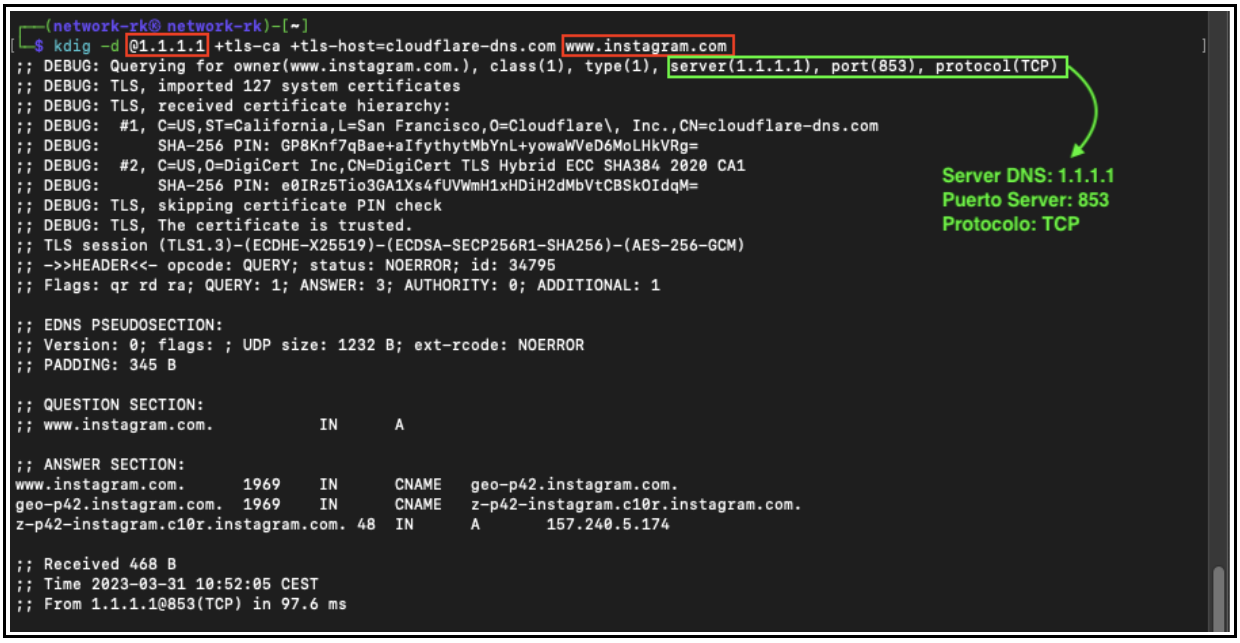
1.- Análisis de la aplicación

Enviaremos una petición a los servidores de resolución de nombres de *Cloudflare* a través de comando KDIG, concretamente pediremos la resoluciones de nombre de www.instagram.com al servidor 1.1.1.1

Cloudflare dispone de 3 servidores primarios diferentes que podemos configurar como cliente y su contraparte el servidor secundario:

- **1.1.1.1/1.0.0.1** → Unicamente actúa como servidor ***DNSoverTLS***
- **1.1.1.2/1.0.0.2** → Realiza un bloqueo de malware y de sitios sospechosos
- **1.1.1.3/1.0.0.3** → Bloqueo parental para contenido adulto entre otros..

Para esta practica usaremos el servidor 1.1.1.1 para realizar la petición de resolución, **KDIG** enviara las peticiones DNS:



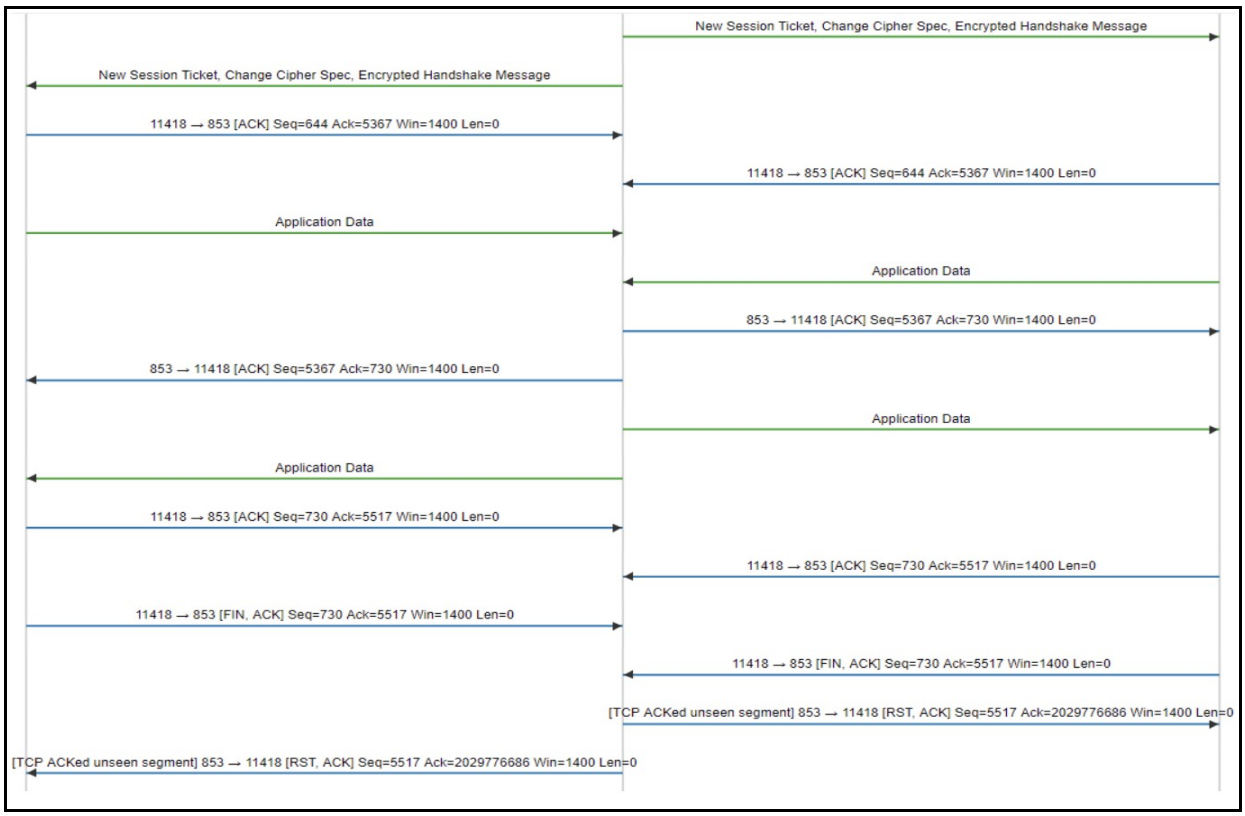

```
┌──(network-rk㉿network-rk)-[~]
└─$ kdig -d @1.1.1.1 +tls-ca +tls-host=cloudflare-dns.com www.instagram.com
;; DEBUG: Querying for owner(www.instagram.com.), class(1), type(1), server(1.1.1.1), port(853), protocol(TCP)
;; DEBUG: TLS, imported 127 system certificates
;; DEBUG: TLS, received certificate hierarchy:
;; DEBUG:  #1, C=US,ST=California,L=San Francisco,O=Cloudflare\, Inc.,CN=cloudflare-dns.com
;; DEBUG:      SHA-256 PIN: GP8Knf7qBae+aIfythytMbYnL+yowaWVeD6MoLHkVRg=
;; DEBUG:  #2, C=US,O=DigiCert Inc,CN=DigiCert TLS Hybrid ECC SHA384 2020 CA1
;; DEBUG:      SHA-256 PIN: e0IRz5Tio3GA1Xs4fUVWmH1xHDiH2dMbVtCBSkOIdqM=
;; DEBUG: TLS, skipping certificate PIN check
;; DEBUG: TLS, The certificate is trusted.
;; TLS session (TLS1.3)-(ECDHE-X25519)-(ECDSA-SECP256R1-SHA256)-(AES-256-GCM)
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 34795
;; Flags: qr rd ra; QUERY: 1; ANSWER: 3; AUTHORITY: 0; ADDITIONAL: 1

;; EDNS PSEUDOSECTION:
;; Version: 0; flags: ; UDP size: 1232 B; ext-rcode: NOERROR
;; PADDING: 345 B

;; QUESTION SECTION:
;; www.instagram.com.           IN      A

;; ANSWER SECTION:
www.instagram.com.      1969    IN      CNAME   geo-p42.instagram.com.
geo-p42.instagram.com.  1969    IN      CNAME   z-p42-instagram.c10r.instagram.com.
z-p42-instagram.c10r.instagram.com. 48  IN      A       157.240.5.174

;; Received 468 B
;; Time 2023-03-31 10:52:05 CEST
;; From 1.1.1.1@853(TCP) in 97.6 ms
```



1.1- Syscalls

Usaremos el comando STRACE para vigilar las llamadas al sistema de red que utiliza para realizar la transacción DoT:

```
┌──(network-rk㉿network-rk)-[~]
└─$ strace -c -e trace=network kdig -d @1.1.1.1 +tls-ca +tls-host=cloudflare-dns.com www.instagram.com
;; DEBUG: Querying for owner(www.instagram.com.), class(1), type(1), server(1.1.1.1), port(853), protocol(TCP)
;; DEBUG: TLS, imported 127 system certificates
;; DEBUG: TLS, received certificate hierarchy:
;; DEBUG:  #1, C=US,ST=California,L=San Francisco,O=Cloudflare\, Inc.,CN=cloudflare-dns.com
;; DEBUG:      SHA-256 PIN: GP8Knf7qBae+aIfythytMbYnL+yowaWVeD6MoLHkVRg=
;; DEBUG:  #2, C=US,O=DigiCert Inc,CN=DigiCert TLS Hybrid ECC SHA384 2020 CA1
;; DEBUG:      SHA-256 PIN: e0IRz5Tio3GA1Xs4fUVWmH1xHDiH2dMbVtCBSkOIdqM=
;; DEBUG: TLS, skipping certificate PIN check
;; DEBUG: TLS, The certificate is trusted.
;; TLS session (TLS1.3)-(ECDHE-X25519)-(ECDSA-SECP256R1-SHA256)-(AES-256-GCM)
;; ->>HEADER<<- opcode: QUERY; status: NOERROR; id: 15277
;; Flags: qr rd ra; QUERY: 1; ANSWER: 3; AUTHORITY: 0; ADDITIONAL: 1

;; EDNS PSEUDOSECTION:
;; Version: 0; flags: ; UDP size: 1232 B; ext-rcode: NOERROR
;; PADDING: 345 B

;; QUESTION SECTION:
;; www.instagram.com.           IN      A

;; ANSWER SECTION:
www.instagram.com.      3554    IN      CNAME   geo-p42.instagram.com.
geo-p42.instagram.com.  3554    IN      CNAME   z-p42-instagram.c10r.instagram.com.
z-p42-instagram.c10r.instagram.com. 14  IN      A       157.240.5.174

;; Received 468 B
;; Time 2023-03-31 11:16:42 CEST
;; From 1.1.1.1@853(TCP) in 90.2 ms
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 42,00    0,000341          68         5           sendmsg
 35,96    0,000292          22        13         3 recvfrom
 11,70    0,000095          95         1         1 connect
  8,37    0,000068          68         1           bind
  1,97    0,000016          16         1           getsockopt
  0,00    0,000000           0         1           socket
------ ----------- ----------- --------- --------- ----------------
100,00    0,000812          36        22         4 total
```

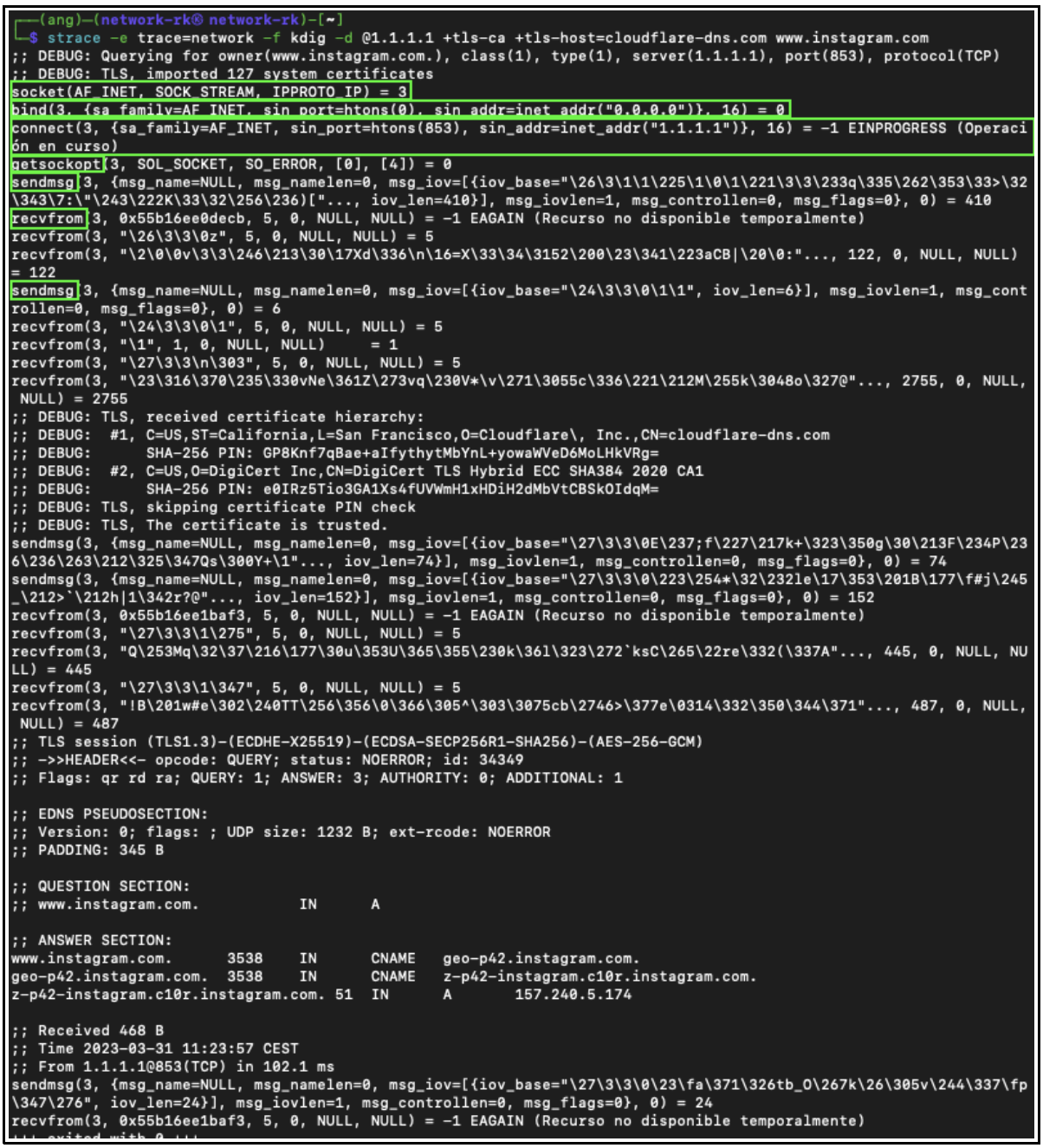
Se inicia una comunicación TLS 1.3 en la que se involucran diferentes algoritmos asimétricos y simétricos:

- **ECDHE:** Es un algoritmo asimétrico, Diffie Hellman de curva elíptica, a a partir del cual se genera un secreto compartido del que deriva una llave con la que cifrar la comunicación.
- **ECDSA:** Genera una firma única para verificar la autenticidad de los mensajes, el cual emplea operaciones sobre puntos de curvas elípticas
- **AES-256-GCM:** un algoritmo simétrico de cifrado por bloques autenticado, tiene 2 componentes AES-CTR para el cifrado y GMAC para el autenticado.

Haciendo un pequeño análisis de las llamadas al sistema que utiliza obtenemos lo siguiente:

- **sendmsg:** Las llamadas al sistema **send**(), **sendto**() y **sendmsg**() se utilizan para transmitir un mensaje a otro socket.
- **Recvfrom:** Las llamadas **recvfrom** () y **recvmsg** () se usan para recibir mensajes de un socket y se pueden usar para recibir datos en un socket, esté o no orientado a la conexión.
- **connect:** La llamada al sistema **connect**() conecta el socket al que hace referencia el descriptor de archivo s*ockfd* a la dirección especificada por *addr*.
- **Bind:** asigna la dirección especificada por *addr* al socket al que hace referencia el descriptor de archivo *sockfd*. *Addrlen* especifica el tamaño, en bytes, de la estructura de dirección a la que apunta *addr*
- **getsockopt:** y setsockopt() manipulan las opciones para el socket al que hace referencia el descriptor de archivo *sockfd* .Las opciones pueden existir en múltiples niveles de protocolo; siempre están presentes en el nivel más alto del zócalo.
- **Socket:** crea un punto final para la comunicación y devuelve un descriptor.

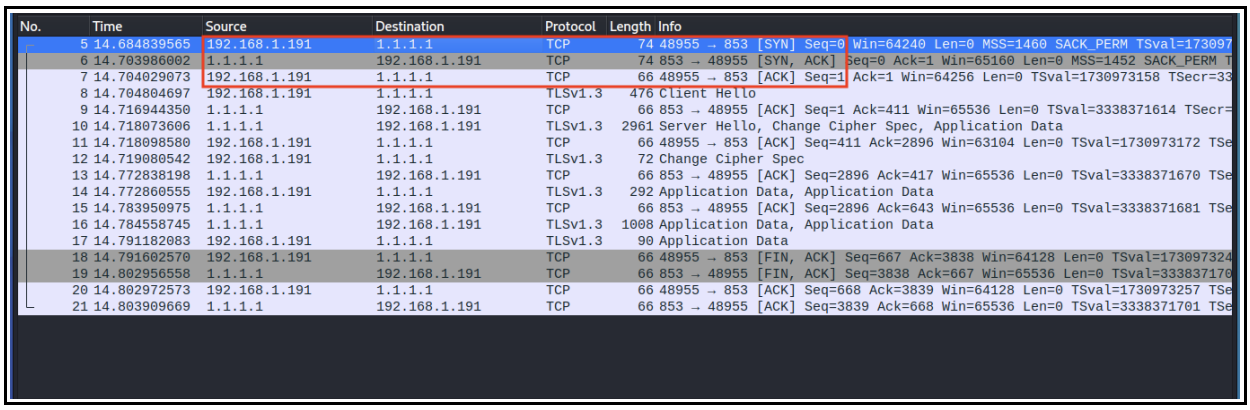
Observemos mas de cerca las llamadas las sistema con los parámetros que maneja durante la transacción:



Realicemos un pequeño análisis de las **syscall** y sus parámetros:

- **socket**(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
  - **Crea un socket con estos parametros:**
    - **AF_INTET → Protocolos de Internet Ipv4**
    - **SOCK_STREAM → Los protocolos de comunicación que lo implementan aseguran que los datos no se pierdan ni se dupliquen**
- **bind**(3, {sa_family=AF_INET, sin_port=htons(0), sin_addr=inet_addr("0.0.0.0")}, 16) = 0
  - **Vincula el socket a la IP local 0.0.0.0**
- **connect**(3, {sa_family=AF_INET, sin_port=htons(853), sin_addr=inet_addr("1.1.1.1")}, 16) = -1 EINPROGRESS
  - **Se incia la conexión a la direccion 1.1.1.1 y puerto 853 del servidor DNS de cloudlfare**
- **getsockopt**(3, SOL_SOCKET, SO_ERROR, [0], [4]) = 0
  - **En caso de éxito al bindear el socket nos devuelve 0**
- **sendmsg**(3, {msg_name=NULL, msg_namelen=0, msg_iov=[.....], msg_iovlen=1, msg_controllen=0, msg_flags=0}, 0) = 410
  - **Cuando el socket esta en estado conectado se procede a enviar los elementos de la matriz a la que apunta *msg_iov***
  -

Realizamos un seguimiento de la comunicación DoT con wireshark:

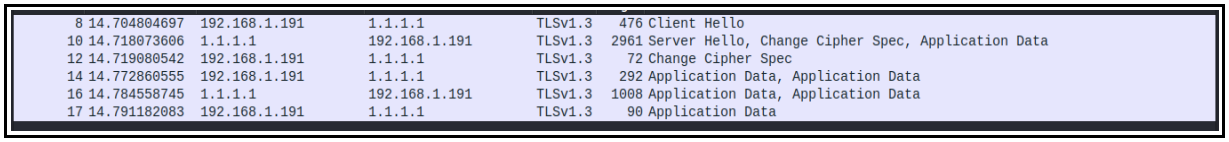


No.	Time	Source	Destination	Protocol	Length	Info
5	14.684839565	192.168.1.191	1.1.1.1	TCP	74	48955 → 853 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM TSval=173097
6	14.703986002	1.1.1.1	192.168.1.191	TCP	74	853 → 48955 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1452 SACK_PERM T
7	14.704029073	192.168.1.191	1.1.1.1	TCP	66	48955 → 853 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=1730973158 TSecr=33
8	14.704804697	192.168.1.191	1.1.1.1	TLSv1.3	476	Client Hello
9	14.716944350	1.1.1.1	192.168.1.191	TCP	66	853 → 48955 [ACK] Seq=1 Ack=411 Win=65536 Len=0 TSval=3338371614 TSecr=
10	14.718073606	1.1.1.1	192.168.1.191	TLSv1.3	2961	Server Hello, Change Cipher Spec, Application Data
11	14.718098580	192.168.1.191	1.1.1.1	TCP	66	48955 → 853 [ACK] Seq=411 Ack=2896 Win=63104 Len=0 TSval=1730973172 TSe
12	14.719080542	192.168.1.191	1.1.1.1	TLSv1.3	72	Change Cipher Spec
13	14.772838198	1.1.1.1	192.168.1.191	TCP	66	853 → 48955 [ACK] Seq=2896 Ack=417 Win=65536 Len=0 TSval=3338371670 TSe
14	14.772860555	192.168.1.191	1.1.1.1	TLSv1.3	292	Application Data, Application Data
15	14.783950975	1.1.1.1	192.168.1.191	TCP	66	853 → 48955 [ACK] Seq=2896 Ack=643 Win=65536 Len=0 TSval=3338371681 TSe
16	14.784558745	1.1.1.1	192.168.1.191	TLSv1.3	1008	Application Data, Application Data
17	14.791182083	192.168.1.191	1.1.1.1	TLSv1.3	90	Application Data
18	14.791602570	192.168.1.191	1.1.1.1	TCP	66	48955 → 853 [FIN, ACK] Seq=667 Ack=3838 Win=64128 Len=0 TSval=173097324
19	14.802956558	1.1.1.1	192.168.1.191	TCP	66	853 → 48955 [FIN, ACK] Seq=3838 Ack=667 Win=65536 Len=0 TSval=333837170
20	14.802972573	192.168.1.191	1.1.1.1	TCP	66	48955 → 853 [ACK] Seq=668 Ack=3839 Win=64128 Len=0 TSval=1730973257 TSe
21	14.803909669	1.1.1.1	192.168.1.191	TCP	66	853 → 48955 [ACK] Seq=3839 Ack=668 Win=65536 Len=0 TSval=3338371701 TSe

Primero se establece una comunicación TCP desde 192.168.1.191 sobre el servidor DNS de cloudflare 1.1.1.1, se realiza mediante la comunicación por 3 pasos:

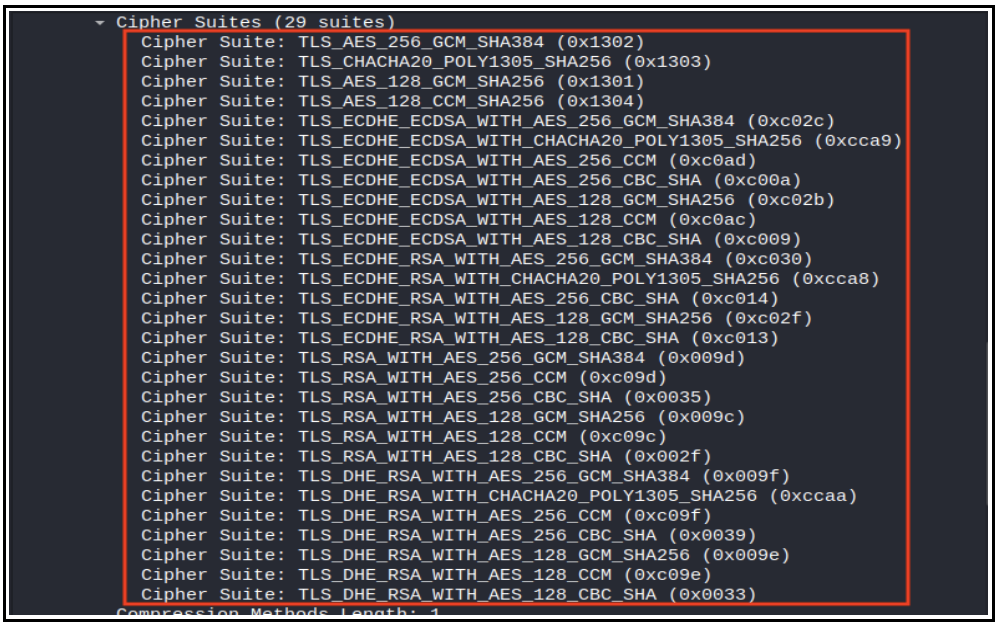
- SYN → seq = x
- SYN-ACK → ack = x + 1 syn = y
- ACK → ack = y + 1 seq = x + 1

A partir de aquí se estable la comunicación TLS 1.3 a través de un intercambio de paquetes entre el cliente y servidor:



8	14.704804697	192.168.1.191	1.1.1.1	TLSv1.3	476	Client Hello
10	14.718073606	1.1.1.1	192.168.1.191	TLSv1.3	2961	Server Hello, Change Cipher Spec, Application Data
12	14.719080542	192.168.1.191	1.1.1.1	TLSv1.3	72	Change Cipher Spec
14	14.772860555	192.168.1.191	1.1.1.1	TLSv1.3	292	Application Data, Application Data
16	14.784558745	1.1.1.1	192.168.1.191	TLSv1.3	1008	Application Data, Application Data
17	14.791182083	192.168.1.191	1.1.1.1	TLSv1.3	90	Application Data

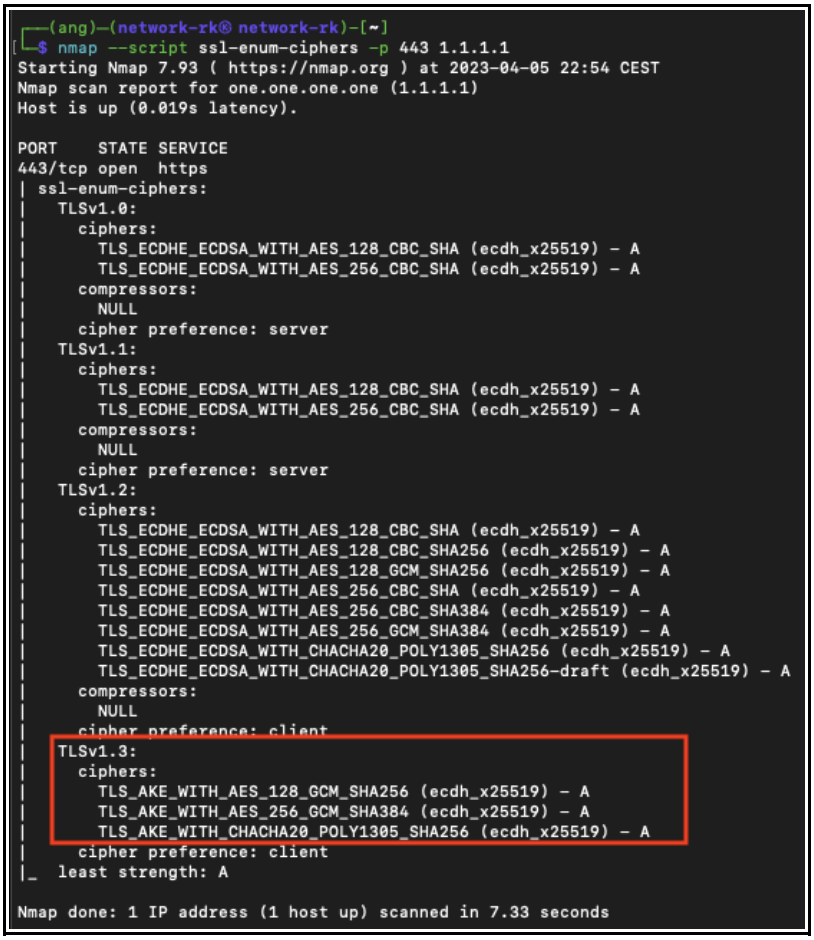
- Primero con el Client Hello se envía al servidor una lista de algoritmos con los que puede trabajar:



```
▾ Cipher Suites (29 suites)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_AES_128_CCM_SHA256 (0x1304)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CCM (0xc0ad)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CCM (0xc0ac)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_256_CCM (0xc09d)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CCM (0xc09c)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)
    Cipher Suite: TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xccaa)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CCM (0xc09f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x009e)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CCM (0xc09e)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
  Compression Methods Length: 1
```

- En este punto el servidor compara con su propia lista y negocian los algoritmos que usaran para el resto de la

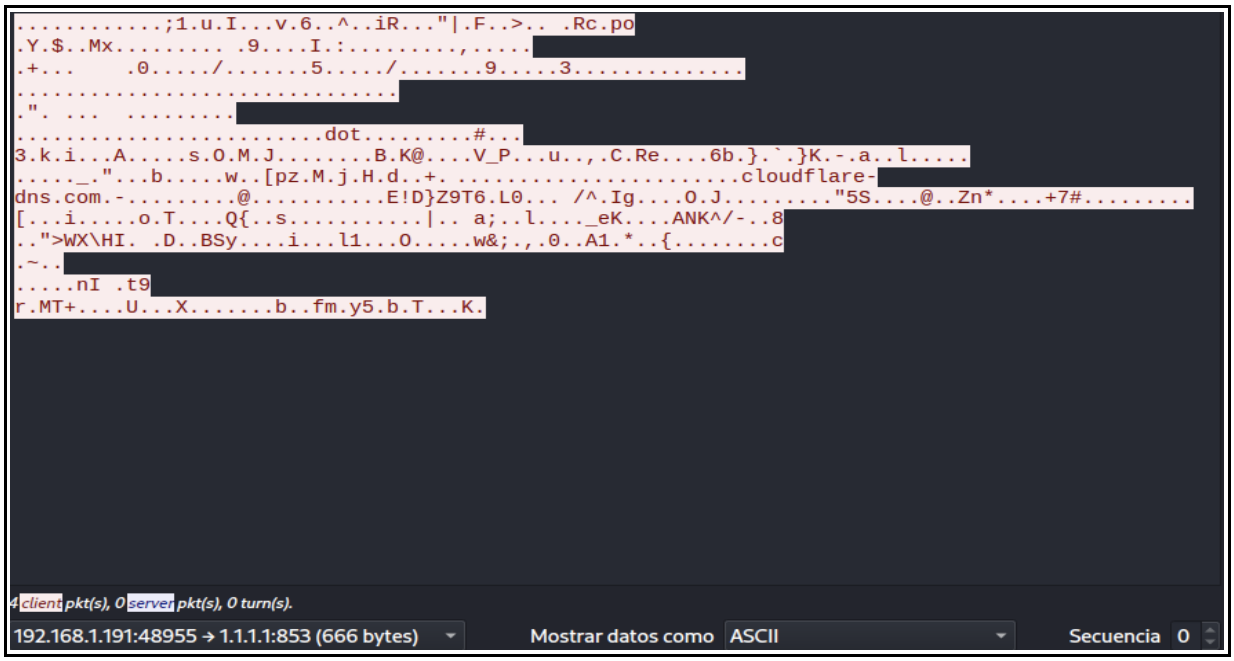
transmisión: (usando **nmap** vemos también la lista de algoritmo que posee el servidor dependiendo de la versión de TLS)



- En nuestro caso se usara **TLS_AKE_WITH_256_GCM_SHA384** que tiene el siguiente desglose:
  - **TLS** → protocolo
  - **AKE** → *authenticated key exchange*
  - **256** → longitud en bits de la clave simétrica
  - **GCM** → modo del cifrado simétrico
  - **SHA384** → Hash

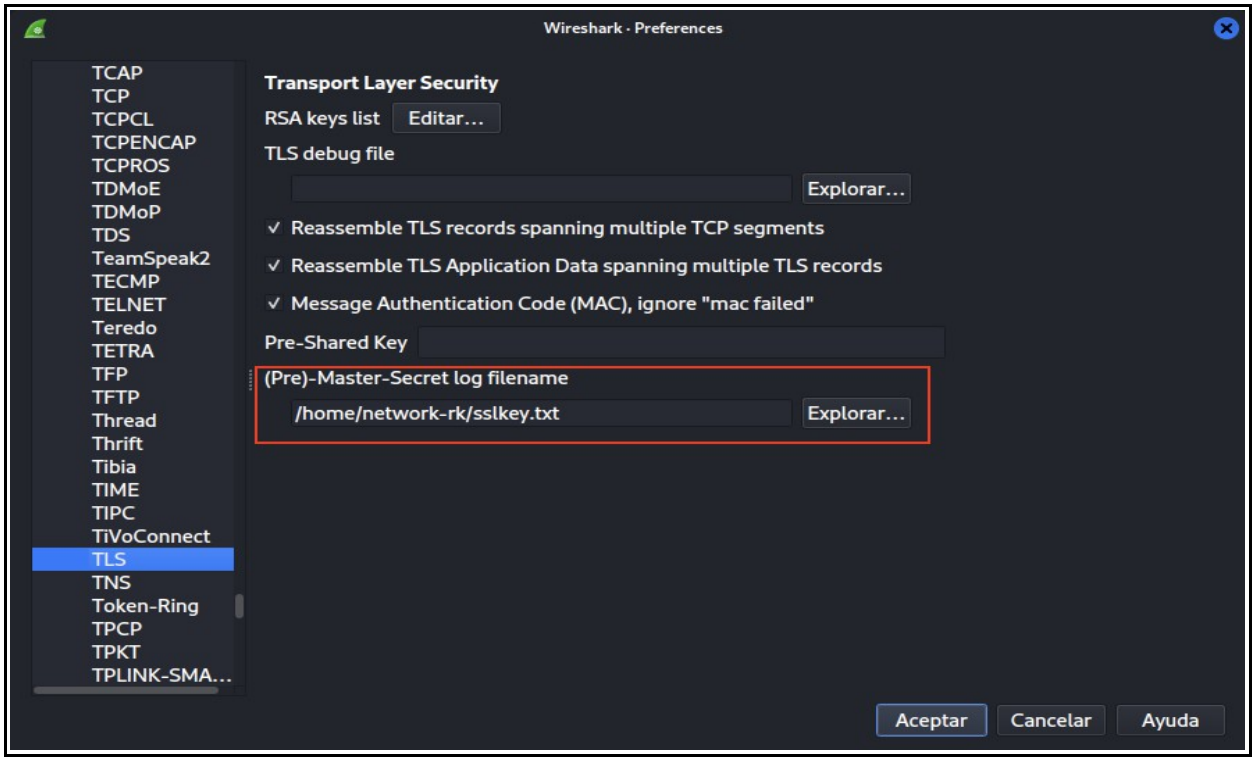
- Una vez acordado y compartido se intercambian los paquetes cifrados entre C/S:

8	14.704804697	192.168.1.191	1.1.1.1	TLSv1.3	476	Client Hello
10	14.718073606	1.1.1.1	192.168.1.191	TLSv1.3	2961	Server Hello, Change Cipher Spec, Application Data
12	14.719080542	192.168.1.191	1.1.1.1	TLSv1.3	72	Change Cipher Spec
14	14.772860555	192.168.1.191	1.1.1.1	TLSv1.3	292	Application Data, Application Data
16	14.784558745	1.1.1.1	192.168.1.191	TLSv1.3	1008	Application Data, Application Data
17	14.791182083	192.168.1.191	1.1.1.1	TLSv1.3	90	Application Data

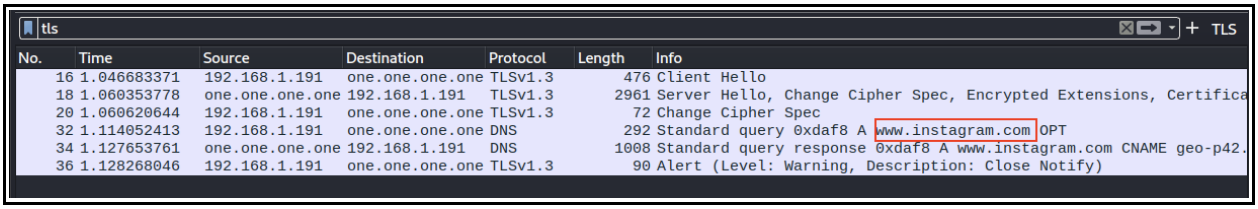


2.- Descifrando con las claves de sesión TLS

Para descifrar el trafico TLS con Wireshark tendremos que apuntar la variable de entorno **SSLKEYLOGFILE** a un archivo de texto en el que se almacenaran los secretos de C/S y en *wireshark* en el apartado PREFERENCIAS>PROTOCOLOS>TLS seleccionar el mismo log para que pueda descifrar el trafico:



Una vez configurado y con Wireshark capturando el trafico enviaremos peticiones al servidor **DNS** de cloudflare con el comando kdig para que nos resuelva el nombre de dominio www.instagram.com, filtraremos en wireshark por paquetes TLS:

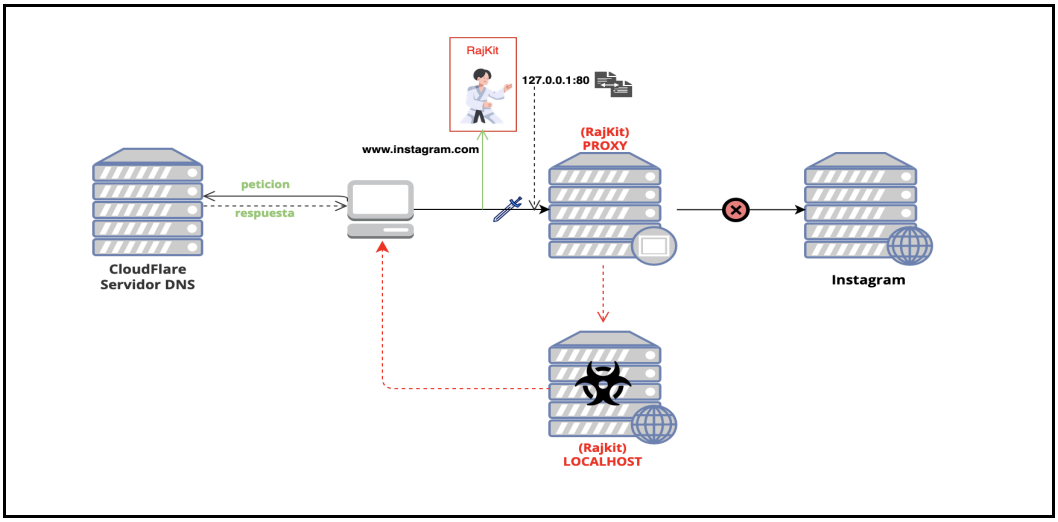


Comprobamos el archivo log en el que se almacenaron los **Master-Secret** para verificar:

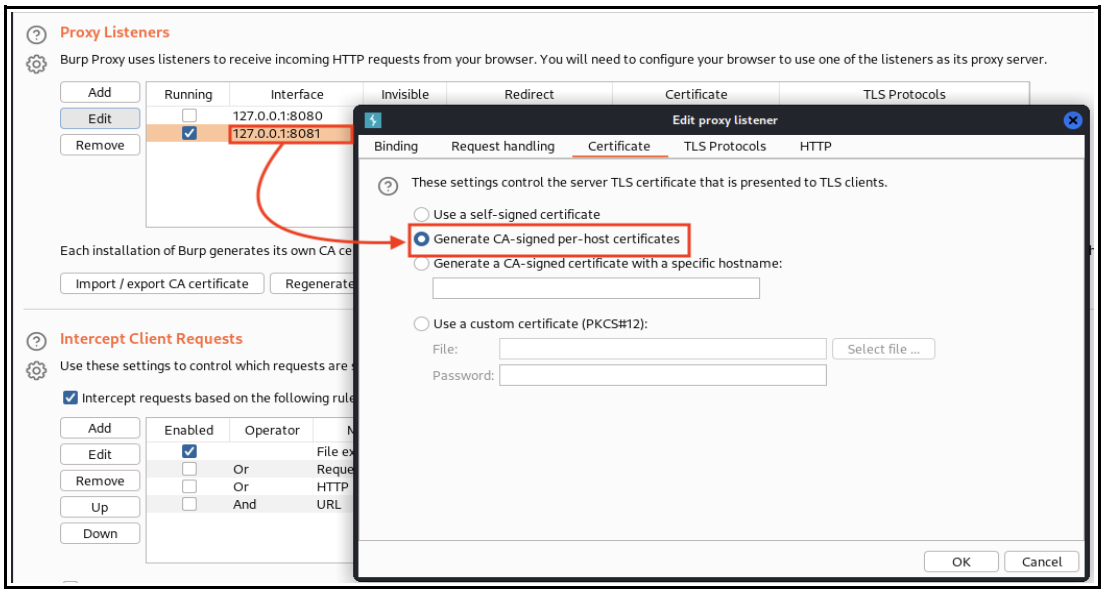
2.1- Mitm a TLS

Para realizar un ataque Mitm a TLS necesitaremos generar nuestros propios certificados **CA** firmados para tratar de que el cliente confíe en que es el servidor quien se los entrega, para ello usaremos la herramienta Burp Suite Community que nos va a facilitar el generar automáticamente esos certificados en función del host al que quiere acceder el cliente, ademas podremos configurar un servidor proxy en la **IP:PUERTO** que queramos para poder interceptar los paquetes y modificarlos como queramos.

Por otra parte montare también un servidor http básico en el puerto **80**, el cual arrancare en una carpeta donde guardo un indice web que solicitara automáticamente al hacer un *"request"*, para esta prueba e realizado una simple copia de un login de Instagram.

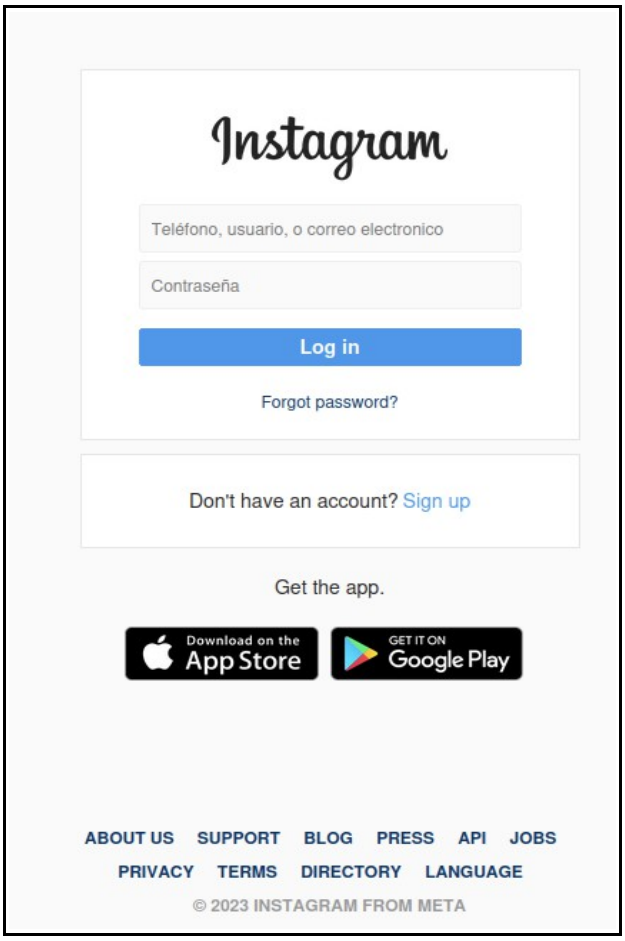


- Configuramos el proxy a la escucha en el puerto **8081** y generamos certificados **CA** firmados per-host:

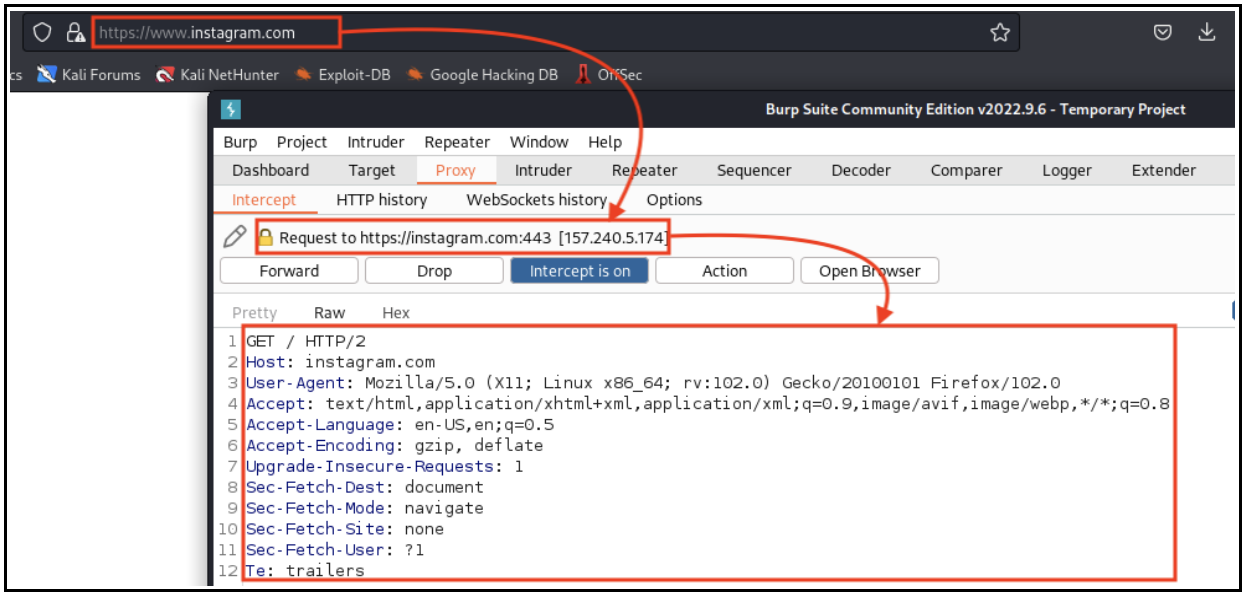


- Ponemos en el puerto **80** un servidor **HTTP** a la escucha para redireccionar las peticiones:

- Una copia falsa del login de Instagram *"index.html"* bajo nuestro control atenderá al cliente desde el servidor:



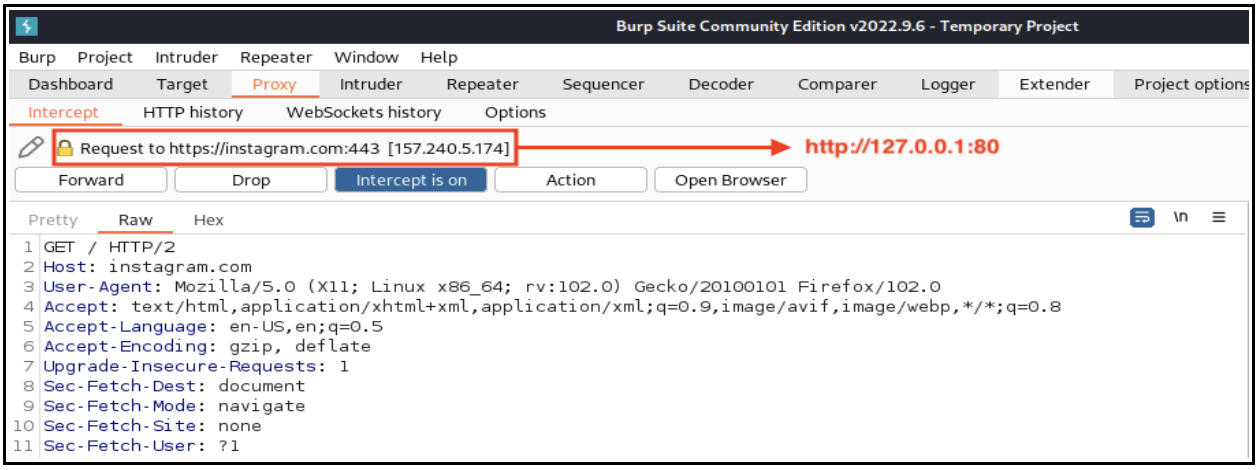
- Ponemos la interceptación activada y realizamos una petición a **https://www.instagram:443** desde la capa de aplicación:



- Una vez interceptada la petición la modificamos completamente para redirigirla a nuestro servidor, nuestro servidor local no admite peticiones **HTTP/2** por lo tanto modificaremos la cabecera a **HTTP/1.1** , el puerto de escucha del **80** al **443** y el host a nuestro servidor:

```
GET / HTTP/2
Host: www.instagram.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image
/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
Cookie: csrftoken=YL7tIdDUfezuVgybmWqXmcwOYiJRuPeY; mid=
ZDV6bQAEAAG9U3nPdYriTGsX6plO; ig_did=
148FBBD3-12F9-46E5-8C8E-4DFDEB9FF55A; datr=
fno1ZKgfJMKZdlb4Q8IRUipE
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```

```
GET / HTTP/1.1
Host: 127.0.0.1
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:102.0)
Gecko/20100101 Firefox/102.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image
/avif,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
Connection: close
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
```



- *Forward request* y veamos como hemos obtenido las credenciales del usuario a través del ataque:
  - **username:** *rajkit@rajkit.com*
  - **password:** *rajkit*