Discord, bots and privacy

Rasmus Lindgren & Jonathan Andersson Kraft
Group 28 - Language Based Security

May 2021

Contents

1 Introduction

Bots and apps integrated into other services can be very useful, incorporating a wide variety of features from third-party creators. In the chat service Slack, for instance, productivity apps such as schedulers and time trackers are popular. Other chat services such as Microsoft Teams and Discord also contain third-party code integrations, providing users with powerful tools.

However, incorporating foreign code into a service poses security risks, such as sharing sensitive personal information or risking possible remote-code-execution attacks on user systems. This is a serious issue, considering the millions of people using such services on a regular basis[1].

Therefore, this report investigates possible vulnerabilities in Discord with an emphasis on Discord bots. Discord bots are one of the types of third-party code integrations present in the Discord service. The goal is to find out to which extent a malicious bot could do harm, including if it could compromise the privacy of a user. In addition, we want to explore and discuss the language based security measures present in Discord, resulting in our suggestions for improvements.

This report is structured into four sections (*excluding this introduction and the appendix*). In section *2: Discord* we begin by giving a brief introduction of Discord, Discord bots, different security measures present in Discord, as well as some previous exploits of the Discord software. Inspired by these previous exploits, we conceptualize and implement a few different approaches to using Discord bots maliciously in section *3: Abusing Discord using bots*, which also explains how to execute our code. Finally, in sections *4: Discussing issues and possible solutions* through *5: Conclusion* we discuss and conclude the findings of this report, aiming to answer the goals posed above. The appendix contains the complete code for our bot implementations, along with images of the concept bot in action.

2 Discord

Discord is a popular messaging application with over 100 million monthly users [1]. The application was designed with the purpose of creating and maintaining online communities, initially growing out of the gaming community, as well as the streaming community on Twitch [2]. Discord was created by CEO and founder Jason Citron, launched in 2015 [3]. What differentiates Discord from other similar services such as Slack or Microsoft teams is that Discord targets a largely different user base. Whereas Microsoft teams and Slack seemingly focuses on more professional contexts, such as work or school, Discord has mostly been used by friends and online communities. However, Discord has recently tried to branch out to reach larger audiences, growing from its initial communities [1].

In Discord, communication is either handled through servers or private messages. Both of these kinds of communication are hosted through Discords services, as it is not possible to host a Discord server locally. Servers on Discord contain members, providing them with both voice- and text channels. The channels themselves can only be created by administrators of the server in question. Any user can create their own Discord server, which they automatically become administrators of.

2.1 Bots

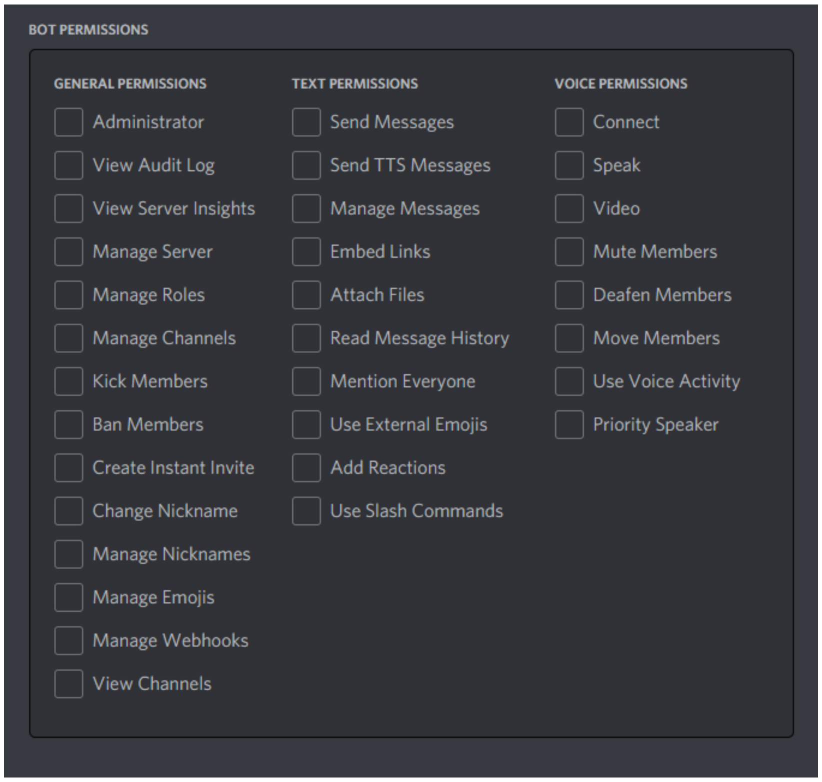
The third-party code integrations discussed in this report are called bots, namely Discord bots. These bots are connected to Discord servers, visualized as regular users except for a bot tag next to their name. Similar to regular users, they can be given privileges in what they can interact with and in which channels they can be active. For example, some purposes of these bots could be to automate tasks such as moderation, relaying important information or to perform other simple tasks. The bots themselves are not hosted by Discord, they are hosted by the creator of the bot. Bot creators can create links for adding their bot to servers, which is usually given to third-party websites. Finding bots to add to a server is then handled by these websites, which distribute these links for adding bots to a Discord server.

2.2 Security measures

There are several security measures in place to prevent malicious use of *Discord bots*. Here, we explain the use of a few of them.

Each request from a bot to Discord requires the use of a token, unique for each bot. The token is generated by Discord and is then compared to the locally stored version, acting as a shared secret between the bot and Discord. The use of this token safely verifies the bot, as long as it is kept secret.

Besides this token, there is also a list of permissions given to a bot when it is added to a Discord server. The given permissions are saved as a new *role* for the bot, describing what the bot is allowed to do. These permissions could include, for example, the permission to send messages, or to join a voice call. For a complete list of permissions, please refer to figure 1. Without having appropriate permissions, the bot is blocked from performing the respective actions such as reading messages of a channel it is not a part of, or banning members. Administrators of a server can create other roles, mostly used for regular members. However, there is a default role applying to all members of a server (including bots), namely the *@everyone* role. The permissions present in the *@everyone* role forms the baseline permissions everyone has. If a member is part of several roles, their resulting permissions is the union of all their respective roles permissions.



BOT PERMISSIONS

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
☐ Administrator	☐ Send Messages	☐ Connect
☐ View Audit Log	☐ Send TTS Messages	☐ Speak
☐ View Server Insights	☐ Manage Messages	☐ Video
☐ Manage Server	☐ Embed Links	☐ Mute Members
☐ Manage Roles	☐ Attach Files	☐ Deafen Members
☐ Manage Channels	☐ Read Message History	☐ Move Members
☐ Kick Members	☐ Mention Everyone	☐ Use Voice Activity
☐ Ban Members	☐ Use External Emojis	☐ Priority Speaker
☐ Create Instant Invite	☐ Add Reactions	
☐ Change Nickname	☐ Use Slash Commands	
☐ Manage Nicknames		
☐ Manage Emojis		
☐ Manage Webhooks		
☐ View Channels		

Figure 1: All permissions available for bots.

Discord also implements more technical and common security measures such as sanitation of input and a content security policy (CSP).

Popular bots which have been added to more than 100 servers will need to be verified by Discord, adding another layer of security.

2.3 Previous exploits

By investigating previous Discord vulnerabilities and exploits, the most prevalent vulnerability seems to be phishing and scam attempts. These usually come in the form of malicious links infecting clients with malware. There are also examples of bots claiming to be official representatives, wanting access to servers with promises of free games or services [4].

However, there has been serious breaches of security that do not depend on users actively interacting with malicious actors. As recently as 2020, an XSS attack was successful in exploiting Discord [5], discovered by Masato Kinugawa. The attack was performed using the Discord desktop app and was made possible due to vulnerabilities in Electron, a software framework for developing graphical user interfaces [6]. The vulnerability in question was that *contextIsolation* was not enabled, which made remote code execution possible. Context isolation separates Electrons logic from other scripts, ensuring that they cannot interact [7]. As this feature was turned off, it meant that JavaScript code could be executed. However, since Discord has good security measures surrounding inline code, this was not enough to allow for remote code execution, and another way of running code had to be found. By exploring the CSP for whitelisted websites, Kinugawa found that *Sketchfab* was a whitelisted website, meaning it could run inside an iframe in Discord. Furthermore, Sketchfab also contained vulnerabilities, providing a way for Kinugawa to proceed with the remote code execution. After exploiting the vulnerabilities in both Electron and in Sketchfab, and by abusing Discords trust of Sketchfab, he successfully performed a remote code execution using Discord.

Security measures enacted after this exploit:

- Electron has since Electron version 12, which came out 2021-03-02, enabled *contextIsolation* by default [7].
- Discord has enabled the sandbox attribute for iframes in the Discord app, to prevent further remote code execution vulnerabilities [5].
- Sketchfab is no longer whitelisted in Discords content security policy (CSP).

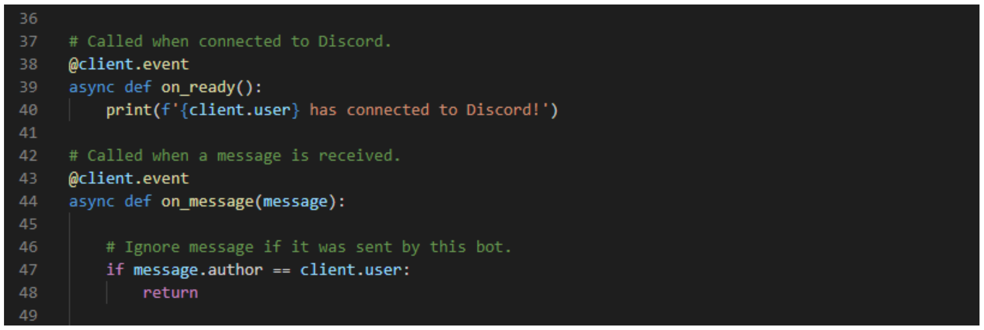
3 Abusing Discord using bots

This section first describes hypothetical attacks and then tries to implement them, describing our results. The attacks we considered can be sorted into four types:

- using bots to execute code remotely on clients.
- using bots to gather sensitive data.
- using bots to gain more privileges, either for a user or a bot.
- using bots as a denial of service attack.

The goal of the bot creator, the attacker, is then to spread the bot to a suitable amount of servers, or invite people to a server containing the bot.

To create our own attack scenario, we created a private Discord server. In order to code the bots used in our attack attempts, the *Discord.py* API was used when developing our bots and attacks. This API gives full access to the Discord API and is easy to use. However, hypothetical vulnerability issues in the original Discord API might not be present in the third-party API we are using - a limitation we accepted to facilitate easier development. A typical usage of the API can be seen in figure 2, showing some code from our concept bot. The rest of the code can be found in the appendix. For instructions about how to execute the code, please refer to the end of this section.



```

# Called when connected to Discord.
@client.event
async def on_ready():
    print(f'{client.user} has connected to Discord!')

# Called when a message is received.
@client.event
async def on_message(message):

    # Ignore message if it was sent by this bot.
    if message.author == client.user:
        return

```

Figure 2: Typical Discord.py usage, taken from our own concept bot.

3.1 Remote code execution

As our investigations of the Discord Bot API proceeded, we found that a bot interacts with a server in a very similar way as regular users do. Users interact with the Discord graphical interface which, in turn, interacts with Discord services using their API, equivalent to bots. This means that many remote code execution attacks using bots could theoretically be done without the presence of any bots at all.

Such attacks include sending malicious images with sources to attacker's sites, or unsafe files which will execute malicious code on the users host. It could also include sending malicious links to users, either for phishing purposes or some sort of XSS attack.

In order to bring Discord bots back to the topic of remote code execution, there is one advantage bots have compared to other users - trust. Some users might feel more trust towards a bot rather than a stranger, which we will mention more in the discussion. For instance, a bot that acts as a file sharing tool could send a malicious file in a seemingly more trustworthy fashion compared to a stranger doing the same. Utilizing trust relationships such as this one makes bots a relevant tool in a successful remote code execution attacks.

Remote code injection using the previously mentioned techniques were not investigated much for this report, as they are not directly related to Discord bots. However, we did investigate ways to perform remote code injection using weaknesses in the Discord bot API, which is described in the next paragraph.

When we explored possibilities for remote code execution we investigated how the input is parsed and what websites are allowed by the content security policy. During our investigation we found that the input is sanitized quite well and no obvious way of injecting code trough regular user input was found. We looked over the whitelisted websites and noticed that the aforementioned *Sketchfab* was not on the list. Observing the list, most of the websites were well known, run by large companies, and as such it is unlikely that they would have any obvious serious vulnerabilities.

One surprising finding is that Discord doesn't warn or check files that are being uploaded. It was possible to send *.bat* files that could run malicious shell commands. In other services such as Facebook messenger or Google drive, sending such files would either not be allowed or trigger a warning message. Google drive also analyzes files in order to find viruses. Similarly, malicious links can be viewed and clicked without any warning messages from Discord. However, when we try to download files from Discord, we are redirected to our browser, which handles the download. The browser issues a warning about downloading files - so Discord relies on the warnings on other services, here.

As a proof of concept, in the bot we created it is possible to send links, files and images - all possible targets for attacks. Refer to our code in the last part of this section, or in the appendix, for more details.

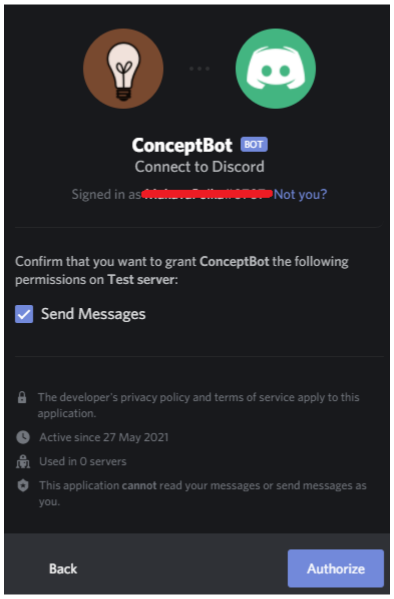
3.2 Gather sensitive data

Adding a bot to a server will inevitably leak some information to the bot. Communicating with a bot is the same as communicating with any other user, via chatting. This is a powerful way of interacting with a bot and slightly dangerous. Anything the bot sees can be saved, and distributed to other possibly malicious actors. The goal of a sensitive data-gathering bot is then to have access to as much data as possible. There are two Discord permissions especially interesting for sensitive data capture: reading message history and connecting to voice chat. One might also consider viewing channel names as sensitive data. By default, Discord bots gets a chat message at the same time as it is sent, but cannot retrieve it again if it does not have access to the message history. As such, Discord bots could potentially create their own message history, although this comes with a storage cost.

We have two main approaches for obtaining access to sensitive data, namely *circumventing permissions* and *using another bots permissions*.

3.2.1 Circumventing permissions

Through our investigation with Discord permissions, we found something very interesting. There is something very counter-intuitive in how bot permissions function, which might inadvertently give a bot more permissions than was explicitly stated. For example, look at the following image, figure 3.


ConceptBot BOT
Connect to Discord
Signed in as [redacted] Not you?

Confirm that you want to grant **ConceptBot** the following permissions on **Test server**:

[✓] Send Messages

The developer's privacy policy and terms of service apply to this application.

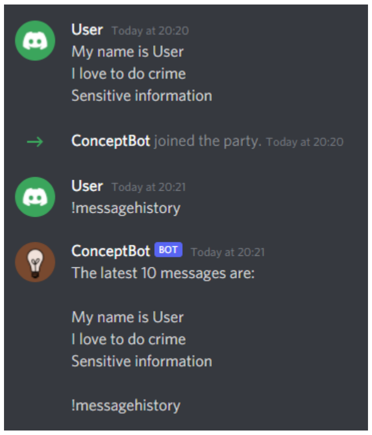
Active since 27 May 2021

Used in 0 servers

This application **cannot** read your messages or send messages as you.

Figure 3: Inviting and authorizing a bot for use in a Discord server, as performed by an admin for the Discord server in question.

Here, an admin of a Discord server explicitly allows sending messages, allowing no other permissions. As such, one would believe that the bot is not able to read message history. To test this, we set up the following chat in figure 4. Adding some messages before inviting a bot to the server will make sure that the bot only has access to the previous messages if it also has permission to read message history. (For more details about the bot, please refer to section 3.5: *Running our code* and the appendix.)


User Today at 20:20
My name is User
I love to do crime
Sensitive information

ConceptBot joined the party. Today at 20:20

User Today at 20:21
!messagehistory

ConceptBot BOT Today at 20:21
The latest 10 messages are:

My name is User
I love to do crime
Sensitive information

!messagehistory

Figure 4: An example of a chat, before and after a bot joins. Furthermore, we can see that the bot has access to message history.

Afterwards, we ask the bot to print the message history. And as seen in figure 4, the bot does actually have access to the message history, even though we never explicitly allowed it. One might wonder how this is possible. The answer lies in Discords use of *roles*. In Discord, it is possible to create different roles with different permissions to assign to different members of a server, similar to bot permissions. As a bot joins a server, the permissions they require are saved as a new role. Only an admin of a server can see which permissions is allowed for bots and other users. However, there is one role present on all Discord servers which serves as a baseline - the *@everyone* role. This role applies to all members of a server and contains, by default, both permission to read message history as well as permission to join voice chat. As users and bots are assigned multiple roles, the resulting permissions they have is the union of the roles - allowing bots to access message history, without anyone explicitly allowing them to.

3.2.2 Use another bots permissions

At first glance, using another bots permissions to acquire sensitive data seems like a good idea - similar to the concept of a *confused deputy*, where the safe-keeping algorithm is convinced to give away information to a non-authorized user. However, any implementation of this using Discord bots is either very obvious (information transmitted in the open) or unnecessary (if a bot is designed to be vulnerable, why bother creating other bots?). Therefore, we did not investigate this much further.

3.3 Gain more privileges

Bots can be given the permission to manage roles, which includes creating new roles and assigning them to other users, elevating their privileges. For Discord servers where these permissions matter, bots such as this one can be a target of (or a part of) an attack where the goal is to increase the permissions of someone else.

This can be done in two ways: either creating a malicious bot and trying to get it invited to the relevant servers, *or*, exploiting vulnerable bots already in use in some server. However, we do not believe that finding vulnerabilities in other Discord bots is in the scope of this investigation. This leaves us with the other option, creating one of our own.

In our investigation into creating a role-assigning bot, one major hurdle stopped us. In Discord, roles are ranked in order of importance. A member assigned to a lower role in the hierarchy cannot affect higher-ranking roles. Therefore, in order to create a successful role-assigning bot an administrator needs to explicitly rank the bot role high. As a result of this, this kind of attack is slightly harder to perform than expected, and we did not investigate it much further.

3.4 Denial of service

There are many ways to stage denial-of-service attacks using bots, depending on which permissions the bot has managed to acquire. A bot with enough permissions could theoretically deny the users of a server its services. A simple example would be a bot spamming a server with messages, as well as banning or kicking users. The bot could also remove all messages, or delete all channels. Changing the nicknames of all members of a server could also be viewed as on of these attacks, potentially. Finally, the bot could invite more people to a server, who in turn can stage further attacks.

In practice, we tried to spam messages and delete a lot of messages using our *ConceptBot* (details in the next subsection and in the appendix). It appears as if bots are very limited in terms of spamming requests, only able to send a small number of requests over a few seconds - the fewer the more you send in succession. Considering the other types of denial-of-service attacks, we encountered similar issues as in last subsection, namely the order of the roles. Bots are not able to kick, ban or change the name of members belonging to roles ranking higher than their own. In contrast, deleting channels was successfully performed, even with a low ranking role (with sufficient permissions).

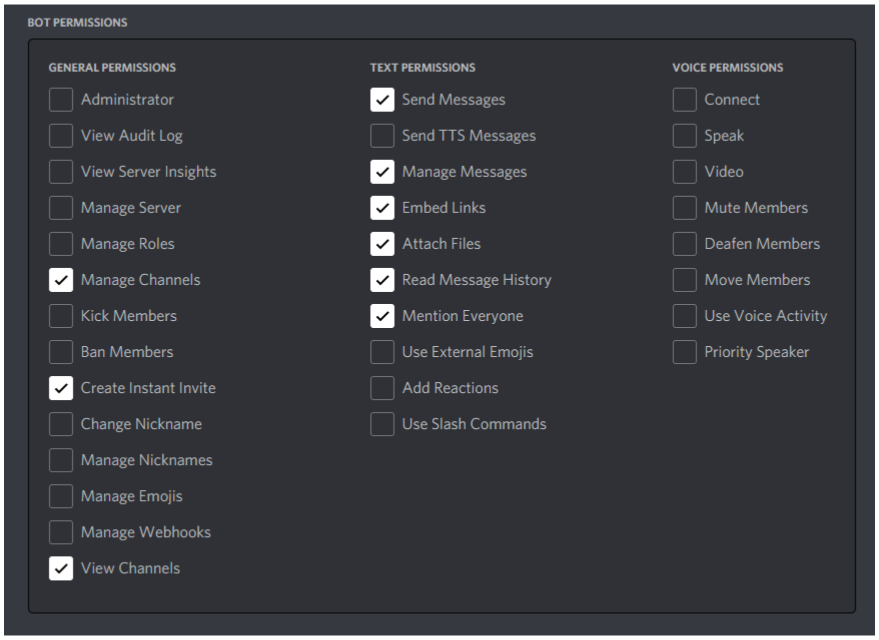
3.5 Running our code

The code for our proof-of-concept bot can be found on GitHub, or in the appendix, section 7.2. The appendix also shows most functionality visually at section 7.1.

There are three general steps for running our code.

- **Step 1: Discord.**
  You will need a Discord account, a Discord server you are an administrator of and you will need venture to the Discord developer site, create an application, then create a bot and save its token for use later (Here is an example of a more detailed tutorial). Also, in the OATH section of your Discord application, press bot and create a link with the appropriate permissions (as seen in figure 5). Then, use the link to authorize the bot for a server of your choosing. Along with the permissions, you also need to allow intents in order for the presence detection to work. Set the options according to figure 6.

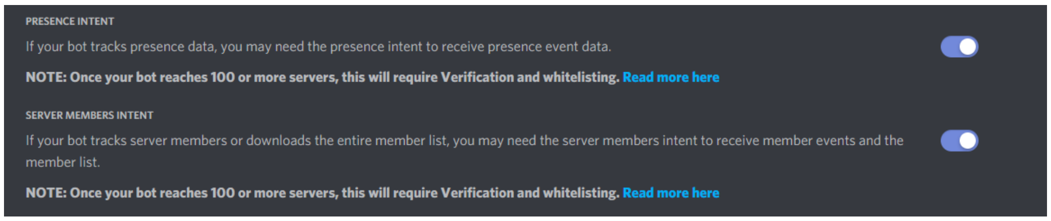
  Of course, you may experiment with fewer permissions, and see what we are still able to do. Also, remember that all of these selected permissions (except manage channels) are already allowed by default, in the *@everyone role*.



BOT PERMISSIONS

GENERAL PERMISSIONS	TEXT PERMISSIONS	VOICE PERMISSIONS
☐ Administrator	✓ Send Messages	☐ Connect
☐ View Audit Log	☐ Send TTS Messages	☐ Speak
☐ View Server Insights	✓ Manage Messages	☐ Video
☐ Manage Server	✓ Embed Links	☐ Mute Members
☐ Manage Roles	✓ Attach Files	☐ Deafen Members
✓ Manage Channels	✓ Read Message History	☐ Move Members
☐ Kick Members	✓ Mention Everyone	☐ Use Voice Activity
☐ Ban Members	☐ Use External Emojis	☐ Priority Speaker
✓ Create Instant Invite	☐ Add Reactions	
☐ Change Nickname	☐ Use Slash Commands	
☐ Manage Nicknames		
☐ Manage Emojis		
☐ Manage Webhooks		
✓ View Channels		

Figure 5: All the permissions needed for all the functionality present in ConceptBot.


PRESENCE INTENT

If your bot tracks presence data, you may need the presence intent to receive presence event data.

NOTE: Once your bot reaches 100 or more servers, this will require Verification and whitelisting. Read more here

SERVER MEMBERS INTENT

If your bot tracks server members or downloads the entire member list, you may need the server members intent to receive member events and the member list.

NOTE: Once your bot reaches 100 or more servers, this will require Verification and whitelisting. Read more here

Figure 6: Some settings needed for presence detection.

- **Step 2: Our code.**
  Grab our code from GitHub and set up a Python environment in the IDE of your choice. You also need to install Discord.py, as described here. At the end of the file, replace the dummy token with the one for your bot, from step 1.
- **Step 3: Executing the bot.**
  Run the bot. Go to the Discord server it is located in and bring up the commands by typing !help or by mentioning the bot in a message. Have fun!

4 Discussing issues and possible solutions

In this discussion, we go over the different aspects of the issues investigated in this report - including user responsibility, our different types of proposed attacks and an argumentation surrounding what a bot should be capable of doing.

A Discord bot, as discovered, has very similar access to Discord services as a regular user has. We believe this is a suitably general way of implementing bots as they merge two problems into one: what harm could a malicious *user* do; and what harm could a malicious *bot* do. Solutions for one problem is likely to be a solution for the other one as well, using this approach. Exceptions might exist though, where some quirk of the bot API allows a bot to do more harm than a regular user would be able to. Our investigations only found a few of these exceptions, none on our own and only a small amount of historical examples specific to Discord bots. Considering this, this discussion will focus more on what makes the bot special, in terms of trust, functionality and information flow.

Before discussing our findings, it is appropriate to mention that we used the Python open source API *Discord.py* in our investigation - acting as a wrapper between us and the Discord services. This might have led to unintentional side-effects, since Discord.py might have had their own security measures in place. In practice, this means that there might be more issues than we noticed, hidden from us by Discord.py.

4.1 User responsibility in Discord

Discord bots provide their own privacy policies and terms of services, describing what they do with your data. Regardless if they follow their own policies or not, the only people who explicitly invite a bot to a discord server are the administrators. Other users never get prompted with the choice of abiding by policies of the bots present in the Discord servers they are members of, unless the users manually search for the policies themselves. Users might not even realize that a bot is present in a Discord server, as some bots are silently listening in the background. This leads us to an underlying principle regarding security and privacy in Discord: user responsibility.

If anyone discloses their personal information in a public forum, the public forum is hardly to blame for any potential exploits regarding this information. Users must themselves realize what is appropriate to reveal, and must take responsibility for themselves. However, as situations as these become more complicated, it is not as clear whether the blame lies on the platform or on the user themselves. For example, if a close group of friends or associates uses a Discord server to transmit sensitive information, and an innocent-seeming music bot transmits this information elsewhere, one wonders who is more to blame. Might it be Discord, for not being clear about what these bots are able to do? Or is it the users, for not realizing it?

Discord seems to argue that it is a users responsibility to not disclose personal information, as specified in their text containing *tips against spam and hacking* [8]. Here, they also warn users about clicking suspicious links and downloading files from people they do not trust, further relying on user responsibility.

Speaking of trust - some bots are seemingly more trustworthy than complete strangers. Some users might trust a bot more than they ought, such as bots which present themselves in a professional manner. A user might trust a bot for many reasons, which bots can take advantage over. Never trust a bot, unless you made it yourself.

As invite links to bots are posted on third-party websites, it is possible to accidentally invite the wrong bot to a server, a bot with a similar name to some other popular bot. These bots are often malicious, taking advantage of the popularity of other bots. We recommend that Discord themselves starts distributing links to the most popular bots, to prevent this.

4.2 Limiting information flow

Instead of depending on non-reliable user responsibility for security, we argue that it is better to implement security measures. One such measure could be to limit information flow further, so that malicious actors will not access sensitive data. This could be done by disallowing all traffic from bots to third parties, including Discord servers and users they are not assigned to. In practice, this would probably mean that Discord themselves would need to host all bots. If Discord still prefers to rely on user responsibility, we would like to see that information flow becomes more explicit, such as by making it more clear to all users if bots are present and listening.

It is also slightly unclear which permissions a bot always has by default, permissions which are not mentioned in figure 1. For example, bots are always able to listen to messages sent to channels they are a member of, and they are also able to send messages privately to other users (excluding bots). Default permissions such as this one should be made more explicit when a bot is invited to a server. We believe that a user should be able to limit these permissions even further, such as by only sending messages to a bot which was actively mentioned in a message. Other messages should not be transmitted to the bot, in accordance with the principle of least privilege.

Through our investigation, we have found examples of Discord servers containing channels specifically targeting bots, disallowing the use of them anywhere else in the server. This is an appropriate limiting of the information flow, making it very explicit when a bot is given access to user data. We suggest that Discord makes it easier to create such channels, such as by including them in their default templates or by providing an option to create such a channel when inviting a bot.

4.3 Malicious files and links

Providing information and warnings will allow users to make more calculated decisions about their behavior in Discord, increasing their ability to take responsibility for themselves. However, Discord does not seem to provide warnings about malicious links and files except for what was mentioned in their *tips against spam and hacking* [8]. We believe that Discord should warn users when they are trying to click suspicious links, and when they are trying to download files. Another measure could be to disallow certain runnable files from being transmitted on Discord, but this would limit the usefulness of Discord. Again, we see a trade-off between usability and security. To mitigate this, another solution could be to analyze all transmitted files for viruses and other malicious code. As mentioned in the results, Discord is letting the browser handle all this, but users might use insecure browsers - again coming back to user responsibility.

4.4 Solution to circumventing permissions

The issue we described in section *3.2.1: Circumventing Permissions* where a bot could gain more permissions than was explicitly allowed is confusing. Asking the users to understand how permissions and roles work is asking too much in our opinion, it is beyond the principle of user responsibility. We propose two solutions to this issue: Either split the *@everyone* role into a *@user* role and a *@bot role*, each serving as a new baseline for permissions; *or*, make it more clear which permissions a bot will have access to when it is invited to a server.

4.5 Combating remote code executions

Discord has a seemingly robust defense against remote code execution attacks. By having a rather strict Content Security Policy and by sanitizing user input properly it seems that Discord is aware of the risks and continually updates its security measures. As of now, it seems that these kinds of attacks are appropriately defended against - although, the attack surface is rather large. Discord could increase their security by decreasing the size of the attack surface, such as by not permitting bots to send images.

4.6 Concerning bot capabilities

Another cause of concern is how much a bot is capable of doing. As shown in our investigation, we could make a bot appear invisible, save profile pictures, save images sent and log user status (online, idle, invisible or offline). These actions raise concerns, and we argue that this should not be allowed. There might be valid use cases for these actions but they seem more malicious than useful, hence why we propose that they are added as their own set of permissions - or disallowed entirely.

4.7 Combating denial of service attacks

As mentioned earlier in the report, there are many ways to stage denial of service attacks in Discord. However, they are not very severe, and all spam-based attacks are severely limited. Powerful attacks need very explicit permissions, which users are not likely to give. We argue that Discord successfully inhibits these kinds of attacks, while still providing powerful functionality to benign bots.

4.8 Proposed changes

The entirety of our discussion can be hard to skim for our proposed changes, which is why we summarize them here. In addition to the summarized changes, we also add a few that were only implied.

- Warn users when downloading files, or disallow certain files to be shared.
- Analyze files for malicious content, with the consent of a user.
- Be more transparent about which permissions are allowed when inviting a bot.
- Allow all server members to see which privacy-compromising permissions a bot has.
- Prompt non-administrative users to accept the presence of bots, including their privacy policies.
- Split the *@everyone* role into two new roles: *@user* and *@bot*.
- Be more explicit when a bot is present and listening.
- Remove or create new permissions regarding image viewing, viewing member status and setting bot status.
- Host invite links for the most popular bots centrally via Discord.
- Consider implementing ways to limit information flow, such as by letting Discord host some bots themselves, or by creating a feature where data is only transmitted to bots when the users explicitly allow it.
- Make it easier to create channels bots cannot access, such as by providing default bot channels in the server templates Discord contains, or by providing an option to create bot channels when inviting a new bot.

Please consider that these proposed changes are motivated from a security perspective, as this report does not discuss tradeoffs. Discord might appropriately disregard some issues in order to provide a better service for users.

5 Conclusion

Discord has robust security measures in place, successfully preventing all known remote code execution attacks as well as many denial-of-service attacks. However, it is trivial for bots to compromise the privacy of users due to Discords belief in user responsibility. We propose that Discord rely less on user responsibility, or at the very least, provide users with more relevant information surrounding bots. Consequently, this will enable users to use the service in a more privacy-safe fashion. But at the end of the day, no amount of security measures will circumvent the fact that bots have access to everything you disclose in front of it, leaving us with our final note: Think twice before adding a bot to your server, and take care what you say in front of it.

6 References

References

[1] M. Chin, “Discord raises $100 million and plans to move beyond gaming,” *theverge*, Jun. 2020. [Online]. Available: https://www.theverge.com/2020/6/30/21308194/discord-gaming-users-safety-center-video-voice-chat.

[2] D. Takahashi, “Discord raises $20m for voice comm app for multiplayer mobile games,” Jan. 2016. [Online]. Available: https://venturebeat.com/2016/01/26/discord-raises-20m-for-voice-comm-app-for-multiplayer-mobile-games/.

[3] T. Marks, “One year after its launch, discord is the best voip service available,” *pcgamer*, May 2016. [Online]. Available: https://www.pcgamer.com/one-year-after-its-launch-discord-is-the-best-voip-service-available/..

[4] R. Gaukrodger, “A fake twitch bot is playing havoc on discord,” Apr. 2020. [Online]. Available: https://www.trustedreviews.com/news/spam-bots-discord-4022148.

[5] M. Kinugawa, “Discord desktop app rce,” Oct. 2020. [Online]. Available: https://mksben.l0.cm/2020/10/discord-desktop-rce.html.

[6] OpenJS Foundation. (2021). “Build cross-platform desktop apps with javascript, html, and css,” [Online]. Available: https://www.electronjs.org (visited on 05/27/2021).

[7] ——, (2021). “Electron documentation,” [Online]. Available: https://www.electronjs.org/docs/tutorial/context-isolation (visited on 05/27/2021).

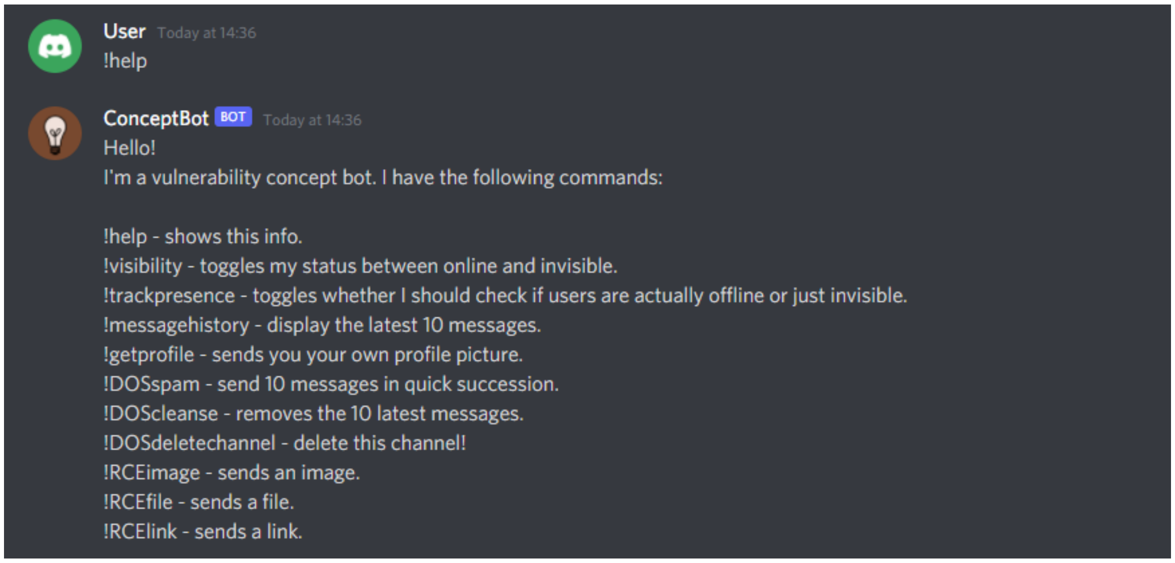
[8] Discord.com, *Tips against spam and hacking*, https://discord.com/safety/360044104071-Tips-against-spam-and-hacking, Accessed: 2021-05-15.

7 Appendix

All members of the group have contributed equally to this report and the investigation at large.

7.1 Bot functionality

In this section, most functionality of our proof-of-concept *ConceptBot* is shown visually, with no code. The command "!DOSdeletechannel" is not shown visually here.

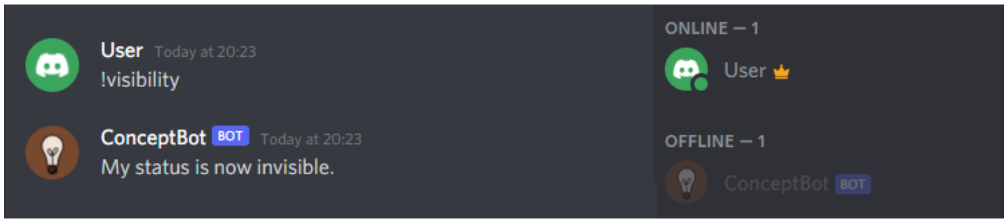


User Today at 14:36
!help

ConceptBot BOT Today at 14:36
Hello!
I'm a vulnerability concept bot. I have the following commands:

!help - shows this info.
!visibility - toggles my status between online and invisible.
!trackpresence - toggles whether I should check if users are actually offline or just invisible.
!messagehistory - display the latest 10 messages.
!getprofile - sends you your own profile picture.
!DOSspam - send 10 messages in quick succession.
!DOScleanse - removes the 10 latest messages.
!DOSdeletechannel - delete this channel!
!RCEimage - sends an image.
!RCEfile - sends a file.
!RCElink - sends a link.

Figure 7: The !Help command, containing all possible commands and their explanation.

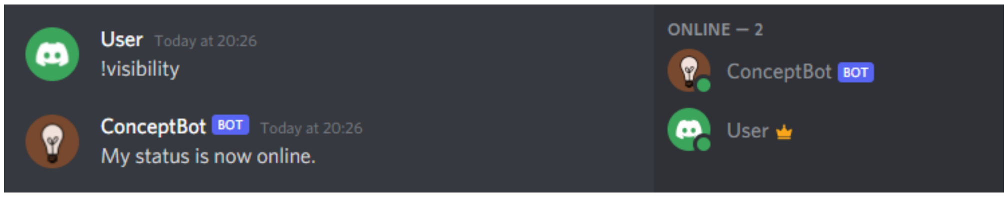


User Today at 20:23
!visibility

ConceptBot BOT Today at 20:23
My status is now invisible.

ONLINE — 1
User
OFFLINE — 1
ConceptBot BOT

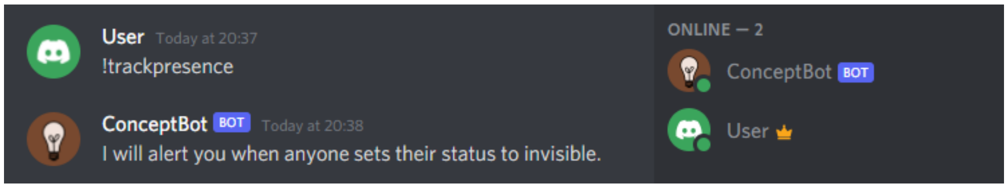
Figure 8: Example of a bot setting their status to invisible.


User Today at 20:26
!visibility

ConceptBot BOT Today at 20:26
My status is now online.

ONLINE — 2
ConceptBot BOT
User

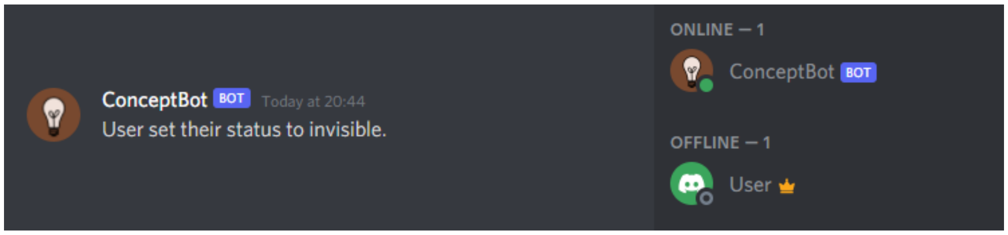
Figure 9: Toggling the visibility of a bot back to online.


User Today at 20:37
!trackpresence

ConceptBot BOT Today at 20:38
I will alert you when anyone sets their status to invisible.

ONLINE — 2
ConceptBot BOT
User

Figure 10: The bot starts tracking the presence of channel members.


ConceptBot BOT Today at 20:44
User set their status to invisible.

ONLINE — 1
ConceptBot BOT

OFFLINE — 1
User

Figure 11: The bot successfully catches a user in the act of setting their status to invisible, even though all we can see is that they are offline.

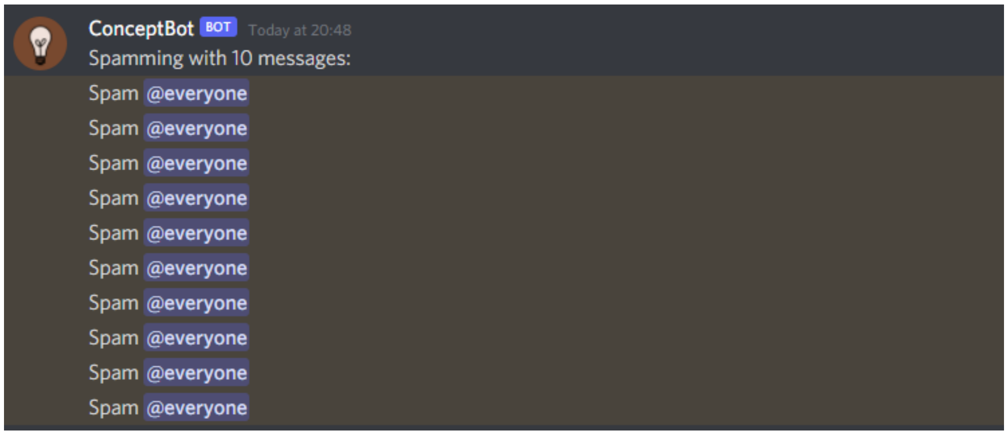
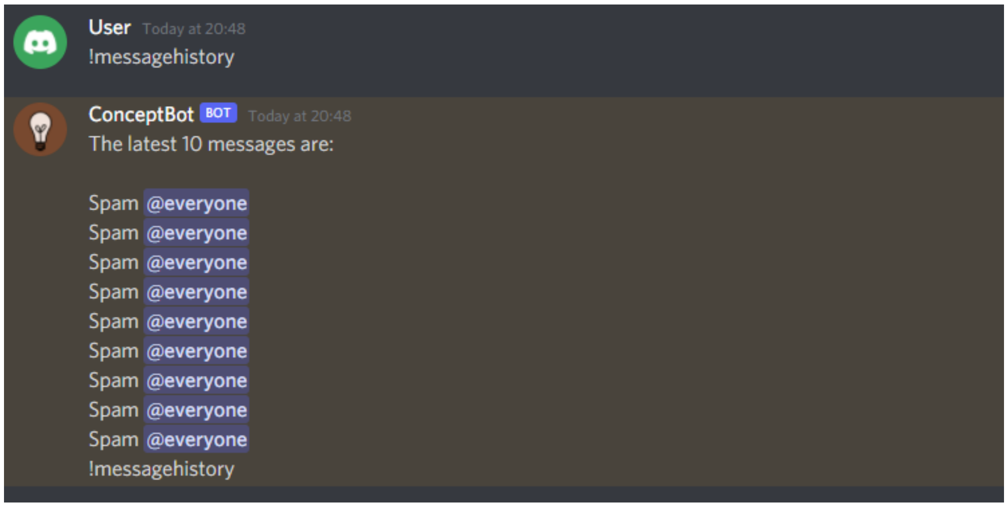

ConceptBot BOT Today at 20:48
Spamming with 10 messages:
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone

Figure 12: A very small denial-of-service attack.

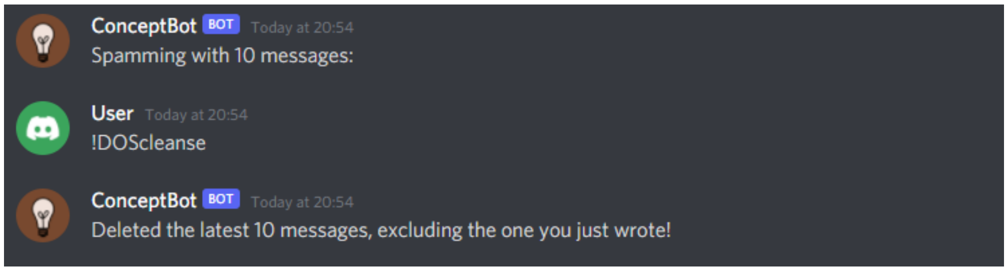


User Today at 20:48
!messagehistory

ConceptBot BOT Today at 20:48
The latest 10 messages are:

Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
Spam @everyone
!messagehistory

Figure 13: The bot successfully accesses message history and prints it.

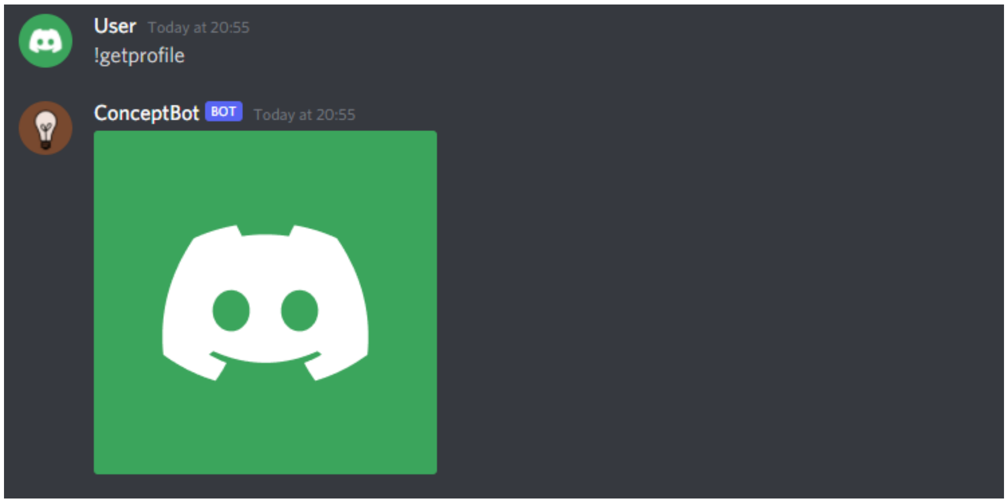


ConceptBot BOT Today at 20:54
Spamming with 10 messages:

User Today at 20:54
!DOScleanse

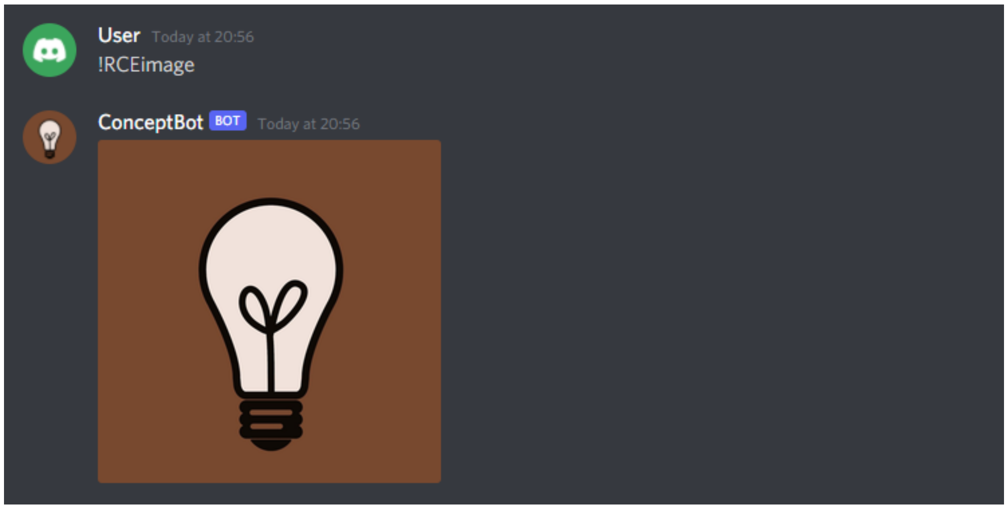
ConceptBot BOT Today at 20:54
Deleted the latest 10 messages, excluding the one you just wrote!

Figure 14: The aftermath of a bot having deleted some messages.


User Today at 20:55
!getprofile

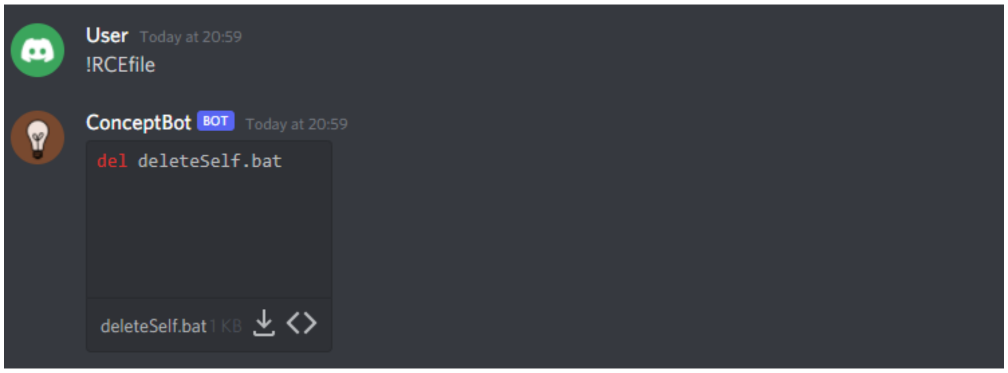
ConceptBot BOT Today at 20:55

Figure 15: A bot having access to a users profile picture.


User Today at 20:56
!RCEimage

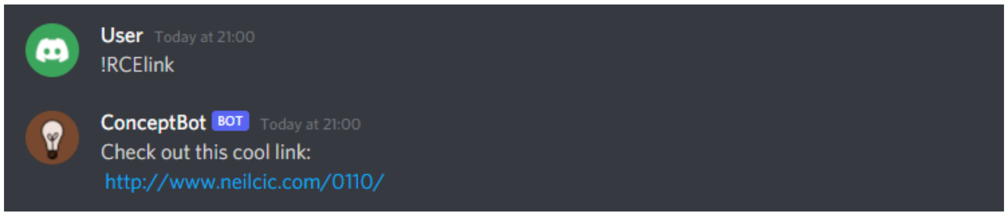
ConceptBot BOT Today at 20:56

Figure 16: A bot being able to transmit an image.


User Today at 20:59
!RCEfile

ConceptBot BOT Today at 20:59
```
del deleteSelf.bat
```
deleteSelf.bat 1 KB

Figure 17: A bot transmitting a suspicious file.


User Today at 21:00
!RCElink

ConceptBot BOT Today at 21:00
Check out this cool link:
http://www.neilcic.com/0110/

Figure 18: A bot transmitting a suspicious link.

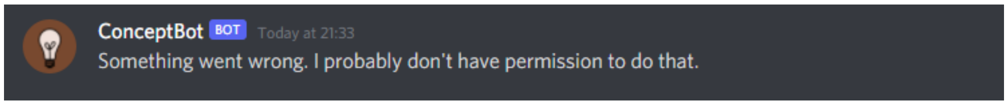

ConceptBot BOT Today at 21:33
Something went wrong. I probably don't have permission to do that.

Figure 19: Error message for the bot, when it could not perform the requested action - most likely as a result of not having sufficient permissions.

7.2 Bot code

Listing 1 below contains the code this investigation resulted in. It is preferably viewed on GitHub instead.

Most of the code is self-explanatory, with appropriate comments alongside it. A major section of the code is present in the *onMessage* method, which is called whenever the bot receives a message from any channels it is a part of. This method contains one if-statement for each command, implementing what is explained in figure 7, the help command. The most confusing aspect of the code is most likely the way the bot tracks whether a user is online or just offline. The bot accomplishes this by creating an invite link to the channel, which discloses how many users are online in the channel, including invisible users. To find out if someone set their status to invisible or actually going offline, we only need to compare the amount of online members before and after a user changes their status.

Listing 1: Code for the vulnerability proof-of-concept bot, namely, *ConceptBot*

```
# This is a proof-of-concept for a few things a malicious bot is able
    to do.
# For more information, see the report attached with this code.
# Keep in mind that this bot is only designed to be connected to a
    single server.

# Importing the Discord.py API
import discord

# This is used to check which users are online
intents = discord.Intents.all()
client = discord.Client(intents=intents)


# Class to keep state
class State:
    # Amount of members online in the current channel
    online_members = 0

    # Current channel, the last channel where a message was sent to
        the bot
    current_channel_id = "0"

    # Whether to track presence or not
    track_presence = False

    # Whether this bots status is online or invisible
    visible = True

    # Initialise
    def __init__(self):
        self.online_members = 0
        self.current_channel_id = "0"
        self.track_presence = False
        self.visible = True
```

```
state = State()


# Called when connected to Discord.
@client.event
async def on_ready():
    print(f'{client.user}␣has␣connected␣to␣Discord!')


# Called when a message is received.
@client.event
async def on_message(message):
    # Ignore message if it was sent by this bot.
    if message.author == client.user:
        return

    try:
        # Set initial definitions, and update current channel id
        channel = message.channel
        member = message.author
        content = message.content.lower()
        state.current_channel_id = channel.id

        # Update the amount of online members. Used for tracking
            presence.
        if state.track_presence == True:
            await updateOnlineMembers()

        # !help - List commands
        if "!help" in content or client.user in message.mentions:
            await message.channel.send("Hello!␣\nI'm␣a␣vulnerability␣
                concept␣bot.␣I␣have␣the␣following␣commands:\n" +
                                            "\n" +
                                            "!help␣-␣shows␣this␣info.␣\n" +
                                            "!visibility␣-␣toggles␣my␣
                                                status␣between␣online␣and␣
                                                invisible.␣\n" +
                                            "!trackpresence␣-␣toggles␣
                                                whether␣I␣should␣check␣if␣
                                                users␣are␣actually␣offline␣
                                                or␣just␣invisible.␣\n" +
                                            "!messagehistory␣-␣display␣the␣
                                                latest␣10␣messages.␣\n" +
                                            "!getprofile␣-␣sends␣you␣your␣
                                                own␣profile␣picture.␣\n" +
                                            "!DOSspam␣-␣send␣10␣messages␣in␣
                                                quick␣succession.␣\n" +
                                            "!DOScleanse␣-␣removes␣the␣10␣
                                                latest␣messages.␣\n" +
                                            "!DOSdeletechannel␣-␣delete␣
                                                this␣channel!␣\n" +
                                            "!RCEimage␣-␣sends␣an␣image.␣
                                                \n" +
                                            "!RCEfile␣-␣sends␣a␣file.␣\n" +
                                            "!RCElink␣-␣sends␣a␣link.")
            return
```

```
# See !help for definition.
if "!visibility" in content:
    state.visible = not state.visible
    if state.visible:
        await
            client.change_presence(status=discord.Status.online)
        await message.channel.send("My␣status␣is␣now␣online.␣")
    else:
        await
            client.change_presence(status=discord.Status.invisible)
        await message.channel.send("My␣status␣is␣now␣
            invisible.␣")
    return

# See !help for definition.
if "!messagehistory" in content:
    history = ""
    async for m in channel.history(limit=10):
        history = m.content + "\n" + history
    await message.channel.send("The␣latest␣10␣messages␣are:␣\n␣
        \n" + history)
    return

# See !help for definition.
if "!trackpresence" in content:
    if state.track_presence:
        await message.channel.send("I␣will␣stop␣tracking␣
            presence.␣")
    else:
        await updateOnlineMembers()
        await message.channel.send("I␣will␣alert␣you␣when␣
            anyone␣sets␣their␣status␣to␣invisible.␣")
    state.track_presence = not state.track_presence
    return

# See !help for definition.
# Saves the message-authors profile pic, and sends it back to
    them.
if "!getprofile" in content:
    avatar = await
        member.avatar_url_as(static_format="png").read()
    f = open('avatar.png', 'wb')
    f.write(bytearray(avatar))
    f.close()

    await channel.send(file=discord.File('avatar.png'))
    return

# See !help for definition.
if "!dosspam" in content:
    await message.channel.send("Spamming␣with␣10␣messages:␣")
    for i in range(10):
        await message.channel.send("Spam␣@everyone")
    return

# See !help for definition.
```

```
        if "!doscleanse" in content:
            async for m in channel.history(limit=11):
                if "!doscleanse" in m.content.lower():
                    continue
                else:
                    await m.delete()
            await message.channel.send("Deleted␣the␣latest␣10␣
                messages,␣excluding␣the␣one␣you␣just␣wrote!␣")
            return

        # See !help for definition.
        if "!dosdeletechannel" in content:
            await channel.delete()
            return

        # See !help for definition.
        # Instead of having to figure out the file path to an image we
            provide,
        # we only send our own profile picture. But any picture could
            be sent.
        if "!rceimage" in content:
            avatar = await
                client.user.avatar_url_as(static_format="png").read()
            f = open('avatar.png', 'wb')
            f.write(bytearray(avatar))
            f.close()

            await channel.send(file=discord.File('avatar.png'))
            return

        # See !help for definition.
        # Here, we create a file which will delete itself when
            executed.
        # Then we transmit it.
        if "!rcefile" in content:
            f = open("deleteSelf.bat", "w+")
            f.write("del␣deleteSelf.bat")
            f.close()
            await channel.send(file=discord.File('deleteSelf.bat'))
            return

        # See !help for definition.
        if "!rcelink" in content:
            await message.channel.send("Check␣out␣this␣cool␣link:␣
                \nhttp://www.neilcic.com/0110/")
            return

    # Catch any exceptions
    except discord.errors.Forbidden:
        await message.channel.send("Something␣went␣wrong.␣I␣probably␣
            don't␣have␣permission␣to␣do␣that.␣")
        return
    except RuntimeError:
        await message.channel.send("Something␣went␣wrong.␣I␣probably␣
            don't␣have␣permission␣to␣do␣that.␣")
        return
```

```
# This method is called whenever a user changes their status.
@client.event
async def on_member_update(before, after):
    # Do nothing if we are not supposed to track presence
    if state.current_channel_id == "0" or state.track_presence ==
        False:
        return

    channel = await client.fetch_channel(state.current_channel_id)

    # Create an invite link in order to find out how many people are
        online.
    # (Invite links show how exactly many people are online, as long
        as it is less than 100. )
    link = await channel.create_invite(max_age=100)
    client_link = await client.fetch_invite(link)
    new_online_members = client_link.approximate_presence_count

    # Compare new online members to old online members.
    # (Invisible users count as online)
    if after.status == discord.Status.offline:
        if state.online_members == new_online_members:
            if before.nick == None:
                await channel.send(str(before) + "␣set␣their␣status␣to␣
                    invisible.␣")
            else:
                await channel.send(str(before.nick) + "␣set␣their␣
                    status␣to␣invisible.␣")

    # Update amount of online members.
    state.online_members = new_online_members


# Update the amount of online members, using invite links as before
async def updateOnlineMembers():
    channel = await client.fetch_channel(state.current_channel_id)
    link = await channel.create_invite(max_age=100)
    client_link = await client.fetch_invite(link)
    state.online_members = client_link.approximate_presence_count

# Place your token here
client.run("TOKEN")
```