WebRTC e Man in the Middle con Janus
esame di Network Security

Farina Giorgio - Mat. M63/000861 Luca Giamattei - Mat. M63/000825

16 febbraio 2021

Indice

Capitolo 1

WebRTC e Man in the Middle

1.1 Introduzione

In questo elaborato è stata sviluppata un' applicazione web che consente di effettuare videochiamate tra più utenti all'interno di una "stanza virtuale". L'applicazione è basata sulla tecnologia open source WebRTC, disponibile su tutti i moderni browser, che consente di effettuare videochat P2P in tempo reale. A supporto di tale scelta tecnologica si è deciso di utilizzare il WebRTC server Janus. Questo infatti fornisce delle API in javascript che astraggono i dettagli implementativi della tecnologia e ne consentono un rapido e semplice utilizzo in svariati casi d'uso. In particolare sono messi a disposizione una serie di "plugin" da aggiungere al codice base così da personalizzarne l'utilizzo. In questo caso si è deciso di utilizzare il plugin "VideoRoom" che fornisce l'implementazione di una SFU (Selective Forwarding Unit). Una SFU è una entità in grado di ricevere più stream e decidere ognuno di questi a quali partecipanti deve essere reindirizzato. Uno dei principali vantaggi di questa architettura a confronto con una P2P è la scalabilità. Infatti con un P2P ogni peer connesso tra di loro caricherà flussi multimediali (audio, video, ecc.) N-1 volte (dove N è il numero di persone nella chiamata).

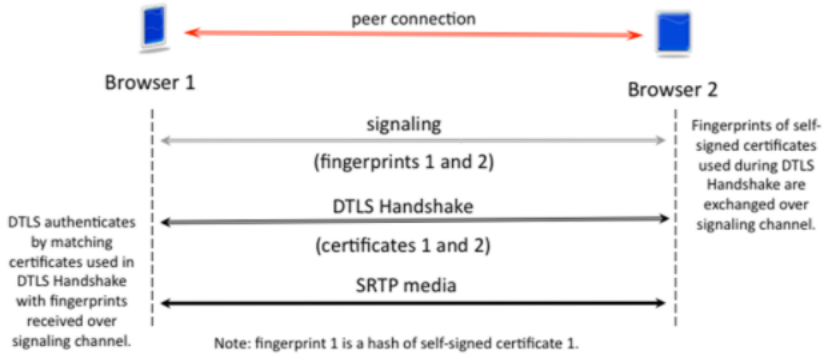
L'utilizzo di una SFU consente ai peer di caricare file multimediali e flussi di dati solo una volta. Il server quindi gestisce la distribuzione agli N-1 partecipanti. Questo essenzialmente aiuta a limitare la larghezza di banda consumata da tutti i partecipanti e consente di avere un numero maggiore di partecipanti in chiamata.

Si è deciso di sfruttare l'applicazione per eseguire un attacco Man-In-The-Middle (MITM) così che l'attaccante possa vedere gli stream dati delle vittime senza che questi se ne accorgano. Come vedremo in breve, questo attacco può essere svolto in diversi modi, e nei paragrafi finali sarà mostrato il metodo preso in considerazione.

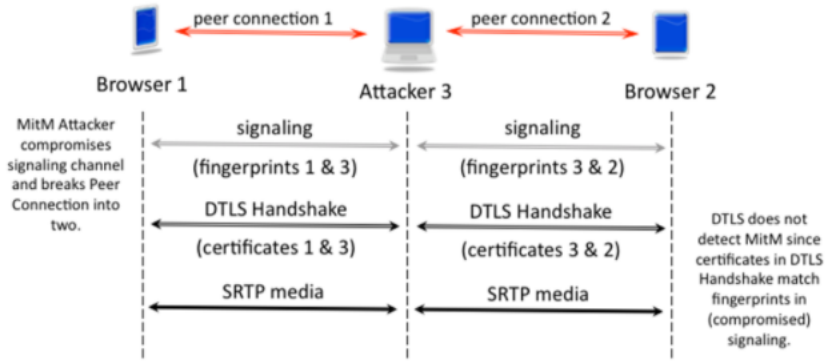
1.2 WebRTC e MITM

Come anticipato WebRTC è una tecnologia completamente peer-to-peer per lo scambio in tempo reale di audio, video e dati. Affinché due dispositivi su reti diverse possano localizzarsi l'un l'altro e concordare dei paramentri per la trasmissione multimediale è necessaria una forma di individuazione e negoziazione del formato multimediale. Questo processo è chiamato segnalazione. In questa fase, come è possibile notare dalla figura, vengono scambiati dei fingerprint (hash dei certificati), utilizzati per l'autenticazione delle parti durante l'handshake di DTLS. L'utilizzo di

un canale crittato con DTLS è obbligatorio in WebRTC, a differenza del protocollo per la fase di segnalazione, il quale non è specificato.



Nella maggior parte dei casi allora questa risulta la più grossa vulnerabilità nel binding di peer in una connessione con WebRTC. In particolare, compromettendo il server di segnalazione, è possibile eseguire un attacco Man-In-The-Middle nel seguente modo:



Quando uno degli utenti chiama l'altro stabilendo una nuova connessione peer WebRTC, mettendosi nel mezzo, è possibile intercettare l'offerta SDP e creare una nuova connessione peer all'altro utente, inviando la propria offerta SDP. L'applicazione esegue la negoziazione dell'offerta / risposta SDP e l'handshake DTLS con ciascuno degli utenti e il risultato dello scambio saranno due connessioni peer al posto di una, entrambe verso l'applicazione JavaScript MITM. Ciascuna connessione peer è considerata completamente autenticata da entrambi i browser dei peer, sfortunatamente però, autenticate dall'attaccante. Superata la fase di stabilimento della connessione sarà sufficiente effettuare l'inoltro dei flussi multimediali provenienti da ambo i lati sulle rispettive connessioni opposte per poter leggere i flussi senza che i peer si accorgano di nulla.

Per quanto visto, ne consegue che l'unica protezione possibile contro questo tipo di attacco consiste nell'autenticare e controllare l'integrità dello scambio SDP.

1.3 Scenario corretto Janus

Si introduce adesso lo scenario preso in considerazione, senza la presenza del man-in-the-middle. Un peer, interfacciandosi con l'applicazione web, può creare una nuova room sul server janus o prendere parte ad una già esistente. In uno scenario senza attaccanti, tutti i flussi, compresa la segnalazione, passano attraverso 3 sole entità nello stabilimento di una connessione tra 2 peer: i due utenti e il

server Janus. Nell'implementazione senza multistream di Janus ogni utente all'interno della room avrà 1 connessione in modalità publisher e N-1 connessioni subscribe per N peer all'interno della stanza. Nello specifico, le funzionalità offerte ad un utente sono:

- Creazione di una nuova stanza: all'utente sarà chiesto di inserire il nome della stanza da creare (numerico) e nel caso in cui tutti i controlli siano passati, l'utente si ritroverà all'interno della nuova stanza in qualità di publisher. La creazione di una stanza da parte di un utente prevede l'invio di due richieste. Una di "create", utile alla creazione e memorizzazione della configurazione della nuova room su server, e una di "join". Infatti l'utente, creata la stanza, dovrà effettuare ugualmente una join per accedervi, e lo farà nella callback di "success" della create (riferirsi al codice per maggiori dettagli).
- Join in una stanza esistente: l'utente potrà inserire il nome di una stanza preesistente in cui vuole entrare, e nel caso il valore immesso nel prompt corrisponda effettivamente ad una stanza disponibile, verrà portato all'interno in qualità di publisher. Questo significa che la stanza deve essere già stata precedentemente creata da un utente o potrebbe essere una stanza di "default", quindi sempre attiva.

I peer all'interno della stanza sono gli unici a poter vedere gli stream audio e video di ogni utente, ricevendo i flussi multimediali dalla SFU Janus come brevemente accennato nel paragrafo introduttivo.

Capitolo 2

Descrizione scenari

2.1 Attacco

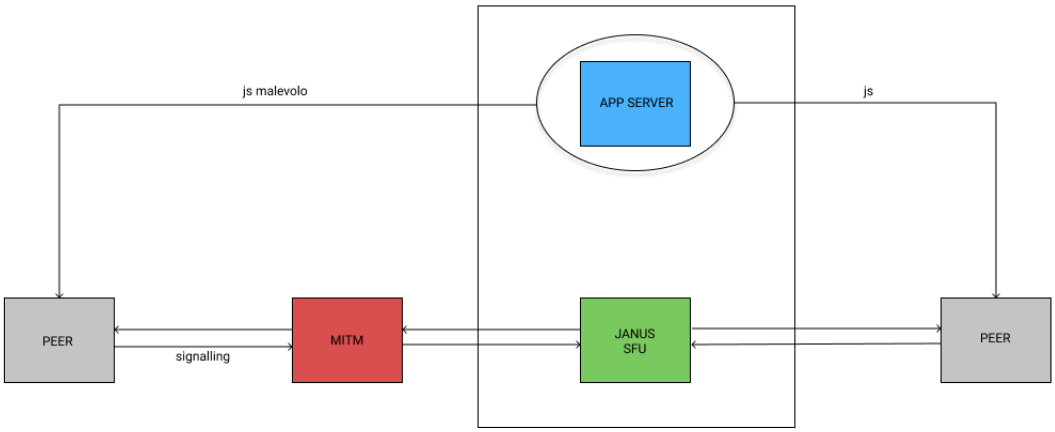
2.1.1 Potenziali vulnerabilità

Si consideri uno scenario generico dove un application server consegna un file javascript a un client affinchè quest'ultimo possa interagire con il server janus per instaurare una videoroom con altri utenti. Per portare a termine un attacco di tipo man in the middle ed effettuare lo snooping della conversazione sono state individuate due minacce derivate da due potenziali vulnerabilità:

- La prima consiste nello sfruttare una connessione poco sicura tra il peer client e l'application server. Tale vulnerabilità potrebbe essere sfruttata per passare al client un js malevolo al posto del valido javascript client. In questo modo sarebbe possibile manomettere la connessione inserendo nella comunicazione tra il peer client e il Janus SFU un ente malevolo.

- La seconda soluzione consiste nello sfruttare una possibile connessione poco sicura tra il Peer client e il server di segnalazione, che in questo caso specifico è un server Janus, per intercettare e manomettere l'handshake di signaling. Si osservi che in WebRTC non vi è un protocollo standard per la segnalazione; Janus usufruisce del protocollo http (long poll) per instaurare una connessione bidirezionale tra il server e il client, quando non è possibile adoperare le WebSocket.

In questo elaborato si è scelto di approfondire il primo scenario e sarà presentato nella sezione successiva.

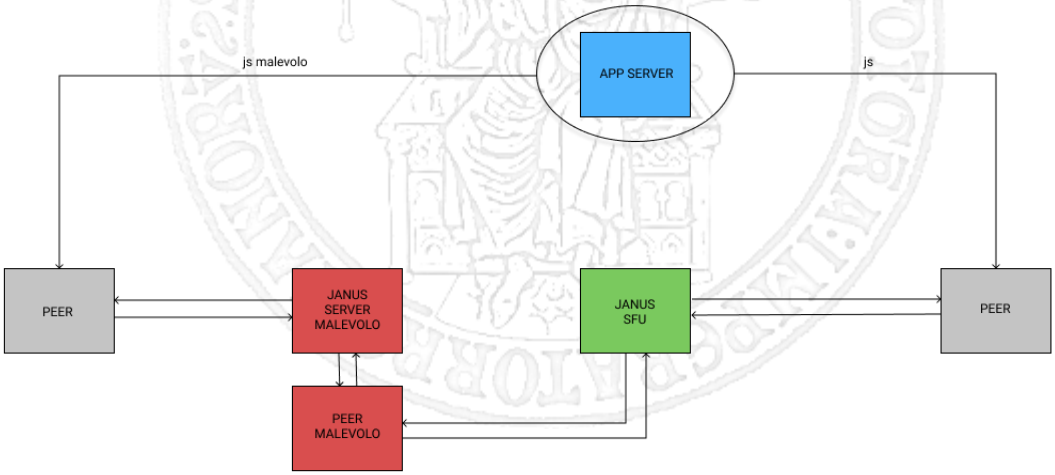


2.1.2 Attacco con javascript malevolo

In questo scenario si ha un application server che, su richiesta del client, consegna un file javascript per istaurare una connessione con l'SFU Janus.

Le entità in questo scenario sono le seguenti:

- Application server: application server non malevolo che, su richiesta del client, consegna un file javascript per istaurare una connessione con l'SFU Janus al fine di creare o unirsi a una videoroom. La risposta dell'app server verso il client si suppone possa essere manomessa.
- PeerHack: Peer hackerato che ha ricevuto dall'application server un javascript manomesso.
- Peer: Peer non hackerato che ha ricevuto la corretta risposta dall'application server con un javascript non manomesso.
- MITM: l'entità che da un lato comunica con il peerHack e dall'altro con l'SFU Janus.
- Janus SFU: server Janus in modalità plugin videoroom.



2.1.2.1 Entità MITM

L'entità MITM deve far credere al PeerHack che sta interagendo con il server janus, in particolare con la video room a cui partecipa. Tuttavia deve anche apparire all'SFU di Janus come un peer client.

Per raggiungere tale intento si è pensato di costruire l'entità MITM come un insieme di più enità:

- JanusMITM: che funge da SFU janus per il Peer Hack
- PeerMITM: javascript client che finge da peer client per l'SFU Janus
- Application Server MITM: Ottiene informazioni dal Peer Hack come il "nome della stanza" e le consegna al Peer MITM per poi agire in modo duale con l'SFU Janus.

2.1.2.2 Sequenza di attacco

Il flusso seguito da questo elaborato è il seguente:

- Il client hackerato riceve un javascript malevolo che, senza alterare l'interfaccia grafica:
  - Istaura una Peer connection con l'SFU Janus MITM su una stanza di nome "1", piuttosto che con SFU Janus su una stanza richiesta dal client
  - Invia il "nome della room originale", quello richiesto realmente dal client, all'application server MITM
- Il PeerMITM, essendo di default già sottoscritto alla stanza "1" del Janus MITM, riceve lo stream (audio e video) del client hackerato e aspetta di ricevere il "nome della room originale " dall'application server MITM per poi:
  - Istaurare con l'SFU Janus una Peer Connection e fare la join alla room di nome "nome della room originale "
  - Inoltrare lo stream del peer hackerato a Janus SFU
- Lo stream del peer hackerato è anche "attaccato" all'oggetto video HTML del PeerMITM
- Il PeerMITM, una volta fatta la join alla room sul Janus SFU, riceverà uno o più stream del partecipanti alla room, i quali saranno inoltrati all'SFU Janus MITM per poi essere a loro volta recapitati al Peer Hackerato.

Capitolo 3

Esecuzione dell'attacco

3.1 Esempio Pratico

3.1.1 Configurazione dell'ambiente

L'ambiente per semplicità di debugging è stato cofigurato nel modo seguente:

- SFU JanusMITM: in esecuzione su localhost su porta 8080
- SFU Janus: in esecuzione su localhost su porta 8088 (quella di default)
- Application Server in esecuzione su localhost sulla porta 4000
- Application Server MITM in esecuzione su localhost sulla porta 3000

3.1.2 Attivazione del MITM

L'SFU Janus MITM è stato messo in esecuzione in local host sulla porta 8080.

Successivamente il Peer MITM si ottiene con una richiesta get all'indirizzo http://localhost:3000/mainpageMITM.html all'application server MITM.

Il codice javascript creerà una stanza su Janus MITM di nome '1' per poi fare la "join".

Inoltre il Peer MITM è in ascolto di un evento "chat message" da parte dell'application server MITM attraverso il quale sarà informato del "nome della room originale".

3.1.3 Attivazione del Peer Hack

Il peer hack ottiene, a causa di una manomissione, un codice javascript in risposta a una richiesta GET all 'indirizzo http://localhost:4000/mainpage.html (indirizzo http://localhost:4000/mainpageHack.html per ottenere il codice js malevolo).

Il codice js differisce da quello normale perchè

- l'indirizzo dell'SFU è forgiato con quello malevolo
- vi è uno snippet di codice che inoltra il "nome della room originale richiesta dall'utente" all'application server malevolo
- effettua la "join" sulla stanza "1" dell'SFU malevolo.

Il PeerMITM, essendo di default già sottoscritto alla stanza "1" del Janus MITM, riceve lo stream (audio e video) del client hackerato e aspetta di ricevere il "nome della room originale " dall'applicatioon server MITM per poi

- Istaurare con l'SFU Janus non malevolo una Peer Connection e unirsi alla room di nome "nome della room originale richiesta dall'utente"
- inoltrare lo stream del peer hackerato

In particolare il codice js del PeerMITM (mainpageMITM.js), una volta istanziato un handler RemoteFeedMITM per ricevere lo stream da parte del PeerHack, oltre a mostrare lo stream sul display, richiamerà la funzione di joinToRoom(), la quale

- Attende il "nome della room originale"
- Invia la richiesta di "join" con nome room pari al "nome room originale richiesta dall'utente" verso l'SFU Janus non malevolo

Attenzione: L'username potrebbe essere preso dagli attributi dello stream, tuttavia si è preferito lasciare un nome "MITM" per identificare i flussi media visivamente. Con questa configurazione, quindi, il MITM verrebbe scoperto.

Alla ricezione dell'evento "joined" sarà invocata la funzione publishOwnFeedClientMITMto-Janus() la quale si occuperà di pubblicare lo stream ricevuto dal PeerHack allo JanusSFU non malevolo.

A questo punto il PeerMITM, una volta fatta la join alla room sul Janus SFU, riceverà uno o più stream dei partecipanti alla room, i quali dovranno essere inoltrati all'SFU Janus MITM per poi essere a loro volta recapitati al Peer Hackerato.

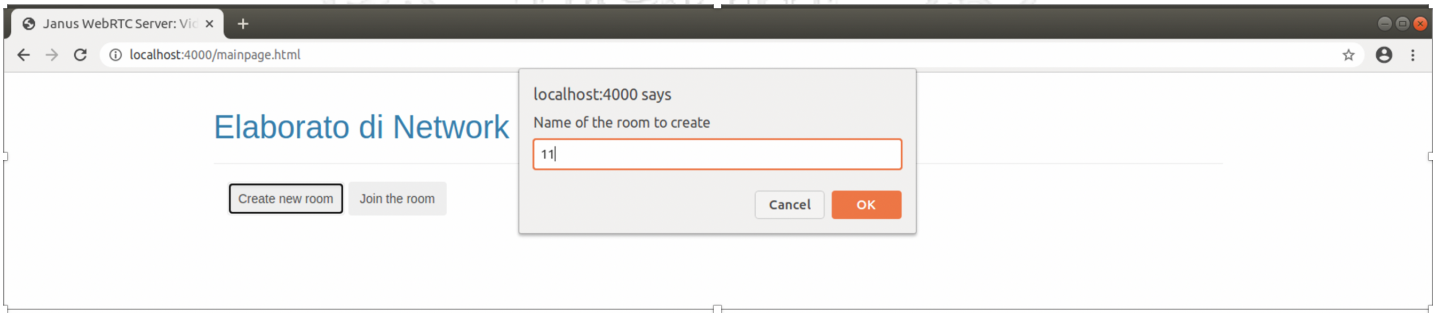
Il ragionamento è lo stesso e anche l'implementazione.

Per comprendere meglio il codice è stata impiegata la seguente nomenclatura:

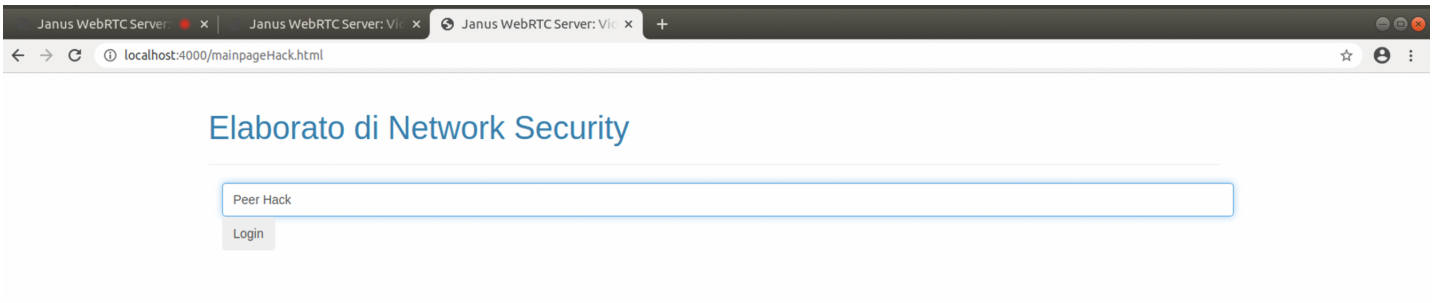
- Gli handler "handler_vrMITM" e "remoteFeedMITM" si interfacciano con l'SFU janus MITM
- gli handler "handler_vr" e "remoteFeed" si interfacciano con l'SFU janus non malevolo.

3.1.4 Esecuzione e interfacce utente

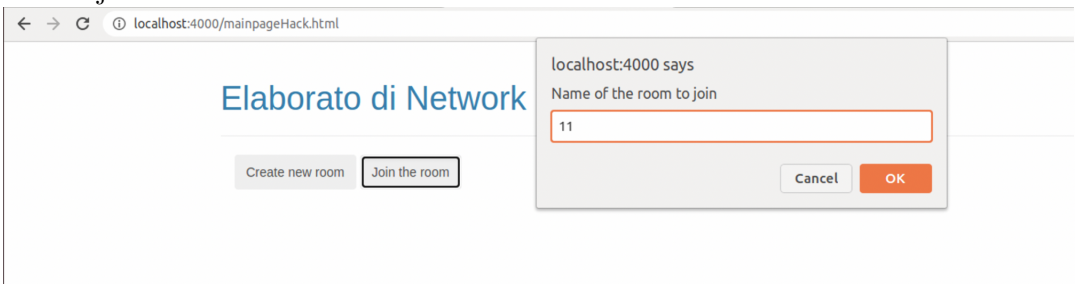
- Esecuzione del MITM
  - node ./janusMITM -> application server in esecuzione sulla porta 3000 del local host
  - URL http://localhost:3000/mainpageMITM.html
- Il Peer (non hackerato) si logga all'applicazione come "Peer" e crea la stanza "11"



- Il Peer Hackerato si logga come "PeerHack"



- e fa la join alla stanza "11"



- Visuale dell'attacco: come si può osservare il MITM (quello centrale) non ha la lucina rossa perchè non richiede l'accesso al microfono e al video della webcam; esso inoltra solo degli stream già esistenti.