Forensic Analysis of Artifacts in the Matrix Protocol and Riot.IM application

By:
Guido Schipper, Rudy Seelt and Nhien-An Le-Khac

From the proceedings of
The Digital Forensic Research Conference
DFRWS EU 2021
March 29 - April 1, 2021

Forensic Science International: Digital Investigation 36 (2021) 301118

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

Full Paper

Forensic analysis of Matrix protocol and Riot.im application

Guido Cornelis Schipper [a], Rudy Seelt [b], Nhien-An Le-Khac [c, *]

[a] Dutch National Police, The Hague, The Netherlands
[b] Police Academy of the Netherlands, P.O. Box 834, 7301 BB, Apeldoorn, the Netherlands
[c] University College Dublin, Belfield, Dublin 4, Ireland

ARTICLE INFO

Article history:
Available online 23 March 2021

Keywords:
Instant messaging forensics
Matrix protocol
Riot.im
Forensic analysis
Database forensics

ABSTRACT

Instant messaging (IM) has been around for decades now. Over the last few decades IM has become more and more popular with varied protocols, both open source and closed source. One of the new recent open source ones is the Matrix protocol with the first stable version released in 2019 and the IM application based on this protocol is "Riot.im". In recent years many organizations started using the Matrix protocol to setup and manage their own IM platforms. In addition, the number of users who are using the public Matrix protocol-based servers is also increasing. However, because the Matrix protocol and the Riot.im application are very new, there is a knowledge gap when it comes to investigators in relation to the forensic acquisition and analysis of Riot.im application and the Matrix protocol. Yet, there is very little research in literature on the Matrix protocol forensics. The goal of this paper is to fill this gap by presenting a forensic approach to analyze forensic artifacts of Riot.im and the Matrix protocol.

1. Introduction

Nowadays instant messaging (IM) is an integral part of our lives. A large part of the population has a smartphone or computer with one or more IM applications. Besides the well-known companies like WhatsApp and Facebook, who offer centralized IM platforms, there are also IM protocols that can be used without any license fees to setup and manage a decentralized IM platform. Although centralized IM platforms mostly work very well and are very accessible for the average consumer, there are also some disadvantages. For example you have to trust the company that manages the IM platform. Legal restrictions can also be a challenge when using centralized IM platforms, especially when the IM platform has to be used by businesses or governments. Consequently, several decentralized IM protocols have been developed in recent decades. In 2014 the development of the Matrix protocol started and it was called "Amdocs Unified Communications" (Makkonen, 2019). The goal of the Matrix protocol is to create an open platform for secure, decentralized and real-time communication (Matrix.org, 2020a). Riot.im is an IM-application, which has been developed by the developers of the Matrix protocol, and uses the Matrix protocol.

In recent years the Matrix protocol and Riot.im has increased in popularity with companies, governments and individuals alike. For example, the French government announced in 2019 that they were going to use the Matrix protocol and a modified version of Riot.im for internal communication for their six million employees (Jacob et al., 2019). In 2020 the German armed forces announced that they are going to use the Matrix protocol as their main IM platform in the near future (BWI, 2020). Another big shift to the Matrix protocol, which was announced in 2020 was the shift Mozilla made. They switched their 22 years old Internet Relay Chat (IRC) network off and migrated to the Matrix protocol (Irc and Matrix, 2020). Besides the legitimate use of decentralized IM-protocols by governments and businesses, criminals and terrorists (MEMRI, 2020) also use these techniques to secure their communication. This makes it harder for Law Enforcement (LE) agencies to intercept and read their communications. The fact that nowadays most modern IM-protocols support end-to-end encryption by default makes it even harder for LE-agencies to access communications between criminals. The Matrix protocol and Riot.im for example are using end-to-end encryption by default as of May 2020 (Matrix.org, 2020b). Endpoint forensics becomes more important because the popularity of end-to-end encryption. The reason for this is because at the endpoints the messages are unencrypted. Also, the key management to encrypt and decrypt the messages is handled at the endpoints, which makes the endpoints even more interesting for digital investigators.

* Corresponding author. University College Dublin, Belfield, Dublin 4, Ireland.
E-mail addresses: guido.schipper@ucdconnect.ie (G.C. Schipper), Rudy.Seelt@politieacademie.nl (R. Seelt), an.lekhac@ucd.ie (N.-A. Le-Khac).

https://doi.org/10.1016/j.fsidi.2021.301118

The Matrix protocol and Riot.im are relatively new with the first stable version of the Matrix protocol released in June 2019 (Matrix.org, 2019). Up to the time of writing this paper, a number of studies have been conducted into the use of cryptography by the Matrix protocol and Riot.im, but no research has yet been conducted from a forensic point of view. This creates a knowledge gap when it comes to forensic relevant artifacts regarding sent and received messages and their value. For example, how are messages handled by Riot.im and the Matrix protocol and what information can be found related to messages sent? Another knowledge gap has to do with how Riot.im stores its data on the hard drive. At which location on the hard drive does Riot.im store its data and how can it be made accessible for further analysis? Hence, the objective of this paper tries to close this gap by presenting a forensic approach for analyzing Matrix protocol and Riot.im to give more insight in the inner working of the Matrix protocol and Riot.im regarding message handling.

The rest of this paper is organized as follows: Section 2 presents related work on IM forensics and analyzing of Matrix protocol and Riot.im. We describe our forensic approach in Section 3. Our experiments and discussion are illustrated in Section 4. Finally, we provide a conclusion in Sections 5.

2. Related work

2.1. IM forensics

Sudozai et al. (2018) did research into the working and the forensic artifacts of the IMO IM application on Android and iOS.

Ababneh et al. (2017) also conducted research into the IMO IM application from a forensic perspective on Android and Windows. A step-by-step method was used for this research including the memory acquisition and analysis.

Sgaras et al. (2015) focused on the forensic acquisition and analysis of different IM applications on both iOS and Android platforms. Authors also discussed IM chat cloning and communication interception.

The research conducted by Majeed et al. (2016) focused on the forensic analysis of Viber, Facebook and Skype on a Windows 10. Gregorio et al. (2017) conducted a forensic analysis of the Telegram messenger on a Windows Phone with three main steps: Open knowledge; Analysis of artifacts and Source code analysis.

In (Thantilage and Le-Khac, 2019), authors described a volatile memory forensic approach for the retrieval of social media and IM evidences for some popular apps. However, this approach is not for Riot.im.

Anglano et al. (2016) has conducted a forensic analysis of the ChatSecure IM application on Android smartphones. To conclude this research different scenarios were conducted that simulated user actions. After each scenario a forensic image was created of the internal storage. After the scenarios were conducted, the different forensic images of the storage were analyzed.

Thantilage et al. (Thantilage and Le-Khac, 2020) looked at the forensic analysis of e-dating applications – Tinder and Coffee Meets Bagel – by examining iPhone backups. They did not investigate at the protocol level.

Alyahya et al. (Alyahya and Kausar, 2017) conducted a forensic analysis of the Snapchat application on Android by creating and sending multiple different messages and files and then analyzing the forensic images of the internal storage of the smart phones.

In (Wijnberg and Le-Khac, 2020), authors presented a forensic approach based on wiretaps to intercept the SMS message sent by WhatsApp and use this SMS to install and verify WhatsApp on the examiner phone.

Cents et al. (Cents and Le-Khac, 2020) also proposed an approach to identify different patterns of WhatsApp's communication using only wiretap data.

2.2. Matrix protocol and Riot.im analysis

Floris Hendriks conducted research into the techniques used in relation to key management in the Matrix protocol (Hendriks et al., 2020). Hendriks found that the Matrix protocol does a good job when it comes to encrypting data end-to-end. It however turned out that although the content of the message is encrypted, there is still a lot of user identifiable data that is not encrypted and can be seen by potential attackers.

Authors in (Jacob et al., 2019) address scalability issues of the Matrix protocol. The scalability problems particularly occur in relation to group messaging and a large number of users who are using one of the participating servers. Rebalancing users over multiple servers is hard due to the trust between servers, which is limited.

In (Mujaj, 2017) authors conduct the research and comparison of multiple secure messaging protocols and their implementation is described. This research focusses on the protocols and mobile chat applications of Signal, WhatsApp, Wire, Viber, Riot and Telegram protocols and mobile chat applications. A comparison is done of different aspects of the different protocols. In relation to the Riot application and the Matrix protocol some recommendations for improvement where done. These recommendations are related to the verification options offered by Riot, the implementation of two-step verification, passcode protecting the application and screen security (blocking the ability to take screenshots).

The Matrix protocol and Riot.im are relatively new. Over the last few years there has been done some research into for example the implementation of encryption into the Matrix protocol and how encryption keys are handled by the protocol. There has also been done a study into the performance and scalability of the Matrix protocol. To the best of our knowledge there haven't been done any research from a forensically point of view yet.

3. Methodology

3.1. Experimental platforms

The testing environment consists of two Windows 10 virtual machines hosted by VirtualBox (6.1.10) that offers a snapshot functionality, which allows to restore previous states of a virtual machine. Riot.im application (1.6.5) is installed on each virtual machine. To be able to send messages, three different accounts will be used. Two accounts will be used on virtual machine one (PC-01) and one account will be used on virtual machine two (PC-02). On PC-01, one user will be logged in using the Riot.im desktop application and one user will be logged in using the web client of Riot.im via the URL *https://riot.im/app*. The reason for this is that to be able to create a group conversation more than two participants are needed. Because Matrix.org is currently the most used Matrix server, the Matrix.org Matrix server will be used for this research. This Matrix server can be reached at the URL *https://matrix-client.matrix.org*. Fig. 1 shows a graphical overview of the testing environment.

In total there are three rooms created. One room is created for a one-to-one conversation between two accounts. Because end-to-end encryption is enabled by default for one-to-one conversations and cannot be disabled, there is only one room created for one-to-one conversations. In addition, there are two rooms created for group conversations. One of these rooms uses end-to-end encryption and the other will not. The generated dataset for testing is consist of text messages and multimedia messages of different file types. These messages are sent in different rooms by different users.

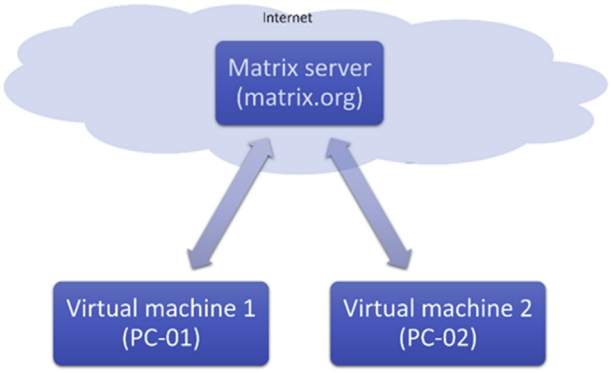

Fig. 1. Overview of the testing environment.

3.2. Riot.im application analysis

During this phase a comprehensive analysis is conducted of the user interface of the Riot.im application. The focus is on the way text and multimedia messages are presented, how multimedia messages are handled by the application and what other relevant information from a digital investigator point of view can be found by using the Riot.im application interface. Because the Riot.im application downloads the messages and files from the Matrix server, a comparison will also be done of available information in a logged-on client that is online and a logged-on client, but then in an offline state. The reason for these scenarios and the comparison is to get more insight in what information is available in different stages. This comparison will be done based on the following three scenarios: online Riot.im application; offline Riot.im application without downloaded messages before; offline Riot.im application with downloaded messages before. All of the three scenarios will be conducted using PC-01 (cf. Fig. 1). This comparison gives an idea of the way information is presented to the user, what information is available locally in different scenarios and what information is not. This comparison will be conducted on all messages sent in all rooms (cf. Section 3.1).

3.3. Matrix protocol analysis

The Matrix protocol uses events for the communication between components in the network. Message events are the type of events that are used for text and multimedia messages. The Matrix protocol describes a couple of key–value pairs and JSON objects that can be added to message events, which give more information about the sent message. Only a small portion of these key–value pairs and objects are mandatory, the others are optional. This means every client can choose whether or not these optional key–value pairs and objects are added to the message event. In this phase, we will take a better look at the message events send by a Riot.im application. This comparison gives an overview of the key–value pairs and objects that are added by the Riot.im application per message type. The comparison will also be conducted between the events of encrypted messages and unencrypted messages of the same message type. The properties stored in the messages event, which are related to files will also be compared with the real properties of the files. The goal is to validate the value of the properties in the event. This comparison gives more insights in what information is available and how it is stored within the message event. This comparison will focus on the following JSON event message types: (i) Text messages (m.text); (ii) Picture messages (m.picture); (iii) Audio messages (m.audio); (iv) Video messages (m.voice); (v) Generic files (m.file).

This comparison will be conducted using the messages and files send and received in the testing environment (cf. Section 3.1). Because the Riot.im application offers extensive functionalities to view and extract the unencrypted, encrypted and decrypted message events, these functionalities will be used to retrieve the message events.

3.4. Artifact analysis

The analysis will take place by using the Riot.im application and by analyzing the message events. The result will give an overview of the forensic value of the different artifacts. The following artifacts and functionalities will be examined: (i) Properties of sent and received files; (ii) *"origin_server_ts"* timestamp in message events; (iii) Message events in offline mode; and (iv) Unsigned age key–value pair.

3.4.1. Properties of sent and received files

The file properties of sent files will be compared with the properties of the received version on the file. The goal is to determine if files are altered in any way during transmission. During this comparison the technical properties including meta-information will be compared. The sent and received files will also be compared based on their SHA256 hash value. Based on this comparison we will be able to determine whether the files are altered during transmission.

3.4.2. "origin_server_ts" timestamp in message events

Riot.im and the Matrix protocol are using timestamps at multiple locations. In message events for example there are two key–value pairs that use timestamps or are related to timestamps. The first one is the *"origin_server_ts"* key–value pair, the second one is the "*age*" key–value pair. Based on the description in (Client-Server API, 2020), we can conclude that the timestamp stored in the key–value pair *"origin_server_ts"* is in fact the timestamp of the Matrix server, instead of the date and time of the client that sent the message. The following operations will be performed to determine the forensic value of the *"origin_server_ts"* artifact:

1. PC-01 was started;
2. The system data and time will be changed to four days back of the current date and time;
3. Start the Riot.im application;
4. Send messages in both encrypted and unencrypted rooms (room01, room02 and room03);
5. Analyze the "origin_server_ts" key–value pair in the message events of the send messages to determine the timestamps;
6. Compare the timestamps of the events with the date and time of PC-01 of the moment the messages where send.

When the timestamp is generated by the client, the expectation is that the timestamp in the message event is also four days back in time compared to the current date and time. When this is not the case it can be concluded that the timestamp is not generated by the client, but by the Matrix server.

3.4.3. Message events in offline mode

Messages can be created while Riot.im is offline and that these messages will be send when the connection to the Matrix server is restored. To examine how these offline created messages are handled by the Riot.im application, the following operations were performed in our experiments:

1. PC-01 was started;
2. Riot.im application was started on PC-01;

3. PC-02 was started;
4. Riot.im application was started on PC-02;
5. The network adapter of PC-01 was disabled in the control panel of Windows 10;
6. A text message was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted;
7. A picture was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted;
8. A video file was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted;
9. The message events of the send messages in steps 6, 7 and 8 are exported using the Riot.im application;
10. After 15 min the network adapter of PC-01 was enabled again. The time of this action was noted;
11. After the connection was restored and the messages were delivered the message events were exported out of the Riot.im application;
12. A comparison was made between the message events in offline mode and the message events after delivery.

The result of this experiment will be more knowledge of how sent messages in offline modes are handled, how message events are handled in an offline state and in what way they differ from message events that were generated in an online state.

3.4.4. Unsigned "age" key–value pair

As mentioned earlier, message events contain a JSON object called "unsigned". This object contains key–value pairs that are not part of the signing process. This means the key–value pairs in this object can be altered after the massage event has been signed and added to a room graph. One of the key–value pairs that is part of almost every message event is a key–value pair called *"age"*. Looking at the description of this artefact in the Matrix document (GitHub, 2020a), it can potentially be a very interesting artefact from a digital forensics point of view. It could for example possible tell something about the moment the message event was downloaded from the Matrix server. This is under the assumption that the server updates the value at the moment the event was downloaded. To be able to determine the value from the *"age"* artefact, we analyzed newly created message events. The analysis is based on three scenarios. To be sure the date and time of the matrix server was correct while conducting the experiments, the timestamp in the *"origin_server_ts"* key–value pair was constantly compared to the actual date and time the messages were send to the Matrix server. Based on the description above, the hypothesis is that the value in the *"origin_server_ts"* key–value pair and the value in the "age" key–value pair together will be the timestamp of the moment the message event was downloaded from the server. The first scenario will be used to validate this hypothesis. The following three scenarios will be conducted:

3.4.4.1. Scenario 1. In this scenario, both the sending and receiving client have an active connection to the Matrix server. The scenario was conducting by performing the following steps:

1. Start PC-01;
2. System time of PC-01 was the same as the real date and time;
3. Start Riot.im application on PC-01;
4. Start PC-02;
5. System time of PC-02 was the same as the real date and time;
6. Start Riot.im application on PC-02;
7. A text message was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted. The messages were sent from PC-02;
8. The time of receiving the messages on PC-01 was noted;
9. A picture was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted. The messages were sent from PC-02;
10. The time of receiving the messages on PC-01 was noted;
11. A video file was sent in Room03 (encrypted) and Room02 (unencrypted). The time of sending was noted. The messages were sent from PC-02;
12. The time of receiving the messages on PC-01 was noted;
13. The message events of the send messages of steps 6, 7 and 8 were exported using the Riot.im application of PC-01, the receiving client;

After retrieving the messages events, an analysis was conducted of the noted dates and times and the timestamps in the *"origin_server_ts"* and *"age"* key–value pair of the exported message events.

3.4.4.2. Scenario 2. Scenario two will be the scenario in which the sending client (PC-02) has a connection to the Matrix server, and the receiving client (PC-01) has no connection to the Matrix server at first. After the messages have been send, a couple of minutes later the connection of the receiving client is restored and the messages are downloaded. After receiving the messages, the timestamps are compared to the timestamps of the moment the messages were downloaded. This scenario is performed to determine when the timestamp in the *"age"* key–value pair is added: is it added when creating the message event or when downloading the message event? After retrieving the messages events, an analysis was made of the noted dates and times and the timestamps in the *"origin_server_ts"* and *"age"* key–value pair of the exported message events.

3.4.4.3. Scenario 3. Scenario three is the scenario in which the date and time of the receiving client computer (PC-01) will be set four days back in time. The goal of this scenario is to determine if a wrong date and time of the client computer affects the timestamp in the "age" key–value pair. After retrieving the messages events, an analysis was made of the noted dates and times and the timestamps in the *"origin_server_ts"* and *"age"* key–value pair of the exported message events.

3.5. Disk analysis

During the previous phases we have looked at how messages are handled by Riot.im and the Matrix protocol and what the forensic value is of different artifacts from a digital forensic point of view. After determining their value, it would be important to know where these artifacts are stored on a hard disk and how they can be retrieved. This phase consists of multiple steps. The first step is to determine the storage locations used by the Riot.im application. Disk monitoring method is used to get more insight in which locations on disk are used by the Riot.im application to store log files and chat history.

After the storage location was found, the directories and files at this location were analyzed. This analysis was conducted on the hard drive of virtual machine PC-01. The goal of this phase was to determine what information was stored and could be extracted.

4. Finding and analysis

4.1. Riot.IM application analysis

No administrator rights are required when installing the Riot.im application on Windows 10. That means all Riot.im related files are stored within the user environment. Riot.im does not offer the ability to change the account's username. The Riot.im application

does not have the option to protect the application with some sort of password or pin code. A logged in user will stay logged in, even after restarting the application.

The user interface of the Riot.im application offers by default a lot of options that could be very interesting from a digital forensics point of view. Although the chat interface shows a lot of information, there is more information available underneath. This information is stored in the state and message events. A more in-depth review of these events and their contents will be given in the next section. The Riot.im application has the ability to easily retrieve the message events belonging to a message. The DevTools, which are integrated into the Riot.im application offers the ability to easily retrieve state event messages, such as state messages related to invitations to a room or right management in a room. Riot.im application has by default the ability to view conversations in a way suspects would and it has options to dive deeper into the source of messages (message events) and rooms (state events) and retrieve more information and evidence that way.

After running the three scenarios, it became clear that if the Riot.im application has been used, there is a very good change that the contents of the chats and multimedia files are cached somewhere. Multimedia files that have not been watched before are not cached in advance. The scenarios also showed that when device identifiers and their public names are interesting from a digital forensics point of view, they cannot be retrieved using the Riot.im application in an offline mode.

4.2. Message event analysis

As mentioned in Section 3.3, the contents of a message event is based on the extensible JSON format. Each message event has a couple of fixed key–value pairs and two fixed JSON objects. When a client sends a message to the server, it only sends the contents together with some extra key–value pairs. Key-value pairs not sent by the client are generated by the Matrix server. Table 1 gives an overview of which key–value pairs are generated by the Matrix server and which are generated by the client, in this research the Riot.im client.

The *"event_id"* uniquely identifies each message event, this value can be relevant during investigations to identify which message event is about. This key–value pair is generated by the Matrix server, so users cannot alter this value.

The *"origin_server_ts"* turned out to be the time in milliseconds from January 1st 1970. This key–value pair will be described further in Section 4.3.

The *"content"* object is used to store the message contents. Based in the message type (i.e. text, multimedia) the contents can different. In the case of text messages, for example, this object contains two required key–value pairs, a "body" key–value pair, which is used to store the content of the text message and an "msgtype", which is used to store the message type "m.text".

The *"sender"* key–value pair is used to store the full username of the user that sent the message. With full username is meant the at (@) sign at the beginning of the username, which is followed by the username, a colon and then the domain name of the Matrix server the username is registered on.

Table 1
Overview of creation of fixed key–value pairs.

Key-value pairs	Matrix (server)	Riot.im (cient)
event_id	x	
origin_server_ts	x	
content (object contents)		x
sender		x
type		x
Unsigned (object contents)	x	x
room_id		x

The event type is defined by the value in the *"type"* key–value pair. In the case of clear text message, the value in this key–value pair is *"m.room.message"*. When an end-to-end encrypted message is sent, the value in this key–value pair is *"m.room.encrypted"*. With events of this type, the contents in the *"content"* object have a fixed format. The *"content"* object contains the at least three required key–value pairs. When the Megolm algorithm is used, another two key–value pairs are required: *"cyphertext"* and *"device_id"*. The most interesting key–value pair is the *"device_id"*. Each device used with the Matrix protocol and supports end-to-end encryption, has a unique device identifier. This identifier in combination with session keys are used to decrypt messages send by the device. The device identifier of a device can be found in the general settings of the Riot.im application. This identifier can be used to determine which device was used to send a message. Each device identifier also has an editable *device description*. This description can be edited by the user of the device at any given time and is not part of message events. The description however, is visible in the Riot.im interface. This description can tell something about which devices a suspect is using, or which language a suspect speaks.

The *"unsigned"* object contains key–value pairs that are not part of the signing process. This means they can be altered after signing the event. Examples of key–value pairs that are often part of this object are the "*age*" and "*transaction_id*" key–value pairs. Both these key–value pairs will be discussed in section 4.3.

The *"room_id"* is a unique identifier of a room. Room identifiers start with an exclamation mark, followed by a random string of characters, a colon followed by the domain name of the Matrix server.

As already mentioned, all fixed key–value pairs are added by the server. This means that, under the assumption that a suspect has no access to the server, these values cannot be altered by a suspect after the event was created.

4.3. Artifact analysis

4.3.1. Properties of sent and received files

The first experiments were conducted to determine how files properties and meta-information are handled by the Matrix protocol and Riot.im. Every file has properties and meta- and EXIF-information. Some other popular IM applications like WhatsApp are removing meta-information and sometimes altering the properties of files sent to for example reduce the file size. To determine if the Matrix protocol and/or the Riot.im application also alters file properties or removes meta-information, experiments were conducted.

The results of these experiments showed that the properties of the files were not altered. The SHA256 hashes of the received files matched the SHA256 hashes of the original files.

It also turned out that the EXIF-information was unchanged. This information can be very relevant during law enforcement investigations. GPS locations present as EXIF-information can help investigators for example to determine the physical location of a suspect.

Other meta-information about for example the date and time the file was last edited turned out to be changed. This behavior however was expected, because when files are downloaded the operating system in fact creates a new file, which is used to store the data of the downloaded file. This means that the data in the file is the same as the original, but file on the hard disk is in fact a new file with a new date and time of creation. This means that during

investigations the date and time of downloaded files could be used to determine when a file was downloaded, but it cannot be used to determine when the sender of the file created or modified the file.

4.3.2. "origin_server_ts" timestamp in message events

The creation date and time of a message event is stored in the "*origin_server_ts*" key–value pair. The date and time is stored in the Unix epoch format (Client-Server API, 2020). Through our experiments, it's noticed that the "*origin_server_ts*" key–value pair is generated by the Matrix server and that the user of the Riot.im application cannot influence the timestamp stored in the message event. Even if the date and time of a suspect's computer is out of sync, that would not influence the timestamp. Because the Matrix server generates the timestamp the date and time of the Matrix server has to have the correct time. It can be a challenge during investigations to determine whether the date and time of the server were correct in the past. By sending messages as an investigator, of course not by using the computer or account of a suspect, the correctness of the date and time of the server can be determined at that moment. This can be done by matching the timestamp in the message event against the noted date and time of sending the message.

4.3.3. Message events in offline mode

During the experiments it turned out that the Riot.im didn't send the messages created in offline state automatically. Instead two options were shown after the connection was restored: resend or cancel the message. It also turned out that sending multimedia messages while offline was not possible at all. An error message was shown when trying to send a multimedia message. The results of the experiments also showed that the Riot.im application creates a temporary message events, which is a little bit different than message events created by the Matrix server (Fig. 2).

The most relevant difference is the way a temporary event identifier (*event_id*) is created. This is done by combining the room identifier and the transaction identifier. The temporary event identifier starts with a tilde (~) followed by an exclamation mark of the room identifier.

The temporary message event has also an *"origin_server_ts"* key–value pair. The value stored in this key–value pair turned out to be the time the temporary message event was created. This means that the date and time of the computer is used to determine the timestamp. The value of the *"origin_server_ts"* key–value pair in the final message event which is created by the Matrix server turned out to be the date and time of the moment the Matrix server created the message event. This means the final message event contains the timestamp of the moment the message was sent to the Matrix server.


```
{
  "type": "m.room.message",
  "content": {
    "msgtype": "m.text",
    "body": "test"
  },
  "event_id": "~!ebYlkYiBNdbfSNCQgl:matrix.org:m1596398240724.0",
  "user_id": "@user010101:matrix.org",
  "sender": "@user010101:matrix.org",
  "room_id": "!ebYlkYiBNdbfSNCQgl:matrix.org",
  "origin_server_ts": 1596398240724
}
{
  "content": {
    "body": "test",
    "msgtype": "m.text"
  },
  "origin_server_ts": 1596399135204,
  "sender": "@user010101:matrix.org",
  "type": "m.room.message",
  "unsigned": {
    "age": 121,
    "transaction_id": "m1596398240724.0"
  },
  "event_id": "$1hFZ7QdjE2wN4PR6Z4DlZvUMBbgegDkjlAMf6LvjX5Q",
  "room_id": "!ebYlkYiBNdbfSNCQgl:matrix.org"
}
```


Fig. 2. Message events: temporally (a) and after sending (b).

4.3.4. Unsigned "age" key–value pair

The timestamp in this key–value pair can be used to determine the age of the message event. During the experiments with 6 sending messages, it turned out that the value in this key–value pair is updated by the Matrix server when a message event is downloaded by the Riot.im client. By combining the timestamp in the *"origin_server_ts"* key–value pair and the timestamp in the *"age"* key–value pair, the date and time can be determined of the moment the message event was downloaded by the Riot.im application (Table 2). The experiments also showed that the date and time of the computer on which the Riot.im application runs doesn't affect the value in the *"age"* key–value pair (Table 3).

4.3.5. "transaction_id" key–value pair

This transaction identifier is used by the Matrix server to uniquely identify each message that is send from a client. This is necessary, because the Matrix server generated the event identifier when a message event is created. This key–value pair is then added to the "unsigned" object of the message event copy of the client that was used to send the message. This key–value pair is not visible for other participants in the room.

We also found that the transaction identifier generated by the Riot.im application contains a timestamp and the date and time of the client's computer influences this timestamp. This means that if the date and time of the computer is wrong, the timestamp in the transaction identifier is also wrong. Based on these findings, the timestamp in the transaction identifier can be used to tell something about the date and time of the computer at a certain moment. This is under the assumption that the date and time of the server was correct when the *"origin_server_ts"* key–value pair was generated.

This *"transaction_id"* key–value pair could also be used to determine which messages were sent by using the Riot.im application that is being investigated because only the client who sent a message has this key–value pair.

When working with the *"transaction_id"* key–value pair digital investigators have to keep in mind however, that the *"transaction_id"* key–value pair is part of the *"unsigned"* object and as such not part of the signing process. This means that the value can be changed after creating the message event.

4.4. Riot.im and locally stored data

The Riot.im application is installed at the location <user>\AppData\Local\riot-desktop\ and stores data in <user>\AppData\Roaming\Riot\, where<user> is the user's directory e.g. C:\Users\User01.

One LevelDB database at the location: C:\Users\User01\AppData\Roaming\Riot\IndexedD\ vector_vector_0.indexeddb.leveldb. This LevelDB is used to store all message events, both unencrypted and encrypted message events. It also contains a lot of log information about the Riot.im application. The LevelDB consists of multiple files in a folder e.g. the *".log"* and files with the *".ldb"* extension turned out to be used to store information.

It seems that the ".log" file is used as a temporary file to store

Table 2
Value of *origin_server_ts* + *age* in Scenario 2 (cf. 3.4).

	origin_server_ts + *age* value	Human readable time	Real time received PC-01
1	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04
2	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04
3	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04
4	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04
5	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04
6	1596485050594	3 Aug. 2020 22:04:10	3 Aug. 2020 22:04

Table 3
Age vs. *origin_server_ts* + *age* in Scenario 3 (cf. 3.4).

	age	*origin_server_ts* + *age* value	Human readable time	Time received PC-01
1	2085	1596486785273	3 Aug. 2020 22:33:05	30 Jul. 2020 22:33
2	1609	1596486915518	3 Aug. 2020 22:35:15	30 Jul. 2020 22:35
3	130	1596487027721	3 Aug. 2020 22:37:07	30 Jul. 2020 22:37
4	955	1596486847205	3 Aug. 2020 22:34:07	30 Jul. 2020 22:34
5	326	1596486986509	3 Aug. 2020 22:36:26	30 Jul. 2020 22:36
6	474	1596487082620	3 Aug. 2020 22:38:02	30 Jul. 2020 22:38

information that is later stored in a file with the ".ldb" extension. Using the "*leveldb-tools*" (GitHub, 2020a) in our experiments, all messages sent in the unencrypted room where recovered using this method, including the message events of multimedia messages e.g. a Riot.im log-record (Fig. 3) consists of date and time, URL of the Matrix server.

Another log-record found was a record that logs the user name of the logged in user name. In this log-record the *"mxid"* key is used to identify the user name. It also shows the device identifier of the device used and again the Matrix server that is used to logon. The *"deviceId"* key is used to identify the device identifier and the *"hs"* key is used to identify the Matrix server URL (Fig. 4). This information can be relevant during investigations, because it can be used to determine the user name which was used to logon, the devices device identifier and the Matrix server that was used.

To be able for *Riot.im* to encrypt messages, which are sent in an encrypted room, *Riot.im* and the Matrix protocol use key synchronization protocols to share session data between the clients of participants in a room. Each participant of an encrypted room has copies of the session keys of all other participants in that room. A log entry in LevelDB handles the synchronization of these session keys and this log entry starts with the following text: "*I got device keys for @[username]:matrix.org:*". This log entry contains different keys and signatures of the device the keys belong to. To be able to connect the keys to the correct device and eventually to be able to decrypt the sent message from that device, the record also contains information about the device. The log records turned out to contain the following relevant information related to the user and the device: unique device identifier; full username including the server the user belongs to; the device display name (the device description). Fig. 5 shows the user information extracted from these log records in our experiments.

As the LevelDB only stores the encrypted version of message events, it is not possible to search in these messages. This is why the developers of Riot.im decided to create a second data store, which is an encrypted SQLite database called "events.db". This database is stored at the following location: <user> \AppData\Roaming\Riot\EventStore/. Because the integration of this database is still in development, the password of the database is stored in plain text in the source code of the Riot.im application (GitHub, 2020b; "GitHub - vector-im/riot-, 2020). By using this password, we were able to open the database and analyze its contents. The "events.db" database consist of nine tables, in which the *"event"*, *"room"* and *"profile"* tables are the most relevant from a digital forensic perspective. The other tables turned out to be mostly empty and not relevant. The *"event"* table is used to store unencrypted parts of the encrypted message events (Fig. 6). The *"rooms"* table contains the unique room identifier. This is the room identifier that is also stored in the key–value pair called *"room_id"* of the events. Each record in the table also has an identifier, which is unique in the *"rooms"* table.

The *"profile"* table contains information about users. This table contains four columns, namely *"id"*, *"user_id"*, *"displayname"* and *"avatar_url"*. The first column, *"id"*, contains a unique identifier for each record in the table. This identifier is an ascending number starting at one. The *"user_id"* column contains the full username, for instance *@user010101:matrix.org*. The *"displayname"* column contains the display name of the user and the *"avatar_url"* contains a URL to the avatar of the user, if any is configured.

Because the Riot.im uses Electron as a wrapper and Electron uses the Chromium engine, some of the directories are the same as the file structure of the Chrome web browser. So does the cache directory. The contents of this directory can be made visible by using ChromeCache viewer applications. Thumbnails and in some cases complete images can be recovered from this directory. Cache file (Boucher and Le-Khac, 2018) are set to be deleted after 24 h after they were downloaded.

Another interesting file*"preferences"* is stored in the root of the Riot.im data directory. This file has no extension, but can be opened with a normal text editor. This file contains the following artifacts: the default download directory; the last directory that was used to upload a file; the spellcheck language that was used. This information can give digital investigators an idea of which directory was last used to upload files and in which directory the downloaded files can be found.

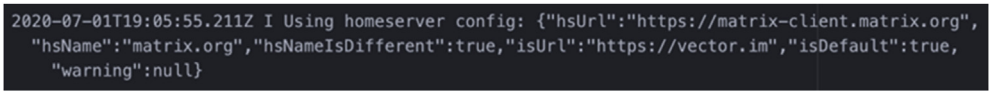

Fig. 3. Used Matrix server.

```
2020-07-01T19:05:55.242Z I setLoggedIn: mxid: @user010101:matrix.org deviceId: RGTIXTWYGO
  guest: false hs: https://matrix-client.matrix.org softLogout: false
```

Fig. 4. Logged in user and device identifier.

Date	Time	Full username	Short username	Server domain	Device identifier	Device name
01/07/2020	19:25:38.352Z	@user020202:matrix.org:	@user020202	matrix.org	RXDSIOBLZN	PC-02
01/07/2020	19:25:47.840Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	PC-01
01/07/2020	19:27:02.249Z	@user020202:matrix.org:	@user020202	matrix.org	RXDSIOBLZN	PC-02
01/07/2020	19:27:42.324Z	@user020202:matrix.org:	@user020202	matrix.org	RXDSIOBLZN	PC-02
01/07/2020	19:28:22.745Z	@user030303:matrix.org:	@user030303	matrix.org	MXBVJNQHSN	PC-01-Web
01/07/2020	19:28:31.065Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	PC-01
01/07/2020	19:28:32.077Z	@user030303:matrix.org:	@user030303	matrix.org	MXBVJNQHSN	PC-01-Web
01/07/2020	19:29:10.125Z	@user030303:matrix.org:	@user030303	matrix.org	MXBVJNQHSN	PC-01-Web
01/07/2020	19:29:51.531Z	@user030303:matrix.org:	@user030303	matrix.org	MXBVJNQHSN	PC-01-Web
24/06/2020	11:38:12.372Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
24/06/2020	11:40:01.539Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
24/06/2020	12:15:24.052Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
24/06/2020	12:16:33.547Z	@user020202:matrix.org:	@user020202	matrix.org	RXDSIOBLZN	PC-02
26/06/2020	09:36:34.985Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
26/06/2020	09:37:09.826Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
26/06/2020	09:39:38.828Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
26/06/2020	09:42:16.701Z	@user010101:matrix.org:	@user010101	matrix.org	RGTIXTWYGO	Riot Desktop (Windows)
26/06/2020	09:42:16.710Z	@user020202:matrix.org:	@user020202	matrix.org	RXDSIOBLZN	PC-02

Fig. 5. User information extracted.


```
1  {"algorithm":"m.megolm.v1.aes-sha2","content":{"body":
   "TextMessage012","msgtype":"m.text"},"curve25519Key":
   "I5HMtZAGZGGyyyKbc+espNXzWX1htj/CN/aE1NUX5gM","ed25519Key":
   "491Y9exb3c1x67PozLWdxBrrdtqVB0Pdah5BYuE5ujU","event_id":
   "$LZEwDfBKEcGgClcmuo0r5xrMWAFpXdDlLKURHdMRLUc",
   "forwardingCurve25519KeyChain":[],"origin_server_ts":1593628481972,
   "room_id":"!rasnbltJOyNTTHFbCq:matrix.org","sender":
   "@user010101:matrix.org","type":"m.room.message","unsigned":{"age":
   420483863}}
```


Fig. 6. Unencrypted parts of an encrypted event.

5. Conclusion and future work

The objective of this paper is to fill the existing knowledge gap that exists regarding how messages are handled by the Riot.im application and Matrix protocol from a digital forensics point of view. We demonstrated there is a lot of relevant information that can be retrieved from Matrix protocol and Riot.im application under a digital forensics' perspective. There is for example information about the user that did send the message and on which server that user is registered. The Riot.im application itself offers a lot of functionalities that can be used during investigations to for example look at the messages and their meta-information in the form of extended JSON message events.

During this research we found that there is a lot more very interesting information regarding state events, which are used to manage room states for example and the forensic analysis of them is left for future work. Besides, the new research has to be conducted to find new approaches to access the contents of the database in the case of the encryption.

References

Mar. 2018 Ababneh, A., Awwad, M.A., Al-Saleh, M.I., 2017. IMO forensics in Android and windows systems. In: 2017 8th International Conference on Information, Intelligence, Systems and Applications, 2018-Janua. IISA, pp. 1–6. https://doi.org/10.1109/IISA.2017.8316377.

Alyahya, T., Kausar, F., 2017. Snapchat analysis to discover digital forensic artifacts on android smartphone. Procedia Comput. Sci. 109, 1035–1040. https://doi.org/10.1016/j.procs.2017.05.421.

Anglano, C., Canonico, M., Guazzone, M., 2016. Forensic analysis of the ChatSecure instant messaging application on android smartphones. Digit. Invest. 19, 44–59. https://doi.org/10.1016/j.diin.2016.10.001.

Boucher, J., Le-Khac, N.-A., 2018. Forensic Framework to identify local vs synced artefacts. Digit. Invest. 24 (1), 2018. https://doi.org/10.1016/j.diin.2018.01.009.

BWI, May, 2020. IT für Deutschland: Artikel. https://www.bwi.de/news-blog/news/artikel/kommunikation-in-covid-19-zeiten-bundeswehr-setzt-instant-messaging-ein.

Cents, R., Le-Khac, N.-A., 202019. Towards A new approach to identify WhatsApp messages. In: 19th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom-20), Guangzhou, China, December, 2020.

Client-Server API. https://matrix.org/docs/spec/client_ server/r0.6.1 accessed Jun, 2020.

GitHub. rchunping/leveldb-tools: import, export or repair leveldb. https://github.com/rchunping/leveldb-tools accessed Jul., 2020.

GitHub. Encrypted seshat by poljar · Pull Request #15 · matrix-org/seshat. https://github.com/matrix-org/seshat/pull/15 accessed Jul., 2020.

Gregorio, J., Gardel, A., Alarcos, B., 2017. Forensic analysis of Telegram messenger for windows phone. Digit. Invest. 22, 88–106. https://doi.org/10.1016/j.diin.2017.07.004.

Hendriks, F., Mennink, B.J.M., Dobraunig, C.E., 2020. Analysis of Key Management in Matrix. BSc Thesis. Radboud University.

Irc, M., Matrix, M., 2020. Moznet IRC Is Dead; Long Live Mozilla Matrix!. https://matrix.org/blog/2020/03/03/moznet-irc-is-dead-long-live-mozilla-matrix.

Jacob, F., Grashöfer, J., Hartenstein, H., 2019. A glimpse of the matrix: scalability issues of a new message-oriented data synchronization middleware. In: 20th Int. Middlew. Conf. Demos Posters, Part Middlew. 2019, pp. 5–6.

Majeed, A., Zia, H., Imran, R., Saleem, S., 2016. Forensic analysis of three social media apps in windows 10. In: 2015 12th Int. Conf. High-Capacity Opt. Networks Enabling/Emerging Technol. HONET-ICT 2015.

Makkonen, T., 2019. Implementing Encryption for Qt-Based Matrix Client [Online]. Available: https://www.theseus.fi/bitstream/handle/10024/264169/makkonen_teemu.pdf?sequence=2. Accessed May, 2020).

Matrixorg. Introducing matrix 1.0 and the Matrix.org foundation. https://matrix.org/blog/2019/06/11/introducing-matrix-1-0-and-the-matrix-org-foundation.

Matrixorg. https://matrix.org/. Accessed May, 2020.

Matrixorg, May, 2020. Cross-signing and End-To-End Encryption by Default Is HERE!!!. https://matrix.org/blog/2020/05/06/cross-signing-and-end-to-end-encryption-by-default-is-here.

"Pro-ISIS Tech Group Announces Weekly Q&A To Be Held On Encrypted Decentralized Chat App 'Riot.im' For ISIS Supporters | MEMRI." https://www.memri.org/cjlab/pro-isis-tech-group-announces-weekly-qa-to-be-held-â€žon-encrypted-decentralized-chat-app-riot-im-for-isis-â€žsupporters (accessed May, 2020).

Mujaj, A., 2017. A Comparison of Secure Messaging Protocols and Implementations. Master Thesis. University of Oslo, Spring.

Sgaras, C., Kechadi, M.T., Le-Khac, N.-A., June 2015. Forensics acquisition and analysis of instant messaging and VoIP applications. LNCS 8915, 188–199. https://doi.org/10.1007/978-3-319-20125-2_16.

Sudozai, M.A.K., Saleem, S., Buchanan, W.J., Habib, N., Zia, H., 2018. Forensics study of IMO call and chat app. Digit. Invest. 25, 5–23.

Thantilage, R., Le-Khac, N.-A., 2019. Framework for the retrieval of social media and instant messaging evidence from volatile memory. In: 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (Trustcom 2019). https://doi.org/10.1109/TrustCom/BigDataSE.2019.00070.

Thantilage, R., Le-Khac, N.-A., 2020. Retrieving E-dating application artifacts from iPhone backups. In: Advances in Digital Forensics XVI, vol. 589. Springer, Cham. https://doi.org/10.1007/978-3-030-56223-6_1.

Wijnberg, D., Le-Khac, N.-A., 2020. Identifying Interception Possibilities for WhatsApp arXiv:2011.03732. https://arxiv.org/abs/2011.03732.

GitHub. vector-im/riot-desktop: a glossy Matrix collaboration client for desktop. https://github.com/vector-im/riot-desktop accessed Jul., 2020.