الف: تمام کد های پیاده سازی شده موجود هستند. دو کلاینت یکی بر روی خود node و دیگری بر روی بر روی browser هست.

سرویس تحت وب سوکت ساخته شده. یک پیام از سمت کلایند فرستاده میشود. و همان پیام با مقداری تغییر از سرور به کلاینت برگشت داده میشود.

در nodeClient بعد از یک پیام بستن وب سوکت فرستاده میشود که در زیر توضیحات آن را میدهم.

ب: طبق فایلهایی که سیو شده اند و در همین پوشه پیوست شده اند. این بررسی را انجام میدهم.

فایل nocompression

ابتدا یک overview از تمام قضیه داشته باشیم و سپس هر فریم را به صورت جداگانه بررسی میکنیم.

بعد از هندشیک tcp یک درخواست httpget برای سرور فرستاده میشود که درخواست تغییر پروتوکل به وبسوکت است.

سرور درخواست را با مقدار 101 به کلایند میدهد. که نشان دهنده تایید است.

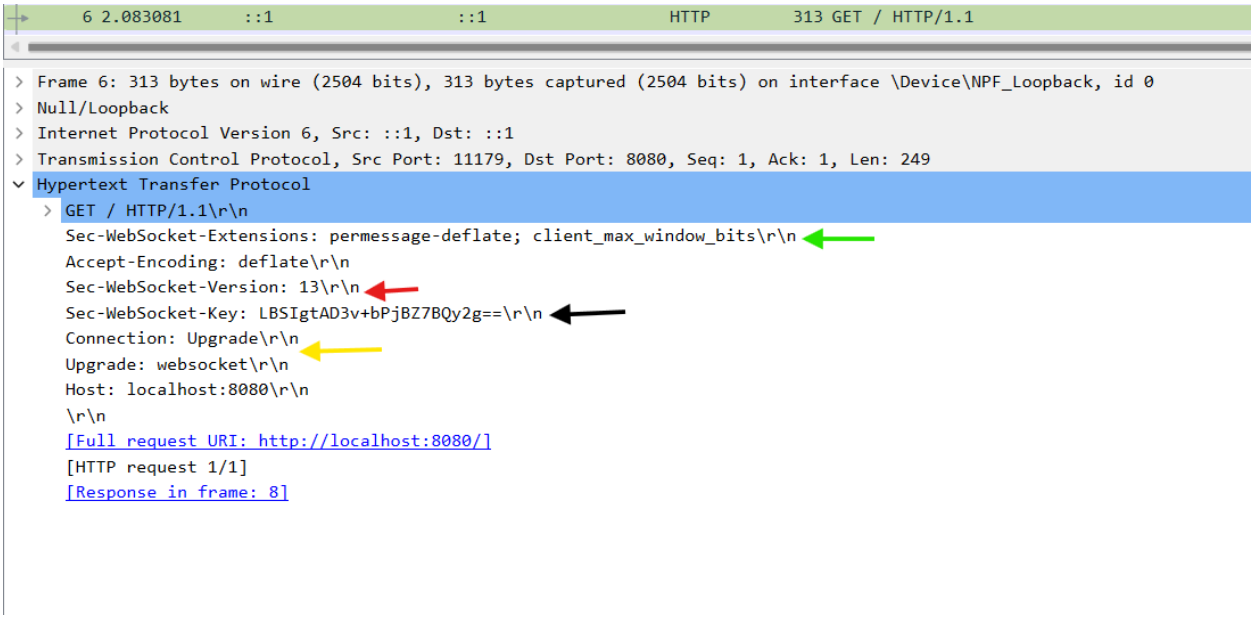
یک متن از کلاینت به سرور میرود که مسک شده است.

یک متن از سرور به کلاینت می رود که مسک شده نیست.

کلاینت درخواست close میدهد

سرور درخواست کلوز را تایید میکند.

1	0.000000	127.0.0.1	127.0.0.1	TCP	45 3806 → 11019 [ACK] Seq=1 Ack=1 Win=8442 Len=1
2	0.000019	127.0.0.1	127.0.0.1	TCP	56 11019 → 3806 [ACK] Seq=1 Ack=2 Win=8434 Len=0 SLE=1 SRE=2
3	2.081064	::1	::1	TCP	76 11179 → 8080 [SYN] Seq=0 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
4	2.081157	::1	::1	TCP	76 8080 → 11179 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65475 WS=256 SACK_PERM
5	2.081224	::1	::1	TCP	64 11179 → 8080 [ACK] Seq=1 Ack=1 Win=2160640 Len=0
6	2.083081	::1	::1	HTTP	313 GET / HTTP/1.1
7	2.083126	::1	::1	TCP	64 8080 → 11179 [ACK] Seq=1 Ack=250 Win=2160384 Len=0
8	2.091569	::1	::1	HTTP	193 HTTP/1.1 101 Switching Protocols
9	2.091622	::1	::1	TCP	64 11179 → 8080 [ACK] Seq=250 Ack=130 Win=2160384 Len=0
10	2.106896	::1	::1	WebSocket	29309 WebSocket Text [FIN] [MASKED]
11	2.106981	::1	::1	TCP	64 8080 → 11179 [ACK] Seq=130 Ack=29495 Win=2130944 Len=0
12	2.113682	::1	::1	WebSocket	29324 WebSocket Text [FIN]
13	2.113752	::1	::1	TCP	64 11179 → 8080 [ACK] Seq=29495 Ack=29390 Win=2131200 Len=0
14	3.090513	::1	::1	WebSocket	70 WebSocket Connection Close [FIN] [MASKED]
15	3.090559	::1	::1	TCP	64 8080 → 11179 [ACK] Seq=29390 Ack=29501 Win=2130944 Len=0
16	3.091659	::1	::1	WebSocket	66 WebSocket Connection Close [FIN]
17	3.091714	::1	::1	TCP	64 11179 → 8080 [ACK] Seq=29501 Ack=29392 Win=2131200 Len=0
18	3.093606	::1	::1	TCP	64 11179 → 8080 [FIN, ACK] Seq=29501 Ack=29392 Win=2131200 Len=0
19	3.093647	::1	::1	TCP	64 8080 → 11179 [ACK] Seq=29392 Ack=29502 Win=2130944 Len=0
20	3.094006	::1	::1	TCP	64 8080 → 11179 [FIN, ACK] Seq=29392 Ack=29502 Win=2130944 Len=0
21	3.094041	::1	::1	TCP	64 11179 → 8080 [ACK] Seq=29502 Ack=29393 Win=2131200 Len=0



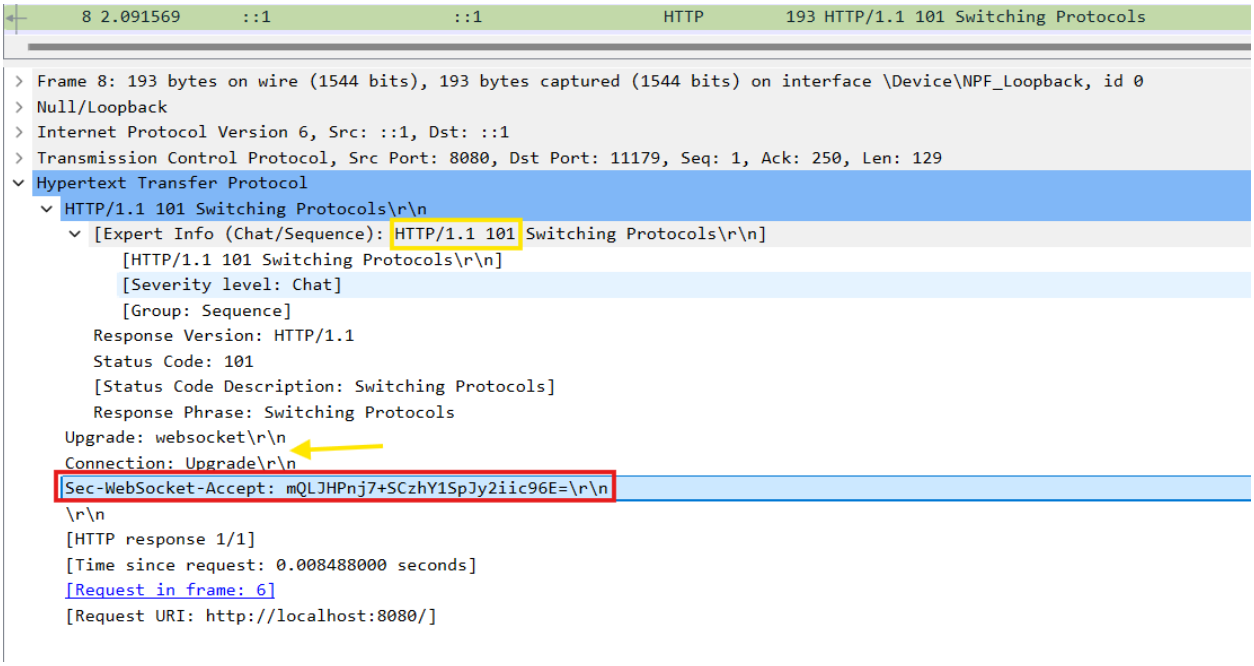
در پیام اول که درخواست http است در هدر یک سری داده ها وجود دارد برای تغییر پروتوگل به وب سوکت. این داده ها همان مذاکره پروتوکل هستند.

فلس سبز: افزونه هایی است که کلاینت ساپورت میکند. یا میتواند ساپورت کند و باید توسط سرور تایید شود.

فلش قرمز: ورژن وب سوکت مورد استفاده توسط کلاینت است.

فلش سیاه: کلیدی است که که کلاینت به سرور میدهد. سپس سرور باید این کلید را گرفته و یک مقدار guid را با این کلید و پروتوکل رمزنگاری sha-1 رمز کند و برگرداند.

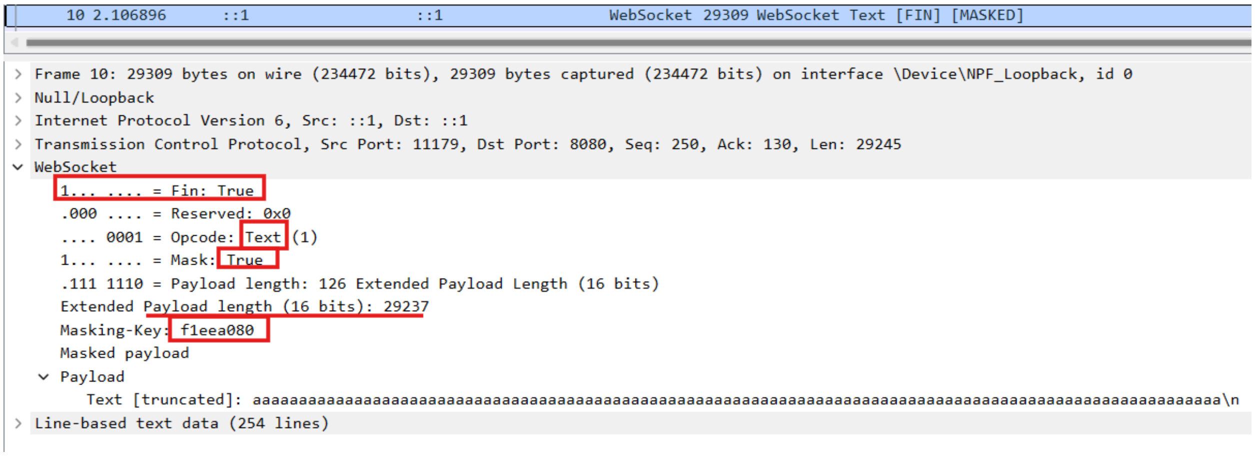
فلش زرد: این دو درخواست برای اپگرید کردن پروتوکل به وب سوکت هستند



پیام سرور به کلایند با کد ۱۰۱ به معنی تایید تغییر کانکشن است.

و بلاک قرمز شده همان ترکیب کلید ارسال شده توسط کلاینت و guid گلوبال برای کل وبسوکت هاست. که مقدار آن 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 است.

بعد از ترکیب این دو استرینگ. توسط sha-1 هش میشوند و این مقدار نشان داده شده همان مقدار هش شده است. که کلید کلایند نیز در آن وجود دارد. کلاینت نیز بعد از انجام همین فرایند. و در صورت برابر بودن مقدار دو استرینگ. تایید نسبی به سرور میدهد. البته این نمیتواند کاملا secure باشد. اما از برخی حملات جلوگیری میکند.



اولین پیام کلاینت به سرور. Fin نشون دهند این هست که این فریم اخرین فریم برای این پیام هست. اگر پیام یخلی طولانی باشد فرگمنتیشن انجام میشود.

در خط بعدی نشان میدهند payload یک پیام متنی است.

سایز payload نشان داده شده.

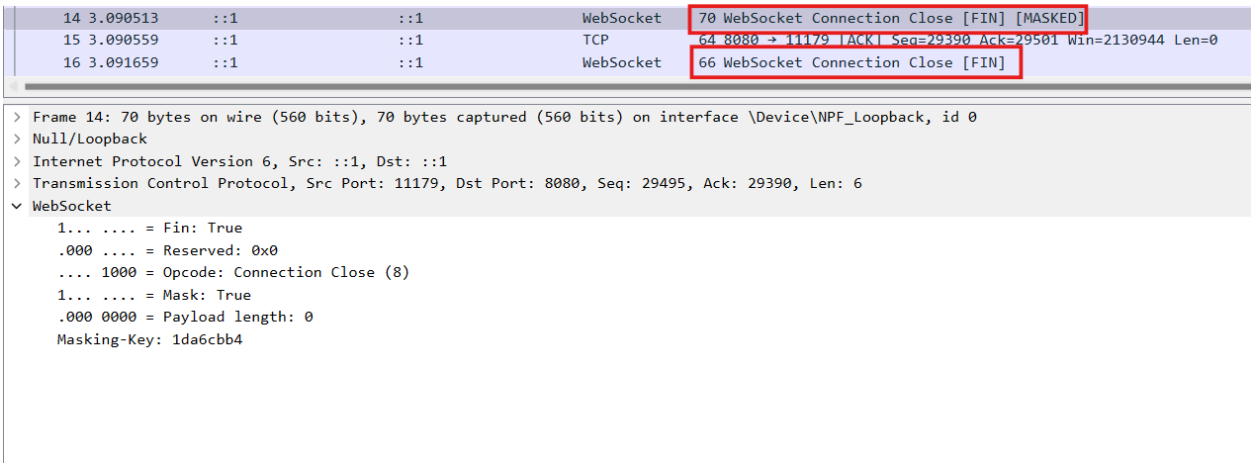
و masking key که فقط در پیام هایی که از سمت کلاینت به سرور میرود وجود دارد. هدف از این کلید ایجاد یک encryption دو طرفه نیست. بلکه به خاطر cash poision هست. یعنی پیام توسط پروکسی ها کش نشود. و هر بار که این پیام ارسال میشود. یک درخواست جداگانه ایجاد کند. Masking key یک مقدار رندوم است. و جلوگیری از دستکاری داده ها توسط واسطه ها.

```
12 2.113682    ::1    ::1    WebSocket 29324 WebSocket Text [FIN]

> Frame 12: 29324 bytes on wire (234592 bits), 29324 bytes captured (234592 bits) on interface \Device\NPF_Loopback, id 0
> Null/Loopback
> Internet Protocol Version 6, Src: ::1, Dst: ::1
> Transmission Control Protocol, Src Port: 8080, Dst Port: 11179, Seq: 130, Ack: 29495, Len: 29260
v WebSocket
    1... .... = Fin: True
    .000 .... = Reserved: 0x0
    .... 0001 = Opcode: Text (1)
    0... .... = Mask: False
    .111 1110 = Payload length: 126 Extended Payload Length (16 bits)
    Extended Payload length (16 bits): 29256
  v Payload
      Text [truncated]: Hello from server, aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
> Line-based text data (254 lines)
```

پیام سرور به کلاینت که مانند قبلیست. فقط مسک نشده. هدف مسکی کردن حفاظت از سرور در مقابل Cache Poisoning است.

RFC 6455 - The WebSocket Protocol



دو پیام نهایی که برای بستن کانکش هست. این دو پیام اولی از کلاینت هست. و بعدی از سرور. که پیام سمت کلاینت مسک شده.

پیام سرور نیاز به مسک ندارد.

این پیام ها payload ندارند و در opcode با پیام های دیگر تفاوت دارند.

پ:

در سوالات بالا این مورد توضیح داده شده. اما توضیح دقیق با توجه به همان لینکی که در بالا آمده:

۱: فرض کنیم حمله کننده یک درخواست به یک هاست میدهد و یک اسکریپت معروف را درخواست میکند. اما ip مقصد را ip خود میگذارد.

۲: در مرحله بعد cache miss داریم توسط پروکسی و پروکسی از ip حمله کننده آن اسکریپت را میگیرد.

۳: حال هر کسی که این درخواست را انجام دهد script حمله کننده را به جای اسکریپت درخواست دریافت میکند.

برای یک همچین مساله ای masking میگذاریم. که هر درخواست. با درخواست قبلی کاملا فرق داشته باشد. Masking باید طوری باشد که هر بار کلیدی که ارسال می شود یونیک باشد و همچنین از روی کلید فعلی نتوان کلید های بعدی را حدس زد.

در این صورت هر بار که کاربر درخواستی برای سرور بفرستد توسط پروکسی سرورها miss میشود.

و این مسک کردن به همین دلیل فقط برای کلاینت به سرور استفاده میشود. و نیازی نیست سرور هم پیام های خود را مسک کند

ج:

در هر دو حالت دو پیام ارسال شده که در کل های کلاینت موجود می باشد. یکی از پیام ها فقط a است. برای تست کردن تاثیر میزان تکرار کاراکتر بر الگوریتم فشرده سازی.

پیام با تکرار a: در حالت بدون فشرده سازی ۲۰۳۰۹ بایت.

پیام با تکرار a: در حالت فشرده شده ۲۲۹ بایت. تقریبا یک هزارم شده.

پیام کوتاه رندوم: در حالت بدون فشرده سازی ۱۷۶ بایت

پیام کوتاه رندوم: در حالت فشرده شده ۱۴۸ بایت. تقریبا ۱۶ درصد کم حجم تر شده.

نکته: روی فریم های کنترلی تاثیری ندارد. و فقط payload را تغییر میدهد. که فریم های کنترلی اصلا payload ندارند

سایز فریم های کنترلی در برنامه من ۶۶ و ۷۰ است.

به طور کلی اگر سایز پیام ها تا حد مناسبی بزرگ باشد. و خود الگوریتم از نظر زمانی order مناسبی داشته باشد. می تواند باعث افزایش بهره وری پروتوکل شود. اما از آنجایی که تقریبا ۶۰ بایت برای لایه های پایینی هست. و مثلا میانگین پیام های ما ۱۰۰ بایت باشد. تقریبا ۳۰ بایت مورد فشرده سازی قرار گرفته میشود که در بهترین حالت به طور میانگین تبدیل به ۲۰ الی ۲۵ بایت خواهد شد. در این حالت خیلی افزوده ای برای ما نخواهد داشت.

اما برای مثال درصد بسیار زیادی از داده های درون وب سوکت به صورت json هستند که علاوه بر حجم زیاد، تکرار در انها زیاد خواهد بود. و الگوریتم مناسب میتواند تاثیر بسیاز زیادی روی حجم داده ها داشته باشد. و تغییر حجم ممکن است باعث شود fragmentation رخ ندهد. که این کار تاثیر زیادی بر روی لتنسی خواهد داشت