5. Case of study: Twitch.tv

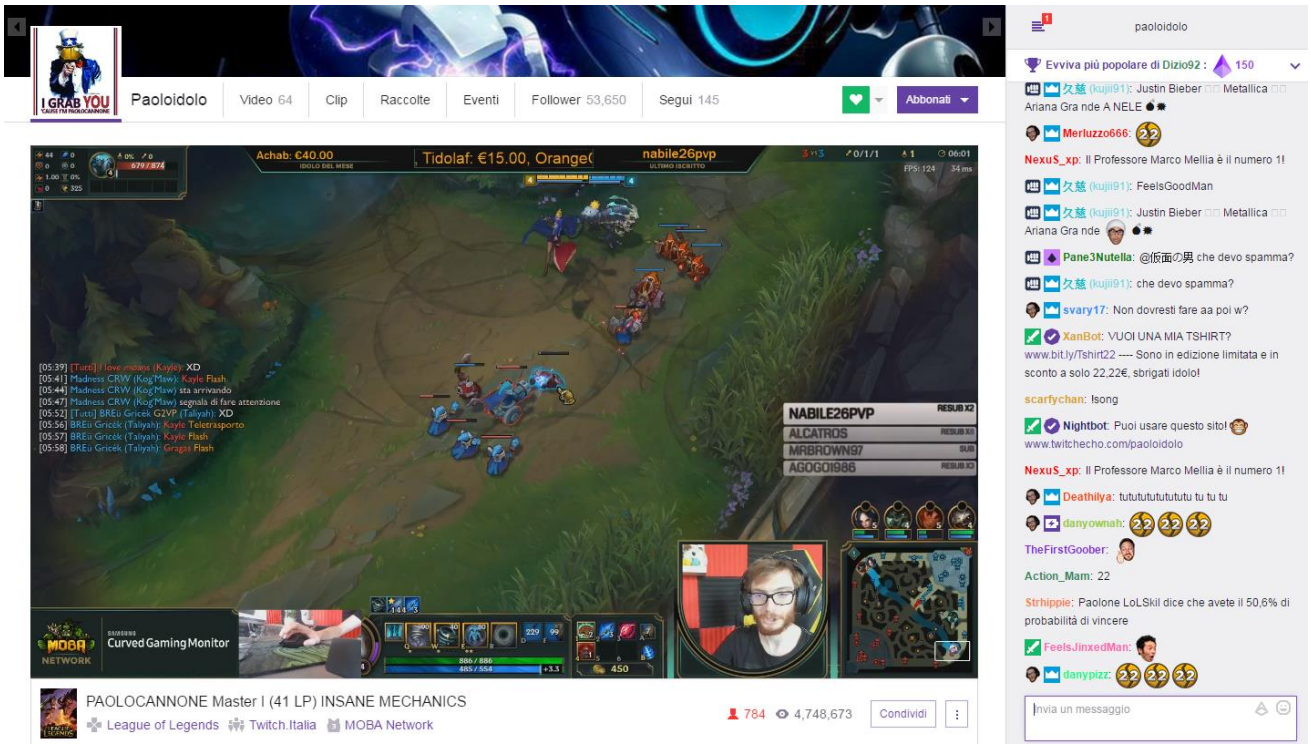
5.1 Introduzione

Twitch.tv è una piattaforma di video streaming di proprietà di Twitch Interactive, filiale di Amazon. Il sito si focalizza principalmente sui videogiochi, tra cui $playthroug^1$ di videogiochi, trasmissioni di $eSport^2$ e, più recentemente, trasmissioni musicali. I contenuti del sito possono essere visualizzati in diretta o on demand.

Il sito consente di filtrare le ricerche per tipo di gioco ed in seguito scegliere tra gli streaming proposti quello che più si aggrada. Per ogni streaming è presente una chat grazie alla quale, solo se si possiede un account, è possibile interagire con tutti gli utenti che stanno visualizzando lo streaming in quel momento. È possibile inoltre effettuare una donazione a chi esegue lo streaming come segno di supporto, la piattaforma infatti è del tutto gratuita e non richiede alcun tipo di abbonamento né da parte degli utenti né da parte di chi esegue lo streaming. Gli "streamer" spesso utilizzano una webcam per mostrare le proprie reazioni emotive.

Ultimamente Twitch dispone anche di un client che permette di effettuare chiamate vocali e/o video oppure semplicemente scambio di messaggi testuali.

Ecco come si presenta un tipico streaming su Twitch.



5.2 Obiettivi e Strumenti utilizzati

In questa trattazione verranno affrontati tutti i dettagli tecnici relativi ai servizi principali offerti da Twitch. Un elemento interessante è la varietà di protocolli utilizzati all'interno della stessa pagina web, in virtù

---

[1] L'atto di giocare un gioco dall'inizio alla fine.
[2] Abbreviativo di Sport Elettronici: indica il giocare videogiochi a livello competitivo organizzato.

della necessità di offrire servizi di natura eterogenea. Maggiori attenzioni verranno rivolte sulla Fase di Login, Chat e Video Streaming.

Per il test sono stati utilizzati i seguenti strumenti:

- Laptop con sistema operativo Linux
- Chromium-browser, con le funzionalità offerte dal comando “ispeziona elemento”
- Wireshark e Ermes (https://www.ermes.polito.it/project)

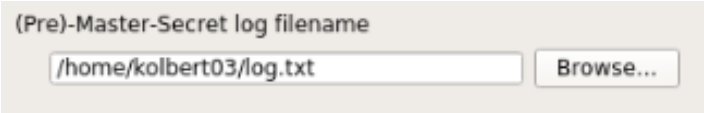
5. 3 Decriptare il traffico generato

Nonostante lo streaming sia accessibile a chiunque, le connessioni relative ai vari servizi offerti da twitch risultano completamente cifrate, per questo motivo è stato necessario de-crittografare tutto il traffico generato.

È stato necessario esportare la variabile d'ambiente SSLKEYLOGFILE in un file accessibile all'utente, in seguito è stato avviato il browser all'interno del medesimo terminale.

```
kolbert03@Ares ~ $ export SSLKEYLOGFILE='/home/kolbert03/log.txt'
kolbert03@Ares ~ $ chromium-browser
Using PPAPI flash.
 --ppapi-flash-path=/usr/lib/adobe-flashplugin/libpepflashplayer.so --ppapi-flash-version=
[WARNING:flash/platform/pepper/pep_module.cpp(63)] SANDBOXED
```

Inoltre, si è settato in wireshark il percorso del file in cui la variabile d'ambiente produrrà il proprio output, in questo modo wireshark è nelle condizioni da poter vedere in chiaro tutto il traffico generato.



(Modifica->Preferenze->Protocols->SSL)

5.4 Fase di Login

Dopo essersi recati all’home page del sito in questione (https://www.twitch.tv/) si è effettuata la procedura di login inserendo le proprie credenziali. Utilizzando Wireshark ed Ermes ed analizzando rispettivamente il traffico catturato e il file log generato, è stato possibile ricavare alcune utili informazioni.

Per visualizzare la “conversazione” tra il client ed il server di login si è usato il seguente filtro sui pacchetti catturati da wireshark: $ip.addr == 52.27.1.153$, dove l’indirizzo IP in questione, situato a Wilmington (Stati Uniti), è uno dei server che Twitch usa per l’autenticazione dei propri utenti. L’apertura della connessione è stata effettuata tramite three-way handshake, procedura utilizzata per instaurare, in modo affidabile, una connessione TCP tra due host che prevede lo scambio di tre messaggi tra client e server.

```
192.168.1.132   52.27.1.153     TCP      74 38358→443 [SYN] Seq=0 Win=29
52.27.1.153     192.168.1.132   TCP      74 443→38358 [SYN, ACK] Seq=0 A
192.168.1.132   52.27.1.153     TCP      66 38358→443 [ACK] Seq=1 Ack=1
```

Successivamente vi è uno scambio di pacchetti che utilizzano il protocollo TLS. Il protocollo in oggetto, permette una comunicazione sicura tra client e server permettendo lo scambio di varie informazioni.

```
192.168.1.132 52.27.1.153   TLSv1.2    272 Client Hello
52.27.1.153   192.168.1.132 TCP         66 443→38358 [ACK] Seq=1 Ack=207 Win=28160 Len=0 TSval=538297143 TSecr=340909
52.27.1.153   192.168.1.132 TLSv1.2   3966 Server Hello, Certificate, Server Key Exchange, Server Hello Done
192.168.1.132 52.27.1.153   TCP         66 38358→443 [ACK] Seq=207 Ack=3901 Win=37120 Len=0 TSval=341037 TSecr=538297143
192.168.1.132 52.27.1.153   TLSv1.2    192 Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
52.27.1.153   192.168.1.132 TLSv1.2    117 Change Cipher Spec, Encrypted Handshake Message
```

Il pacchetto Client Hello contiene degli attributi che il client manda al server, tra cui:

- **Session ID:** contiene l'ID della sessione che il client vorrebbe usare durante la connessione.
- **Chiper Suite:** contiene l'elenco degli algoritmi di cifratura supportati dal client in ordine di preferenza.

Il server risponde con un Server Hello comunicando l'ID per la connessione e l'algoritmo scelto tra quelli proposti in precedenza dal server.

Attraverso i pacchetti Client Key Exchange e Server Key Exchange si generano le chiavi per la cifratura e per l'autenticazione dei dati provenienti da livello applicativo.

Il messaggio finale è Server Hello Done che indica la fine la fine della fase di "Hello" al client. Dopodiché, con l'invio di Change Chiper Spec si comunica che la connessione è cifrata e i dati possono essere trasmessi in modo sicuro.

```
192.168.1.132 52.27.1.153   TLSv1.2          566 Application Data
52.27.1.153   192.168.1.132 TCP               66 443→38358 [ACK] Seq=3952 Ack=1781 Win=30976 Len=0 TSval=538297374 TSecr=341151
52.27.1.153   192.168.1.132 TCP               66 443→38358 [ACK] Seq=3952 Ack=2281 Win=33792 Len=0 TSval=538297377 TSecr=341182
52.27.1.153   192.168.1.132 TCP             2962 [TCP segment of a reassembled PDU]
192.168.1.132 52.27.1.153   TCP               66 38358→443 [ACK] Seq=2281 Ack=6848 Win=42880 Len=0 TSval=341258 TSecr=538297378
52.27.1.153   192.168.1.132 TLSv1.2          1314 Application Data
52.27.1.153   192.168.1.132 TLSv1.2          2040 Application Data
192.168.1.132 52.27.1.153   TCP               66 38358→443 [ACK] Seq=2281 Ack=10070 Win=49664 Len=0 TSval=341259 TSecr=538297378
192.168.1.132 52.27.1.153   TCP             1514 [TCP segment of a reassembled PDU]
192.168.1.132 52.27.1.153   TLSv1.2          658 Application Data
```

Ovviamente il metodo appena descritto non avviene solamente durante la fase di login ma ogniqualvolta un client instaura una connessione con un server che utilizza il protocollo *https*. Da questo momento in poi, onde evitare ridondanze, non verrà più descritto il protocollo TLS poiché la procedura risulta la medesima per ogni server contattato.

Analizzando invece il traffico http si può notare come il client chieda tramite il metodo GET la pagina di autenticazione. Il server dopo aver controllato la validità della richiesta (pagine disponibile o meno), risponde con un 200 OK e ritorna il codice html della pagina in questione.

```
192.168.1.132   52.27.1.153     HTTP      566 GET /authentications/new?client_id=369268924
52.27.1.153     192.168.1.132   HTTP     2040 HTTP/1.1 200 OK  (text/html)
192.168.1.132   52.27.1.153     HTTP      658 GET /client_event/new?client_id=369268924953
192.168.1.132   52.27.1.153     HTTP      511 GET /test_cookie HTTP/1.1
52.27.1.153     192.168.1.132   HTTP     1139 HTTP/1.1 200 OK
52.27.1.153     192.168.1.132   HTTP      274 HTTP/1.1 200 OK
```

Viene anche effettuato un test dei cookie per assicurarsi che le richieste provengano dallo stesso browser.

Dopo che il client ha ricevuto il codice html della pagina di login si è proceduto con l'inserimento, volutamente errato, di username e password. Più precisamente, lo username è stato inserito in maniera corretta (Nindo7), mentre la password inserita è stata "prvmellia" invece di "provamellia". Inoltre, grazie ai settaggi di wireshark descritti nel punto **5.3**, è stato possibile visualizzare in chiaro i dati.

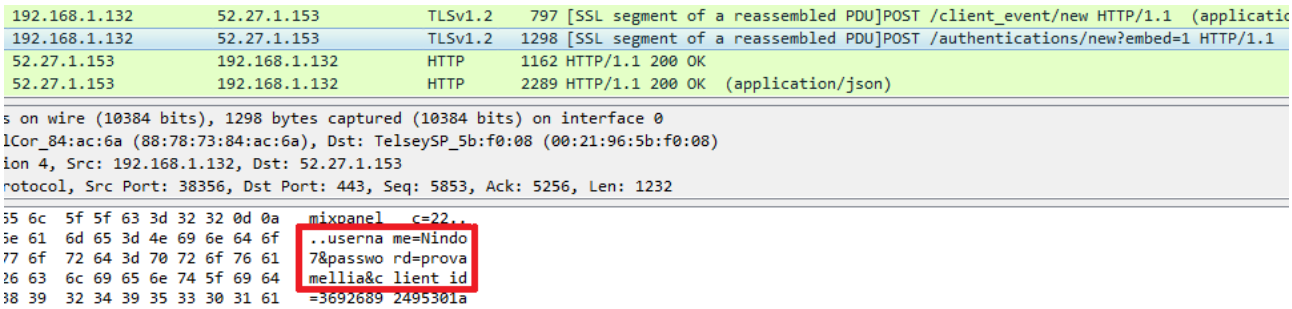
```
192.168.1.132   52.27.1.153     TLSv1.2    797 [SSL segment of a reassembled PDU]POST /client_event/new HTTP/1.1  (applicati
192.168.1.132   52.27.1.153     TLSv1.2   1298 [SSL segment of a reassembled PDU]POST /authentications/new?embed=1 HTTP/1.1
52.27.1.153     192.168.1.132   HTTP      1162 HTTP/1.1 200 OK
52.27.1.153     192.168.1.132   HTTP      1258 HTTP/1.1 400 Bad Request  (text/plain)
```

```
 on wire (10384 bits), 1298 bytes captured (10384 bits) on interface 0
Cor_84:ac:6a (88:78:73:84:ac:6a), Dst: TelseySP_5b:f0:08 (00:21:96:5b:f0:08)
on 4, Src: 192.168.1.132, Dst: 52.27.1.153
otocol, Src Port: 38358, Dst Port: 443, Seq: 5769, Ack: 11143, Len: 1232
```

```
0 6f  72 74 5f 63 6f 6f 6b 69   ; passpo rt_cooki
f 77  65 64 3d 3b 20 6d 70 5f   es_allow ed=; mp_
5 6c  5f 5f 63 3d 32 32 0d 0a   mixpanel  c=22..
e 61  6d 65 3d 4e 69 6e 64 6f   ..userna me=Nindo
7 6f  72 64 3d 70 72 76 61 6d   7&passwo rd=prvam
3 6c  69 65 6e 74 5f 69 64 3d   ellia&cl ient_id=
```

Com'è chiaramente visibile dall'immagine infatti il server risponde con 400 Bad Request poiché la procedura di autenticazione non è andata a buon fine.

Quando invece, sia username che password sono inseriti in modo corretto, il client risponde con 200 OK, indicando che l'esito della procedura è andato a buon fine.



Inoltre, scremando il file log generato da Ermes con alcuni script, è stato possibile visualizzare i dati del traffico generato in maniera più chiara e ordinata.

```
unique_id=8185cf43d10bc237;
language=it;
session_unique_id=d320ed1466ad5606;
_twitch_session_id=9733be7a2fc3a0c7ae53f639be2fe500;
api_token=f48a52aedca40d04238bfc52d8c45668;
last_login=2017-05-29+09%3A53%3A19+UTC;
login=nindo7;
name=Nindo7;
```

Come si può facilmente notare da queste informazioni vi è un settaggio per la lingua, un id per la sessione attuale di twitch, un token per le API, l'ultima data di login ed il nome dell'utente attualmente loggato.

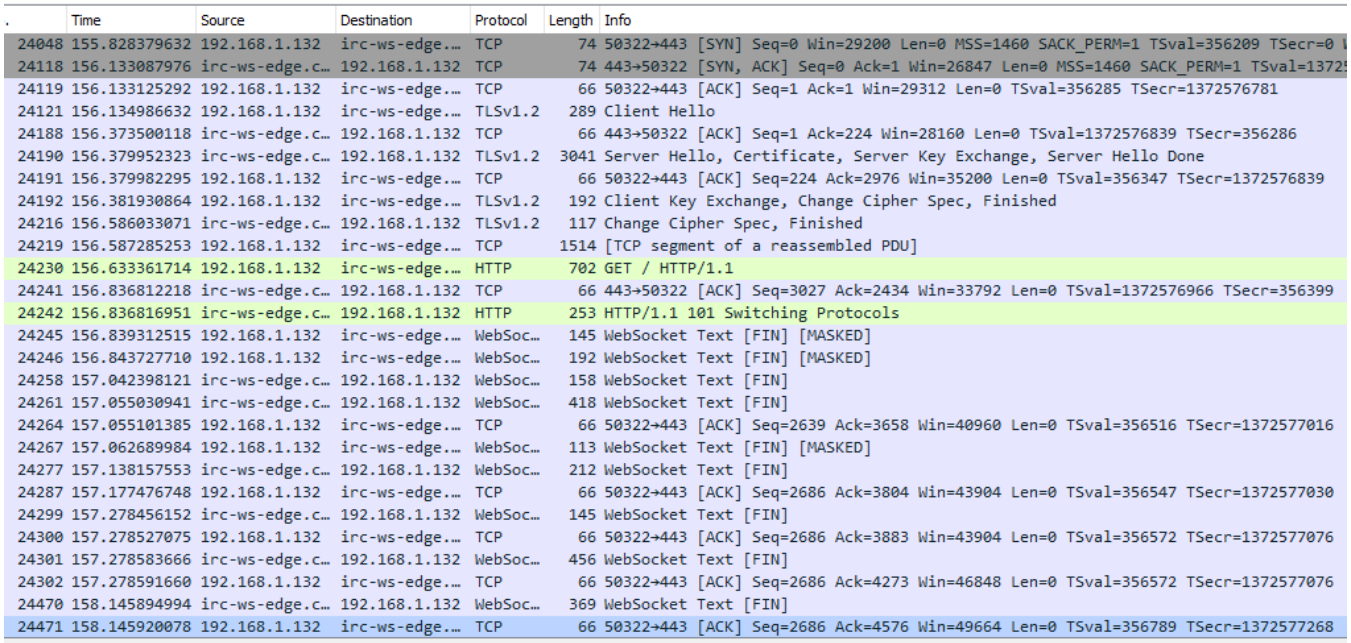
```
Req:    1496051556760   GET     api.twitch.tv   https://api.twitch.tv/v4/users/undefined/friends/recommendations
Req:    1496051602471   GET     api.twitch.tv   https://api.twitch.tv/kraken/users/85756562/status/friends
Req:    1496051603323   OPTIONS api.twitch.tv   https://api.twitch.tv/kraken/users/nindo7/friends/notifications
Req:    1496051603794   GET     api.twitch.tv   https://api.twitch.tv/kraken/users/nindo7/friends/notifications
Req:    1496051604159   OPTIONS api.twitch.tv   https://api.twitch.tv/v5/users/85756562/friends/requests?limit=21
Req:    1496051604647   OPTIONS api.twitch.tv   https://api.twitch.tv/v4/users/nindo7/friends/recommendations
Req:    1496051605637   GET     api.twitch.tv   https://api.twitch.tv/v5/users/85756562/friends/requests?limit=21
Req:    1496051606040   GET     api.twitch.tv   https://api.twitch.tv/v4/users/nindo7/friends/recommendations
```

È possibile notare anche come vengano effettuate richieste per lo status degli amici, Twitch infatti invia una notifica qualora un utente, presente nella propria lista amici, effettua il login oppure comincia a visualizzare lo stesso streaming dell'account analizzato. La stringa *kraken* presente nel path della richiesta *https://* non è nient'altro che un API fornita da Twitch.

5.5 Chat

In ogni canale è presente una chat che permette di scambiare messaggi fra gli spettatori dello streaming. Il protocollo di livello applicativo utilizzato per la chat è IRC definito in svariati RFC, ciascuno che propone implementazioni differenti. Twitch segue il protocollo definito nell RFC 1459, con qualche modifica. Volendo fornire una trasmissione cifrata il protocollo di livello inferiore utilizzato sarà TLS e proprio per questo motivo la comunicazione è avvenuta attraverso la porta 443.

È possibile filtrare tutto il traffico generato da Twitch per ottenere solamente il traffico della Chat. L'IP Address del server in questo caso è 52.42.192.125 situato a Portland, Stati Uniti.



.	Time	Source	Destination	Protocol	Length	Info
24048	155.828379632	192.168.1.132	irc-ws-edge.…	TCP	74	50322→443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=356209 TSecr=0
24118	156.133087976	irc-ws-edge.c…	192.168.1.132	TCP	74	443→50322 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PERM=1 TSval=1372
24119	156.133125292	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=356285 TSecr=1372576781
24121	156.134986632	192.168.1.132	irc-ws-edge.…	TLSv1.2	289	Client Hello
24188	156.373500118	irc-ws-edge.c…	192.168.1.132	TCP	66	443→50322 [ACK] Seq=1 Ack=224 Win=28160 Len=0 TSval=1372576839 TSecr=356286
24190	156.379952323	irc-ws-edge.c…	192.168.1.132	TLSv1.2	3041	Server Hello, Certificate, Server Key Exchange, Server Hello Done
24191	156.379982295	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=224 Ack=2976 Win=35200 Len=0 TSval=356347 TSecr=1372576839
24192	156.381930864	192.168.1.132	irc-ws-edge.…	TLSv1.2	192	Client Key Exchange, Change Cipher Spec, Finished
24216	156.586033071	irc-ws-edge.c…	192.168.1.132	TLSv1.2	117	Change Cipher Spec, Finished
24219	156.587285253	192.168.1.132	irc-ws-edge.…	TCP	1514	[TCP segment of a reassembled PDU]
24230	156.633361714	192.168.1.132	irc-ws-edge.…	HTTP	702	GET / HTTP/1.1
24241	156.836812218	irc-ws-edge.c…	192.168.1.132	TCP	66	443→50322 [ACK] Seq=3027 Ack=2434 Win=33792 Len=0 TSval=1372576966 TSecr=356399
24242	156.836816951	irc-ws-edge.c…	192.168.1.132	HTTP	253	HTTP/1.1 101 Switching Protocols
24245	156.839312515	192.168.1.132	irc-ws-edge.…	WebSoc…	145	WebSocket Text [FIN] [MASKED]
24246	156.843727710	192.168.1.132	irc-ws-edge.…	WebSoc…	192	WebSocket Text [FIN] [MASKED]
24258	157.042398121	irc-ws-edge.c…	192.168.1.132	WebSoc…	158	WebSocket Text [FIN]
24261	157.055030941	irc-ws-edge.c…	192.168.1.132	WebSoc…	418	WebSocket Text [FIN]
24264	157.055101385	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=2639 Ack=3658 Win=40960 Len=0 TSval=356516 TSecr=1372577016
24267	157.062689984	192.168.1.132	irc-ws-edge.…	WebSoc…	113	WebSocket Text [FIN] [MASKED]
24277	157.138157553	irc-ws-edge.c…	192.168.1.132	WebSoc…	212	WebSocket Text [FIN]
24287	157.177476748	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=2686 Ack=3804 Win=43904 Len=0 TSval=356547 TSecr=1372577030
24299	157.278456152	irc-ws-edge.c…	192.168.1.132	WebSoc…	145	WebSocket Text [FIN]
24300	157.278527075	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=2686 Ack=3883 Win=43904 Len=0 TSval=356572 TSecr=1372577076
24301	157.278583666	irc-ws-edge.c…	192.168.1.132	WebSoc…	456	WebSocket Text [FIN]
24302	157.278591660	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=2686 Ack=4273 Win=46848 Len=0 TSval=356572 TSecr=1372577076
24470	158.145894994	irc-ws-edge.c…	192.168.1.132	WebSoc…	369	WebSocket Text [FIN]
24471	158.145920078	192.168.1.132	irc-ws-edge.…	TCP	66	50322→443 [ACK] Seq=2686 Ack=4576 Win=49664 Len=0 TSval=356789 TSecr=1372577268

TCP [SYN]TCP [SYN,ACK] TCP [ACK]	Viene istaurata una connessione TCP fra il client e il server IRC
TLSv1.2 Client Hello TLSv1.2 Server Hello,..,Server Key Exc TLSv1.2 Client Key exchange TLSv1.2 Change Cipher Spec	Viene istaurata una connessione Cifrata fra il client ed il server, seguendo vari passaggi definiti nell RFC 5346.
HTTP GET /HTTP /1.1 HTTP/1.1 101 Switching Protocols	Viene istaurata una connessione WebSocket, seguendo i passaggi definiti nell RFC 6455. Il client invia un messaggio in cui comunica che vuole aprire una connessione WebSocket con protocollo di livello superiore IRC. Il server comunica che la connessione è stata accettata.
WebSocket	Vengono scambiati dei messaggi attraverso la connessione WebSocket al fine di settare i parametri della chat IRC, dopodiche avviene l'effettivo scambio di messaggi della chat.

Messaggi Scambiati attraverso la connessione WebSocket: Client Server

CAP REQ :twitch.tv/tags twitch.tv/commands\r\n

Il client abilitare l'uso dei comandi per la chat di Twitch.

Esempio: CLEARCHAT -> permette Twitch di bannare permanentemente o temporaneamente l'utente.

PASS oauth:kwm7uhhx4n4i5wmrrhwem882ppto32\r\n
NICK nindo7\r\n

Il client manda il Token fornito dalle API e il nickname dell'account Twitch al fine di connettersi al servizio.

```
:tmi.twitch.tv CAP * ACK :twitch.tv/tags twitch.tv/commands\r\n
```

Il server comunica al client che l'uso dei comandi è stato correttamente abilitato.

```
:tmi.twitch.tv 001 nindo7 :Welcome, GLHF!\r\n
:tmi.twitch.tv 002 nindo7 :Your host is tmi.twitch.tv\r\n
:tmi.twitch.tv 003 nindo7 :This server is rather new\r\n
:tmi.twitch.tv 004 nindo7 :-\r\n
:tmi.twitch.tv 375 nindo7 :-\r\n
:tmi.twitch.tv 372 nindo7 :You are in a maze of twisty passages,all alike.\r\n
:tmi.twitch.tv 376 nindo7 :>\r\n
```

Il server indica che l'autenticazione è andata a buon fine.

```
JOIN #gosu\r\n
```

l client richiede di connettersi al canale "gosu".

```
@badges=;color=;display-name=Nindo7;emote-sets=0,19151;user-id=85756562;user-type= :tmi.twitch.tv GLOBALUSERSTATE\r\n
```

Il server invia delle informazioni necessarie all'utente per l'applicazione, come l'id dell'utente e il set di emoticon che è autorizzato ad utilizzare in QUALSIASI canale.

```
:nindo7!nindo7@nindo7.tmi.twitch.tv JOIN #gosu\r\n
```

Il server comunica che la richiesta di accesso al canale "gosu" è stata accettata.

```
@badges=;color=;display-name=Nindo7;emote-sets=0,19151;mod=0;subscriber=0;user-type= :tmi.twitch.tv USERSTATE #gosu\r\n
```

Il server comunica le informazioni relative al proprio account, ovvero se risulta essere iscritto al canale, il nome da visualizzare e il codice del set di Emote utilizzabili in QUESTO canale.

```
@broadcaster-lang=en;emote-only=0;followers-only=-1;r9k=0;room-id=41939266;slow=3;subs-only=0 :tmi.twitch.tv ROOMSTATE #gosu\r\n
```

Il server comunica le informazioni relative alla Chat room, ovvero a chi è consentito scrivere, se è possibile utilizzare Emote e la lingua del broadcaster.

:nindo7.tmi.twitch.tv 353 nindo7 = #gosu :nindo7\r\n

:nindo7.tmi.twitch.tv 366 nindo7 #gosu :End of /NAMES list\r\n

Il server comunica la lista degli utenti connessi alla chat, tuttavia la chat room prevede l'invio solo di messaggi Broadcast nel canale.

@sent-ts=1496049101666 PRIVMSG #gosu :pasta

Questo è il messaggio inviato dal client al server, dove "pasta" è il messaggio inviato e "@sent-ts=%numero" indica il time-stamp del messaggio

Il traffico senza la procedura indicata in **5.3** risulterebbe con il protocollo TLS, infatti gli unici messaggi visualizzati da Wireshark come TLS sono i pacchetti atti ad istaurare una connessione cifrata.

Il protocollo utilizzato sopra TLS è WebSocket, tecnologia internet sviluppata da W3C standardizzata nell RFC 6455. Tale protocollo permette una maggiore facilità a gestire dati in tempo reale, in quanto il server ha il permesso di inviare dati al client senza dover essere sollecitato dal client in alcun modo. In questo modo il client non necessità di aggiornare la pagina HTML ogni qualvolta il server deve inviare nuovi dati.

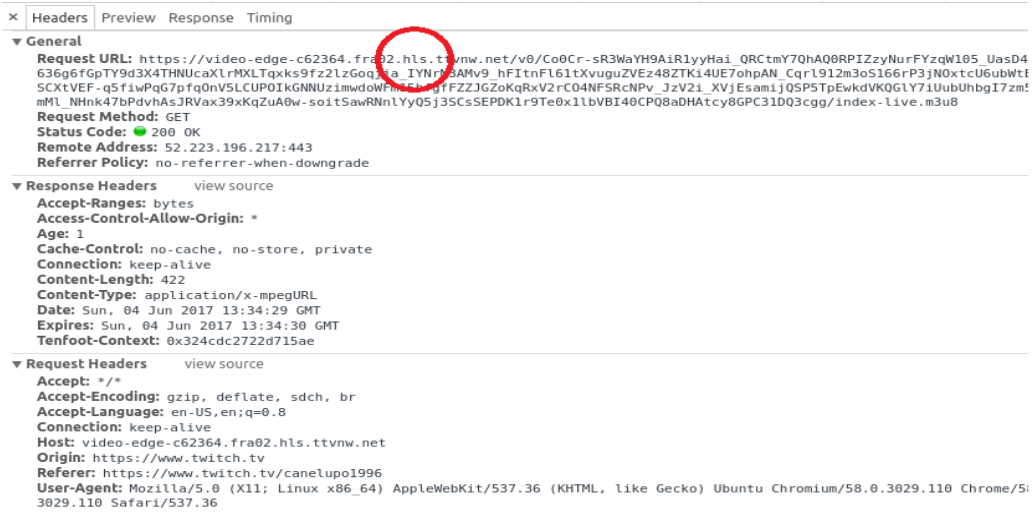
La complessità del protocollo IRC rende impossibile la trattazione completa delle sue funzionalità in breve, per tale motivo è stato riportato solo il setup della chat e l'invio di qualche messaggio, tuttavia è giusto sottolineare che il protocollo IRC presenta una vastità di funzionalità non trattate in questa relazione. Infatti attraverso il traffico analizzato si è potuto notare che nel caso in cui l'utente si connetta ad uno Streaming senza avere un account Twitch, viene fornito un nickname casuale per la chat, che permette di vedere i messaggi inviati ma non di inviarne.

5.6 Lo streaming

Il principale servizio offerto da Twitch è lo streaming video.

Per l'analisi del traffico generato dallo streaming, oltre a Wireshark, si è utilizzato il comando "Ispeziona elemento" dal browser web Chromium. Con l'opzione "Network" è infatti possibile visualizzare tutto il traffico di rete catturato a livello applicativo.

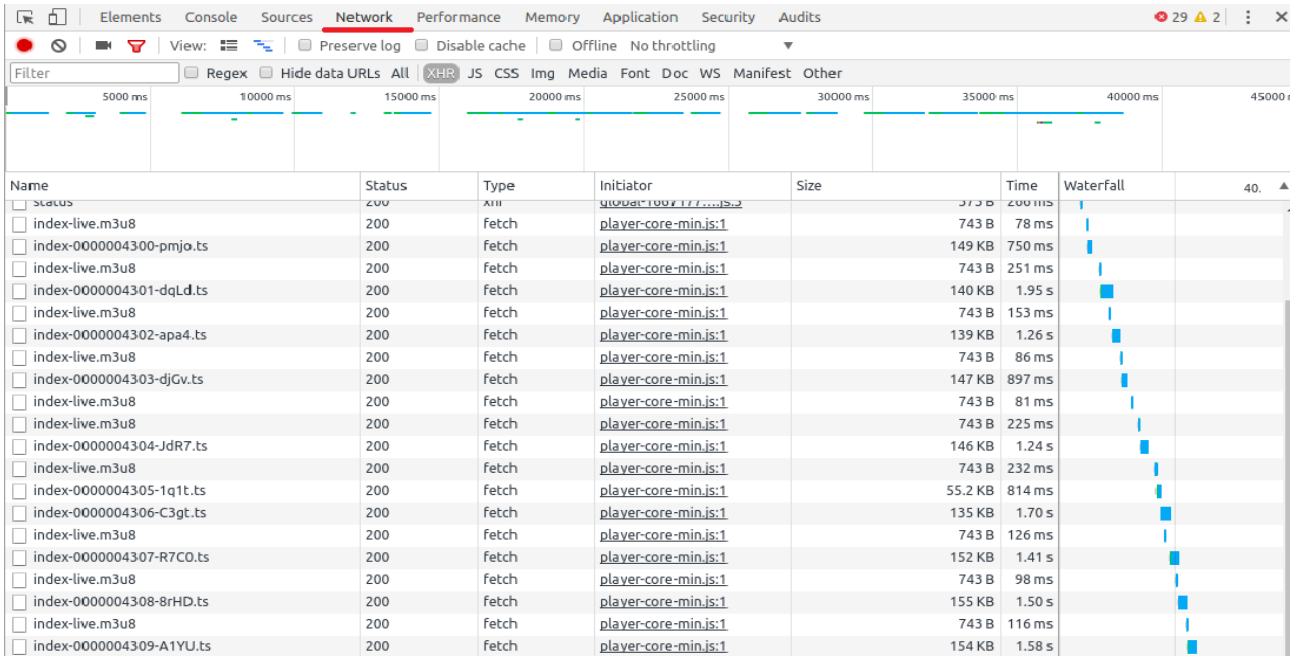
Analizzando i pacchetti si è notato che il protocollo streaming utilizzato da Twitch risulta essere HLS (HTTP Live Streaming).



HLS è un protocollo creato da Apple.inc che, oltre a fornire un servizio di streaming affidabile, permette di adattare il bit rate dei file multimediali inviati dal server con le condizioni della rete del client. Ciò è di grande utilità per l'utente poiché permette di continuare a vedere lo streaming ad una minore risoluzione anche quando le condizioni della rete non sono ottimali.

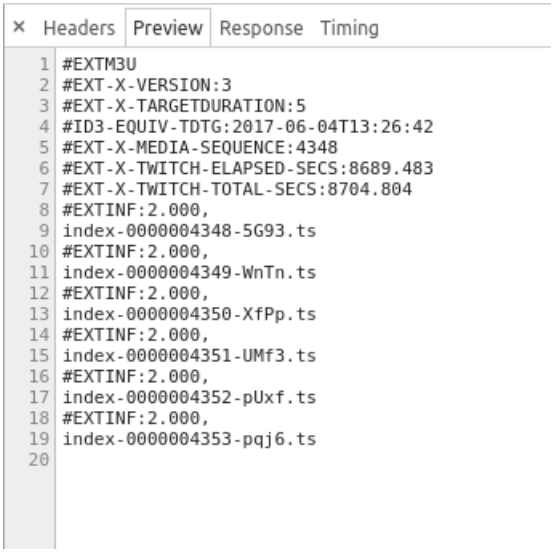
HLS si basa sul protocollo standard HTTP, che, essendo lo stesso utilizzato nella classica navigazione web, risulta essere ben tollerato dai firewall e dai proxy che regolano il normale traffico sulla porta standard HTTP. Al contrario, altri protocolli più specifici come RTP hanno bisogno di porte dedicate e corrono maggiormente il rischio di essere bloccati.

Ecco come si presenta il flusso di dati catturato da Chromium.



Il principio di funzionamento si basa sul dividere in piccoli frammenti il contenuto da scambiare (il filmato in questo caso). Si può notare come ciascun segmento del filmato è un file con estensione ".ts" ovvero un container audio/video nel formato MPEG-2 Transport Stream.
All'avvio dello streaming il client richiederà prima il file di playlist (in formato .m3u8), in cui è riportato l'ordine in cui devono essere scaricati e riprodotti i file del filmato (.ts appunto), e poi i vari frammenti di video.

Ecco come si presenta un singolo file ".m3u8"



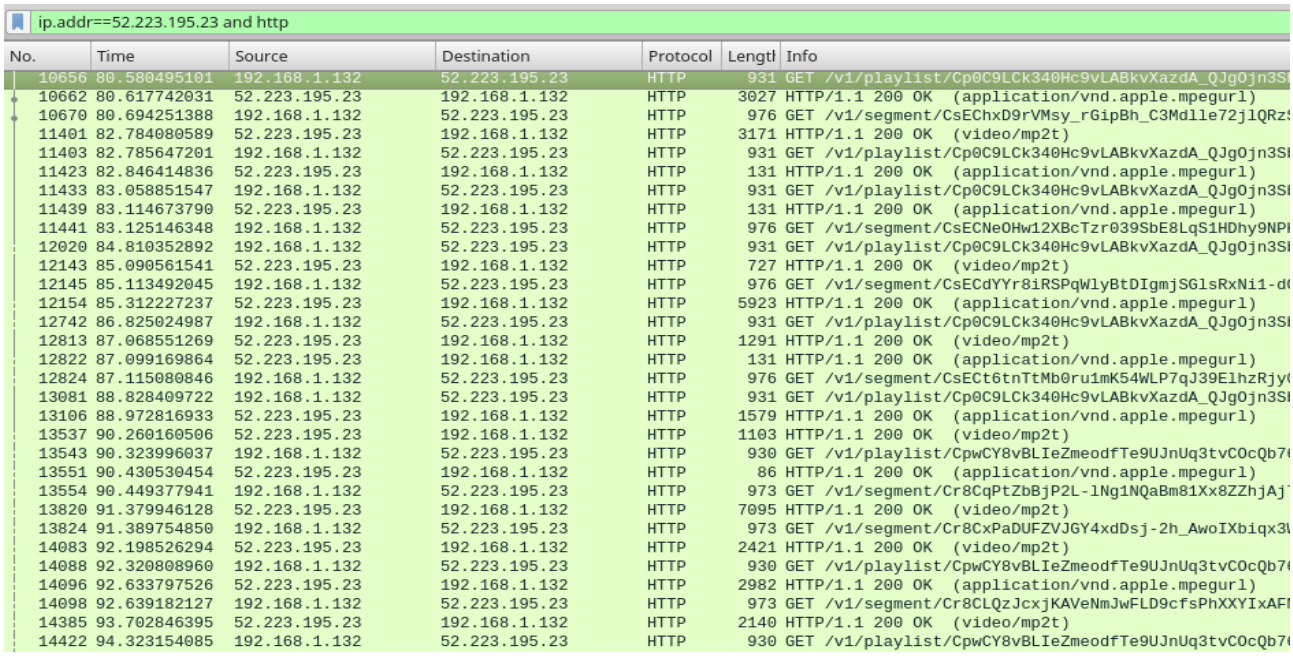
```
Headers  Preview  Response  Timing

#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:5
#ID3-EQUIV-TDTG:2017-06-04T13:26:42
#EXT-X-MEDIA-SEQUENCE:4348
#EXT-X-TWITCH-ELAPSED-SECS:8689.483
#EXT-X-TWITCH-TOTAL-SECS:8704.804
#EXTINF:2.000,
index-0000004348-5G93.ts
#EXTINF:2.000,
index-0000004349-WnTn.ts
#EXTINF:2.000,
index-0000004350-XfPp.ts
#EXTINF:2.000,
index-0000004351-UMf3.ts
#EXTINF:2.000,
index-0000004352-pUxf.ts
#EXTINF:2.000,
index-0000004353-pqj6.ts
```

Dal contenuto del file si possono ricavare importanti informazioni sul tipo di file che successivamente saranno ricevuti dal client.

#EXT-X-TARGETDURATION:5 → specifica che i file multimediali avranno una lunghezza massima di 5 secondi.
#EXT-X-MEDIA-SEQUENCE: X → specifica che X è il numero di sequenza del primo segmento.
#EXTINF:2.000 → specifica l'effettiva durata del segmento.

Si è usato il seguente filtro sui pacchetti catturati da wireshark: $ip.addr == 52.223.195.23$ and http, dove l'indirizzo IP in questione, situato a Siattle (Stati Uniti), è uno dei server che Twitch usa per ricevere il contenuto multimediale sopra citato.



ip.addr==52.223.195.23 and http

No.	Time	Source	Destination	Protocol	Lengtl	Info
10656	80.580495101	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
10662	80.617742031	52.223.195.23	192.168.1.132	HTTP	3027	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
10670	80.694251388	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsEChxD9rVMsy_rGipBh_C3Mdlle72jlQRz
11401	82.784080589	52.223.195.23	192.168.1.132	HTTP	3171	HTTP/1.1 200 OK (video/mp2t)
11403	82.785647201	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
11423	82.846414836	52.223.195.23	192.168.1.132	HTTP	131	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
11433	83.058851547	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
11439	83.114673790	52.223.195.23	192.168.1.132	HTTP	131	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
11441	83.125146348	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECNeOHw12XBcTzr039SbE8LqS1HDhy9NP
12020	84.810352892	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
12143	85.090561541	52.223.195.23	192.168.1.132	HTTP	727	HTTP/1.1 200 OK (video/mp2t)
12145	85.113492045	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECdYYr8iRSPqWlyBtDIgmjSGlsRxNi1-d
12154	85.312227237	52.223.195.23	192.168.1.132	HTTP	5923	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
12742	86.825024987	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
12813	87.068551269	52.223.195.23	192.168.1.132	HTTP	1291	HTTP/1.1 200 OK (video/mp2t)
12822	87.099169864	52.223.195.23	192.168.1.132	HTTP	131	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
12824	87.115080846	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECt6tnTtMb0ru1mK54WLP7qJ39ElhzRjy
13081	88.828409722	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjn3S
13106	88.972816933	52.223.195.23	192.168.1.132	HTTP	1579	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
13537	90.260160506	52.223.195.23	192.168.1.132	HTTP	1103	HTTP/1.1 200 OK (video/mp2t)
13543	90.323996037	192.168.1.132	52.223.195.23	HTTP	930	GET /v1/playlist/CpwCY8vBLIeZmeodfTe9UJnUq3tvCOcQb7
13551	90.430530454	52.223.195.23	192.168.1.132	HTTP	86	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
13554	90.449377941	192.168.1.132	52.223.195.23	HTTP	973	GET /v1/segment/Cr8CqPtZbBjP2L-lNg1NQaBm81Xx8ZZhjAj
13820	91.379946128	52.223.195.23	192.168.1.132	HTTP	7095	HTTP/1.1 200 OK (video/mp2t)
13824	91.389754850	192.168.1.132	52.223.195.23	HTTP	973	GET /v1/segment/Cr8CxPaDUFZVJGY4xdDsj-2h_AwoIXbiqx3
14083	92.198526294	52.223.195.23	192.168.1.132	HTTP	2421	HTTP/1.1 200 OK (video/mp2t)
14088	92.320808960	192.168.1.132	52.223.195.23	HTTP	930	GET /v1/playlist/CpwCY8vBLIeZmeodfTe9UJnUq3tvCOcQb7
14096	92.633797526	52.223.195.23	192.168.1.132	HTTP	2982	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
14098	92.639182127	192.168.1.132	52.223.195.23	HTTP	973	GET /v1/segment/Cr8CLQzJcxjKAVeNmJwFLD9cfsPhXXYIxAF
14385	93.702846395	52.223.195.23	192.168.1.132	HTTP	2140	HTTP/1.1 200 OK (video/mp2t)
14422	94.323154085	192.168.1.132	52.223.195.23	HTTP	930	GET /v1/playlist/CpwCY8vBLIeZmeodfTe9UJnUq3tvCOcQb7

Dalla cattura si può notare un'alternanza delle richieste da parte del client

HTTP GET /v1/playlist/.../*.m3u8

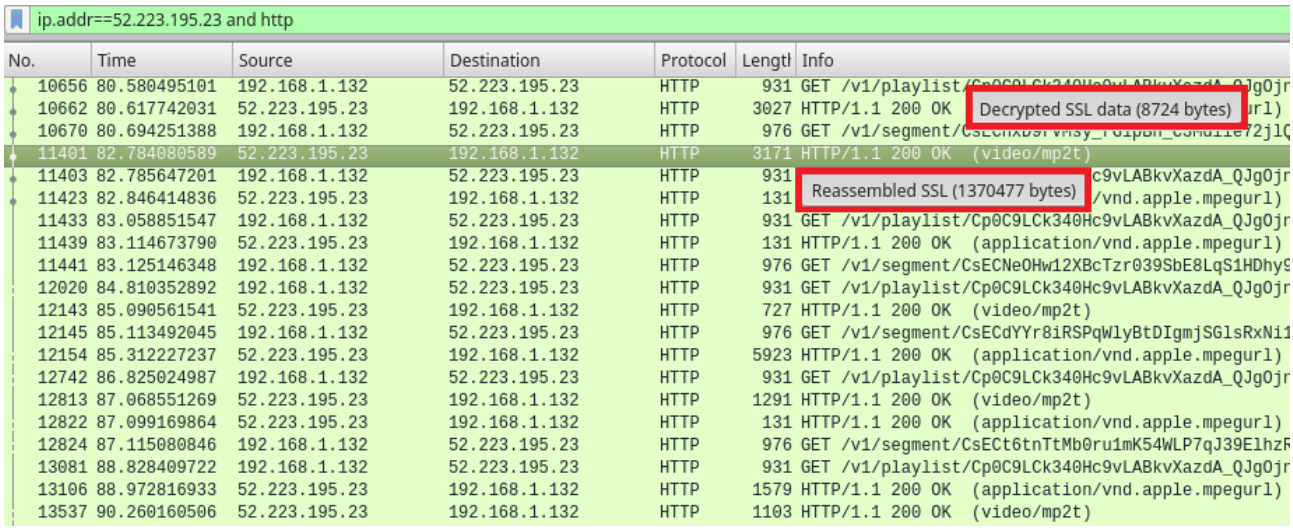
HTTP GET /v1/segment/.../*.ts

e le relative risposte del server

HTTP/1.1 200 OK (application/vnd.apple.mpegurl)

HTTP/1.1 200 OK (video/mp2t)

Controllando la dimensione dei pacchetti catturati si può notare come questi abbiano grandezze notevolmente differenti. Le risposte infatti contengono rispettivamente un file “playlist” e un file multimediale e ciò giustifica la loro dimensione maggiore rispetto alle richieste.



No.	Time	Source	Destination	Protocol	Lengtl	Info
10656	80.580495101	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/
10662	80.617742031	52.223.195.23	192.168.1.132	HTTP	3027	HTTP/1.1 200 OK
10670	80.694251388	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/
11401	82.784080589	52.223.195.23	192.168.1.132	HTTP	3171	HTTP/1.1 200 OK (video/mp2t)
11403	82.785647201	192.168.1.132	52.223.195.23	HTTP	931	
11423	82.846414836	52.223.195.23	192.168.1.132	HTTP	131	
11433	83.058851547	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjr
11439	83.114673790	52.223.195.23	192.168.1.132	HTTP	131	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
11441	83.125146348	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECNeOHw12XBcTzr039SbE8LqS1HDhy9
12020	84.810352892	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjr
12143	85.090561541	52.223.195.23	192.168.1.132	HTTP	727	HTTP/1.1 200 OK (video/mp2t)
12145	85.113492045	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECdYYr8iRSPqWlyBtDIgmjSGlsRxNi1
12154	85.312227237	52.223.195.23	192.168.1.132	HTTP	5923	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
12742	86.825024987	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjr
12813	87.068551269	52.223.195.23	192.168.1.132	HTTP	1291	HTTP/1.1 200 OK (video/mp2t)
12822	87.099169864	52.223.195.23	192.168.1.132	HTTP	131	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
12824	87.115080846	192.168.1.132	52.223.195.23	HTTP	976	GET /v1/segment/CsECt6tnTtMb0ru1mK54WLP7qJ39ElhzR
13081	88.828409722	192.168.1.132	52.223.195.23	HTTP	931	GET /v1/playlist/Cp0C9LCk340Hc9vLABkvXazdA_QJgOjr
13106	88.972816933	52.223.195.23	192.168.1.132	HTTP	1579	HTTP/1.1 200 OK (application/vnd.apple.mpegurl)
13537	90.260160506	52.223.195.23	192.168.1.132	HTTP	1103	HTTP/1.1 200 OK (video/mp2t)

5.7 Altre Funzionalità

La piattaforma permette molte altre funzionalità, che per motivi di sintesi non sono state trattate in questa relazione. Tali funzionalità sono:

- Chat vocale tra utenti registrati
- Chat privata fra utenti registrati
- Notifiche quando si connettono i propri Streamer preferiti o i propri amici
- Possibilità di ospitare uno streaming fungendo uno streaming
- Condurre uno streaming
- Inserire pubblicità nel proprio Streaming
- Inserire Bot nella propria chat

5.8 Conclusioni finali

Twitch è una piattaforma che ha rivoluzionato il mondo dell’intrattenimento legato ai videogiochi.
Fino ad allora il mondo dell’intrattenimento videoludico consisteva in video caricati su YouTube che venivano visualizzati in momenti successivi alla loro creazione. Con l’avvento di Twitch.tv Proplayer e intrattenitori hanno avuto la possibilità di giocare ai propri giochi preferiti in diretta, mostrando le proprie reazioni emotive in tempo reale. Questo cambio di modalità ha permesso un maggiore contatto fra il creatore di contenuti e i propri fan.
Attraverso l’inserimento di pubblicità all’interno del proprio canale è possibile trarre profitto economico dalla trasmissione. Essere uno “streamer” per alcuni è diventato un vero e proprio lavoro.

5.9 Fonti

- https://it.wikipedia.org/wiki/Twitch.tv
- https://dev.twitch.tv/docs
- https://www.ietf.org/rfc/
- https://tools.ietf.org/html/draft-pantos-http-live-streaming-19