[RE018-2] Analyzing new malware of China Panda hacker group used to attack supply chain against Vietnam Government Certification Authority - Part 2

blog.vincss.net/2020/12/re018-2-analyzing-new-malware-of-china-panda-hacker-group-used-to-attack-supply-chain-against-vietnam-government-certification-authority.html

IV. The relevant evidence to China Panda hacker group

Smanager_ssl.dll was built with Visual Studio (VS) 2015, build timestamp: Sunday, 26.04.2020 15:11:24 UTC, which was 04/26/2020 - 10:11:24 PM Vietnam time (GMT +7). Linker version 14.00 is from VS 2015 and after that, VS 2017, 2019,… still remains 14.xx.

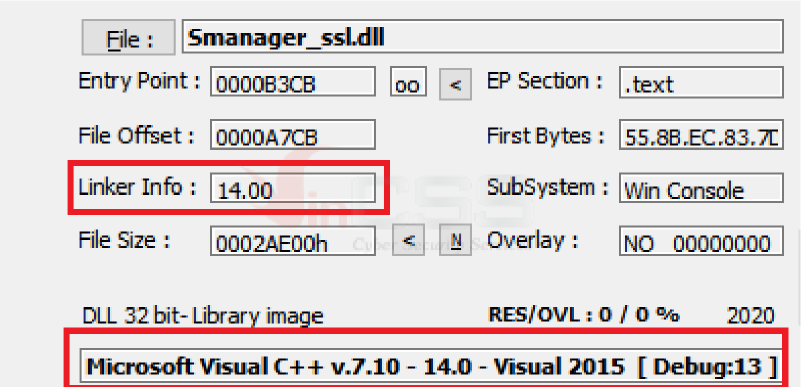
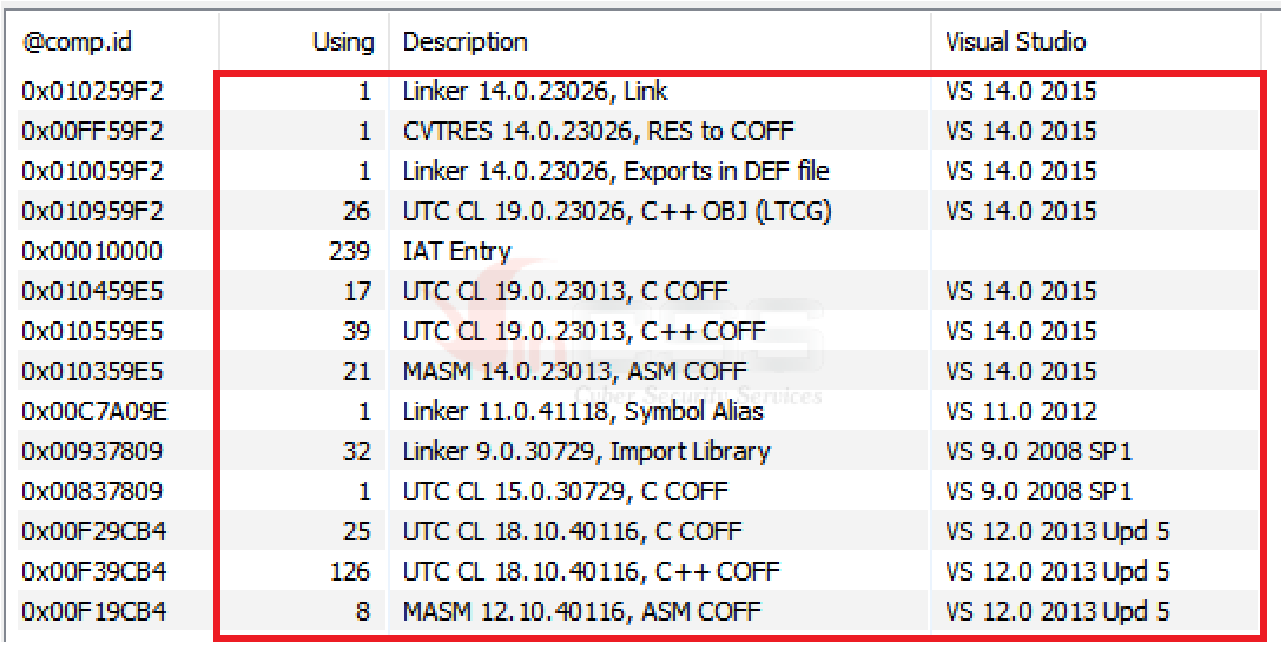

File : Smanager_ssl.dll
Entry Point : 0000B3CB
EP Section : .text
File Offset : 0000A7CB
First Bytes : 55.8B.EC.83.7
Linker Info : 14.00
SubSystem : Win Console
File Size : 0002AE00h
Overlay : NO 00000000
DLL 32 bit- Library image
RES/OVL : 0 / 0 % 2020
Microsoft Visual C++ v.7.10 - 14.0 - Visual 2015 [Debug:13]

Figure 1. Linker information

Information about PE RichID of **Smanager_ssl.dll**:



@comp.id	Using	Description	Visual Studio
0x010259F2	1	Linker 14.0.23026, Link	VS 14.0 2015
0x00FF59F2	1	CVTRES 14.0.23026, RES to COFF	VS 14.0 2015
0x010059F2	1	Linker 14.0.23026, Exports in DEF file	VS 14.0 2015
0x010959F2	26	UTC CL 19.0.23026, C++ OBJ (LTCG)	VS 14.0 2015
0x00010000	239	IAT Entry	
0x010459E5	17	UTC CL 19.0.23013, C COFF	VS 14.0 2015
0x010559E5	39	UTC CL 19.0.23013, C++ COFF	VS 14.0 2015
0x010359E5	21	MASM 14.0.23013, ASM COFF	VS 14.0 2015
0x00C7A09E	1	Linker 11.0.41118, Symbol Alias	VS 11.0 2012
0x00937809	32	Linker 9.0.30729, Import Library	VS 9.0 2008 SP1
0x00837809	1	UTC CL 15.0.30729, C COFF	VS 9.0 2008 SP1
0x00F29CB4	25	UTC CL 18.10.40116, C COFF	VS 12.0 2013 Upd 5
0x00F39CB4	126	UTC CL 18.10.40116, C++ COFF	VS 12.0 2013 Upd 5
0x00F19CB4	8	MASM 12.10.40116, ASM COFF	VS 12.0 2013 Upd 5

Figure 2. PE RichID information

Based on PE RichID and VS version, our Threat Intelligence Platform for malware detected a subset of a sample set with the same PE RichID and VS version. This set of samples was also used by a group of hackers in an APT campaign targeted a large corporation in Vietnam from the end of 2018 to the end of 2019. We collected the sample and analyzed them afterwards. But for some reasons, we couldn't publish the analysis.

In the subset, we paid special attention to the following samples, which are PE x64:

1. **msiscsi.dll**:

- MD5: F61B44ECF57EA6D0F49A7DC2C4456E89

- SHA256: F654E98695E642416A74AF92776A4D24DC55249CEE354D1E868D7C3ACD26030

- Build timestamp: Tuesday, 24.09.2019 01:03:41 UTC

- PDB Path: N:\DEV\MMPro\x64\Release\8.1.pdb (*8.1.dll*)

- Export: ServiceMain, run as a Service Dll.

2. **verifierpr.dll**:

- MD5: FD35D50D1D30275DC216263B906F9F9A

- SHA256: 9B2C8D17F4296DF83F5AE05CFA049DF2243A5303A0310C38C4C4796319A53234

- Build timestamp: Thursday, 24.01.2019 23:55:44 UTC

- PDB Path: C:\Dev\18M\x64\Release\8.pdb (*8.dll*)

- Export: DllGetClassObject

3. **wercplsupport.dll**:

- MD5: 2644C5916A7B49FD216DA16B1F798D3A

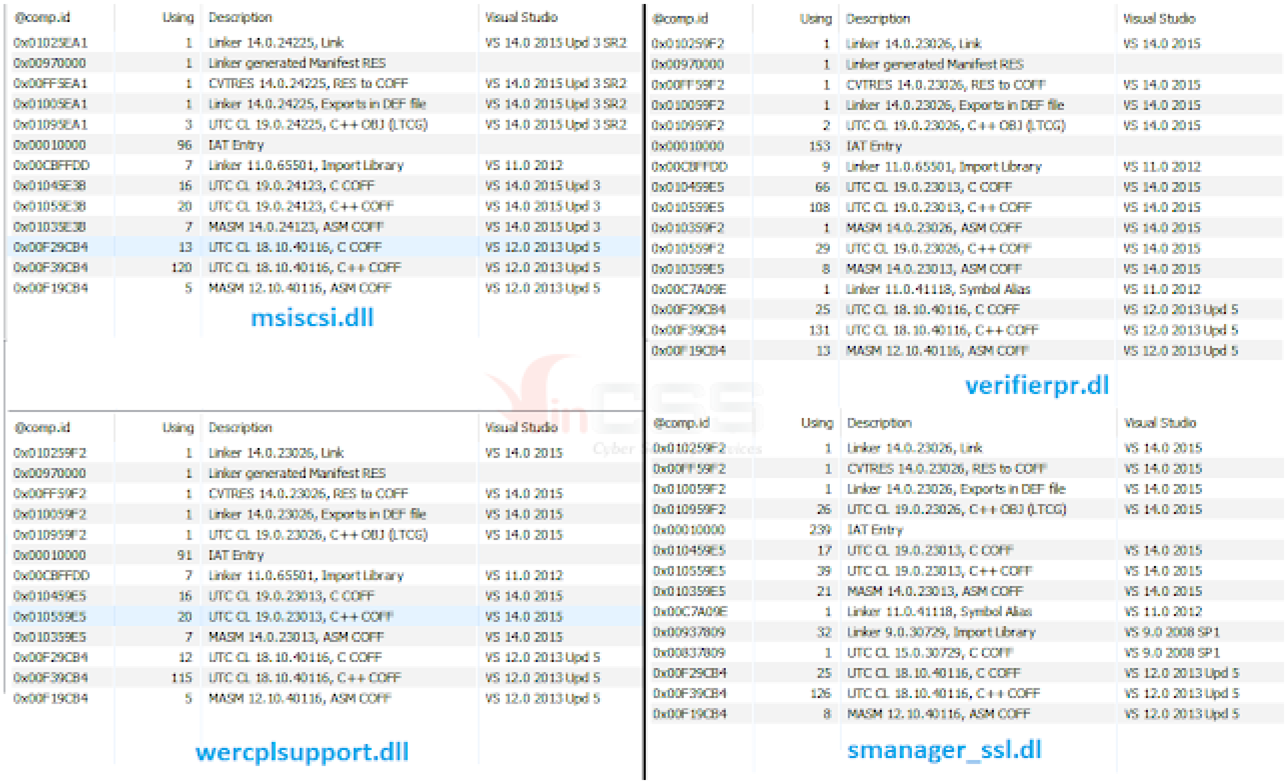
- SHA256: B9E07FF5109CC340D6CB371AFD8D112EBE29BFC1E2D395A28F04761E627D0E39

- Build timestamp: Thursday, 24.01.2019 23:56:17 UTC

- PDB Path: C:\Dev\18M\x64\Release\8.1.pdb

- Export: ServiceMain, run as a Service Dll

Comparison table for PE RichID of the above files and **smanager_ssl.dll** file:



@comp.id	Using	Description	Visual Studio
0x01025EA1	1	Linker 14.0.24225, Link	VS 14.0 2015 Upd 3 SR2
0x00970000	1	Linker generated Manifest RES	
0x00FF5EA1	1	CVTRES 14.0.24225, RES to COFF	VS 14.0 2015 Upd 3 SR2
0x01005EA1	1	Linker 14.0.24225, Exports in DEF file	VS 14.0 2015 Upd 3 SR2
0x01095EA1	3	UTC CL 19.0.24225, C++ OBJ (LTCG)	VS 14.0 2015 Upd 3 SR2
0x00010000	96	IAT Entry	
0x00CBFFDD	7	Linker 11.0.65501, Import Library	VS 11.0 2012
0x01045E38	16	UTC CL 19.0.24123, C COFF	VS 14.0 2015 Upd 3
0x01055E38	20	UTC CL 19.0.24123, C++ COFF	VS 14.0 2015 Upd 3
0x01035E38	7	MASM 14.0.24123, ASM COFF	VS 14.0 2015 Upd 3
0x00F29CB4	13	UTC CL 18.10.40116, C COFF	VS 12.0 2013 Upd 5
0x00F39CB4	120	UTC CL 18.10.40116, C++ COFF	VS 12.0 2013 Upd 5
0x00F19CB4	5	MASM 12.10.40116, ASM COFF	VS 12.0 2013 Upd 5

msiscsi.dll

@comp.id	Using	Description	Visual Studio
0x010259F2	1	Linker 14.0.23026, Link	VS 14.0 2015
0x00970000	1	Linker generated Manifest RES	
0x00FF59F2	1	CVTRES 14.0.23026, RES to COFF	VS 14.0 2015
0x010059F2	1	Linker 14.0.23026, Exports in DEF file	VS 14.0 2015
0x010959F2	2	UTC CL 19.0.23026, C++ OBJ (LTCG)	VS 14.0 2015
0x00010000	153	IAT Entry	
0x00CBFFDD	9	Linker 11.0.65501, Import Library	VS 11.0 2012
0x010459E5	66	UTC CL 19.0.23013, C COFF	VS 14.0 2015
0x010559E5	108	UTC CL 19.0.23013, C++ COFF	VS 14.0 2015
0x010359F2	1	MASM 14.0.23026, ASM COFF	VS 14.0 2015
0x010559F2	29	UTC CL 19.0.23026, C++ COFF	VS 14.0 2015
0x010359E5	8	MASM 14.0.23013, ASM COFF	VS 14.0 2015
0x00C7A09E	1	Linker 11.0.41118, Symbol Alias	VS 11.0 2012
0x00F29CB4	25	UTC CL 18.10.40116, C COFF	VS 12.0 2013 Upd 5
0x00F39CB4	131	UTC CL 18.10.40116, C++ COFF	VS 12.0 2013 Upd 5
0x00F19CB4	13	MASM 12.10.40116, ASM COFF	VS 12.0 2013 Upd 5

verifierpr.dl

@comp.id	Using	Description	Visual Studio
0x010259F2	1	Linker 14.0.23026, Link	VS 14.0 2015
0x00970000	1	Linker generated Manifest RES	
0x00FF59F2	1	CVTRES 14.0.23026, RES to COFF	VS 14.0 2015
0x010059F2	1	Linker 14.0.23026, Exports in DEF file	VS 14.0 2015
0x010959F2	1	UTC CL 19.0.23026, C++ OBJ (LTCG)	VS 14.0 2015
0x00010000	91	IAT Entry	
0x00CBFFDD	7	Linker 11.0.65501, Import Library	VS 11.0 2012
0x010459E5	16	UTC CL 19.0.23013, C COFF	VS 14.0 2015
0x010559E5	20	UTC CL 19.0.23013, C++ COFF	VS 14.0 2015
0x010359E5	7	MASM 14.0.23013, ASM COFF	VS 14.0 2015
0x00F29CB4	12	UTC CL 18.10.40116, C COFF	VS 12.0 2013 Upd 5
0x00F39CB4	115	UTC CL 18.10.40116, C++ COFF	VS 12.0 2013 Upd 5
0x00F19CB4	5	MASM 12.10.40116, ASM COFF	VS 12.0 2013 Upd 5

wercplsupport.dll

@comp.id	Using	Description	Visual Studio
0x010259F2	1	Linker 14.0.23026, Link	VS 14.0 2015
0x00FF59F2	1	CVTRES 14.0.23026, RES to COFF	VS 14.0 2015
0x010059F2	1	Linker 14.0.23026, Exports in DEF file	VS 14.0 2015
0x010959F2	26	UTC CL 19.0.23026, C++ OBJ (LTCG)	VS 14.0 2015
0x00010000	239	IAT Entry	
0x010459E5	17	UTC CL 19.0.23013, C COFF	VS 14.0 2015
0x010559E5	39	UTC CL 19.0.23013, C++ COFF	VS 14.0 2015
0x010359E5	21	MASM 14.0.23013, ASM COFF	VS 14.0 2015
0x00C7A09E	1	Linker 11.0.41118, Symbol Alias	VS 11.0 2012
0x00937809	32	Linker 9.0.30729, Import Library	VS 9.0 2008 SP1
0x00837809	1	UTC CL 15.0.30729, C COFF	VS 9.0 2008 SP1
0x00F29CB4	25	UTC CL 18.10.40116, C COFF	VS 12.0 2013 Upd 5
0x00F39CB4	126	UTC CL 18.10.40116, C++ COFF	VS 12.0 2013 Upd 5
0x00F19CB4	8	MASM 12.10.40116, ASM COFF	VS 12.0 2013 Upd 5

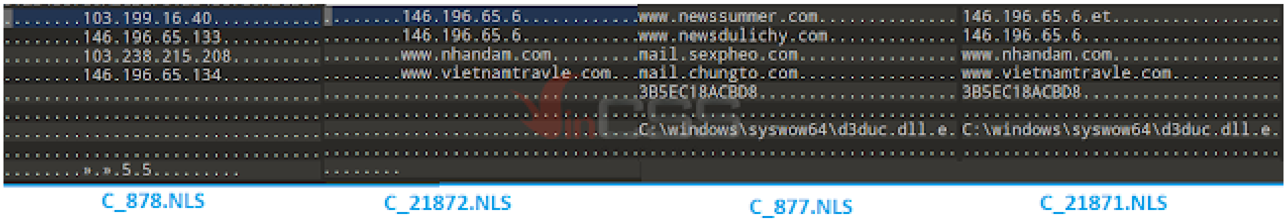
smanager_ssl.dl

Figure 3. Comparison table for PE RichID

Focus on the Description column, version of the components compiler/linker/... in the Visual Studio. For many of the samples in that sample set, we think that this hacking group has many members and also has a Source Code Control server.

The C&C info is stored in **.nls**, impersonating the main Windows **.nls** files, in the **Windows\System32** folder. NLS file is National Language Support files.

We decoded some of the C&C as follow:



C_878.NLS

```
.........103.199.16.40.............
.........146.196.65.133............
.........103.238.215.208...........
.........146.196.65.134............
...................................
...................................
...................................
...................................
.........».».5.5.........
```

C_21872.NLS

```
.........146.196.65.6..............
.........146.196.65.6..............
........www.nhandam.com............
........www.vietnamtravle.com......
...................................
...................................
...................................
...................................
.........
```

C_877.NLS

```
www.newssummer.com.................
www.newsdulichy.com................
mail.sexpheo.com...................
mail.chungto.com...................
3B5EC18ACBD8.......................
...................................
C:\windows\syswow64\d3duc.dll.e.
...................................
```

C_21871.NLS

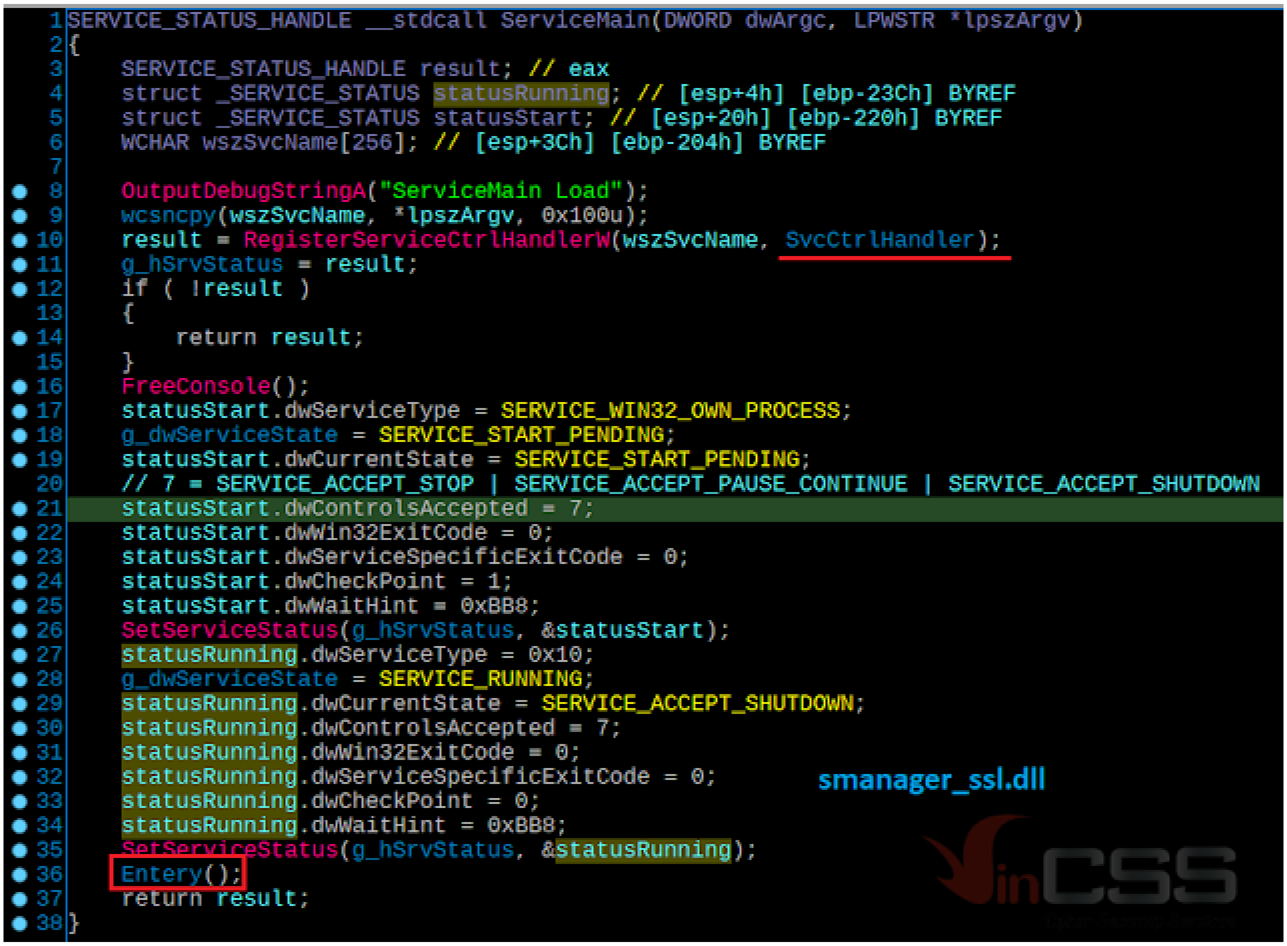
```
146.196.65.6.et....................
146.196.65.6.......................
www.nhandam.com....................
www.vietnamtravle.com..............
3B5EC18ACBD8.......................
...................................
C:\windows\syswow64\d3duc.dll.e.
...................................
```

Figure 4. C&Cs information

With **smanager_ssl.dll** and almost all of the samples we have collected, we noticed that the hacker changed the default calling convention of the VC ++ compiler in the VS IDE (*or command line*) to **__fastcall**. This made for difficult analyzing, recreate the source code of the malware, give the correct definition of the function protytype.

As mentioned in previous part, **Smanager_ssl.dll** is registered by **eToken.exe** (**VVSup.exe**) and run as a Service Dll. We compare the **ServiceMain** function (*which is required of a Service Dll*) and find almost the same code and coding style. We speculate that the code for Service is a file and is generally used for many samples. The **ServiceMain** function is always responsible for calling the main function, which is the function that performs the main tasks of malware.

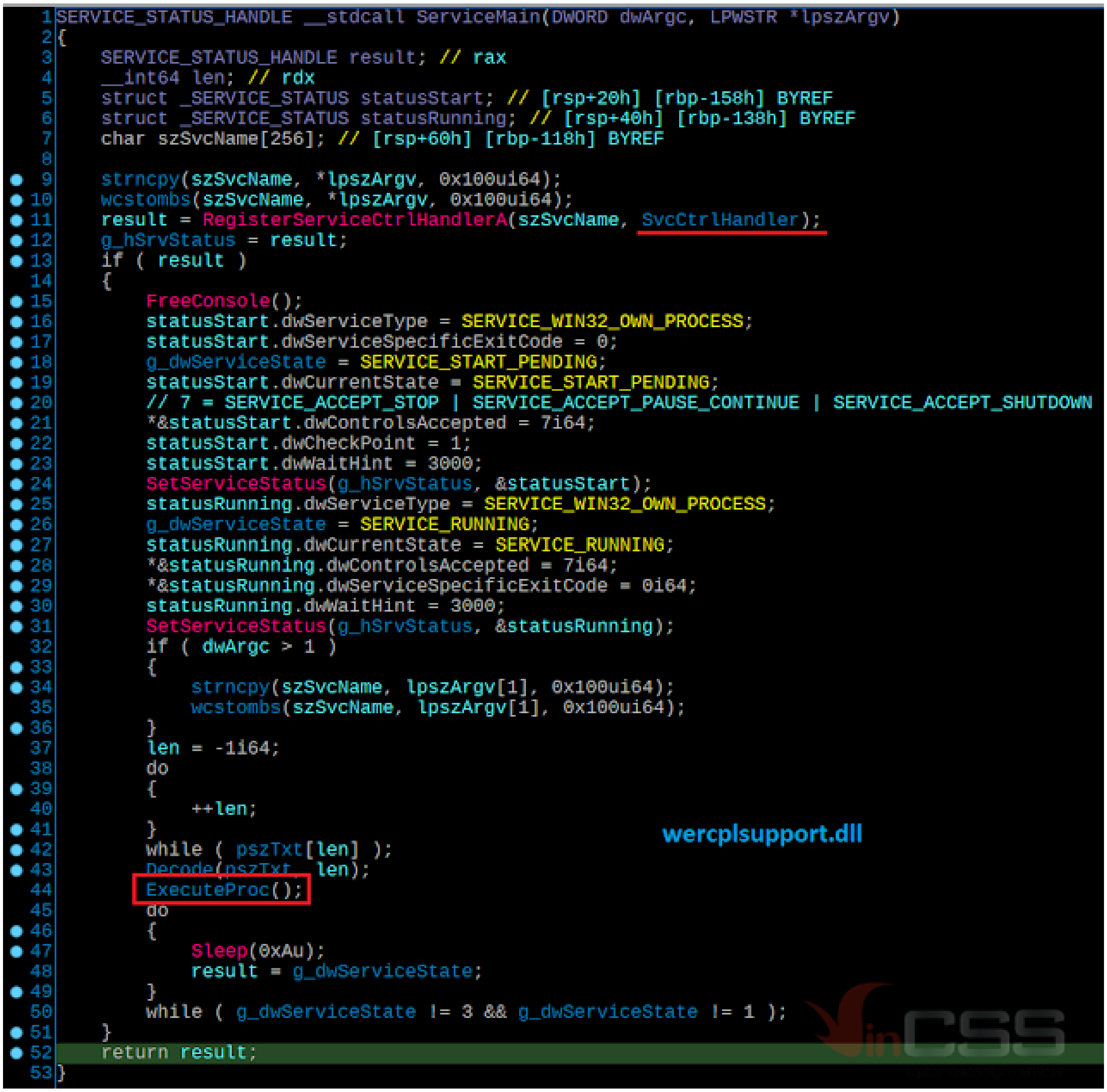
The **ServiceMain** function of **smanager_ssl.dll**:



```
SERVICE_STATUS_HANDLE __stdcall ServiceMain(DWORD dwArgc, LPWSTR *lpszArgv)
{
    SERVICE_STATUS_HANDLE result; // eax
    struct _SERVICE_STATUS statusRunning; // [esp+4h] [ebp-23Ch] BYREF
    struct _SERVICE_STATUS statusStart; // [esp+20h] [ebp-220h] BYREF
    WCHAR wszSvcName[256]; // [esp+3Ch] [ebp-204h] BYREF

    OutputDebugStringA("ServiceMain Load");
    wcsncpy(wszSvcName, *lpszArgv, 0x100u);
    result = RegisterServiceCtrlHandlerW(wszSvcName, SvcCtrlHandler);
    g_hSrvStatus = result;
    if ( !result )
    {
        return result;
    }
    FreeConsole();
    statusStart.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    g_dwServiceState = SERVICE_START_PENDING;
    statusStart.dwCurrentState = SERVICE_START_PENDING;
    // 7 = SERVICE_ACCEPT_STOP | SERVICE_ACCEPT_PAUSE_CONTINUE | SERVICE_ACCEPT_SHUTDOWN
    statusStart.dwControlsAccepted = 7;
    statusStart.dwWin32ExitCode = 0;
    statusStart.dwServiceSpecificExitCode = 0;
    statusStart.dwCheckPoint = 1;
    statusStart.dwWaitHint = 0xBB8;
    SetServiceStatus(g_hSrvStatus, &statusStart);
    statusRunning.dwServiceType = 0x10;
    g_dwServiceState = SERVICE_RUNNING;
    statusRunning.dwCurrentState = SERVICE_ACCEPT_SHUTDOWN;
    statusRunning.dwControlsAccepted = 7;
    statusRunning.dwWin32ExitCode = 0;
    statusRunning.dwServiceSpecificExitCode = 0;
    statusRunning.dwCheckPoint = 0;
    statusRunning.dwWaitHint = 0xBB8;
    SetServiceStatus(g_hSrvStatus, &statusRunning);
    Entery();
    return result;
}
```

Figure 5. ***ServiceMain*** *function of* ***smanager_ssl.dll***

wercplsupport.dll’s **ServiceMain**:

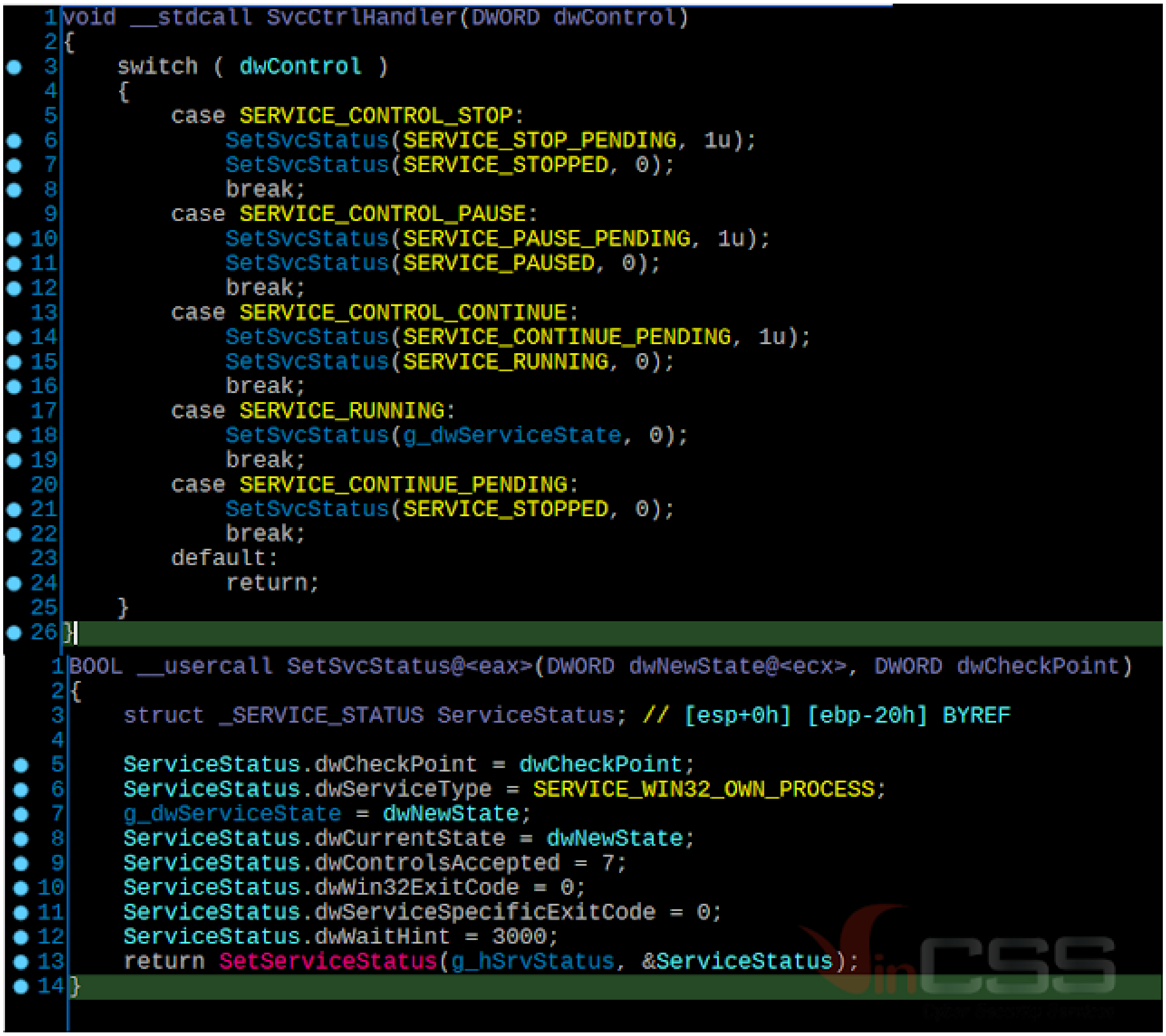


```
SERVICE_STATUS_HANDLE __stdcall ServiceMain(DWORD dwArgc, LPWSTR *lpszArgv)
{
    SERVICE_STATUS_HANDLE result; // rax
    __int64 len; // rdx
    struct _SERVICE_STATUS statusStart; // [rsp+20h] [rbp-158h] BYREF
    struct _SERVICE_STATUS statusRunning; // [rsp+40h] [rbp-138h] BYREF
    char szSvcName[256]; // [rsp+60h] [rbp-118h] BYREF

    strncpy(szSvcName, *lpszArgv, 0x100ui64);
    wcstombs(szSvcName, *lpszArgv, 0x100ui64);
    result = RegisterServiceCtrlHandlerA(szSvcName, SvcCtrlHandler);
    g_hSrvStatus = result;
    if ( result )
    {
        FreeConsole();
        statusStart.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
        statusStart.dwServiceSpecificExitCode = 0;
        g_dwServiceState = SERVICE_START_PENDING;
        statusStart.dwCurrentState = SERVICE_START_PENDING;
        // 7 = SERVICE_ACCEPT_STOP | SERVICE_ACCEPT_PAUSE_CONTINUE | SERVICE_ACCEPT_SHUTDOWN
        *&statusStart.dwControlsAccepted = 7i64;
        statusStart.dwCheckPoint = 1;
        statusStart.dwWaitHint = 3000;
        SetServiceStatus(g_hSrvStatus, &statusStart);
        statusRunning.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
        g_dwServiceState = SERVICE_RUNNING;
        statusRunning.dwCurrentState = SERVICE_RUNNING;
        *&statusRunning.dwControlsAccepted = 7i64;
        *&statusRunning.dwServiceSpecificExitCode = 0i64;
        statusRunning.dwWaitHint = 3000;
        SetServiceStatus(g_hSrvStatus, &statusRunning);
        if ( dwArgc > 1 )
        {
            strncpy(szSvcName, lpszArgv[1], 0x100ui64);
            wcstombs(szSvcName, lpszArgv[1], 0x100ui64);
        }
        len = -1i64;
        do
        {
            ++len;
        }
        while ( pszTxt[len] );
        Decode(pszTxt, len);
        ExecuteProc();
        do
        {
            Sleep(0xAu);
            result = g_dwServiceState;
        }
        while ( g_dwServiceState != 3 && g_dwServiceState != 1 );
    }
    return result;
}
```

*Figure 6. **ServiceMain** function of **wercplsupport.dll***

Not only the code is identical, there’s also another special point, a global variable that we named **g_dwServiceState** in our pseudocode. We will see this variable in the **SvcCtrlHandler** callback function.

SvcCtrlHandler function of **smanager_ssl.dll**:

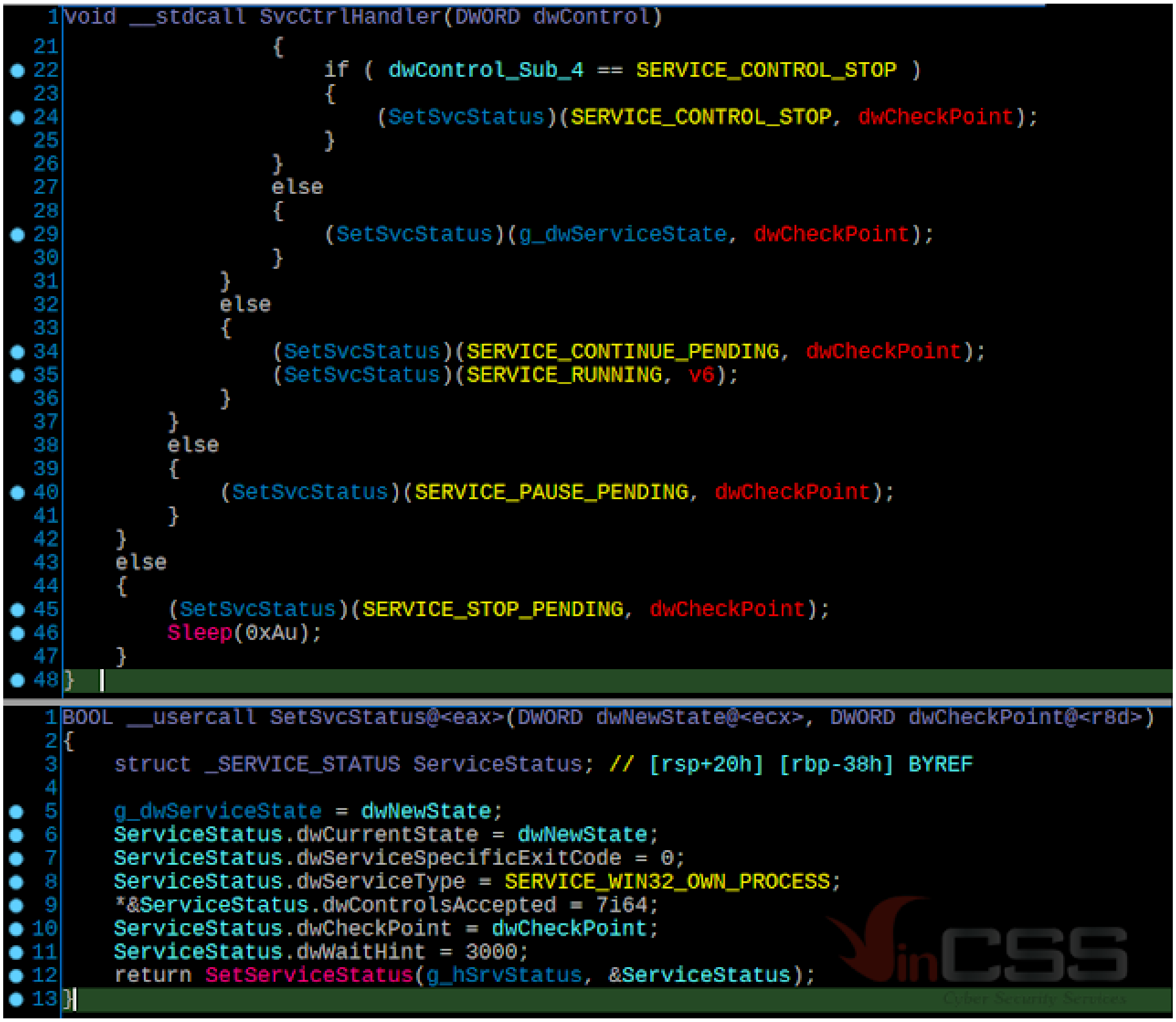


```
void __stdcall SvcCtrlHandler(DWORD dwControl)
{
    switch ( dwControl )
    {
        case SERVICE_CONTROL_STOP:
            SetSvcStatus(SERVICE_STOP_PENDING, 1u);
            SetSvcStatus(SERVICE_STOPPED, 0);
            break;
        case SERVICE_CONTROL_PAUSE:
            SetSvcStatus(SERVICE_PAUSE_PENDING, 1u);
            SetSvcStatus(SERVICE_PAUSED, 0);
            break;
        case SERVICE_CONTROL_CONTINUE:
            SetSvcStatus(SERVICE_CONTINUE_PENDING, 1u);
            SetSvcStatus(SERVICE_RUNNING, 0);
            break;
        case SERVICE_RUNNING:
            SetSvcStatus(g_dwServiceState, 0);
            break;
        case SERVICE_CONTINUE_PENDING:
            SetSvcStatus(SERVICE_STOPPED, 0);
            break;
        default:
            return;
    }
}
```

```
BOOL __usercall SetSvcStatus@<eax>(DWORD dwNewState@<ecx>, DWORD dwCheckPoint)
{
    struct _SERVICE_STATUS ServiceStatus; // [esp+0h] [ebp-20h] BYREF

    ServiceStatus.dwCheckPoint = dwCheckPoint;
    ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    g_dwServiceState = dwNewState;
    ServiceStatus.dwCurrentState = dwNewState;
    ServiceStatus.dwControlsAccepted = 7;
    ServiceStatus.dwWin32ExitCode = 0;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwWaitHint = 3000;
    return SetServiceStatus(g_hSrvStatus, &ServiceStatus);
}
```

Figure 7. ***SvcCtrlHandler*** *function of* ***smanager_ssl.dll***

wercplsupport.dll’s **SvcCtrlHandler**:



```
void __stdcall SvcCtrlHandler(DWORD dwControl)
                {
                    if ( dwControl_Sub_4 == SERVICE_CONTROL_STOP )
                    {
                        (SetSvcStatus)(SERVICE_CONTROL_STOP, dwCheckPoint);
                    }
                }
                else
                {
                    (SetSvcStatus)(g_dwServiceState, dwCheckPoint);
                }
            }
            else
            {
                (SetSvcStatus)(SERVICE_CONTINUE_PENDING, dwCheckPoint);
                (SetSvcStatus)(SERVICE_RUNNING, v6);
            }
        }
        else
        {
            (SetSvcStatus)(SERVICE_PAUSE_PENDING, dwCheckPoint);
        }
    }
    else
    {
        (SetSvcStatus)(SERVICE_STOP_PENDING, dwCheckPoint);
        Sleep(0xAu);
    }
}
```

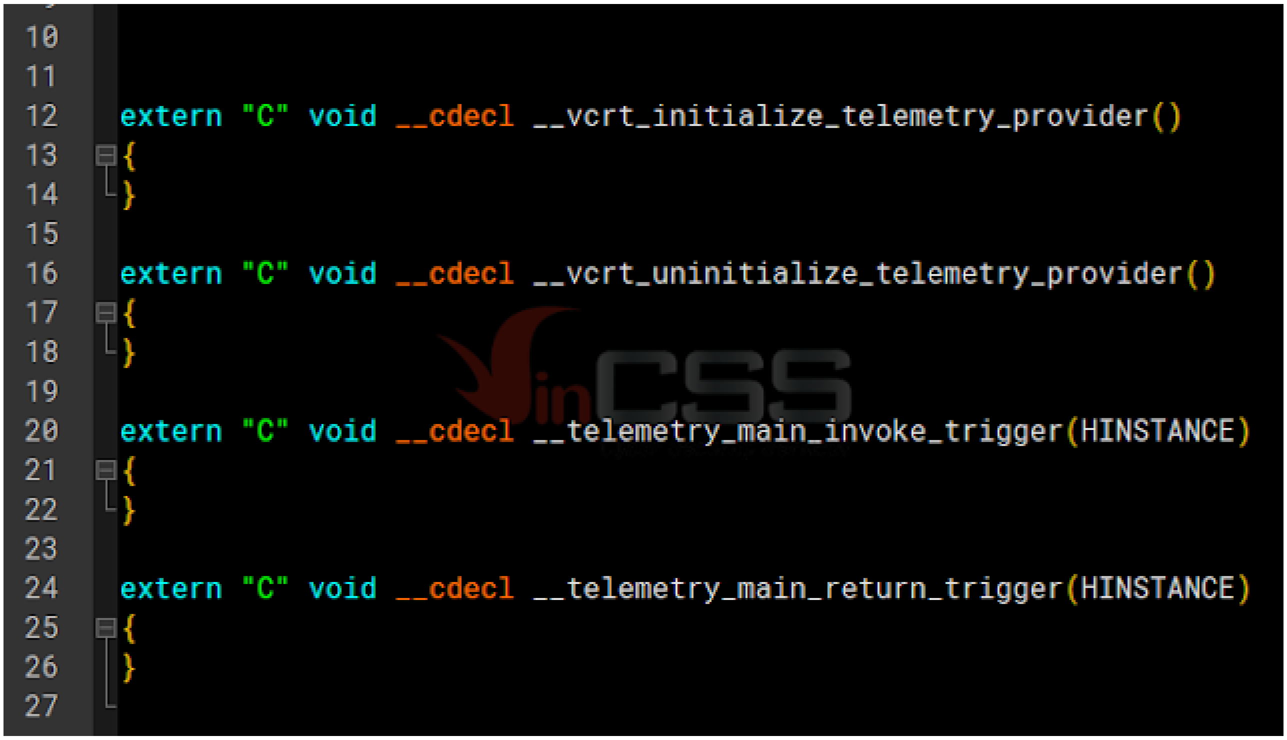
```
BOOL __usercall SetSvcStatus@<eax>(DWORD dwNewState@<ecx>, DWORD dwCheckPoint@<r8d>)
{
    struct _SERVICE_STATUS ServiceStatus; // [rsp+20h] [rbp-38h] BYREF

    g_dwServiceState = dwNewState;
    ServiceStatus.dwCurrentState = dwNewState;
    ServiceStatus.dwServiceSpecificExitCode = 0;
    ServiceStatus.dwServiceType = SERVICE_WIN32_OWN_PROCESS;
    *&ServiceStatus.dwControlsAccepted = 7i64;
    ServiceStatus.dwCheckPoint = dwCheckPoint;
    ServiceStatus.dwWaitHint = 3000;
    return SetServiceStatus(g_hSrvStatus, &ServiceStatus);
}
```

Figure 8. ***SvcCtrlHandler*** *function of* ***wercplsupport.dll***

If we conclude based on the above evidences only, it still be uncertain, as you know hackers often share malwares source code with each other. However, we have discovered one particular feature that hackers themselves may have overlooked and missed when building these malwares.

Since Visual Studio 2005, Microsoft has included **.h** and **.lib** files for Telemetry feature, and has been supporting gradually since Windows Vista. During build application, Telemetry feature will be added default in the binary. If we want to disable it, we have to link it with **notelemetry.obj**. The Microsoft's **Telemetry.cpp** file is not included in the Visual Studio 2015. You can find **notelemetry.cpp** file in the new Windows SDKs later.

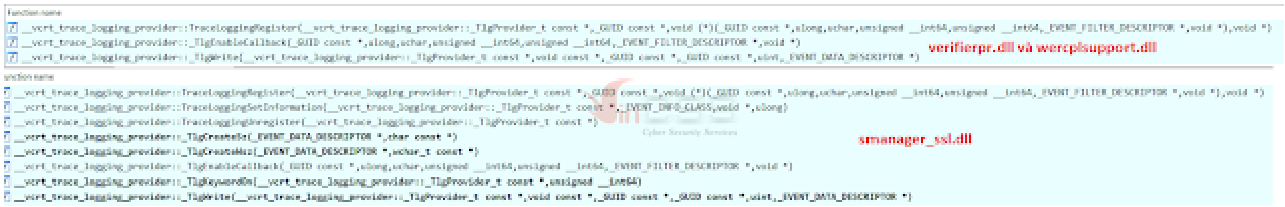
The code of **notelemetry.cpp** is to NULL sub the VC CRTL functions for Telemetry.



```cpp


extern "C" void __cdecl __vcrt_initialize_telemetry_provider()
{
}

extern "C" void __cdecl __vcrt_uninitialize_telemetry_provider()
{
}

extern "C" void __cdecl __telemetry_main_invoke_trigger(HINSTANCE)
{
}

extern "C" void __cdecl __telemetry_main_return_trigger(HINSTANCE)
{
}

```

*Figure 9. **notelemetry.cpp** to NULL sub the VC CRTL functions for Telemetry*

During the analysis, we discovered that in addition to **smanager_ssl.dll**, two samples in the above subsamples were linked to Telemetry VC CRTL: **verifierpr.dll** and **wercplsupport.dll**.



```
Function name
__vcrt_trace_logging_provider::TraceLoggingRegister(__vcrt_trace_logging_provider::_TlgProvider_t const *,_GUID const *,void (*)(_GUID const *,ulong,uchar,unsigned __int64,unsigned __int64,_EVENT_FILTER_DESCRIPTOR *,void *),void *)
__vcrt_trace_logging_provider::_TlgEnableCallback(_GUID const *,ulong,uchar,unsigned __int64,unsigned __int64,_EVENT_FILTER_DESCRIPTOR *,void *)
__vcrt_trace_logging_provider::_TlgWrite(__vcrt_trace_logging_provider::_TlgProvider_t const *,void const *,_GUID const *,_GUID const *,uint,_EVENT_DATA_DESCRIPTOR *)
```

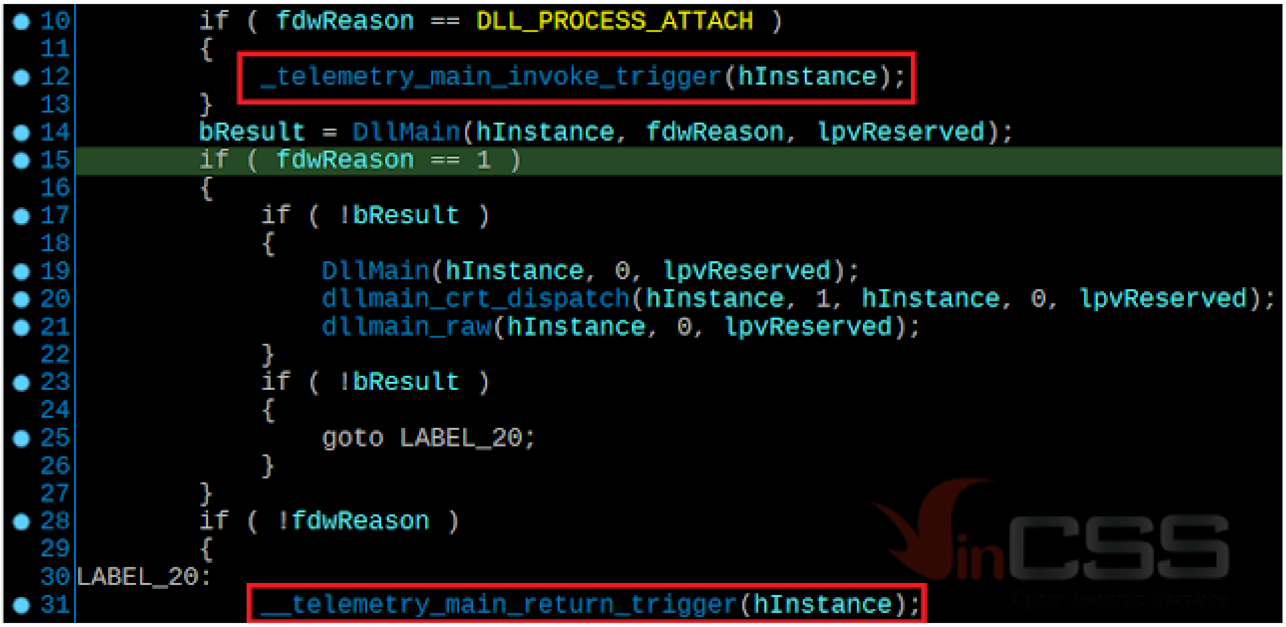
verifierpr.dll và wercplsupport.dll

```
unction name
__vcrt_trace_logging_provider::TraceLoggingRegister(__vcrt_trace_logging_provider::_TlgProvider_t const *,_GUID const *,void (*)(_GUID const *,ulong,uchar,unsigned __int64,unsigned __int64,_EVENT_FILTER_DESCRIPTOR *,void *),void *)
__vcrt_trace_logging_provider::TraceLoggingSetInformation(__vcrt_trace_logging_provider::_TlgProvider_t const *,_EVENT_INFO_CLASS,void *,ulong)
__vcrt_trace_logging_provider::TraceLoggingUnregister(__vcrt_trace_logging_provider::_TlgProvider_t const *)
__vcrt_trace_logging_provider::_TlgCreateSz(_EVENT_DATA_DESCRIPTOR *,char const *)
__vcrt_trace_logging_provider::_TlgCreateWsz(_EVENT_DATA_DESCRIPTOR *,wchar_t const *)
__vcrt_trace_logging_provider::_TlgEnableCallback(_GUID const *,ulong,uchar,unsigned __int64,unsigned __int64,_EVENT_FILTER_DESCRIPTOR *,void *)
__vcrt_trace_logging_provider::_TlgKeywordOn(__vcrt_trace_logging_provider::_TlgProvider_t const *,unsigned __int64)
__vcrt_trace_logging_provider::_TlgWrite(__vcrt_trace_logging_provider::_TlgProvider_t const *,void const *,_GUID const *,_GUID const *,uint,_EVENT_DATA_DESCRIPTOR *)
```

smanager_ssl.dll

Figure 10. Other samples linked to Telemetry VC CRTL

__telemetry_mai_invoke_trigger will be called before **DllMain** or **WinMain/main** function. And **__telemetry_main_return_trigger** will be called as soon as our above functions exit.



```
        if ( fdwReason == DLL_PROCESS_ATTACH )
        {
            _telemetry_main_invoke_trigger(hInstance);
        }
        bResult = DllMain(hInstance, fdwReason, lpvReserved);
        if ( fdwReason == 1 )
        {
            if ( !bResult )
            {
                DllMain(hInstance, 0, lpvReserved);
                dllmain_crt_dispatch(hInstance, 1, hInstance, 0, lpvReserved);
                dllmain_raw(hInstance, 0, lpvReserved);
            }
            if ( !bResult )
            {
                goto LABEL_20;
            }
        }
        if ( !fdwReason )
        {
LABEL_20:
            __telemetry_main_return_trigger(hInstance);
```

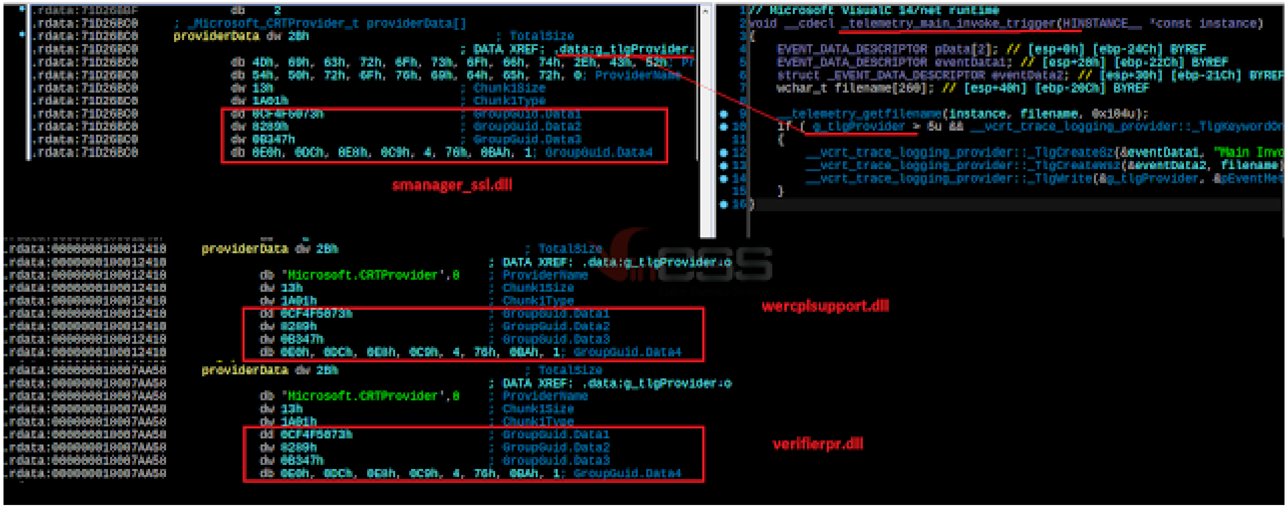
Figure 11. ***__telemetry_mai_invoke_trigger*** *will be called before* ***DllMain*** *or* ***WinMain/main*** *function*

The Telemetry API is provided by Microsoft in the **TraceLoggingProvider.h** file of the newer Windows SDK distributions. Since there is no source code of **telemetry.cpp**, we rely on the **.h** file above and reanalyze the VC CRTL functions for Telemetry. We have identified **ProviderMetaData** on **smanager_ssl.dll** file. And especially the **providerData** of both **verifierpr.dll** and **wercplsupport.dll** are the same. **GroupGuid** is a type of GUID that is generated when an attacker uses an IDE wizard or a GuidGen.exe tool or something similar. GUIDs never match.

We searched this GUID: **{CF4F5073 - 8289 - B347 - E0DC - E8C90476BA01}** on the Internet and sites as below but we couldn't find any result:

- The Magic Number Database
- GLOBAL UUID DATABASE
- ...

Through all the points we just mentioned, we conclude, the code of **smanager_ssl.dll** is built on a version of Visual Studio 2015, using a source that accidentally embedded Telemetry feature.



```
.rdata:71D26BBF                 db    2
.rdata:71D26BC0         ; _Microsoft_CRTProvider_t providerData[]
.rdata:71D26BC0         providerData dw 28h                   ; TotalSize
.rdata:71D26BC0                                         ; DATA XREF: .data:g_tlgProvider↓
.rdata:71D26BC0                 db 4Dh, 69h, 63h, 72h, 6Fh, 73h, 6Fh, 66h, 74h, 2Eh, 43h, 52h; Pr
.rdata:71D26BC0                 db 54h, 50h, 72h, 6Fh, 76h, 69h, 64h, 65h, 72h, 0; ProviderName
.rdata:71D26BC0                 dw 13h                  ; Chunk1Size
.rdata:71D26BC0                 dw 1A01h                ; Chunk1Type
.rdata:71D26BC0                 dd 0CF4F5073h           ; GroupGuid.Data1
.rdata:71D26BC0                 dw 8289h                ; GroupGuid.Data2
.rdata:71D26BC0                 dw 0B347h               ; GroupGuid.Data3
.rdata:71D26BC0                 db 0E0h, 0DCh, 0E8h, 0C9h, 4, 76h, 0BAh, 1; GroupGuid.Data4
```

```
// Microsoft VisualC 14/net runtime
void __cdecl _telemetry_main_invoke_trigger(HINSTANCE__ *const instance)
{
  EVENT_DATA_DESCRIPTOR pData[2]; // [esp+0h] [ebp-24Ch] BYREF
  EVENT_DATA_DESCRIPTOR eventData1; // [esp+20h] [ebp-22Ch] BYREF
  struct _EVENT_DATA_DESCRIPTOR eventData2; // [esp+30h] [ebp-21Ch] BYREF
  wchar_t filename[260]; // [esp+40h] [ebp-20Ch] BYREF

  __telemetry_getfilename(instance, filename, 0x104u);
  if ( g_tlgProvider > 8u && __vcrt_trace_logging_provider::_TlgKeywordOn
  {
    __vcrt_trace_logging_provider::_TlgCreateSz(&eventData1, "Main Invo
    __vcrt_trace_logging_provider::_TlgCreateWsz(&eventData2, filename)
    __vcrt_trace_logging_provider::_TlgWrite(&g_tlgProvider, &pEventMet
  }
}
```

```
.rdata:0000000180012410     providerData dw 28h                ; TotalSize
.rdata:0000000180012410                                  ; DATA XREF: .data:g_tlgProvider↓o
.rdata:0000000180012410             db 'Microsoft.CRTProvider',0   ; ProviderName
.rdata:0000000180012410             dw 13h                       ; Chunk1Size
.rdata:0000000180012410             dw 1A01h                     ; Chunk1Type
.rdata:0000000180012410             dd 0CF4F5073h                ; GroupGuid.Data1
.rdata:0000000180012410             dw 8289h                     ; GroupGuid.Data2
.rdata:0000000180012410             dw 0B347h                    ; GroupGuid.Data3
.rdata:0000000180012410             db 0E0h, 0DCh, 0E8h, 0C9h, 4, 76h, 0BAh, 1; GroupGuid.Data4
```

```
.rdata:000000018007AA58     providerData dw 28h                ; TotalSize
.rdata:000000018007AA58                                  ; DATA XREF: .data:g_tlgProvider↓o
.rdata:000000018007AA58             db 'Microsoft.CRTProvider',0   ; ProviderName
.rdata:000000018007AA58             dw 13h                       ; Chunk1Size
.rdata:000000018007AA58             dw 1A01h                     ; Chunk1Type
.rdata:000000018007AA58             dd 0CF4F5073h                ; GroupGuid.Data1
.rdata:000000018007AA58             dw 8289h                     ; GroupGuid.Data2
.rdata:000000018007AA58             dw 0B347h                    ; GroupGuid.Data3
.rdata:000000018007AA58             db 0E0h, 0DCh, 0E8h, 0C9h, 4, 76h, 0BAh, 1; GroupGuid.Data4
```

*Figure 12. **smanager_ssl.dll** is built on a version of Visual Studio 2015 and embedded Telemetry feature*

To learn more about Telemetry of VS 2015 and Windows, you can read the following links (1, 2). With the GUIDs of the **eToken.exe** and the **providerData** GUID of the three dll above, we could write Yara rules as follows:

1. **eToken.exe** (**VVSup.exe**):

- **GUID_1 = { 5AD5B72A - 853B - 456E – AF92 – 0F4DFF9D8BAF }**

Hex string = “2A B7 D5 5A 3B 85 6E 45 AF 92 0F 4D FF 9D 8B AF”

- **GUID_2 = { 798E265A - CC96 – 4623 - BA97023B575502B8 }**

Hex string = “5A 26 8E 79 96 CC 23 46 BA 97 02 3B 57 55 02 B8”

- **GUID_1 and GUID_2**

2. **Smanager_ssl.dll**:

- **Text = “Microsoft.CRTProvider”**

- **GUID = { CF4F5073 – 8289 - B347 – E0DC – E8C90476BA01 }**

Hex string = “73 50 4F CF 89 82 47 B3 E0 DC E8 C9 04 76 BA 01”

- **Text and GUID**

Combining all the indicators and TTPs we've got, we considered this was the another campaign of the Chinese Panda group aimed at agencies, organizations and businesses in Vietnam over past few years.

In the next part, we will describe in detail the C++ code of the **smanager_ssl.dll** that we analyzed and recreated.

Merry Christmas & Happy New Year!

(To be continued …)

Click here for Vietnamese version: Part 3

Trương Quốc Ngân (aka HTC)

Malware Analysis Expert - VinCSS (a member of Vingroup)