KCon

Where’s My Session Pool

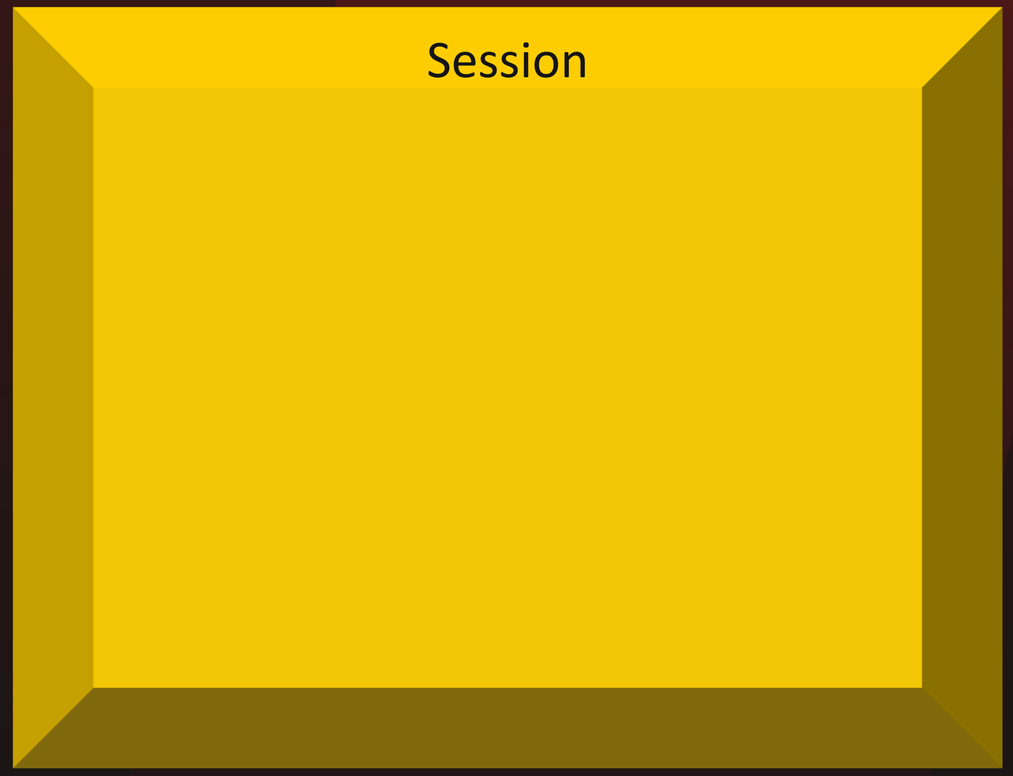
绿盟科技 天机实验室 张云海

+1 进阶，
护航未来

KCon

+1

Session Pool 是什么



Session

代表用户的登陆会话

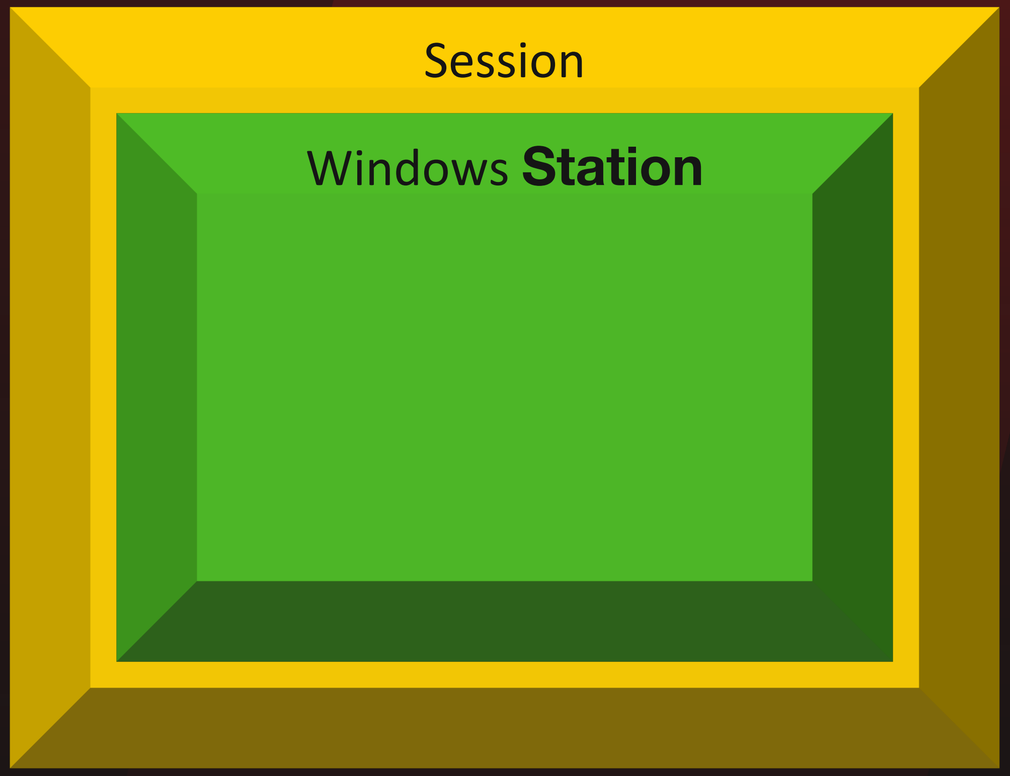
由该用户的所有进程与系统对象所构成

其中包含多个 Windows Station

KCon

+1

Session Pool 是什么



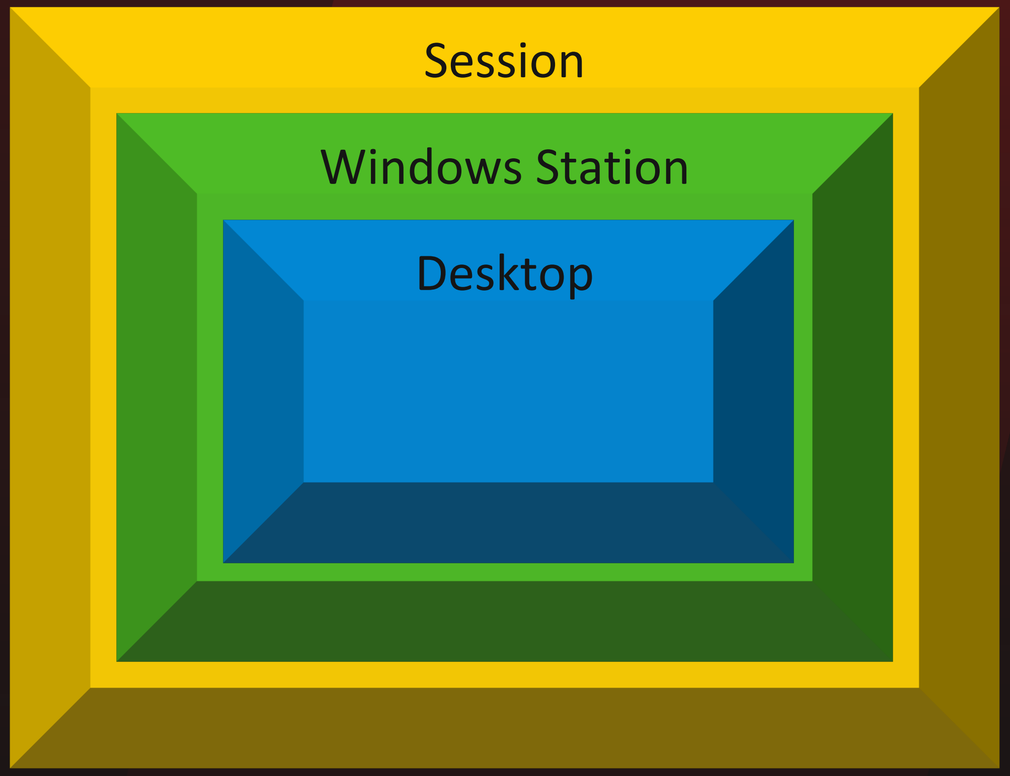
Windows Station

用于保护登录用户的一个安全边界

只有一个 Windows Station 可以与用户进行交互

其中包含多个 Desktop

Session Pool 是什么



Desktop

用于加载逻辑显示表面的对象

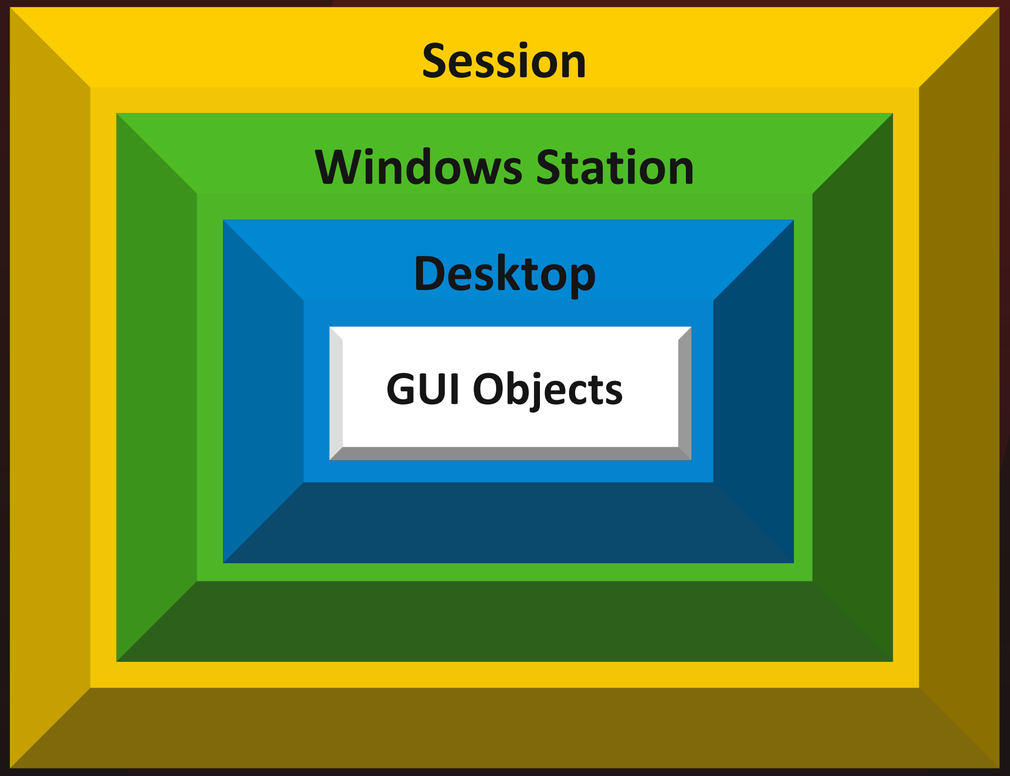
Winsta0 包含4个 Desktop：

- Winlogon
- Default
- Secure
- Disconnect

KCon

+1

Session Pool 是什么



Session Space

Session 专属的私有内核内存空间
用于分配 Desktop 内的 GUI 对象

同一 Session 中的所有进程共享 Session Space
不同 Session 中的进程的 Session Space 相互隔离

Session Space 分为4个区域：

- Session Structure
- Session Image Space
- Session View Space
- Session Pool

KCon

+1

Session Pool 的历史

Windows 2000

引入 Session Pool
基于 _POOL_DESCRIPTOR 的内存管理

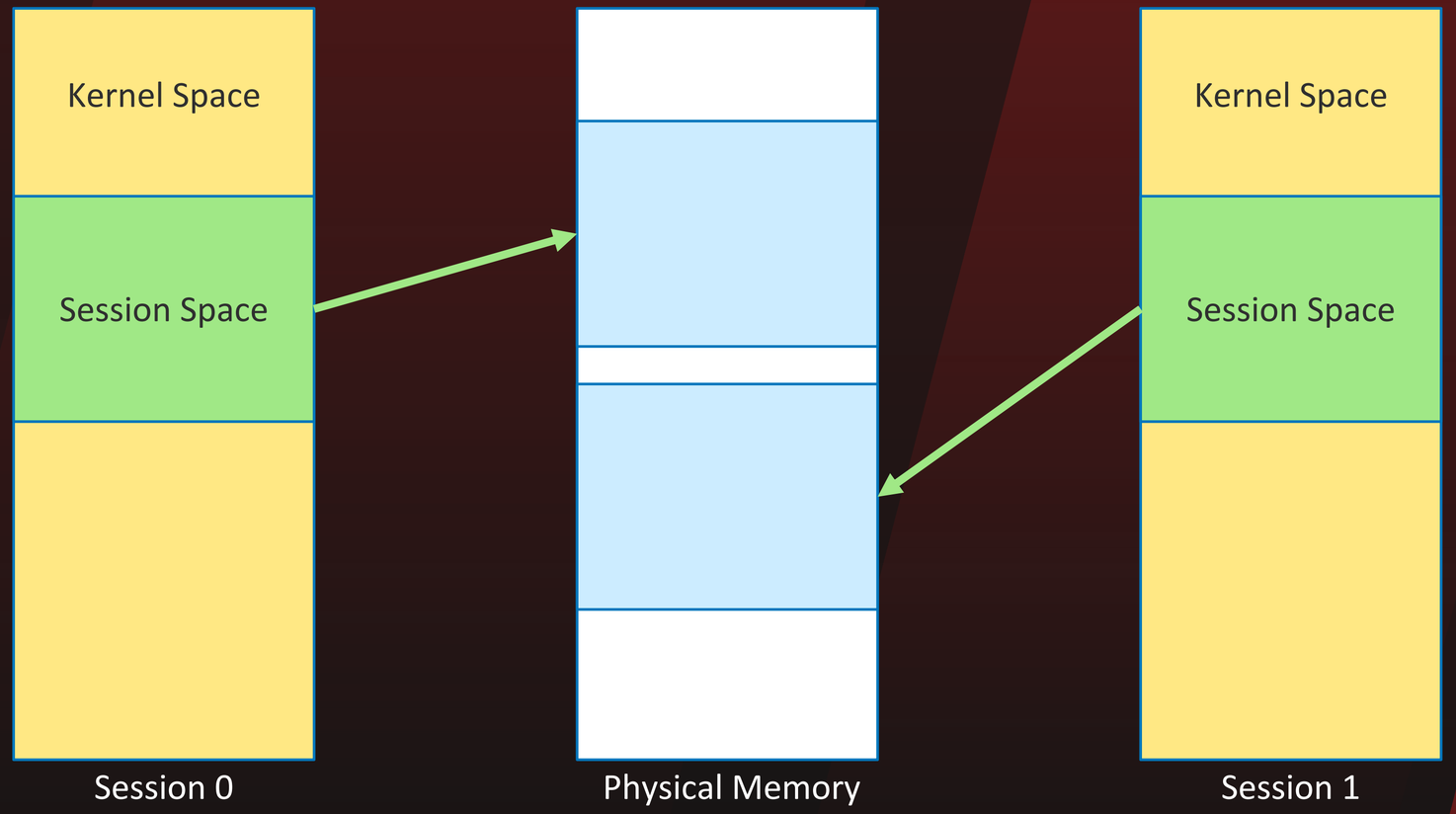
Windows 10 19H1

基于 Heap 的内存管理

Windows 11 22H2

移除 session Pool

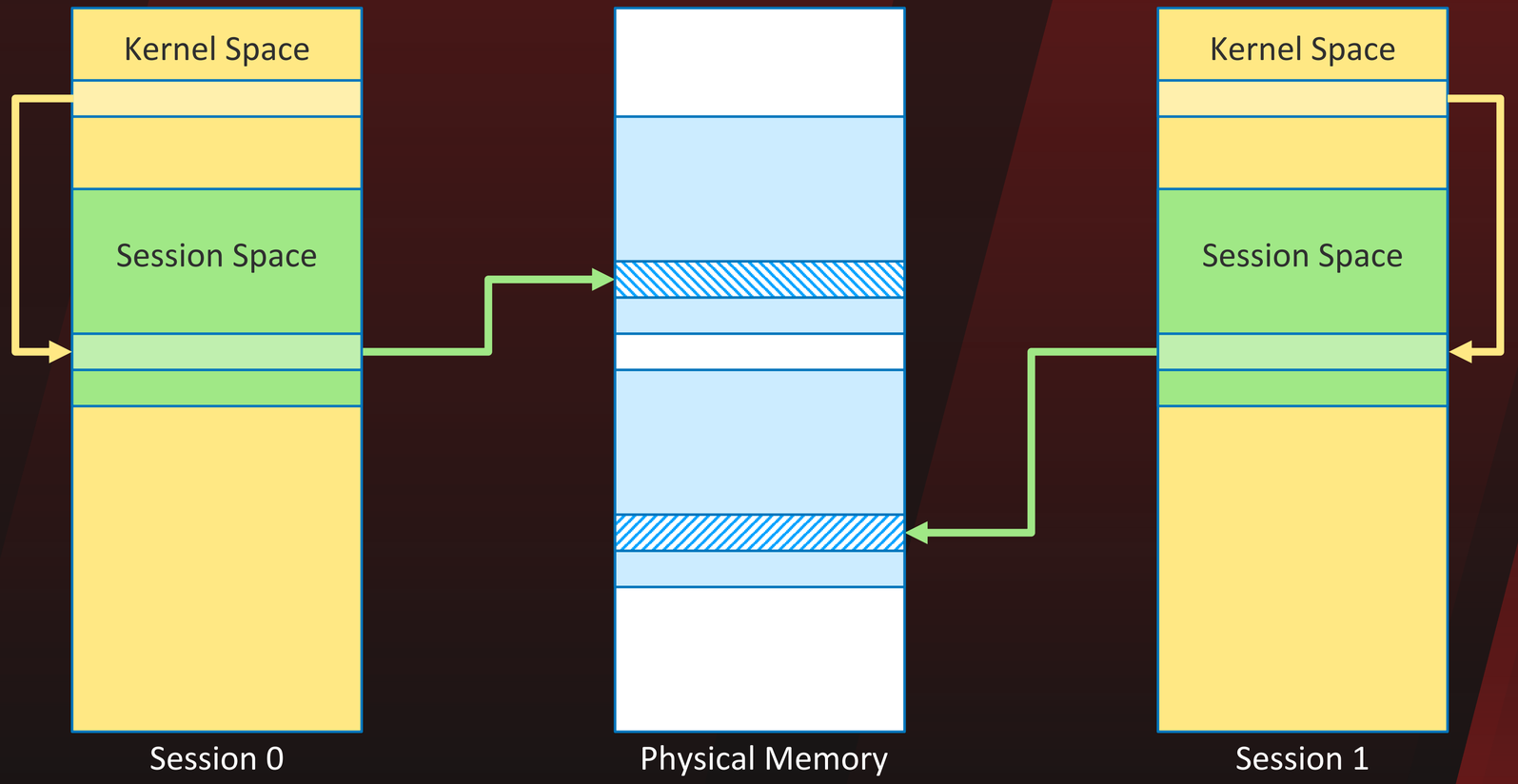
Session Pool 导致的问题



KCon

+1

Session Pool 导致的问题



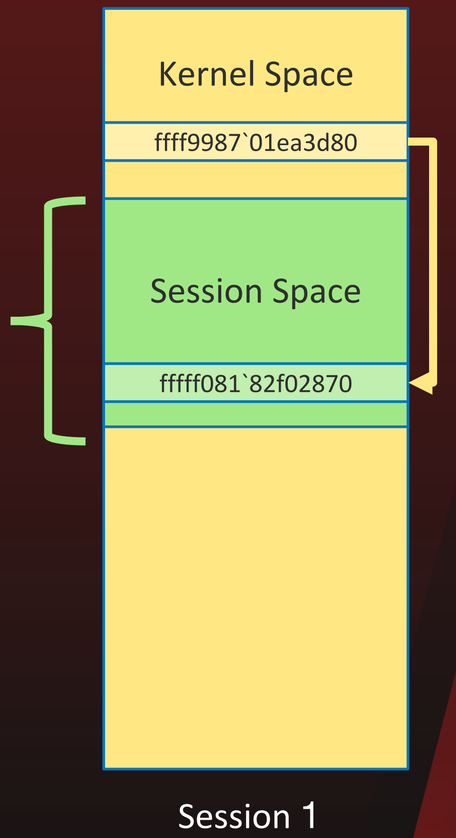
Session Pool 导致的问题

Kernel Space 中是否存在指向 Session Space的指针？

```
1: kd> dt nt!_KTHREAD Win32Thread
   +0x1c8 Win32Thread : Ptr64 Void
```

```
1: kd> dq poi(@$thread + 1c8) l1
ffff9987`01ea3d80  fffff081`82f02870
```

```
1: kd> dt -r nt!_EPROCESS Session Session.PagedPoolStart Session.PagedPoolEnd @$proc
   +0x400 Session                  : 0xffff8380`bcf00000 _MM_SESSION_SPACE
      +0x038 PagedPoolStart        : 0xfffff081`80000000 Void
      +0x040 PagedPoolEnd          : 0xfffff0a1`7fffffff Void
```



Session Pool 导致的问题

是否存在跨会话的指针解引用？

```
nt!PsGetThreadWin32Thread:
fffff805`26b0cf40 488b81c8010000  mov     rax,qword ptr [rcx+1C8h]
fffff805`26b0cf47 c3              ret
```

```
win32kbase!HmgAllocateDcAttr:
fffff0c0`3143fc74 48895c2410      mov     qword ptr [rsp+10h],rbx
fffff0c0`3143fc79 48896c2418      mov     qword ptr [rsp+18h],rbp
fffff0c0`3143fc7e 4889742420      mov     qword ptr [rsp+20h],rsi
fffff0c0`3143fc83 57              push    rdi
fffff0c0`3143fc84 4883ec20        sub     rsp,20h
fffff0c0`3143fc88 65488b0c2588010000 mov   rcx,qword ptr gs:[188h]
fffff0c0`3143fc91 33db            xor     ebx,ebx
fffff0c0`3143fc93 8bfb            mov     edi,ebx
fffff0c0`3143fc95 48ff159c3d2200  call    qword ptr [win32kbase!_imp_PsGetThreadWin32Thread (fffff0c0`31663a38)]
```

KCon

+1

Session Pool 导致的问题

是否存在跨会话的指针解引用？

```
0: kd> !pcr
KPCR for Processor 0 at fffff805252b5000:
    Major 1 Minor 1
      NtTib.ExceptionList: fffff8052ce8dfb0
          NtTib.StackBase: fffff8052ce8c000
         NtTib.StackLimit: 0000000000000000
       NtTib.SubSystemTib: fffff805252b5000
            NtTib.Version: 00000000252b5180
        NtTib.UserPointer: fffff805252b5870
            NtTib.SelfTib: 000000dae3292000

                  SelfPcr: 0000000000000000
                     Prcb: fffff805252b5180
                     Irql: 0000000000000000
                      IRR: 0000000000000000
                      IDR: 0000000000000000
            InterruptMode: 0000000000000000
                      IDT: 0000000000000000
                      GDT: 0000000000000000
                      TSS: 0000000000000000

            CurrentThread: ffff9986f7568080
               NextThread: 0000000000000000
               IdleThread: fffff80526f8c400

                DpcQueue: Unable to read nt!_KDPC_DATA.DpcListHead.Flink @ fffff805252b7f80
0: kd> dq fffff805252b5000 + 188 l1
fffff805`252b5188  ffff9986`f7568080
```

Session Pool 导致的问题

是否存在跨会话的指针解引用？

KeStackAttachProcess function (ntifs.h)

Article • 10/22/2021 • 2 minutes to read

The **KeStackAttachProcess** routine attaches the current thread to the address space of the target process.

Caution Use this routine with extreme caution. (See the note in the Remarks section.)

Syntax

C++ Copy

```
void KeStackAttachProcess(
        PRKPROCESS    PROCESS,
  [out] PRKAPC_STATE ApcState
);
```

Session Pool 导致的问题

存在问题的模式

KCon

+1

CVE-2019-0892

NtTerminateProcess 在关闭句柄时调用 KiStackAttachProcess

```
 # Child-SP          RetAddr           Call Site
00 fffff08d`7d967828 fffff802`37d334cc nt!KiStackAttachProcess
01 fffff08d`7d967830 fffff802`37d055e0 nt!ExSweepHandleTable+0x13f0ec
02 fffff08d`7d9678e0 fffff802`37b1fa9b nt!PspRundownSingleProcess+0x19069c
03 fffff08d`7d967960 fffff802`37bcb7c8 nt!PspTerminateAllThreads+0x21f
04 fffff08d`7d9679d0 fffff802`37bcb599 nt!PspTerminateProcess+0xe0
05 fffff08d`7d967a10 fffff802`377c4085 nt!NtTerminateProcess+0xa9
06 fffff08d`7d967a80 00007fff`7f26eb14 nt!KiSystemServiceCopyEnd+0x25
07 0000008d`4ab2f7c8 00007fff`7b73cec0 ntdll!NtTerminateProcess+0x14
```

KCon

+1

CVE-2019-0892

DxgkCompositionObject 对象析构时调用 RGNMEMOBJ::vPushThreadGuardedObject

```
 # Child-SP          RetAddr           Call Site
00 ffff820d`4b13f3c8 ffff8e12`683dbf1a win32kbase!RGNMEMOBJ::vPushThreadGuardedObject
01 ffff820d`4b13f3d0 ffff8e12`683dbe59 win32kbase!CRegion::InternalCombine+0xb6
02 ffff820d`4b13f430 fffff803`19206dd2 win32kbase!CRegion::Combine+0x9
03 ffff820d`4b13f460 fffff803`1922a235 dxgkrnl!CCompositionToken::UpdateDirtyRegions+0x11a
04 ffff820d`4b13f4b0 fffff803`192070fb dxgkrnl!CCompositionToken::Discard+0x13ab5
05 ffff820d`4b13f4e0 fffff803`19217469 dxgkrnl!CCompositionToken::MarkInvalid+0x3b
06 ffff820d`4b13f510 fffff803`19216fd3 dxgkrnl!CCompositionToken::Delete+0x29
07 ffff820d`4b13f540 fffff803`18a4a5f0 dxgkrnl!DxgkCompositionObject::Delete+0x73
```

KCon

+1

CVE-2019-0892

RGNMEMOBJ::vPushThreadGuardedObject 调用 PsGetThreadWin32Thread

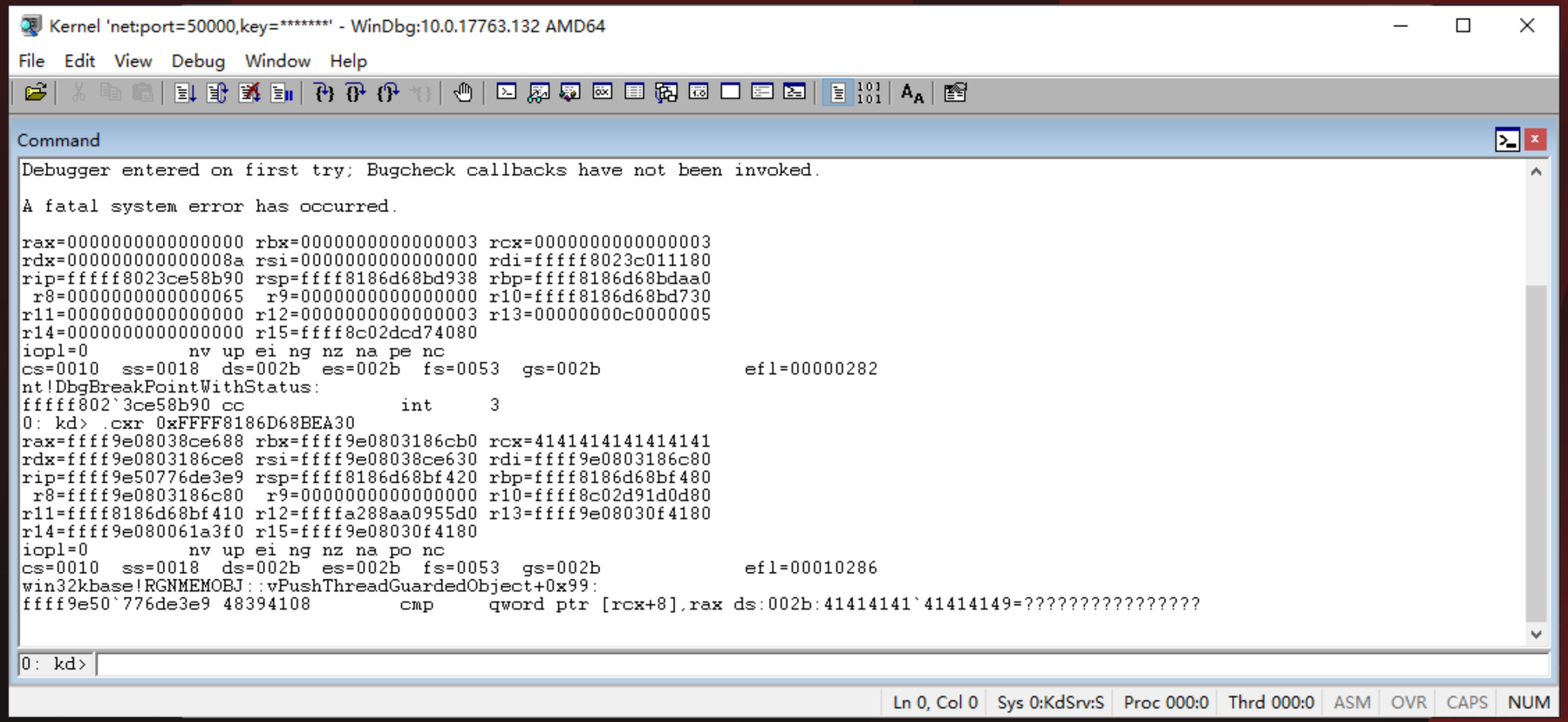
```
void __fastcall RGNMEMOBJ::vPushThreadGuardedObject(RGNMEMOBJ *this)
{
  __int64 *Win32Thread; // rax MAPDST
  _QWORD *v3; // rdi
  _QWORD *v4; // rbx
  _QWORD *v5; // rsi
  __int64 v7; // rcx
  _QWORD *v8; // rax

  Win32Thread = PsGetThreadWin32Thread(__readgsqword(0x188u));
  if ( Win32Thread )
  {
    if ( *Win32Thread )
    {
      v3 = *this;
      if ( v3 )
      {
        v4 = v3 + 6;
        if ( v3 != 0xFFFFFFFFFFFFFFD0i64 )
        {
          KeEnterCriticalRegion();
          v5 = 0i64;
          Win32Thread = PsGetThreadWin32Thread(__readgsqword(0x188u));
          if ( Win32Thread )
            v5 = *Win32Thread;
```

KCon

+1

CVE-2019-0892



KCon

+1

微软的修复方案

引入函数 IsThreadCrossSessionAttached 进行检查

```
_BOOL8 IsThreadCrossSessionAttached()
{
  PEPROCESS CurrentProcess; // rax
  int Id; // ebx
  PEPROCESS CurrentThreadProcess; // rax
  _BOOL8 result; // rax

  result = 0;
  if ( KeIsAttachedProcess() )
  {
    CurrentProcess = PsGetCurrentProcess();
    Id = PsGetProcessSessionIdEx(CurrentProcess);
    CurrentThreadProcess = PsGetCurrentThreadProcess();
    if ( Id != PsGetProcessSessionIdEx(CurrentThreadProcess) )
      result = 1;
  }
  return result;
}
```

```
_EPROCESS *PsGetCurrentProcess()
{
  return KeGetCurrentThread()->ApcState.Process;
}
```

```
_EPROCESS *PsGetCurrentThreadProcess()
{
  return KeGetCurrentThread()->Process;
}
```

KCon

+1

微软的修复方案

需要使用方主动调用进行检查

NtGdiDeleteObjectApp

bDeleteDCOBJ

GreGetDeviceCaps

ReleaseCacheDC

_GetDCEx

GreGetBounds

XDCOBJ::bCleanDC

HmgDecrementShareReferenceCountEx

HmgLockEx

HANDLELOCK::vLockHandle

ResetOrg

HANDLELOCK::bLockHobj

hbmSelectBitmap

DCMEMOBJ::DCMEMOBJ

W32GetThreadWin32Thread

SURFMEM::bCreateDIB

XDCOBJ::bDeleteDC

GreIntersectClipRect

KCon

+1

微软的修复方案

提供了 PsGetThreadWin32Thread 的封装 W32GetThreadWin32Thread

```
__int64 __fastcall W32GetThreadWin32Thread(PETHREAD Thread)
{
  __int64 v2; // rbx
  __int64 *Win32Thread; // rax

  v2 = 0i64;
  if ( !IsThreadCrossSessionAttached() )
  {
    Win32Thread = PsGetThreadWin32Thread(Thread);
    if ( Win32Thread )
      v2 = *Win32Thread;
  }
  return v2;
}
```

KCon

+1

Windows 11 22H2 的变化

_MM_SESSION_SPACE 中移除了 Session Pool 相关成员

```
0: kd> dt nt!_MM_SESSION_SPACE
   +0x000 ReferenceCount    : Int4B
   +0x004 u                 : <unnamed-tag>
   +0x008 SessionId         : Uint4B
   +0x00c ProcessReferenceToSession : Int4B
   +0x010 ProcessList       : _LIST_ENTRY
   +0x020 NonPagablePages   : Uint8B
   +0x028 CommittedPages    : Uint8B
   +0x030 PagedPoolStart    : Ptr64 Void
   +0x038 PagedPoolEnd      : Ptr64 Void
   +0x040 SessionObject     : Ptr64 Void
   +0x048 SessionObjectHandle : Ptr64 Void
   +0x050 ImageTree         : _RTL_AVL_TREE
   +0x058 LocaleId          : Uint4B
   +0x05c AttachCount       : Uint4B
   +0x060 AttachGate        : _KGATE
   +0x078 WsListEntry       : _LIST_ENTRY
   +0x088 WsTreeEntry       : _RTL_BALANCED_NODE
   +0x0a0 PagedPoolInfo     : _MM_PAGED_POOL_INFO
   +0x0b8 CombineDomain     : Uint8B
   +0x0c0 Vm                : _MMSUPPORT_FULL
   +0x200 WorkingSetList    : _MMWSL_INSTANCE
   +0x240 AggregateSessionWs : _MMSUPPORT_AGGREGATION
   +0x260 HeapState         : Ptr64 Void
   +0x268 DriverUnload      : _MI_SESSION_DRIVER_UNLOAD
   +0x270 TopLevelPteLockBits : [32] Uint4B
   +0x2f0 SessionVaLock     : _EX_PUSH_LOCK
   +0x2f8 DynamicVaBitMap   : _RTL_BITMAP_EX
   +0x308 DynamicVaHint     : Uint8B
   +0x310 PageTables        : [1] _MMPTE
   +0x318 PoolBigEntriesInUse : Int4B
   +0x31c PagedPoolPdeCount : Int4B
   +0x320 DynamicSessionPdeCount : Uint4B
   +0x328 PoolTrackTableExpansion : Ptr64 Void
   +0x330 PoolTrackTableExpansionSize : Uint8B
   +0x338 PoolTrackBigPages : Ptr64 Void
   +0x340 PoolTrackBigPagesSize : Uint8B
   +0x348 PermittedFaultsTree : _RTL_AVL_TREE
   +0x350 IoState           : _IO_SESSION_STATE
   +0x354 IoStateSequence   : Uint4B
   +0x358 IoNotificationEvent : _KEVENT
   +0x370 ServerSilo        : Ptr64 _EJOB
   +0x378 CreateTime        : Uint8B
   +0x380 PoolTags          : Ptr64 Void
```

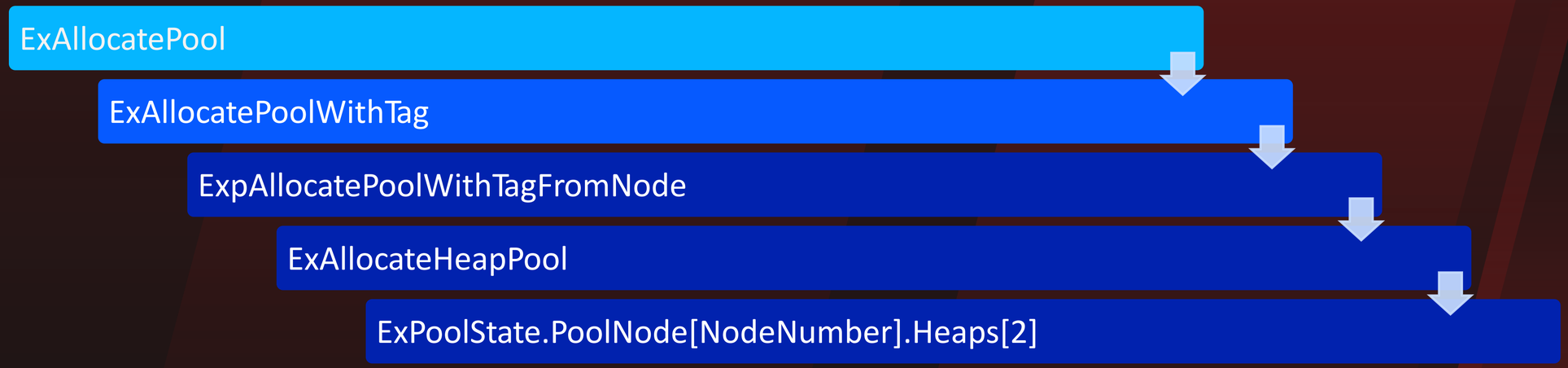
```
0: kd> dt nt!_MM_SESSION_SPACE
   +0x000 ReferenceCount    : Int4B
   +0x004 u                 : <unnamed-tag>
   +0x008 SessionId         : Uint4B
   +0x00c ProcessReferenceToSession : Int4B
   +0x010 ProcessList       : _LIST_ENTRY
   +0x020 NonPagablePages   : Uint8B
   +0x028 CommittedPages    : Uint8B
   +0x030 SessionObject     : Ptr64 Void
   +0x038 SessionObjectHandle : Ptr64 Void
   +0x040 ImageTree         : _RTL_AVL_TREE
   +0x048 LocaleId          : Uint4B
   +0x04c AttachCount       : Uint4B
   +0x050 AttachGate        : _KGATE
   +0x068 AttachersUsingPxeCopies : [2] Ptr64 _EPROCESS
   +0x078 WsListEntry       : _LIST_ENTRY
   +0x088 WsTreeEntry       : _RTL_BALANCED_NODE
   +0x0a0 CombineDomain     : Uint8B
   +0x0c0 Vm                : _MMSUPPORT_FULL
   +0x200 WorkingSetList    : _MMWSL_INSTANCE
   +0x240 AggregateSessionWs : _MMSUPPORT_AGGREGATION
   +0x260 DriverUnload      : _MI_SESSION_DRIVER_UNLOAD
   +0x268 TopLevelPteLockBits : [32] Uint4B
   +0x2e8 PageTables        : [1] _MMPTE
   +0x2f0 IoState           : _IO_SESSION_STATE
   +0x2f4 IoStateSequence   : Uint4B
   +0x2f8 IoNotificationEvent : _KEVENT
   +0x310 ServerSilo        : Ptr64 _EJOB
   +0x318 CreateTime        : Uint8B
```

KCon

+1

Windows 11 22H2 的变化

ExAllocatePool 忽略 SESSION_POOL_MASK



KCon

+1

Windows 11 22H2 的变化

ExFreePool 忽略 MiVaSessionSpace

KCon

+1

移除 Session Pool 的影响

Session Pool 实际上在 Paged Pool 中分配

```
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD49410
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD49C10
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A010
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4B110
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4BE10
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4B210
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A210
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A910
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A610
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4C010
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4AA10
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A110
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A410
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A710
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4A510
ExAllocatePool(0x21, 0xF0) = 0xFFFFE3830FD4AC10
```

```
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD48410
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B610
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B510
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B710
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B810
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4AF10
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B310
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B410
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4B910
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4BB10
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4C610
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4CD10
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4DC10
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4CB10
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4D210
ExAllocatePool(0x1, 0xF0) = 0xFFFFE3830FD4DB10
```

KCon

+1

移除 Session Pool 的影响

解决了内核的内存不一致性问题

Win32Thread 对象实际上在 Paged Pool 中分配

- 任意进程中所有的 Win32Thread 对象都位于有效内存中

```
1: kd> !process ffffb8861ac8e140 4
PROCESS ffffb8861ac8e140
    SessionId: 1  Cid: 0294    Peb: 8b620af000  ParentCid: 0278
    DirBase: 10fce0002  ObjectTable: ffffe38302102e40  HandleCount: 373.
    Image: csrss.exe

        THREAD ffffb8861a10d080  Cid 0294.02a4  Teb: 0000008b620b6000 Win32Thread: ffffb8861acd5a00 WAIT
        THREAD ffffb8861acb60c0  Cid 0294.02b4  Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT
        THREAD ffffb8861ace3080  Cid 0294.02d4  Teb: 0000008b620ba000 Win32Thread: ffffb8861ae5f700 WAIT
        THREAD ffffb8861ace5080  Cid 0294.02dc  Teb: 0000008b620be000 Win32Thread: ffffb8861b327140 WAIT
        THREAD ffffb8861ace6080  Cid 0294.02e0  Teb: 0000008b620c0000 Win32Thread: ffffb8861c4be670 WAIT
        THREAD ffffb8861ace7080  Cid 0294.02e4  Teb: 0000008b620c2000 Win32Thread: 0000000000000000 WAIT
        THREAD ffffb8861abb7340  Cid 0294.02fc  Teb: 0000008b620c4000 Win32Thread: ffffb8861ad948a0 WAIT
        THREAD ffffb8861a0db080  Cid 0294.0304  Teb: 0000008b620c6000 Win32Thread: ffffb8861a0d7d80 WAIT
        THREAD ffffb8861acec080  Cid 0294.0308  Teb: 0000008b620c8000 Win32Thread: ffffb8861acfa060 WAIT
        THREAD ffffb8861ae59080  Cid 0294.01e0  Teb: 0000008b620ca000 Win32Thread: 0000000000000000 WAIT
        THREAD ffffb8861af40080  Cid 0294.0198  Teb: 0000008b620cc000 Win32Thread: ffffb8861af2f340 WAIT
        THREAD ffffb8861af41080  Cid 0294.03e8  Teb: 0000000000000000 Win32Thread: 0000000000000000 WAIT
        THREAD ffffb8861c96f080  Cid 0294.1ca8  Teb: 0000008b620d4000 Win32Thread: ffffb8861c9ef590 WAIT
        THREAD ffffb8861b79a080  Cid 0294.1d50  Teb: 0000008b620e2000 Win32Thread: ffffb8861c732bd0 WAIT
        THREAD ffffb8861b7cc040  Cid 0294.0478  Teb: 0000008b620ee000 Win32Thread: ffffb8861cd3df30 WAIT
```

```
1: kd> !process -1 0
PROCESS ffffb8861a111080
    SessionId: 0  Cid: 0238    Peb: 6ada780000  ParentCid: 021c
    DirBase: 10e61c002  ObjectTable: ffffe38301eabc00  HandleCount: 552
    Image: csrss.exe

1: kd> dq poi(ffffb8861acd5a00)
ffffe383`043f2620  ffffb886`1a10d080 00000000`00000001
ffffe383`043f2630  00000000`00000000 00000239`1fdd09a0
ffffe383`043f2640  00000000`00000000 ffffe383`043f2648
ffffe383`043f2650  ffffe383`043f2648 00000000`00000000
ffffe383`043f2660  00000000`00000000 00000000`00000000
ffffe383`043f2670  ffffb886`1aaf8260 ffffe383`043f2678
ffffe383`043f2680  ffffe383`043f2678 00000000`00000000
ffffe383`043f2690  00000000`00000000 00000000`00000000
```

KCon

+1

移除 Session Pool 的影响

增加了内存布局的不确定性

GUI 对象实际上在 Paged Pool 中分配

- 特定 GUI 对象的地址不再可预测
- GUI 对象之间的相对位置更加随机
- 内存占位时的条件竞争更加复杂

KCon

+1

移除 Session Pool 的影响

在 Paged Pool 中利用 Bitmap

完全可控的 Paged Pool 内存分配

- 大小可控
- 数据可控

可以在用户态随时读写分配的内存

- GetBitmapBits
- SetBitmapBits

KCon

+1

总结

Session Pool 导致了内核空间的内存不一致性

Windows 11 22H2 开始移除了 Session Pool

用 SESSION_POOL 分配的内存实际上在 Paged Pool 中分配

KCon

+1

感谢您的观看

THANK YOU FOR YOUR WATCHING

KCon 2022 黑客大会