Forensic Investigation of Instant Messaging Services on Linux OS: Discord and Slack as Case Studies

By:
Megan Davis (Virginia Commonwealth University), Bridget McInnes (Virginia Commonwealth University), and Irfan Ahmed (Virginia Commonwealth University)

From the proceedings of
The Digital Forensic Research Conference
DFRWS USA 2022
July 11-14, 2022

DFRWS is dedicated to the sharing of knowledge and ideas about digital forensics research. Ever since it organized the first open workshop devoted to digital forensics in 2001, DFRWS continues to bring academics and practitioners together in an informal environment.
As a non-profit, volunteer organization, DFRWS sponsors technical working groups, annual conferences and challenges to help drive the direction of research and development.
https://dfrws.org

Forensic Science International: Digital Investigation 42 (2022) 301401

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

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

DFRWS 2022 USA - Proceedings of the Twenty-Second Annual DFRWS USA

Forensic investigation of instant messaging services on linux OS: Discord and Slack as case studies

Megan Davis*, Bridget McInnes, Irfan Ahmed

Department of Computer Science, Virginia Commonwealth University, Richmond, VA, 23284, USA

ARTICLE INFO

Article history:

Keywords:
Discord
Slack
Memory forensics
Volatile memory
Discord forensics
Slack forensics

ABSTRACT

Instant messaging applications have gained considerable market share over the past decade and have become one of the most used applications for users worldwide. However, due to the low-barrier to entry and ease of use, these applications (apps) have also garnered the attention of criminals wanting to use the apps to facilitate criminal activities. The memory forensic examination of Discord and Slack on Linux operating system (specifically, Ubuntu 20.04.3 LTS), two popular instant messaging apps, has gone largely unexplored. In this paper, we examined both apps and found data remnants of users' activities that are of forensic interest. We detected a variety of information including: Slack specific data, Discord specific data, username, emails, passwords, messages, conversations, and uploaded attachments, all of which could be utilized in a forensic investigation.

© 2022 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

Instant messaging applications (IMAs) have been around since 1969 when a student sent random messages over the internet on a program dubbed Party Line (A history of instant messaging, 2020). The initial IMA applications were fairly limited in their capabilities, however in the past few years IMAs have expanded their capabilities to include: direct messaging , group messaging, dedicated chat rooms/channels, the ability to upload files, integrating bots, embedded gifs, and voice-over IP. While this has been seen as a boon for team collaborations, it has also become an area for criminal activity to take place. According to Insight, a cyber threat intelligence company, they detected nearly 223,000 mentions of Telegram and 392,000 mentions of Discord on criminal-based community group boards (Vijayan, 2020). As such IMAs have become an area that can provide crucial investigatory data for forensic investigators.

In 2019 the COVID pandemic changed the way workers interacted on a day-to-day basis. Nearly 71% of employees shifted from an in-office working environment to working from home (Parker et al., 2021). Companies and individuals started adopting new ways of communication, including IMAs. Two popular instant messaging applications, Discord and Slack, had seen growth prior to COVID but the COVID work transition helped push wider adoption.

Discord is an application developed by Jason Citron (Curry, 2021a) and it has been traditionally defined as an application for facilitating communication around gaming. This is due to its general association with the streaming platform Twitch which mostly caters to video gaming audiences. But it has tried to broaden its consumer base by working with experts in communities such as gardening and sports (R.E.S.P.A. et al., 2021) (sal19, 2021). The increased adoption of the tool has also attracted hate groups and criminal enterprises. As reported by Kotaku, in 2021 Discord noted that they had removed thousands of servers including those that were related to "child harm material, cybercrime, doxxing, exploitative content, and extremist or violent content" (Grayson, 2020). As of 2021, it boasted 140 million active users (Curry, 2021a).

Whereas, Slack has positioned itself as a more professional application designed for more traditional businesses. Over the past few years its focus has been to gain market share against its main competitor, Microsoft Teams, Slack has also attracted hate groups. In the last report issued by Slack directly in 2019, they stated that they had banned 27 user accounts tied to hate groups (Slack). As of 2019, its last published user count, there were 12 million Slack users (Curry, 2021b).

Discord and Slack are frequently used IMAs that allow the upload of images, videos, and files. Both applications are built using the Electron framework. The Electron framework, released in 2013,

* Corresponding author.
E-mail addresses: davisma24@vcu.edu (M. Davis), btmcinnes@vcu.edu (B. McInnes), iahmed3@vcu.edu (I. Ahmed).

https://doi.org/10.1016/j.fsidi.2022.301401
2666-2817/© 2022 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Table 1
Breakdown of operating systems investigated.

OS	Count
Android	7
iOS	2
Linux	1
Mac	1
Windows 7	2
Windows 8	1
Windows 10	3

is designed to abstract away the knowledge required to build native desktop applications. Instead these apps can be built using web technologies like HTML, CSS, and JS. It works by creating an application window which renders pages that can be interacted with through the native graphical user interface (GUI) of the OS. The data stored in volatile memory of these applications could be valuable artifacts for memory forensic investigators.

In this paper, we present a memory forensic examination of both Discord and Slack on Linux OS, which has gone largely unexplored in the literature (refer to Table 2 for summary). The closest to our work is DiscFor (Motylinski et al., 2020) created by Motylinski et al. for Discord data extraction on disk without memory analysis. We only focus on memory forensics involving examining the memory snapshots taken after IMA-specific activities. Our results show that application information such as usernames, emails, passwords, messages, conversations, and files can be acquired from system and app process memory dumps, which may be useful for forensic investigators. We obtained URLs created when uploading files and found that Discord file URLs can be accessed without restriction. Whereas Slack file URLs require user authentication to ensure that only members of the file's corresponding workspace can access it.

The remainder of the paper is structured as follows. Section 2 describes previous related works. In section 3, we describe our method of experimental setup, memory acquisition, data sanitization, and analysis. Third, we present and discuss our results for both our Discord and Slack experiments in Sections 4, 5, 6. Finally, Section 7 presents our conclusions and future work.

2. Related work

IMAs have been the subject of numerous investigations as seen in Table 2. This table provides a snapshot of the operating systems, focus, applications, and tools used in previous works. This subset of papers is a rather small sampling of the papers published in the last ten years regarding retrieving artifacts from IMAs. While there were papers published prior to 2011, most of them were focused on older operating systems and applications that no longer hold a majority of the market share. The October 2021 GlobalStats report on Operating Systems showed that Android holds 39.75%, Windows 32.44%, iOS 16.7%, OS X 6.85%, Unknown 1.9%, and Chrome OS 1.1% (Operating system market share worldwide, 2021). Table 1 shows the operating systems investigated in the reviewed papers. Like GlobalStats Android had the most papers, followed by Windows, iOS, Linux and Mac. The reviewed papers did not present any reasoning on why they chose a particular operating system over another. However, Motylinski et al. (2020), Wu et al. (Thantilage and Khac, 2019), Choi et al. (Choi et al., 1016), Sgaras et al. (2015), Dezfouli et al. (2015) investigated multiple operating systems.

Most of the apps selected for study followed market trends. As reported by Statista, the most popular global applications are: 1)

Table 2
Existing forensic analysis of instant messaging applications.

Paper	Year	System	Analysis Focus	Application	Tools Used
Salamh et al. (Salamh et al., 2020)	2020	Android	Hard disk	WhatsApp	HxD Hex Editor, Magnet ACQUIRE, OSForensics, SQLLite Forensic Explorer, WinHex
Choi et al. (Choi et al., 1016)	2019	Windows 7 & 10	Hard disk/ Volatile memory	KakaoTalk, NateOn, QQ messenger	CheatEngine, Process Monitor
Motylinski et al. (Motylinski et al., 2020)	2020	Android, Mac, Linux, Windows 10	Hard disk	Discord	DiscFor
Bashir et al. (Bashir et al., 2019)	2019	Windows 10	Hard disk/ Volatile memory	Linkedin	Dumpit, FTK Imager, WinHex
Anglano et al. (Anglano et al., 2016)	2016	Android	Hard disk/ Volatile memory	ChatSecure	Android Emulator, Android Device Bridge, LiME, Volatility
Wu et al. (Wu et al., 2017)	2017	Android	Hard disk	WeChat	Android Debug Bridge
Sgaras et al. (Sgaras et al., 2015)	2015	iOS, Android	Hard disk	WhatsApp, Skype, Viber, Tango	Cellebrite UFED Physical Analyzer & UFED Touch Ultimate, SQLite Studio
Thantilage et al. (Thantilage and Khac, 2019)	2019	Mac, Windows 10	Volatile memory	Facebook, Twitter, FB Messenger, Viber, WhatsApp, Gmail, Hotmail, Yahoo Mail, Skype, Google Search, Yahoo Search, Google Maps	DumpIT, EmEditor, OSXpmem
Dezfouli et al. (Dezfouli et al., 2015)	2015	iOS, Android	Hard disk/ Volatile memory	Facebook, Twitter, Linkedin, Google+	Access Data FTK, DCode, HxD Editor, iBackupBot, Plist Editor for Windows, Sqlite Database Browser, Wireshark
Kazim et al. (Kazim et al., 2019)	2019	Windows 7	Volatile Memory	Google Hangout	Dumpit, Volatility, WinHex
Chu et al. (Chu et al., 2013)	2013	Android	Volatile memory	Skype, MSN	Access Data FTK
Yang et al. (Yang et al., 1371)	2016	Windows	Hard disk/ Volatile memory	Facebook, Skype	FTK Imager, Volatility, Wireshark
Nisioti et al. (Nisioti et al., 2017)	2017	Android	Volatile memory	Facebook, Viber, Whatsapp	Android Device Bridge, LiMe, Hexeditor

WhatsApp, 2) Facebook Messenger, and 3) WeChat (by Statista, 2021). The most notable difference of apps was in Choi et al. (Choi et al., 1016), but it focused on popular IMAs in China and South Korea. The only IMA no longer available is Google+, which shut down on April 2, 2019 (Snider, 2019).

Kazim et al. (2019) used WinHex to evaluate a Window 7 memory dump, they search on keywords such as "chat", "hangouts", "email address", and "@gmail", all that are common words in IM services. They were able to find the corresponding email address and reconstruct an entire IM chat between two users.

Salamh et al. (2020) used Magnet ACQUIRE to create a memdump from a rooted Android device. While they do not specify the Android device or version, they were able to find the WeChats WAL file, which is a file that WeChat writes to before it commits remotely. They used the binwalking method and were able to find the location of the SQLite database and a subset of images. Then, they analyzed the memdump using the Linux String command and found shared URLs, shared images, call logs, deleted messages, random group messages with participants' info.

Wu et al. (2017) installed WeChat on six different android devices. Normally in order to get access to the WeChat directories in Android root privilege is required, however Wu et al. were able to extract the user data by using the Android Debug Bridge (ADB) backup command. Using this method they were able to analyze the data and found user info and WeChat moments.

Bashir et al. (2019) analyzed Linkedin's Windows 10 App. They generated fake data for their users and then used DumpIt to acquire the memory dump. Using WinHex, HexEditor, and manual exploration, they were able to find the location of several artifacts including credentials, images, private info, notifications, job searches, connections, and install time.

Nisioti et al. (2017) utilized LiME wire to acquire memory from an Samsung Galaxy III rooted Android device. They investigated real life data generated over a year for Facebook, WhatsApp, and Viber. In order to gather this data, they generated a set of python scripts that would open the memory dump, search for data using regular expressions, decode any matched data, and then store the results. They extracted data such as messages, group participants, and timestamps from each application.

Thantilage et al. (Thantilage and Khac, 2019) proposed a comprehensive framework that would work as an end to end tool to capture the RAM, extract the data, and then present the evidence in a report. The framework utilized multiple RAM capture tools so it could support multiple OS. Due to the amount of applications it needed to support Thantilage et al. generated a set of regexes for each app. Then they tested their framework on Windows 10 Pro, Windows 10 Home, and macOS High Sierra. They were able to acquire messages, user details and phone numbers from each application. However, the downside to this framework is that for each new app, a new set of regexes needs to be generated and added to gather additional data.

Chu et al. (2013) mimicked messaging between an Android phone on MSN, Skype and a PC. Then used the Android SDK to get the RAM and Access Data FTK to analyze the dump. By using a targeted search for the username of each application, they were able to find instances stored within RAM. Even after the device had been power cycled and a new dump was taken there were still references to the username indicating that there was non-volatile data stored.

Sgaras et al. (2015) investigated the available data found from WhatsApp, Skype, Viber, and Tango on iOS 6.1.3 and Android 2.3.5. They specifically noted that neither device was jail-broken or rooted. For both OS's, they were able to use Filesystem Extraction using Cellebrite UFED Physical Analyzer where they found contacts, installation, traffic, content, and user profile data for Skype. However, they found only a subset of these for Viber, Tango, and WhatsApp. In addition, they found Logical Extraction data for Skype and WhatsApp on Android but not on ioS.

Dezfouli et al. (2015) used a Samsung Galaxy Tab II running Android 4.2 and iPhone 5s running iOS 7.1.2 to generate a fake set of accounts and evaluated what data was retrievable from Facebook, Twitter, Linkedin, and Google+. They dumped the memory using dd Command for Android and Apple itunes for iOS. Then, they looked for keywords, headers, and signatures. They found usernames, names, profile pictures, contact information, location, friends list, posts, messages, comments, and IP address. However, for Google + iOS they were unable to find comments, messages, friend list, location, or education but did not expand upon why that data was not found.

Yang et al. (Yang et al., 1371) set up Windows 8.1 machines that would utilize the Facebook and Skype app. They mimicked a set of tasks and took a snapshot after each one. From the memdumps they were able to determine that both Facebook and Skype had local application structures that utilized local databases. They also found that Facebook caches were only available when logged into the app whereas, Skype maintained theirs throughout the installation lifetime.

A search in Google Scholar for "Memory Forensics Slack" yielded only results regarding slack space, the left over space on harddrives. Whereas, a search for "Memory Forensics Discord" yielded one result, Digital Forensic Acquisition and Analysis of Discord Applications, Motylinski et al. created DiscFor, a tool for Discord data extraction on disk (Motylinski et al., 2020). However, DiscFor did not analyze memory content and only focused on disk. The main reason for this tool creation was to alleviate the issues of identifying and extracting data from the various files and filetypes associated with Discord. By automating this process with DiscFor they were able to quickly extract the data and generate a report. The process flow of DiscFor works as the following: find the Discord directory (identified by what OS) or pass in a custom path to Discord, generate the output path, then create a backup, extract the relevant data, carve the files from the cache and lastly produce a report. DiscFor evaluates 3 different types of data the Disk Cache, Simple Cache and the activity log. This process finds these files and extracts the data based on the carving data types set. Motylinski et al. tested DiscFor on a Windows 10 machine, where they were able to reconstruct Discord entries that showed full messages, user email, channel id, server id, timestamps, attachments, chat logs, user avatars, and javascript files.

3. Forensic examination method

We performed a series of steps to complete our volatile memory forensic examination of both Discord and Slack. Disk based forensics were not in scope for this. The first step of the experiment was to set up the accounts that would be used for testing. We decided to use Google's gmail service since it is one of the most popular and easy to use email services. We created three Gmail accounts belonging to James Kirk, Nyota Uhura, and Leonard McCoy and then created the corresponding Slack and Discord accounts, see Table 3 for all related information.

The second step was to set up Ubuntu 20.04.3 LTS OS. We decided to use virtual machines (VMs) in order to reduce the amount of set up time. In addition, by using VMs, we were able to constrain the amount of RAM to 4 GB, which reduced the amount of data to analyze in the post experiment. The Ubuntu 20.04.3 LTS DiscImage was downloaded and three VMs, Machine1, Machine2, and Machine3, were created using VMware Workstation. As part of this process, we also prepared the files that would be used as part of the testing. Machine1 and Machine2 had a set of files that they

Table 3
Account details.

Experiment	Machine	Username	Email
Discord	Machine1	JamesK654test#5651	jamesK654test@gmail.com
	Machine2	BonesMccoy#4762	leonardmccoync007@gmail.com
	Machine3	NyotaUhura#1273	nyotauhuranc007@gmail.com
Slack	Machine1	James Kirk	jamesK654test@gmail.com
	Machine2	Leonard McCoy	leonardmccoync007@gmail.com
	Machine3	Nyota Uhura	nyotauhuranc007@gmail.com

would upload based on the scenario: through direct messaging between two users they would upload a text document containing the script for an original Star Trek episode and a.jpg image of a starship pulled from Google. For the server interactions we used the same file types but different episode transcripts and images.

Next, we took a total of 54 snapshots for Machine1, Machine2, and Machine3 as highlighted in Table 4. A base snapshot was created before either application was installed, so it could then be used for comparison if needed. It should be noted that there was a subset of snapshots in which only one or two machines were designated to take a snapshot. This was due to how the application was designed. For example once a friend request is accepted in Discord the other account cannot reissue a friend request without removing them as a friend. So in this instance, Machine1's account sent the friends request first, then Machine2's account accepted. Thus, we created two separate snapshots.

Based on background research, we decided to use Volatility 2.7 to dump the process memory for Discord and Slack. In order to use Volatility we generated a profile for the Linux Version. We then used linux_pslist and collected the process ids for Discord and Slack. The process ids remained the same for the duration of the tests; Discord and Slack only generate new process ids when the application is restarted. We then took the process ids and used Volatility's linux_dump_map command. The linux_dump_map command walks through each page of every memory range for a specific process and dumps to disk, however to maintain alignment it will write all zeros for pages not present. This caused an issue where an excess amount of empty files were being created. This specific functionality was due to the zread() call within the plugin that returns an all zero buffer when a page is not present. We modified linux_dump_map from using zread() to use read() instead and then checked if the value was none, if so we skipped printing it out. This significantly reduced the amount of data generated. We ran the modified linux_dump_map command for each memory dump.

Next we grepped on a set of keywords in the outputted files generated from linux_dump_map: Discord, bones, mccoy, uhura, nyota, gmail, james, kirk, discordapp, txt, username, password, and Slack which returned a list of the files which contained these strings. From this list we used the strings command with a limit of n −4, which was set to the smallest length word in our keywords. We were able to eliminate files that did not contain pertinent data. Next, we collected the strings that could contain pertinent information for these applications. Lastly, for the snapshots where the process was not running i.e., Quit Discord Snapshot we used WxHexeditor to inspect the snapshots and search on keywords.

Table 4
Details of the snapshots taken.

Experiment	Snapshot	Account	Description
Discord	Base Snapshot	All	A control snapshot that was made prior to the installation of Slack or Discord.
	Install Snapshot	All	Snapshots were taken after Discord was installed using snap.
	Login Snapshot	All	Snapshots were taken after logging into Discord.
	Profile Image and Status Update Snapshot	All	Snapshots were taken after updating status message and user profile image.
	Friend Request Snapshot	Machine1	A snapshot was taken after the Friend Request Process was initiated. In this, Machine1 sent a friend request to Machine2 and Machine3.
	Friend Request Accepted Snapshot	Machine2/ 3	Snapshots were taken after accepting the Friend Request.
	Direct Messaging Snapshot	Machine1/ 2	Snapshots were taken after the direct message interaction between Machine1 and Machine2.
	Created Server Snapshot	Machine1	A snapshot was taken after the Machine1 account created a Discord Server.
	Joined Sever Server Snapshot	All	Snapshots were taken after Machine2 and Machine3 joined Machine1's Discord server and sent messages in general chat.
	Assigned Roles Snapshot	All	Snapshots were taken after Machine1 assigned roles to Machine2 and Machine 3.
	Added Role to Private Channel Snapshot	All	Snapshots were taken after Machine1 changed Machine's 3 role to see the hidden channel.
	Voice Chat Snapshot	All	Snapshots were taken while Machine1 and Machine2 were in the General Voice chat channel.
	Quit Discord Snapshot	All	Snapshots were taken after each user quit Discord.
Slack	Install Snapshot	All	Snapshots were taken after Slack was installed using snap.
	Login/ Create Workspace Snapshot	Machine1	A snapshot was taken after logging into Slack and creating a workspace.
	Login/ Joined Slack Workspace Snapshot	Machine2/ 3	Snapshots were taken after Machine2 and Machine3 joined Machine1's workspace.
	Added Giphy Bot Snapshot	Machine1	A snapshot was created after Machine1 added the Giphy bot to the workspace.
	Updated Profile Snapshot	All	Snapshots were taken after updating status message and user profile image.
	Direct Chat Message Snapshot	All	Snapshots were taken after Machine1 sent direct messages to Machine2 and Machine3.
	Private Chat Snapshot	Machine1/ 2	Snapshots were taken after messages were sent in a private Chat.
	Private Chat Added Snapshot	All	Snapshots were taken after Machine3 was added to the private Chat.
	Quit Slack Snapshot	All	A snapshot was taken after each user quit Slack.

4. Discord results & discussion

This section discusses our findings of memory forensics on Discord application.

4.1. Install snapshot

This snapshot was taken after using the snap command "sudo snap install discord" to install Discord. The only information found was the reference to snap installation.

4.2. Login snapshot

This snapshot was taken after the Discord login process had been completed. At first, we attempted a search for @gmail, which yielded over a hundred matches. Most of the matches belonged to open source scripts that Discord utilized. Using the full @gmail address for each machine we refined it to around 33–35 matches.

From this data we gathered the following information: username, id, avatar id, channel ids, the phone number associated with the account, guild positions, guild folders, show current game status, region, and geolocation. Most importantly, we found a JSON string containing the user's login and password, as seen in Fig. 1. The login JSON was found in all subsequent Discord snapshots.

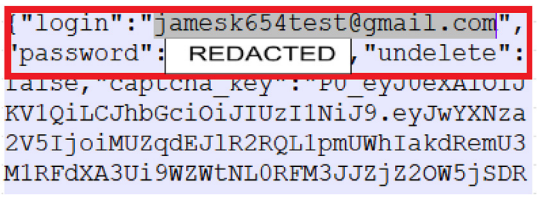


Fig. 1. Machine1 username password, password redacted.

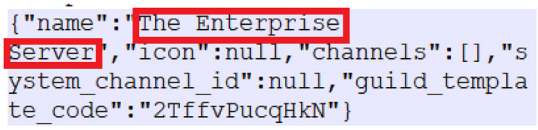


Fig. 2. Discord server JSON

4.3. Profile image and status update

This snapshot was taken after the user's avatar image and status were updated. We found the URL that Discord created for the uploaded avatar image; this is discussed in more detail in 4.6.

The custom status messages were more difficult to locate, initially we looked for a JSON structure that contained status but were unable to find any. However, after searching for the exact status update phrase we found the full string, for example for Machine1 we found

"Captain of a starship!".

4.4. Friend request snapshot

We made Machine1 send friend requests to Machine2 and Machine3 using their Discord ids. From this snapshot, we found data referring to each user and a success message from Machine1:

Your friend request to BonesMccoy#4762 was sent.

4.5. Friend request accepted snapshot

Accepting the Friend Request didn't yield any new information.

4.6. Direct messaging snapshot

As mentioned in the set up, Machine1 and Machine2 had a set of files that they would upload to each other. We decided to limit this interaction to only Machine1 and Machine2 to reduce the amount of set up and snapshots. Machine1 and Machine2 uploaded their set of files to each other with a short message, gif, and reaction (applied to a message). On each snapshot we found data relating to the messages including discordapp links for the attachments. Discordapp links are the way Discord handles file retention, it follows this basic format: https://(cdn/media).discordapp.com/(avatars/attachments)/id/file_id/file_name. With this information we were able to take the URLs found[1][2] and paste them into a separate browser and retrieve the uploaded files and images.

4.7. Created server snapshot

Machine1 created a new Discord server called "The Enterprise Server". In this snapshot we found JSON objects, see Fig. 2, referencing the server name, as well as direct references to the Discord Guild id.[3]

4.8. Joined sever snapshot

Machine1 sent server invitation links to Machine2 and Machine3; both machines accepted and joined the server. Then, Machine1 and Machine2 sent messages in the Discord server #general chat channel that included images, gifs, text files, reactions, and text-only messages. Machine3 only responded to Machine1 and Machine2 messages with gifs and text. In all three snapshots we found JSON that contained the image URLS, text files URLs, message text, and gif URLs. Fig. 3 shows a small snippet of the information available including gif link, channel id, author name, author id, timestamp, and edited timestamp.

4.9. Assigned roles snapshot

On the Discord server, Machine1 created two roles, communications and medical, and assigned them to Machine2 and Machine3 respectively. Then Machine1 created a private chat channel called #officers-only that at first could only be seen by the server owner (Machine1) and users with the medical role (Machine2). In this channel Machine1 and Machine2 exchanged text messages.

Within the Machine1 and Machine2 snapshots we found JSON structures that referenced the conversations shown in Fig. 4. However, in the Machine3 snapshot, we found a string reference to the channel #officers-only but no additional information.

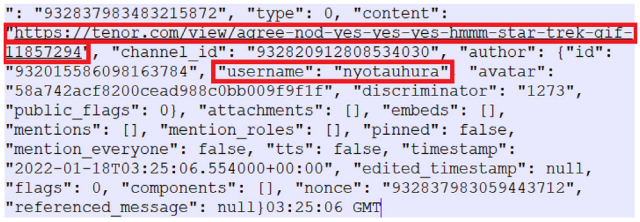
4.10. Added role to private channel snapshot

Machine1 updated the private channel to allow those with the communication role to see it. After this change we found references

[1] https://cdn.discordapp.com/attachments/932809094883082250/932813445072584764/s1-04.

[2] https://cdn.discordapp.com/attachments/932809094883082250/932813785197072495/starship-1-1-message.jpg.

[3] https://discordapp.com/channels/932820912808534027/932820912808534030.



```
": "932837983483215872", "type": 0, "content":
"https://tenor.com/view/agree-nod-yes-yes-yes-hmmm-star-trek-gif-
11857294", "channel_id": "932820912808534030", "author": {"id":
"932015586098163784", "username": "nyotauhura", "avatar":
"58a742acf8200cead988c0bb009f9f1f", "discriminator": "1273",
"public_flags": 0}, "attachments": [], "embeds": [],
"mentions": [], "mention_roles": [], "pinned": false,
"mention_everyone": false, "tts": false, "timestamp":
"2022-01-18T03:25:06.554000+00:00", "edited_timestamp": null,
"flags": 0, "components": [], "nonce": "932837983059443712",
"referenced_message": null}03:25:06 GMT
```


Fig. 3. Machine3 discord gif response.

to the text messages sent in #officers-only in Machine3's snapshot. This data can be seen in Fig. 5 and Fig. 6.

4.11. Voice chat snapshot

For the voice chat snapshot, Machine1 and Machine2 joined the general voice channel while Machine3 did not. The only reference we found about voice chat in the snapshots was one line that specifically references the General voice channel stating :"General (voice channel), 2 users". There were no references to which users were connected nor how long they had been connected.

4.12. Quit Discord Snapshot

No new data was found, but the Discord data did remain in memory.

5. Slack results & discussion

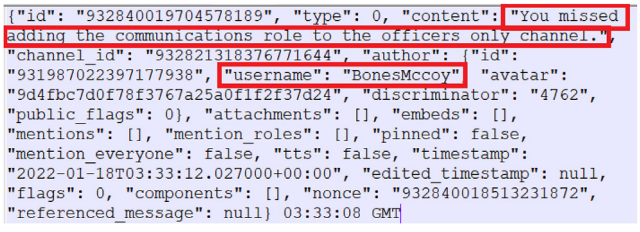
This section discusses our findings of memory forensics on Slack application.

5.1. Install snapshot

This snapshot was taken after using the snap command "sudo snap install slack —classic" to install Slack. We found the same snap install references as we had found in the Discord Install Snapshot.

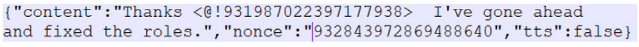
5.2. Login/ create workspace snapshot

There are two main differences between the initial login process of Discord and Slack. First, the login process for Slack is routed through an external login server. We utilized the Gmail login service since we had already created gmail accounts for each user. Second, Slack limits the ability for users to interact on a one-to-one basis. In order to have one-to-one conversations both users must be a part of the same workspace because users are only defined in the context of a workspace. We had Machine1 login first, create a workspace, and send Slack invitations to Machine2 and Machine3. The snapshot data revealed emails, team id, name, real name, timezone, display info, statuses, workspace invite and workspace name, as seen in Fig. 7. However, we were unable to find any reference to the gmail password or an outh2 token.



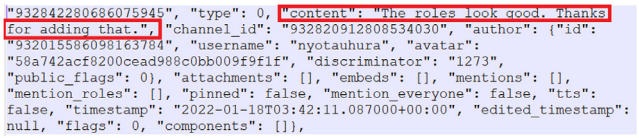
```
{"id": "932840019704578189", "type": 0, "content": "You missed
adding the communications role to the officers only channel.",
"channel_id": "932821318376771644", "author": {"id":
"931987022397177938", "username": "BonesMccoy", "avatar":
"9d4fbc7d0f78f3767a25a0f1f2f37d24", "discriminator": "4762",
"public_flags": 0}, "attachments": [], "embeds": [],
"mentions": [], "mention_roles": [], "pinned": false,
"mention_everyone": false, "tts": false, "timestamp":
"2022-01-18T03:33:12.027000+00:00", "edited_timestamp": null,
"flags": 0, "components": [], "nonce": "932840018513231872",
"referenced_message": null} 03:33:08 GMT
```


Fig. 4. Discord private channel text.



```
{"content":"Thanks <@!931987022397177938>  I've gone ahead
and fixed the roles.","nonce":"932843972869488640","tts":false}
```


Fig. 5. Machine1 message denoting adding communications role to private channel.



```
"932842280686075945", "type": 0, "content": "The roles look good. Thanks
for adding that.", "channel_id": "932820912808534030", "author": {"id":
"932015586098163784", "username": "nyotauhura", "avatar":
"58a742acf8200cead988c0bb009f9f1f", "discriminator": "1273",
"public_flags": 0}, "attachments": [], "embeds": [], "mentions": [],
"mention_roles": [], "pinned": false, "mention_everyone": false, "tts":
false, "timestamp": "2022-01-18T03:42:11.087000+00:00", "edited_timestamp":
null, "flags": 0, "components": []},
```


Fig. 6. Machine3 posting a message in the private channel.

5.3. Login/ joined slack workspace snapshot

The snapshot data did not reveal any new information.

5.4. Added giphy bot snapshot

Unlike Discord, where gifs are an inherent part of the chat options, Slack requires the addition of a bot that interacts with gifs. We decided to have Machine1 add the Giphy bot to the workspace. We found that once added we were able to see JSON references to gifs as seen in Fig. 8.

5.5. Updated profile snapshot

For each user we had them update their profile image and status message. From the snapshot were we able to find references to the uploaded avatar files,[4] and accessed them. In addition to the image, we found references to the custom status in JSON format, as seen in Fig. 9.

5.6. Direct chat message snapshot

Now that all users existed within the same workspace they could send direct messages to one another. We had Machine1 send direct messages to Machine2 and Machine 3. The snapshot analysis from all machines revealed a wealth of information including file uploads, message data, and channel data. However, unlike Discord, where we were able to use the URL to retrieve the files, when we attempted to access the URL from Slack[5] [6] it redirected us[7][8] to sign in to the Enterprise Corp workspace. Slack, unlike Discord, has an extra layer of security where it ties the file to the team id and thus when trying to access an URL in that workspace requires verification that the user is part of the workspace. This change makes sense for Slack since it has been marketed at a more professional environment, where files and data are generally considered more sensitive.

5.7. General chat messages snapshot

We had each machine send messages including text, gifs and reactions in the general chat channel. We were able to find timestamp, text, author, author icon, author id, and author link.

[4] https://avatars.slack-edge.com/2022-01-18/2971087465987_79e735b0ed7d27e40afe_192.jpg.

[5] https://files.slack.com/files-tmb/T02UJ1FRX7U-F02UMAGGG4C-e09e01d758/starship-1-1-message_80.jpg.

[6] https://files.slack.com/files-pri/T02UJ1FRX7U-F02UCAXFG1L/s1-04.

[7] https://the-enterprise-corp.slack.com/?redir=%2Ffiles-tmb%2FT02UJ1FRX7U-F02UMAGGG4C-e09e01d758%2Fstarship-1-1-message_80.jpg.

[8] https://the-enterprise-corp.slack.com/?redir=%2Ffiles-pri%2FT02UJ1FRX7U-F02UCAXFG1L%2Fs1-04.


{"results":[{"id":"U02V7P3H8SC","team_id":"T02UJ1FRX7U","name":"jamesk654test","deleted":false,"color":"9f69e7","real_name":"J TKirk","tz":"America\/Los_Angeles","tz_label":"Pacific Standard Time","tz_offset":-28800,"profile":{"title":"","phone":"","skype":"","real_name":"J TKirk","real_name_normalized":"J TKirk","display_name":"","display_name_normalized":"","fields":null,"status_text":"","status_emoji":"","status_emoji_display_info":[],"status_expiration":0,"avatar_hash":"33ac72c54169","image_original":"https:\/\/s3-us-west-2.amazonaws.com\/slack-files2\/avatars\/2022-01-18\/2993785704752_33ac72c5416911390064_original.png","is_custom_image":true,"email":"jamesk654test@gmail.com","first_name":"J","last_name":"TKirk","status_text_canonical":"","team":"T02UJ1FRX7U"},"is_admin":true,"is_owner":true,"is_primary_owner":true,"is_restricted":false,"is_ultra_restricted":false,"is_bot":false,"is_app_user":false,"updated":1642529936,"is_email_confirmed":true,"who_can_share_contact_card":"EVERYONE","presence":"active"}],"presence_active_ids":["U02V7P3H8SC"],"ok":true}

Fig. 7. Server workspace creation JSON


{"type":"message","user":"U02V7P3H8SC","bot_id":"B02U4FVEE3H","suppress_notification":false,"text":"star trek","team":"T02UJ1FRX7U","bot_profile":{"id":"B02U4FVEE3H","deleted":false,"name":"giphy","updated":1642541670,"app_id":"A0F827J2C","icons":{"image_36":"https:\/\/a.slack-edge.com\/dc483\/img\/plugins\/giphy\/se

Fig. 8. Slack giphy bot JSON

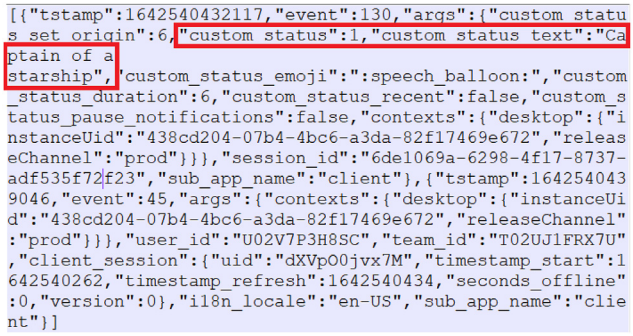

[{"tstamp":1642540432117,"event":130,"args":{"custom_status_set_origin":6,"custom_status":1,"custom_status_text":"Captain of a starship","custom_status_emoji":":speech_balloon:","custom_status_duration":6,"custom_status_recent":false,"custom_status_pause_notifications":false,"contexts":{"desktop":{"instanceUid":"438cd204-07b4-4bc6-a3da-82f17469e672","releaseChannel":"prod"}}},"session_id":"6de1069a-6298-4f17-8737-adf535f72f23","sub_app_name":"client"},{"tstamp":1642540439046,"event":45,"args":{"contexts":{"desktop":{"instanceUid":"438cd204-07b4-4bc6-a3da-82f17469e672","releaseChannel":"prod"}}},"user_id":"U02V7P3H8SC","team_id":"T02UJ1FRX7U","client_session":{"uid":"dXVpO0jvx7M","timestamp_start":1642540262,"timestamp_refresh":1642540434,"seconds_offline":0,"version":0},"i18n_locale":"en-US","sub_app_name":"client"}]

Fig. 9. Machine1 custom status JSON

5.8. Private chat snapshot

Similar to our Discord Private chat, we had Machine1 create a private channel #officers-table and only invited Machine2. Unlike Discord there is not a roles feature and Machine1 only had to reference Machine2's username to add them. Within this chat Machine1 and Machine2 corresponded. We found JSON data referencing those messages on Machine1 and Machine2 snapshots. As expected we could not find any reference to the channel or the messages from Machine3's snapshot.

5.9. Private chat added snapshot

We made Machine1 add Machine3 to the private channel #officers-table. We were then able to find direct references to the messages exchanged.

5.10. Quit slack snapshot

No new data was found, but Slack data remained in memory.

6. Overall discussion

In Table 5, we present a summary of a subset of artifacts obtained and from which application. This is not a fully comprehensive list because each of the artifacts are part of a JSON object which contains multiple attributes. Each of the attributes is dependant on the type of object being fetched. For example the message JSON object from Discord has the following attributes: type, channel_id, avatar, discriminator, public_flags, attachments, embeds, mentions, mentions_roles, pinned, mention_everyone, tts, timestamp, edit_timestamp, flags, and components. In addition some Slack artifacts are defined with x* since the we were able to find the artifact but not access it.

Table 5
Artifacts found in discord and slack.

Artifact	Discord	Slack
Avatar Image	x	x*
Discord Server	x	N/A
Email	x	x
Emojis	x	x
Friend Request	x	N/A
Gifs	x	x
Image	x	x*
Message	x	x
Password	x	–
Restricted Channels	–	–
Status	x	x
Slack Workspace	N/A	x
Text File	x	x*
User Roles	x	N/A
Username	x	x
Voice Chat	–	N/A

Finding the Discord password is not entirely unexpected as we are inspecting volatile memory. Within volatile memory we expect to see passwords or other sensitive information. We were able to find the login JSON in the subsequent Discord snaps shots. However, these snapshots were all performed without restarting Discord. Discord does store the login information after the first successful login attempt so the user does not have to login in every time, however it possible that instead of finding the login information on a restart we would find a login token. While Discord has each user create a Discord specific account, Slack utilizes a third party for account generation. In this instance we used gmail authentication and were unable to find any reference to the gmail password nor an oauth token.

As noted in Section 4.6, Discord does not restrict access to the images and files uploaded to Discord, they generate URLs that can be accessed without any authentication. Whereas Slack requires the user to authenticate against the workspace that owns the image or file. We were unable to find any articles on why Discord does not restrict access to files or images by Discord Guild.

Of the tested artifacts the two that remained inaccessible were the restricted channels and voice chat. For the restricted channels we were unable to retrieve any JSON objects regarding the restricted channel by the user excluded from the channel. Neither application will push data to users that should not have access, the data only becomes available once the users are given permission to see the restricted channels. As for the Discord voice chat, we were able to find a single string that references the general voice channel however there was not a JSON object associated with it or any other information. One possibility is that at the time of the snapshot the JSON object that is pushed client side to update the application's user interface was not captured or overwritten.

7. Conclusion and future work

Instant messaging applications (IMAs) have become a staple of day-to-day communication for individuals and businesses alike. However, for the same reasons companies find IMAs useful so do criminals. Two of the more popular IMA applications Discord and Slack have had limited analysis regarding what artifacts can be retrieved from volatile memory, data that could be useful to forensic investigators.

This paper presented the findings from a forensic examination of two popular IMAs Discord and Slack. The experiment conducted

examined snapshots taken after installation, login, direct messages, server/workspace messages, file uploads and other IMA specific activities related to Discord and Slack. The results showed that Discord and Slack data could be obtained from volatile memory. We were able to acquire login information, phone numbers, chat messages, attachments, reactions, bot ids, and app related data. Also, we identified the lack of security restrictions built into Discord's uploaded files and the restrictions placed upon Slack uploaded files. This data could be useful for forensic investigators.

While we found data stored within volatile memory there is still room for exploration of other situations that could yield additional data. Future work should explore users editing messages, Discord bots, retrieving different file types like videos, users leaving Discord servers, users deleting their accounts, users deleting messages, administrators deleting messages, pinned messages, detect if a user was muted or banned, what data exists after a prolonged period of the application running, what data exists after quitting the application, what data exists after reopening the application, Discord screen sharing/ streaming, and if any data remains after uninstalling the application. Lastly, a framework could be created to parse the files for pertinent JSON objects quickly, reconstruct them, and then display the data in a manner that is easier to read.

References

2021, R.E.S.P.A., Peterson, R.E.S., Peterson, S., Peterson, M.G.S., Aug 2021. With 150 Million Users, Does Discord Have the Goods to Be a Platform? https://omr.com/en/discord-marketing-branded-communities/.

A History of Instant Messaging and Chat, Sep 2020. https://www.maize.io/news/lizshemaria-historyof-instant-messaging/.

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

Bashir, S., Abbas, H., Shafqat, N., Iqbal, W., Saleem, K., 2019. Forensic analysis of linkedin's desktop application on windows 10 os, 16th International Conference on Information Technology-New Generations (ITNG 2019). Adv. Intell. Syst. Comput. 57–62. https://doi.org/10.1007/978-3-030-14070-0_9.

by Statista, P., Nov 2021. Research Department, N. 2, Most popular messaging apps. https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps/.

J. Choi, J. Yu, S. Hyun, H. Kim, Digital forensic analysis of encrypted database files in instant messaging applications on windows operating systems: case study with kakaotalk, nateon and qq messenger, Digit. Invest. 28. doi:10.1016/j.diin.2019.01.011.

Chu, H.-C., Lo, C.-H., Chao, H.-C., 2013. The disclosure of an android smartphone's digital footprint respecting the instant messaging utilizing skype and msn. Electron. Commer. Res. 13 (3), 399–410. https://doi.org/10.1007/s10660-013-9116-1.

Curry, D., Sep 2021. Discord Revenue and Usage Statistics, 2021. https://www.businessofapps.com/data/discord-statistics/.

Curry, D., May 2021. Slack Revenue and Usage Statistics, 2021. https://www.businessofapps.com/data/slack-statistics/.

Dezfouli, F.N., Dehghantanha, A., Eterovic-Soric, B., Choo, K.-K.R., 2015. Investigating social networking applications on smartphones detecting facebook, twitter, linkedin and google artefacts on android and ios platforms. Aust. J. Forensic Sci. 48 (4), 469–488. https://doi.org/10.1080/00450618.2015.1066854.

Grayson, N., 2020 (Apr 2021. Discord Deleted Thousands of Violent Extremist and Criminal Servers in. https://kotaku.com/discord-deleted-thousands-of-violent-extremist-and-crim-1846623284.

A. Kazim, F. Almaeeni, S. A. Ali, F. Iqbal, K. Al-Hussaeni, Memory forensics: recovering chat messages and encryption master key, 2019 10th International Conference on Information and Communication Systems (ICICS)doi:10.1109/iacs.2019.8809179.

M. Motylinski, A. Macdermott, F. Iqbal, M. Hussain, S. Aleem, Digital forensic acquisition and analysis of discord applications, 2020 International Conference on Communications, Computing, Cybersecurity, and Informatics (CCCI)doi:10.1109/ccci49893.2020.9256668.

A. Nisioti, A. Mylonas, V. Katos, P. D. Yoo, A. Chryssanthou, You can run but you cannot hide from memory: extracting im evidence of android apps, 2017 IEEE Symposium on Computers and Communications (ISCC)doi:10.1109/iscc.2017.8024571.

Operating System Market Share Worldwide, Oct 2021. https://gs.statcounter.com/os-market-share.

Parker, K., Horowitz, J.M., Minkin, R., May 2021. How Coronavirus Has Changed the Way Americans Work. https://www.pewresearch.org/social-trends/2020/12/09/how-the-coronavirus-outbreak-has-and-hasnt-changed-the-way-americans-work/.

sal19, May 2021. Discord Is Rapidly Expanding beyond Gaming, Attracting Suitors like Sony and Microsoft. https://www.cnbc.com/2021/05/08/what-is-discord-chat-service-fosters-community-expands-beyond-gaming.html.

F. E. Salamh, U. Karabiyik, M. K. Rogers, Asynchronous forensic investigative approach to recover deleted data from instant messaging applications, 2020 International Symposium on Networks, Comput. Commun.doi:10.1109/isncc49221.2020.9297227.

Sgaras, C., Kechadi, M.-T., Le-Khac, N.-A., 2015. Forensics acquisition and analysis of instant messaging and voip applications. Compu. Foren. Lect. Notes in Comp. Sci. 188–199. https://doi.org/10.1007/978-3-319-20125-2_16.

Slack, Slack, Mar 2019. Removes Hate Groups. https://slack.com/blog/news/slack-statement-hate-groups.

Snider, M., Feb 2019. Google Sets April 2 Closing Date for Google , Download Your Photos and Content before Then. URL. https://www.usatoday.com/story/tech/talkingtech/2019/02/01/google-close-google-social-network-april-2/2741657002/.

R. D. Thantilage, N. A. L. Khac, Framework for the retrieval of social media and instant messaging evidence from volatile memory, 2019 18th IEEE International Conference on Trust, Security and Privacy in Computing and Communications/13th IEEE International Conference on Big Data Science and Engineering (TrustCom/BigDataSE)doi:10.1109/trustcom/bigdatase.2019.00070.

Vijayan, J., Jun 2020. Criminals turn to im platforms to avoid law enforcement scrutiny. https://www.darkreading.com/risk/criminals-turn-to-im-platforms-to-avoid-law-enforcement-scrutiny.

Wu, S., Zhang, Y., Wang, X., Xiong, X., Du, L., 2017. Forensic analysis of wechat on android smartphones. Digit. Invest. 21, 3–10. https://doi.org/10.1016/j.diin.2016.11.002.

T. Y. Yang, A. Dehghantanha, K.-K. R. Choo, Z. Muda, Windows instant messaging app forensics: facebook and skype as case studies, PLoS One 11 (3). doi:10.1371/journal.pone.0150300.