Discord Server Forensics: Analysis and Extraction of Digital Evidence

Farkhund Iqbal[1], Michał Motyliński[2], Áine MacDermott[2]

[1]*College of Technological Innovation, Zayed University, United Arab Emirates*
[2]*Department of Computer Science, Liverpool John Moores University, Liverpool, UK*
farkhund.iqbal@zu.ac.ae; motylm66@gmail.com; a.m.macdermott@ljmu.ac.uk

Abstract—**In recent years we can observe that digital forensics is being applied to a variety of domains as nearly any data can become valuable forensic evidence. The sheer scope of web-based investigations provides a vast amount of information. Due to a rapid increase in the number of cybercrimes the importance of application-specific forensics is greater than ever. Criminals use the application not only to communicate but also to facilitate crimes. It came to our attention that the gaming chat application Discord is one of them. Discord allows its users to send text messages as well as exchange image, video, and audio files. While Discord's community is not as large as that of the most popular messaging apps the stable growth of its userbase and recent incidents indicate that it is used by criminals. This paper presents our research into the digital forensic analysis of Discord client-side artefacts and presents experimental development of a tool for extraction, analysis, and presentation of the data from Discord application. The work then proposes a solution in form of a tool, 'DiscFor', that can retrieve information from the application's local files and cache storage.**

Keywords—digital forensics, cache analysis, discord, discord server, forensic analysis, VoIP.

I. INTRODUCTION

The evolution of the Internet and technology has changed our lives and the way we interact with each other forever. In recent Instant Messaging (IM) and Voice over IP (VoIP) communications have become prevalent and allowed us to share thoughts at low cost and with unparalleled speed [1]. There is a plethora of IM applications available across all mobile platforms and the daily transmission of data varying. The advanced capabilities of digital forensic tools for analysis and presentation of potential 'evidence' available from these devices has been enhanced by the facilities of tools such as Cellebrite UFED and MSAB XRY, with some comparable literature exploring these applications and different operating systems (OS). 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 user [2]. With a large number of potential victims' this application is an ideal space for criminals. Moreover, as it allows the creation of private servers it is easier to form closed groups and hide criminal activities. There are more servers that have not been shut down which members can be involved in other illegal activities like selling stolen goods and personal information, child grooming, harassment or spreading malware. In the Discord Transparency Report Q1 2019 [4] there were over 50000 user reports of actions violating the Community Guidelines, ranging from spamming, harassment, threatening behaviour and exploitative content. While spam is the most common reason for banning exploitative content comes second with 10642 bans. It clearly shows that the

problem exists, and it is likely to grow with the number of Discord users. More evidence of Discord related crime can be found on YouTube where many Discord users publish videos documenting their encounters with child grooming and other criminal activity. Whether these situations are real or if they are just another way of gaining viewership remains unknown but a growing number of videos with such disturbing content is alarming. Considering that Discord has now a large community it is surprising that there can be found little to no academic research investigating this platform.

Therefore, the project focuses on answering the following research question "*What digital artefacts of forensic value can be recovered from Discord application?*". The work then proposes a solution in form of a tool, 'DiscFor', that can retrieve information from the application's local files and cache storage. Written in Python, our tool can be run from the command line as well as from within another forensic software supporting Python-like Cellebrite. DiscFor performs extraction and presentation of the Discord data in a forensically sound manner. The structure of this paper is as follows: Section II provides background on the VoIP instant messaging applications and cache in digital forensics. In Section III, we presented the analysis of Discord data sources. In Section IV our tool DiscFor is presented. Section V details our findings and the performance of the program. The conclusion of our research and further work areas are discussed in Section VI.

II. BACKGROUND AND RELATED WORK

In recent years we have observed an increase in the use of web and smart phone applications to facilitate cybercrimes. Many criminals shifted their interest towards communication software due to almost unlimited possibilities of spreading malicious software and conducting illegal activities. 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. Such activity has led to the emergence of new sub-branches of digital forensics focused solely on research and development of tools needed for the recovery of evidence from new sources of data, i.e. application-specific forensics.

Application-specific forensics focuses on the recovery, analysis, and presentation of data from various applications, also ensuring this evidence is taken in a forensically sound manner and admissible in the court of law. Residual data, file remnants and cache data are of interest when considering application-specific data and client-side storage. Cache is a hardware or software component that stores data so that future requests for that data can be served faster. When a user decides to access the same resource again it is loaded from the local cache to avoid redownloading the content from the server. This method results in the earlier computation of data,

reduction in the amount of information that needs to be transmitted across the network and provides better application user experience. Client-side storage can provide forensic investigators with valuable information about recent activities of the suspect. However, it can be challenging to read the data from the cache structure especially when neither source code nor design is available to the public.

Although Discord was designed for gaming many communities of people have adopted Discord to share information. This mass adoption has opened Discord as a good source for digital evidence. Discord is available both on a desktop environment with any variety of OS choices as well as on both iOS and Android mobile platforms. Each server has a variety of channels that can be joined based on specific topic areas or one can be made for the group that you create. In [10], the authors focused on an analysis of Google Chrome cache structure adopted by Discord. A detailed description of cache format is provided, in addition to the potential data available. The storage features of data blocks and cache addresses are explained also. 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, in [11], analyses of cache storage have been performed to extract YouTube and Facebook stream content. The case study 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 [11].

As noted previously, while there is limited literature on Discord, there are some tools that can be used for the partial recovery of the application's contents. For example, ChromeCacheView (CCV) [12] is a free cache viewing tool for Google Chrome web browser. CCV allows viewing the content of the cache storage (including metadata) and allows extraction of files found within. However, CCV is available only on Microsoft Windows systems. While CCV is currently the best solution for forensic analysis of the Chromium cache structure, there are specific limitations discovered during analysis of Discord data sources. Firstly, CCV does not analyse all files of Simple Cache thus missing some potentially crucial data. Secondly, audio and video files are often stored in two separate parts which are not reconstructed by the tool.

III. DISCORD APPLICATION

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. Location of the Discord directory depends on the OS the app is installed on. Table II presents default installation locations of Discord. Within the installation directory multiple files and folders can be found that contain various data. We have recognised 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.

TABLE I. DISCORD INSTALLATION DIRECTORY

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

The cache structures are 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 though it does not hold any forensically valuable information. The last directory contains data about recently viewed messages, channels, servers etc. and is 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.

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

Disk Cache

Disk Cache was developed as part of a Chromium project by Google and became a base for multiple web browsers such as Google Chrome, Opera and Brave. The structure is available under an open-source licence [15] and was well described in [8]. There is a significant difference between the index file and data_0 that may be of great value for forensic analysis and recovery of evidence. 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. The rankings file contains more forensically valuable data in comparison to the index file. The use of data_0 can significantly simplify the process of data recovery. The eviction information that can be found in the file may provide vital information on when the file was created and accessed for the last time. Figure 2 presents the structure of a data_0 file and its blocks of rankings data.
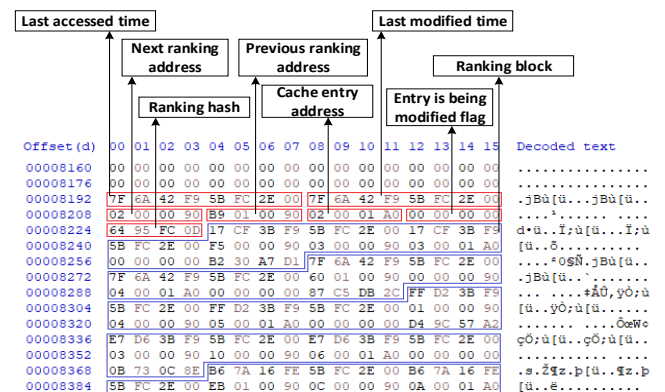

Fig. 2. Ranking block (data_0) structure

The cache entry block files (data_1, data_2, data_3) are responsible for holding control data as well as addresses for resources stored in separate chunks. These separate data streams include server HTTP response and actual resource data. The general structure of the files is similar to data_0 while data streams are well explained in [10].

Simple Cache

Simple Cache is currently used by Discord as the main cache storage for Linux and Android distributions. The Simple Cache folder contains a fake index file, several cache entry files and folder index-dir which contains the-real-index. The-real-index file contains cache addresses for each cache entry. An overview of the Simple Cache folder in presented in Figure 3. The real-index file, as shown in Figure 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 amount of entries in the hash table, cache size and control data. The hash table contains a list of entries and each entry is 24 bytes in size. The cache entry structure is comprised of three parts, as shown in Table II.
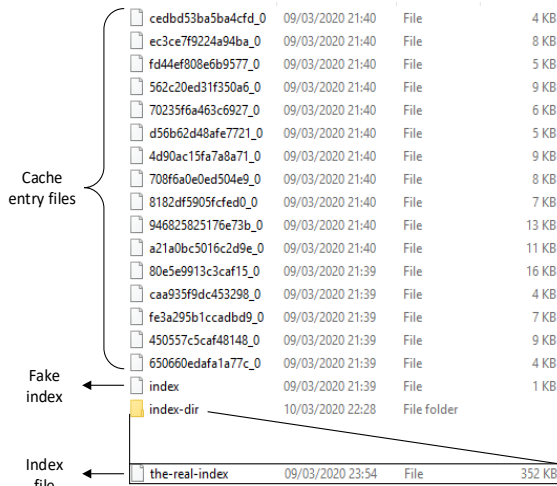

Fig. 3. Simple Cache directory structure

The header contains the following information:

- Payload size – Represents the size of the entire the-real-index file

- Payload CRC32 – Error checking the hash of the file

- Magic - Unique identification value of the file which in case of for every "the-real-index" file is "6F 79 20 72 65 74 6E 65"

- Version – value used for verification of file integrity

- Number of entries – Value representing a number of cache entries stored in a the-real-index file

- Cache size - Represents the size of the entire cache storage

- Reason – Value used to signal the current state of the file and trigger other functions for example flushing function that clears the file of all the content
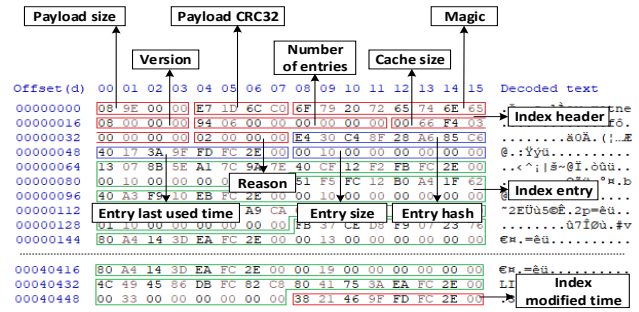

Fig. 4. The-real-index file structure

TABLE II: THE-REAL-INDEX CACHE ENTRY STRUCTURE

Offset	Size (bytes)	Description
0	8	Cache entry hash (name of the cache entry file)
8	8	Cache entry last accessed time
16	8	Cache entry size

The main difference between Simple Cache and Disk Cache is that the former method does not make use of block files. The overall structure is simpler because resource content and HTTP response are stored in a single file. This structure simplifies not only the process of reading and writing cached data but also the analysis and recovery of potential evidence. The name of a cache entry file is a reverse entry hash address from the index file in hexadecimal with an underscore and a stream number which can be either 0, 1 or s. The most used format is with stream number 0 and its structure is as follows:

- A file header

- URL

- Resource content

- EOF (End of File)

- HTTP response
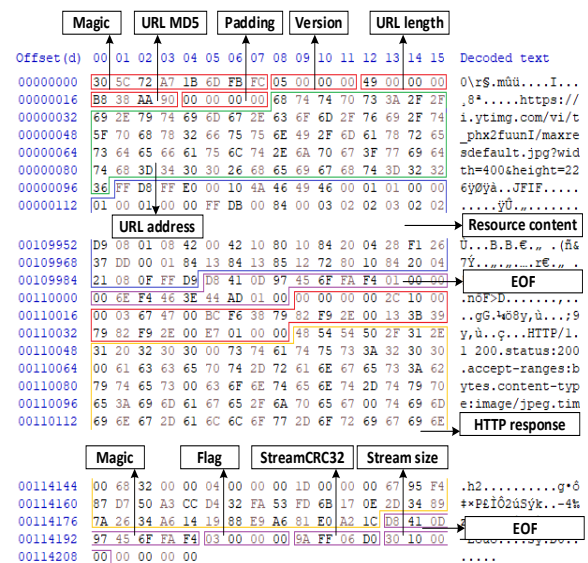
- SHA256 of the URL (optional part)

- EOF

Fig. 5. Structure of the Simple Cache entry file

File reading is being done from the end of the file with the use of EOF sections that contain information about the data

stream above. Our analysis of stream format "1" files does not indicate an existence of potential information of forensic value although this may change in future. #####_s keeps payload of large media or downloadable files. This file contains a list of multiple partial resource copies of the same source which the full version is stored at the beginning of the file. HTTP response of the resource payload is stored within #####_0 of the same name.

Resource content

Resource content describes any file stored in the 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 application content such as app images, emoticons, etc. All content uploaded by users includes text messages, images, audio, video files, linked attachments, etc. As Discord is a messaging app the most interesting data can be found in chat logs stored in a JSON format – shown in Figure 6. As attachments are parts of many messages they are also stored in the 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 the cache entry.

Fig. 6 Discord chat log structure

Server HTTP response provides valuable information on the file from the server perspective and contains several entity headers. The number of headers is not fixed, and different files and applications use various combinations. In Discord, it is often the case that a lot of files either miss some of the mentioned headers or contain entirely different ones. The list below covers the most important headers from the perspective of forensic investigation of Discord application.

- Server response code: indicates whether a specific request has been successfully completed
- Content type: Multipurpose Internet Mail Extension (MIME) content type is a standard that indicates format of a document, file, or assortment of bytes. It consists of type (such as text) and subtype (such as plain) for example text/plain, image/png, and video/mpeg
- ETag: Unique identifier of the specific version of a resource

- Response time: Time when the requested resource was loaded last time
- Last modified time: Time when the resource was last modified
- Expiry time: Time when the resource will be removed from the cache storage
- Max age: Amount of time that source can be kept in cache
- Server name: Name of the server hosting the service
- Server IP: IP address of the server
- Content encoding: This header indicates the method that was used to compress the data

Figure 7 presents an overview of the server HTTP response. The headers of interest are highlighted yellow. While in this state data is legible it must be cleaned from unnecessary characters to improve its readability.

Fig. 7. Discord server HTTP response overview

Activity Log

The last source of interest is an activity log which can be found in the same location for all distributions:

/discord/Local Storage/leveldb

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 as illustrated in Figure 8.

Fig. 8. Activity log - example of recoverable data

IV. DISCFOR DESIGN AND IMPLEMENTATION

To uncover the story behind the criminal activity of a suspect, a digital forensic examiner needs tools that will allow him to view information stored in different formats and sometimes dispersed among multiple locations. The goal of DiscFor is to automate the process of evidence collection from

client-side Discord directories. To achieve this, our tool consists of four main functionalities that address digital forensic process stages. In this section, we present an overview of the architecture used for DiscFor (Figure 9) and implementation details.
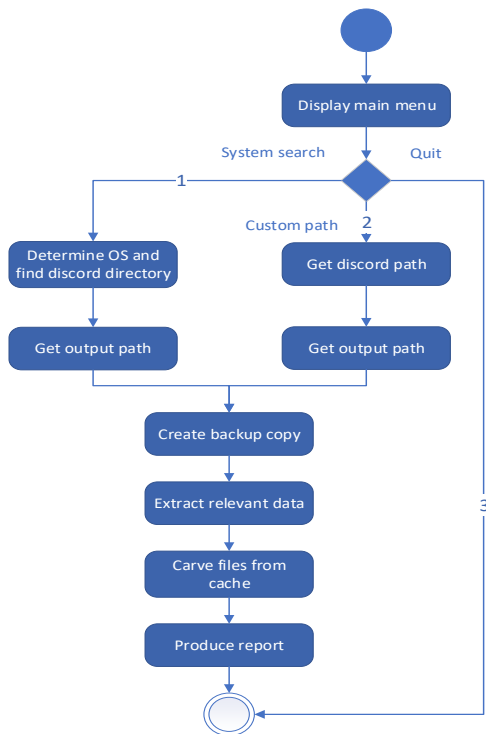

Fig. 9. DiscFor main functionalities overview

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. The next stage is an extraction of data that has been identified as potential evidence. We have designed three different modules to address all three possible sources: Disk Cache, Simple Cache and activity log.

Figure 7 shows the functionality of the module (*maincache.py*) responsible for the recovery of the data from Disk Cache used on Windows OS and macOS. The dispersion of data across multiple locations requires a lengthy process of recovery. The process starts with reading all blocks of rankings table in data_0 and fetching all addresses found within. Next, DiscFor iterates through the addresses on the list and finds cache entries located in one of three data block files (data_1, data_2 and data_3). The cache entry address provides all the information required to locate the position of the entry in the block file.

The following calculation is used in the algorithm to find a block of data in cache:

*Block offset = (Block Number * Block Size) + 8192*

With addresses calculated the tool performs extraction of

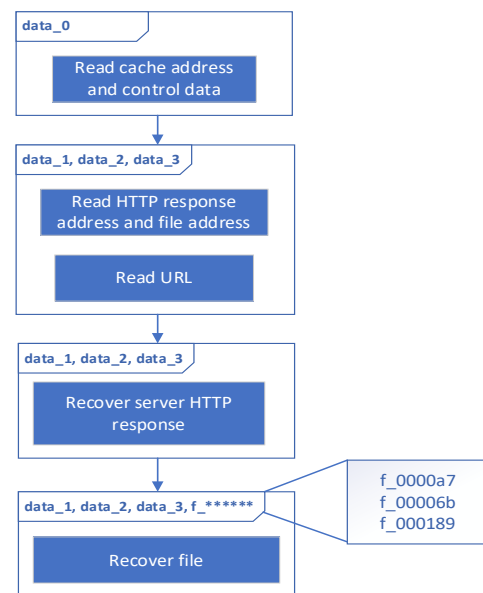server HTTP response and stores all pulled data in a class instance separate for each entry.

Fig. 10. Functionality graph of Disk Cache recovery script

The Simple Cache as the name suggests is less complex than Disk Cache, thus the process of evidence acquisition is simpler as most of the data is stored in a single location. The diagram in Figure 8 shows simple overview of this process. The algorithm responsible for finding entries does not start from reading index file but instead loops through cache directory and reads every single entry found within. This allows to pull information and recover files which trail does not exist in the index file in a situation where the application did not manage to update a file before it was closed. With the cache entry file open, DiscFor proceeds to read data found inside. End of file sections are read first and information about corresponding entries is pulled.
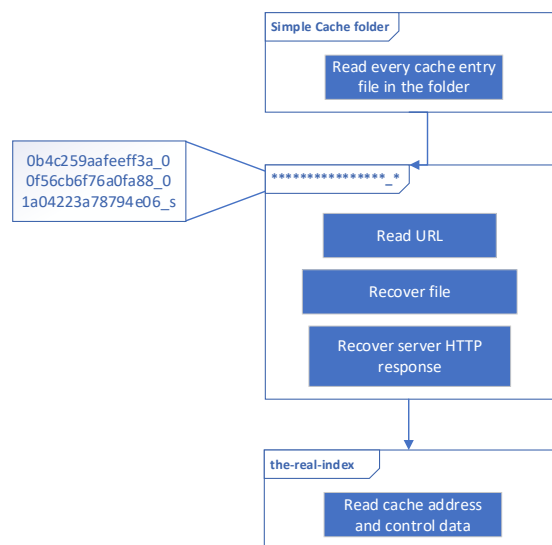

Fig. 11. Functionality graph of Simple Cache recovery script

The case of partial data is sorted during entry read as data is stored in the separate file of similar name where only stream number is changed from "_0" to "_s". If data does not

exist on #####_0 file, the tool opens its #####_s equivalent and reads information required for extraction. The next stage of recovery is to read information from the the-real-index file which has a familiar structure to the data_0 file. The process is similar and is based on reading data from index entry blocks. The tool then compares addresses found in the the-real-index file with ones saved in the main class instance and if there is a match the data is joined.

Recovery of the data from the activity log is performed by employing a simple pattern recognition algorithm which pulls all necessary information. The identification numbers of channels and servers are usually saved in a dictionary where server ID is a key and channel ID is a value and this format looks as follows:

"187217643009212416":"225448446956404738"

To recover this data, the following pattern was developed:

([0-9]+|null)\":\"([0-9]+|null)

For email address detection the following capture group was used:

([a-zA-Z0-9._-]+@[a-zA-Z0-9._-]+\.[a-zA-Z0-9_-]+)

The last component of our tool is responsible for presentation of findings. Information from 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, a decompression function was also implemented. If gzip compression is detected the algorithm decompresses data and the new version is passed further. A MD5 hash value is calculated for each recovered item and stored in the report.

The last reporting feature developed is used to parse data from chat logs (in JSON format) to HTML format files. It was found that some chat log files contain more than one conversation. In the first phase program finds those files, pulls conversations, and saves them to separate files. The channel ID becomes the name of the file. After a basic structure of the file is created the chat log data is loaded, and the program reads all messages found within the conversation. For every message found DiscFor recovers message ID, timestamp, author name, ID and avatar, message content and all attachments. All recovered objects are then inserted into a prepared HTML template of a message. After all, messages are reconstructed, they are joined and appended to HTML structure. Lastly, the whole conversation structure is written into a new HTML file that can be viewed in a web browser.

The last component of our tool is responsible for the presentation of findings. All files found in cache storage are listed in a CSV file with a corresponding server HTTP response, eviction information as well as location in the cache. Because Discord compresses each file uploaded the MD5 hash value is calculated for each recovered file and stored in the report. DiscFor reconstructs chat messages into HTML format

from chat logs and corresponding multimedia files as shown in Figure 9.

Fig. 12. Example of reconstructed message

V. TESTING AND ANALYSIS OF RESULTS

During the testing phase, it was discovered that Disk Cache was flushing its older content when reaching between 3500-4000 files in a directory. This procedure effectively reduces the possible number of evidences that can be collected in the case when the application was extensively used, and users joined many active servers. For testing purposes, we created test scenarios by populating servers with a generic user account and random content exchange. Joining multiple servers allowed the creation of datasets, with varying content such as server data, timestamps, user accounts, message digests, etc. We have generated enough data to determine the importance of Discord forensics and the usefulness of our tool. Figure 13 shows an example message displayed by the application presenting detailed information about each recovery performed with DiscFor. 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.

Fig. 13. DiscFor extraction messages

Table III summarizes the results of running DiscFor on all datasets created for the experiment. 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.

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 tool recovers all data of a forensic value that we have discovered including server HTTP responses, content files, as well as information listed in the activity log. All content types of potential interest are listed in Table IV. Text messages, images, audio, and video files are one of the most important data types for digital investigation having high likelihood of becoming an evidence. 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.

TABLE IV: DISCORD 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

In addition to messages on their content, we were also able to clean the HTTP response and extract information about files sent. Table V presents information found in the response.

TABLE V. DATA TYPES FOUND IN HTTP RESPONSE

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

DiscFor results were compared to ChromeCacheView which also allows recovering most of the data from Discord cache however it does not provide reporting features and some of the data is not being recovered as can be seen in Table 5. Apart from verification of completeness of the recovered data we have also tested the speed of recovery of both solutions.

The performance of both programmes is good even when analysing a large number of files however clearer difference can be observed when performing extraction of the greater number of files (Figure 11). The performance of DiscFor can be improved with further optimisation in order to decrease the time of extraction. Analysis of the reports that DiscFor produced shows that the information displayed is accurate and complete. The CSV format used to save data provides a clear view of details about the files recovered.

TABLE V. COMPARISON OF DICFOR AND CHROMECACHEVIEW RECOVERY RESULTS

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

VI. CONCLUSION AND FUTURE WORK

With an increasing number of Internet users, messaging applications like Discord are seeing rapid popularity growth. With millions of users exchanging messages every day they become rich sources of data which if extracted and appropriately pre-processed can become valid evidence. From the forensic investigator's perspective, it is important to always have access to tools allowing the recovery of important information. While large software solutions are perfect in data recovery from most common sources they often cannot retrieve as much information from other applications. The existence of smaller solutions targeting specific application can often be the only way to collect evidence quickly without the need to perform a lengthy manual extraction. The main advantage of our solution is its ability to perform a full recovery of important data in a timely manner which is often crucial in digital forensic investigation. While available data is limited to the last 30 days from extraction this period of time should allow collecting substantial information for a given case.

Reporting features allow the examiner to find relevant data quickly without manual investigation of cache or JSON files and if such analysis is needed a location of each cache information is included in the report. Future work may involve the examination of mobile application and web variants of Discord as our study was limited to the PC version of the application. Further examination of the activity log is required to ensure if more valuable information can be found within its entries. Due to frequent updates that Discord receives it can be assumed that the structure of the described storage will change.

REFERENCES

[1] C. Sgaras, M. T. Kechadi, and N. A. Le-Khac, "Forensics acquisition and analysis of instant messaging and VoIP applications," 2015, doi: 10.1007/978-3-319-20125-2_16.

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

[3] "Discord: The $2 Billion Gamer's Paradise Coming To Terms With Data Thieves, Child Groomers And FBI Investigators." .

[4] Discord, "Discord Transparency Report," Nelly, Discord Blog, 2019.

[5] M. N. Yusoff, A. Dehghantanha, and R. Mahmod, "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, Elsevier Inc., 2017, pp. 41–62.

[6] G. Hongtao, "Forensic method analysis involving VoIP crime," 2011, doi: 10.1109/KAM.2011.71.

[7] T. Dargahi, A. Dehghantanha, and M. Conti, "Forensics Analysis of Android Mobile VoIP Apps," in Contemporary Digital Forensic Investigations of Cloud and Mobile Applications, Elsevier Inc., 2017, pp. 7–20.

[8] G. S. Suma, S. Dija, and A. T. Pillai, "Forensic Analysis of Google Chrome Cache Files," 2018, doi: 10.1109/ICCIC.2017.8524272.

[9] G. Horsman, "Reconstructing streamed video content: A case study on YouTube and facebook live stream content in the chrome web browser cache," 2018, doi: 10.1016/j.diin.2018.04.017.

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

[11] O. Bryant, "Retrieval of Digital Artefacts from TeamSpeak and Discord: A Forensic Investigation and Analysis of the Malicious Use of Gaming Communication Clients", 2018.

[12] Alfuananzo, "A discord forensics tool.", GITHUB, 2018; https://github.com/alfuananzo/discrecorder

[13] Cristi8, "Parse Chromium cache files on Linux (using 'simple' disk_cache backend)." 2020.

[14] Abrignoni, "JSON-to-HTML-and-XLS: Simple script to convert JSON to html or excel", 2020, GITHUB; https://github.com/abrignoni/JSON-to-HTML-and-XLS

[15] Chronium, The Chronium Projects, "bloomberg/chromium.bb: Chromium source code and modifications." https://www.chromium.org/developers/how-tos/getting-around-the-chrome-source-code