10-10-2015

WhatsApp Network Forensics: Decrypting and Understanding the WhatsApp Call Signaling Messages

Filip Karpisek
Brno University of Technology

Ibrahim Baggili
University of New Haven, ibaggili@newhaven.edu

Frank Breitinger
University of New Haven, fbreitinger@newhaven.edu

Publisher Citation

F. Karpisek, I. Baggili, F. Breitinger, WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages, Digital Investigation, Available online 10 October 2015, ISSN 1742-2876, http://dx.doi.org/10.1016/j.diin.2015.09.002.

Comments

Dr. Ibrahim Baggili was appointed to the University of New Haven's Elder Family Endowed Chair in 2015.

This is the peer reviewed version of the following article:F. Karpisek, I. Baggili, F. Breitinger, WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages, Digital Investigation, Dec. 2015, Vol. 15, pp. 110-118, which is published in final form at http://dx.doi.org/10.1016/j.diin.2015.09.002. This article may be used for non-commercial purposes in accordance with the CC/BY/NC/ND license.

WhatsApp network forensics: Decrypting and understanding the WhatsApp call signaling messages

F. Karpisek[a], I. Baggili[b], F. Breitinger[b]

[a]*Faculty of Information Technology, Brno University of Technology*
[b]*Cyber Forensics Research & Education Group, Tagliatela College of Engineering, ECECS,*
University of New Haven, 300 Boston Post Rd., West Haven, CT, 06516

Abstract

WhatsApp is a widely adopted mobile messaging application with over 800 million users. Recently, a calling feature was added to the application and no comprehensive digital forensic analysis has been performed with regards to this feature at the time of writing this paper. In this work, we describe how we were able to decrypt the network traffic and obtain forensic artifacts that relate to this new calling feature which included the: a) WhatsApp phone numbers, b) WhatsApp server IPs, c) WhatsApp audio codec (Opus), d) WhatsApp call duration, and e) WhatsApp's call termination. We explain the methods and tools used to decrypt the traffic as well as thoroughly elaborate on our findings with respect to the WhatsApp signaling messages. Furthermore, we also provide the community with a tool that helps in the visualization of the WhatsApp protocol messages.

Keywords: WhatsApp, reverse engineering, proprietary protocol, signaling protocols, network forensics, decryption, mobile forensics, digital forensics, cyber security, audio encoding

1. Introduction

WhatsApp is one of the most widely used personal-messaging mobile applications for free texting and content sharing (namely audio, video, images, location and contacts), boasting over 800 million users worldwide and was bought by facebook in 2014 for $19 Billion[1]. The calling feature was added recently in version 2.11.552, which was released 2015-03-05 (Arce, 2015).

From its wide adoption, it is obvious how WhastApp communication exchanges may be used during an investigation, making the artifacts it produces of compelling forensic relevance. Therefore, we see a strong necessity for both researchers and practitioners to gain a comprehensive understanding of the networking protocol used in WhatsApp, as well as the type of forensically relevant data it contains. Most importantly, due to the newly introduced calling feature, it becomes essential to understand the signaling messages used in the establishment of calls between the WhatsApp clients and servers. The methods and tools used in this research could be relvant to investigations where proving that a call was made at a certain date and time is necessary.

Our contribution outlines the WhatsApp messaging protocol from a networking perspective and provides a solution to explore and study WhatsApp network communications. In terms of novelty, to our knowledge, this is the first paper that discusses the WhatsApp signaling messages used when establishing voice calls. The work has impact on practitioners in the field that have obtained network traffic for a potential suspect, as well as providing scientists literature for better understanding the network protocol itself.

The rest of the paper is organized as follows. In Section 2 we review existing work, while Section 3 describes the WhatsApp protocol. Then, in Section 4 we describe the tool we created for visualizing exchanged WhatsApp messages. In Section 6, we describe the process of obtaining decrypted connections between the WhatsApp client and the WhatsApp server. Then in Section 7 we examine the message contents and discuss the meaning of the signaling messages during a WhatsApp call. Finally, in Section 8, we offer concluding remarks and outline some future research.

2. Related work

There has been research conducted on the forensics of WhatsApp but the majority of that work focused on the data that WhatsApp stores on the mobile device when compared to our work which focuses on the network forensics of WhatsApp.

Email addresses: xkarpi03@stud.fit.vutbr.cz (F. Karpisek), IBaggili@newhaven.edu (I. Baggili), FBreitinger@newhaven.edu (F. Breitinger)

URL: http://www.unhcfreg.com/ (I. Baggili), http://www.FBreitinger.de/ (F. Breitinger)

[1] http://money.cnn.com/2014/02/19/technology/social/facebook-whatsapp/, last accessed 2015-07-03.

2.1. Network protocol forensics

At the time of writing this paper, the work on network protocol forensics of WhatsApp was sparse. The only work that provided any detail on WhatsApp's networking protocol was the Hancke (2015) report. Hancke (2015)'s work focused more on Realtime Transport Protocol (RTP) media streams (Schulzrinne et al., 2003). The report fails to uncover the call signaling messages used by WhatsApp, which is elaborated on by our work.

2.2. Mobile device forensics

Anglano (2014) performed an in-depth analysis of WhatsApp on Android devices. The work provided a comprehensive description of the artifacts generated by WhatsApp and discussed the decoding, interpretation and relationship between the artifacts. Anglano (2014) was able to provide an analyst with the means of reconstructing the list of contacts and chronology of the messages that have been exchanged by users.

The works by Thakur (2013) and Mahajan et al. (2013) are similar to previous studies since they both focused on the forensic analysis of WhatsApp on Android. These studies uncovered the forensic acquisition of the artifacts left by WhatsApp on the device. Thakur (2013) focused on the forensic analysis of WhatsApp artifacts on an Android phone's storage and volatile memory. The results showed that one is able to obtain many artifacts such as phone numbers, messages, media files, locations, profile pictures, logs and more. Mahajan et al. (2013) analyzed WhatsApp and Viber artifacts using the Cellebrite Forensic Extraction Device (UFED) toolkit. They were able to recover contact lists, exchanged messages and media including their timestamps and call details.

Walnycky et al. (2015) examined 20 different popular mobile social-messaging applications for Android including WhatsApp. In their work, they focused on unencrypted traffic that could be easily reconstructed. WhatsApp was found to be favorable at encrypting its network traffic when compared to other mobile social-messaging applications. Therefore, based on the primarily findings by Walnycky et al. (2015), our study aimed at further investigating and dissecting the WhatsApp protocol, and in specific, focusing on the signaling messages used when establishing WhatsApp calls given this new feature. However, in order to dive deeper into the signaling messages, one must understand some known attributes of the WhatsApp protocol which we discuss in Section 3 below.

3. WhatsApp protocol

WhatsApp uses the FunXMPP protocol for message exchange which is a binary-efficient encoded Extensible Messaging and Presence Protocol (XMPP) (WHAnonymous, 2015c). The WhatsApp protocol is also briefly described by LowLevel-Studios (2012) from an implementation perspective. To fully describe the FunXMPP protocol is beyond this paper's scope. For more information on the protocol the readers may want visit a website outlining the protocol[2].

3.1. Authentication procedure

There are two types of authentication procedures the WhatsApp client can use when connecting to the servers. If it is the first time the client is connecting to the server, a full handshake is performed as illustrated in Figure 1. Subsequently, for any consecutive connections, only a half handshake is executed using data provided from the initial full handshake.

We note that a half handshake therefore results in using the same session keys multiple times, which can be deemed as a plausible protocol security weakness.

3.1.1. Full handshake

The authentication procedure as described by the developers of WhatsAPI consists of three messages (WHAnonymous, 2015a). This is synonymous with the well known three-way handshake and is described in detail in the following paragraphs. These messages can be observed in Figure 1 which was created using our developed tool (for more details see Section 4).

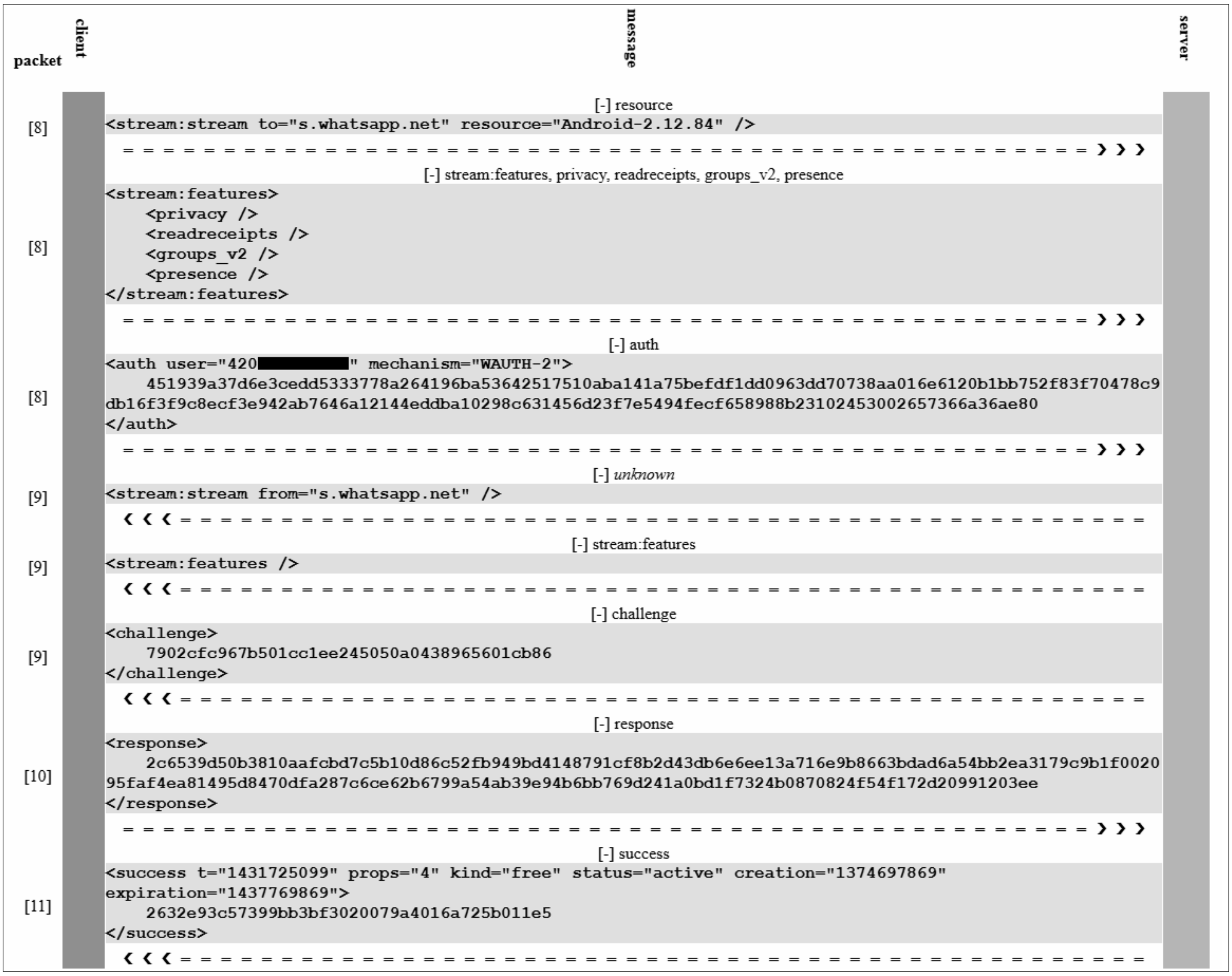
As shown in Figure 1, first, the client sends an `<auth>` message to the server. This message is not encrypted and contains the client number and authentication method the client wants to use.

Then, the server replies with a `<challenge>` message containing a 20 byte long nonce for the session key generation. Session keys are then generated using the Password-Based Key Derivation Function 2 (PBKDF2) algorithm using the password as a passphrase and the nonce as a salt. Both the server and the client know the password and nonce so the generated keys are the same on both ends. Four keys are generated in total: two keys for confidentiality (one for each direction - from the server and to server) and two keys for the integrity check (again one for each direction).

The client then creates a `<response>` message that consists of a concatenated client phone number in ASCII, nonce sent by the server in binary, current Unix timestamp in ASCII and other device description data. This message is encrypted using the generated session keys and it is prepended by the hash of the message for integrity checking purposes. Decrypted contents of the response message are illustrated in Figure 2, where we can see the aformentioned fields – their hexadecimal value and also the ASCII representation as displayed by Wireshark.

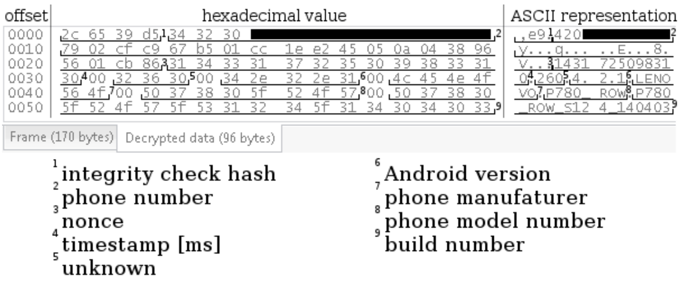
If registration is successful, the server replies with a `<success>` message that is encrypted. Otherwise, the

[2] `https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol`, last accessed 2015-07-03.



```
[-] resource
[8]   <stream:stream to="s.whatsapp.net" resource="Android-2.12.84" />
      = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = ❯❯❯
[-] stream:features, privacy, readreceipts, groups_v2, presence
[8]   <stream:features>
          <privacy />
          <readreceipts />
          <groups_v2 />
          <presence />
      </stream:features>
      = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = ❯❯❯
[-] auth
[8]   <auth user="420[redacted]" mechanism="WAUTH-2">
          451939a37d6e3cedd5333778a264196ba53642517510aba141a75befdf1dd0963dd70738aa016e6120b1bb752f83f70478c9
      db16f3f9c8ecf3e942ab7646a12144eddba10298c631456d23f7e5494fecf658988b23102453002657366a36ae80
      </auth>
      = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = ❯❯❯
[-] unknown
[9]   <stream:stream from="s.whatsapp.net" />
      ❮❮❮ = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
[-] stream:features
[9]   <stream:features />
      ❮❮❮ = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
[-] challenge
[9]   <challenge>
          7902cfc967b501cc1ee245050a0438965601cb86
      </challenge>
      ❮❮❮ = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
[-] response
[10]  <response>
          2c6539d50b3810aafcbd7c5b10d86c52fb949bd4148791cf8b2d43db6e6ee13a716e9b8663bdad6a54bb2ea3179c9b1f0020
      95faf4ea81495d8470dfa287c6ce62b6799a54ab39e94b6bb769d241a0bd1f7324b0870824f54f172d20991203ee
      </response>
      = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = ❯❯❯
[-] success
[11]  <success t="1431725099" props="4" kind="free" status="active" creation="1374697869"
      expiration="1437769869">
          2632e93c57399bb3bf3020079a4016a725b011e5
      </success>
      ❮❮❮ = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =
```

Figure 1: Full handshake between WhatsApp client and server.
Note: Numbers on the left side represent packet numbers (see Appendix A for the source pcap file). Also, there can be multiple messages in one packet.



```
offset  hexadecimal value                                   ASCII representation
0000    2c 65 39 d5 34 32 30 [redacted]                     .,e9.420[redacted]
0010    79 02 cf c9 67 b5 01 cc  1e e2 45 05 0a 04 38 96    y...g... ..E...8.
0020    56 01 cb 86 31 34 33 31  37 32 35 30 39 38 33 31    V...1431 72509831
0030    30 00 32 36 30 00 34 2e  32 2e 31 00 4c 45 4e 4f    0.260.4. 2.1.LENO
0040    56 4f 00 50 37 38 30 5f  52 4f 57 00 50 37 38 30    VO.P780_ ROW.P780
0050    5f 52 4f 57 5f 53 31 32  34 5f 31 34 30 34 30 33    _ROW_S12 4_140403
Frame (170 bytes)  Decrypted data (96 bytes)
```

[1] integrity check hash
[2] phone number
[3] nonce
[4] timestamp [ms]
[5] unknown
[6] Android version
[7] phone manufaturer
[8] phone model number
[9] build number

Figure 2: Content of <response> message with marked regions

server replies with a <failure> message that is not encrypted.

3.1.2. Half handshake

A half handshake consists only of an <auth> message that already contains the data of a <response> message described above, and the server's reply, a <success> message. The client uses the nonce from the earlier session which means that this nonce is not known by outsiders, therefore, it is not possible to decrypt such a session, as session encryption keys cannot be determined.

4. Tool for visualizing WhatsApp protocol messages

4.1. Description

Our tool is a command-line program written in Python (version 2.7). It is named convertPDML.py as it converts the PDML file exported from Wireshark to an HTML report. It is available in the form of source code, see Appendix A for more details. It requires one input parameter; a path to an XML file containing the details of dissected packets. See the step 5 in Section 6.3 for details on how to create the XML file.

The output of the tool is a report file containing all the messages exchanged between the WhatsApp client and the WhatsApp servers in HTML format as shown in Figure 1. Hence, any standard browser can be used to view the results. Messages are ordered chronologically as they appear in the input XML file.

4.2. Usage

As mentioned above, the tool requires am XML file as an input parameter. Example:
`convertPDML.py INPUT.xml`

5. Network traffic collection

This section explains how we collected the WhatsApp network traffic. More details are presented in Sections 5.1 and 5.2.

5.1. Experimental setup

We used the setup exemplified in Figure 3 for capturing network traffic between the WhatsApp messenger running on an Android phone and the WhatsApp servers. The hardware and software used in the experimental setup are listed below:

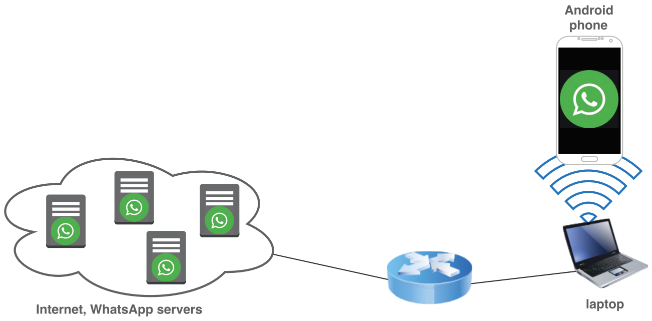


Figure 3: Experimental setup

Equipment used in experimental setup:

- Phone: Lenovo P780, Android 4.2.1, running
  - Whatsapp v2.12.84 which was downloaded from the Google play store.
  - Password Extractor v1.0[3].
- Laptop: Lenovo ThinkPad T420s with Windows 7 64-bit with the following installed software:
  - Wireshark v1.12.5, 32-bit, with the WhatsApp dissector [4].
  - Pidgin v2.12.11[5], 32-bit, with the WhatsApp plugin [6].

5.2. High level methodology

First, we disconnected the Android phone from any Internet connection and used the Password Exctractor application to gain access to the WhatsApp password. We note that the phone had to be rooted to use this application. We would also like to mention that there could have been multiple ways to gain access to the password on the device such as using commercially available tools to acquire a forensic image of the phone, and in some cases gaining access to the password can be achieved without rooting the phone if the acquistion method allows the investigator to acquire the image without rooting the device.

We then utilized Pidgin messenger with the WhatsApp plugin and obtained the WhatsApp password for connecting to the WhatsApp servers in order to desynchronize the WhatsApp client installed on the Android phone. This was performed in order for us to capture the full handshake (see Section 3.1.1 for more details).

The next step included setting up a wifi access point (see Figure 3) on the laptop and sharing the Internet connection from the ethernet port to the wifi adapter. The laptop now acted as a wifi router. We then started capturing all the traffic on the access point's network. In the next step, we connected the phone to the created wifi network and made a WhatsApp call to a user with phone number `1-203-xxx-xxxx`. Finally, we finished capturing the traffic and saved the created pcap file.

Following the aforementioned methodology allowed us to collect network traffic enabling us to perform exploratory analysis. In the following Section 6, we outline the resultant steps that we were able to reproduce for decrypting the WhatsApp messaging traffic.

6. Decryption

According to LowLevel-Studios (2012) and WHAnonymous (2015a) , encryption and decryption in WhatsApp is performed with a symmetric RC4 stream cipher using keys generated during authentication which is described in the Section 3.1.

Therefore, in order to decrypt the communication between the WhatsApp servers and the WhatsApp client, session keys for each direction (as WhatsApp uses one key

[3] `https://www.mgp25.com/downloads/pw.apk`, last accessed 2015-07-06

[4] `https://davidgf.net/page/37/whatsapp-dissector-for-wireshark`, last accessed 2015-07-06

[5] `https://pidgin.im/`, last accessed 2015-07-06

[6] `https://davidgf.net/whatsapp/`, last accessed 2015-07-06

for communication from device to the server and a different one for communication from the server to the device) are required. The process of obtaining these keys is provided in the Section 3.1.1.

6.1. Prerequisites

Our work showed that there are two mandatory requirements for the successful decryption of WhatsApp messaging connections:

- The password associated with the WhatsApp account.
- The record of the full handshake between the WhatsApp client and the server.

6.2. Tools used

We outline the list of software tools that were used in the decryption process:

- To obtain the password, there are multiple options based on the mobile device being used (WHAnonymous, 2015b). As we were using an already rooted Android phone, the easiest way was to extract password using Password Extractor application.
- To force WhatsApp to establish a full handshake the next time mobile device connected to the server, it was necessary to break the synchronization between the WhatsApp client and the server. The simplest way for doing that was to connect using a different client. For that purpose, we used the IM client Pidgin alongside the WhatsApp plugin.
- To decrypt the WhatsApp connection between the client and server, we used Wireshark and a WhatsApp-specific dissector.
- To visualize the WhatsApp protocol message exchange we created a command-line tool described in Section 4.

6.3. Decryption procedure

In this section, we elaborate using a step-by-step procedure describing how to successfully decrypt and visualize the exchange of WhatsApp protocol's messages between the WhatsApp client and the servers.

1. As the Android phone we were using, was rooted, obtaining the password was as easy as installing and running an application mentioned in the Section 6.2. In our case, the username (phone number) was `420xxxxxxxxx` with the following extracted password `627XlMqch8i5Ncy2tRSbZLXs2m0=`.
2. After obtaining credentials for the WhatsApp account (phone number and password), we disconnected the mobile device running WhatsApp from the wifi network and used the IM client Pidgin with the WhatsApp plugin and used the obtained credentials to log in to our WhatsApp account. This broke the synchronization between the WhatsApp client on the mobile device and the WhatsApp server forcing the client to authenticate using a full handshake.
3. We then connected the mobile device running the WhatsApp client back to the wifi access point capturing all the communication from and to the mobile device as explained in Section 5.1. After the WhatsApp client logged into the WhatsApp account, we placed a WhatsApp call to another device. All recorded communication was saved to a pcap file. Access to the pcap file is presented in the Appendix A.
4. After we captured all the communication between the WhatsApp client and the WhatsApp server, we provided the WhatsApp dissector in Wireshark with the credentials we obtained in the prior steps. To do that we used Wireshark's menu `Edit -> Preferences` and in the `Protocols` section we set up the WhatsApp dissector with the same options exemplified in Figure 4.
   After setting up the WhatsApp dissector correctly, we were able to observe the content of encrypted messages and the content of the `<response>` message should start with the number used in `<auth>` message as shown in Figure 2.
5. When the communication was decrypted we exported it to XML format using Wireshark's function `File -> Export Packet Dissections -> as XML - "PDML" (packet details) file...`. We provide access to this XML file in the Appendix A. Part of this XML file (namely `<auth>` is illusrated in Listing 1 where we can see the same values as in `<auth>` message from Figure 1 – attribute `user` with value `420xxxxxxxxx` (lines 33-38) and attribute `mechanism` with value `WAUTH-2` (lines 39-44).
6. The final step involved using our tool to generate a report of the WhatsApp message exchange between the WhatsApp client and WhatsApp servers. For that we used the XML file generated in the previous step. For more details refer to the Section 4 for details.

7. Findings

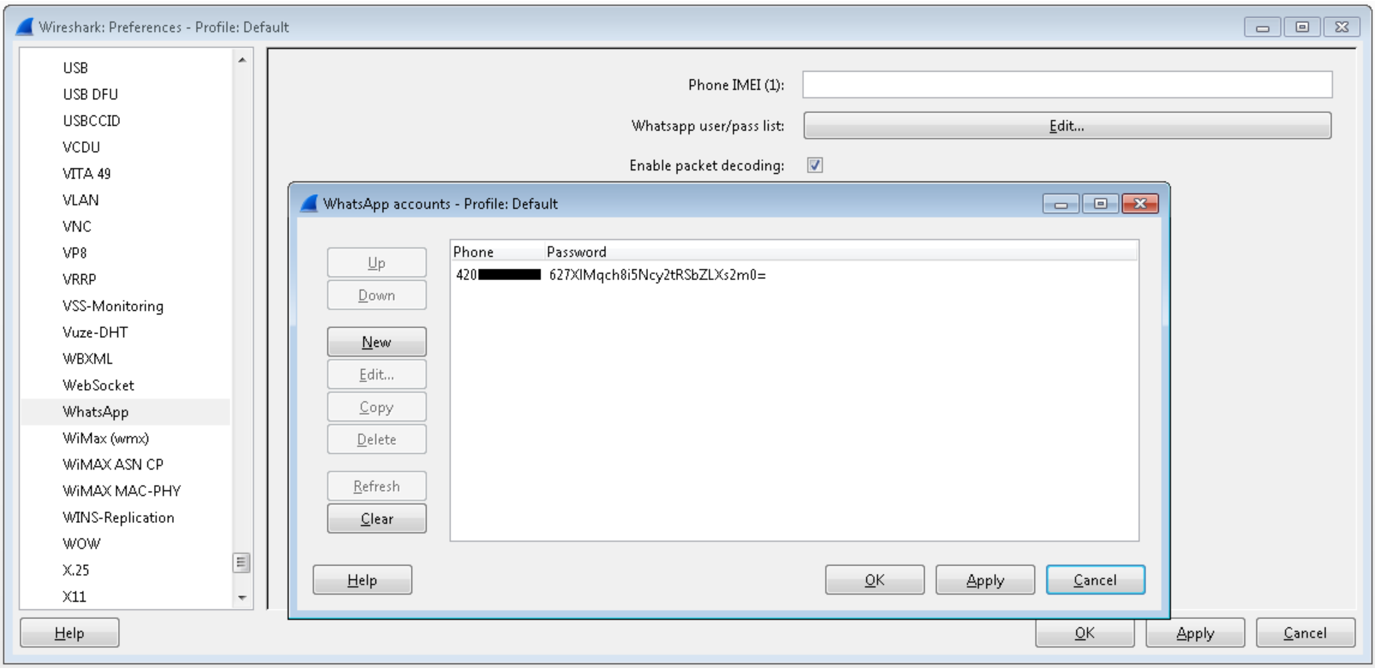
In the following subsections, we describe our findings on the signaling messages used for call establishment in WhatsApp. For a visual representation of our findings readers may want to refer to Figure 5.

7.1. Protocol analysis of call signaling messages

In this section we elaborate on messages that we hypothesize are part of the establishment a WhatsApp call as we observed it in the decrypted captured communication traffic. We used the captured pcap file and the HTML report generated from the same pcap file (refer to the Section 6.3 for more details). Both of these files can be downloaded

```
<packet>
  <proto name="geninfo" pos="0" showname="General information" size="239">
    <!-- omitted --> </proto>
  <proto name="frame" showname="Frame 8: 239 bytes on wire (1912 bits), 239 bytes captured
    (1912 bits)" size="239" pos="0">
    <!-- omitted --> </proto>
  <proto name="eth" showname="Ethernet II, Src: LenovoMo_62:f0:0c (c8:dd:c9:62:f0:0c),
    Dst: IntelCor_6a:38:85 (10:0b:a9:6a:38:85)" size="14" pos="0">
    <!-- omitted --> </proto>
  <proto name="ip" showname="Internet Protocol Version 4, Src: 192.168.137.208 (192.168.137.208),
    Dst: 174.37.231.87 (174.37.231.87)" size="20" pos="14"> <!-- omitted --> </proto>
  <proto name="tcp" showname="Transmission Control Protocol, Src Port: 44863 (44863),
    Dst Port: xmpp-client (5222), Seq: 1, Ack: 1, Len: 173" size="32"pos="34">
    <!-- omitted --> </proto>
  <proto name="whatsapp" showname="WhatsApp XMPP protocol" size="173" pos="66">
    <field name="whatsapp.message" showname="Message" size="26" pos="70" show="" value="">
      <!-- omitted --> </field>
    <field name="whatsapp.message" showname="Message" size="28" pos="96" show="" value="">
      <!-- omitted --> </field>
    <field name="whatsapp.message" showname="Message" size="115" pos="124" show="" value="">
      <field name="whatsapp.message" showname="Message size: 112" size="2" pos="125" show="112"
        value="0070"/>
      <field name="whatsapp.flags" showname="Flags: 0x00" size="1" pos="124" show="0" value="00">
        <field name="whatsapp.crypted" showname="0... .... = Crypted: False" size="1" pos="124"
          show="0" value="0" unmaskedvalue="00"/>
        <field name="whatsapp.compressed" showname=".0.. .... = Compressed: False" size="1" pos="124"
          show="0" value="0" unmaskedvalue="00"/>
      </field>
      <field name="whatsapp.node" showname="Node" size="112" pos="127" show="" value="">
        <field name="whatsapp.node" showname="Size: 6" size="1" pos="128" show="6" value="06"/>
        <field name="whatsapp.keyenc15" showname="Key: auth (12)" size="1" pos="129" show="12"
          value="0c"/>
        <field name="whatsapp.attr" showname="Attribute" size="9" pos="130" show="" value="">
          <field name="whatsapp.keyenc15" showname="Key: user (181)" size="1" pos="130" show="181"
            value="b5"/>
          <field name="whatsapp.nibbleencoded15" showname="Nibble encoded number (420xxxxxxxxx)"
            size="8" pos="131" show="" value=""/>
        </field>
        <field name="whatsapp.attr" showname="Attribute" size="2" pos="139" show="" value="">
          <field name="whatsapp.keyenc15" showname="Key: mechanism (86)" size="1" pos="139" show="86"
            value="56"/>
          <field name="whatsapp.valueenc15" showname="Value: WAUTH-2 (191)" size="1" pos="140"
            show="191" value="bf"/>
        </field>
        <field name="whatsapp.nodevalueplain" showname="Value [truncated]:
          E\0319\357\277\275}n&lt;\357\277\275\357\277\27537x\357\277\275d\031k\357\277\2756BQu\020
          \357\277\275\357\277\275A\357\277\275[\357\277\275\357\277\275\035\357\277\275\357\277\275=
          \357\277\275\a8\357\277\275\001na \357\277" size="96" pos="143" show="E\x199\xef\xbf
          \xbd}n&lt;\xef\xbf\xbd\xef\xbf\xbd37x\xef\xbf\xbdd\x19k\xef\xbf\xbd6BQu\x10\xef\xbf\xbd\xef
          \xbf\xbdA\xef\xbf\xbd[\xef\xbf\xbd\xef\xbf\xbd\x1d\xef\xbf\xbd\xef\xbf\xbd=\xef\xbf\xbd\x78
          \xef\xbf\xbd\x1na\xef\xbf\xbd\xef\xbf\xbdu/\xef\xbf\xbd\xef\xbf\xbd\x4x\xef\xbf\xbd\xef\xbf
          \xbd\x16\xef\xbf\xbd\xef\xbf\xbd\xef\xbf\xbd\xef\xbf\xbd\xef\xbf\xbd\xef\xbf\xbdB\xef\xbf
          \xbdvF\xef\xbf\xbd!D\xef\xbf\xbd\xef\xbf\xbd\xef\xbf\xbd\x2\xef\xbf\xbd\xef\xbf\xbd1Em#\xef
          \xbf\xbd\xef\xbf\xbdIO\xef\xbf\xbd\xef\xbf\xbdX\xef\xbf\xbd\xef\xbf\xbd#\x10$S"
          value="451939a37d6e3cedd5333778a264196ba53642517510aba141a75befdf1dd0963dd70738aa016e6120b1
          bb752f83f70478c9db16f3f9c8ecf3e942ab7646a12144eddba10298c631456d23f7e5494fecf658988b2310245
          3002657366a36ae80"/>
      </field>
    </field>
  </proto>
</packet>
```

Listing 1: Packet 8 containing WhatsApp `<auth>` message exported from Wireshark in XML format



Phone IMEI (1):

Whatsapp user/pass list: Edit...

Enable packet decoding: ☑

Phone	Password
420[redacted]	627XIMqch8i5Ncy2tRSbZLXs2m0=

Figure 4: WhatsApp Wireshark dissector settings

from Appendix A. In the rest of this section, we refer to the packet numbers displayed on the leftmost side in the flow diagram of signaling message exchange in Figure 5.

First (in packets [8]-[32]), the WhatsApp client connects and authenticates with the first WhatsApp server `174.37.231.87` but there is no activity regarding a call.

Starting with packet [33], the WhatsApp client connects and authenticates to a second WhatsApp server `174.36.210.45` and starts placing a call.

Right after connecting to the second server, in packet [41], the client asks for the presence of the called party (phone number `1-203-xxx-xxxx`) and starts the call establishment process by sending `<offer>` message to the called party. This happens in packet [42]. There we can observe the property `call-id="1431719979-2"` for the first time. This property remains constant throughout the rest of the signaling messages during the whole signaling process and it identifies the call as it is unique for each call and therefore changes every time a call is initiated.

In this first message we can also observe that the caller is offering to use the Opus codec (Valin et al., 2012) (in property `<audio>`) for voice data in two sampling rates, 8kHz and 16kHz. We also observe the properties `<p2p>` (value of 16 bytes = 128 bits) and `<srtp>` (192 bytes = 1536 bits) values which we were not able to decode. We postulate that they might be some kind of initialization vectors for encryption of media streams and/or description of these streams. The last property is `<te>` contains a 6 byte value that we decoded as the endpoint (IP address and port) where the client announces the endpoint address for the media stream. Its value is `192.168.137.208:46416`.

The server replies with `<ack relay>` in packet [43] which contains property `<token>` (value of 204 bytes = 1632 bits) which we were also unable to decode, multiple properties `<te>` that announce endpoint addresses of relay servers (8 servers in total), and properties `<encode>`, `<agc>` (gain control) and `<ns>` (noise suppression) that we hypothesize further specify media encoding.

Packets [44] and [45] carry messages `<receipt>` and `<ack>` of the receipt. To the best of our knowledge, these messages do not contain any data of interest.

Packet [46] going from the server to the client carries the message `<preaccept>` and has the property `<audio>` that asserts that the used codec for media streams will be the Opus codec at the sampling rate of 16kHz. It also contains the property `<srtp>` that has the same length as the same `<srtp>` property in packet [42] (192 bytes = 1536 bits) but carries different value.

Packet [47] carries the message `<transport>` which contains the client's endpoint address but from an external point of view - a public endpoint address. This address is found out by the client using Traversal Using Relays around NAT (TURN) mechanism (Mahy et al., 2010) – client asks the TURN server what is its (client's) IP address from the outside point of view. Its value is `64.251.61.74:62334` which differs from the value in packet [42] - `192.168.137.208:46416`. Packet [48] carries the `<ack>` message to the previous message.

We can observe a relay server election in packets [49]-[65]. The client finds out latency between itself and the relay servers obtained from message `<relay>` from packet [43] and one of the servers is elected.

The last message of the call establishment process is

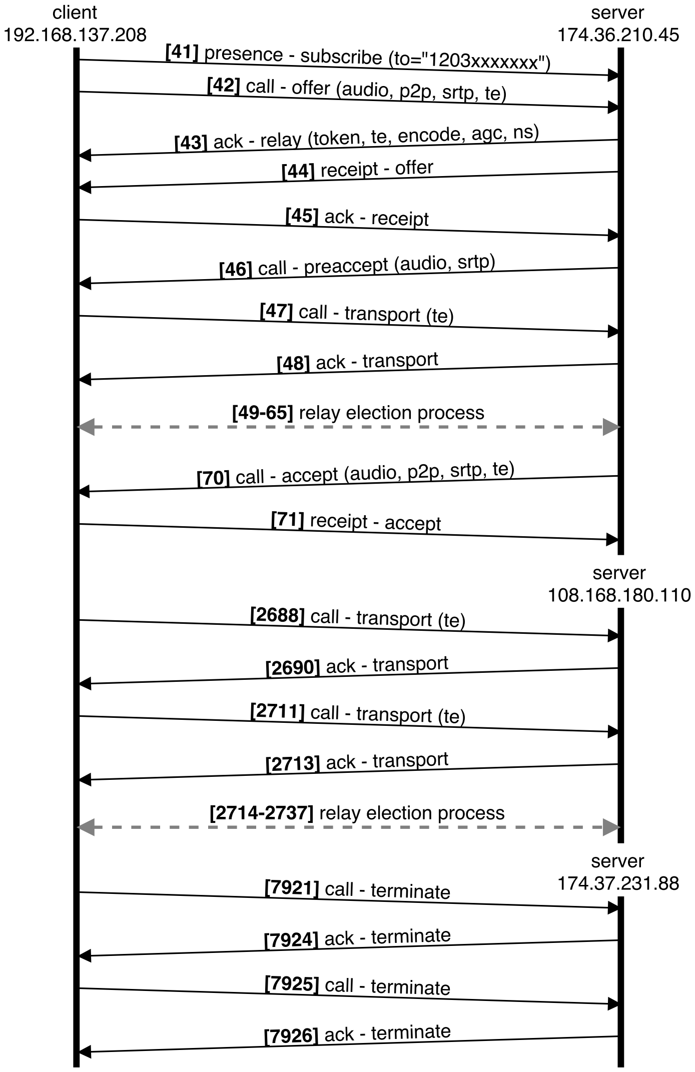

client
192.168.137.208
server
174.36.210.45
[41] presence - subscribe (to="1203xxxxxxx")
[42] call - offer (audio, p2p, srtp, te)
[43] ack - relay (token, te, encode, agc, ns)
[44] receipt - offer
[45] ack - receipt
[46] call - preaccept (audio, srtp)
[47] call - transport (te)
[48] ack - transport
[49-65] relay election process
[70] call - accept (audio, p2p, srtp, te)
[71] receipt - accept
server
108.168.180.110
[2688] call - transport (te)
[2690] ack - transport
[2711] call - transport (te)
[2713] ack - transport
[2714-2737] relay election process
server
174.37.231.88
[7921] call - terminate
[7924] ack - terminate
[7925] call - terminate
[7926] ack - terminate

Figure 5: Signaling messages of WhatsApp call (numbers refer to packet numbers)

message `<accept>` in packet [70]. It contains the property `<audio>` that confirms that used codec is Opus, sampling rate 16kHz, properties `<p2p>` and `<srtp>` (with the same value as in packet [47]) and two endpoint addresses: private - `192.168.1.22:55607` and public - `64.251.61.74:55607`. These endpoint addresses are used when trying to establish a direct peer-to-peer (P2P) connection. Packet [71] contains `<receipt>` message confirming the previous message.

After that, both-way media stream is established from `192.168.137.208:46416` to `31.13.74.48:3478` using RTP. These addresses were announced in a message in packet [42] and during the relay server election.

After about 30 seconds of the ongoing call, the client connects to another WhatsApp server (`108.168.180.110`) and signaling messages start flowing through this server. The client then announces new endpoint addresses in packets [2688] and [2711] and after a new relay election process, a new media stream is created replacing the previous one using a new endpoint address.

Finally, the client connects to another WhatsApp server (`174.37.231.88`) and sends two identical messages `<terminate>` in packets [7921] and [7925] and the call is terminated.

7.2. Media streams

Hancke (2015) mentioned in his report that WhatsApp uses a codec at 16kHz sampling rate with bandwidth of about 20kbit/s. Unlike us, Philipp Hancke did not have access to the decrypted signaling messages and thus we can now declare that WhatsApp is using the Opus codec for voice media streams at either 8kHz or 16kHz sampling rate which is decided at call setup.

We attempted to decode the media using the open-source implementation of the Opus codec[7] but the decoded result was not voice audio. From that and from the fact that we can observe `<srtp>` properties (SRTP stands for Secure Realtime Transport Protocol (Baugher et al., 2004)) we infer that these media streams are being encrypted.

7.3. Analysis summary

Through the analysis of signaling messages exchanged during a WhatsApp call we were able to:

- Closely examine the authentication process of WhatsApp clients.
- Discover what codec WhatsApp is using for voice media streams - Opus at 8 or 16kHz sampling rates.
- Understand how relay servers are announced and the relay election mechanism.
- Understand how clients announce their endpoint addresses for media streams.

Gaining insight into these signaling messages is essential for the understanding of the WhatsApp protocol especially in the area of WhatsApp call analysis from a forensic networking perspective.

7.4. Forensically relevant artifacts

As shown in Table 1, forensically relvant artifacts may be extracted from the network traffic using the outlined methodology. Most notably (see Figure 1), we were able to acquire the following artifacts from the network traffic:

- WhatsApp phone numbers.
- WhatsApp phone call establishment metadata and datetime stamps.
- WhatsApp phone call termination metadata and datetime stamps.

[7] http://www.opus-codec.org/, last accessed 2015-07-06

- WhatsApp phone call duration metadata and date-time stamps.

- WhatsApp's phone call voice codec (Opus)

- WhatsApp's relay server IP addresses used during the calls.

8. Conclusions

In this work, we decrypted the WhatsApp client connection to the WhatsApp servers and visualized messages exchanged through such a connection using a command-line tool we created. This tool may be useful for deeper analysis of the WhatsApp protocol.

We also uncovered the hypothesized signaling messages of the WhatsApp call which revealed what codec is being actually used for media transfer (Opus), as well as forensically relevant metadata about the call establishment, termination, duration and phone numbers assocaited with the call.

9. Future work

In this work we were unable to decode media RTP streams as they seem to be encrypted. However, we hypothesize that encryption keys are most likely being transfered inside the signaling messages during the set up of a WhatsApp call and therefore we postulate that it should be possible, in theory, to decrypt these media streams as well. The main challenge for this task is to find out the encryption keys and encryption algorithm used.

We would also like to note that a limitation of our work is that it was tested on an Android device. Although we hypothesize that the protocol used in the communication will be constant accross platforms, recreating the experiments with different devices and operating systems running WhatsApp is needed to validate that claim. Also, we would like to note that as more features are added to WhatsApp, more experiments need to be conducted to ensure that the design of the protocol does not change.

We would also like to encourage other researchers to apply the techniques explained in our work to analyze the network traffic of other popular messaging applications so that the forensic community can gain a better understanding of the forensically relevant artifacts that may be extracted from the network traffic, and not only the data stored on the devices.

10. References

Anglano, C. (2014). Forensic analysis of whatsapp messenger on android smartphones. *Digital Investigation*, *11*, 201–213. URL: http://www.sciencedirect.com/science/article/pii/S1742287614000437. , last accessed 2015-07-06.

Arce, N. (2015). Whatsapp calling for android and ios: How to get it and what to know. URL: http://www.techtimes.com/articles/38291/20150309/whatsapp-calling-for-android-and-ios-how-to-get-it-and-what-to-know.htm , last accessed 2015-05-27.

Baugher, M., McGrew, D., Naslund, M., Carrara, E., & Norrman, K. (2004). The secure real-time transport protocol (SRTP). URL: https://www.ietf.org/rfc/rfc3711.txt , last accessed 2015-07-06.

Hancke, P. (2015). Whatsapp exposed: Investigative report. URL: https://webrtchacks.com/wp-content/uploads/2015/04/WhatsappReport.pdf , last accessed 2015-06-03.

LowLevel-Studios (2012). Whatsapp protocol 1.2: A brief explanation. URL: http://lowlevel-studios.com/whatsapp-protocol-1-2-a-brief-explanation/ , last accessed 2015-06-03.

Mahajan, A., Dahiya, M., & Sanghvi, H. (2013). Forensic analysis of instant messenger applications on android devices. *arXiv preprint arXiv:1304.4915*, . URL: http://arxiv.org/abs/1304.4915. , last accessed 2015-07-06.

Mahy, R., Matthews, P., & Rosenberg, J. (2010). Traversal using relays around NAT (TURN). URL: https://tools.ietf.org/html/rfc5766 , last accessed 2015-07-06.

Schulzrinne, H., Casner, S., Frederick, R., & Jacobson, V. (2003). RTP: A transport protocol for real-time applications. URL: https://www.ietf.org/rfc/rfc3550.txt , last accessed 2015-07-06.

Thakur, N. S. (2013). *Forensic analysis of WhatsApp on Android smartphones*. Master's thesis University of New Orleans. URL: http://scholarworks.uno.edu/td/1706/ , last accessed 2015-07-06.

Valin, J., Vos, K., & Terriberry, T. (2012). Definition of the opus audio codec. URL: http://tools.ietf.org/html/rfc6716 , last accessed 2015-07-06.

Walnycky, D., Baggili, I., Marrington, A., Moore, J., & Breitinger, F. (2015). Network and device forensic analysis of android social-messaging applications. *Digital Investigation*, *14*, S77–S84.

WHAnonymous (2015a). Authentication overview (WAUTH 2). URL: https://github.com/WHAnonymous/Chat-API/wiki/Authentication-Overview-(WAUTH-2) , last accessed 2015-06-03.

WHAnonymous (2015b). Extracting password from device. URL: https://github.com/WHAnonymous/Chat-API/wiki/Extracting-password-from-device , last accessed 2015-06-12.

WHAnonymous (2015c). Funxmpp-protocol. URL: https://github.com/WHAnonymous/Chat-API/wiki/FunXMPP-Protocol , last accessed 2015-06-03.

Appendix A. Reference files

These are files that were used throughout this paper. These files can be provided to researchers by visiting our website http://www.unhcfreg.com under Tools & Data.

- `whatsapp_register_and_call.pcap` - pcap file containing user with phone number `420xxxxxxxxx` connecting to multiple WhatsApp servers and placing a call to the user with phone number `1-203-xxx-xxxx`.

- `whatsapp_register_and_call.xml` - content of previous pcap file exported from Wireshark in XML format.

- `whatsapp_register_and_call.html` - HTML file that was generated from previous XML file using our tool.

- `convertPDML.py` - command-line tool for converting XML files exported from Wireshark to a visual

Table 1: Forensically relevant data, their location and sample data

Data type	**Location**	**Sample data**
WhatsApp password	device storage	/data/data/com.whatsapp/files/pw

phone numbers	database files, network traffic	”user” values in `<auth>` messages and ”from” and ”to” values in `<call>` and `<ack>` messages:
		`<auth user="420xxxxxxxxx" mechanism="WAUTH-2">`
		`<call to="1203xxxxxxx" id="1431719979-3">`
		`<ack from="1203xxxxxxx" id="1431719979-3" class="call" type="offer">`

phone call establishment	database files, network traffic	timestamp of `<accept>` message: `<field name="timestamp" pos="0" show="May 15, 2015 23:26:48.025662000 Central Europe Daylight Time" showname="Captured Time" value="1431725208.025662000" size="453" />`
phone call termination	database files, network traffic	timestamp of `<terminate>` message: `<field name="timestamp" pos="0" show="May 15, 2015 23:28:12.177489000 Central Europe Daylight Time" showname="Captured Time" value="1431725292.177489000" size="134" />`
phone call duration	database files, network traffic	”duration” value in `<terminate>` message: `<terminate call-id="1431719979-2" duration="84000" />`

phone call voice codec	network traffic	”audio” value in `<call>` and `<accept>` messages:
		`<audio enc="opus" rate="8000" />`
		`<audio enc="opus" rate="16000" />`

relay server used during call	network traffic	”te” value in `<relayelection>` messages:
		`<relayelection call-id="1431719979-2">`
		`  <te latency="-98122">`
		`    31.13.74.48:3478 (1f0d4a300d96)`
		`  </te>`
		`</relayelection>`

HTML report containing flow of WhatsApp messages exchanged between WhatsApp Messenger and the WhatsApp servers.