INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2020, VOL. 66, NO. 3, PP. 405-410
Manuscript received February 19, 2020; revised July, 2020. DOI: 10.24425/ijet.2020.131892

# Access Logs – Underestimated Privacy Risks

Michał Glet and Kamil Kaczyński

***Abstract*—Access logs may offer service providers a lot of information about specific users. Depending on the type of the service offers, the operator is capable of obtaining the user's IP, location, communication habits, device information and so on. In this paper, we analyze a sample instant messenger service that is operating for a certain period of time. In our sandbox, we gathered enough data to correlate user communication habits with their localization, and even contacts. We show how seriously metadata may impact the user's privacy and make some recommendations about mitigating the quantity of data collected in connection with this type of services.**

***Keywords*—metadata, privacy, cybersecurity, instant messaging, access logs**

## I. Introduction

METADATA are providing information about other data. Metadata serve a number of purposes - from describing resources, to describing data structures to providing statistical information about data processing. From the point of view of user privacy, the last of the metadata types listed above has the highest value, as it may identify the purpose of data, time and date of their creation, author, location (via IP address), source and the process used for their creation.

According to [1], metadata are extensively being used by governments as a great source of information about citizens' activity. When it comes to analyzing the activity of employees, especially at government agencies, the list of data types collected is much longer. It is possible to recover information about the activity of a given user in specific databases, to analyze their search history, communication patterns, software installed, telephone bills, etc. In cooperation with companies such as Google, Twitter or Facebook, it is even possible to disclose information about personal contacts, with their email and physical addresses, phone numbers and history of their payments. Metadata contains information about the time at which the user has logged in to specific service, the time at which their online activity began, and information about the source IP address. Metadata may be also used for linking different accounts used by the same user during a specific period of time.

Instant messenger services, such as Signal or WhatsApp, are also collecting great amounts of metadata. Even if all communications are encrypted, the service provider still collects metadata about those encrypted messages and calls. Thus, the service provider will have, at their possession, data about the source and destination IP (location) of communicating parties, their phone numbers, date and time of each communication, duration of each call, device identifiers, operating system version, time of reading the message, etc. Usually, such a set of data is sufficient to identify user habits, their contacts, places they have visited, location of their home and office, and even to define the probability of meeting them physically at a specific location.

Many papers have dealt with the privacy of instant messenger service users. In [2], the authors focus on explicit policies for protecting user privacy. [3] presents privacy concerns regarding receipts confirming that messages have been read. The authors of [4] discuss the impact of Snapchat's disappearing messages, from the perspective of acceptance of social media technologies. However, no papers have been dedicated strictly to the technological aspects of collecting and analyzing access logs, and to identifying this specific type of user privacy-related risk.

In this paper, we present a model of an instant messaging service that has been implemented for the purpose of collecting user metadata. The amounts and types of data collected do not differ from the actual practices relied upon by current market leaders. This study focuses on the technical aspect of the data analysis process, i.e. on the collection of data, their correlations, and on obtaining extra information, such as user location - all with the assistance of a dedicated analytical app. The results gathered and their implications are discussed as well, thus offering the service providers an opportunity to implement changes required to guarantee the privacy of their service users.

## II. Instant Messaging Service Model

In order to collect sufficient amounts of metadata, a toy version of an instant messaging service had to be created. In the following chapters of this paper, the term TIMS (Toy Instant Messaging Service) will be used to identify the service and the apps that have been created. TIMS specification is similar to that of such services as WhatsApp [5] or Viber [6], namely:

- Users are identified based on their phone number;
- All communications are end-to-end encrypted;
- Users are sending their messages via a central server;
- Users are making VoIP calls – this process requires that metadata be sent to the central server;
- The central server is responsible for registering users and for sending push notifications.

The network communication process in TIMS is shown in Fig. 1. The central server is storing user credentials required for authentication (FCM/APN push tokens) and is responsible for delivering messages between users. VoIP calls may be established in two modes – peer-to-peer or via a proxy (central server). Regardless of which type of communication is chosen, the server has to send, to the users, push notifications with the

This work was presented at the International Scientific Conference Mathematical Cryptology & Cybersecurity (MC&C 2020), Warsaw, 16-17.01.2020.

Michał Glet and Kamil Kaczyński are with Military University of Technology, Faculty of Cybernetics, Institute of Mathematics and Cryptology (e-mail: michal.glet@wat.edu.pl, kamil.kaczynski@wat.edu.pl).

call's metadata – such as caller ID, IP address and communication port. If communications are based on the proxy mode, then all encrypted voice packets exchanged between users are sent through the proxy server.

The TIMS server collects access logs and other metadata, containing information about the following:

- Sending message from user A to user B;
- Making a voice call between user A and B and distinguishing between the caller and the recipient;
- Getting message from the server;
- Registration events;
- User IP address;
- Users' phone numbers;
- Date and time of activity;
- User's time zone.

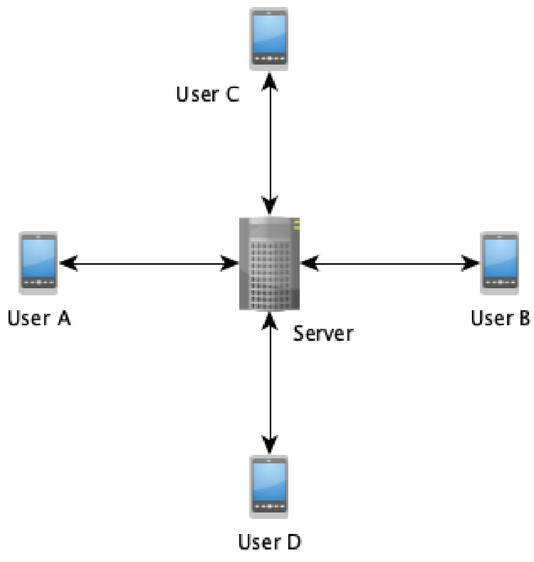

Fig. 1. TIMS network communication diagram

TIMS relies on a mobile app for smartphones – in our case for Android and iOS devices - and on a server application for controlling the registration process, delivering push notifications, exchanging messages and proxying or informing the recipient about an incoming voice call. The server application has no access to the content of messages and to voice packets – these are encrypted at smartphone level with keys that are not known to the server. The server application creates logs with every activity performed. The TIMS server has no access to those logs created at mobile endpoints.

## III. DATA ANALYSIS METHODS

TIMS uses a *nginx* proxy server to distribute user requests between different services (e.g. messaging service or phone service). Nginx is also responsible for managing TLS sessions. All logs analyzed are standard access logs created by the nginx server. Access logs were turned on in the nginx.conf file, by the following line of code:

*access_log /var/log/nginx-access.log*

Logs were created in the nginx-access.log file. An application written in Java and a PostgreSQL database have been relied upon to analyze those logs and to perform research-related activities.

The main purposes of the Java application were as follows:

- Parsing nginx access logs files.
- Adding processed data to the PostgreSQL database.
- Retrieving location information based on IP addresses.
- Updating location data in the PostgreSQL database.

The main purposes of the PostgreSQL database were as follows:

- Storing the processed access log data.
- Performing analysis queries to process access log data.

The data analysis process consisted of the following steps:

1. Collecting access logs (nginx server).
2. Parsing access logs and adding them, as records, to the database (Java application).
3. Processing access log data using SQL queries (PostgreSQL database).
4. Updating location information with access log processing results (Java application).

### A. *Nginx access logs*

In order to show how sensitive information may be reconstructed from plain access logs, we have used the standard nginx access log format, without any extraordinary tweaks introduced for our research purposes. The default access log format used by nginx consists of the following:

*$remote_addr - $remote_user [$time_local] "$request" $status $body_bytes_sent "$http_referer" "$http_user_agent"*

where:

- *$remote_addr* is the IP address of a TIMS user,
- *$remote_user* is a user identification number (e.g. user login in systems that rely on logins and passwords for authentication purposes),
- *$time_local* is information about the time and date at which the request was performed,
- *$request* is combined information about the request protocol, request method and requested URI,
- *$status* is a status of the response to the user's request,
- *$body_bytes_sent* is a number of bytes sent to the user in response to the processed request,
- *$http_referer* is information about the request referrer,
- *$http_user_agent* is information about the user agent that made the request (about the operating system, Internet browser, etc.)

The sample access log records that have been analyzed analysis had the following form:

- Record 1: *31.179.57.255 – USER_A [10/Feb/2019:17:56:12 +0100] "PUT /tims/tokens/tokens HTTP/1.1" 200 894 "-" "TIMS/1.15 (iPhone; iOS 12.1.2; Scale/3.00)"*
- Record 2: *37.47.24.167 – USER_B [23/Jun/2019:23:31:30 +0200] "GET /tims/accounts/info/ HTTP/1.1" 200 49 "-" "Dalvik/2.1.0 (Linux; U; Android 8.0.0; SM-A600FN Build/R16NW)"*
- Record 3: *89.66.210.147 – USER_C [20/Nov/2019:17:11:17 +0100] "DELETE /v1/messages/USER_C/1575216676287 HTTP/1.1" 204 0 "-" "okhttp/3.6.0"*

Table I shows the results of parsing sample access log records in accordance with the default nginx access log format.

TABLE I
PARSED LOG RECORDS

| Data | Record 1 | Record 2 |
|---|---|---|
| $remote_addr | 31.179.57.255 | 37.47.24.167 |
| $remote_user | USER_A | USER_B |
| $time_local | [10/Feb/2019:17:56:1 2 +0100] | [23/Jun/2019:23:31:3 +0200] |
| $request | PUT /tims/tokens/tokens HTTP/1.1 | GET /tims/accounts/info/ HTTP/1.1 |
| $status | 200 | 200 |
| $body_bytes_sent | 894 | 49 |
| $http_referer | - | - |
| $http_user_agent | TIMS/1.15 (iPhone; iOS 12.1.2; Scale/3.00) | Dalvik/2.1.0 (Linux; U; Android 8.0.0; SM A600FN Build/R16NW) |

## B. *Java application*

A Java application, known as *AccessLogsAnalyzer,* was written with Java 1.8 SDK. It uses three external dependencies:

- com.google.code.gson:gson:2.8.5 – library for JSON serialization and deserialization to/from Java Objects,
- io.ipgeolocation:ipgeolocation:1.0.11 – library with SDK for location service,
- org.postgresql:postgresql:42.2.8 – JDBC driver for PostgreSQL.

*AccessLogsAnalyzer* consisted of three classes:

- *pl.edu.wat.accesslogsanalyzer.AccessLogsAnalyzer*
- *pl.edu.wat.accesslogsanalyzer, AccessLogsAnalyzerAddLocalization,*
- *pl.edu.wat.accesslogsanalyzer.bean.LogData.*

The *pl.edu.wat.accesslogsanalyzer.bean.LogData* class was a standard java bean class (POJO). It was responsible for parsing and storing the access log record data in object style. the main part of class code is presented below:

```
public static String INSERT_SQL =
        "INSERT INTO access_log(" +
                "ip," +
                 ……
                "agent) "
        + "VALUES(?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)";

// create LogData object from line with one access log record
public static LogData parseLine(String line) throws Exception
{
    LogData logData = new LogData();
        ……
    String datas[] = line.split("\"");
        ……
    logData.setIp(datas1[0]);
            ……
    logData.setAgent(datas[5]);
        ……
    return logData;
}
```

The *pl.edu.wat.accesslogsanalyzer.AccessLogsAnalyzer* class offered the main entry point for a program running in the parsing mode. In this mode, the application read data from the access log file, line by line. For each line that was read, the application attempted to create a *pl.edu.wat.accesslogsanalyzer.bean.*LogData object. Then, this object was inserted into the PostgreSQL database. The main part of class code is presented below:

```
pstmt =
connection.prepareStatement(LogData.INSERT_SQL);
    ……
LogData logData = LogData.parseLine(line);
    ……
pstmt.setString(1, logData.getIp());
    ……
pstmt.setString(15, logData.getAgent());
    ……
pstmt.executeBatch();
```

The *pl.edu.wat.accesslogsanalyzer.AccessLogsAnalyzer AddLocalization* code offered the main entry point for a program updating the location mode. In this mode, the application read data from the PostgreSQL database. For each database record that was read, the application attempted to retrieve location information based on the IP address. Then, the database record was updated and stored in the PostgreSQL database. The main part of class code is presented below:

```
// update
pstmt = connection.prepareStatement("update " + tableName
+ " set localization = ? where id = ?");
    ……
// select
rs = st.executeQuery("select id, ip from " + tableName + "
where localization is NULL");
while (rs.next())
{
        ……
    String ip = rs.getString(2);
    String localization = getLocalization(ip);

    pstmt.setString(1, localization);
        ……
    pstmt.executeUpdate();
}
```

## C. *PostgreSQL database*

For analysis purposes, the parsed access log data were stored in the *access_log* table of the PostgreSQL database. Definition of the *access_log* table is presented below:

```
create table if not exists access_log
(
  id serial not null
    constraint access_log_pk
      primary key,
  ip varchar(255),
  unused1 varchar(255),
  userid varchar(8192),
  datestring varchar(1024),
  date timestamp,
  requestdatastring varchar(8192),
  requestdatamethod varchar(1024),
  requestdatauri varchar(8192),
  requestdataprotocol varchar(1024),
  responsecodestring varchar(255),
  responsecode integer,
  resposebytesstring varchar(255),
  resposebytes integer,
  unused2 varchar(255),
  agent varchar(8192)
);
```

*D. SQL queries*

The access log analysis was carried out with SQL queries executed by the PostgreSQL database engine. Three SQL queries had been created for the purpose of the analysis:

- *ip_mettings*,
- *ip_network_count*,
- *ip_network_list*.

The *ip_mettings* query was used to identify all users with the same IP addresses (three first IP v4 sub-groups) reported within this same period of time (limited to 30 minutes). The SQL query is presented below:

```
-- query ip_mettings
select distinct sub.a1_user, sub.a2_user, sub.ip, sub.date
from
(
select a1.userid as "a1_user",
       a2.userid as "a2_user",
       regexp_replace(a1.ip, '\.\d+$', '.0') as "ip",
       a1.date as "a1_date",
       a2.date as "a2_date",
       DATE(a1.date) as date
from access_log a1 inner join access_log a2 on
regexp_replace(a1.ip, '\.\d+$', '.0') = regexp_replace(a2.ip,
'\.\d+$', '.0')
where a1.userid in
     (
      'USER_A',
      ……
      'USER_G'
     )
and a2.userid in
     (
      'USER_A',
      ……
      'USER_G'
     )
and a1.ip != TIMS_SERVER_IP and a1.userid != a2.userid
and a1.date >= a2.date and a1.date - a2.date < '30 minutes'
   ) as sub
order by sub.date asc;
```

Results of the *ip_mettings* query were stored in the *ip_mettings* database table. The definition of the *ip_mettings* table is presented below:

```
create table if not exists ip_mettings
(
  id serial not null
    constraint ip_mettings_pk
      primary key,
  user1 varchar(255),
  user2 varchar(255),
  ip varchar(255),
  date date,
  localization varchar(8192)
);
```

The *ip_network_count* query was used to find locations that were most frequently visited by a given user. The SQL query is presented below:

```
-- query ip_network_count
select count(sub.ip), sub.ip
from
(
select regexp_replace(a1.ip, '\.\d+$', '.0') as ip,
       DATE(a1.date) as date
from access_log a1
where a1.userid = 'USER A'
and a1.ip != TIMS_SERVER_IP
order by a1.date asc
   ) as sub
group by sub.ip
order by count(sub.ip) desc;
```

Results of the *ip_network_count* query were stored in the *ip_network_count* database table. Definition of the *ip_network_count* table is presented below:

```
create table if not exists ip_network_count
(
  id serial not null
    constraint ip_network_count_pk
      primary key,
  ip varchar(255),
  count integer,
  localization varchar(8192)
);
```

The *ip_network_list* query was used to find locations that were most frequently visited by a given user. The SQL query is presented below:

```
-- query ip_network_list
select distinct sub.ip, sub.date
from
(
select regexp_replace(a1.ip, '\.\d+$', '.0') as ip,
       DATE(a1.date) as date
from access_log a1
where a1.userid = 'USER A'
and a1.ip != TIMS_SERVER_IP
   ) as sub
order by sub.date asc;
```

Results of the *ip_network_list* query were stored in the *ip_network_list* database table. Definition of the *ip_network_list* table is presented below:

```
create table if not exists ip_network_list
(
  id serial not null
    constraint ip_network_list_pk
      primary key,
  ip varchar(255),
  date date,
  localization varchar(8192)
);
```

### E. *Execution environment*

All analyses were carried out on a MacBook Pro (2017 model with an Intel Core i5 3.1GHz processor and 8GB RAM) running macOS Mojave 10.14.6. IntelliJ IDEA CE 2019.3, DataGrip 2019.2.6, JDK 1.8.0_161 and PostgreSQL 10.2 were used. The *AccessLogsAnalyzer* application was lunched directly from IntelliJ IDEA CE. Database tables and SQL queries were written and executed in DataGrip. *ip_mettings* was the longest running query, taking over 1 hour and 12 minutes to complete. The remaining queries took no longer than 2 minutes to complete.

## IV. RESULTS

### A. *Data collected*

The TIMS service was deployed and used by a limited number of users – the authors of the paper and their family members. All data was collected between 12 February 2019 and 20 November 2019. For the purpose of this paper, the real phone numbers of the users will not be revealed – we will use such terms as User A (or USER_A), User B (or USER_B), etc.

During the period of time referred to above, we managed to collect 372,340 access log records. According to Table II, there were 209,498 access logs for USER_A, 60,850 access logs for USER_B, 56,716 access logs for USER_C, 25,904 access logs for USER_D, 8,079 access logs for USER_E, 6,267 access logs for USER_F, and 5,026 access logs for USER_G. Each log was related to a voice call made/received or a text message sent/received. From each log, we were able to get the following:

- Date and time of event;
- Identifiers of the communicating users;
- Type of event (sending/receiving text message, making/receiving voice call);
- User IP address;
- Device metadata – OS version.

TABLE II
NUMBER OF LOG ENTRIES COLLECTED

| User | Access log records |
|---|---|
| USER_A | 209,498 |
| USER_B | 60,850 |
| USER_C | 56,716 |
| USER_D | 25,904 |
| USER_E | 8,079 |
| USER_F | 6,267 |
| USER_G | 5,026 |

The data were originally stored in nginx access log files. Therefore, for further analysis, it was necessary to parse the data into the database. Parsing and analysis processes were performed with the use of tools and procedures described in section III. All steps of the analysis process were performed with the use of data from the database created.

The collected IP addresses were geolocated using the ipgeolocation.io [7] API service. With that service, we were able to get location information about each IP address, including, inter alia, the following:

- Country name;
- State/province;
- City;
- District;
- Postal code;
- Latitude and longitude;

Geolocation data were added to the database, for further analysis of user habits. The geolocation data were stored in the JSON format. The location data obtained from API were analyzed to identify user habits and their location over time.

### B. *Geolocation analysis*

The geolocation data acquired from API enabled us to come up with several statistics regarding TIMS users. These included, inter alia, the following:

- Determine the most frequent location of a user over a specific period of time– data acquired by the *ip_network_count* query;
- Determine all IP addresses and locations of the users – data acquired by the *ip_network_list* query;
- Predict a probable physical meeting place and time for two or more users – data acquired by the *ip_mettings* query.

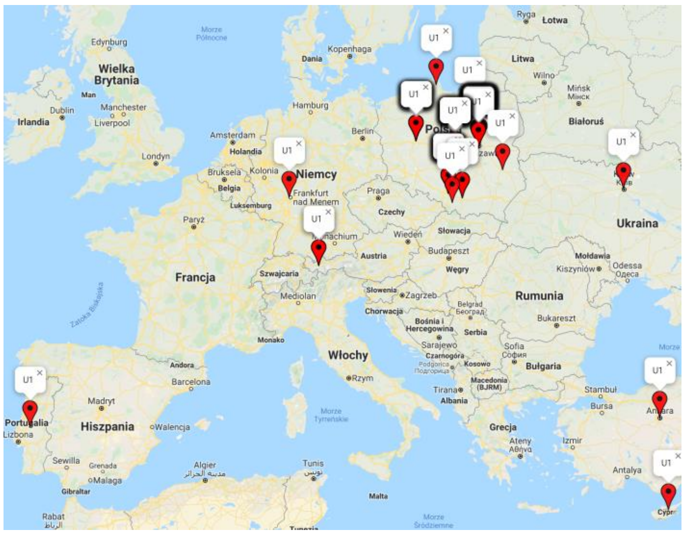

Fig. 2. Top places visited by User A

All locations were predicted based on IP data. The accuracy of that data allows us to visualize the location with a specific city district. 128 most frequently visited locations were chosen for User A. The Locations were visualized using Google Maps and they may be seen in Fig. 2. Each pin depicts one place that was visited. The map does not show the frequency of visits, but it is possible to derive that information from the database and to generate an appropriate report. No summary of all user locations has been presented in a graphic form, as it is similar to the map presenting the most frequently visited locations. It consists of much larger data set, e.g. for User A it has 1,011 entries identified during the period of the experiment.

We have also drawn up a report predicting the probable physical meeting places and times for two or more users. While creating that report, we assumed that two users were able to meet if the first 24 bits of their IP (version 4) addresses were equal and if their log times were the same (with the accuracy of 30 minutes). We identified 1,011 potential meeting locations over the entire duration of the experiment. Figure 3 presents the user meeting locations, globally. Each pin informs us about the users being present at a specific location, e.g. A B means a meeting of User A with User B.

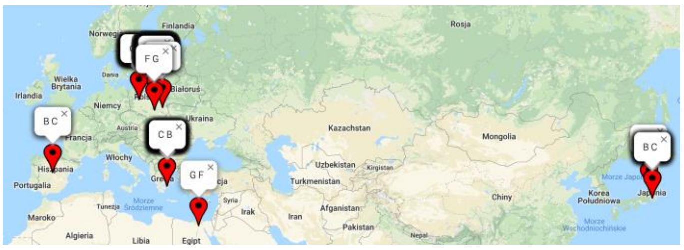

Fig. 3. Physical meetings of the TIMS users

Figure 4 presents the users' meeting locations in Warsaw. The accuracy of the API geolocation service enables us to identify user meeting locations in several city districts. More accurate data can be collected from telecom operators. Therefore, the level of accuracy may be easily increased to specific street names.

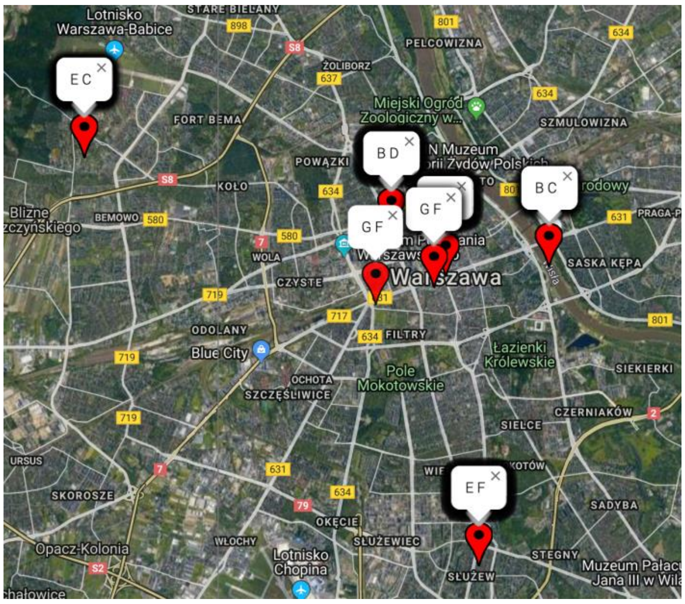

Fig. 4. Physical meetings of TIMS users in Warsaw, Poland

## C. *Result improvements and future work*

The maps presented in Fig. 2, Fig. 3 and Fig. 4 were created using geolocation data acquired from ipgeolocation.io. No information about accuracy of the service is available, but our experimental data confirms that the users were physically present at individual locations at the time in question. Some interesting results of research focusing on improving the accuracy of geolocation data are presented in [8]. Good results may also be achieved by testing other API geolocation services, especially those dedicated to the specific region of interest. Geolocation data may be also more accurate if TIMS collects some more metadata, e.g. latency of communication.

Other analyses may take into account logs generated by dedicated TIMS services, such as telephone or messaging services. This will provide additional information about user calls and messaging and will enable to identify, for instance, when and for how long selected users where talking on the phone.

A sample analysis, focusing on the users' location only, has been presented in this paper. In future work, we plan to build the users' contact network, analyze relations between users and define the shortest path between them. That type of analysis requires larger amounts of data and a larger set of users. We estimate, according to [9], that at least 10,000 users will be required. In order to collect a sufficient amount of data, the service will have to be operated for at least 6 months.

## CONCLUSION

Access logs are capable of revealing a lot of information about instant messenger users. Even small amounts of data – date, time, IP address, phone number and communication method - allowed us to identify the users' location, habits and contacts. The set of data collected by TIMS is the minimum logging level for that type of service, so one can assume that commercial services collect more data and are capable of obtaining even more accurate results in profiling their users. This seems to be a serious issue affecting the privacy of instant messenger service users. Whenever possible, they should be using applications from privacy-friendly developers who guarantee that no access logs are collected and analyzed. However, it is extremely difficult to determine what is being collected on the server side in a production environment, because the users are not able to access any production reports. They should rely on information published by privacy-focused foundations and independent journalists.

There are also other ways of ensuring privacy for instant messenger users. If they have to provide their phone number in order to register, they may provide a phone number that is different than the one they are using on a daily basis. This information is only used for the purpose of initial registration – afterwards, communications is established fully independently of that number. The privacy of location may be protected by using VPN services – especially those of the self-hosted variety. This ensures that the service provider does not know the current IP address of the user, rendering IP-based tracking impossible.

## REFERENCES

[1] M. Lee, "The metadata trap," [Online]. Available: https://theintercept.com/2019/08/04/whistleblowers-surveillance-fbi-trump/ [Accessed: 03-Dec-2019]

[2] I. K. W. Lai, G. Shi, "The impact of privacy concerns on the intention for continued use of an integrated mobile instant messaging and social network platform" *International Journal of Mobile Communications*, vol. 13(6), 2015, pp. 641–669 [*Conference*, 2016, pp. 300-307].

[3] R. Hoyle, S. Das, A. Kapadia, A. J. Lee, K. Vaniea, "Was my message read?: Privacy and Signaling on Facebook Messenger." *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (pp. 3838-3842). ACM, May 2017.

[4] D. Lemay, T. Doleck, P. Bazelais, "Passion and concern for privacy" as factors affecting snapchat use: A situated perspective on technology acceptance." *Computers in Human Behavior*, vol. *75*, 2017, pp. 264-271.

[5] WhatsApp "Frequently Asked Questions" [Online]. Available: https://faq.whatsapp.com/en/general/ [Accessed: 05-Dec-2019]

[6] Viber Support Portal [Online]. Available: https://support.viber.com [Accessed: 05-Dec-2019]

[7] IP Geolocation [Online] Available: https://ipgeolocation.io [Accessed 04-Dec-2019]

[8] O. Dan, V. Parikh, B. D. Davison, "Improving IP Geolocation using Query Logs", *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, ACM, 2016, pp. 347-356

[9] J. Lin, L. Zhang, M. He, H. Zhang, G. Liu, X. Chen, Z. Chen, "Multi-path relationship preserved social network embedding", *IEEE Access*, 2019, pp. 26507-26518