# **Deciphering WhatsApp Web**

# **CSEC380 - Computer Security Blogpost - Mehul Sen**

### Introduction

WhatsApp is a free messaging and voice-over-IP service available on multiple platforms. It is owned by Facebook, Inc. and enables users to send text and voice messages, make voice and video calls, as well as share images, documents, user locations, and other content[1].

One of its notable features is known as WhatsApp Web[2], which allows users to access the messaging service on desktop computers as long as their mobile device remains connected to the internet. This capability was introduced in 2017 and it syncs the phone application with the desktop app, using the same WhatsApp account, thus retrieving all the messages and media. Unlike the mobile application, WhatsApp Web works on web browsers, which allows for further investigation into its infrastructure to better understand its behavior.

## Setup

As part of my research, I used an Ubuntu machine to log my SSL handshakes. The machine had a Firefox web browser and Wireshark installed, along with a stable internet connection and a working WhatsApp account.

Since WhatsApp, like most messaging platforms, uses SSL to encrypt their traffic, I first needed to find a way to bypass the SSL encryption.

To confirm this, I took a small Wireshark capture while using WhatsApp web and observed that all the traffic going to the WhatsApp server from the web browser was indeed encrypted using TLS. I was also able to see the TLS Client Hello and the change Cipher protocols in the Wireshark capture.

29 1.458528581	192.168.232.129	shatsapp-cdn-shv-82_ TCP	74 30946 + 643 [575] Seq+8 Min+64248 Lan+8 MSS+1468 SACK_PERH-1 TSval+3622653696 TSecr+8 MS+12
30 1.503668854	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	60 44) + 39948 [SYN, ACK] Seq=0 Ack=1 W1r=64240 Len=0 PSS=1460
	192.168.252.129	whatsapp-cdn-shv-82_ TCP	54 39948 + 443 [ACK] Seq=1 Ack=1 Nin=64248 Len=8
32 1.505397310	192.168.232.129	whatsapp-cdn-shv-82. TLSv1.3	571 Client Hello
33 1.505720235	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	60 443 = 33948 [ACK] Seq+1 Ack+518 Win+64240 Len+0
36 1.551434317	whatsagp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TLSv1.3	3185 Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data
37 1.551464898	192.168.252.129	shatsapp-cdn-shv-82_ TCP	54 30948 + 443 [ACK] Seq+518 Ack+3852 Win+62780 Len+0
42 1.581859381	192.168.232.129	whatsapp-cdn-shv-82, TLSv1.3	118 Change Cipher Spec, Application Data
43 1.582177198	whatsapp-cdn-shv-02-ort2.fbcdn.net	192,168,232,129 TCP	60 443 = 30948 [ACK] Seg=3052 Ack=582 Win=64240 Len=0
44 1.582290007	192.168.232.129	whatsapp-cdn-shv-82, TLSv1.3	224 Application Data
45 1.582425918	192.168.232.129	shatsapp-cdn-shv-82_ TLSv1.5	301 Application Data
46 1.582518878	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	60 443 = 39948 [ACK] Seq=3052 ACK=752 MIn=64240 Len=0
	whatsapp-cdn-shv-82-ort2,fbcdn.net	192,168,232,129 TCP	68 443 + 39948 [ACK] Seg+3951 Ack+999 Min+54349 Len+8
	whatsago-cdn-shv-82-ort2.fbcdn.net	192.168.232.129 TLSv1.3	
	192,168,232,129	what sapp-odn-shv-82_ TCP	54 30948 = 443 [ACK] Seq=909 Ack=3383 Min=62780 Len=0
	192,168,232,129	whatsapp-cdn-shv-82, TLSv1.3	
	whatsapp-cdn-shv-82-ort2.fbcdn.net	192,168,232,129 TCP	68 443 + 30948 [ACK] Seq+3385 Ack+1838 Win+64248 Len+8
	whatsagp-cdn-shv-82-ort2.fbcdn.net	192.168.232.129 TLSv1.3	
	192,168,232,129	whatsapp-cdn-shv-82_ TCP	54 30948 = 443 [ACC] Seg=1030 Ack=3347 Win=62780 Len=0
	whatsapp-cdn-shv-02-ort2.fbcdn.net		4164 Application Data, Application Data, Application Data
	192,168,212,129	whatsapp_cdn_siv-82_ TCP	54 30948 = 645 [ACK] Seq=1830 Ack=7417 Win=61320 Len=0
	192.168.232.129	whatsapp-cdn-shv-82_ TLSv1.3	
	whatsago-cdn-shv-82-ort2,fbcdn.net	192,168,232,129 TCP	68 443 + 39948 (ACK) Seg=7457 Ack=1167 Win=64248 Len=8
	192.168.232.129		
		whatsapp-cdn-shv-02. TLSv1.3	
	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	68 443 = 33948 [ACK] Seq+7457 Ack+1266 Win+64248 Len+8
	192.168.232.129	whotsapp-cdn-shv-02_ TLSv1.3	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		68 443 + 39948 [ACK] Seq=7457 Ack=1353 Win=64248 Len=8
	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TLSv1.3	
	192.168.232.129	shatsapp-cdn-shv-82_ TCP	54 39948 + 443 [ACK] Seq+1353 Ack+7492 Win+62788 Len+8
	whatsapp-cdn-shv-02-ort2.fbcdn.net		2838 Application Data
	192.168.252.129	whatsapp-cdn-shv-82_ TCP	54 30948 + 643 [ACK] Seq=1353 Ack=18276 WEx=62788 Lex=8
	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 + 39948 [PSH, ACK] Seq=10276 Ack=1353 NIn=64240 Len=2784 [TCP segment of a reassembled
	192.168.252.129	whatsapp-cdn-shv-82_ TCP	54 39948 + 443 [ACK] Seq=1353 Ack=13868 Win=62788 Len=8
	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 - 39948 [PSH, ACK] Seq-13060 Ack-1353 Nin-64240 Len-2784 [TCP segment of a reassembled
	192,168,252,129	shatsapp-cdn-shv-82_ TCP	54 30948 = 443 [ACK] Seq=1353 Ack=15044 Min=62780 Len=0
97 1.912979000	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 = 39948 [PSH, ACK] Seq=15844 Ack=1353 H1n=64240 Len=2784 [TCP segment of a reassembled
	192.168.252.129	whatsapp-cdn-shv-82_ TCP	54 39948 + 443 [ACK] Seq=1355 Ack=18628 Min=62780 Len=0
99 1.916003189	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 + 39948 [PSH, ACK] Seq-18628 Ack-1353 Win-64240 Len-2784 [TCP segment of a reassembled
	192,168,232,129	whatsapp-cdn-shv-B2_ TCP	54 39948 - 443 [ACK] Seq=1353 Ack=21412 Win=62780 Len=0
101 1.919314172	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TLSv1.3	2838 Application Data [TCP segment of a reassembled POU]
182 1.919339841	192.168.232.129	whatsapp-cdn-sivv-82_ TCP	54 30948 - 443 [ACK] Seq+1353 Ack+24106 Win+62780 Len+8
103 1.922321417	whatsapp-cdn-shv-82-ort2.fbcdn.net	192.168.232.129 TLSv1.3	2838 Application Data [TCP segment of a reassembled PDU]
184 1.922550274	192.168.252.129	whatsapp-cdn-shv-82_ 1CP	54 39948 + 443 [ACK] Seq=1353 Ack=26980 Win=52780 Len=0
105 1.925599498	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 - 39948 [PSH, ACK] Seq-16900 Ack-1353 Nin-64240 Len-2784 [TCP segment of a reassembled
186 1.925722225	192,168,252,129	shatsapp-cdn-shv-82_ TCP	54 30948 - 443 [ACC] Seq=1353 Ack=29764 Win=62780 Len=0
107 1.920555911	whatsapp-cdn-shv-02-ort2.fbcdn.net	192.168.232.129 TCP	2838 443 - 39948 [PSH, ACK] Seq-29764 Ack-1353 Min-66248 Len-2784 [TCP segment of a reassembled
186 1.928574859	192,168,252,129	shatsapp-cdn-shv-82, TCP	54 30948 + 443 [ACK] Seg-1353 Ack-52548 Win-62780 Len-0
119 1.948178877	whatsago-cdn-shv-82-ort2.fbcdn.net	192.168.232.129 TCP	1446 44) - 39948 [PSH, ACK] Seq-32548 Ack-1353 Win-64240 Len-1392 [TCP segment of a reassembled
120 1.948341854	192,168,252,129	what sapp-cdn-shv-82_ TCP	54 30948 = 443 [ACK] Seg=1353 Ack=33940 Win=62780 Len=0
121 1.948676821	whatsago-cdn-shv-02-ort2.fbcdn.net	192,168,232,129 TCP	2838 443 = 39948 [PSH, ACK] Seq=33940 Ack=1353 NIn=64240 Len=2784 [TCP segment of a reassembled
122 1.948688575		what sapp-odn-siv-82_ TCP	54 30940 + 643 [ACK] Seq+1353 Ack+36724 Win+62700 Len+0
	what same odneshy did north floors net	192,168,232,129 TCP	2638 643 + 39648 (PSN, ACK) Sepul6724 Arks 193 Minu64248 (PRN-7284 (TCP segment of a reassembled

#### Wireshark - Packet 44 - Blog.pcapng

```
Identification: 0x59f6 (23030)
   > Flags: 0x40, Don't fragment
     Fragment Offset: 0
     Time to Live: 64
     Protocol: TCP (6)
     Header Checksum: 0x86e1 [validation disabled]
     [Header checksum status: Unverified]
     Source Address: 192.168.232.129 (192.168.232.129)
     Destination Address: whatsapp-cdn-shv-02-ort2.fbcdn.net (157.240.18.52)
> Transmission Control Protocol, Src Port: 39948, Dst Port: 443, Seq: 582, Ack: 3052, Len: 170

→ Transport Layer Security

▼ TLSv1.3 Record Layer: Application Data Protocol: http-over-tls

        Opaque Type: Application Data (23)
        Version: TLS 1.2 (0x0303)
        Length: 165
        Encrypted Application Data: 5c7565b7419a83d710571255a698ff1aad5c8f22c6818d3a78e1961580b95d01dbb37b1d...
        [Application Data Protocol: http-over-tls]
```

A bit of digging around provided me with an article on the AskF5 forum[2]. To get a better explanation of this method, I would suggest going over the link itself however the basic steps to go decipher the TLS packets are as follows:

• Create a Log file to store all the TLS handshakes.

```
$ touch sslkey.log
```

- Modify the SSLKEYLOGFILE environment variable to store the handshakes within that log file.
  - \$ export SSLKEYLOGFILE="/home/user/sslkey.log"
- Open Firefox within the same shell
  - \$ firefox &"

```
user@ubuntu:~$ touch sslkey.log
user@ubuntu:~$ export SSLKEYLOGFILE="/home/user/sslkey.log"
user@ubuntu:~$ firefox &
[2] 3759
user@ubuntu:~$ [
```

After using WhatsApp web using this web browser and capturing all this traffic through Wireshark we now have everything we need to see what is within the TLS packets.

This is what the generated sslkey.log file looks like once it had logged all the handshakes.

```
1# SSL/TLS secrets log file, generated by NSS
 2 CLIENT_HANDSHAKE_TRAFFIC_SECRET
  6ebfc2fde2468ad973a16fe122f9701dd9dd78c1a6fbbd06aa72548b9437d618
  a7665028243be1c447920ead01b79810d70833aae3a509be589086dea285feb0
 3 SERVER_HANDSHAKE_TRAFFIC_SECRET
  6ebfc2fde2468ad973a16fe122f9701dd9dd78c1a6fbbd06aa72548b9437d618
  db07376dd3c5344eafc8bcbc227e1e72ccb18e174be30cf879901bafb95b8b5d
 4 CLIENT_TRAFFIC_SECRET_0 6ebfc2fde2468ad973a16fe122f9701dd9dd78c1a6fbbd06aa72548b9437d618
  cab81afadaad8e4b5b8e02f3c163cfd34cc10a8540d419dc5b84a5fba6127560
 5 SERVER_TRAFFIC_SECRET_0 6ebfc2fde2468ad973a16fe122f9701dd9dd78c1a6fbbd06aa72548b9437d618
  157a0901d6077893e295f10679ed1319fd16466ee49fca195da375b15d9b7934
 6 EXPORTER_SECRET 6ebfc2fde2468ad973a16fe122f9701dd9dd78c1a6fbbd06aa72548b9437d618
  2aa66430a6ab3d7d8dd313ef805d2f76a2b3bfd764af5f4ecea88d0bb7efef7b
 7 CLIENT_HANDSHAKE_TRAFFIC_SECRET
  c3d9e137d1a7d9788bf94e9476d6efbcc8b81d1c89184bfab88d4f615a8a1492
  b670e59020c8b8b2e25bd6a9d9be97382ba4c9927802a1264902b1f59aae977c
 8 SERVER_HANDSHAKE_TRAFFIC_SECRET
  c3d9e137d1a7d9788bf94e9476d6efbcc8b81d1c89184bfab88d4f615a8a1492
  94f907c4cf117375a70bb5b9204c9563d970a537d6f561be9680b90d86de9abc
 9 CLIENT_TRAFFIC_SECRET_0 c3d9e137d1a7d9788bf94e9476d6efbcc8b81d1c89184bfab88d4f615a8a1492
  a6cfc1b8204380f6e74fa901edecbc278d749a06ca6ee420e630169aa649efa7
10 SERVER_TRAFFIC_SECRET_0 c3d9e137d1a7d9788bf94e9476d6efbcc8b81d1c89184bfab88d4f615a8a1492
  ab26f8fbc4ac24975032a50587f01da433090471cf93d21420220d2af3cbfc4e
11 EXPORTER_SECRET c3d9e137d1a7d9788bf94e9476d6efbcc8b81d1c89184bfab88d4f615a8a1492
  9083568740f633bb607cccb5f1cffedcadddd580c088761ef86b8fcb10bf2792
12 CLIENT_HANDSHAKE_TRAFFIC_SECRET
  2a3a9550f79ebe4f3e5e42506738c74411175a48b2767d0398f58828eede4d7c
  1d0c04fb5f65646b472193874a9283f90c8966cd57d7bb78db204bf34c91b32e
13 SERVER HANDSHAKE TRAFFIC SECRET
  2a3a9550f79ebe4f3e5e42506738c74411175a48b2767d0398f58828eede4d7c
  92b53d3d40e051970da22428c14d34c8b4224642e09c5f57bca2d7804aefd3e0
14 CLIENT_TRAFFIC_SECRET_0 2a3a9550f79ebe4f3e5e42506738c74411175a48b2767d0398f58828eede4d7c
  78911d8aaaae00094dfe0f1442d1382dba3b4195a893603cad16d516fe7eed5a
15 SERVER_TRAFFIC_SECRET_0 2a3a9550f79ebe4f3e5e42506738c74411175a48b2767d0398f58828eede4d7c
  8caff8d2a050dd7574e9bf9976cf78b272839c8c10105305f4999a589460141b
16 EXPORTER SECRET 2a3a9550f79ebe4f3e5e42506738c74411175a48b2767d0398f58828eede4d7c
  h7m0173d72c4mm42a3h000Cafha401727m70a0ff042m7032K0c034h10403K04c
```

Wireshark has a built-in capability to read these logs and decrypt the .pcap traffic. This option is available under Edit > Preferences > Protocols > TLS > (Pre)-Master-Secret Log, selecting the generated sslkey.log, we are now able to look within all the encrypted TLS traffic.

By doing this, we are immediately able to see all the TLS traffic between the WhatsApp server and our web browser.

	192.168.232.129	shatsapp-cdn-shv-82. 7	
30 1.503668854	whatsapp-cdn-shv-02-ort2.fbcdr.net		TCP 60 443 = 39948 [SYN, ACK] Seq=0 Ack=1 Min=64240 Len=0 PSS=1460
	192.168.252.129	shetsepp-cdn-shv-82. 7	
	192.168.232.129	whatsapp-cdn-shv-82. T	
	whatsapp-cdn-shv-82-ort2.fbcdn.net		TCP 68 445 + 39948 [ACK] Seq=1 Ack=518 Win=64248 Len=8
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TLSv1.3 3185 Server Hello, Change Cipher Spec, Encrypted Extensions, Certificate, Certificate Verify, Finished
	192,168,232,129	shatsapp-cdn-shv-82. 7	
	192.168.232.129	whatsapp-cdn-shv-82. T	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 68 443 = 3994E [ACK] Seq=3852 Ack=582 bin=64248 Len=8
	192.168.232.129	whatsapp-cdn-shv-82. H	
	192,168,232,129	what sapp-cdn-shv-82 H	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 68 443 = 39948 [ACK] Seq=3852 ACK=752 NIn=64248 Len=8
47 1.583634563	what sapp-cdn-shu-02-ort2.fbcdn.net		TCP 68 443 = 39948 [ACK] Seq=3852 Ark=999 Min=64240 Len=8
48 1.627648296	whatsapp-cdn-shv-02-ort2.fbcdn.net		HTTP2 306 SETTINGS(0), NINDOW_UPDATE[0]
49 1.627661246		shatsapp-cdn-shv-82 7	
50 1.627979281	292.168.232.129	whatsapp-cdn-shv-82. H	
51 1.628379466	whatsapp-cdn-shu-82-ort2.fbcds.net		TCP 68 443 + 39948 [ACK] Seq+3383 Ark+1838 Win+58248 Len+8
52 1.671622267			HTTP2 98 SETTINGS(8], NINON_UPONTE[15]
	192.168.232.129	what sapp-cdn-shv-82 7	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		MTTP2 4164 DATA[15] (DEXT/MORE)
	192.168.232.129	whatsapp-cdn-shv-82. 7	
	192.168.232.129	whatsapp-cdn-shv-82. H	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 68 443 = 39948 [ACK] Seq=7457 Ack=1167 Win=64248 Len=8
	192.168.232.129	whatsapp-cdn-shv-82. H	
	whatsapp-cdn-shu-02-ort2.fbcdn.net		TCP 68 443 = 39948 [ACK] Seq-7457 Ack-5266 Win-6A248 Len-8
	192.168.232.129	whatsapp-cdn-shv-82. H	
76 1.862384837	whatsapp-cdn-shu-02-ort2.fbcdn.net		TCP 68 443 = 39948 [ACK] Seq-7457 Ack-5353 Win-6A248 Len-8
83 1.993683448	whatsapp-cdn-shv-02-ort2.fbcdn.net		HTTP2 89 HIMDON_UPDATE[17]
84 1.983783178	192.168.232.129	whatsapp-cdn-shv-82. 7	
87 1.904428603	whatsapp-cdn-shv-02-ort2.fbcdn.net		HTTP2 2838 HEADERS[17]: 2800 OK
	192.168.232.129	shatsapp-cdn-shv-82 7	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 2838 440 = 39948 [PSM, ACK] Seq=18276 Ack=1353 NEn=64240 Len=2784 [TCP segment of a reassembled PDV]
	192.168.232.129	whatsapp-cdn-shv-82 7	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 2838 443 + 39948 [PSM, ACK] Seq=13000 Ack=1353 NLm=64240 Len=2784 [TCP segment of a reasonabled PDV]
92 1.910804754	192.168.232.129	whatsapp-cdn-shv-82 7	
97 1.912979888	whatsapp-cdn-shv-02-ort2.fbcdn.net		TCP 2838 843 + 39948 [PSM, ACK] Seq=13684 Ack=1253 Min=64240 Lon=2784 [TCP segment of a reassembled PDU] TCP 54 39948 + 643 [ACK] Seq=1353 Ack=18628 Min=62780 Lon=0
98 1.912994888	192.168.232.129	whatsapp-cdn-shv-82. 7	
	whatsapp-cdn-shv-82-ort2.fbcdn.net		TCP 2838 443 = 39948 [FSR, ACK] Sog-18038 Ack-1933 Min-64240 Len-2704 [TCP segment of a reassembled PDU]
100 1.916020029		whatsapp-cdn-shv-82. T	TCP 54 39948 = 443 [ACK] Seq=3353 Ack=21412 MSn=62788 Len=8 HTTP2 2838 DATA[17] [TCP segment of a reasonabled PDU]
	whatsapp-cdn-shv-82-ort2.fbcdn.net		
102 1.919339041		whatsapp-cdn-shv-82. T	TCP 54 39948 = 443 [ACK] Seq+3353 ACK+24196 NS1+62708 Len+8 TLSv1.5 2838 [TLS segment of a reassembled PDU] [TCP segment of a reassembled PDU]
	whatsapp-cdn-shv-82-ort2.fbcdn.net		
184 1.922338274		whatsapp-cdn-shv-82. 7	
	whatsapp-cdn-shv-82-ort2.fbcdn.net		TCP 2838 443 = 39948 [PSH, ACK] Seq=28988 Ack=1353 Min=64248 Lon=2784 [TCP segment of a reassembled PDU]
	192.168.232.129	whatsapp-cdn-shv-82. T	
	whatsapp-cdn-shv-02-ort2.fbcdn.net		
188 1.928574859	192.160-232.129 what sapp-odn-shv-02-ort2, Flodn, nat	whatsapp-cdn-shv-82. 7 192,168,292,129 7	TCP 54 39948 = 643 [ACK] Seq=3353 ACK=32548 NGN=62700 Len=8 TCP 1646 643 = 30948 [PSH, ACK] Seq=32548 Ack=1353 Min=64260 Len=1302 [TCP segment of a reassumbled PDU]
120 1.948341854		whatsapp-cdn-shv-82, 7	
	192-100-232-129 what sapp-cdn-shu-02-ort2, Fbcdn, nat		TCP 54 39548 = 443 [ACK] 569-1353 ACK=33548 NGN=62708 Len=0 TCP 2838 443 = 30548 [PSR, ACK] 5e9-33548 Ask=1353 Min=64248 Len=2704 [TCP segment of a reassembled PDU]
122 1.948688575			
144 1.740000373	276: 200: 626: 267	whatsapp-cdn-shv-82. T	VL 34 NIME a set Part Schritts without a structure (Single

We can now go further and analyze these packets to get a better clue of how WhatsApp works.

# **Analysis**

After the TLS handshakes and the TCP connections, the first observation is that WhatsApp uses HTTP2 for most of its communication, the first packet sends out a Stream consisting of a couple of different headers.

These are MAGIC, SETTINGS, WINDOW\_UPDATE, and PRIORITY.

```
Y HyperText Transfer Protocol 2
> Stream: Magic
> Stream: SETTINGS, Stream ID: 0, Length 18
> Stream: WINDOW_UPDATE, Stream ID: 0, Length 4
> Stream: PRIORITY, Stream ID: 3, Length 5
> Stream: PRIORITY, Stream ID: 5, Length 5
> Stream: PRIORITY, Stream ID: 7, Length 5
> Stream: PRIORITY, Stream ID: 9, Length 5
> Stream: PRIORITY, Stream ID: 11, Length 5
> Stream: PRIORITY, Stream ID: 13, Length 5
```

MAGIC - This contains the protocol and basic information about how data is being sent.

```
Stream: Magic
Magic: PRI * HTTP/2.0\r\n\r\nSM\r\n\r\n
```

SETTINGS – Looking at the parameters which are Header Table Size, Initial Window Size, and Max Frame Size, This header would be used to set the initial browser window size, optimizing itself for the max size of frames it should receive based on the connection.

**WINDOW\_UPDATE** – This header might provide any changes made to the window itself, updating the server using a parameter called "Window Size Increment".

PRIORITY – This header contains five parameters, Reserved, Stream Identifier, Exclusive, and Stream Dependency and Weight. Looking at multiple PRIORITY headers, we learn that the Stream Identifier is the ID pertaining to the Data that gets sent at a later stage, The Stream Dependency is the number on which that PRIORITY header is dependent on and it is always a value lower than its own Stream ID. The Weight ranges from 0 to 240, and this might signify how important any particular Stream ID is.

Although we can see HEADERS and WINDOW\_UPDATE as well as SETTINGS throughout the packet capture, the MAGIC and PRIORITY headers are not sent again which might mean that they are used for only the initial setup for WhatsApp Web.

After the initial packet, the next packet contains a new header called HEADERS. This contains information like the method used, the path, the authority, scheme, user-agent, data accepted, the language accepted, encoding accepted, upgrade-insecure-requests, and the trailers.

```
✓ Stream: HEADERS, Stream ID: 15, Length 203, GET /
    Length: 203
    Type: HEADERS (1)

✓ Flags: 0x25, Priority, End Headers, End Stream
       00.0 ..0. = Unused: 0x00
       ..1. .... = Priority: True
       .... 0... = Padded: False
       .... .1.. = End Headers: True
       .... 1 = End Stream: True
    0... = Reserved: 0x0
     .000 0000 0000 0000 0000 0000 0000 1111 = Stream Identifier: 15
    [Pad Length: 0]
     0... ---- = Exclusive: False
    .000 0000 0000 0000 0000 0000 0000 0111 = Stream Dependency: 7
    Weight: 41
    [Weight real: 42]
    Header Block Fragment: 82048163418cf058d7f138d281d75ae43d3f877abad07f66a281b0dae053fafc087ed4e1...
    [Header Length: 397]
    [Header Count: 10]
  > Header: :method: GET
  > Header: :path: /
  > Header: :authority: web.whatsapp.com
  > Header: :scheme: https
  > Header: user-agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
  > Header: accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  > Header: accept-language: en-US,en;q=0.5
  > Header: accept-encoding: gzip, deflate, br
  > Header: upgrade-insecure-requests: 1
  > Header: te: trailers
```

The next couple of HEADERS send GET requests to "/bootstrap\_qr-e892ca30934b9f1b9db6.css" for CSS, and "/vendor1~bootstrap\_qr.69e960b196d7d6aa3d46.js",

"/bootstrap\_qr.df9188bb4cd23ee53e79.js" for JavaScript.

As a reply, the server sends back a packet containing a DATA header. The first couple of DATA headers contain HTML, CSS and JavaScript requested by the Web Browser.

The DATA header is much simpler than its counterparts, it has a Stream Identifier parameter and a Data parameter that contain that data. It also has flags like Unused, Padded, and End-Stream. Padded is set to 1 when data is being sent, after the data has been completed, it gets set to 0 and the End Stream Flag is set to 1.

```
    Stream: DATA, Stream ID: 21, Length 4887
    Length: 4887
    Type: DATA (0)

    Flags: 0x01, End Stream
    0...... = Reserved: 0x0
    .000 0000 0000 0000 0000 0001 0101 = Stream Identifier: 21
    [Pad Length: 0]
    [Reassembled body in frame: 392]
    Data: 84b019faa977967a07e0e735f53edb9bf6f6fa01d19deca7c582ecde7d22d49f849f3349...
```

Observing all this data, we can see that it is all encrypted as well, this would be because of the implemented end-to-end encryption by WhatsApp, where each chat is individually encrypted among the recipients. WhatsApp provides its white pages which have a good explanation of how its Encryption and various features work[3].

Although the chat itself is encrypted, we are still able to catch some data types that come from the server. One example of this is when a picture is shared, we can get the PNG signature, the image header, palette, image data chunk, and the image trailer.

The emojis shared can also be seen when the web browser uses a HEADERS header to send a GET request for those resources, which goes as follows GET /img/[emoji-name].webp

```
✓ Stream: HEADERS, Stream ID: 65, Length 37, GET /img/emoji-3-40_0b9fe45.webp

     Length: 37
    Type: HEADERS (1)
  > Flags: 0x25, Priority, End Headers, End Stream
     0... = Reserved: 0x0
     .000 0000 0000 0000 0000 0000 0100 0001 = Stream Identifier: 65
     [Pad Length: 0]
     0... = Exclusive: False
     .000 0000 0000 0000 0000 0000 0000 0111 = Stream Dependency: 7
     Weight: 21
     [Weight real: 22]
     Header Block Fragment: 82059560d4ccc1693f432ccacd022046fca569b5fc163affd587d4ccd2d1cdcf
     [Header Length: 375]
     [Header Count: 10]
  > Header: :method: GET
  > Header: :path: /img/emoji-3-40_0b9fe45.webp
  > Header: :authority: web.whatsapp.com
  > Header: :scheme: https
   > Header: user-agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:88.0) Gecko/20100101 Firefox/88.0
  > Header: accept: */*
  > Header: accept-language: en-US,en;q=0.5
  > Header: accept-encoding: gzip, deflate, br
  > Header: referer: https://web.whatsapp.com/serviceworker.js
  > Header: te: trailers
```

Another interesting header that occasionally comes up is the RST\_STREAM header which contains a Stream Identifier parameter and an Error parameter.

```
Stream: RST_STREAM, Stream ID: 59, Length 4
    Length: 4
    Type: RST_STREAM (3)
> Flags: 0x00
    0..... = Reserved: 0x0
    .000 0000 0000 0000 0000 0001 1011 = Stream Identifier: 59
    Error: CANCEL (8)
```

This is being used by the web browser to tell the server to stop processing certain streams or convey any other such errors.

The last header that gets sent to close the connections is the GOAWAY header, this is first sent by the web browser to the server and then the server echoes it back to the web browser, this contains a Stream Identifier, a reserved parameter, a Promised-Stream-ID parameter, and an Error parameter. A Successful closing takes place with the Error parameter set to 0 which is the NO\_ERROR condition.

#### **Future Research Possibilities**

Further research can be conducted on the protocol, traffic can be captured while performing various actions such as sending different media through chat, adding new users, deleting messages, creating group chats, etc. This will provide us with an even clearer understanding of the headers and protocols used by WhatsApp. We can even use the technical white pages [4] provided

by WhatsApp to further decrypt the end-to-end encrypted data, finding out exactly how each text message is sent.

#### Conclusion

WhatsApp Messenger is a very sophisticated environment with its own set of protocols and headers, WhatsApp Web allows us to monitor this traffic through HTTP and get a better understanding of how it behaves. We can bypass the SSL encryption and get a clearer view of the interactions taking place between WhatsApp servers and the Web browser.

#### **Disclaimer**

Since WhatsApp web is a private organization, it does not disclose its exact source code. We cannot know exactly how it behaves in the backend, the most we can do is observe the traffic and make educated guesses on how it might work. I analyzed the web traffic captured and tried making the best estimates as to how I believe these headers might be used. This information may not be completely accurate.

#### References

- [1] https://en.wikipedia.org/wiki/WhatsApp
- [2] <a href="https://web.whatsapp.com/">https://web.whatsapp.com/</a>
- [3] https://support.f5.com/csp/article/K50557518
- [4] https://scontent.whatsapp.net/v/t39.8562-34/122249142 469857720642275 21525275869075 31259 n.pdf/WA Security WhitePaper.pdf?ccb=1-3& nc sid=2fbf2a& nc ohc=Uz1ThVZZLqUAX 10j -p& nc ht=scontent.whatsapp.net&oh=29e00608fc684abe813d602a491f5757&oe=60B54C19