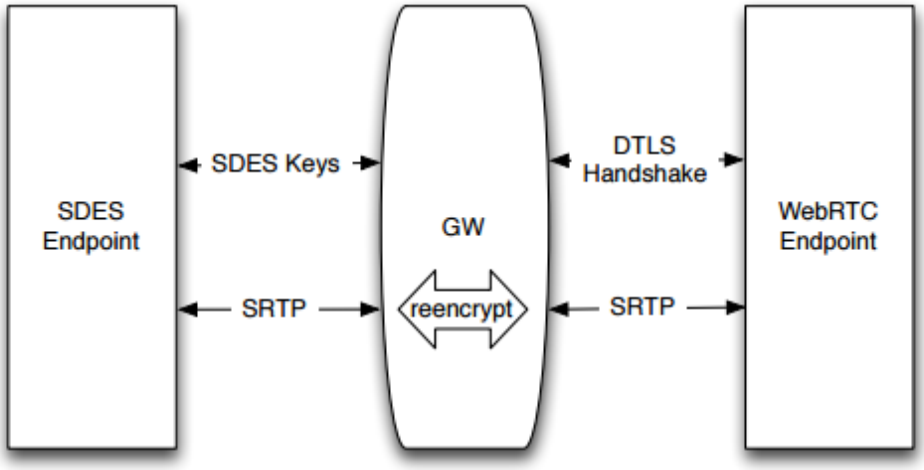

Figura 10 [51]

In generale, comunque, quando è previsto che vi sia crittazione end-to-end, bisogna impedire che WebRTC proponga API in grado di leggere le chiavi utilizzate e qualunque altro parametro legato a queste per impedire che l'attaccante possa sfruttarle per un replay attack o inserirsi nel corso della conversazione. Inoltre, è importante che anche l'interfaccia grafica del browser permetta all'utente di capire facilmente come viene criptata la conversazione e le caratteristiche della sicurezza riguardante i flussi audio e video utilizzati e visualizzati [37]. La parte riguardante le caratteristiche, anche di interfaccia grafica, che il browser deve fornire per garantire la sicurezza all'utente verranno comunque trattate in maniera più approfondita in seguito.

4.1.6 Stream Control Transmission Protocol: SCTP

Per quanto riguarda il Data Channel, invece, viene utilizzato SCTP: questo protocollo viene descritto dalla RFC4960 [52]. È in grado di trasmettere qualunque tipologia di dati garantendo controllo di congestione e la gestione dei pacchetti persi. È stato sviluppato per competere con TCP in alcuni tipi di applicazioni che, per esempio, richiedono una latenza inferiore (come nel caso di WebRTC). Inoltre TCP è vulnerabile in attacchi DoS, che invece SCTP cerca di moderare.

Nello specifico SCTP risulta resistente agli attacchi DoS di tipo flooding grazie al suo four-way handshake iniziale che finchè non è concluso non permette di inviare altri dati. Inoltre, prevede un Verification Tag in grado di individuare i pacchetti estranei all'interno del flusso dati. Un attaccante potrebbe però utilizzare una grande quantità messaggi INIT (utilizzati per iniziare l'handshake) verso vari host di un determinato dominio: così per

risolvere l'indirizzo utilizzeranno tutti lo stesso server DNS provocandone il blocco per le troppe richieste ricevute. Anche un attacco masquerade (quindi in cui l'attaccante usa una falsa identità per accedere) viene prevenuto grazie al four-way handshake e i Verification Tag; osservando il log delle richieste INIT potrebbe inoltre essere possibile rilevare un attacco in corso.

WebRTC usa SCTP al di sopra di DTLS come specificato nella documentazione sulla sicurezza ([36]) e segue il modello indicato nella Internet Draft per incapsulare i pacchetti SCTP in DTLS [53]. Prima di tutto la connessione DTLS deve essere stabilita prima che venga chiamato in causa SCTP; inoltre, i messaggi iniziali di SCTP non devono contenere nessun parametro riguardo gli indirizzi IP. Sono anche previsti determinati meccanismi per trovare e testare il Maximum Transmission Unit Path (cioè il percorso che permette di inviare il pacchetto di dimensioni più grandi): anche se SCTP si trova incapsulato in DTLS e in altri protocolli, deve sempre eseguire i suoi meccanismi per calcolare l'MTU Path.

4.2 Sicurezza nel browser

La scelta dei protocolli per WebRTC è di assoluta importanza perché è la base di partenza per poter costruire un'applicazione sicura. Questo modello crea una gerarchia in cui il livello più alto deve potersi fidare di quello più basso; all'apice di questa gerarchia si trova il browser: le API Javascript di WebRTC sono perfettamente integrate all'interno del browser e, quindi, è necessario che anche questo garantisca all'utente determinate misure di prevenzione agli attacchi. L'utente deve poter capire con chi sta avviando una conversazione, se si può fidare del servizio che sta utilizzando, quali risorse del sistema sta condividendo tramite WebRTC.

Rescorla definisce il browser come Trusted Computing Base (TCB) [37]. È al suo interno che, infatti, viene scaricato dal server tutto il codice HTML e Javascript necessario a far funzionare un'applicazione WebRTC. La prima misura di sicurezza per l'utente è utilizzare un browser di cui può fidarsi e averlo ottenuto da una fonte affidabile. Se questo presupposto è soddisfatto, l'utente può essere certo che tutte le API e librerie del browser seguono gli standard indicati e quindi limitare i possibili attacchi. Ovviamente, come qualunque applicazione nel mondo dell'informatica, WebRTC e il browser utilizzato sono considerati sicuri fino alla scoperta della prossima vulnerabilità.

4.2.1 Same Origin Policy

All'interno del browser ogni script viene eseguito in maniera indipendente dagli altri all'interno di sandbox. Esistono varie tecniche per permettere agli script di eseguire richieste al server (ad esempio AJAX o XMLHttpRequest) ma viene adottata la Same Origin Policy (SOP). Questa prevede che le richieste possano essere inoltrate solo al server da cui proviene lo script che le sta facendo. In questo modo, se uno script è considerato proveniente da un'origine sicura (intesa come URL e porta utilizzate), si suppone che ogni richiesta fatta verso la stessa origine possa essere considerata affidabile. Questa politica permette anche di limitare gli attacchi DDoS: basti pensare che ad uno script posto su molte pagine scaricate da migliaia di utenti, se questo potesse mandare tutte le richieste che vuole verso un target a piacere sarebbe semplice saturare la banda del target. Infine grazie alla SOP vengono limitate le interazioni tra gli script presenti nella pagina: per esempio uno script utilizzato per mostrare un annuncio pubblicitario o per gestire le statistiche dei visitatori non può (e non deve) avere accesso alle variabili Javascript utilizzate da WebRTC sulla stessa pagina.

Dal punto di vista della sicurezza questa politica è una garanzia, ma dal punto di vista degli sviluppatori può essere un grande scoglio: per esempio all'interno di WebRTC come si farebbe a mandare la richiesta per avviare la fase di signaling verso un server diverso? Esistono alcune tecniche per aggirare questo ostacolo, come per esempio CORS (Cross-origin Source Request) o le WebSocket. L'idea è semplicemente quella di verificare se l'host a cui viene fatta la richiesta voglia davvero rispondere e questa verifica viene svolta tramite un semplice handshake.

4.2.2 Javascript Injection

Un attacco possibile sfruttando le caratteristiche peer-to-peer di WebRTC è Javascript Injection [34]: questo attacco potrebbe permettere, per esempio, di mandare messaggi all'interno della chat di testo utilizzata da WebRTC o di fare sniffing delle informazioni. Potenzialmente potrebbe anche permette all'attaccante di catturare le credenziali di accesso al server. Per quanto riguarda la pagina web da cui è possibile accedere al servizio di WebRTC è comunque bene notare che molti browser non permettono di utilizzare codice Javascript ottenuto tramite HTTP in una pagina scaricata tramite HTTPS: questo garantisce

che, anche se la sorgente della pagina è sicura tramite TLS, poi l'applicazione Javascript utilizzi canali non criptati fornendo anche protezione contro la Javascript Injection.

4.2.3 Diffusione di malware tramite WebRTC

WebRTC potrebbe anche essere utilizzato per diffondere un malware tramite il DataChannel [34]. Siccome i file vengono mandati direttamente da un utente senza il controllo del server (come nel caso degli allegati email), sarebbe utile creare un modo di controllare se i pacchetti spediti contengono qualche software dannoso. Questo può essere fatto secondo tre modelli:

- Il peer manda il file prima a un server che lo verifica e poi lo spedisce al peer destinatario. Questo potrebbe essere il modo che fornisce maggiore sicurezza se il server è sicuro ma ovviamente annulla tutti i vantaggi forniti dal P2P.
- Il mittente manda il file al server che lo controlla e manda un hash del file al destinatario; poi il mittente invia il file direttamente all'altro peer che calcola di nuovo l'hash e lo confronta con quello inviato dal server. Anche questa soluzione non risulta assolutamente efficiente e non azzera comunque tutti i rischi possibili.
- Infine, il mittente manda il file direttamente al destinatario che lo controlla. Questo controllo potrebbe essere implementato direttamente nel browser che almeno per alcune tipologie di file (come gli eseguibili) dovrebbe verificarne il certificato.

4.2.4 Considerazioni generali per la protezione dagli attacchi in WebRTC

I possibili attacchi descritti mettono in luce la necessità di distinguere le entità autenticate da quelle non autenticate, come indicato nell'architettura della sicurezza di WebRTC [36]. Le prime sono distinte nel servizio che fornisce WebRTC (che quindi può essere verificato tramite HTTPS(e negli altri utenti (che possono essere verificati tramite i meccanismi di autenticazione forniti da DTLS-SRTP). I secondi, invece, racchiudono qualunque entità che non sia autenticata e che quindi è necessario considerare non sicura a prescindere. È utile notare come autenticato non voglia dire necessariamente fidato. Verrà approfondita in seguito la tematica dell'autenticazione analizzando le possibili tecniche che la permettono applicata a WebRTC.

Infine esiste un'altra considerazione importante riguardo a WebRTC e il browser come Trusted Computing Base: qualunque altra applicazione precedente necessitava di un software specifico (ad esempio Skype) o l'aggiunta di un plugin per il browser (ad esempio le piattaforme basate su Flash). Questo introduceva ulteriori problemi di sicurezza perché spesso l'utente scarica il software da fonti non verificate (rischiando di scaricare versioni corrotte delle applicazioni) o non lo aggiorna con la frequenza necessaria (rendendosi vulnerabile a problemi noti, magari già risolti dagli sviluppatori da molto tempo). I browser oggi prevedono ormai aggiornamenti automatici e molto frequenti a causa della velocità nello sviluppo di nuove tecnologie sul web; così ad ogni aggiornamento corrisponde l'aggiornamento delle API di WebRTC. Nel caso di Google Chrome, il browser con la miglior performance di aggiornamento, si arriva a quasi il 100% di dispositivi aggiornati entro 20 giorni dal rilascio limitando moltissimo la finestra temporale in cui un attaccante potrebbe sfruttare una vulnerabilità nota [54].

4.2.5 Accesso ai contenuti e alle periferiche

Quando un utente accede a servizi di videochiamate tramite il browser, spesso non si rende conto di essere esposto a giganteschi rischi dal punto di vista della privacy. Sono stati trattati tutti i protocolli implementati da WebRTC necessari a impedire che vengano catturati pacchetti con dati personali, ma cosa succede se un utente utilizza una sola volta un servizio online per videochiamare con un amico e questo servizio riesce a prendere il controllo della webcam? Si potrebbero immaginare scenari in cui quel determinato servizio, utilizzato una volta, abbia sempre accesso alla videocamera ogni volta che il browser è aperto (o semplicemente il computer è acceso dato che molti browser rimangono aperti in background). Nel capitolo 3.3 è stata trattata l'API getUserMedia per accedere alla webcam e al microfono dell'utente: cosa succede se visitando un sito web quest'API viene richiamata e inizia a registrare video e audio? Ovviamente è necessario che il browser provveda innanzitutto a chiedere il consenso all'utente per l'accesso a queste periferiche; come mostrato negli esempi in figura 11, tutti i browser attualmente più diffusi prevedono questa pratica.

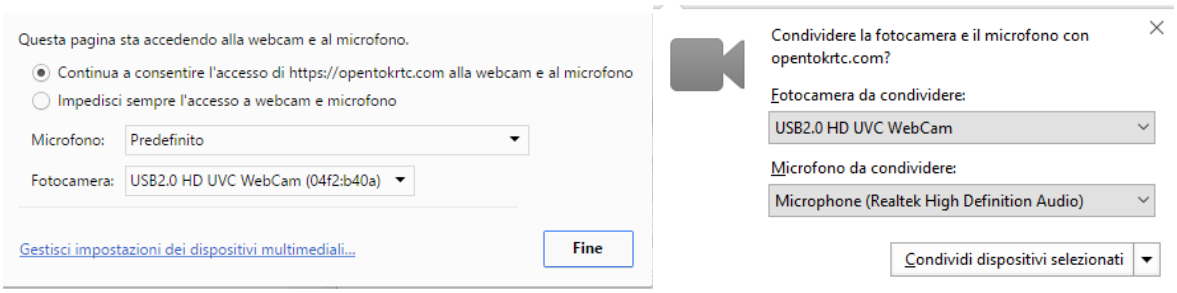

Figura 11 Esempi di consenso all'accesso di webcam e microfono all'interno del browser Google Chrome (a sinistra) e Mozilla Firefox (a destra)

Inoltre è necessario differenziare il caso di una pagina scaricata tramite HTTPS e quello di una scaricata tramite HTTP: nel primo caso è indicato che potrebbe essere richiesto l'accesso per una sola volta oppure permanentemente, mentre nel secondo caso è obbligatorio che il browser chieda esplicitamente ogni volta il consenso all'utente. Il rischio principale consentendo pagine tramite HTTP è che, ad esempio in una rete poco protetta (come una rete aperta o quella di un internet point), l'attaccante riesca facilmente a reindirizzare l'utente verso un altro sito web (o modificare il codice HTML/Javascript) e quindi dirottare la chiamata verso di lui.

Il browser deve anche mostrare chiaramente le finestre in cui vi è accesso alle proprie periferiche (vedi figura 12) e dovrebbe fermare la condivisione di webcam e microfono nel momento in cui le finestre vengono ridotte a icona o nascoste (e l'utente potrebbe quindi dimenticarsi di essere ripreso e ascoltato). Le indicazioni sulla sicurezza in WebRTC precisano anche che deve essere possibile impedire l'utilizzo di metodi per registrare o modificare il video da parte del fornitore del servizio [36].

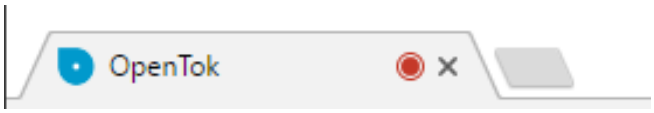

Figura 12 Esempio di interfaccia grafica che mostra l'accesso da parte del browser a webcam e microfono (Google Chrome)

In realtà fidarsi di una determinata origine (anche se protetta tramite HTTPS) potrebbe non essere una soluzione adeguata: infatti se un utente autorizza permanentemente, ad esempio, https://serviziowebrtc.com questo potrebbe riuscire a iniziare la comunicazione in ogni momento, anche mentre non si sta visitando le pagine web specifiche di quel dominio. Infatti, è sufficiente che su un'altra pagina web siano presenti un <*iframe*> o uno script che hanno

come origine https://serviziowebrtc.com per permettere una nuova chiamata. Un'altra situazione particolare è l'autorizzazione di un servizio per assistenza clienti (ad esempio di una compagnia telefonica o di un esercizio commerciale) implementato tramite WebRTC: una volta consentito l'accesso, l'utente deve essere sicuro che sia solo per la durata della chiamata che ha intenzione di effettuare.

Il problema però non trova facile soluzione. La prima idea è quella di chiedere l'autorizzazione ogni volta che sta per iniziare una sessione di WebRTC: in questo caso l'utente dovrebbe acconsentire ogni volta; secondo numerosi studi, però, l'utente tende ad accettare senza analizzare a chi e per cosa sta dando l'autorizzazione, soprattutto se la stessa richiesta viene effettuata molto spesso. Un'altra possibilità è quella di permettere le chiamate solo ad una determinata lista di utenti che si è sicuri di conoscere (esattamente come avviene con Skype, dove prima di poter contattare qualcuno bisogna averlo nella propria lista contatti): anche in questo caso possono sorgere problemi: ad esempio, il sito che si utilizza come servizio (o perché malevolo o perché sotto attacco) potrebbe dirottare una chiamata verso un altro utente o dichiarare che l'utente A vorrebbe iniziare la chiamata, quando in realtà si tratta dell'utente B. Questa tematica è risolvibile studiando a fondo il problema dell'autenticazione (si veda il capitolo 4.2.2) e fornendo delle interfacce grafiche diverse in base al tipo utente che si sta chiamando (se è chiaramente identificabile o meno): invece che come mostrato in figura 11, sarebbe bene mostrare anche a chi sta per essere indirizzata la chiamata con un messaggio tipo "Condividere la propria camera e microfono su questa pagina per la comunicazione con user@host?". Rimane necessariamente aperto il problema della registrazione della videochiamata da parte dell'altro utente contro cui non sono possibili contromisure: bisogna fidarsi dell'utente con cui si svolge la videochiamata.

L'ultimo argomento da affrontare riguardo l'accesso ai contenuti da parte di applicazioni WebRTC è lo screen sharing. Sono nati moltissimi servizi ed estensioni per i browser che permettono questo servizio estremamente utile e interessante, è sufficiente navigare all'interno di *webrtc-experiment.com* (portale che raccoglie moltissime estensioni e applicazioni sviluppate per WebRTC) o all'interno degli store che raccolgono le estensioni dei browser per rendersene conto. Eppure lo screen sharing mette in pericolo i dati dell'utente in maniera sostanziale e, potenzialmente, molto più della condivisione della propria webcam o microfono. Infatti moltissime persone lavorano sul proprio PC, accedono a servizi personali come la propria banca attraverso i servizi online, scrivono mail confidenziali e così via. È

necessario chiedersi quali sono le implicazioni nel caso in cui un utente attivi la condivisione del proprio schermo (o anche solo della schermata del browser) e se ne dimentichi; oppure se compaiono notifiche push da parte del browser all'interno dell'area di schermo condivisa; oppure se esiste la possibilità per l'applicazione di accedere a una scheda del browser diversa da quella che l'utente vorrebbe condividere; o, ancora, se l'attaccante riesce ad aprire una finestra con il codice sorgente di una pagina web, potrebbe catturare i caratteri di sicurezza utilizzati a volte per evitare attacchi Cross-Site Resource Forgery (questo attacco permette di inviare richieste HTTP al server riuscendo a sfruttare la sessione del vero utente e si tenta di neutralizzarlo inserendo caratteri casuali ogni volta che la pagina viene scaricata). Rescorla in più documenti affronta questa tematica ([37], [36], [55]) che però è ancora in ampia fase di discussione all'interno del Working Group di WebRTC poiché è molto delicata e richiede sicuramente una maggiore richiesta di consenso prima che l'utente inizi a condividere il proprio schermo. Le linee generali descritte da Rescorla prevedono che il browser non debba permettere il consenso permanente all'accesso da parte dell'utente, che debba sempre mostrare una finestra di dialogo separata da quella di webcam e microfono in maniera che l'utente abbia più chiaro cosa sta andando a condividere, che debba indicare le finestre che sono condivise in ogni momento e che le finestre o schede nascoste non debbano essere condivise. La versione attuale di Google Chrome non abilita di default questa funzionalità, mostrando una finestra di dialogo come quella mostrata in figura 13.

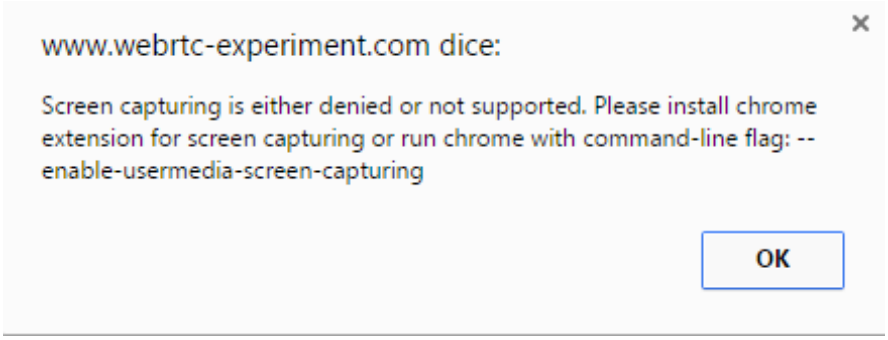

Figura 13 Finestra di dialogo di default per applicazioni screen sharing in Google Chrome.

Una volta attivata la funzione che lo permette, viene invece richiesto il consenso ad ogni utilizzo e, appena iniziata la condivisione dello schermo, compare un avviso sempre in primo per ricordare l'attività, come è possibile vedere in figura 14. Sebbene attivare questa funzione potrebbe non essere scontato per l'utente medio, è possibile anche installare alcune estensioni che semplificano il processo ma allo stesso aumentano i possibili rischi per la sicurezza.

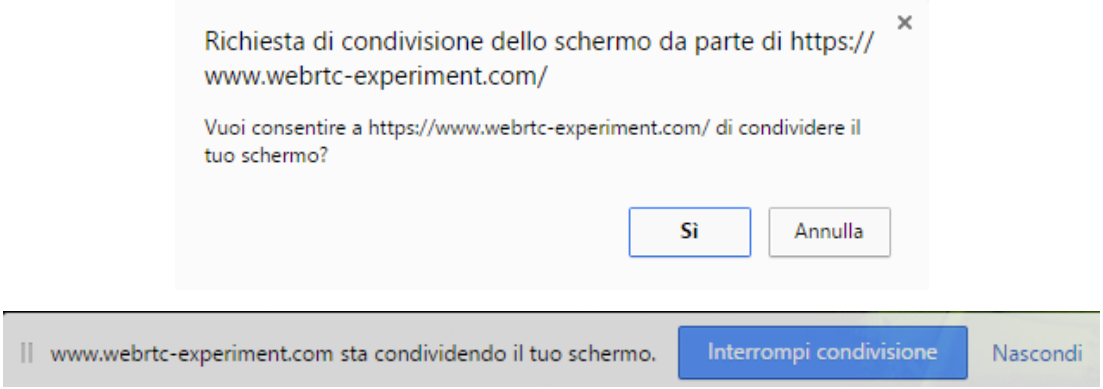

Figura 14 La richiesta di condivisione dello schermo e l'avviso dell'attività in corso su Google Chrome.

4.2.6 Autenticazione dei peer

Nella parte sulla sicurezza dei protocolli utilizzati da WebRTC sono state descritte anche alcune tecniche di autenticazione che è possibile utilizzare (ad esempio in SIP) e allo stesso tempo perché sono essere vulnerabili. In molte applicazioni WebRTC, però, potrebbe essere desiderabile parlare con amici o parenti o altri contatti che si conoscono e che si vorrebbe avere la certezza che siano loro quando viene effettuata o ricevuta una chiamata. È stato già stato esposto il problema di quanta fiducia si possa riporre nel servizio che offre l'applicazione e, proprio perché spesso non si può avere piena fiducia, sono nati diversi servizi che permettono l'autenticazione tramite il Web: sono ormai molto diffusi e permettono di identificarsi su servizi di diverso tipo. Sono definiti sistemi Single-Sign-On (SSO) e sono gestiti da IdP (Identity Provider) come, ad esempio, Facebook Connect, o Google Sign-in: ognuno di questi garantisce tramite la connessione sicura del browser che l'utente loggato sia realmente lui. Al momento i protocolli utilizzati maggiormente da questo tipo di serivizi sono OAuth2.0 e OpenID Connect, anche se è un settore in forte espansione; un protocollo più recente è invece BrowserID che garantisce anche di rispettare maggiormente la privacy dell'utente non tenendo traccia dei siti utilizzati. In ogni caso il server di signaling e qualunque altra parte dell'applicazione WebRTC non ha accesso a nessun dato riguardante l'autenticazione, ha solo la certezza che è avvenuta. Così l'utente A che riceve una videochiamata dall'utente B può essere certo che sia davvero l'utente B perché, per esempio, è un suo contatto su Facebook. È il browser che garantisce che la connessione si sia svolta in maniera sicura e l'IdP che garantisce l'identità dell'utente; il vantaggio è anche nel fatto che qualunque IdP possa essere implementato in maniera molto

semplice senza aggiungere altri protocolli oltre a quelli già citati. Bisogna anche distinguere tra due tipi di IdP: quelli *authoritative* e quelli *third-party*. I primi sono i servizi simili a Facebook Connect che gestiscono in maniera diretta l'identità dell'utente e che quindi non prevedono ulteriori controlli da parte del servizio che fornisce WebRTC; i secondi, invece, non gestiscono direttamente le informazioni dell'utente ma verificano l'identità tramite altri meccanismi ed è quindi necessario avere una lista all'interno del browser che garantisca quali IdP sono affidabili (come avviene, per esempio, con i certificati SSL e le certification authority).

È però necessario indicare le linee guida di come devono svolgersi le operazioni di identificazione e verifica. Di nuovo è Rescorla ad occuparsene per WebRTC nella sua proposta di architettura della sicurezza [36].

Uno schema riassuntivo di come dovrebbe funzionare l'autenticazione tramite IdP è visibile in figura 15.

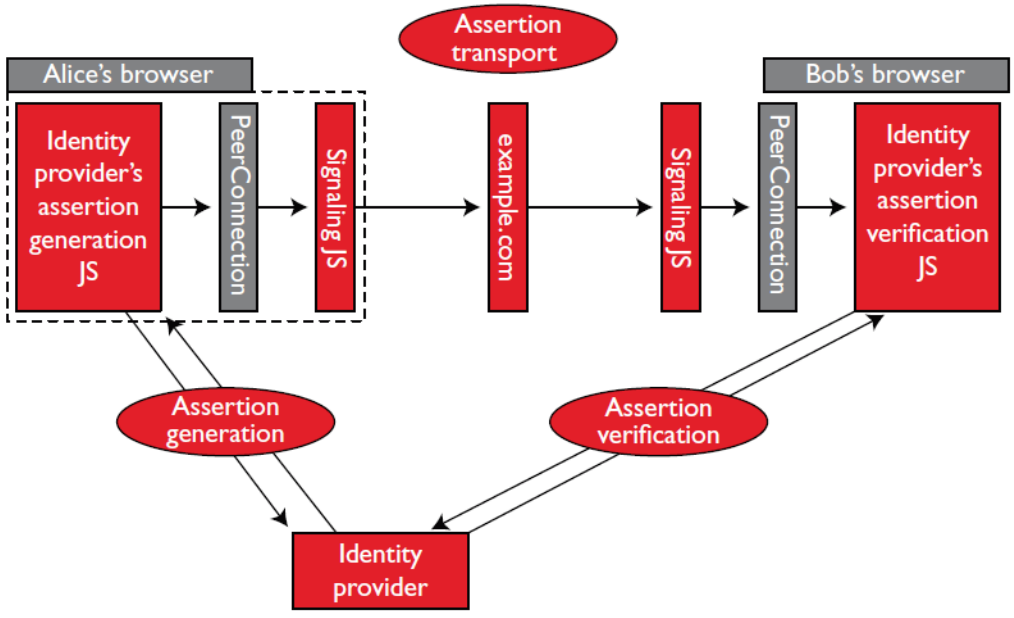

Figura 15 Ognuno dei due peer riceve un assertion generata tramite l'IdP, questa viene mandata tramite il server di signaling all'altro peer che la verifica sempre tramite l'IdP [56]

L'idea generale prevede di associare le chiavi generate e scambiate tramite DTLS-SRTP con l'identità degli utenti verificata tramite l'IdP e infine utilizzare la rubrica contatti (magari tratta dagli amici Facebook o dai contatti di Google) dell'utente per avere tutte le informazioni del contatto con cui si sta iniziando la chiamata. In questo modo si ha la certezza

che la persona con cui si sta iniziando la conversazione è la stessa con cui si sono negoziate le chiavi del flusso multimediale. È già stata affrontata la prima parte su DTLS-SRTP, mentre ora viene descritta la sequenza di passaggi per associare la chiave pubblica con l'autenticazione tramite IdP.

La prima fase per l'utente è, ovviamente, registrarsi e avere un account presso un IdP e configurare l'accesso tramite il sito che fornisce il servizio WebRTC o direttamente tramite il browser (sia Google Chrome che Mozilla Firefox permettono di associare la propria identità al browser). Come è visibile in figura 15, la PeerConnection si occupa di gestire la comunicazione con l'IdP e poi di trasmettere i dati all'altro utente. Per poter accedere tramite un IdP, è necessario potersi fidare della sorgente da cui si scarica il codice Javascript usato per creare l'IdP Proxy all'interno del browser e quindi l'URI deve seguire queste caratteristiche [36]:

- Deve necessariamente essere HTTPS.
- Può contenere un particolare numero di porta che non viene però considerato per verificare l'identità.
- Non deve contenere informazioni dell'utente.
- Dopo il nome del dominio deve contenere una stringa di questo tipo "/.well-known/idp-proxy/" e il nome del protocollo.

Quindi dopo aver settato e inizializzato un IdP proxy all'interno del browser, sarà possibile per l'utente richiedere una identity assertion. La richiesta viene eseguita quando si crea un'offerta o una risposta SDP oppure con il metodo *getIdentityAssertion()*. Il metodo utilizzato dalla RTCPeerConnection è *generateAssertion()* che genera i parametri necessari all'IdP. Questi parametri si basano sulla chiave DTLS-SRTP e specificano l'algoritmo di codifica usato per creare la *fingerprint* della sessione (che rimane però in locale e non viene mandata all'IdP). Se tutto va come previsto l'IdP restituisce l'Identity Assertion richiesta secondo le procedure e i protocolli specifici di quel determinato IdP: questo crea un vantaggio significativo per WebRTC perché risulta compatibile con qualunque servizio IdP presente e futuro. Ora l'utente A può inviare, tramite un messaggio SDP, la sua Identity Assertion all'utente B che deve verificarla. Questo processo incomincia quando viene chiamato il metodo *setRemoteDescription()* della RTCPeerConnection che identifica l'IdP utilizzato e, dopo aver creato l'IdP proxy nel browser, richiede di validare l'Identity Assertion. L'IdP utilizza le sue procedure specifiche e restituisce il risultato che viene controllato infine dalla

RTCPeerConnection mostrando, se disponibile nell'applicazione, anche altre informazioni dell'utente [21].

Ognuna delle fasi di autenticazione e validazione avvengono in maniera asincrona impedendo di bloccare le operazioni in caso di rallentamenti della rete; ovviamente in caso di errore la comunicazione non viene consentita e non è possibile proseguire. Un errore possibile è dato dal formato con cui viene indicata l'identità dell'utente. Viene infatti specificato che l'IdP deve identificare il peer secondo una stringa del tipo "<user>@<domain>" e spetta alla RTCPeerConnection verificare che sia corretta. Inoltre viene anche verificato se il <domain> corrisponde con l'IdP o, nel caso sia di tipo third party, se l'IdP può essere considerato affidabile [36].

La sicurezza di questo meccanismo si basa sull'affidabilità dell'IdP e sulla corretta implementazione della RTCPeerConnection che svolge un'importante funzione di controllo in locale. Siccome chiunque potrebbe creare un servizio di IdP, è fondamentale che l'utente abbia consapevolezza dei servizi che usa. In ogni caso, l'IdP non ha accesso ad alcun dato sensibile che riguardi la comunicazione dell'utente poiché la chiave DTLS-SRTP non viene mandato in chiaro all'IdP. Per evitare che un utente dell'IdP possa impersonificare l'IdP stesso grazie al possesso di alcuni contenuti sullo stesso dominio, è obbligatorio sempre controllare che l'URL da cui proviene il codice sia del tipo https://domain.com/.well-known/idp-proxy/protocol.

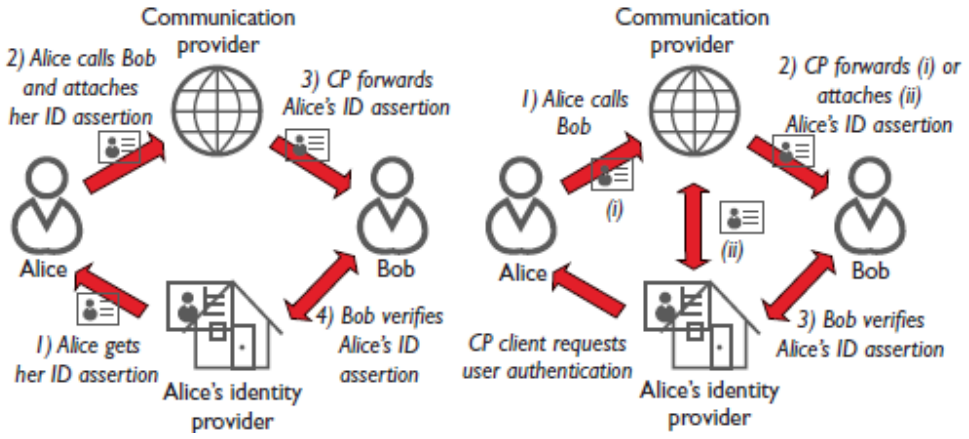

Figura 16 Modelli di autenticazione per WebRTC. Quello a sinistra è il modello richiesto nelle specifiche standard di WebRTC in cui il fornitore del servizio non è affidabile e non ha alcun contatto diretto con l'IdP (come in BrowserID); nel modello a destra invece è il fornitore del servizio a essere autorizzato dall'IdP e scambia direttamente con l'IdP.

Il modello proposto per WebRTC tende quindi a considerare il fornitore del servizio di chiamate non affidabile e a porre l'attenzione esclusivamente sul browser, come mostrato in figura 16. È questo che si trova al centro e comunica con l'IdP e con il sito web. Nel mondo attuale di internet però l'unico modello che si adatta perfettamente alla proposta di WebRTC è BrowserID. Esistono però alcuni servizi più noti e utilizzati come, ad esempio, OAuth2.0. Questi servizi non mettono al centro l'utente, bensì il fornitore del servizio che quindi comunica direttamente con l'IdP. Esiste però un modo per rendere anche queste tipologie conformi a quanto descritto nella documentazione i WebRTC. All'interno del protocollo OAuth2.0 è possibile mettere l'IdP proxy al centro della comunicazione svolgendo i compiti che svolgerebbe il server del fornitore del servizio.

BrowserID e OAuth2.0 sono le uniche due possibilità che vengono analizzate da Rescorla nel suo documento [36] ma secondo un altro studio OpenID Connect potrebbe svolgere meglio il compito dell'autenticazione rispetto a OAuth2.0 [57]. Infatti con questo protocollo quando un utente si autentica presso un IdP genera un ID token che può essere facilmente inserito all'interno di un messaggio SDP. Inoltre l'ID token permette di essere personalizzato con anche con informazione aggiuntive e di rispondere direttamente al browser senza dover passare dal fornitore del servizio rimanendo molto più in linea con il modello di WebRTC. Infine, presenta anche due vantaggi rispetto a BrowserID: prima di tutto non è associato all'indirizzo email e garantisce quindi una miglior tutela della privacy, seconda cosa all'interno del browser non viene generato alcun codice di autorizzazione (come invece accade in BrowserID) garantendo quindi una maggiore affidabilità.

4.2.7 Privacy

Il discorso sull'autenticazione dell'identità dell'utente si va necessariamente a scontrare con il discorso sulla privacy. Ma prima ancora dell'autenticazione, in WebRTC, il problema della privacy riguarda gli indirizzi IP. Per poter comunicare direttamente tra loro, gli utenti devono scambiarsi i candidati ICE e quindi venire a conoscenza dell'indirizzo IP reale dell'altro peer. Non solo, anche il server di signaling viene necessariamente a conoscere questa informazione. Tramite l'indirizzo IP potrebbe essere possibile fare IP spoofing, risalire al luogo in cui si trova l'altra persona, ricavare informazioni sulla rete in cui si trova, rendere tracciabili i movimenti di quel determinato indirizzo IP, all'interno di

un'organizzazione un attaccante potrebbe essere in grado di ricavare la struttura di un team e dei ruoli del personale in base alla struttura dei loro indirizzi IP [34].

La prima soluzione a questa problematica è permettere all'utente di scegliere di non utilizzare i servizi STUN ma solo quelli TURN e quindi rimanere protetto dietro l'indirizzo IP del server TURN. Questa soluzione però fa cadere completamente la struttura peer-to-peer che caratterizza WebRTC, oltre al fatto che diventa estremamente più costoso acquistare traffico dati su un server Turn per chiunque voglia creare un servizio WebRTC.

Una soluzione parziale è, invece, iniziare la negoziazione ICE solo dopo che l'utente ha accettato la chiamata in entrata. Questo permette di far conoscere l'indirizzo IP solo dopo che l'utente ha dato il consenso a rispondere. Questo risolve il caso di un attaccante che invia richieste STUN a un utente che si trova online in un servizio WebRTC per ottenere l'indirizzo IP della vittima e attaccarlo in qualche maniera. Lo svantaggio principale di questa soluzione è che non è possibile sapere lo stato di un utente (se si trova online oppure no).

L'ultima soluzione potrebbe essere quella di utilizzare un servizio come Tor oppure un proxy o un Virtual Private Network. Nel caso di Tor la privacy dell'indirizzo IP risulta totalmente tutelata ma diventa quasi impossibile avere un servizio che sia veramente real-time ed efficiente per la complicata struttura della rete Tor, ma potrebbe essere utilizzato per la creazione di chat anonime che non richiedono la stessa immediatezza e larghezza di banda. Il vantaggio di questi sistemi, comunque, è quello di nascondere il proprio indirizzo IP anche al server di signaling e al sito da cui si scarica la pagina web con l'applicazione WebRTC.

In generale, è anche bene notare che all'interno dei protocolli utilizzati da WebRTC esistono più identificatori (come i certificati DTLS o gli RTCP CNAMES) e che quindi sarebbe buona regola non riutilizzarli per più chiamate oltre a prevedere un meccanismo che generi forzatamente nuovi identificatori se l'utente utilizza servizi che permettono l'anonimato.

Infine, per quanto riguarda l'autenticazione, si può prendere in considerazione una ricerca svolta sui principali protocolli citati in precedenza (OAuth2.0, BrowserID, OpenID Connect) [57]. Questo studio ha considerato sei caratteristiche legate alla privacy dell'utente e ha analizzato quali vengono garantire da ogni possibile implementazione in WebRTC. Le caratteristiche scelte sono:

- Identificazione: consiste nella possibilità per un utente di controllare che l'utente con cui sta comunicando è chi dice di essere.
- Anonimato verso i partecipanti: la possibilità per un utente di autenticarsi tramite un IdP ma rimanere anonimo agli altri utenti che entrano in comunicazione con lui.
- Anonimato verso il fornitore del servizio: la possibilità per un utente di autenticarsi tramite il suo IdP senza che il fornitore del servizio entri in contatto con informazioni sensibili.
- Non linkability dell'identità: si ha quando il fornitore del servizio non può essere in grado di tracciare l'utente in base alle attività che svolge.
- Confidenzialità dell'identità: la possibilità per l'utente di mandare le proprie informazioni su un canale sicuro senza che il fornitore del servizio sia in grado di leggerle.
- Non linkability del fornitore del servizio: si ha quando l'IdP non è in grado di tracciare l'utente attraverso i vari servizi che utilizzato autenticandosi tramite lo stesso IdP.

Purtroppo non esiste alcuna tecnologia in grado di fornire tutte e sei le caratteristiche, soprattutto per quanto riguarda la confidenzialità dell'identità per cui, come già detto, l'unica soluzione sarebbe utilizzare reti come Tor con tutti gli svantaggi che comporta. Le migliori soluzioni risultano essere i protocolli OAuth2.0 o OpenID Connect (che come descritto in precedenza potrebbe funzionare ancora meglio con WebRTC) in un modello di non fiducia verso il fornitore del servizio, in linea anche con la Internet Draft di Rescorla.

Capitolo 5
Conclusioni

Questo elaborato ha cercato di raccogliere i principali documenti e ricerche e analizzare WebRTC dal punto di vista della sicurezza in ogni suo aspetto, partendo dai protocolli a livello più basso fino alle caratteristiche più specifiche dell'API. Il risultato è che si tratta di una tecnologia strutturata attentamente per garantire i migliori risultati in termini di semplicità, affidabilità e velocità per la trasmissione dei dati. Tutto questo senza però trascurare in nessuna maniera l'aspetto riguardante la sicurezza e la privacy per l'utente. All'interno di un mondo sempre più interconnesso come quello odierno, WebRTC ha la possibilità di giocare un ruolo molto importante, anche grazie alle sue caratteristiche peer-to-peer che rendono la sua una tecnologia scalabile e al fatto che sia open source.

La sicurezza delle applicazioni WebRTC è studiata per essere obbligatoria, non è possibile modellare un'applicazione WebRTC senza la sicurezza sia in primo piano: questo lo rende un passo avanti a tutti gli altri servizi VoIP e di comunicazione tra gli utenti che spesso hanno considerato la sicurezza come una funzionalità aggiuntiva e non integrata nel modello del software solo per una questione di risparmio.

Un'altra caratteristica fondamentale è l'integrazione diretta con il browser e non più tramite software esterni come è stato per ogni servizio simile precedente. Solo così è possibile garantire aggiornamenti costanti e interoperabilità tra più browser grazie all'ormai universale Javascript. Inoltre questo garantisce agli sviluppatori una meccanica molto semplice per sviluppare nuove applicazioni basate su WebRTC.

Durante questo studio sulla sicurezza in WebRTC sono emersi tre soggetti principali: il working group al lavoro sullo sviluppo di WebRTC, lo sviluppatore che vuole creare

un'applicazione basata su WebRTC e l'utente che utilizza un servizio WebRTC. Ognuno ha un ruolo per garantire che l'applicazione funzioni bene e sia sicura. Il working group è la base da cui ogni implicazione sulla sicurezza viene valutata e, se prevista, implementata in WebRTC: è il caso, per esempio, della getUserMedia (con la stretta politica di consenso all'accesso alle periferiche) o dei protocolli scelti per lo scambio dei dati (DTLS-SRTP). Lo sviluppatore che ha intenzione di creare un servizio che utilizzi WebRTC deve avere ben chiare tutte le politiche imposte e scegliere con attenzione il protocollo di signaling che vuole implementare o come garantire l'identificazione degli utenti perché, come risulta da questo elaborato, le misure di sicurezza e i protocolli criptati scelti dal working group potrebbero non essere sufficienti se non vengono supportate in ogni fase della comunicazione tra gli utenti; queste caratteristiche non sono auspicabili solo per fornire un miglior servizio all'utente ma anche per evitare che un servizio venga preso di mira da malintenzionati. Infine, un ruolo importante sia per sé stesso che per le altre persone è giocato dall'utente finale che utilizza WebRTC: deve essere sempre chiaro all'utente tramite quali sito web si sta connettendo al servizio prestando attenzione che sia tramite HTTPS, deve dare importanza alle finestre di dialogo riguardo il consenso all'accesso a webcam, microfono e, ancora di più, per la condivisione dello schermo; qualunque mancanza di attenzione da parte dell'utente mette in pericolo la sua privacy ma anche quella delle altre persone che entrano in contatto con lui e, potenzialmente, anche di chiunque altro se venisse utilizzato per un attacco DDoS.

WebRTC è certamente un punto di svolta nella comunicazione sul web e nel campo della sicurezza perché sta dimostrando quanto sia importante implementarla fin dal primo momento e non aggiungerla successivamente e a parte. Ha portato alla nascita di moltissimi nuovi servizi e applicazioni e non è difficile immaginarlo come standard ancora per molto tempo siccome è ancora in fase di pieno sviluppo e molte funzionalità sono ancora in discussione, soprattutto per poterne garantire un uso sicuro agli utenti.

Bibliografia

[1] P. Edholm, «Facebook Messenger - The Biggest WebRTC App?,» 13 Gennaio 2016. [Online]. Available: http://www.webrtcworld.com/topics/webrtc-world/articles/415961-facebook-messenger-biggest-webrtc-app.htm.

[2] WeAreSocial.com, «Digital In 2016 - Analisi,» 28 Gennaio 2016. [Online]. Available: http://wearesocial.com/it/blog/2016/01/report-digital-social-mobile-in-2016.

[3] Wikipedia, «Internet,» [Online]. Available: https://it.wikipedia.org/wiki/Internet.

[4] J. Kurose e K. Ross, Reti di calcolatori e internet - Un approccio top-down, Milano: Pearson Education, 2013.

[5] N. I. o. S. a. Technology, An Introduction to Computer Security: The NIST Handbook, Special Publication 800–12, 1995.

[6] S. Bellovin e W. Cheswick, «Network Firewalls,» *IEEE Communications*, pp. 50-57, September 1994.

[7] Wikipedia, «RSA,» [Online]. Available: https://it.wikipedia.org/wiki/RSA.

[8] T. L. I. Project, 26 Gennaio 2006. [Online]. Available: http://www.linfo.org/authentication.html.

[9] S. Corporation, Aprile 2016. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-21-2016-en.pdf.

[10] S. Karnouskos, «Stuxnet Worm Impact on Industrial Cyber-Physical System Security,» 2011. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.7495&rep=rep1&type=pdf.

[11] W. Stallings, Network Security Essentials - Applications and Standards - IV Edition, Pearson Education, 2011.

[12] Carl, Kesidis, Brooks e Rai, «Denial-of-service attack-detection techniques,» *IEEE Internet Computing,* pp. 82-89, 23 Gennaio 2006.

[13] Ansari, Rajeev e Chandrashekar, «Packet sniffing: a brief introduction,» *IEEE Potentials,* pp. 17-19, 22 Gennaio 2003.

[14] Hu e Mao, «Accurate Real-time Identification of IP Prefix Hijacking,» in *2007 IEEE Symposium on Security and Privacy (SP '07)*, Oakland, California, 20-23 Maggio 2007.

[15] The Open Web Application Security Project, «Session hijacking attack,» [Online]. Available: https://www.owasp.org/index.php/Session_hijacking_attack.

[16] Noiumkar e Chomsiri, «Top 10 Free Web-Mail Security Test Using Session Hijacking,» in *Convergence and Hybrid Information Technology, ICCIT '08.*, 2008.

[17] WebRTC, «WebRTC Frequent Asked Question,» 2016. [Online]. Available: https://www.webrtc.org/faq.

[18] S. Dutton, «Getting Started with WebRTC,» 21 Febbraio 2014. [Online]. Available: https://www.html5rocks.com/en/tutorials/webrtc/basics/.

[19] Uberti e Jennings, «Javascript Session Establishment Protocol,» 21 Ottobre 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-jsep-17.

[20] Nandakumar, «SDP for the WebRTC,» 23 Febbraio 2013. [Online]. Available: https://tools.ietf.org/id/draft-nandakumar-rtcweb-sdp-01.html.

[21] W3C WebRTC Working Group, «WebRTC 1.0: Real-time Communication Between Browsers,» 2016 Settembre 2016. [Online]. Available: https://www.w3.org/TR/webrtc/.

[22] W3C WebRTC Working Group, «Media Capture and Streams,» 2016 Maggio 2016. [Online]. Available: https://www.w3.org/TR/mediacapture-streams/.

[23] Wikipedia, «Network address translation,» 2016. [Online]. Available: https://it.wikipedia.org/wiki/Network_address_translation.

[24] Rosenberg, Mahy, Matthews e Wing, «Session Traversal Utilities for NAT (STUN) (RFC5389),» Ottobre 2008. [Online]. Available: https://tools.ietf.org/html/rfc5389.

[25] Mahy, Matthews e Rosenberg, «Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN) (RFC5766),» Aprile 2010. [Online]. Available: https://tools.ietf.org/html/rfc5766.

[26] Rosenberg, «Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,» Aprile 2010. [Online]. Available: https://tools.ietf.org/html/rfc5245.

[27] Rosemberg, Keranen, Lowekamp e Roach, «TCP Candidates with Interactive Connectivity Establishment (ICE),» Marzo 2012. [Online]. Available: https://tools.ietf.org/html/rfc6544.

[28] Jesup, Loreto e Tuexen, «WebRTC Data Channels,» 4 Gennaio 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-data-channel-13.

[29] Jesup, Loreto e Tuexen, «WebRTC Data Channel Establishment Protocol,» 1 Aprile 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-data-protocol-08.

[30] D. Dias, «WebRTC Explorer,» Marzo 2016. [Online]. Available: https://github.com/diasdavid/webrtc-explorer.

[31] Grigorik e O'Reilly, «High Performance Browser Networking,» 2013. [Online]. Available: https://hpbn.co/webrtc/.

[32] P. Thermos, «Two attacks against VoIP,» 2 Novembre 2010. [Online]. Available: https://www.symantec.com/connect/articles/two-attacks-against-voip.

[33] T. Levent-Levi, «WebRTC and Man in the Middle Attacks,» 12 Giugno 2015. [Online]. Available: https://webrtchacks.com/webrtc-and-man-in-the-middle-attacks/.

[34] Feher, Sidi, Shabtai e Puzis, «The Security of WebRTC,» *Ben-Gurion University of the Negev*, 2015.

[35] Rosenberg e Schulzrinne, «An Offer/Answer Model with the Session Description Protocol (SDP),» Giugno 2002. [Online]. Available: https://tools.ietf.org/html/rfc3264.

[36] Rescorla, «WebRTC Security Architecture,» 8 Giugno 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-security-arch-12.

[37] Rescorla, «Security Considerations for WebRTC,» 26 Febbraio 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-security-08.

[38] Perumal, Wing, Ravindranath, Reddy e Thomson, «Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness,» Ottobre 2015. [Online]. Available: https://tools.ietf.org/html/rfc7675.

[39] Rescorla e Modadugu, «Datagram Transport Layer Security Version 1.2,» Gennaio 2012. [Online]. Available: https://tools.ietf.org/search/rfc6347.

[40] Sheffer, Porticor e Saint-Andre, «Summarizing Known Attacks on Transport Layer Security (TLS),» ietf.org, Febbraio 2015. [Online]. Available: https://tools.ietf.org/html/rfc7457.

[41] Rizzo e Duong, «The CRIME Attack,» in *EKOparty Security Conference*, Buenos Aires, Argentina, 2012.

[42] Schulzrinne, Casner, Frederick e Jacobson, «RTP: A Transport Protocol for Real-Time Applications,» Luglio 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3550.txt.

[43] Baugher, McGrew, Naslund, Carrara e Norrman, «The Secure Real-time Transport Protocol (SRTP),» Marzo 2004. [Online]. Available: https://www.ietf.org/rfc/rfc3711.txt.

[44] Alexander, Wijesinha e Karne, «An Evaluation of Secure Real-time Transport Protocol (SRTP) Performance for VoIP,» in *Third International Conference on Network and System Security*, 2009.

[45] Perkins, Westerlund e Ott, «Web Real-Time Communication (WebRTC): Media Transport and Use of RTP,» 17 Marzo 2016. [Online]. Available: https://tools.ietf.org/html/draft-ietf-rtcweb-rtp-usage-26.

[46] Ott e Carrara, «Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF),» ietf.org, Febbraio 2008. [Online]. Available: https://tools.ietf.org/html/rfc5124.

[47] Andreasen, Baugher e Wing, «Session Description Protocol (SDP) Security Descriptions for Media Streams,» Luglio 2006. [Online]. Available: https://tools.ietf.org/html/rfc4568.

[48] Zimmermann, Johnston e Callas, «ZRTP: Media Path Key Agreement for Unicast Secure RTP,» Aprile 2011. [Online]. Available: https://tools.ietf.org/html/rfc6189.

[49] Aghila e Chandirasekaran, «An Analysis of VoIP Secure Key Exchange Protocols against Man-in-the-Middle Attack,» *International Journal of Computer Applications*, pp. 46-52, 2011.

[50] McGrew e Rescorla, «Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP),» Maggio 2010. [Online]. Available: https://tools.ietf.org/html/rfc5764.

[51] Rescorla, «SDES,» in *IETF 87*, Berlino, 2013.

[52] Stewart, «Stream Control Transmission Protocol,» ietf.org, Settembre 2007. [Online]. Available: https://tools.ietf.org/html/rfc4960.

[53] Tuexen, Stewart, Jesup e Loreto, «DTLS Encapsulation of SCTP Packets,» ietf.org, 24 Gennaio 2015. [Online]. Available: https://tools.ietf.org/html/draft-ietf-tsvwg-sctp-dtls-encaps-09.

[54] C. Reis, A. Barth e C. Pizano, «Browser Security: Lessons from Google Chrome,» *ACM Queue*, 2009.

[55] Rescorla, «Notes on security for browser-based screen/application sharing,» 11 Marzo 2013. [Online]. Available: https://lists.w3.org/Archives/Public/public-webrtc/2013Mar/0024.html.

[56] Barnes e Thomson, «Browser-to-Browser Security Assurances for WebRTC,» *IEEE Internet Computing* , vol. 18, pp. 11-17, Novembre-Dicembre 2014.

[57] V. Beltran, E. Bertin e N. Crespi, «User Identity for WebRTC Services: A Matter of Trust,» *IEEE Internet Computing*, vol. 18, pp. 18-25, Novembre-Dicembre 2014.