CAMPAIGN KCPHOENIX:

INSIDE SILVER FOX'S LATEST VALLEYRAT CHAIN

Author: Poonam Dongare, Wojciech Bohatyrewicz
No of pages: 44
Read time: 35 minutes

Contents

1 Executive Summary

In July 2026, Atos Threat Research Center published[1] research documenting a Silver Fox APT (also known as TA4922) campaign distributing **ValleyRAT** through SEO-poisoned fake software installers targeting Chinese-speaking users. We continued tracking activity linked to this threat actor and identified a new cluster of samples sharing the same malware family but with a substantially evolved delivery chain.

This report documents that evolution. We are tracking this cluster as Campaign **KCPhoenix**: a designation derived from two artifacts found during analysis - the KCP[2] transport protocol used, and the string BOP_phoenix_release_2.51.0 recovered from the PDB path of a Blizzard signed binary embedded in the chain.

The campaign was active from at least July 2026 through August 2026, delivered through a trojanized AnyDesk installer served from an AnyDesk impersonation site. What sets this cluster apart from our earlier findings is the presence of a **kernel-level defense evasion** capability: a Bring Your Own Vulnerable Driver (**BYOVD**)[3] technique using a re-signed K7RKScan driver, alongside a Donut[4] loader with a Chaskey[5] based cipher, and a fake antivirus product registered with Windows that keeps the **system reporting as protected while real security controls are disabled**.

Persistence is also handled differently in this campaign. The scheduled task does not execute the malware directly; instead, it launches a **signed Blizzard binary** which then starts the malware.

The final payload remains ValleyRAT, a remote access trojan with **KCP/UDP based C2 capabilities**, file collection, command execution, and process control, consistent with the Silver Fox toolset documented in our prior research.

[1] ValleyRAT distribution campaign delivered through a fake AnyDesk website - Atos

[2] kcp/README.en.md at master · skywind3000/kcp · GitHub

[3] What Is a BYOVD Attack? Bring Your Own Vulnerable Driver, Explained

[4] https://github.com/thewover/donut

[5] https://mouha.be/chaskey/

2 Delivery and Execution Chain

The infection proceeds through 5 tightly coupled stages. The diagram below maps the complete chain from the initial dropper through to command-and-control (C2) communication.

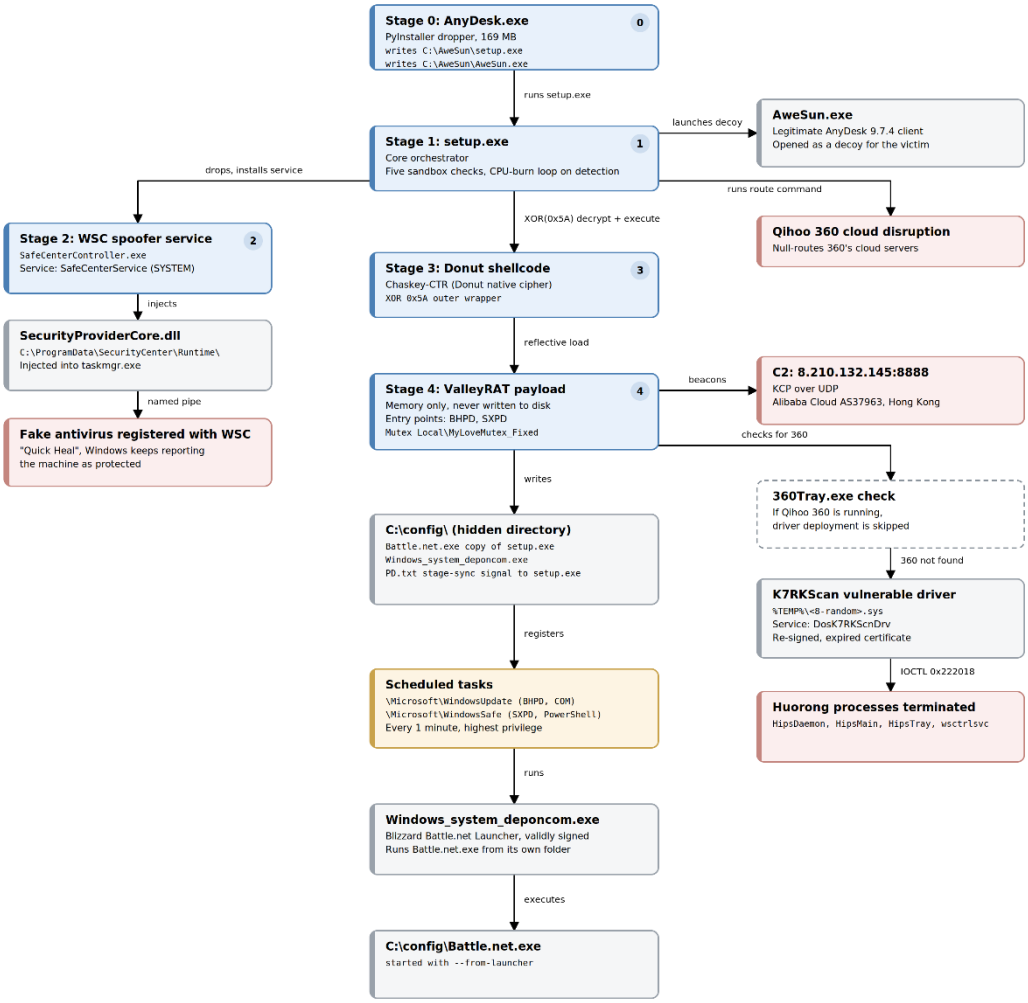

Figure 1: Full Execution Chain of Campaign KCPhoenix

3 Stage 0 – AnyDesk.exe: PyInstaller Dropper

The campaign entry point is a trojanized AnyDesk installer served from anydesk[.]nx[.]cn[6], a site mimicking the official AnyDesk website in Chinese and offering "AnyDesk 9.7.5 for Windows". This distribution method is consistent with the SEO-poisoned fake installer sites documented in our earlier research.

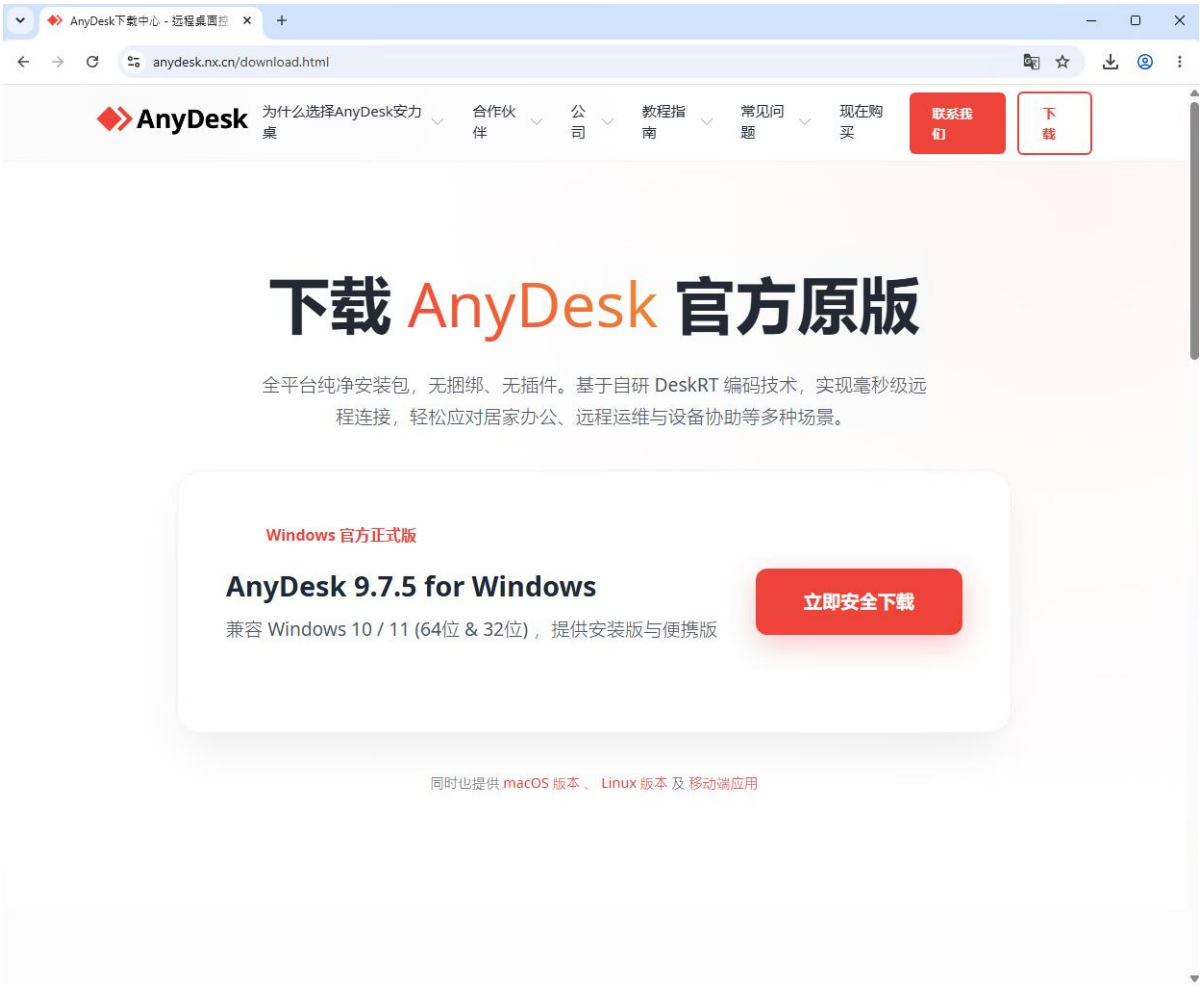

Figure 2: AnyDesk download page

[6] https://www.virustotal.com/gui/domain/anydesk.nx.cn

The file named AnyDesk.exe[7] is a PyInstaller[8] packaged Python 3.8 application weighing 169 MB. The large size is mainly due to a legitimate embedded Python runtime and a 117.8 MB unused data block that is never executed.

Upon execution, the PyInstaller package extracts and runs the Python bytecode module kb_out_AnyDesk.py. The module contains three hex-encoded strings. Two are decoded and written to disk: hex_string_1 produces C:\AweSun\setup.exe, the MFC-based RAT, while hex_string_2 produces C:\AweSun\AweSun.exe, a legitimate AnyDesk v9.7.4 client signed by AnyDesk Software GmbH. The third, hex_string_3, is never decoded or written in any observed code path and appears to be unused bloat. The dropper then launches setup.exe using subprocess.run(), and stays running in the background for as long as the RAT is active.

```
import os, threading, subprocess
hex_string_1 = "4d5a900003000000040000000ffff0000b8000000000000000400000..."  # truncated
hex_string_2 = "4d5a900003000000040000000ffff0000b8000000000000000400000..."  # truncated
hex_string_3 = "4d5a900003000000040000000ffff0000b8000000000000000400000..."  # truncated
target_dir = "C:\\AweSun"
if not os.path.exists(target_dir):
    os.makedirs(target_dir)

# Convert hex strings to binary data
binary_data_1 = bytes.fromhex(hex_string_1)
binary_data_2 = bytes.fromhex(hex_string_2)

# Define file paths
exe_file_path_1 = os.path.join(target_dir, "setup.exe")
exe_file_path_2 = os.path.join(target_dir, "AweSun.exe")

# Write binary data to files
with open(exe_file_path_1, "wb") as exe_file_1:
    exe_file_1.write(binary_data_1)
with open(exe_file_path_2, "wb") as exe_file_2:
    exe_file_2.write(binary_data_2)

class RunExeThread(threading.Thread):
    def __init__(self, exe_path):
        threading.Thread.__init__(self)
        self.exe_path = exe_path
    def run(self):
        subprocess.run([self.exe_path], capture_output=True, text=True)

thread1 = RunExeThread(exe_file_path_1)
thread1.start()
thread1.join()
```

Figure 3: PyInstaller dropper writing setup.exe and AweSun.exe to disk

[7] VirusTotal - File - 8e8c9e00a07b3051699b3426e06db01f57814c1377748de00ca9e719bdd4392b

[8] PyInstaller Manual — PyInstaller 6.22.2 documentation

4 Stage 1- setup.exe: Core Orchestrator

setup.exe[9] is the main component of the execution chain. It is a 22.4 MB x64 MFC C++[10] application that uses Calculator.exe as its internal name. It manages the later stages of the attack, including sandbox detection, decoy execution, service installation, antivirus disruption, and shellcode execution. A copy of this binary is later written to C:\config\Battle.net.exe by the ValleyRAT payload for persistence.

Property	Value
CompanyName	LVM4pfmMLhAvE7b6GzDyO15TPC9EuevkJo1PdepjmgUwVXdu2v
FileDescription	844ZmBS54uYHcB2nBl1b0YnHam9uPg6avhjwEKpEngNXRYDEKN
FileVersion	3.10
InternalName	Calculator.exe
LegalCopyright	iHQfUGerufWMS7caU6dOTIQkmbmgFMs9Y015JTXINndKj3J8Ah
OriginalFilename	Calculator.exe
ProductName	hOVCleSwVNXZQi84xILmQryuw4gJe1BAHVjVszzpr4ylWQNXi2

Figure 4: Metadata fields for setup.exe

4.1 Anti-Analysis: Sandbox Detection

The orchestrator performs five sandbox checks on every startup. If any check detects a sandbox, it enters a loop that calls GetProcAddress[11] ("GetSystemInfo") 5,000,000 times. This keeps the CPU busy and can exhaust the sandbox's analysis time. It then exits with ExitProcess(1)[12].

[9] VirusTotal - File - bad1a1eeb66cfe7d61f559da36112d2f4de8f6df8bdbb66a308764c9cede77ba

[10] https://learn.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications?view=msvc-170

[11] GetProcAddress function (libloaderapi.h) - Win32 apps | Microsoft Learn

[12] https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-exitprocess

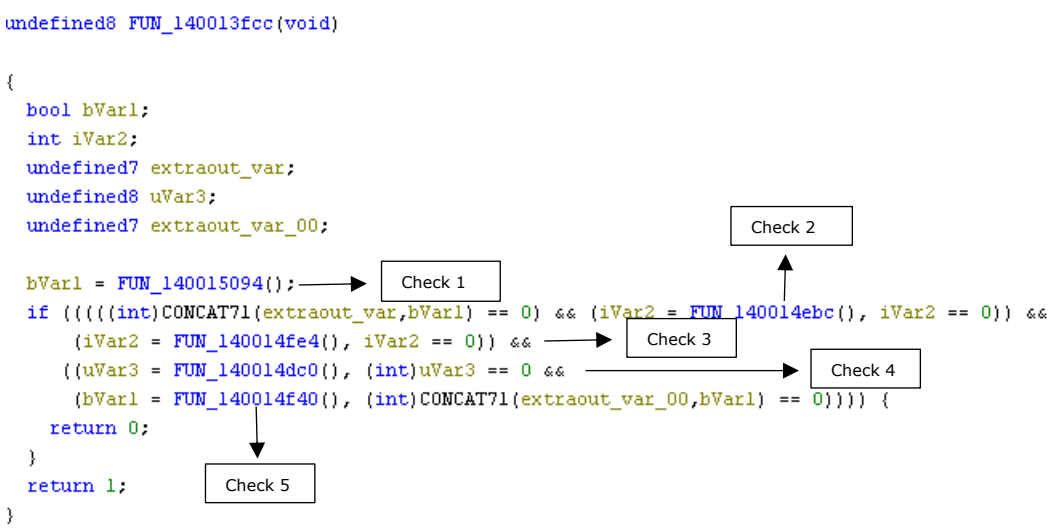

Figure 5: Sandbox Detection Checks

Check 1 (WDAGUtilityAccount): Calls GetUserNameW()[13] and compares the result against the string "WDAGUtilityAccount" using wcscmp. This account name is the default user for Windows Defender Application Guard (WDAG) sandbox sessions.

Check 2 (CExecSvc.exe Process): Creates a process snapshot via CreateToolhelp32Snapshot[14] and searches for a running process named "CExecSvc.exe". This is the Container Execution Service that runs inside Windows Sandbox in isolated environments.

Check 3 (mshome.net DNS Suffix): Calls GetAdaptersAddresses()[15] to enumerate all network adapters and checks whether any adapter's DnsSuffix string equals "mshome.net". This is the default DNS suffix assigned to virtual network adapters in Windows Sandbox environments.

Check 4 (WDAGUtilityAccount RunOnce Key): The orchestrator checks for Windows Defender Application Guard (WDAG)[16], a wmic useraccount command is added to the Windows RunOnce registry key. It reads this key and looks for that command. If it finds it, the malware assumes it is running inside a WDAG environment and exits.

Check 5 (WcSandboxState): Calls NtCreateFile[17] targeting the path \\??\\C:\\WcSandboxState. It attempts to open C:\WcSandboxState, a path associated with Windows Sandbox. If the path can be opened, execution stops.

[13] https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-getusernamew

[14] https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot

[15] https://learn.microsoft.com/en-us/windows/win32/api/iphlpapi/nf-iphlpapi-getadaptersaddresses

[16] https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/microsoft-defender-application-guard/md-app-guard-overview

[17] https://learn.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntcreatefile

4.2 Defense Evasion: 360 AV Disruption

Qihoo 360[18], one of the most widely deployed security products in the Chinese-speaking market, relies heavily on cloud services, its client maintains continuous connections to 360's servers to resolve file reputations and pull updates. Rather than terminating the antivirus, which would trip tamper protection and alert the user: the malware disconnects the link, leaving a process that appears fully operational but is functionally blinded.
Most malware hardcodes a vendor's server addresses, which are easily neutralized once they rotate and easily detected once known. Here, the threat actor discovers 360's infrastructure at runtime, from the antivirus's own live traffic, it enumerates active connections, isolates those belonging to the core 360 processes (360Tray.exe, 360sd.exe) by their process IDs, and for each remote server installs a permanent Windows route redirecting that address to a null gateway, discarding the traffic before it leaves the machine. The commands run through a hidden shell with output suppressed:

```
route -p add <360-server-IP> mask 255.255.0.0  0.0.0.0 metric 1
```

-p: Makes the route persistent. It will remain active even after reboot.

4.3 Legitimate AnyDesk Decoy Launch via Explorer

To keep the victim unsuspecting, the orchestrator opens a legitimate AnyDesk client, dropped under the filename AweSun.exe[19] as a decoy, the user sees a normal program open while the malicious activity continues quietly in the background.
The launch is triggered by C:\config\pd1.txt. The orchestrator launches the decoy only if the file contains data; after launching, it empties the file, making the trigger single-use.
It does not launch AweSun.exe by itself. Instead, it uses the Windows shell namespace associated with "This PC"[20] {20D04FE0-3AEA-1069-A2D8-08002B30309D} and passes the resulting path to a shell resolution function. Windows Explorer then performs the actual launch. As a result, AweSun.exe appears as a child of explorer.exe, rather than being directly spawned by the malware.

[18] 360官网 -360安全中心 - 360安全软件 - 360智能硬件 - 360智能家居 - 360企业服务

[19] VirusTotal - File - 9f1effd1929bb3af3318e1500af2221b990e3ceb5bdae2d327e3fc8929fc41dc

[20] CLSID List (Windows Class Identifiers) | AutoHotkey v1

```
undefined8 FUN_14000ac90(void)

{
  bool bVar1;
  undefined8 uVar2;
  longlong local_538 [34];
  wchar_t local_428 [264];
  wchar_t local_218 [268];

  uVar2 = 0x40;
  FUN_140007254(local_538,"C:\\config\\pdl.txt",1,0x40,1);
  bVar1 = FUN_14001303c((longlong)local_538 + (longlong)*(int *)(local_538[0] + 4));
  if (!bVar1) {
    FUN_14001575c(local_428,0x104,L"C:\\AweSun\\AweSun.exe",uVar2);
    FUN_14001575c(local_218,0x104,L"::{20D04FE0-3AEA-1069-A2D8-08002B30309D}\\%s",local_428);
    FUN_14000bf28(local_218,L"");
  }
  FUN_140012308(local_538);
  FUN_140007544((longlong *)local_428,"C:\\config\\pdl.txt",2,0x40,1);
  FUN_140009abc((longlong)local_428);
  FUN_140009a34((longlong)local_538);
  return 0;
}
```

Figure 6: Decoy launch

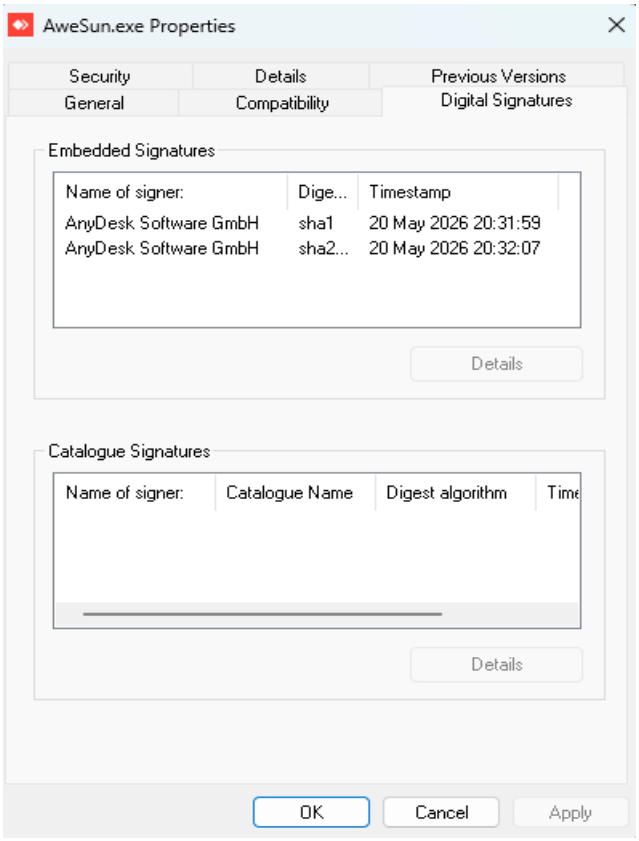

Figure 7: AweSun.exe signed by Valid Certificate

4.4 Shellcode Extraction: XOR Decryption and In-Memory Execution

The shellcode payload is stored in the .data section of setup.exe in encrypted form (DAT_140268000). At runtime, the core orchestrator decrypts 474,187 bytes using a XOR key 0x5A. The decrypted shellcode is written into a memory region allocated with read, write, and execute permissions (RWX) via VirtualAlloc[21] and executed through a direct call. After the shellcode returns, the memory is freed with VirtualFree[22]. The decrypted content is the Stage 3 Donut shellcode loader described in Section 6.

[21] https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualalloc

[22] https://learn.microsoft.com/en-us/windows/win32/api/memoryapi/nf-memoryapi-virtualfree

Figure 8: XOR decryption loop with key 0x5A

5 Stage 2 - SafeCenterController.exe: Windows Security Center Spoofer

SafeCenterController.exe[23] is installed as an auto-start Windows service named SafeCenterService (running as SYSTEM). Its role is to manipulate the Windows Security Center (WSC) view of the host, to register a fake antivirus product so that the Windows Action Center displays a green “protected” status, even after the victim's actual security product has been disabled.

Rather than implementing the WSC registration logic directly inside the service, it drops SecurityProviderCore.dll[24] into C:\ProgramData\SecurityCenter\Runtime\ and injects it into a legitimate Windows process: taskmgr.exe. The service first terminates existing taskmgr.exe instances, then launches a new instance in a suspended state. It places the DLL path into the remote process using VirtualAllocEx[25] and WriteProcessMemory[26], resolves LoadLibraryW[27] from kernel32.dll, and executes it via CreateRemoteThread[28], a classic remote DLL injection.

[23] https://www.virustotal.com/gui/file/0a6241dd8752ad804e5cb882fb8ba0fc46c2c46c53b1c4048dbbb5c8882c497c/

[24] VirusTotal - File - f5449806ad50242a4df9186455434b5957f3fc2044d85a9f1d9135a542b678cc

[25] VirtualAllocEx function (memoryapi.h) - Win32 apps | Microsoft Learn

[26] WriteProcessMemory function (memoryapi.h) - Win32 apps | Microsoft Learn

[27] LoadLibraryW function (libloaderapi.h) - Win32 apps | Microsoft Learn

[28] CreateRemoteThread function (processthreadsapi.h) - Win32 apps | Microsoft Learn

```
lVar3 = FUN_1400035f0((longlong)param_2);
local_20 = lVar3 * 2 + 2;
local_30 = VirtualAllocEx(param_1,(LPVOID)0x0,local_20,0x3000,4);
if (local_30 == (LPVOID)0x0) {
  return 0;
}
lpBuffer = (LPCVOID)FUN_140002c30(param_2);
BVar2 = WriteProcessMemory(param_1,local_30,lpBuffer,local_20,(SIZE_T *)0x0);
if (BVar2 == 0) {
  BVar2 = VirtualFreeEx(param_1,local_30,0,0x8000);
  return CONCAT44(extraout_var,BVar2) & 0xffffffffffffff00;
}
local_18 = GetModuleHandleW(L"Kernel32.dll");
if (local_18 == (HMODULE)0x0) {
  BVar2 = VirtualFreeEx(param_1,local_30,0,0x8000);
  return CONCAT44(extraout_var_00,BVar2) & 0xffffffffffffff00;
}
local_10 = (LPTHREAD_START_ROUTINE)GetProcAddress(local_18,"LoadLibraryW");
if (local_10 == (LPTHREAD_START_ROUTINE)0x0) {
  BVar2 = VirtualFreeEx(param_1,local_30,0,0x8000);
  return CONCAT44(extraout_var_01,BVar2) & 0xffffffffffffff00;
}
local_28 = CreateRemoteThread(param_1,(LPSECURITY_ATTRIBUTES)0x0,0,local_10,local_30,0,
                              (LPDWORD)0x0);
if (local_28 == (HANDLE)0x0) {
  BVar2 = VirtualFreeEx(param_1,local_30,0,0x8000);
  return CONCAT44(extraout_var_02,BVar2) & 0xffffffffffffff00;
}
local_34 = WaitForSingleObject(local_28,10000);
if (local_34 != 0) {
  CloseHandle(local_28);
  BVar2 = VirtualFreeEx(param_1,local_30,0,0x8000);
  return CONCAT44(extraout_var_03,BVar2) & 0xffffffffffffff00;
```

Figure 9: Classic DLL injection

After injection, the service communicates with the injected DLL through a private named pipe (\\.\pipe\Global\SecurityCenterPipe). Through this pipe, the DLL registers a fake Antivirus "Quick Heal"[29] with WSC using the interface a genuine antivirus product would use. The service does not simply connect once and exit. It retries the injection up to four times with five-second intervals if the pipe connection is lost, ensuring persistence is maintained even if the injected DLL is terminated.

[29] Cybersecurity, Antivirus & Network Security Software Companies

```
local_30 = 0xfffffffffffffffe;
basic_string<>(local_80,L"Quick Heal");
basic_string<>(local_28,L"\\\\.\\pipe\\Global\\SecurityCenterPipe");
FUN_140007460(local_60,(basic_string<> *)local_28);
~basic_string<>(local_28);
FUN_1400042c0(local_a0);
uVar4 = FUN_14001b00(&DAT_1400506a0,0x4fe00,(_String_val<> *)local_a0);
```

Figure 10: Quick Heal + pipe initialization

```
while ((int)local_228 < 4) {
  cVar1 = FUN_1400054b0(&DAT_1400b9d0b,5);
  in_RAX = 0;
  if (cVar1 != '\0') break;
  uVar3 = FUN_140004080(param_1);
  if ((uVar3 & 0xff) == 0) {
    if ((int)local_228 < 3) {
      Sleep(5000);
    }
  }
```

Figure 11: Re-injection loop with five seconds interval

6 Stage 3 - Donut Shellcode Loader

The shellcode embedded in setup.exe is built on Donut[4], an open-source shellcode generator that wraps PE payloads for reflective in-memory execution.

The loader uses Chaskey[5] in CTR mode[30] to decrypt the embedded module. The loader also stores and verifies a Chaskey-based MAC[31] associated with the DONUT_INSTANCE structure before executing the decrypted PE.

Attribute	Value
Cipher	Chaskey-CTR with Even-Mansour keying
Cipher key (16 B)	a9 a8 46 9c a0 ac be fa 17 7e 70 03 9b 1d 59 3f
CTR initial value	b9 6f d6 02 bc 6e 69 1f 48 10 79 15 68 68 64 79
DONUT_INSTANCE size	0x6d7c0 (448,448 bytes)
Encrypted module	447,876 bytes

[30] Block cipher mode of operation - Wikipedia

[31] Chaskey: An Efficient MAC Algorithm for 32-bit Microcontrollers

7 Stage 4 - DONUT_MODULE

After Chaskey-CTR decryption of the DONUT_INSTANCE, the Donut loader stub reflectively maps and executes the embedded DONUT_MODULE, a PE file loaded in memory. That PE is the ValleyRAT payload. ValleyRAT subsequently drops the K7RKScan BYOVD kernel driver and the Windows_system_deponcom.exe binary.

SHA-256	Size on disk	Role
380fec9c412cf85f2cded841bd9613d3755fab4a55444537d7ef99f18ce010f3	432 KB	ValleyRAT C2 agent - primary payload
e2461a1724f3676dd861ab38850a19f7f7ca60371d01f69800ffbbf3297581e0	32 KB	K7RKScan BYOVD kernel driver
a899063db35e45811a62c7abc8e63f8a2bc91253c1a875899a7bec8bff924e49	212 KB	Windows_system_deponcom.exe (Blizzard-signed, benign)

7.1 ValleyRAT Payload

This is the primary payload[32] : a 432 KB x64 PE reflectively loaded from the Donut module into memory. It belongs to the ValleyRAT family and communicates over KCP/UDP. This is the stage responsible for establishing communication with the threat actor, maintaining persistence, and orchestrating the BYOVD kernel driver. It is never written to disk as a PE file, existing only as mapped memory for the duration of the infection.

7.1.1 Entry Points (BHPD & SXPD)

This payload has two named exports: BHPD and SXPD. Both exports follow the same initialization sequence, differing only in one step: SXPD first scans running processes via CreateToolhelp32Snapshot[33] to check for the presence of 360Tray.exe: the real-time protection process of Qihoo 360. The result decides whether the kernel driver is used. If Qihoo 360 is not running, the payload goes on to deploy the K7RKScan driver. If Qihoo 360 is running, the driver step is skipped, and the payload moves straight to C2 communication. Targeting Qihoo 360

[32] VirusTotal - File - 380fec9c412cf85f2cded841bd9613d3755fab4a55444537d7ef99f18ce010f3

[33] https://learn.microsoft.com/en-us/windows/win32/api/tlhelp32/nf-tlhelp32-createtoolhelp32snapshot

specifically is not new, prior Silver Fox campaigns delivering ValleyRAT have documented[34] the same product as an evasion target, consistent with the tradecraft where victims are Chinese-speaking users.

Both paths then create the mutex Local\MyLoveMutex_Fixed to prevent duplicate execution, carry out the first-run setup described in Section 7.1.2, and start the C2 manager thread.

```
FUN_18000c3c0((ulonglong *)&local_698,(undefined8 *)L"360Tray.exe",0xb);
FUN_180009710();
iVar3 = FUN_1800096a0();
if (iVar3 == 0) {
  DAT_18006c300 =
       CreateThread((LPSECURITY_ATTRIBUTES)0x0,0,FUN_180009280,(LPVOID)0x0,0,(LPDWORD)0x0);
  WaitForSingleObject(DAT_18006c300,0xffffffff);
  goto LAB_18000ba67;
}
DVar2 = GetFileAttributesW(L"C:\\config\\PD.txt");
if (DVar2 == 0xffffffff) {
  SHCreateDirectoryExW((HWND)0x0,L"C:\\config",(SECURITY_ATTRIBUTES *)0x0);
  SetFileAttributesW(L"C:\\config",2);
  FUN_18000ca60(local_228,L"%s\\Battle.net.exe",L"C:\\config",in_R9);
  GetModuleFileNameW((HMODULE)0x0,(LPWSTR)local_678,0x208);
  lpNewFileName = local_228;
  lpExistingFileName = (LPCWSTR)local_678;
  uVar4 = 1;
  CopyFileW(lpExistingFileName,lpNewFileName,1);
  FUN_180009850(lpExistingFileName,lpNewFileName,uVar4);
  Sleep(1000);
  FUN_18000aa60();
}
```

Figure 12: ValleyRAT initialization sequence

```
void FUN_180009850(undefined8 param_1,undefined8 param_2,undefined8 param_3)

{
  HANDLE hFile;
  undefined8 local_res18 [2];

  local_res18[0] = param_3;
  hFile = CreateFileA("C:\\config\\Windows_system_deponcom.exe",0x40000000,0,
                      (LPSECURITY_ATTRIBUTES)0x0,2,0,(HANDLE)0x0);
  if (hFile != (HANDLE)0xffffffffffffffff) {
    WriteFile(hFile,&DAT_180034fb0,0x34ed0,(LPDWORD)local_res18,(LPOVERLAPPED)0x0);
    CloseHandle(hFile);
  }
  return;
}
```

Figure 13: Call Windows_system_deponcom.exe drop function

[34] Silver Fox's Russian Ruse: ValleyRAT Hits China via Fake Microsoft Teams Attack

7.1.2 Persistence

The payload establishes persistence through a scheduled task, with the mechanism depending on which entry point was invoked. The BHPD branch registers a task named WindowsUpdate under \Microsoft\ through the Task Scheduler COM interface, populating the author field as “Microsoft Corporation” and the description as “Windows Update Helper Service”, neither of which corresponds to a genuine Windows component. The SXPD branch registers a task named WindowsSafe under the same path using a PowerShell Register-ScheduledTask command.

Both tasks are configured with a one-minute repetition interval at the highest privilege level and the task's action is set to execute C:\config\Windows_system_deponcom.exe[35].

```
undefined8 * FUN_18000c9f0(undefined8 *param_1,ulonglong *param_2)

{
  ulonglong *puVar1;

  puVar1 = FUN_18000c170(param_2,(undefined8 *)
                                 "\"\' -WorkingDirectory \'C:\\\'; $trigger = New-ScheduledTaskTrigg
                                 er -Once -At (Get-Date); $trigger.Repetition = (New-ScheduledTaskTr
                                 igger -Once -At (Get-Date) -RepetitionInterval (New-TimeSpan -Minut
                                 es 1)).Repetition; $settings = New-ScheduledTaskSettingsSet -StartW
                                 henAvailable -DontStopOnIdleEnd -AllowStartIfOnBatteries -DontStopI
                                 fGoingOnBatteries; $principal = New-ScheduledTaskPrincipal -UserId
                                 (whoami) -LogonType Interactive -RunLevel Highest; Register-Schedul
                                 edTask -TaskName \'WindowsSafe\' -TaskPath \'\\Microsoft\\\' -Actio
                                 n $action -Trigger $trigger -Settings $settings -Principal $princip
                                 al -Force;\""
                         ,0x25a);
  FUN_18000bc60(param_1,puVar1);
  return param_1;
}
```

Figure 14: PowerShell scheduled task registration

Before registering the task, the payload checks for C:\config\PD.txt. If the file is not present, it creates the hidden directory C:\config\, copies the running host image to C:\config\Battle.net.exe, and drops Windows_system_deponcom.exe. Because the ValleyRAT payload executes reflectively inside the setup.exe process, the image copied to Battle.net.exe is setup.exe itself. The ready-signal file PD.txt is then written to coordinate with the orchestrator.

35 https://www.virustotal.com/gui/file/a899063db35e45811a62c7abc8e63f8a2bc91253c1a875899a7bec8bff924e49

Windows_system_deponcom.exe is a legitimate Blizzard Battle.net Launcher36, carrying a valid Authenticode signature from Blizzard. Its provenance is confirmed by the PDB path retained in the binary:

D:\Jenkins\workspace\BOP_phoenix_release_2.51.0\phoenix\Release\Battle.net Launcher.exe.pdb

The launcher is not a passive decoy. Its normal function is to locate and start Battle.net.exe in its own directory, and the campaign exploits exactly that behaviour: the malware places its self-copy at the precise path and filename the launcher will reach for. On execution, the launcher resolves its own directory via GetModuleFileNameW[37], appends \Battle.net.exe, and starts the resulting path - C:\config\Battle.net.exe, through ShellExecuteExW[38] with the argument --from-launcher.

The effect is that the payload is never started by the scheduled task directly. Each time the task fires, a validly signed Blizzard binary starts it instead.

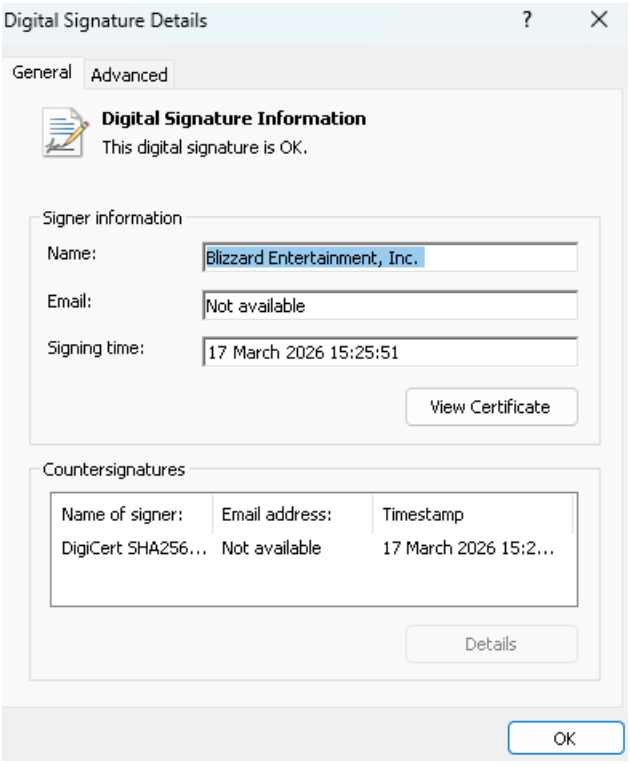

Figure 15: Digital Signature Details for Windows_system_deponcom.exe

[36] https://battle.net

[37] https://learn.microsoft.com/en-us/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulefilenamew

[38] https://learn.microsoft.com/en-us/windows/win32/api/shellapi/nf-shellapi-shellexecuteexw

7.1.3 C2 Configuration and Communication

ValleyRAT's use of a structured C2 configuration with KCP/UDP transport is consistent with previously documented[39] variants of the family. The configuration is stored as reversed UTF-16 strings, requiring each value to be un-reversed before the parser can read it, a simple obfuscation, but one that causes standard string extraction to return nothing usable from the configuration block.

The configuration defines up to three server entries. Each entry specifies an IP address, port, and protocol type, with fields separated by a pipe character.

Field	Value	Role
p1 / o1 / t1	8.210.132.145 / 8888 / 1	Primary C2 - UDP/KCP
p2 / o2 / t2	8.210.132.145 / 8888 / 1	Secondary C2 - same host
p3 / o3 / t3	127.0.0.1 / 80 / 1	Fallback - loopback only

The complete configuration recovered as below:

```
|p1:8.210.132.145|o1:8888|t1:1|p2:8.210.132.145|o2:8888|t2:1|p3:127.0.0.1|o3:80|t3:1|dd:1|cl:1|fz:默认
|bb:1.0|bz:2026. 7. 3|jp:0|bh:0|ll:0|dl:0|sh:0|kl:0|bd:0|
```

The beacon alternates between the first and second servers on each connection attempt, but both point to the same host, so this provides no redundancy if that server goes down. The third entry points to the loopback address and therefore connects to nothing. The configuration also carries build identification, a campaign tag of 默认 (“Default”), version 1.0, and a build date of 3 July 2026, along with a set of feature flags whose behavioural meaning was not established during analysis.

The threat actor can silently rotate the active C2 address without redeploying the implant by writing a new configuration to HKCU\Console\IpDate. The parser checks this registry key on every startup and, if a value is present, overwrites the embedded configuration and re-parses. This allows the threat actor to change the C2 server without redeploying the implant.

The transport is KCP, a reliable data transfer protocol originally designed for low-latency applications such as online gaming, here used over UDP to avoid TCP-based network monitoring.

[39] CL-STA-0048: An Espionage Operation Against High-Value Targets in South Asia

Incoming C2 data is parsed using denglupeizhi (登录配置 - "login configuration") as a payload marker that identifies the boundary between configuration data and shellcode within each received buffer. Received commands are handled through three paths based on the packet type: a heartbeat and module version check, a configuration pull that reads the stored value from the registry, and a configuration push that delivers new executable code. On the push path, the configuration is written to HKLM\SOFTWARE\IpDates_info, while the executable data is prepared in memory for execution. Based on a flag in the received data, the payload is either executed within the current process or passed to tracerpt.exe through the process hollowing routine described in the next section.

For registry operations, the malware uses the fixed value d33f351a4aeea5e608853d1a56661059 under HKCU\Console\1, an identifier previously linked[39] to ValleyRAT.

```
int FUN_180007c50(undefined8 param_1,undefined8 param_2,int param_3)

{
  longlong lVar1;
  int iVar2;
  int iVar3;
  int iVar4;
  longlong lVar5;
  longlong lVar6;

  lVar6 = -1;
  do {
    lVar5 = lVar6 + 1;
    lVar1 = lVar6 + 1;
    iVar6 = lVar5;
  } while ("denglupeizhi"[lVar1] != '\0');
  iVar2 = 0;
  lVar6 = 0;
  iVar4 = (int)lVar5;
  if (0 < param_3) {
    do {
      iVar3 = 0;
      lVar1 = 0;
      if (0 < iVar4) {
        do {
          if (*(char *)(DAT_18006c0e8 + lVar6 + lVar1) != "denglupeizhi"[lVar1]) break;
          lVar1 = lVar1 + 1;
          iVar3 = iVar3 + 1;
        } while (lVar1 < iVar4);
      }
      if (iVar3 == iVar4) {
        return iVar2;
      }
      lVar6 = lVar6 + 1;
      iVar2 = iVar3 + 1;
    } while (lVar6 < param_3);
  }
  return -1;
}
```

Received buffer

Figure 16: ValleyRAT payload marker "denglupeizhi"

7.1.4 Process Hollowing: tracerpt.exe

The payload uses process hollowing to execute its code inside the legitimate Windows process tracerpt.exe[40]. It creates tracerpt.exe in a suspended state, allocates memory, writes the malicious code into it, and then redirects the suspended thread to the injected code. After resuming the thread, the malicious code runs within tracerpt.exe, making the activity appear to originate from a legitimate Windows process.

```
FUN_180007d00(&local_128,"%s%s",&local_128,"Windows\\System32\\tracerpt.exe");
BVar1 = CreateProcessA(&local_128,(LPSTR)0x0,(LPSECURITY_ATTRIBUTES)0x0,(LPSECURITY_ATTRIBUTES)0x0
                       ,0,4,(LPVOID)0x0,(LPCSTR)0x0,&local_668,param_3);
if (((BVar1 != 0) &&
    (lpBaseAddress = VirtualAllocEx(param_3->hProcess,(LPVOID)0x0,(longlong)param_2,0x3000,0x40),
    lpBaseAddress != (LPVOID)0x0)) &&
   (BVar1 = WriteProcessMemory(param_3->hProcess,lpBaseAddress,lpBuffer,(longlong)param_2,
                               (SIZE_T *)0x0), BVar1 != 0)) {
  local_5f8.ContextFlags = 0x10000b;
  BVar1 = GetThreadContext(param_3->hThread,&local_5f8);
  if ((BVar1 != 0) &&
     (local_5f8.Rip = (DWORD64)lpBaseAddress,
     BVar1 = SetThreadContext(param_3->hThread,&local_5f8), BVar1 != 0)) {
    ResumeThread(param_3->hThread);
    return 1;
  }
}
```

Figure 17: tracerpt.exe process hollowing

7.2 K7RKScan BYOVD Kernel Driver

K7RKScan.sys[41] (15.1.0.6) is a legitimate K7 Total Security kernel driver that has been re-signed. It was re-signed with a Thawte certificate issued to Shenzhen yundian Technology Co., Ltd (Serial Number: 4efa7e7bba65ec1ab774f2b31357d599)[42]. This revoked certificate is not new; it is known[43] to be used by other malware families as well.

Its original K7 Computing provenance is preserved in the PDB path:

[40] Tracing - Win32 apps | Microsoft Learn

[41] VirusTotal - File - e2461a1724f3676dd861ab38850a19f7f7ca60371d01f69800ffbbf3297581e0

[42] https://www.loldrivers.io/drivers/04eefdf4-448d-45bb-87fc-93f263fc77f4/

[43] Shedding light on the ABYSSWORKER driver — Elastic Security Labs

k:\k7totalsecurity\ver10\trunk\mainantivirus\k7rkscan\k7rkscandrv\objfre_wlh_amd64\amd64\K7RKScan.pdb

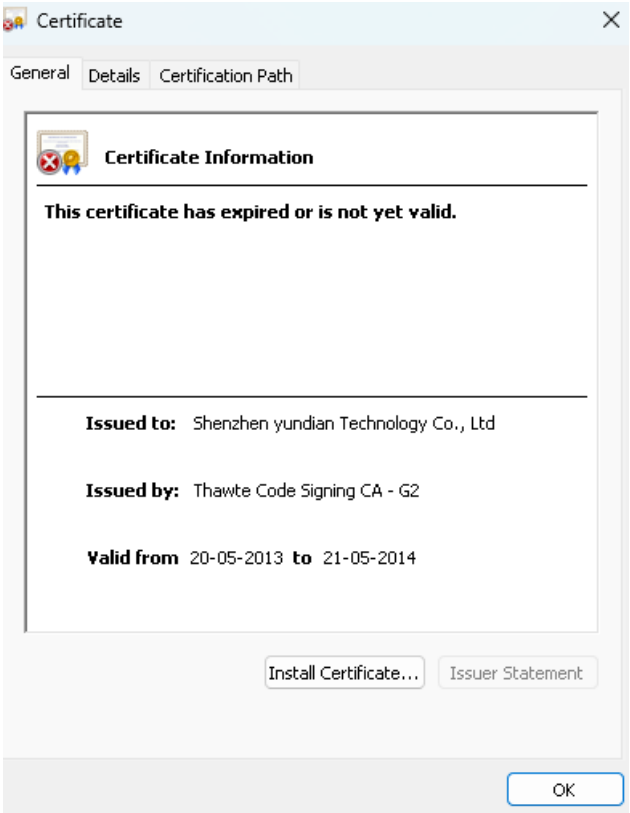

Figure 18: Certificate Information for K7RKScan.sys

That certificate expired in May 2014, more than a decade before this campaign, and Windows flags it as invalid on inspection.

K7RKScan.sys has a documented history of vulnerabilities that make it useful for BYOVD attacks. CVE-2025-1055[44] affected version 15.1.0.6 and allowed low-privileged users to send crafted IOCTL requests to terminate high-privilege processes. Later, CVE-2025-52915[45] was assigned to version 23.0.0.10 after researchers found that the driver still exposed kernel-level process termination functionality, although administrator checks and process filtering had been added. ESET reported[46] this driver being used in the SmilingKiller EDR killer observed during LockBit and

[44] https://nvd.nist.gov/vuln/detail/cve-2025-1055

[45] https://blacksnufkin.github.io/posts/BYOVD-CVE-2025-52915/

[46] https://www.welivesecurity.com/en/eset-research/edr-killers-explained-beyond-the-drivers/

Dire Wolf intrusions, while a 2026 DragonForce campaign[47] used K7RKScan alongside other vulnerable drivers to terminate security processes.

7.2.1 Deployment Sequence

Below sequence was identified through code analysis for driver deployment:

1. Check for administrator privileges:
The malware first checks whether it is running with administrator rights. If the check fails, driver deployment stops.

2. Drop the driver:
The driver is written to %TEMP% using a randomly generated 8-character filename. The filename is generated using the MT19937[48] pseudorandom number generator.

3. Register and load the driver:
ValleyRAT registers the driver with the Windows Service Control Manager as a kernel driver service named DosK7RKScnDrv, configured as DEMAND_START. If an existing service entry causes registration to fail, the malware removes it and retries.

```
  hSCManager = OpenSCManagerW((LPCWSTR)0x0,(LPCWSTR)0x0,0xf003f);
  if (hSCManager == (SC_HANDLE)0x0) {
    return 0;
  }
  pWVar3 = (LPCWSTR)&DAT_180069e80;
  if (7 < DAT_180069e98) {
    pWVar3 = DAT_180069e80;
  }
  hSCObject = CreateServiceW(hSCManager,L"DosK7RKScnDrv",L"DosK7RKScnDrv Driver",0xf01ff,1,3,0,
                             pWVar3,(LPCWSTR)0x0,(LPDWORD)0x0,(LPCWSTR)0x0,(LPCWSTR)0x0,(LPCWSTR)0x0
                            );
  if (hSCObject != (SC_HANDLE)0x0) {
LAB_18000a296:
    CloseServiceHandle(hSCObject);
    BVar2 = CloseServiceHandle(hSCManager);
    return CONCAT71((int7)(CONCAT44(extraout_var,BVar2) >> 8),1);
  }
  DVar1 = GetLastError();
  if (((DVar1 == 0x431) || (DVar1 == 0x7b)) || (DVar1 == 0x57)) {
    CloseServiceHandle(hSCManager);
```

1: kernel driver, 3: demand start

Figure 19: DosK7RKScnDrv service registration via CreateServiceW

[47] https://www.security.com/threat-intelligence/dragonforce-msteams-backdoor

[48] Mersenne Twister - Wikipedia

4. Use the driver to terminate processes:
Once the driver is loaded, ValleyRAT communicates with it using an IOCTL (Input/Output Control) request, the standard Windows mechanism by which an application sends a command to a device driver. Each command has a numeric identifier; the one used here, 0x222018, instructs the driver to terminate a process whose ID the malware supplies.

The driver then locates each target process using PsLookupProcessByProcessId[49] and terminates it with ZwTerminateProcess[50]. It obtains the required kernel-level process handle through ObOpenObjectByPointer[51].

The use of the kernel driver is significant because it allows ValleyRAT to terminate processes protected by Protected Process Light (PPL)[52]. PPL normally prevents untrusted user-mode processes from opening handles to protected security processes. By performing the operation from kernel mode, the malware can bypass this protection.

The observed target set corresponds to Huorong Security[53] processes: HipsDaemon, HipsMain, HipsTray, and wsctrlsvc.

5. Remove the driver:
After terminating the target processes, ValleyRAT deletes the service registration and removes the driver from %TEMP%.

[49] PsLookupProcessByProcessId function (ntifs.h) - Windows drivers | Microsoft Learn
[50] ZwTerminateProcess function (ntddk.h) - Windows drivers | Microsoft Learn
[51] ObOpenObjectByPointer function (ntifs.h) - Windows drivers | Microsoft Learn
[52] Protecting Windows protected processes | Elastic Blog
[53] Huorong Security - Professional Endpoint Protection Software

8 Attribution

Attribution to Silver Fox APT / ValleyRAT is assessed at HIGH confidence. The assessment is based on technical markers within the ValleyRAT payload that match previously published analyses of the malware family.
Beyond these technical markers, the operational patterns observed in this campaign are consistent with Silver Fox activity documented in earlier campaigns, providing additional support for the attribution.

- **Targeting Chinese security products :** The group has a documented[54] focus on disrupting Chinese security products. This campaign continues that pattern with a two-layer approach against Qihoo 360, null-routing its cloud connectivity while terminating Huorong Security processes at the kernel level. Related campaigns show the same targeting logic applied more broadly: an April 2026 campaign[55] queried WMI for Chinese consumer AV processes covering 360 Safe, Tencent PC Manager, and Huorong before selecting an execution path.

- **BYOVD as an established capability :** The use of a vulnerable signed driver to terminate security processes is a recurring Silver Fox technique. A trojanized Telegram installer chain documented[56] in late 2025 leveraged BYOVD to load NSecKrnl64.sys and terminate security solution processes, and an April 2026 MSI campaign[55] deployed the vulnerable wnBios driver to gain access to physical memory. The K7RKScan driver documented here is a different vulnerable driver serving the same operational purpose, consistent with a threat actor that rotates the specific driver while retaining the technique.

- **Trojanized installers and lure infrastructure :** Previous Silver Fox campaigns delivered ValleyRAT and Winos 4.0 through similar lure infrastructure, employing comparable persistence mechanisms and trojanized installer chains. Our own earlier research documented[1] the same actor distributing ValleyRAT through SEO-poisoned fake software download sites.

- **Hosting patterns :** C2 traffic in this campaign targets Alibaba Cloud AS37963 in Hong Kong. Hong Kong hosting is a consistent feature of Silver Fox, the April 2026 Telegram campaign[55] used a C2 server hosted by CTG Server Ltd in Hong Kong, a provider that has appeared[57] in multiple prior Silver Fox operations.

[54] https://threatlabsnews.xcitium.com/blog/silver-fox-apt-spreads-valleyrat-through-fake-microsoft-teams-installer/
[55] https://intel.breakglass.tech/post/silverfox-valleyrat-telegram-chinese-langpack-zpaq-bytedance-ctg
[56] https://www.nextron-systems.com/2025/11/28/thor-vs-silver-fox-uncovering-and-defeating-a-sophisticated-valleyrat-campaign/
[57] https://hunt.io/blog/unearthing-new-infrastructure-by-revisiting-past-threat-reports

8.1 Developer Language Artifacts

Chinese-language strings remain in release builds across multiple chain components, indicating the developers work in a Chinese-language environment.
The denglupeizhi payload marker and the WSC spoofer DLL carries the PDB path D:\WSCM\x64\Release\SecurityProviderCore.pdb. The ValleyRAT payload retains Chinese debug output including 【登录模块.dll】与注册表中记录的版本相同，直接运行 ("Login module.dll matches the version recorded in the registry, running directly"), which would normally be stripped before release.

8.2 A Note on Attribution Confidence

Silver Fox has demonstrated deliberate attempts to mislead attribution in the past: a campaign[34] running from November 2025 used a modified ValleyRAT loader containing Cyrillic elements, likely an intentional move to mislead attribution, impersonating a Russian threat actor while targeting Chinese-speaking users.
No comparable false-flag indicators were observed in this campaign, the Chinese-language artifacts, the targeting of Chinese security products, and the Hong Kong hosting all point in the same direction.

9 Conclusion

Campaign KCPhoenix represents a significant evolution in Silver Fox's ValleyRAT operations. The chain combines sandbox detection, kernel-level process termination through a vulnerable driver, custom payload encryption, and Windows Security Center spoofing to weaken endpoint defenses and maintain control after execution.

The individual components of the chain are replaceable. The K7RKScan driver can be swapped, the encryption layer can be modified, and the infrastructure can be rotated. What is more significant is the underlying pattern: abuse trusted software, disable or bypass security controls, hide execution inside legitimate processes, and remove artifacts when the task is complete.

For defenders, this makes behaviour more valuable than individual indicators. Detection should focus on vulnerable driver loading, attempts to terminate security processes, manipulation of Windows Security Center, process injection and hollowing, suspicious scheduled tasks, and unusual UDP-based C2 activity. These behaviours are likely to remain relevant even as the actor changes the specific tools and infrastructure used in future campaigns.

10 MITRE ATT&CK Mapping

Tactic	ID	Technique	Implementation
Initial Access	T1204.002	User Execution: Malicious File	Trojanized AnyDesk installer distributed via social engineering
Execution	T1059.001	Command and Scripting: PowerShell	Scheduled task registration via Register-ScheduledTask
Execution	T1059.006	Command and Scripting: Python	PyInstaller dropper executing embedded PE via subprocess.run()
Execution	T1106	Native API	VirtualAlloc(RWX) shellcode execution; CreateRemoteThread DLL injection
Persistence	T1053.005	Scheduled Task/Job: Scheduled Task	\Microsoft\WindowsUpdate (COM) and \Microsoft\WindowsSafe (PowerShell), every 1 minute
Persistence	T1543.003	Create or Modify System Process: Windows Service	SafeCenterService (WSC spoofer); DosK7RKScnDrv (BYOVD driver)
Persistence	T1112	Modify Registry	HKLM\SOFTWARE\IpDates_info; HKCU\Console\IpDate C2 rotation
Priv. Escalation	T1068	Exploitation for Privilege Escalation	BYOVD kernel primitive - ObOpenObjectByPointer KernelMode PPL bypass
Defense Evasion	T1055.012	Process Injection: Process Hollowing	tracerpt.exe RIP-register hijack via CreateProcessA(CREATE_SUSPENDED)
Defense Evasion	T1027	Obfuscated Files or Information	XOR(0x5A) shellcode encryption; Chaskey-CTR payload encryption
Defense Evasion	T1562.001	Impair Defenses: Disable or Modify Tools	360 AV null-route (user-mode) + BYOVD kernel kill
Defense Evasion	T1036.005	Masquerading: Match Legitimate Name or Location	"Calculator.exe" internal name; "Battle.net.exe" self-copy; "Quick Heal" WSC registration
Defense Evasion	T1070.004	Indicator Removal: File Deletion	K7RKScan driver file and service deleted after use

Defense Evasion	T1574	Hijack Execution Flow	Malicious Battle.net.exe placed where the signed Blizzard launcher resolves and executes it
Defense Evasion	T1553.002	Subvert Trust Controls: Code Signing	Payload executed via a validly signed Blizzard binary acting as launcher
Discovery	T1057	Process Discovery	Sandbox discovery via CreateToolhelp32Snapshot; 360Tray.exe detection
Discovery	T1497.001	Sandbox Evasion: System Checks	Sandbox checks - DNS suffix, registry keys, file paths, active user
C2	T1095	Non-Application Layer Protocol	KCP/UDP transport (not TCP/HTTP)
C2	T1571	Non-Standard Port	UDP port 8888

11 Detection Opportunities

- **Qihoo 360 cloud disruption**
  Monitor for route -p add commands specifying a null gateway (0.0.0.0), particularly from non-interactive sessions.
- **K7RKScan driver installation**
  Detect creation of the DosK7RKScnDrv service or the \Device\NTK7RKScnDrv device name. Known K7RKScan driver hashes can also be added to blocklists.
- **Suspicious scheduled tasks**
  Monitor for new tasks created under \Microsoft\ named WindowsUpdate or WindowsSafe, particularly when registered by PowerShell or non-system processes.
- **WSC COM manipulation**
  Look for non-security processes calling IWscProduct2::Register() or taskmgr.exe loading unexpected DLLs.
- **ValleyRAT registry activity**
  Monitor creation or modification of HKCU\Console\IpDate, HKLM\SOFTWARE\IpDates_info, and the known d33f351a4aeea5e608853d1a56661059 registry value.
- **tracerpt.exe process hollowing**
  Detect tracerpt.exe being created in a suspended state followed by remote memory allocation or memory writing from another process.
- **Known ValleyRAT mutex**
  Monitor creation of the Local\MyLoveMutex_Fixed
- **Suspicious C:\config directory**
  Detect creation of C:\config with hidden attributes by non-system processes.

12 Indicators of Compromise

12.1 File Hashes

SHA-256	Filename
8e8c9e00a07b3051699b3426e06db01f57814c1377748de00ca9e719bdd4392b	AnyDesk.exe
bad1a1eeb66cfe7d61f559da36112d2f4de8f6df8bdbb66a308764c9ced e77ba	setup.exe
0a6241dd8752ad804e5cb882fb8ba0fc46c2c46c53b1c4048dbbb5c8 882c497c	SafeCenterController.exe
f5449806ad50242a4df9186455434b5957f3fc2044d85a9f1d9135a542b 678cc	SecurityProviderCore.dll
380fec9c412cf85f2cded841bd9613d3755fab4a55444537d7ef99f18ce0 10f3	ValleyRAt payload
e2461a1724f3676dd861ab38850a19f7f7ca60371d01f69800ffbbf329758 1e0	K7RKScan.sys
9f1effd1929bb3af3318e1500af2221b990e3ceb5bdae2d327e3fc8929fc 41dc	AweSun.exe (decoy)
a899063db35e45811a62c7abc8e63f8a2bc91253c1a875899a7bec8bff 924e49	Windows_system_deponc om.exe

12.2 Additional Campaign samples

The following samples were identified as belonging to the same campaign. Each was found to deploy a K7RKScan driver variant and to carry PDB paths matching those recovered from the analyzed chain. They are provided to support hunting across the wider lure set.

SHA-256	Filename
045200c6f89434bbe63d970d4aeaaaee545e2c313eb261b0600385c 388322f4e	TD_setup_x86_7.27.0.9.5.7. exe
cbb932c31c2759e5d78faeb8dfb1bb5f8920eb9838b949f6c1d75faa38f 1401b	LetsVPN.exe

8fd76ab00c09e7dcb23e5f08b5953c364f16fa2412ad04492bfa8d075321fa6b	Bitbrowser_x64_v727.1613.exe
fef2d79555a277b7b344e055ad35e88015f29da615c1c616b97291576124e391	chorme_setup_x64_win_1.540.727.62.exe
17e6770534199a3a734a2df9c7e66a0c615422138Ofe744efdda2e409cd1ab3c	chorme_setup_x64_win_1.744.297.29.exe
764b50abc26a0d14e576c614fbfa8cf4fd66ac27a20cafdce5e257083421e21f	LetsVPN.exe
81b0f7b11389ab132a32210706020c3bda333af528d9f0a05f20b017d5feabd6	chorme_setup_x64_win_2.325.212.68.exe

12.3 Filesystem Paths

Path	Role
C:\config\	Staging directory - created with HIDDEN attribute
C:\config\Battle.net.exe	Core Orchestrator self-copy (persistence)
C:\config\PD.txt	Stage-sync gate file (ValleyRAT >>setup.exe signal)
C:\config\pd1.txt	Decoy launch gate (written by C2 server)
C:\config\Windows_system_deponcom.exe	Signed Blizzard launcher used to start Battle.net.exe
C:\AweSun\setup.exe	Initial core orchestrator location
C:\AweSun\AweSun.exe	AnyDesk v9.7.4 legitimate decoy
C:\ProgramData\SecurityCenter\Runtime\SecurityProviderCore.dll	WSC spoofer DLL drop path

%TEMP%\<8-random-alpha>.sys	K7RKScan driver
Battle.net.exe --from-launcher	Command line used when the signed Blizzard launcher starts core orchestrator execution

12.4 Registry Keys

Key	Purpose
HKCU\Console\1\d33f351a4aeea5e608853d1a56661059	ValleyRAT session data
HKLM\SOFTWARE\IpDates_info	ValleyRAT persistence data
HKCU\Console\IpDate	Threat actor-controlled
HKLM\SYSTEM\CurrentControlSet\Services\SafeCenterService	WSC spoofer service registration
HKLM\SYSTEM\CurrentControlSet\Services\DosK7RKScnDrv	BYOVD driver service

12.5 Services, Scheduled Tasks, Mutex, and Pipes

Indicator	Role
Service: SafeCenterService	AUTO_START, SYSTEM :WSC spoofer controller
Service: DosK7RKScnDrv	BYOVD driver service
Device: \Device\NTK7RKScnDrv	K7RKScan kernel driver device object
Device: \DosDevices\DosK7RKScnDrv	K7RKScan symbolic link
Scheduled task: \Microsoft\WindowsUpdate	Persistence
Scheduled task: \Microsoft\WindowsSafe	Persistence
Mutex: Local\MyLoveMutex_Fixed	Mutex

Named pipe: \\.\pipe\Global\SecurityCenterPipe	WSC spoofer inter-process communication

12.6 Network IOCs

Indicator	Role
8.210.132.145:8888/UDP	Primary and secondary C2 (KCP over UDP)
anydesk[.]nx[.]cn	Distribution site for AnyDesk

13 YARA Rule

```
import "pe"

rule KCPhoenix_Orchestrator_MFC_Setup
{
   meta:
      description   = "setup.exe / C:\\config\\Battle.net.exe - x64 MFC orchestrator masquerading
as Calculator.exe"
      status      = "experimental"
      author      = "Atos TRC"
      date        = "2026-08-28"
       malware_family = "ValleyRAT"
      campaign     = "KCPhoenix"
      sha256      = "bad1a1eeb66cfe7d61f559da36112d2f4de8f6df8bdbb66a308764c9cede77ba"
      mitre_attack  = "T1036.005, T1497.001, T1562.001, T1543.003"

   strings:
      // PE version-resource masquerade
      $internal_name = "Calculator.exe" wide

      // Sandbox detection battery (Section 4.1)
      $sb_wdag      = "WDAGUtilityAccount" wide
      $sb_cexec     = "CExecSvc.exe" wide
      $sb_dns       = "mshome.net" wide
      $sb_wcsandbox  = "WcSandboxState" wide

      // Stage-sync gate files
      $f_pd1       = "C:\\config\\pd1.txt" ascii wide
      $f_pd        = "C:\\config\\PD.txt" ascii wide
      $f_configdir   = "C:\\config" ascii wide

      // Qihoo 360 cloud disruption (Section 4.2)
      $cmd_netstat   = "netstat -ano" ascii wide
      $cmd_route     = "route -p add" ascii wide
```

```
        // WSC spoofer service dropped and installed by this stage
        $svc_safe     = "SafeCenterService" ascii wide
        $svc_seccenter = "SecurityCenterService" ascii wide

        // "This PC" shell namespace CLSID used for the decoy launch
        $clsid_thispc  = "{20D04FE0-3AEA-1069-A2D8-08002B30309D}" ascii wide

    condition:
        uint16(0) == 0x5A4D
        and pe.machine == pe.MACHINE_AMD64
        and filesize > 15MB and filesize < 30MB
        and $internal_name
        and 3 of ( $sb_*, $f_*, $cmd_*, $svc_*, $clsid_thispc )
}

rule KCPhoenix_WSC_Spoofer
{
    meta:
        description   = "SafeCenterController.exe / SecurityProviderCore.dll - registers a fake AV
product with Windows Security Center"
       status      = "experimental"
        author      = "Atos TRC"
        date        = "2026-08-28"
        malware_family = "ValleyRAT"
        campaign     = "KCPhoenix"
        sha256_exe    =
"0a6241dd8752ad804e5cb882fb8ba0fc46c2c46c53b1c4048dbbb5c8882c497c"
        sha256_dll    = "f5449806ad50242a4df9186455434b5957f3fc2044d85a9f1d9135a542b678cc"
        mitre_attack  = "T1562.001, T1055.001, T1543.003, T1036.005"

    strings:
        // Private IPC channel between the service and the injected DLL
        $pipe      = "\\\\.\\pipe\\Global\\SecurityCenterPipe" ascii wide
```

```
    // Fake product name registered with WSC
    $product    = "Quick Heal" ascii wide

    // Service names
    $svc_sec    = "SecurityCenterService" ascii wide

    // DLL drop path and injection target
    $drop_path   = "SecurityCenter\\Runtime" ascii wide
    $inject_tgt  = "taskmgr.exe" ascii wide

    // Command verbs logged by the DLL's WSC request handler
    $verb_reg    = "REGISTER_PROD" ascii
    $verb_name   = "SET_DISPLAY_NAME" ascii
    $handler     = "WscRequestHandler" ascii

    // Developer artefact retained in the DLL
    $pdb         = "SecurityProviderCore.pdb" ascii

  condition:
    uint16(0) == 0x5A4D
    and filesize < 5MB
    and 3 of them
}


rule KCPhoenix_ValleyRAT_KCP_Payload
{
  meta:
    description   = "ValleyRAT KCP/UDP C2 agent - denglupeizhi marker, tracerpt.exe hollowing,
K7RKScan BYOVD"
    status       = "experimental"
    author       = "Atos TRC"
    date         = "2026-08-28"
    malware_family = "ValleyRAT"
    threat_actor  = "Silver Fox APT"
```

```
    campaign     = "KCPhoenix"
    sha256       = "380fec9c412cf85f2cded841bd9613d3755fab4a55444537d7ef99f18ce010f3"
    mitre_attack  = "T1095, T1055.012, T1053.005, T1562.001, T1068"
    note         = "Payload is reflectively loaded and not written to disk as a PE. Intended for
memory scans and carved/extracted samples."

  strings:
    // Family protocol marker - strongest single indicator
    $marker     = "denglupeizhi" ascii wide

    // BYOVD driver service and device names
    $k7_svc     = "DosK7RKScnDrv" ascii wide
    $k7_dev     = "\\\\.\\DosK7RKScnDrv" ascii wide

    // Kernel process-termination IOCTL 0x222018, little-endian DWORD
    $ioctl_kill  = { 18 20 22 00 }

    // Huorong Security processes terminated through the driver
    $tgt_hips1   = "HipsDaemon.exe" wide
    $tgt_hips2   = "HipsMain.exe" wide
    $tgt_hips3   = "HipsTray.exe" wide
    $tgt_hips4   = "wsctrlsvc.exe" wide

    // Qihoo 360 presence check that gates driver deployment
    $gate_360    = "360Tray.exe" wide

    // Process hollowing target
    $hollow     = "tracerpt.exe" ascii wide

    // Scheduled task persistence - both entry-point variants.
    $task_safe   = "WindowsSafe" ascii wide
    $task_update = "WindowsUpdate" wide
    $task_ps     = "Register-ScheduledTask" ascii wide

    // Single-instance mutex
```

```
    $mutex     = "MyLoveMutex_Fixed" ascii wide

    // Staging artefacts
    $f_pd      = "C:\\config\\PD.txt" ascii wide
    $f_battlenet = "Battle.net.exe" ascii wide
    $f_deponcom  = "Windows_system_deponcom.exe" ascii wide

    // Registry artefacts (sub-key paths only - see file header note)
    $reg_ipdates = "IpDates_info" ascii wide
    $reg_guid    = "d33f351a4aeea5e608853d1a56661059" ascii wide

    // C2 configuration stored reversed: "541.231.012.8" = "8.210.132.145"
    $cfg_c2_rev  = "541.231.012.8" ascii wide
    // Reversed config token grammar, e.g. "|1:1t|" and "|1o:"
    $cfg_token   = "|1:1t|" ascii wide

    // Chinese debug output retained in the release build (UTF-8)
    $dbg_denglu  = { E7 99 BB E5 BD 95 E6 A8 A1 E5 9D 97 }  // 登录模块 "login module"
    $dbg_zhijie  = { E7 9B B4 E6 8E A5 E8 BF 90 E8 A1 8C }  // 直接运行 "run directly"

  condition:
    uint16(0) == 0x5A4D
    and pe.machine == pe.MACHINE_AMD64
    and filesize > 200KB and filesize < 1MB
    and $marker
    and 3 of ( $k7_*, $ioctl_kill, $tgt_hips*, $gate_360, $hollow,
           $task_*, $mutex, $f_*, $reg_*, $cfg_*, $dbg_* )
}


rule KCPhoenix_Signed_Launcher_Proxy_Chain
{
  meta:
    description  = "Detects the C:\\config staging set - signed Blizzard launcher co-located with
the ValleyRAT self-copy"
```

```
    status      = "experimental"
    author      = "Atos TRC"
    date        = "2026-08-27"
    tlp         = "TLP:CLEAR"
    campaign    = "KCPhoenix"
    mitre_attack  = "T1574, T1553.002, T1053.005"
    note        = "Low-volume hunting rule. The launcher itself is a clean signed Blizzard binary;
this rule targets components that reference the proxy-execution arrangement."

  strings:
    $deponcom   = "Windows_system_deponcom.exe" ascii wide
    $arg        = "--from-launcher" ascii wide
    $battlenet  = "C:\\config\\Battle.net.exe" ascii wide
    $pdb_phoenix = "BOP_phoenix_release_2.51.0" ascii

  condition:
    uint16(0) == 0x5A4D
    and filesize < 50MB
    and 2 of them
}
```

About Atos Group

Atos Group is a global leader in digital transformation with c. 56,000 employees and annual revenue of c. €7.2 billion (at the go-forward perimeter), operating in 54 countries under two brands - Atos for services and Eviden for products and systems. European number one in cybersecurity and a leader in cloud, Atos Group is committed to a secure and decarbonized future and provides tailored AI-powered, end-to-end solutions for all industries. Atos Group is listed on Euronext Paris.

Find out more about us

atos.net
atos.net/career