Digital Forensic Acquisition and Analysis of Discord Applications

Michał Motyliński[1], Áine MacDermott[1], Farkhund Iqbal[2], Mohammed Hussain[2], Saiqa Aleem[2]
[1]*School of Computer Science and Mathematics, Liverpool John Moores University,* Liverpool, UK
[2]*College of Technological Innovation, Zayed University,* United Arab Emirates
motylm66@gmail.com; a.m.macdermott@ljmu.ac.uk; {farkhund.iqbal, mohammed.hussain, saiqa.aleem}@zu.ac.ue

***Abstract*—Digital forensic analyses are being applied to a variety of domains as the scope and potential of digital evidence available is vast. The importance of forensic analyses of web-based devices and tools is increasing, coinciding with the rise in online criminal activity. Discord - an application that allows text, image, video, and audio communication using VoIP - has become increasingly popular and is consequently subject to increased use by cybercriminals. While researching Discord servers and forensic artefacts, it is apparent that there is limited literature and experimentation in this domain. This paper presents our research into digital forensic analyses of Discord client-side artefacts and presents DiscFor, a novel tool designed for the extraction, analysis, and presentation of Discord data in a forensically sound manner. DiscFor creates a safe copy of said data, presenting the current cache state and converting data files into a readable format.**

Keywords—Digital Forensics, Cache Analysis, Forensic Analysis, VoIP, Discord, Instant Messaging, Data Recovery.

I. INTRODUCTION

The advent of the Internet has significantly transformed the daily activities of millions of people, with one such consequence being the way people communicate where Instant Messaging (IM) and Voice over IP (VoIP) communications have become prevalent [1]. There is a plethora of IM applications available across all digital platforms and the data transmission increases each day. The advanced capabilities of digital forensic tools for analysis and presentation of potential 'evidences' available from these devices has been enhanced by the facilities of tools including AccessData Forensics Toolkit (FTK), Cellebrite UFED touch and XRY, with some comparable literature exploring these applications. Discord is an application that allows text, image, video, and audio communication using VoIP. Unlike other social media platforms Discord does not have a home news feed like Facebook or Twitter. It is built around a network of private and semi-private groups, known as "servers," which are created by mostly anonymous users [2]. In May 2019, Discord announced that it has more than 250 million registered users and over 50 million active users every month [1].

Discord has started to attract more criminals with its growing community – in no small part encouraged by the limited and unfamiliar information on recoverability of data and logs. According to Forbes [3], the FBI is investigating certain groups whose members may be involved in criminal activities, such as the 'HellsGate' server that offered thousands of different online accounts, or 'SentryMBA' server, which offered credential stuffing software that could automate processes of inputting usernames and passwords on various websites [3]. While the aforementioned servers were banned shortly after their detection, other groups remain active and their members continue to be involved in other illegal activities like spreading malware, child grooming, harassment or selling stolen goods and/or personal information. In 2019, Discord released a Transparency Report Q1 2019 [4], which addressed violations within the Discord application community in the first quarter of the year. The total number of occurrences where Community Guidelines were breached and reported by other users exceeded 50000. Considering that Discord is now one of the more popular communication platforms it is surprising that little to no academic research investigating this application is available or has been carried out. Thus, the projects focus is aimed at answering the following question: "What digital artefacts of forensic value can be recovered from the Discord application?". In our attempt to answer this question, the project simultaneously proposes a solution for the aforementioned lack of forensically sound tools for Discord data extraction. The software developed for this project - DiscFor - can retrieve information from the Discord application's local files and cache storage. Written in Python, said tool consists of a series of scripts that can be run independently or as a script using Python interpreter. The paper is organised as follows: Section II provides background on the VoIP instant messaging applications and cache in digital forensics. In Section III, we present our analysis of Discord data sources. In Section IV the design and implementation of DiscFor is presented, with our testing and analysis of results in Section V. The conclusion of our research and further work areas are discussed in Section VI.

II. BACKGROUND

A growing number of messaging applications provides criminals with additional means to perform malicious acts whilst remaining hidden, by being part of closed communities or via the use of less popular applications. Digital forensics is a branch of forensic science that tries to tackle this problem. With everyday social activities becoming more device and application centric it has become apparent that attackers can take advantage of this popularity and use digital devices and software with criminal intent. As a result, increasing cybercriminal activity has led to the emergence of new subbranches of digital forensics focused solely on research and development of tools needed for the recovery of evidence from new sources of data. "Application-specific forensics" is one of them, and as the name suggests, it is concentrated on recovery, analysis and presentation of data from various applications. Although there are many means of research and ways of recovering information, the ultimate objective of obtaining admissible evidence in court remains unchanged. While there is no standardised methodology to follow during digital investigation processes, all methods comprise of the same main stages - Secure, Analyse and Present.

978-1-7281-7315-3/20/$31.00 ©2020 IEEE

A discovery of possible forensic artefacts from multiple instant messaging applications including Facebook, WhatsApp and Skype was performed in [1] and [5]. In their study the authors provided a good overview on the various types of data that can be found within such programs, as well as the techniques used for artefacts recovery. In his work from 2011, Gao Hongtao [6] recognised the growing popularity of IM applications and their impact on number of crimes committed using VoIP. As such, researchers performed an analysis of Skype and provided a guideline for examinations of an electronic device involved in crime.

In [7], authors focused on an analysis of Google Chrome cache structure adopted by Discord. The paper provides a detailed description of cache format and offers insight into the potential data available via its explanation of storing features like data blocks and cache addresses. A cache is a temporary storage where frequently viewed content is kept, so that if the same resource is requested again it does not have to be downloaded from the server. This method provides better user experience and reduces amount of in-formation that needs to be transmitted. While cache storage can be a source of valuable information about recent activities of a suspect, the analysis of its content can be challenging due to its specific structure. Similarly, analyses of cache storage have been performed to extract YouTube and Facebook stream content. Said analyses' methodologies involved X-Ways hex editor and ChromeCacheView, the use of which allowed identification, carving and reconstruction of the video files found within the cache storage [8].

As noted previously, while there is limited literature on Discord, there are some tools that can be used for the partial recovery of application's contents. Nir Sofer's solution "ChromeCacheView" is a commonly used freeware cache viewing tool for Google Chrome web browser [9]. With its GUI it allows viewing the content of the cache storage which includes metadata and allows extraction of files found within. ChromeCacheView is available only on Microsoft Windows systems. While Nir Sofer's tool is currently the best solution for forensic analysis of the Chromium cache structure, there are certain limitations discovered during analysis of Discord data sources. Audio and video files are often stored in two separate parts which are not reconstructed by the tool.

Fire Kitty is a command line tool designed specifically for digital artefact extraction from Discord [10]. FireKitty requires a Python interpreter and an additional Python package (apsw) to be installed on the machine. The "apsw" module is a non-standard Python library providing support for SQLite databases. The program is focused on the recovery of digital evidence from a database file containing user activity logs. However, this tool does not work anymore as cache structure has been implemented in place of the mentioned SQLite file.

Discrecorder is a tool developed specifically for the extraction of Discord cache content [11]. While the tool recovers some of the files most of them are extracted without appropriate extensions, and server HTTP response corresponding with a file is also irretrievable. The method employed in the tool makes use of a "binwalk", a Python module which carves files based on their signature. Unfortunately, this method does not recover all of the files and does not recover any corresponding data that could be useful in the investigation.

Py_chromium_cache_simple was designed for the recovery of data residing in Simple Cache structure, however it does not extract all of the files [12]. Moreover, no server responses are retrieved. This tool extraction method relies on matching magic numbers of the files which can make the tool incompatible with a new format if changed. Not all stream files are being checked for presence of data; thus the tool does not provide accurate recovery means for Discord application.

The last tool, "JSON to HTML and XLS" was designed to parse chat log messages found in Discord local folders [13]. The tool is incapable of recovery and other tools must be used to extract the required files. The tool converts text data found in chat log files and does not attempt to join this information with attachments that also can be recovered from Discord directories.

The analyses of Discord data recovery tools indicates that none of the currently available solutions provides a robust and complete method of data extraction for Discord local files. In our paper we provide insight to Discord local storage internal structure but also describe our attempt to create a reliable solution for complete and accurate data recovery from the Discord application.

III. Discord App

In the literature there are no comparable approaches for analyses of Discord client-side data sources; however, there is commonality among the underlying structures. We have recognized two main sources of data, i.e., the cache storage and activity log. It was also discovered that Discord uses two different caching structures. On Windows and macOS platforms a Chromium Disk Cache was adopted, while for Linux distributions Simple Cache was implemented [14]. After installation, when a user first logs into the app with Discord credentials, the application creates a cache structure and log file which stores information about the user's recent activity. Further use of Discord leads to the generation of new cache files and log entries.

The location of the discord directory depends on the operating system, and default paths may look as follows:

- Windows - C:\Users\[username]\AppData\Roaming\
- MacOS - ~/Library/Application Support/
- Linux - /home/[username]/.config/

Within the installation directory multiple files and folders can be found that contain various data. The aforementioned cache structures are located in the folders "GPUCache", "Code Cache" and "Cache". The "GPUCache" folder contains data used to increase the performance of the application but does not retain any information about the user. The second folder stores code used by the application which also does not hold any forensically valuable information. The last directory contains data about recently viewed messages, channels, servers etc. and is considered to be one of the applications digital artefact sources. The activity log can be found in "discord/Local Storage/leveldb" where it resides among other types of files. The overview of the Discord directory on Windows 10 is shown in Fig. 1.

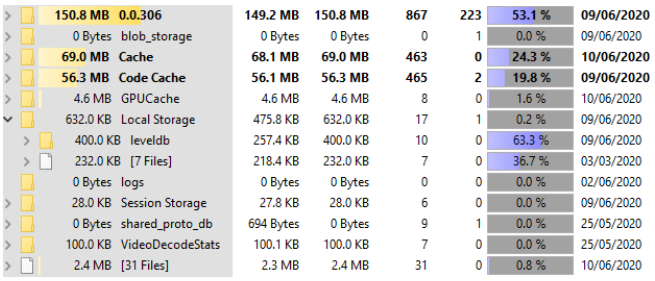

Fig. 1. Discord local directory structure with activity log location

A. *Analysis of Discord Data Sources – Disk Cache*

Disk Cache is a caching solution developed by Google as part of a Chromium Project. Through the years it has become a base for many applications and other web browsers such as Google Chrome, Brave and Opera. Disk Cache is well described in [7], where authors performed an in-depth analysis of data sources. During our study of the Disk Cache structure, we found important information on one of the files that was not covered in this work, and in our opinion is a potential source of evidence, as well as simplifies the process of evidence recovery.

The file data_0 consists of two parts; a header and a rankings table. The header contains control information about the file and the table below. The rankings table is comprised of blocks that store the address of cache entries alongside its eviction information. Each block is 36 bytes in size (Fig. 2). The rankings file contains more forensically valuable data in comparison to the index file (which only stores a list of entry addresses). The use of data_0 can significantly simplify the process of data recovery, while the eviction information provides vital information on when the file was created and accessed for the last time.

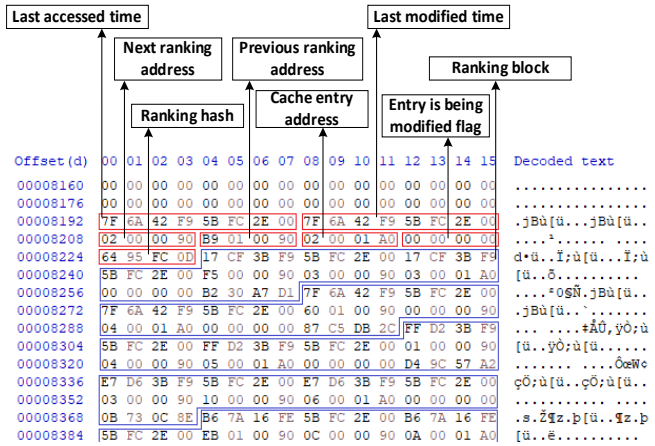

Fig. 2. Ranking block (data_0) structure

B. *Analysis of Discord Data Sources – Simple Cache*

Simple Cache is part of "The Chromium" project intended for low-resource systems and is currently used by Discord as a main cache storage for Linux and Android distributions. The Simple Cache folder contains a fake index file, cache entry files and an index-dir folder in which the real index is kept. The directory structure is shown in Fig. 3.

Fig. 3. Simple Cache directory structure

The real-index file (Fig. 4) consists of three parts: a header, entry hash table and an end of the file, called a footer. The header size is 40 bytes and contains information about the number of entries in the hash table, cache size and control
data.

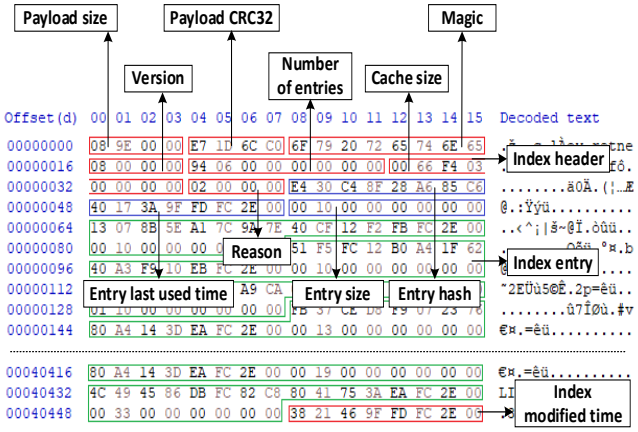

Fig. 4. Structure of the-real-index file

The hash table contains a list of entries and each entry is 24 bytes in size. The cache entry has three parts, as shown in Table 1. The very last 8 bytes of the file are reserved for saving the last modified time of the index file. While Disk Cache structure makes use of block files to store data, Simple Cache resource content and HTTP responses are stored in a single cache entry file. This format significantly simplifies a reading and writing process of cached data which also reduces the complexity of analysis and recovery algorithms. A cache entry name is created by combining a reverse entry hash address from the index file with an underscore and a stream number which can be either 0, 1 or s. The structure of the most commonly used format (stream 0) starts from file header followed by URL, resource content, EOF (End of File), server HTTP response, SHA256 of the URL (optional part) and EOF.

TABLE 1. STRUCTURE OF THE-REAL-INDEX CACHE ENTRY BLOCK

Offset	Size (bytes)	Description
0	8	Cache entry hash (cache entry file name)

8	8	Cache entry last accessed time
16	8	Cache entry size

File reading is performed from the end of the file with the use of EOF sections containing information about the data stream above it. Our analysis of stream 1 format files does not indicate an existence of potential information of forensic value here. Files #####_s keep a payload of large media or downloadable items. This file contains a list of multiple partial resource copies of the same source within which the full version is stored at the beginning of the file. The HTTP response of the resource payload is stored within #####_0 of the same name.

C. *Resource Content*

Resource content describes any file stored in cache in the form of a stream of bytes, and this includes files generated by Discord, as well as files directly uploaded by Discord users.. Generated files include JavaScript files, chat logs (generated from content posted by users) and other files that contain other application content such as emoticons, app images etc. All content uploaded by users includes text messages, images, audio, and video files as well as linked attachments. The application generates separate json files storing 50 messages of each conversation viewed by the user. If a user views more than 50 messages posted on one channel, new files are created to store additional messages (50 per file). As Discord is a messaging app the most interesting data can be found in chat logs stored in a JSON format, an example of which is shown in Fig. 5. As attachments are parts of many messages they are also stored in cache and can be recovered. While attachments are stored separately from the chat logs, they can be traced back using attachment URL which can be also found in cache entry.


```
{
  "id": "674343679355256863",
  "type": 0,
  "content": "",
  "channel_id": "631148747493212173",
  "author": {
    "id": "631086781009362954",
    "username": "forensictesterben",
    "avatar": null,
    "discriminator": "8466"
  },
  "attachments": [
    {
      "id": "674343679296798760",
      "filename": "TextFile.vb",
      "size": 63,
      "url": "https://cdn.discordapp.com/attachments/631148747493212173/674343679296798760
        /TextFile.vb",
      "proxy_url": "https://media.discordapp.net/attachments/631148747493212173
        /674343679296798760/TextFile.vb"
    }
  ],
  "embeds": [],
  "mentions": [],
  "mention_roles": [],
  "pinned": false,
  "mention_everyone": false,
  "tts": false,
  "timestamp": "2020-02-04T20:01:01.858000+00:00",
  "edited_timestamp": null,
  "flags": 0
},
```


Fig. 5. Discord chat log structure

D. *Activity Log*

The log file is comprised of sections that store recorded information about the user's recent activity on Discord. Some sections of the file cannot be decrypted with ASCII or UTF-8 but most of the information is stored in clear text. Our analysis of activity log entries shows that the log file stores lists of servers and channels that the user joined (Fig. 6).

Fig. 6. Activity log - example of recoverable data

IV. DiscFor design and implementation

The main characteristic of application-specific forensics is that it deals with large numbers of various file types and their structures. In order to read often dispersed data from proprietary formats new tools are required. Therefore, the goal of DiscFor is to automate the process of evidence collection and presentation from local Discord directories. While there are solutions available for Discord data recovery, most of them do not acquire complete data. The reason for DiscFor creation was to tackle this issue and develop a robust software solution capable of extracting all information stored on Discord local files that might be of forensic value and presenting them to the application user. In this section we present an overview of the architecture used (Fig. 7). The preservation of data is the first phase of the digital forensic investigation. One of DiscFor's options allows for the creation of a logical copy of the source material. This ensures that data is not being altered by Discord during further examination, and provides a backup of the original data for further use. Prior to backup creation the user has the option to either search a file system for the Discord directory or provide an exact path to the target folder.

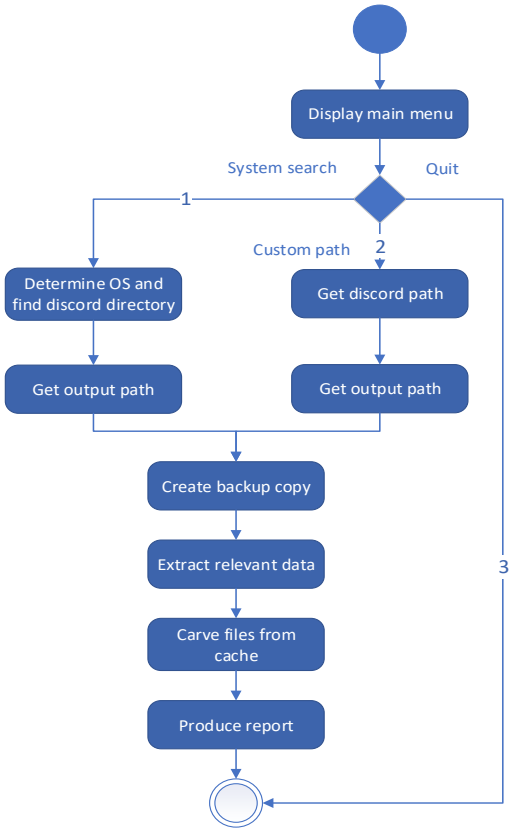

Fig. 7. DiscFor main functionalities overview

The next stage is an extraction of data that has been identified as potential evidence. We have designed three different modules in order to address all three possible sources: Disk Cache, Simple Cache and activity log. Fig. 8 shows the functionality of the module (maincache.py) responsible for the recovery of the data from Disk Cache used on Windows and macOS operating systems. The dispersion of data across multiple locations requires a lengthy process of recovery. As its name suggests, Simple Cache is less complex than Disk Cache, thus the process of evidence acquisition is easier as most of the data is stored in a single location. Recovery of data from the activity log is performed by employing a simple pattern recognition algorithm which pulls all necessary information. The last component of our tool is responsible for presentation of findings. Information on all recovered files is saved to a CSV file, the contents of which includes the file name, server HTTP response and eviction information as well as the addresses of all parts in cache files. Because Discord compresses each file uploaded by its users, an MD5 hash value is calculated for each recovered item and stored in the report. We have developed a separate module which reconstructs chat messages from chat logs and corresponding multimedia files extracted from the cache. Recreated messages are saved in HTML format and can be displayed in a web browser.

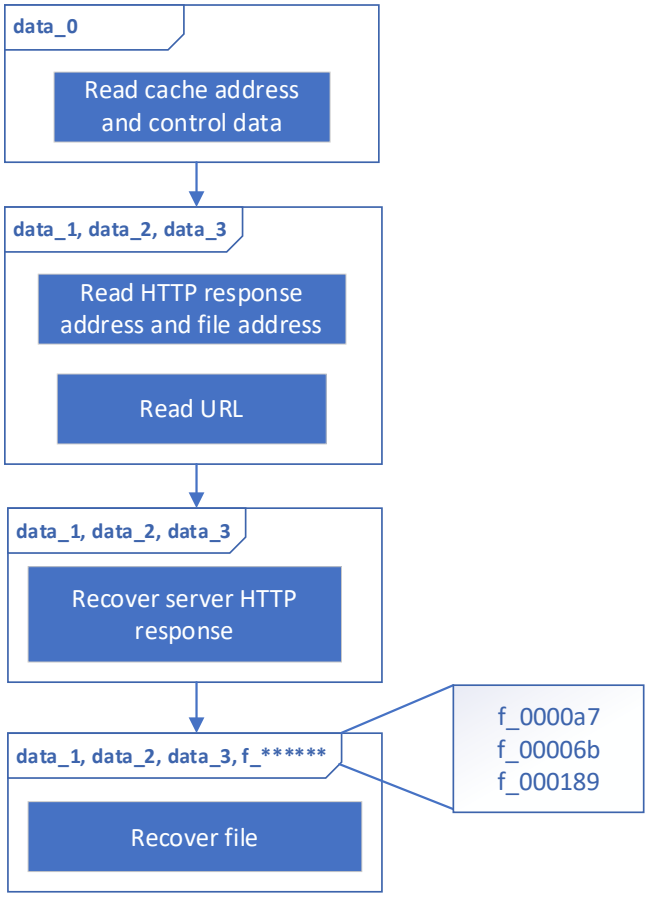

Fig. 8. Functionality graph of Disk Cache recovery script

V. Testing and Analysis of Results

For experimentation and performance testing, we created test scenarios with DiscFor, populated with a generic user account and random content exchange. Cache data recovery is the most time-consuming process of the experiments as the activity log tends to be relatively small (under 100 KB). Nevertheless, a 60 KB activity log was also used in the tests. Datasets were populated manually by joining random publicly available servers. When a user viewed Discord channels the contents of the conversations were saved to local cache. Joining multiple servers allowed the creation of datasets, with varying content such as server data, timestamps, user accounts, message digests, etc. All datasets were tested on a workstation with Windows 10 Education N installed. The results presented in Table 2 show that DiscFor recovers all possible data from both types of cache storage, i.e., Disk Cache (Windows) and Simple Cache (Linux).

Detailed information about each recovery performed with DiscFor gives the user a good look at how many files were recovered and how many were ignored. The total number of entries represents the amount of all entries in the cache structure. The valid entries column holds a number of entries that contain both server HTTP response and resource payload. Ignored entries values represent entries that were either empty or duplicate data. In addition to data recovery, DiscFor also reconstructs partial entries which mostly include audio and video files. In comparison, ChromeCacheView does not recognise valid entries as it attempts to recover empty files. FireKitty does not recover any data, while Discrecorder is able to extract some files, albeit with significant changes made to the source code. Py_chromium_cache_simple only extracts data from Simple Cache structure, and "JSON to HTML and XLS" parses data from chat logs but does not include all information that can be found in the log file. Whilst all of the tools provide different capabilities in terms of Discord data recovery, none of them can perform full, accurate extractions combined with chat log reconstruction and presentation. As ChromeCacheView is the most accurate of all the solutions offered, its performance was specifically selected and compared to DiscFor.

Table 2. DiscFor recovery results

Cache Structure	Files	Entries	Valid	Ignored
Disk Cache	1001	3108	3061	47
Disk Cache	2000	5706	5627	79
Disk Cache	3011	8328	8216	112
Simple Cache	1004	998	986	12
Simple Cache	2000	1991	1970	21
Simple Cache	3003	2994	2952	42
Simple Cache	4013	4004	3956	48
Simple Cache	5002	4990	4932	58
Simple Cache	6000	5986	5920	66
Simple Cache	7000	6979	6910	69
Simple Cache	8034	8006	7929	77
Simple Cache	9071	9036	8927	109
Simple Cache	10038	10001	9880	121
Simple Cache	11023	10997	10872	125
Simple Cache	12071	12052	11927	125
Simple Cache	13002	12993	12563	430
Simple Cache	14014	14004	13539	465
Simple Cache	15029	15014	14547	467

The extraction capabilities of our tool were tested manually to ensure that the correct data and files were recovered. DiscFor recovers 29 different types of information for every file extracted and saves them in a CSV report. Chat logs and attachments are the most valuable types of evidence that can be found in the Discord local directory. The former stores entire conversations including text, graphical content URLs, and timestamps which are vital in

determining the chronology of events. Message attachments may also provide more details about the discussion and also provide further evidence, as reported by Forbes [3]. Email addresses from the activity log are also one of the most interesting findings, while the rest of the data from the log can be used as additional/supplementary findings in cache. The summary of all Discord contents is presented in Table 3.

TABLE 3. DISCORD APP CONTENTS

Content type	Cache	Activity log
User email	No	Yes
User password	No	No
Channel ID	Yes	Yes
Server ID	Yes	Yes
Timestamps	Yes	Yes
Attachments	Yes	No
Chat logs	Yes	No
Users avatars	Yes	No
JavaScript files	Yes	No

The server HTTP response is recovered in the form of a byte string, and the most important parts are cleaned; an example of this is shown in Table 4, while a fully reconstructed message is presented in Fig. 9.

TABLE 4. TYPES OF CARVED AND CLEANED SERVER RESPONSE DATA

Data type	Example
Server response	HTTP/1.1 200
Content type	image/jpeg
Etag	W/"6f76ae9bc6a2779c8300dce5475601db"
Response time	08/03/2020 21:51
Last modified time	04/03/2020 18:13
Max age	2592000 (given in seconds)
Server name	cloudflare
Expire time	08/03/2020 21:56
Content encoding	gzip
Server IP	162.159.130.233
Time zone	GMT

Channel Id: 690238062298791986

Message Id	701650254843478086	**Time**	2020-04-20 04:27:36
Author	**Id**	686252358115786842	None
	Username	forensictesterann	
	Discriminator	4890	
Content	This is a great place!		

https://media.discordapp.net/attachments/690238062298791986/701650254386298989/1920px-Colosseum_in_Rome_Italy_-_April_2007.jpg?width=400&height=235

Fig. 9. Example of reconstructed message

As is evident from the data gathered, the performance of DiscFor is remarkable, even when analysing a large number of files. The comparative results presented in Table 5 show that there is little difference between DiscFor and ChromeCacheView in terms of number of files recovered. However, it must be noted that our test case included very small number of audio and video files which greater number would prove Sofer's solution to be much less efficient. As a result of dividing into parts (Disk Cache) or storing files in different separate files (Simple Cache) ChromeCacheView is not able to recover and reconstruct them.

TABLE 5. DISCFOR AND CHROMECACHEVIEW RECOVERY RESULTS COMPARISON

Cache Structure	Valid entries	Complete files recovered	
		DiscFor	ChromeCacheView
Disk Cache	3061	3061	3055
Disk Cache	5627	5627	5610
Disk Cache	8216	8216	8189
Simple Cache	986	986	981
Simple Cache	1970	1970	1962
Simple Cache	2952	2952	2944
Simple Cache	3956	3956	3948
Simple Cache	4932	4932	4921
Simple Cache	5920	5920	5907
Simple Cache	6910	6910	6890
Simple Cache	7929	7929	7902
Simple Cache	8927	8927	8893
Simple Cache	9880	9880	9844
Simple Cache	10872	10872	10848
Simple Cache	11927	11927	11909
Simple Cache	12563	12563	12555
Simple Cache	13539	13539	13530
Simple Cache	14547	14547	14534

Moreover, Fig. 10 shows that the DiscFor process of data recovery takes less time than its ChromeCacheView counterpart even though it recovers more information and also reconstructs the messages. The results are based on the tests performed on all 18 datasets shown in Table 2. The testing only considered the recovery of the data and creation of the report files as ChromeCacheView does not provide a backup feature. In spite of this result, it is important to mention that the performance of DiscFor recovery may be variable, depending on the hardware and software used.

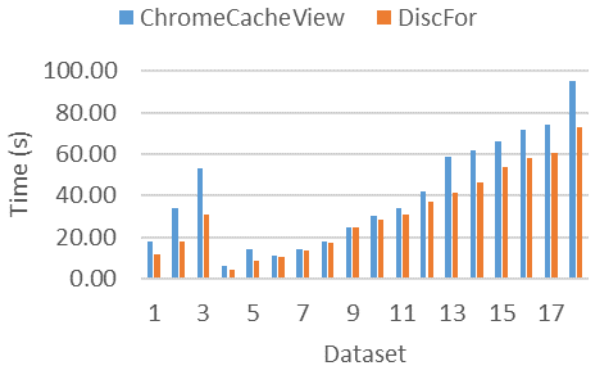

Fig. 10. DiscFor and ChromeCacheView performance comparison

Analysis of the results from the cache storage and activity log clearly indicates that the former contains much more data of a greater value from a forensic point of view. Text messages, images, audio and video files are one of the most important data types for digital investigations having a high likelihood of becoming a piece evidence. While it is possible to retrieve email addresses from the activity log it is very unlikely as this type of data will only appear in the log if the user has recently logged in or has recently switched accounts. While timestamps can be used to identify and confirm when certain message was sent or voice chat took place there is little use of server and channel ID's as they also exist in cache. On its own data from activity log cannot

be used to determine if crime or misuse occurred, thus it may serve only as a compliment to what can be recovered from the cache storage.

VI. Conclusion and Future Work

Communication applications like Discord are experiencing increasing popularity with the rise in the number of internet users worldwide. The use of IM applications offers potentially rich sources of data, which if properly recovered and preserved may become an admissible and invaluable source of evidence in forensic investigations. We have discovered that recent messages can be extracted without accessing a person's account, with the use of tools capable of decoding data stored in cache. Considering that most of the data is kept in the cache memory for 30 days prior to deletion the potential amount of evidence available to forensic investigators is vast. The main advantage of DiscFor lies in its ability to recover important data accurately and in a timely manner from all Discord sources. Moreover, DiscFor can be run either as a Python script without the need for any external modules, or as an executable file on any system which can be extremely important in a digital forensic investigation. Reporting features allow the investigator to find relevant data quickly without the need to manually view cache or JSON files.

There are a few constraints that were not overcome by DiscFor. As it was already mentioned, cached data is kept for a maximum of 30 days. To access all data an examiner must log in to the account and download messages not stored in local storage. Unfortunately, the password is not stored locally which leaves the account inaccessible if the user logged out. Because Discord locks its files in a read-only state, the application must be closed to run DiscFor. This means that if an examiner found Discord open with the user logged in, it would be better to perform a live investigation than running DiscFor. Another issue of DiscFor is that user can delete cached data and prevent extraction. The files must be removed directly from folders. Deleting messages in Discord does not cause their immediate removal from the cache storage and their removal from local files may take even several minutes. If the user closes the program before the update occurs, old information will remain in the local storage. This may also lead to interesting findings as Discord can be used on multiple platforms which means that different information may be found on various devices.

There are avenues for future work in the forensic analysis of Discord. The scope of this project was limited to the PC version of Discord, therefore mobile application and web variants of the Discord application remain open to further study. Additional research is required to determine if the activity log may hold more data of forensic value. As Discord receives constant improvements it is possible that its storage methods may change in future.

The program DiscFor was implemented as a command-line tool to be as small and simple as possible, but later development may include the implementation of GUI. DiscFors' code can be further improved to increase its efficiency. The tool was tested in a controlled environment, and further experiments are required in order to assess and verify its capabilities in live forensic investigations.

Acknowledgments

This study is supported by Research Incentive Fund (R19044 and R18055) and Research Cluster Award (R17082), Zayed University, United Arab Emirates.

References

[1] Sgaras C., Kechadi M. T., and Le-Khac N. A., "Forensics acquisition and analysis of instant messaging and VoIP applications," Lect. Notes Comput. Sci., vol. 8915, pp. 188–199, 2015.

[2] Patterson D., "Cybercriminals are doing big business in the gaming chat app Discord," CBS, 2020, last accessed 2020/04/21.

[3] Brewster T., "Discord: The $2 Billion Gamer's Paradise Coming To Terms With Data Thieves, Child Groomers And FBI Investigators," Forbes, 2020. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2019/01/29/discord-the-2-billion-gamers-paradise-coming-to-terms-with-data-thieves-child-groomers-and-fbi-investigators/#26f268ca3741.

[4] Discord, "Discord Transparency Report," Nelly, Discord Blog, 2019. [Online]. Available: https://blog.discordapp.com/discord-transparency-report-jan-1-april-1-4f288bf952c9

[5] Yusoff, M.N., Dehghantanha, A. and Mahmod, R., 2017. Forensic investigation of social media and instant messaging services in Firefox OS: Facebook, Twitter, Google+, Telegram, OpenWapp, and Line as case studies. In Contemporary Digital Forensic Investigations Of Cloud And Mobile Applications (pp. 41-62). Syngress.

[6] Hongtao G., "Forensic Method Analysis Involving VoIP Crime," 2011 Fourth International Symposium on Knowledge Acquisition and Modeling, Sanya, 2011, pp. 241-243, doi: 10.1109/KAM.2011.71.

[7] Suma G. S., Dija S. and Pillai A. T., "Forensic Analysis of Google Chrome Cache Files," 2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Coimbatore, 2017, pp. 1-5, doi: 10.1109/ICCIC.2017.8524272.

[8] Horsman, G., 2018. Reconstructing streamed video content: A case study on YouTube and Facebook Live stream content in the Chrome web browser cache. Digital Investigation, 26, pp.S30-S37.

[9] ChromeCacheView Homepage, https://www.nirsoft.net/utils/chrome_cache_view.html, last accessed 2020/06/02.

[10] FireKitty Homepage: https://github.com/kittymagician/FireKitty, last accessed 2020/06/02.

[11] Discrecorder Homepage, https://github.com/alfuananzo/discrecorder, last accessed 2020/06/05.

[12] Py_chromium_cache_simple Homepage, https://github.com/cristi8/py_chromium_cache_simple, last accessed 2020/06/05.

[13] JSON-to-HTML-and-XLS Homepage, https://github.com/abrignoni/JSON-to-HTML-and-XLS, last accessed 2020/06/05.

[14] Chromium Homepage, https://github.com/bloomberg/chromium.bb, last accessed 2020/06/09.