RAVEN STEALER UNMASKED: Telegram-Based Data Exfiltration. - CYFIRMA

Archived: 2026-04-05 21:31:01 UTC

Published On : 2025-07-26

EXECUTIVE SUMMARY

Raven Stealer is a modern, lightweight, information-stealing malware developed primarily in Delphi and C++, designed to extract sensitive data from victim machines with minimal user interaction and high operational stealth. Promoted under the guise of “educational use,” Raven Stealer demonstrates functionality consistent with malicious intent, including credential theft, browser data harvesting, and real-time data exfiltration via Telegram bot integration.

The stealer specifically targets Chromium-based browsers (such as Chrome and Edge), extracting passwords, cookies, saving payment details, and autofill information. Its use of a modular architecture and a built-in resource editor allows attackers to embed configuration details (like Telegram bot tokens and chat IDs) directly into the compiled payload. This structure streamlines the deployment process, enabling even low-skill threat actors to launch credential-harvesting campaigns.

Unlike Python-based stealers, Raven's compiled binaries are packed using UPX, reducing their size and improving evasion against static detection mechanisms. The malware executes in a fully hidden state, leaving no visible traces or UI elements during runtime, which significantly lowers the chance of user detection.

Raven Stealer is actively distributed through GitHub repositories and promoted via the Telegram channel. This channel functions as both a development log and distribution platform, regularly sharing builder updates, guides, and promotional content for the stealer. The use of Telegram for C2-like behavior, paired with a clean user interface and dynamic module support, positions Raven Stealer as a commercially attractive tool within the commodity malware ecosystem.

INTRODUCTION

The rise of modular, easy-to-deploy information stealers reflects a growing trend in the cybercriminal ecosystem, lowering the technical barrier to entry for threat actors while increasing the scale and automation of credential theft operations. Raven Stealer is one such tool: a Delphi/C++-based stealer that offers robust credential extraction, stealth execution, and real-time exfiltration via Telegram.

Built with a focus on usability and evasion, Raven enables even novice attackers to deploy credential theft campaigns with minimal setup.

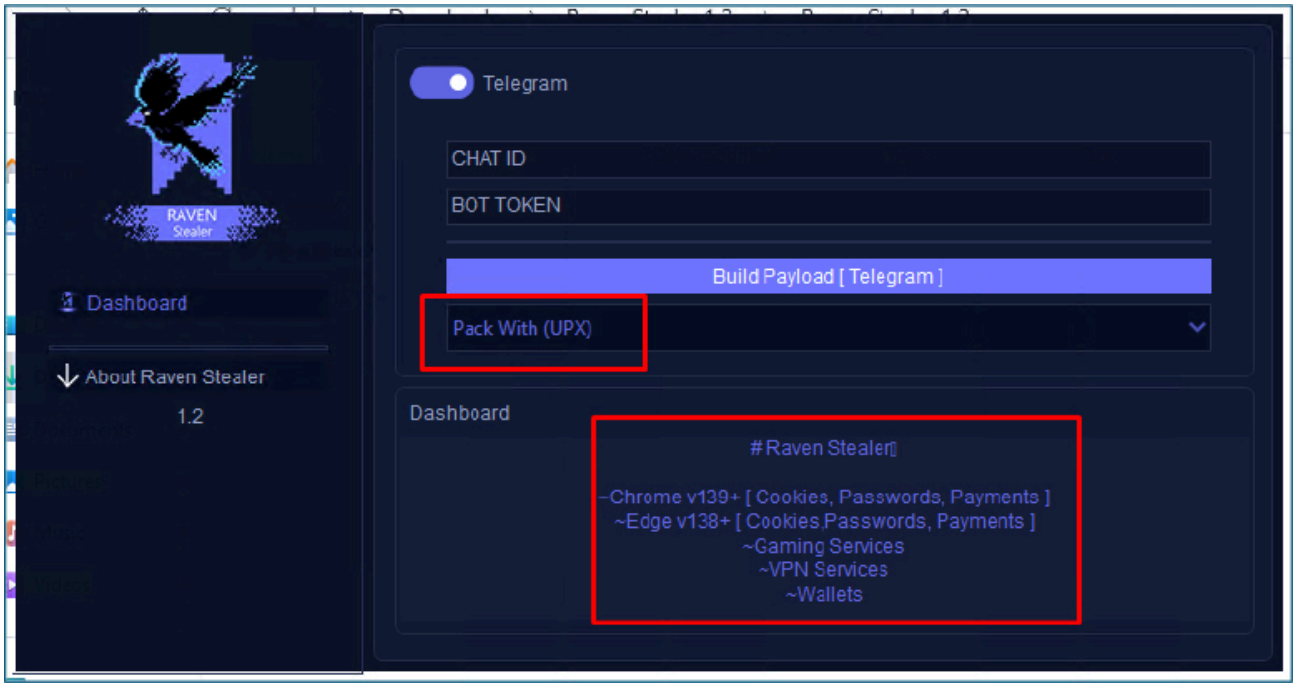
Raven Stealer is actively maintained and distributed through GitHub and a dedicated Telegram channel, ZeroTrace, which functions as the main communication and distribution hub for the tool. Notably, the same threat actor or group associated with Raven is also linked to the release of Octalyn Stealer, a separate but similarly featured stealer, suggesting an evolving portfolio of malware under a single development entity. This signals a broader intent to dominate the low-tier malware-as-a-service (MaaS) space through continuous iteration and brand diversification.

This report provides a technical analysis of Raven Stealer's capabilities, explores its external distribution infrastructure, maps observed behaviors to the MITRE ATT&CK framework, and outlines strategic recommendations for detection and defense.

STATIC ANALYSIS

RavenStealer.exe v8Axs07p.3mf.exe	6237a776e38b6a60229ac12fc6b21fb3 f74ec376aa22ce0b0d55023d8877dc72
Target Technology	Windows OS
Language	RavenStealer.exe **– Delphi** v8Axs07p.3mf.exe **– C++**

The builder, developed using Delphi, features a graphical user interface (GUI) that enables the user to generate a customized stub payload either in its original form or packed with UPX. The generated payload is assigned a randomly generated twelve-character name. By providing a Telegram bot token and chat ID, the user can configure the payload to communicate via Telegram. The embedded stub payload, written in C++, is integrated within the builder.



Analysis of v8Axs07p.3mf.exe

The entropy value of v8Axs07p.3mf.exe exceeds 7, indicating that the file is likely packed using the UPX Cryptor. Such packing techniques are commonly employed to obstruct reverse engineering and complicate static analysis.

property	value
file	
file > sha256	961EC48AA27EA1460BC3C82F2B8462A6F8A60A27AF9D3FAC5543367A55B7C648
file > first 32 bytes (hex)	4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 0
file > first 32 bytes (text)	MZ.....@.....
file > info	size: 1708032 bytes, entropy: 7.764
file > type	executable, 64-bit, console
file > version	n/a

property	value	detail
general		
subsystem	0x0003	console
magic	0x020B	PE+
file-checksum	0x00000000	0x001A6178 (expected)
entry-point > location	0x0020BF50	section[UPX1]
base-of-code > location	0x0006C000	section[UPX1]
size-of-code	0x001A1000	1708032 bytes

The file v8Axs07p.3mf.exe, initially exhibiting high entropy, was successfully unpacked using the UPX utility. This confirmed the use of UPX packing and enabled further static and dynamic analysis.

```
C:\Users\IEUser\Desktop>upx -d v8Axs07p.3mf.exe
                       Ultimate Packer for eXecutables
                          Copyright (C) 1996 - 2020
UPX 3.96w       Markus Oberhumer, Laszlo Molnar & John Reiser   Jan 23rd 2020

        File size         Ratio      Format      Name
   --------------------   ------   -----------   -----------
   1954304 <-   1708032   87.40%    win64/pe     v8Axs07p.3mf.exe

Unpacked 1 file.
```

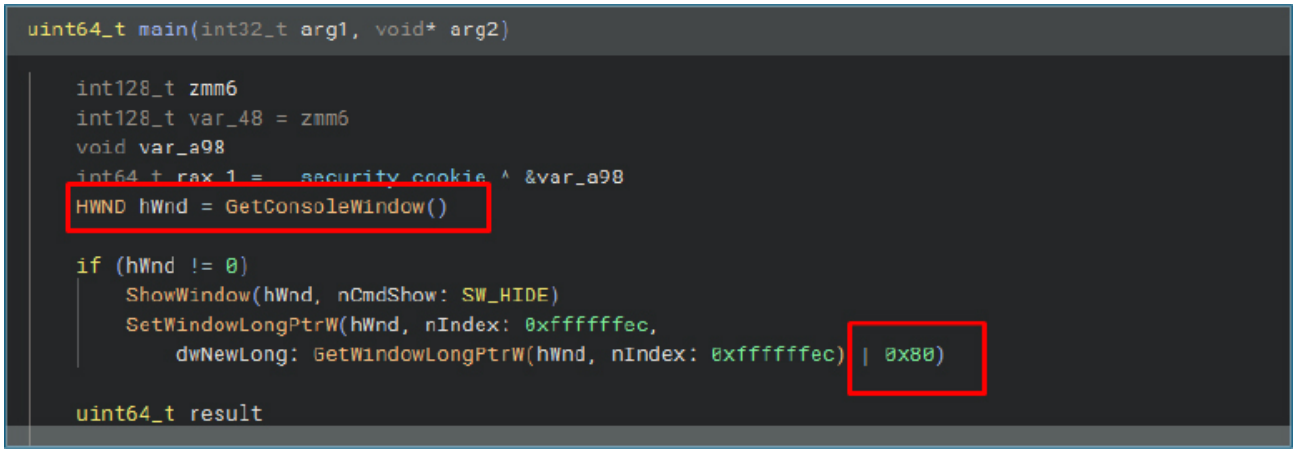
The malware stores the Telegram chat_id and bot token in plain text within embedded resources, specifically under resource IDs 102 and 103. It also includes a PAYLOAD_DLL as an embedded resource, which has a high entropy value of 8, indicating that the DLL is obfuscated. This payload is likely intended for injection into a legitimate process.

name	instance (4)	signature (1)	size (1419661 bytes)	entropy	res...	resource > first 32 bytes (text)
manifest	1	manifest	0x0000017D (381 bytes)	4.912	3C ...	<?xml version='1.0' encoding='UT
rcdata	102	none	0x00000007 (7 bytes)	2.807	43 ...	CHAT ID
rcdata	103	unknown	0x00000009 (9 bytes)	2.725	42 ...	BOT TOKEN
rcdata	PAYLOAD_DLL	unknown	0x0015A800 (1419264 ...	8.000	D7 ...	...C..s....3ZxY....ep....#v.....;....h....

Several imported functions—such as Process32NextW, GetCurrentProcessId, and CreateProcessW—along with the unusually large size of certain resource sections, indicate that the malware is designed to perform DLL injection into legitimate processes.

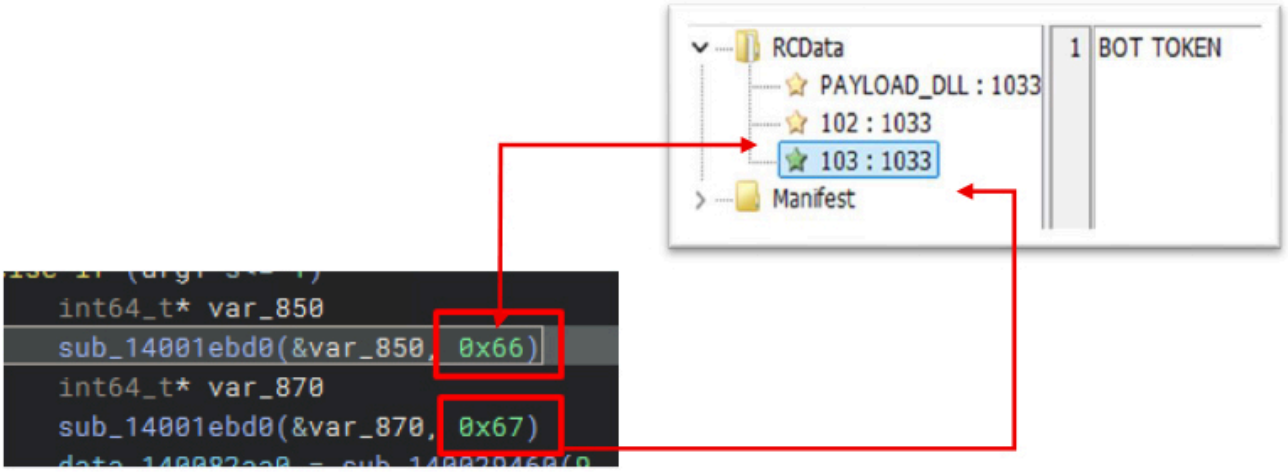
Process32NextW x	SizeofResource
Process32FirstW x	WaitForSingleObject
GetTickCount -	MultiByteToWideChar
IsWow64Process -	Sleep
ConnectNamedPipe -	LockResource
GetCurrentProcessId x	CloseHandle
QueryPerformanceCounter -	LoadResource
IsDebuggerPresent -	FindResourceW
IsProcessorFeaturePresent -	GetProcAddress
GetCurrentProcess x	CreateProcessW
SetUnhandledExceptionFilter -	GetModuleHandleW
UnhandledExceptionFilter -	WideCharToMultiByte
RtlVirtualUnwind -	GetConsoleWindow
RtlLookupFunctionEntry -	ReadFile
RtlCaptureContext -	GetConsoleScreenBufferInfo
GetFileInformationByHandleEx x	SetConsoleTextAttribute
AreFileApisANSI -	GetStdHandle
GetFullPathNameW -	WriteFile
GetFileAttributesExW -	
FindNextFileW x	
FindFirstFileExW x	
FindFirstFileW x	

At its main entry point, the malware hides its console window by calling the Windows API ShowWindow with the SW_HIDE flag. To further enhance stealth, it sets the extended window style 0x80 (WS_EX_TOOLWINDOW), which removes the window from the taskbar and Alt+Tab view, effectively preventing any visible popup and making the process less noticeable to the user.



Extracting Bot Token and ChatID from Embedded Resources:

The malware leverages Windows API functions FindResourceW and LoadResource, using specific resource IDs to extract the embedded Telegram bot token and chat_id. This data is then loaded into memory and used for command-and-control (C2) communication during execution.




```
int64_t* sub_14001ebd0(int64_t* arg1, int16_t arg2)

    HMODULE hModule = GetModuleHandleW(lpModuleName: nullptr)
    HRSRC hResInfo = FindResourceW(hModule, lpName: zx.q(arg2), lpType: 0xa)
    void* const rdx_1
    uint64_t r8

    if (hResInfo != 0)
        HGLOBAL hResData = LoadResource(hModule, hResInfo)

        if (hResData == 0)
            r8 = 0
            rdx_1 = &data_140078e89
        else
            void* rax = LockResource(hResData)

            if (rax == 0)
                r8 = 0
                rdx_1 = &data_140078e89
            else
                r8 = zx.q(SizeofResource(hModule, hResInfo))
                rdx_1 = rax
    else
        r8 = 0
        rdx_1 = &data_140078e89

    __builtin_memset(dest: arg1, ch: 0, count: 0x20)
    sub_140026090(arg1, rdx_1, r8)
    return arg1
```

Decrypt Chromium-based browsers' passwords:

This malware bypasses Chromium's App-Bound Encryption (ABE) using in-memory techniques. It starts a legitimate browser process in a suspended state, then uses Direct Syscall-based Reflective Process Hollowing to inject its payload. By taking over the browser's security context, it can decrypt and steal sensitive data like passwords, cookies, payment info, without touching disk or triggering user-land hooks.

Launches the browser in a suspended state

The malware relaunches the browser with arguments like –headless, –disable-gpu, and –no-sandbox for stealth, stability, and easier exploitation.

–headless: Runs the browser without a visible UI. This avoids user suspicion and saves system resources.

–disable-gpu / –disable-software-rasterizer: Prevents GPU-related issues in headless mode, making the browser run more reliably in a virtualized or background environment.

–no-sandbox: Disables Chromium's built-in sandbox, lowering security barriers and making injection and memory manipulation easier.

```
if (rax_16 u>= 0xa)
    rbx_3 = rsi_2 + (rax_16 << 1)
    rax_17 = j_sub_140071890(rsi_2, rbx_3, u"msedge.exe", 0xa)

if (rax_16 u< 0xa || rax_17 == rbx_3 || (rax_17 - rsi_2) s>> 1 == -1)
    int64_t rax_20 = rdi[2]

    if (rdi[3] u> 7)
        rdi = *rdi

    if (rax_20 u>= 9)
        void* rbx_4 = rdi + (rax_20 << 1)
        uint16_t (* rax_21)[0x10] = j_sub_140071890(rdi, rbx_4, brave.exe", 9)

        if (rax_21 != rbx_4 && (rax_21 - rdi) s>> 1 != -1)
            sub_140023fb0(&var_e0,
                --headless --disable-gpu --disable-software-rasterizer
                  --no-sandbox", 0x44)
    else
        sub_140023fb0(&var_e0,
            --headless --disable-gpu --disable-software-rasterizer --no-sandbox",
            0x44)
else
    sub_140023fb0(&var_e0,
        --headless --disable-gpu --disable-software-rasterizer --no-sandbox", 0x44)
```

In-Memory DLL Injection into Chromium

The malware carries its main payload DLL as a ChaCha20-encrypted resource embedded within itself. This payload is decrypted in memory at runtime, never touching disk, and is injected into a web browser process. Instead of compromising an existing process, the malware creates a new instance of the browser in a suspended state using CreateProcessW. This untouched, suspended process becomes the target for reflective process hollowing, allowing the injected code to run under the browser's identity while avoiding common detection mechanisms.

```
    dwProcessId = dwProcessId_1
label_14006d6c1:

    if (dwProcessId_1 == 0)
        goto label_14006e484

    HANDLE hObject_3 = OpenProcess(dwDesiredAccess: 0x43a,
        bInheritHandle: 0, dwProcessId: dwProcessId_1)

    if (hObject_3 == -1)
        hObject_3 = nullptr

    HANDLE hObject_2 = hObject_3

    if (hObject_3 != 0)
        if (sub_140068970(hObject_3) == 0)
            r14_1 = 1
        else
            __builtin_memset(dest: &var_350, ch: 0, count: 0x20)
            sub_140026090(&var_350,
                "Loading payload DLL from embedded resource.",
                0x2b)
            int128_t var_340_3

            if (data_140082228 != 0)
                HANDLE hConsoleOutput_1 =
                    GetStdHandle(nStdHandle: STD_OUTPUT_HANDLE)
                SetConsoleTextAttribute(
                    hConsoleOutput: hConsoleOutput_1,
                    wAttributes: 0x6)
                int64_t* rax_37 = sub_140023320(std::cout, "[#] ")
                int128_t* rdx_31 = &var_350

                if (var_340_3:8.q u> 0xf)
                    rdx_31 = var_350.q

                std::ostream::operator<<(
                    this: sub_140025470(rax_37, rdx_31,
                        var_340_3.q),
                    sub_140023500)
```

Reflective Injection via Syscalls:

By leveraging direct syscall injection, the malware bypasses antivirus detection and targets processes like chrome.exe and brave.exe to extract sensitive data. It allocates memory in the remote process using NtAllocateVirtualMemory, writes the decrypted payload into that space with NtWriteVirtualMemory, and sets the memory region to executable using NtProtectVirtualMemory. To enable communication, it creates a named pipe and writes its identifier directly into the target process's memory.

```
int64_t rsi = 0
data_140082229 = arg1
__builtin_memset(dest: &data_1400821e0, ch: 0, count: 0x28)
HMODULE rax = GetModuleHandleW(lpModuleName: u"ntdll.dll")
int128_t var_b8

if (rax == 0)
    __builtin_memset(dest: &var_b8, ch: 0, count: 0x20)
    sub_140026090(&var_b8, "GetModuleHandleW for ntdll.dll failed.", 0x26)

    if (data_140082229 != rax.b)
        int64_t* rax_1 = sub_140023320(std::cout, "[#] [Syscalls] ")
        int128_t* rdx_1 = &var_b8
        int128_t var_a8_1
```

Enumeration and Extraction of Stored Browser Credentials

The malware performs system-wide enumeration to locate stored credentials on the infected machine. It specifically searches for saved browser data, including passwords and authentication cookies, from applications such as Chrome, Brave, and other Chromium-based browsers.

```
void* rbx_5 = rdi_7 + rax_168
uint8_t (* rax_169)[0x20] =
    j_sub_1400716a0(rdi_7, rbx_5, "assword", 7)
int64_t var_928_1

if (rax_169 == rbx_5 || rax_169 - rdi_7 == -1)
    rdx_19 = var_8b8
    rax_168 = var_8c0
    rcx_54 = var_8d0
label_140021b8e:
    int64_t* rdi_9 = &var_8d0

    if (rdx_19 u> 0xf)
        rdi_9 = rcx_54

    if (rax_168 u>= 5)
        uint8_t (* rbx_7)[0x20] = rdi_9 + rax_168
        uint8_t (* rax_173)[0x20] =
            j_sub_1400716a0(rdi_9, rbx_7, "ookie", 5)

        if (rax_173 != rbx_7 && rax_173 - rdi_9 != -1)
            void* rdi_10 = i + 0x20
```

After extracting the data, the malware collects information from various sources such as cryptocurrency wallets, saved passwords, browser cookies, gaming platforms, VPN clients, and instant messaging services. It then stores the stolen data in the user's AppData directory in a well-organized folder structure for later exfiltration.

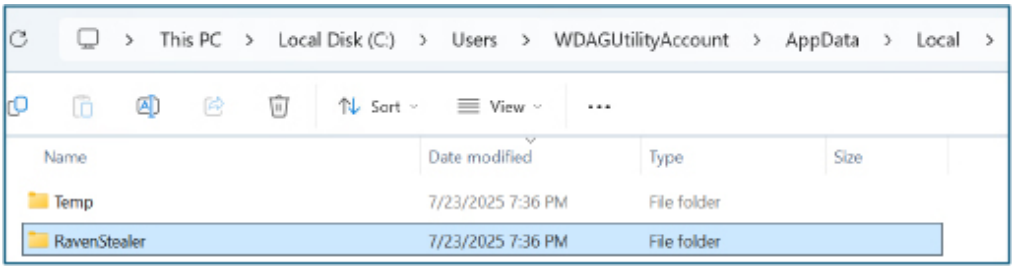
```
14001b356 memcpy(rsi + 0x18, rdi, r15.d)
14001b45e memcpy(rbx_3, "Wallets Found (", rsi_1.d)
14001b477 memcpy(&rbx_3[rsi_1], &data_140078f60[0xf][rsi_1], 0xf - rsi_1.d)
14001b61a memcpy(r14_1 + 2, r12_1, r13.d)
14001b770 memcpy(rbx_6, "Passwords Found: ", rsi_4.d)
14001b786 memcpy(rbx_6 + rsi_4, &data_140078f78[0x11][rsi_4], 0x11 - rsi_4.d)
14001b905 memcpy(rbx_7, "Cookies Found: ", rsi_6.d)
14001b91e memcpy(rbx_7 + rsi_6, &data_140078f90[0xf][rsi_6], 0xf - rsi_6.d)
14001baae memcpy(rbx_8, "Gaming Services Found (", r14_2.d)
14001bac4 memcpy(&rbx_8[r14_2], &data_140078fa0[0x17][r14_2], 0x17 - r14_2.d)
14001bc6a memcpy(r14_4 + 2, r12_3, r13_1.d)
14001bdcd memcpy(rbx_11, "IM Services Found (", r14_5.d)
14001bde3 memcpy(&rbx_11[r14_5], &data_140078fb8[0x13][r14_5], 0x13 - r14_5.d)
14001bf89 memcpy(r14_7 + 2, r12_5, r13_2.d)
14001c0ea memcpy(rbx_14, "VPN Services Found (", r14_8.d)
14001c100 memcpy(&rbx_14[r14_8], &data_140078fd0[0x14][r14_8], 0x14 - r14_8.d)
14001c2a8 memcpy(r14_10 + 2, i_1, r13_3.d)
```

DYNAMIC ANALYSIS:

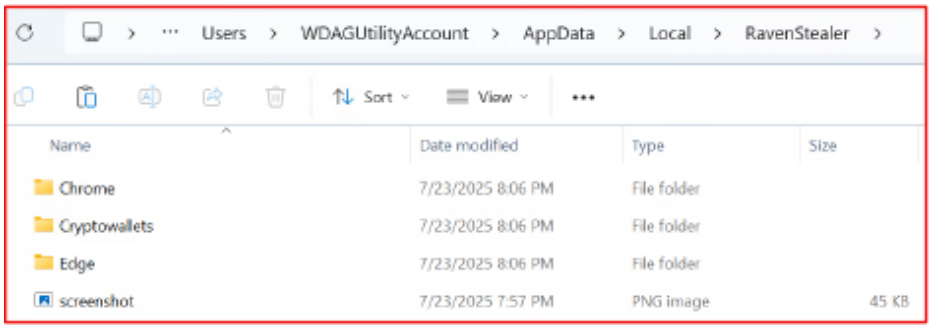
Storage of Exfiltrated Data

The malware executes a PowerShell command to create an archive of the exfiltrated data, packaging files from the AppData directory into a compressed file stored in the Temp folder.

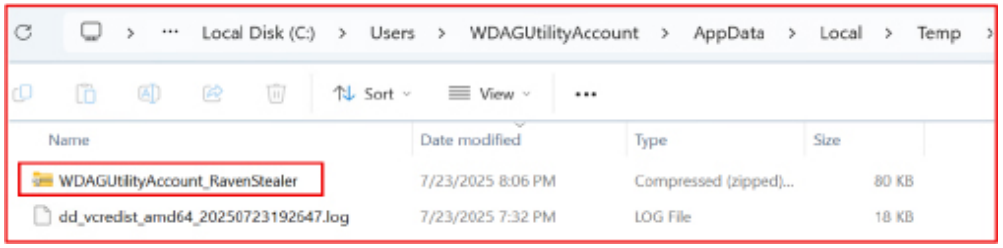
All stolen credentials and system information are stored in an organized directory structure within the %Local%\\ RavenStealer folder.



Subfolders include categories like Chrome, Crypto wallets, Edge, and others. Individual files, such as screenshot.png, further structure the harvested data, making it easy for the attacker to parse and review once exfiltrated through the Telegram bot.

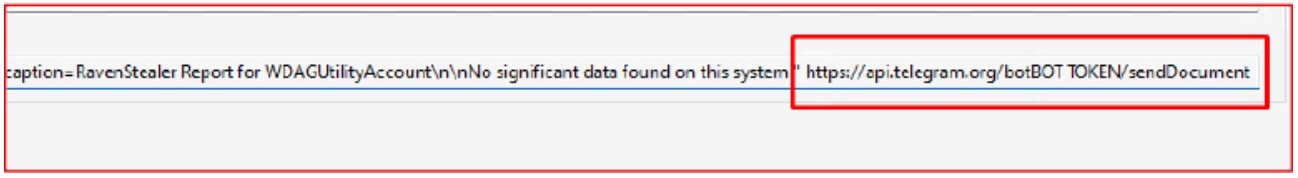


Upon completion of data collection, the contents of the folder are compressed into a ZIP archive and stored in the system's temporary directory. The archive filename includes a suffix containing the current user's username.

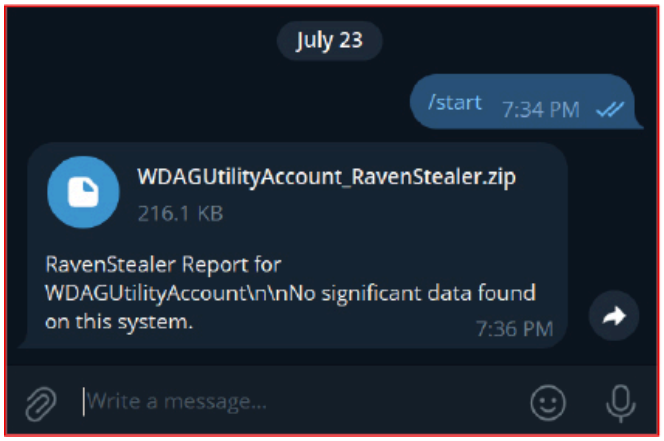


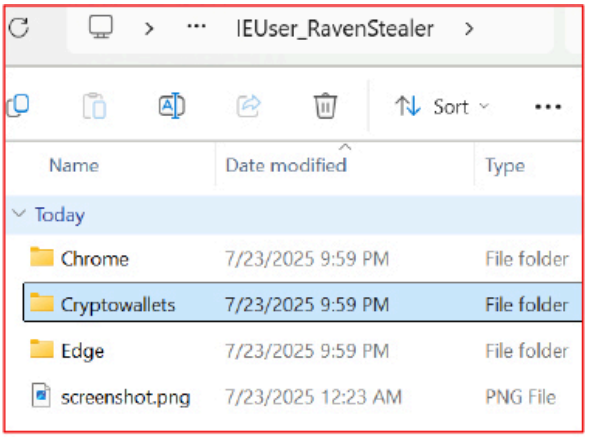
Data Exfiltration

The Network traffic analysis reveals that the malware uses curl.exe with specific arguments to upload the exfiltrated ZIP file via the Telegram API. It leverages the /sendDocument method to directly send the stolen credentials to the attacker's Telegram bot, using the embedded CHAT_ID and BOT TOKEN.



Upon successfully establishing a connection with the attacker-controlled Telegram bot, Raven Stealer transmits the name of a uniquely identifiable ZIP archive to indicate successful infection and data exfiltration. The archive name typically incorporates the victim's system username, appended with a fixed suffix such as RavenStealer.zip. This consistent naming convention allows the threat actor to systematically organize, monitor, and correlate exfiltrated data with individual victims across numerous compromised systems.





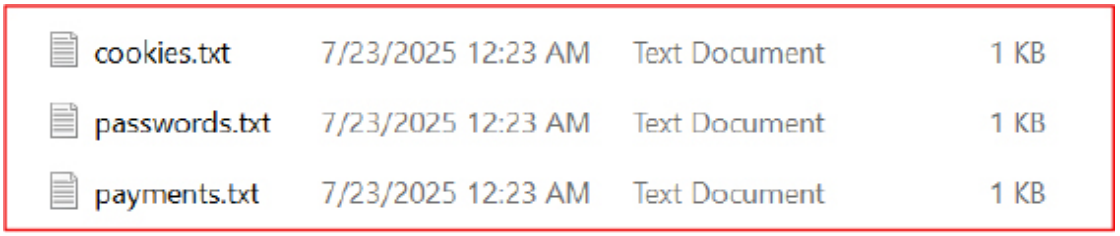
Upon downloading and extracting the exfiltrated ZIP archive, the contents are presented in a structured and systematic format, enabling efficient access and analysis by the threat actor. The archive typically includes several directories and files, each representing specific categories of compromised data.

Folders:

- Chrome > Default: Contains browser-specific data files such as cookies.txt, password.txt, and payment.txt, extracted from the default user profile of Google Chrome.
- Edge > Default: Includes cookies.txt, password.txt, and payment.txt, retrieved from the default user profile of Microsoft Edge.
- Crypto Wallets: Stores information extracted from browser-based or locally installed cryptocurrency wallets, including wallet addresses, private keys, and configuration files.

Files:

- cookies.txt: A compiled collection of cookies gathered from multiple browsers, typically used for session hijacking purposes.
- passwords.txt: Includes decrypted or plaintext credentials saved within web browsers.
- payment.txt: Extracted payment-related data, such as saved credit or debit card details and associated billing information stored in the browser.



- screenshot.png: A desktop screenshot captured at the time of execution, providing the attacker with visual context of the compromised environment.

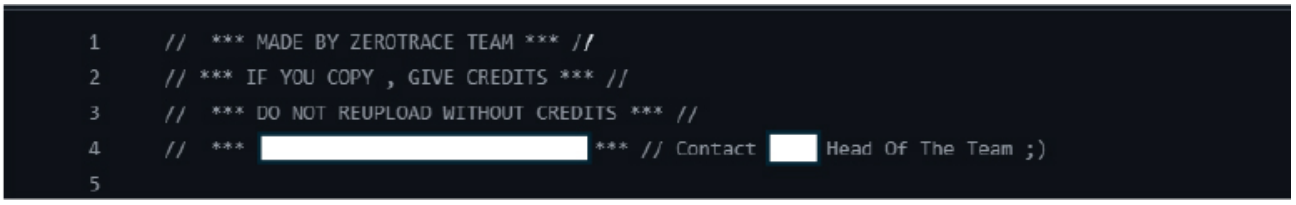
This structured layout allows the threat actor to efficiently locate, interpret, and exploit a wide range of sensitive information. It reflects Raven Stealer's comprehensive data collection capabilities, spanning personal details, login credentials, financial information, and visual evidence of system activity.

EXTERNAL THREAT LANDSCAPE MANAGEMENT

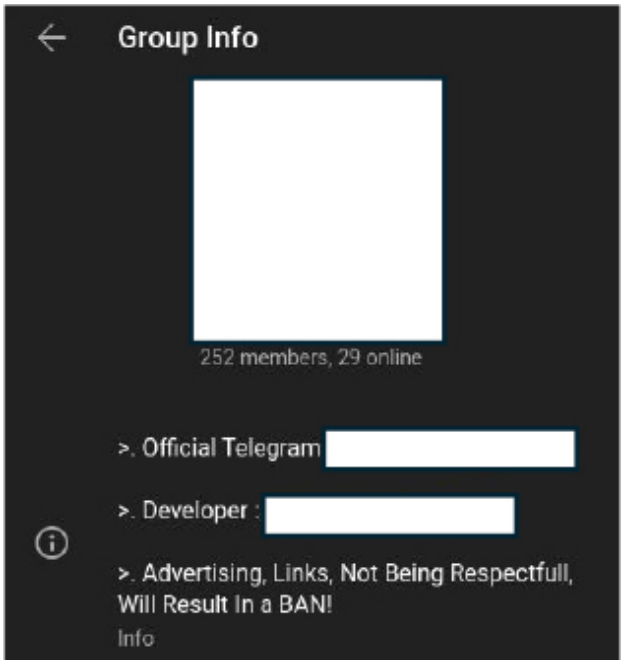
Raven Stealer is attributed to a threat actor or developer group operating under the name ZeroTrace Team, which maintains an active presence across both GitHub and Telegram. The stealer was first publicly uploaded to GitHub on July 15, 2025, and includes numerous indicators pointing to a consistent development identity.

Notably, internal file paths recovered from the repository's build artifacts, such as the '.identcache' file, include references to a development environment that reveal a local Windows machine setup, indicating a specific user account was involved.

Within the source code itself, specifically in the RavenStealer.cpp file, a hardcoded author declaration attributes the work to the ZeroTrace Team. The content includes a name and a Hotmail email address, suggesting deliberate branding and author tagging consistent with attempts to gain visibility within illicit developer communities.



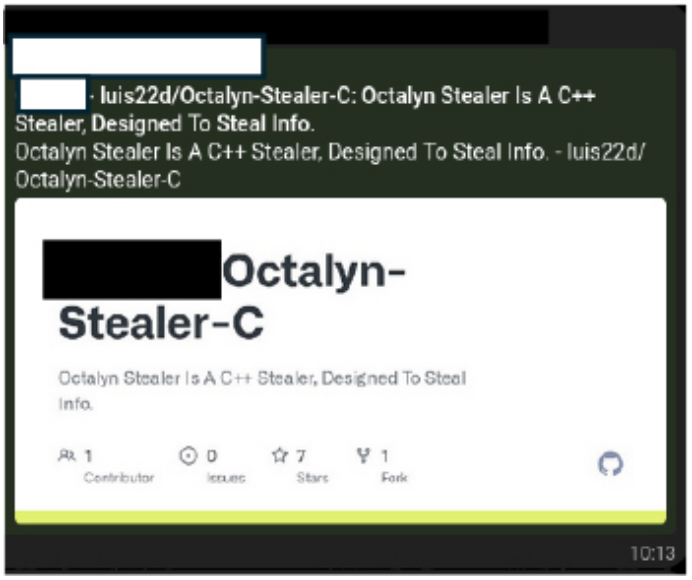
Their Telegram channel frequently shares builder updates, usage tutorials, and stealer variants, and serves as a contact point for threat actors and users. Telegram's anonymity, wide reach, and ease of bot integration make it a popular choice for real-time data exfiltration, especially for credential logs and screenshots.



The ZeroTrace Telegram channel was created on April 30, 2025, initially promoting a custom crypter tool branded as "ZeroTrace", and has since expanded to publish and promote multiple stealer variants, including Raven Stealer and Octalyn Stealer, often hosted and shared via open-source repositories.

The stealer's builder tool allows threat actors to directly embed Telegram Bot Tokens and Chat IDs into the payload, enabling seamless delivery of stolen data to private Telegram chats. Through this mechanism, Telegram acts as both the communication layer and the exfiltration endpoint, replacing traditional C2 servers.

Additionally, the same channel is used to promote another malware strain named Octalyn Stealer, which shares structural and operational similarities with Raven Stealer. The use of a single Telegram hub for both tools suggests that ZeroTrace Team actively maintains multiple infostealer variants, likely to diversify branding, bypass signature-based detection, and expand reach across different threat actor groups.



Together, these infrastructure elements, GitHub repositories, Telegram-based distribution and exfiltration, and direct developer attribution provide a clear external footprint of the Raven Stealer ecosystem.

CONCLUSION

Raven Stealer incorporates advanced stealth capabilities by executing silently without displaying a console or command prompt, thereby significantly reducing the risk of user detection. It facilitates the extraction of sensitive data, including browser credentials, cookies, payment information, and autofill details from Chromium-based browsers, while also extending its targeting to include gaming services, VPN clients, and cryptocurrency wallets. Its modular design, embedded configuration options, and UPX packing enhance its ability to evade static detection and simplify deployment, rendering it accessible even to actors with limited technical expertise. Distributed via GitHub and promoted through Telegram, Raven Stealer employs encrypted exfiltration and seamless integration with Telegram bots for real-time data transmission and campaign management. These characteristics reflect its ongoing development and reinforce its potential as a substantial threat when utilized beyond legitimate research or controlled environments.

MITRE ATTACK FRAMEWORK

Tactic	ID	Technique Name
Execution	T1129	Shared Modules

Persistence	T1542	Pre-OS Boot
	T1542.003	Bootkit
Defense Evasion	T1027	Obfuscated Files or Information
	T1027.002	Obfuscated Files or Information: Software Packing
	T1027.005	Indicator Removal from Tools
	T1497	Virtualization/Sandbox Evasion
	T1542	Pre-OS Boot
	T1542.003	Bootkit
	T1564	Hide Artifacts
	T1564.003	Hide Artifacts: Hidden Window
Discovery	T1057	Process Discovery
	T1082	System Information Discovery
	T1083	File and Directory Discovery
	T1497	Virtualization/Sandbox Evasion
	T1518	Software Discoverys
	T1518.001	Software Discovery: Security Software Discovery
	T1614	System Location Discovery
Command and control	T1071	Application Layer Protocol

INDICATORS OF COMPROMISE

S. N	indicators	type	context
1.	2e0b41913cac0828faeba29aebbf9e1b36f24e975cc7d8fa7f49212e867a3b38	EXE	RavenStealer.exe
2	28d6fbbdb99e6aa51769bde016c61228ca1a3d8c8340299e6c78a1e004209e55	EXE	v8Axs07p.3mf.exe
3.	252fb240726d9590e55402cebbb19417b9085f08fc24c3846fc4d088e79c9da9	DLL	PAYLOAD_DLL.dll

YARA RULES

```
rule RavenStealer_Detection
{
```

```
meta:
description = “Detects Raven Stealer by identifying strings and embedded hash values”
author = “Cyfirma Research”
date = “2025-07-23”
malware_family = “Raven Stealer”

strings:
// IOC strings
$s1 = “api.telegram.org” nocase
$s2 = “RavenStealer” nocase
$s3 = “passwords.txt” nocase
$s4 = “payment.txt” nocase
$s5 = “autofill.txt” nocase
$s6 = “%Local%\\RavenStealer\\Chrome” nocase
$s7 = “%Local%\\RavenStealer\\Edge” nocase
$s8 = “Crypto Wallets” nocase

// Known SHA-256 hashes as string literals (if embedded in binary or visible)
$s9 = “2e0b41913cac0828faeba29aebbf9e1b36f24e975cc7d8fa7f49212e867a3b38”
$s10 = “28d6fbbdb99e6aa51769bde016c61228ca1a3d8c8340299e6c78a1e004209e55”
$s11 =
“252fb240726d9590e55402cebbb19417b9085f08fc24c3846fc4d088e79c9da9” condition:
// Match if at least 3 of the above strings appear
3 of ($s*)
}
```

Recommendations and Mitigation Strategies

1. Endpoint Detection & Response (EDR) and Antivirus

- Deploy advanced EDR solutions capable of behavioral analysis to detect anomalous activities such as credential harvesting, ZIP file creation in temp directories, and clipboard access.
- Ensure antivirus/antimalware software includes real-time protection and heuristic scanning to detect packed binaries (e.g., UPX-packed executables).
- Use YARA rules to proactively hunt for known indicators associated with Raven Stealer, such as strings related to Telegram API usage and stolen data paths.

2. Network and TLS Traffic Monitoring

- Monitor for outbound connections to api.telegram.org, especially from unusual processes or user directories, as this is a key C2 and exfiltration vector.
- Deploy TLS/SSL inspection at the gateway level where legally and technically feasible, to identify suspicious encrypted traffic patterns.
- Implement DNS filtering or IP-based blocking of Telegram C2 endpoints if Telegram is not required for business use.

3. Application Whitelisting and Execution Control

- Restrict execution of unauthorized binaries, especially from %Temp%, %AppData%, and %LocalAppData% directories.
- Prevent execution of binaries with high entropy or known packing signatures (e.g., UPX) in user-writable directories.

4. Email and Download Filtering

- Configure email gateways to block executable attachments, archive files containing executables, or obscure formats (e.g., .3mf.exe) used to evade detection.
- Employ browser sandboxing and download monitoring to prevent unauthorized payload delivery.

5. Credential Management and Browser Hardening

- Advise users not to store credentials or payment information in browsers. Use enterprise password managers with MFA support.
- Disable browser autofill and password saving features.
- Regularly clear stored browser data and cookies across endpoints.

6. Telegram and GitHub Monitoring

- Monitor and restrict the use of unauthorized Telegram applications on corporate systems. Implement DLP (Data Loss Prevention) to identify suspicious data progress.
- Regularly scan GitHub and dark web platforms for new instances of Raven Stealer or associated ZeroTrace artifacts using threat intelligence feeds and code similarity detection tools.

7. User Awareness and Training

- Conduct ongoing phishing simulation campaigns and train employees on malware delivery tactics like disguised executables (e.g., invoice.3mf.exe) and fake educational tools.
- Warn users about the risks of downloading software from GitHub repositories without validation.

8. Incident Response Preparedness

- Ensure incident response plans account for credential-stealing malware and C2 over encrypted messaging platforms.
- In the event of detection, revoke all browser-stored credentials, invalidate sessions, and perform password resets.
- Capture forensic images and logs for deeper analysis and evidence preservation.
