XCON 2020

安全焦点信息安全技术峰会

XFOCUS INFORMATION SECURITY CONFERENCE

AUGUST 2020

CHINA · BEIJING

目 录

CONTENTS

01 背景

Windows 堆管理的演进

Windows XP: NT Heap

Windows Vista: Low Fragmentation Heap

Windows 10: Segment Heap

基 于 堆 的 内 核 池 管 理

- Windows 10 Version 1803: Disabled By Default
- Windows 10 Version 1809: Enabled By Default
- Windows 10 Version 1903: Replace Original Pool

相关研究

Mark Vincent Yason

- Windows 10 Segment Heap Internals

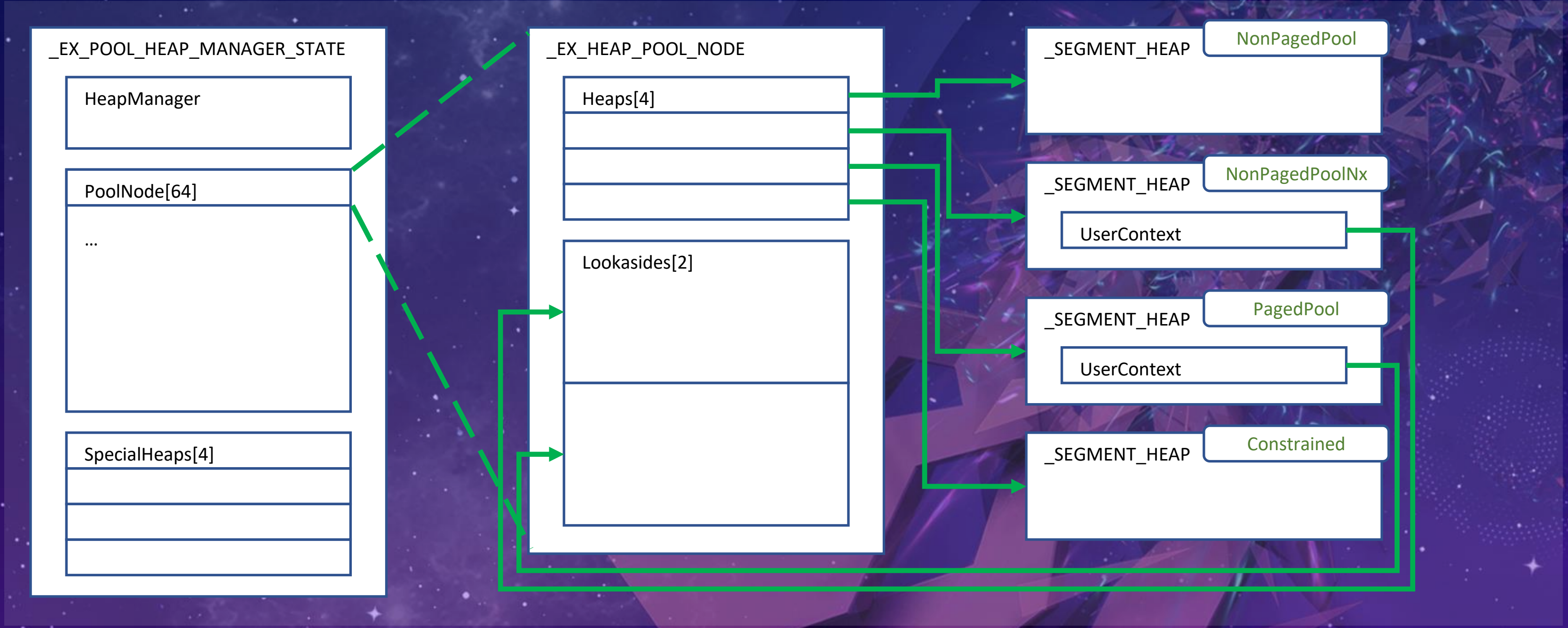
Paul Fariello & Corentin Bayet

- Scoop the Windows 10 Pool!

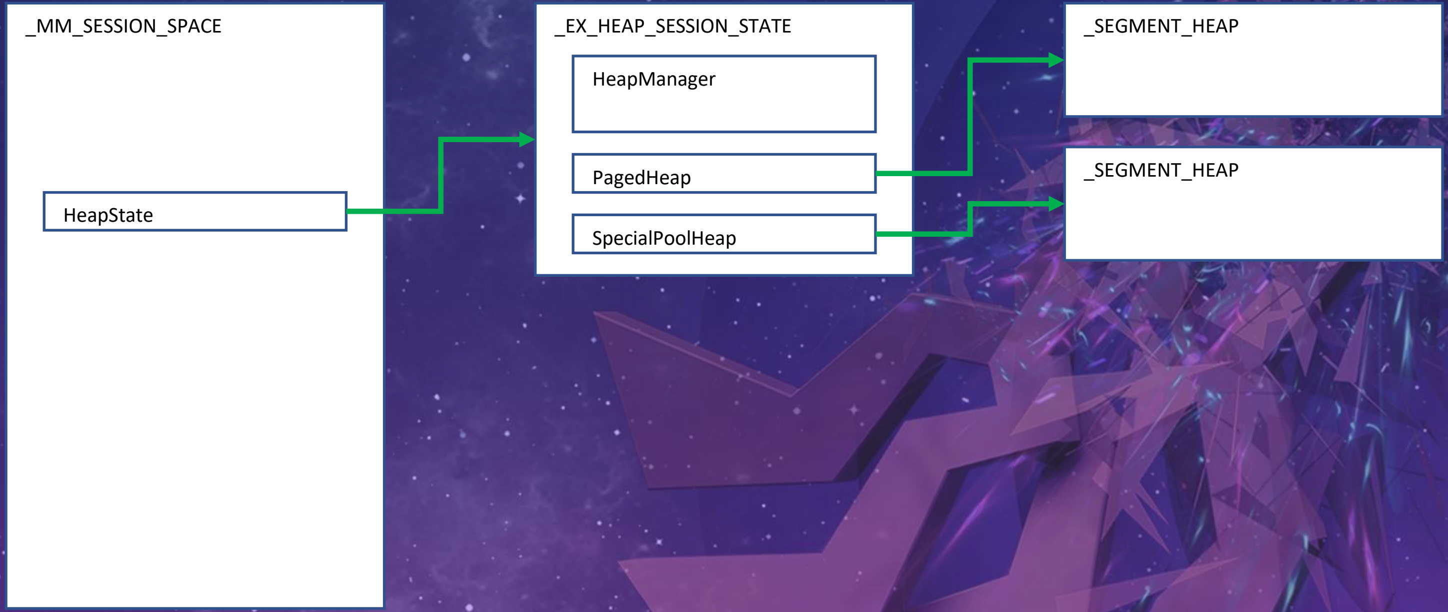
02

实现

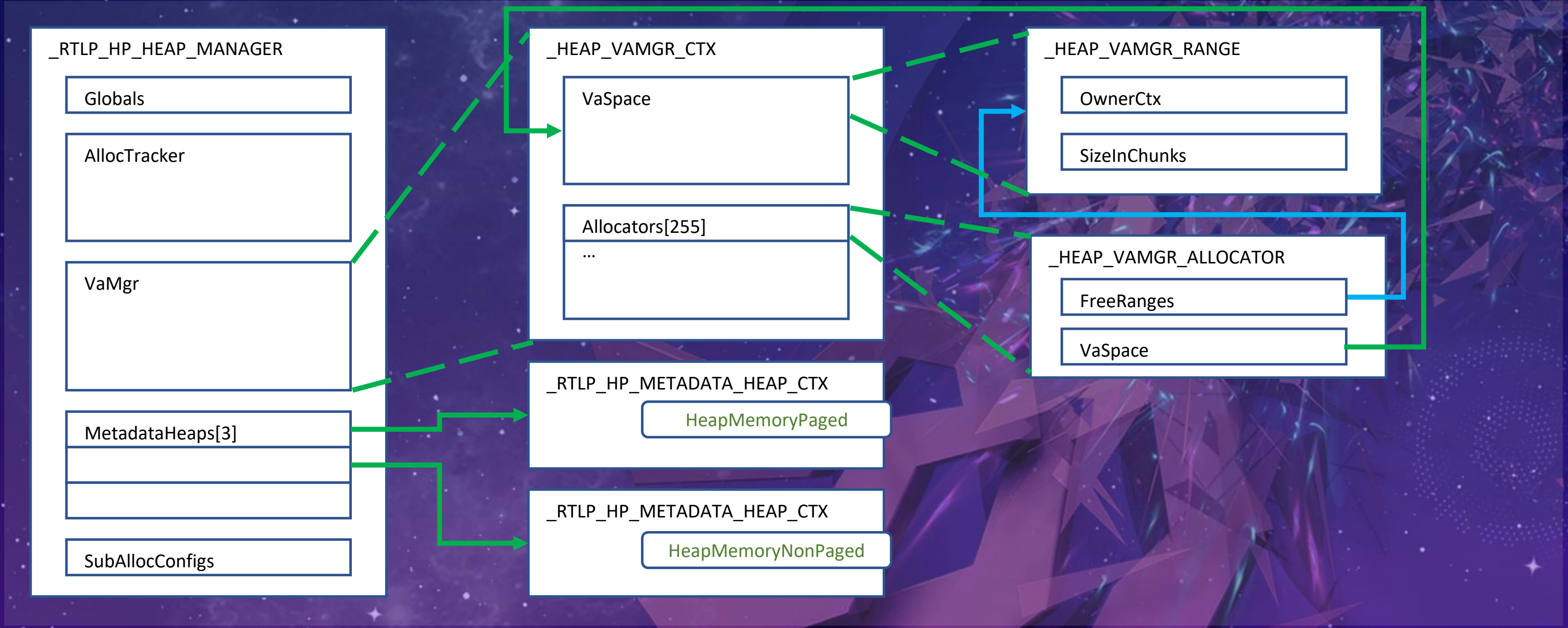
Heap Manager State



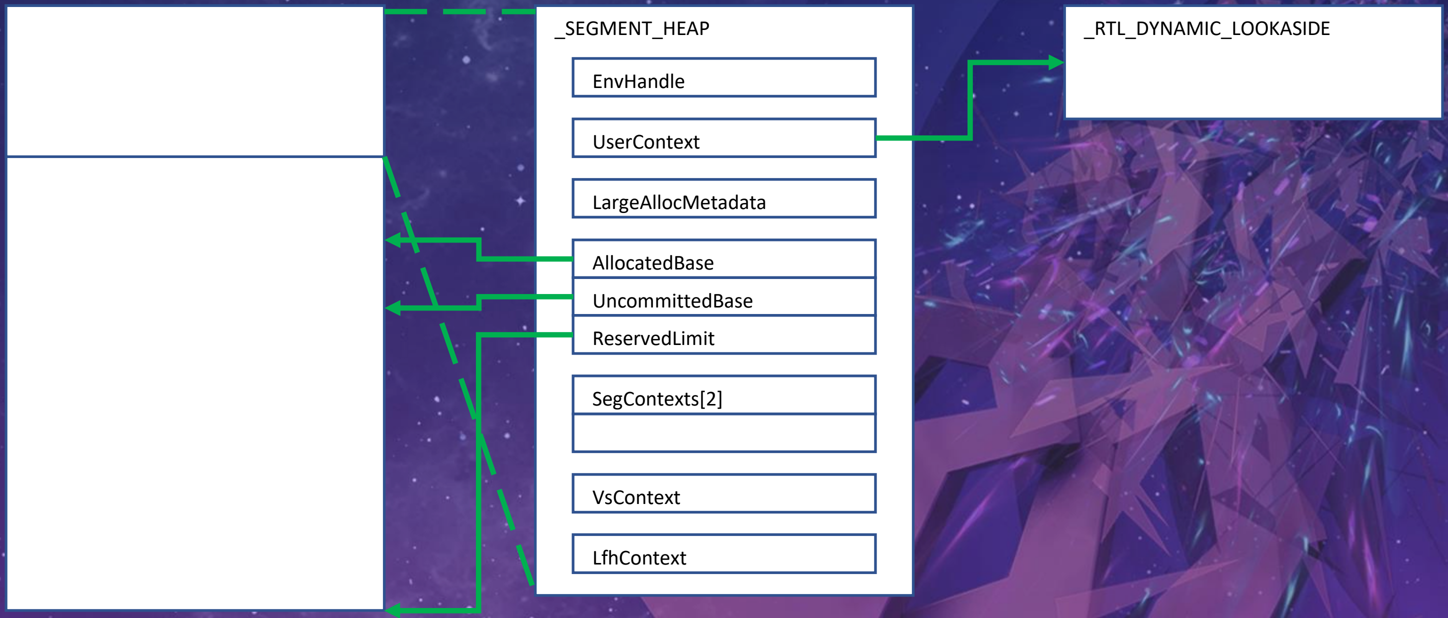
Heap Session State



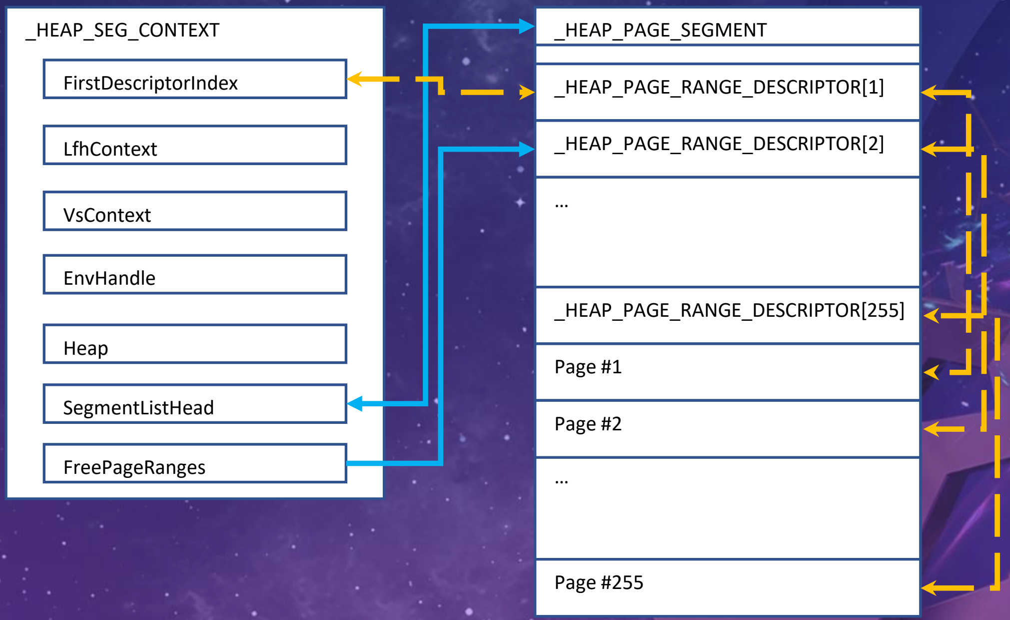
Heap Manager



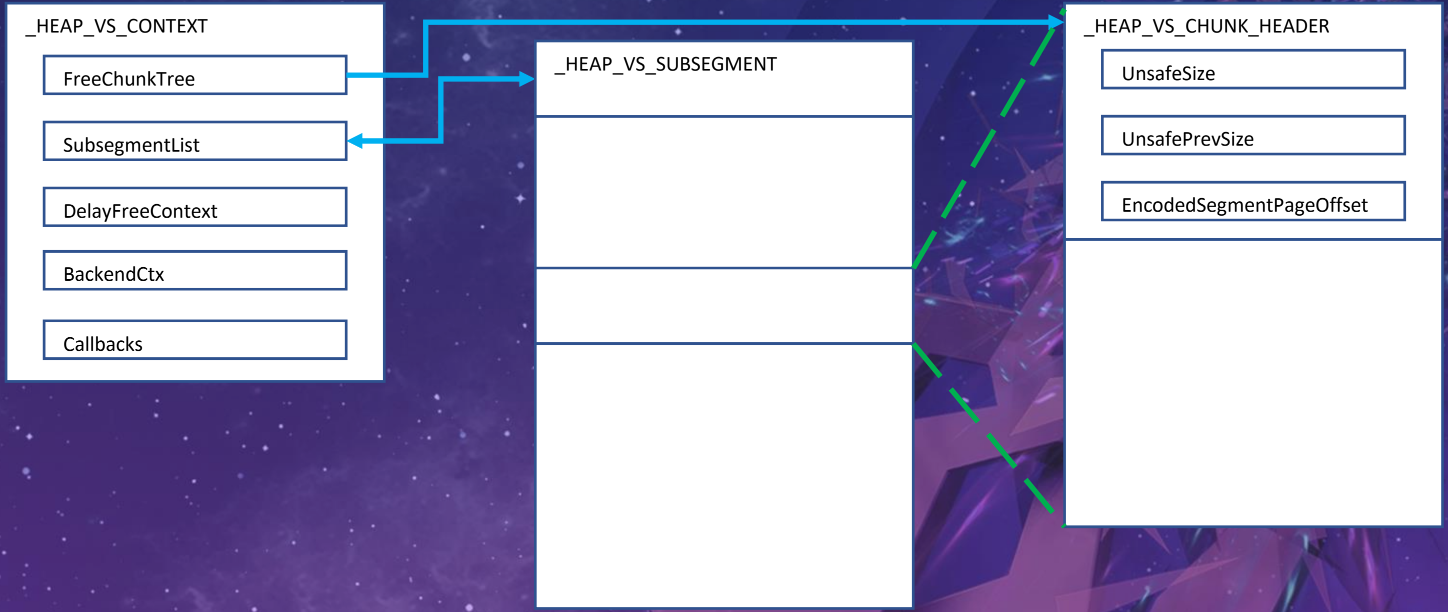
Segment Heap



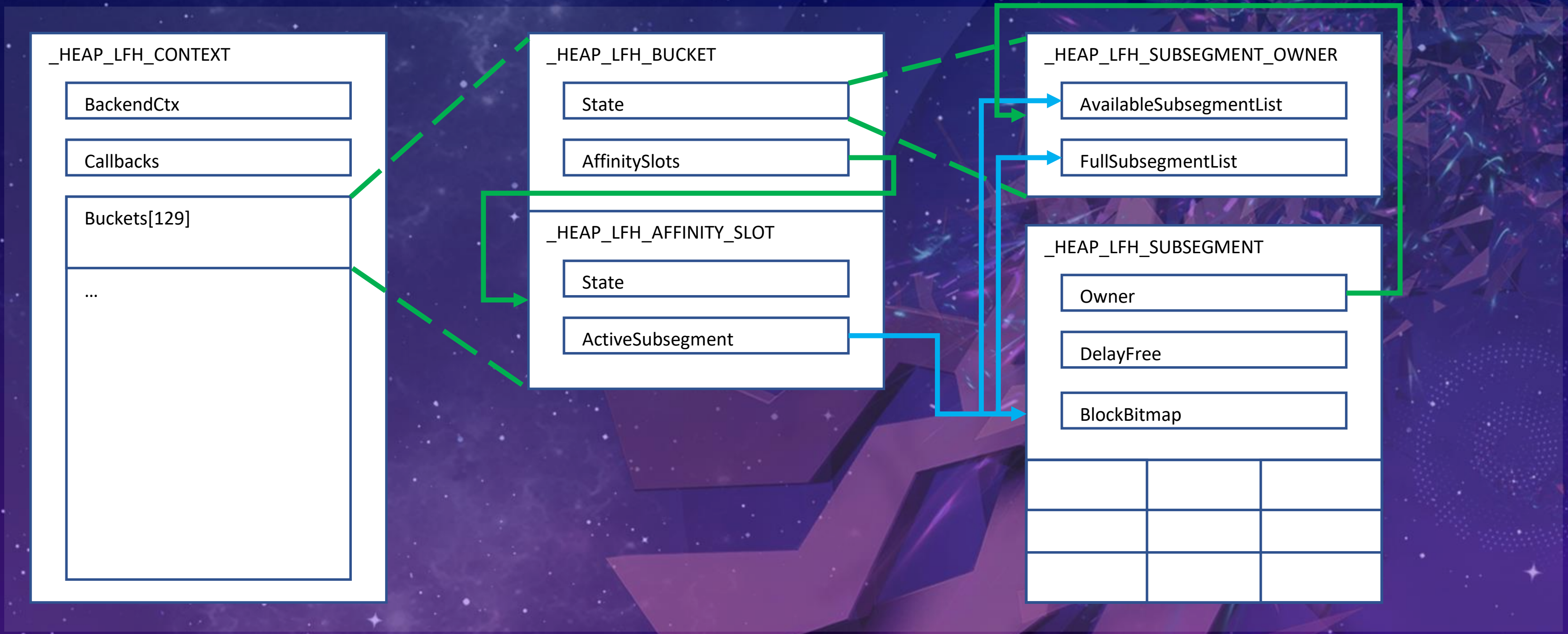
Segment Context



Variable Size Context

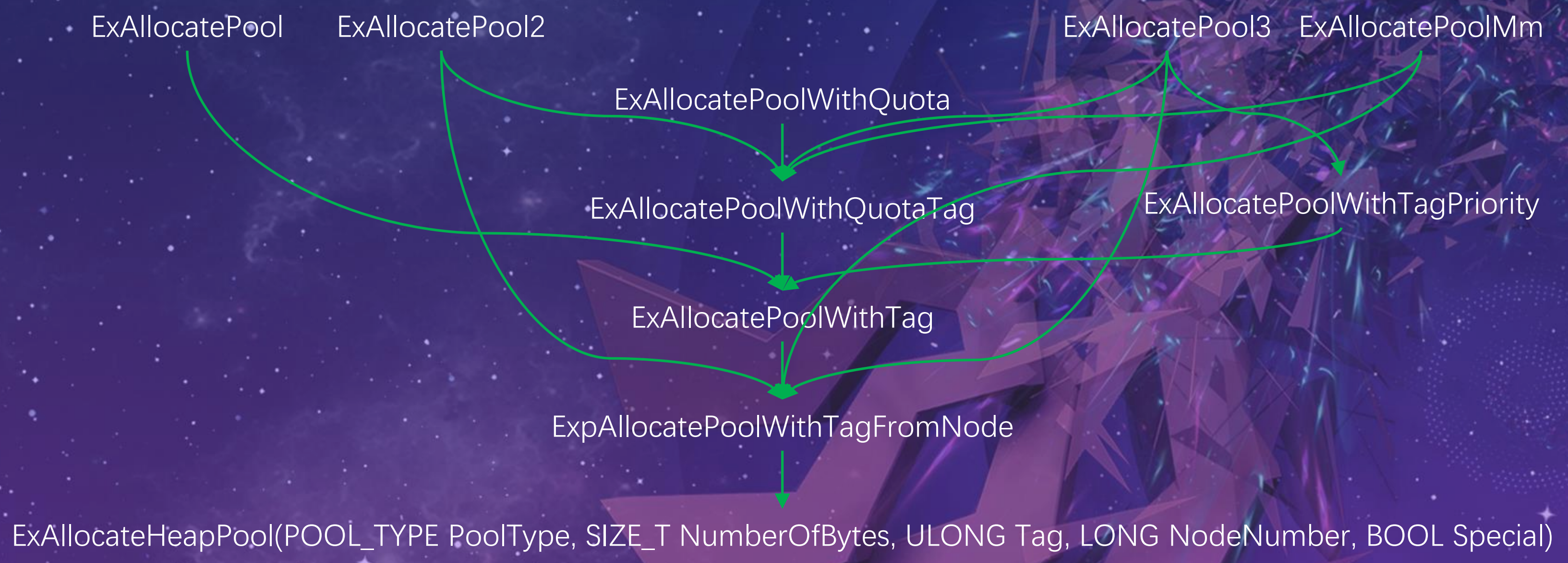


LFH Context



_HEAP_LFH_CONTEXT
BackendCtx
Callbacks
Buckets[129]
...
_HEAP_LFH_BUCKET
State
AffinitySlots
_HEAP_LFH_AFFINITY_SLOT
State
ActiveSubsegment
_HEAP_LFH_SUBSEGMENT_OWNER
AvailableSubsegmentList
FullSubsegmentList
_HEAP_LFH_SUBSEGMENT
Owner
DelayFree
BlockBitmap

内 存 分 配


ExAllocatePool
ExAllocatePool2
ExAllocatePool3
ExAllocatePoolMm
ExAllocatePoolWithQuota
ExAllocatePoolWithQuotaTag
ExAllocatePoolWithTagPriority
ExAllocatePoolWithTag
ExpAllocatePoolWithTagFromNode
ExAllocateHeapPool(POOL_TYPE PoolType, SIZE_T NumberOfBytes, ULONG Tag, LONG NodeNumber, BOOL Special)

内 存 分 配

从哪个堆中分配?

- NonPagedPool   => ExPoolState.PoolNode[NodeNumber].Heaps[0]
- NonPagedPoolNx   => ExPoolState.PoolNode[NodeNumber].Heaps[1]
- PagedPool   => ExPoolState.PoolNode[NodeNumber].Heaps[2]
- PagedPoolSession => PsGetCurrentProcess()->Session->HeapState->HeapManager.PagedHeap

内 存 分 配

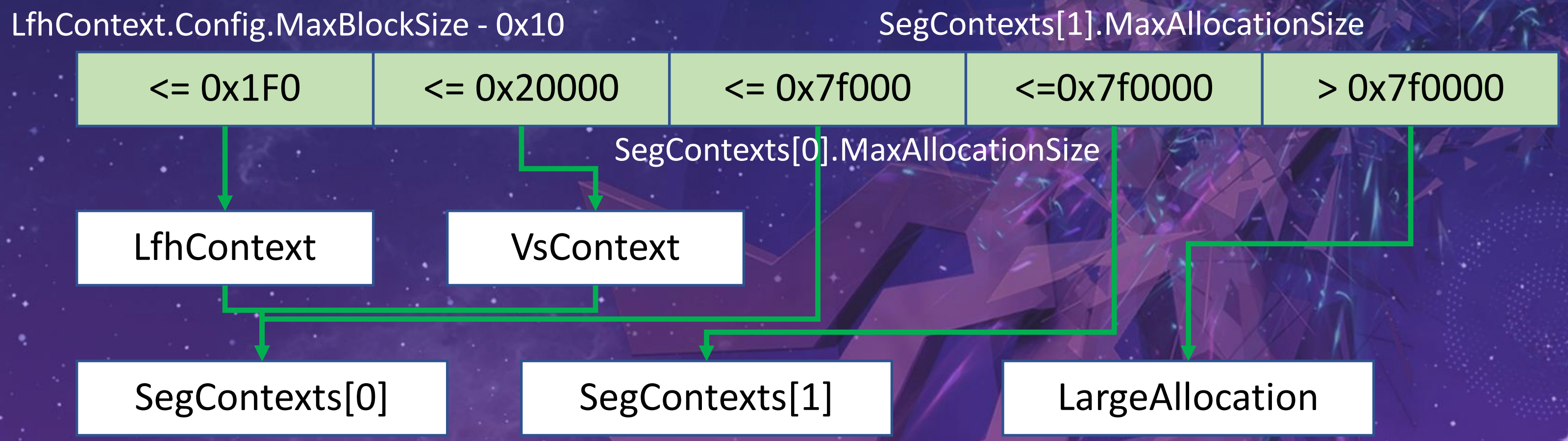
从哪个 Context 中分配?

首先判断是否尝试从 UserContext 中分配

- Size >= 0x201
- Size <= 0xF80
- Lookaside Bucket 已经激活
- Lookaside Bucket 队列不为空

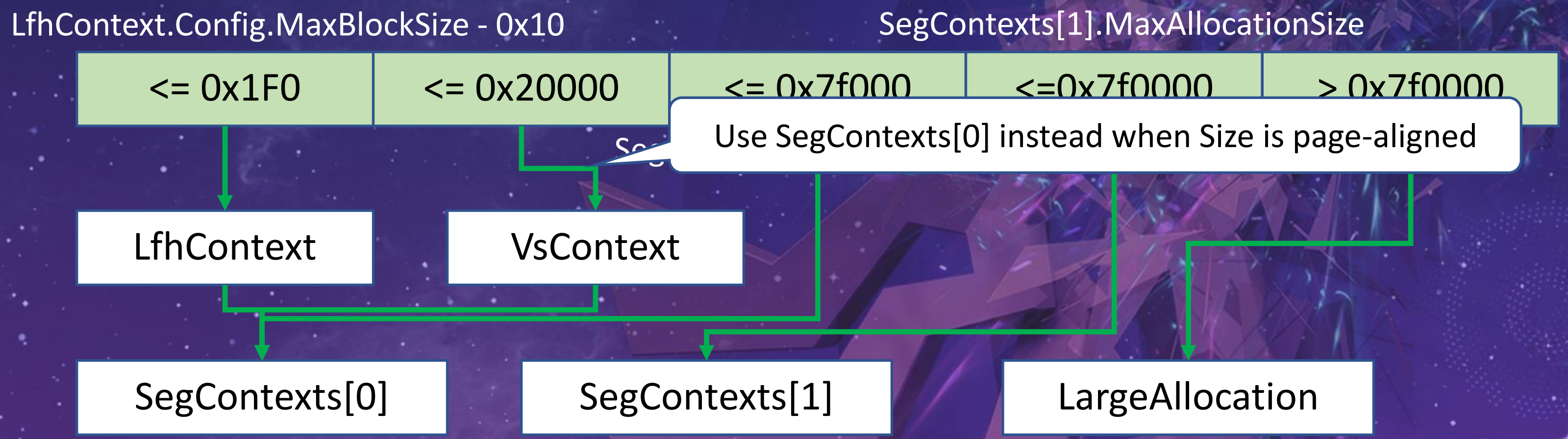
内 存 分 配

从哪个 Context 中分配?



内 存 分 配

从哪个 Context 中分配?



内 存 分 配

分配大小的调整

- Size < 0xFE0: 增加 0x10 字节的 _POOL_HEADER

  CACHE_ALIGNED_POOL_MASK: 增加 ExpCacheLineSize
- Size > 0x10000: 取整到页大小
- (Size & 0xFF0) > 0xFC0: 取整到页大小

内 存 释 放

ExFreePool　　ExFreePoolEx　　ExFreePoolWithTag

ExFreeHeapPool(PVOID P)

内 存 释 放

推断 Pool Type

MiState.Vs.SystemVaType 记录了所有内核空间虚拟地址的类型

```
MiVaUnused = 0n0
MiVaSessionSpace = 0n1
MiVaProcessSpace = 0n2
MiVaBootLoaded = 0n3
MiVaPfnDatabase = 0n4
MiVaNonPagedPool = 0n5
MiVaPagedPool = 0n6
MiVaSpecialPoolPaged = 0n7
MiVaSystemCache = 0n8
MiVaSystemPtes = 0n9
MiVaHal = 0n10
MiVaSessionGlobalSpace = 0n11
MiVaDriverImages = 0n12
MiVaSystemPtesLarge = 0n13
MiVaKernelStacks = 0n14
MiVaSecureNonPagedPool = 0n15
MiVaMaximumType = 0n16
```

=> PagedPoolSession

=> NonPagedPool

=> PagedPool

内 存 释 放

推断 Heap Manager

用 Pool Type 来构造 RTL_HP_ENV_HANDLE

- PagedPoolSession => HeapAddressSession
- NonPagedPool => HeapAddressKernel
- PagedPool => HeapAddressKernel

获取 RTL_HP_ENV_HANDLE 对应的 Heap Manager

- HeapAddressKernel => ExPoolState.HeapManager
- HeapAddressSession => PsGetCurrentProcess()->Session->HeapState->HeapManager

内 存 释 放

推断 Chunk 类型

HeapManager->AllocTracker 记录了所有 chunk 的类型

- 0 => 通过 Large Allocation 分配
- 1 => 通过 SegContexts[0] 分配
- 2 => 通过 SegContexts[1] 分配
- 3 => 通过 Large Allocation 分配

内存释放

推断 Large Allocation 使用的堆

内存元数据中的 OwnerCtx 指向分配时所使用的堆

调用 RtlpHpQueryVA 来获取内存元数据

内 存 释 放

推断 Segment Allocation 使用的堆

通过 _HEAP_SEG_CONTEXT.SegmentMask 来计算 _HEAP_PAGE_SEGMENT 的地址

- SegContexts[0] => 0xffffffff`fff00000
- SegContexts[1] => 0xffffffff`ff000000

通过 _HEAP_PAGE_SEGMENT.Signature 来计算 _HEAP_SEG_CONTEXT 的地址

- Signature = 0xA2E64EADA2E64EAD ^ PageSegment ^ SegContext ^ HeapKey

通过 _HEAP_SEG_CONTEXT 来计算 _SEGMENT_HEAP 的地址

- SegContexts[0] => -0x100
- SegContexts[1] => -0x1C0

内存释放

推断分配时使用的 Context

计算 _HEAP_PAGE_RANGE_DESCRIPTOR 和 Subsegment 的地址

- (P == Subsegment) => SegContext
- ((_HEAP_PAGE_RANGE_DESCRIPTOR->RangeFlags & 0xC) == 8) => LfhContext
- ((_HEAP_PAGE_RANGE_DESCRIPTOR->RangeFlags & 0xC) != 8) => VsContext

03

安全机制

安 全 机 制

保护链表的节点

删除节点时的校验

Entry->Flink->Blink == Entry->Blink->Flink == Entry

插入节点时的校验

Entry->Blink->Flink == Entry

安 全 机 制

保护红黑树的节点

指针编码

EncodedEntry = Entry ^ Entry->Parent

操作父节点时的校验

Entry->Parent->Children[] == Entry

安 全 机 制

Guard Page

在分配内存时多分配一个页面

保留分配的最后一个页面但是不提交该页面

阻止缓冲区溢出越过分配的边界改写后续内存

安 全 机 制

关键数据编码

_HEAP_VS_CONTEXT.SubsegmentList

_HEAP_VS_CONTEXT.BackendCtx

_HEAP_VS_CONTEXT.Callbacks

_HEAP_VS_SUBSEGMENT.ListEntry

_HEAP_VS_SUBSEGMENT.Signature

_HEAP_VS_CHUNK_HEADER

安全机制

关键数据编码

_HEAP_LFH_CONTEXT.Callbacks

_HEAP_LFH_SUBSEGMENT.BlockOffsets

安全机制

LFH 分配的随机化

从 RtlpLowFragHeapRandomData 数组中获取随机数

搜索 LFH bitmap 时用这个随机数来添加随机性

对漏洞利用的影响

移除了先前的池管理的代码

Modern Kernel Pool Exploitation:
Attacks and Techniques

Tarjei Mandt | Infiltrate 2011

对漏洞利用的影响

内存布局控制更加困难

对 漏 洞 利 用 的 影 响

对已释放内存的占位更加困难

LFH 分配的不确定性

- Lookaside Bucket 根据内存使用情况来激活与停用
- 释放内存到当前 subsegment 并不增加其可用内存块数量
- Subsegment 内分配的随机化

VS 分配启用了延迟释放

04

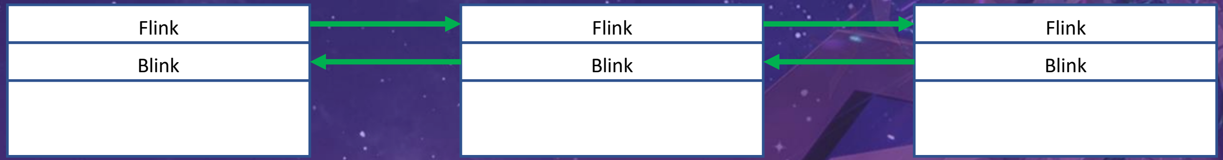
攻击面

攻 击 面

Safe Unlinking 的弱点

Safe Unlinking 设计用来防止缓冲区溢出被转化成 Write-What-Where

通过缓冲区溢出来劫持链表节点是可行的

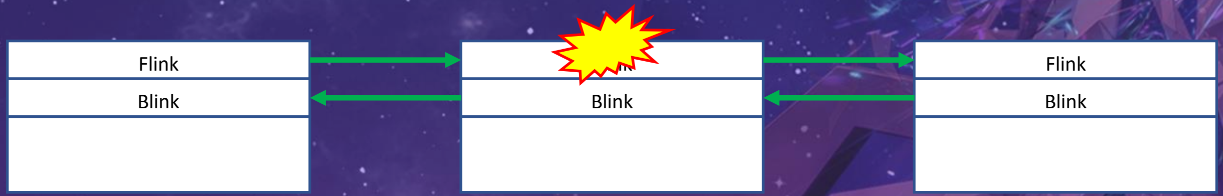


攻 击 面

Safe Unlinking 的弱点

Safe Unlinking 设计用来防止缓冲区溢出被转化成 Write-What-Where

通过缓冲区溢出来劫持链表节点是可行的



攻 击 面

Safe Unlinking 的弱点

Safe Unlinking 设计用来防止缓冲区溢出被转化成 Write-What-Where

通过缓冲区溢出来劫持链表节点是可行的

攻击面

通过 SegContext 分配的内存没有 Guard Page

缓冲区溢出可以改写后续的内存

- VS Subsegment
- LFH Subsegment

攻击面

没有编码的关键数据

_POOL_HEADER

_HEAP_LFH_SUBSEGMENT.ListEntry

利 用 技 巧　1

较小的内存分配中保留了 _POOL_HEADER

```
nt!_POOL_HEADER
   +0x000 PreviousSize     : Pos 0, 8 Bits
   +0x000 PoolIndex        : Pos 8, 8 Bits
   +0x002 BlockSize        : Pos 0, 8 Bits
   +0x002 PoolType         : Pos 8, 8 Bits
   +0x000 Ulong1           : Uint4B
   +0x004 PoolTag          : Uint4B
   +0x008 ProcessBilled    : Ptr64 _EPROCESS
   +0x008 AllocatorBackTraceIndex : Uint2B
   +0x00a PoolTagHash      : Uint2B
```

利用技巧 1

缓存对齐的内存分配

分配 Size + ExpCacheLineSize 大小的内存

在缓存对齐的地址将该内存分为两块

前一块的 _POOL_HEADER

- PreviousSize = 0
- BlockSize = Size + ExpCacheLineSize
- PoolType = PoolType & ~CACHE_ALIGNED_POOL_MASK

后一块的 _POOL_HEADER

- PreviousSize = SplitSize
- BlockSize = Size + ExpCacheLineSize – SplitSize
- PoolType = PoolType | CACHE_ALIGNED_POOL_MASK

利用技巧 1

缓存对齐的内存释放

恢复 _POOL_HEADER Entry

Entry -= Entry->PreviousSize * 0x10

Entry->PoolType |= CACHE_ALIGNED_POOL_MASK

利用技巧 1

溢出 _POOL_HEADER

PreviousSize 0	PoolIndex 0	BlockSize S	PoolType 2	P1	PreviousSize 0	PoolIndex 0	BlockSize S	PoolType 2	P2

利用技巧 1

溢出 _POOL_HEADER

PreviousSize 0	PoolIndex 0	BlockSize S	PoolType 2	P1	PreviousSize 0	PoolIndex 0	BlockSize S	PoolType 2	P2

利用技巧 1

溢出 _POOL_HEADER

PreviousSize 0	PoolIndex 0	BlockSize S	PoolType 2	P1	PreviousSize **S**	PoolIndex 0	BlockSize S	PoolType **6**	P2

利 用 技 巧 1

溢出 _POOL_HEADER

此时释放 P2 实际释放的将会是 P1

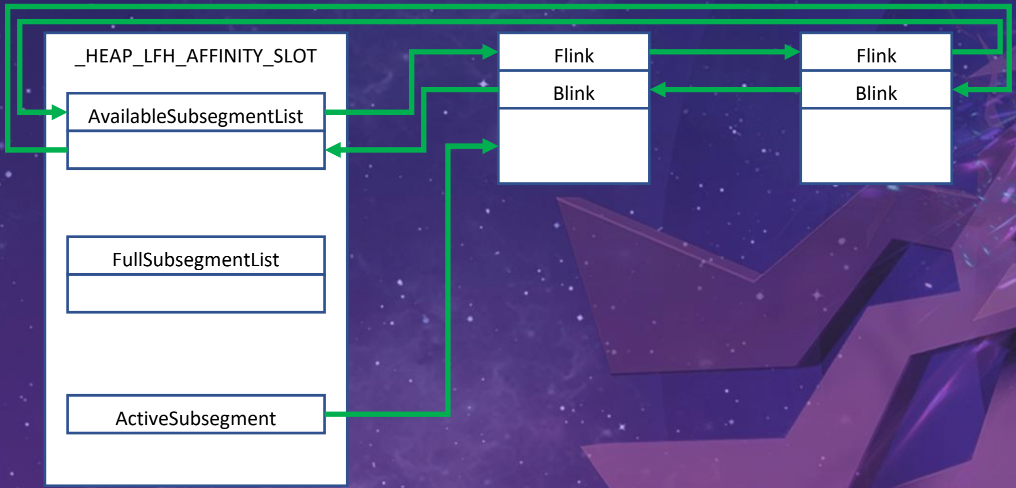
在激活了 Lookaside Bucket 的情况下后续的内存分配将使用 P1

利 用 技 巧　2

溢出 _HEAP_LFH_SUBSEGMENT

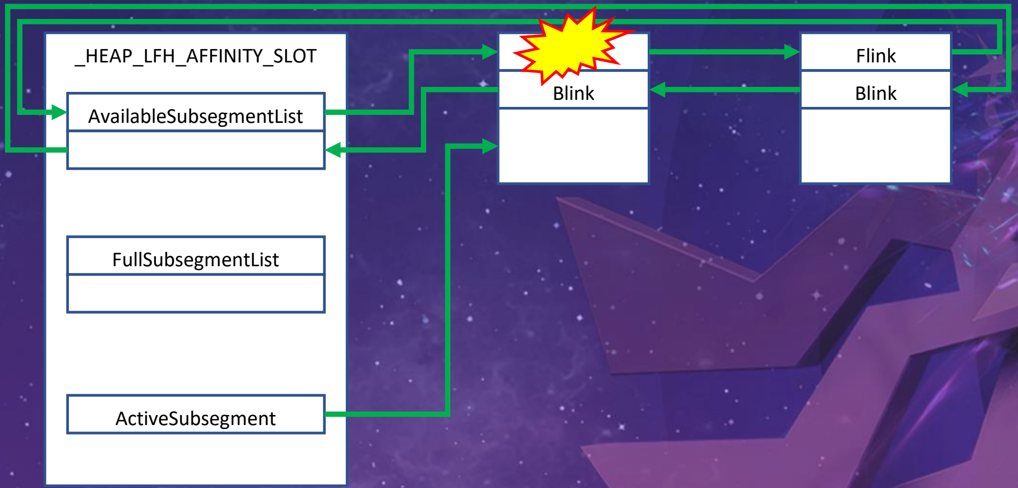
利 用 技 巧 2

溢出 _HEAP_LFH_SUBSEGMENT



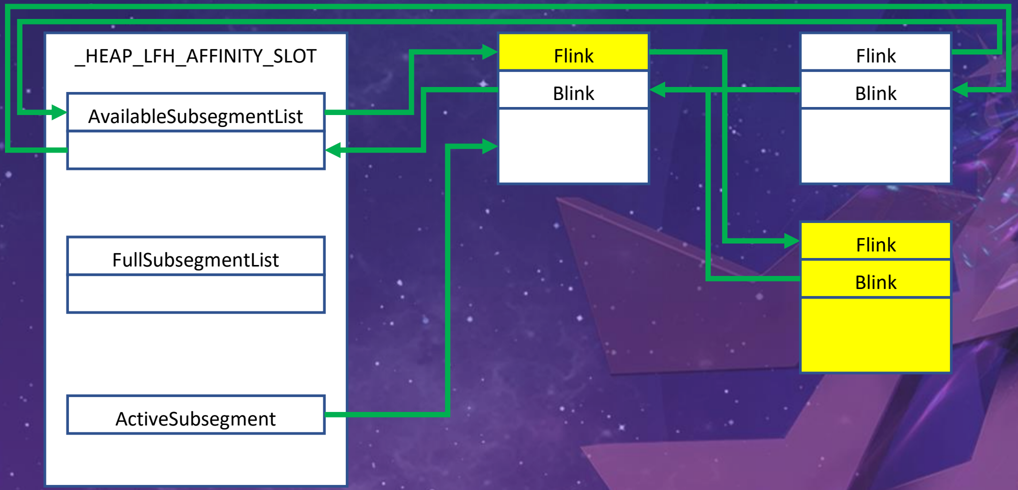
利用技巧 2

溢出 _HEAP_LFH_SUBSEGMENT



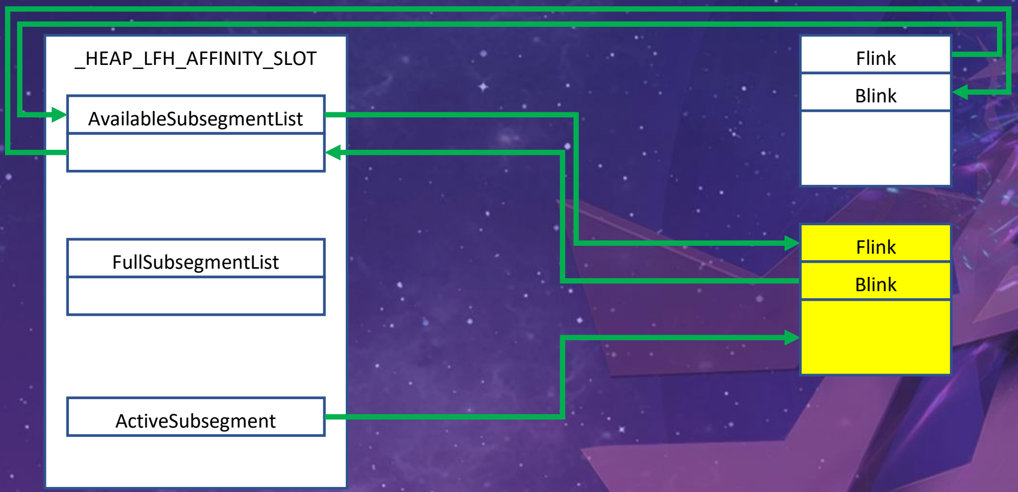
利用技巧 2

溢出 _HEAP_LFH_SUBSEGMENT



利用技巧 2

溢出 _HEAP_LFH_SUBSEGMENT



利用技巧 3

Alex Ionescu 的谜题

Alex Ionescu
@aionescu

Are you on Windows 10 RS5 or later? Do you really wish that you could easily get a kernel RWX page? Here's a way:

?
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)

利 用 技 巧　3

为什么这个页面是 RWX？

```
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)
```

PageSegment->Signature

CONTAINING_RECORD(Prcb->IdleThread
->Process
->MmProcessLinks.Blink,
_EPROCESS, MmProcessLinks)->Session
->HeapState
->HeapManager.Globals.HeapKey

0xa2e64ead`a2e64ea

PageSegment

利用技巧 3

为什么这个页面是 RWX？

```
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)
```

↓

SegContext

利 用 技 巧　3

为什么这个页面是 RWX？

```
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)
```

Heap->UserContext

利 用 技 巧 3

为什么这个页面是 RWX？

```
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)
```

↓

ExPoolState.PoolNode[0].Lookasides[1]

利 用 技 巧　3

为什么这个页面是 RWX？

```
poi(poi((poi((@$proc&-100000)+10)^poi(poi(poi(poi(poi(poi(poi(poi(@$prcb+18)+220)+648)+8)-240)+2a0)))^-0n6708588087252463955^(@$proc&-100000))-d8)-1080)
```

↓

ExPoolState.PoolNode[0].Heaps[0]

利用技巧 3

可预测的 RWX 页面

```
0: kd> !pte poi(ExPoolState+3900)
                                           VA ffff970e6a400000
PXE at FFFFED76BB5DA970    PPE at FFFFED76BB52E1C8    PDE at FFFFED76A5C39A90    PTE at FFFFED4B87352000
contains 0A00000004835863  contains 0A00000004838863  contains 0A0000000483F863  contains 0A0000013CA14863
pfn 4835      ---DA--KWEV  pfn 4838      ---DA--KWEV  pfn 483f      ---DA--KWEV  pfn 13ca14    ---DA--KWEV
```

THANK YOU FOR WATCHING!

非 常 感 谢 您 的 观 看