



ADVANCED HEAP MANIPULATION IN WINDOWS 8



Who Am I

*Zhenhua(Eric) Liu
Senior Security Researcher
Fortinet, Inc.*

Previous:

*Dissecting Adobe ReaderX's Sandbox:
Breeding Sandworms@BlackHat EU 2012*



Agenda

0x01:

Why start this research

0x02:

Quick View of The Idea

0x03:

Implementations

(Kernel Poll / User heap)

Intro



Why start this research. (Motivation)

*Exploiting Memory corruption vulnerability
are more difficult today*

Windows 8: Exploit mitigation improvements.

Possible ways for Sandbox bypassing

- *Kernel Vulnerability*
- *3rd-party plug-ins Vulnerability*
- *Sandbox flaws*

Windows 8 Kernel

-- The patched Win 7 Kernel

A: NULL Dereference protection

B: Kernel pool integrity checks

C: Non-paged pool NX

D: Enhanced ASLR

E: SMEP/PXN

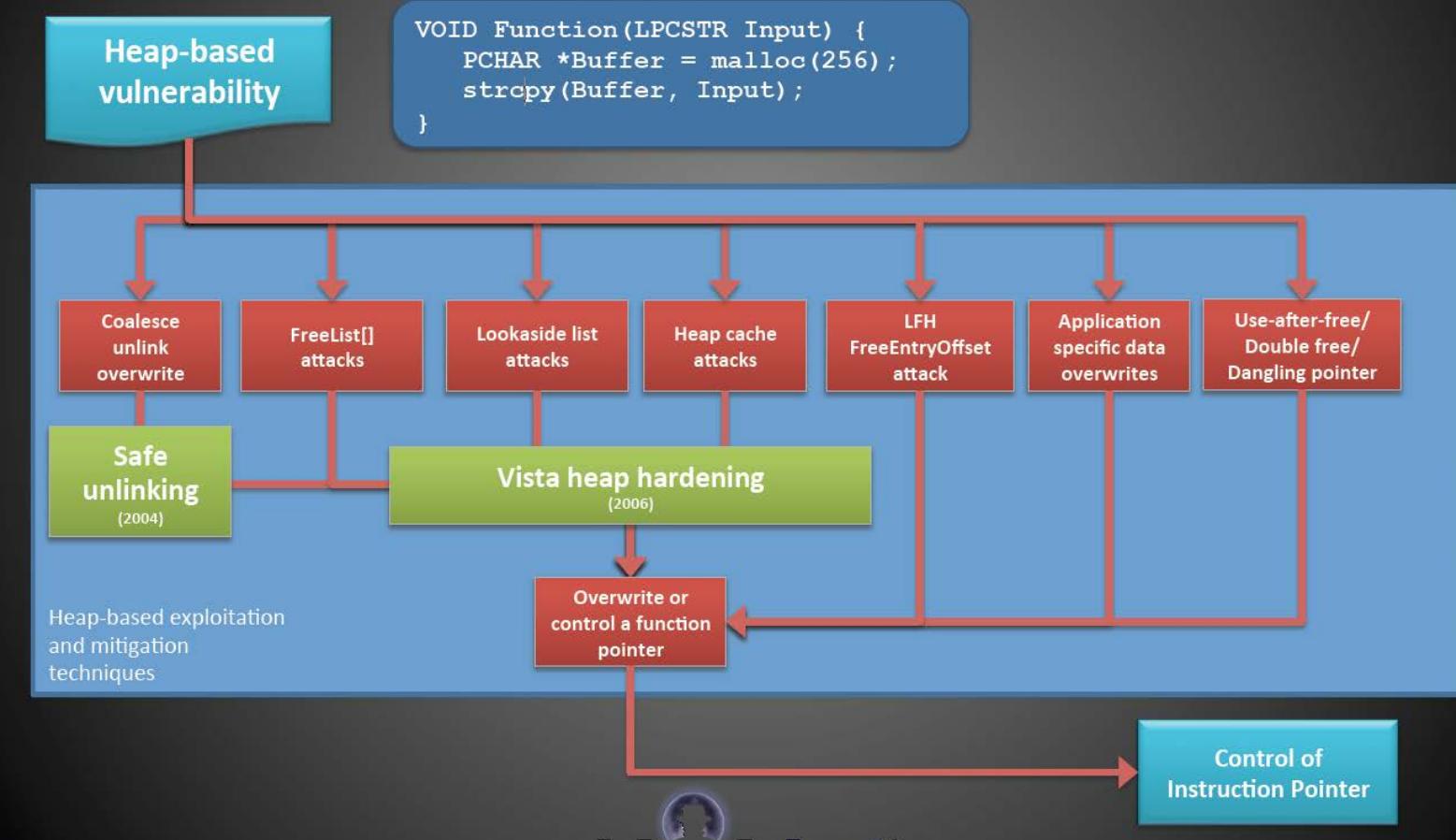
Windows 8 User Heap

-- determinism is at a all time low

A: High entropy Randomized LFH allocator

B: Guard pages

What's left



Matt Miller

http://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf

Why Application Specific Data attacking?

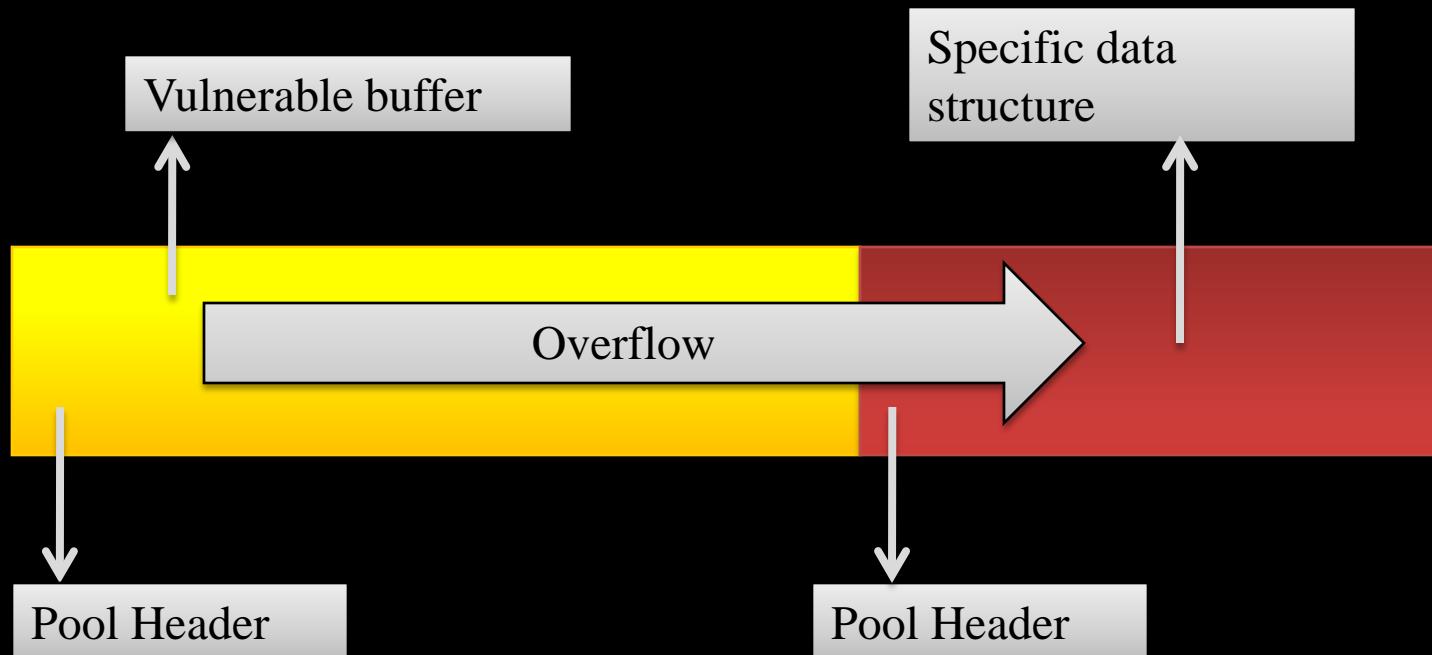
Application Specific data attacking are the future.

- Ben Hawkes

*Compromising Application Specific data are
facilitated by **heap manipulation***

What is ...

Overflow the target application's data stored on the heap.
Adjacent is the key!





风水 feng shui



Taken

Noise

Free

Vul buffer

Defragment

0x200

Taken

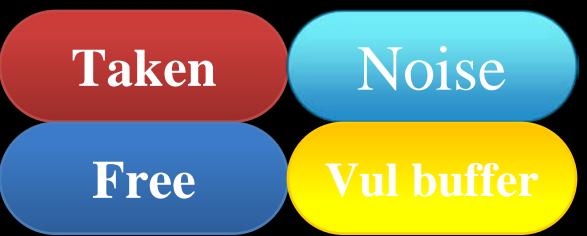
Noise

Free

Vul buffer

Make Holes

0x200



Allocate vulnerable buffer



vulnerable buffer will fall
into this place



The limitations 1:

Arbitrary size of vulnerable buffer?

We can not always find kernel object which size is the same as the vulnerable buffer, and it also contains the data structure for exploitation.



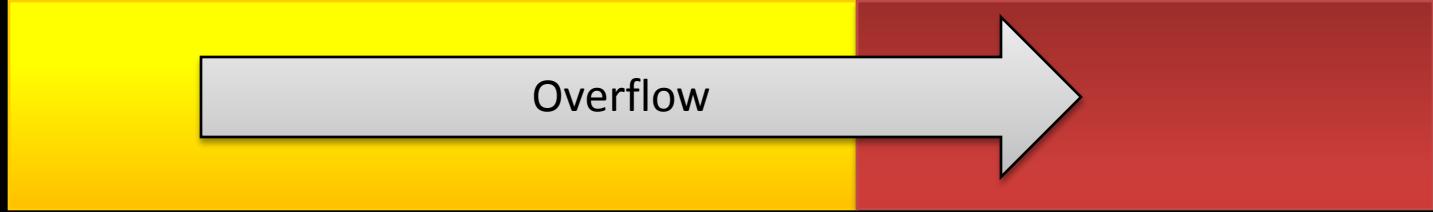
The limitations 2:

Randomized LFH makes it fail

*Defragment will trigger Randomized LFH,
vulnerable buffer will not fall into the hole we made.*

Target of This Research

- let the arbitrary vulnerable buffer adjacent with arbitrary data structure.
- without triggering the LFH in user heap.



Overflow

0x01:

Quick View of The Idea

Windows Objects in Kernel Vulnerability Exploitation

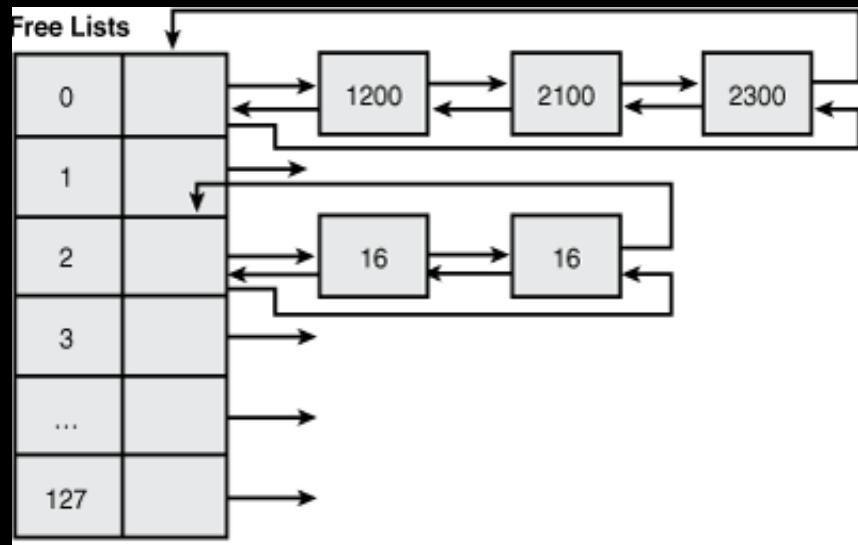
- How to place a desired object just behind the vulnerable buffer?
- Can we place something else other than object?

FreeLists

A: *Doubly linked lists*

B: *For fast allocation and free*

C: *LIFO manner*





Drawbacks

A: Metadata attacking

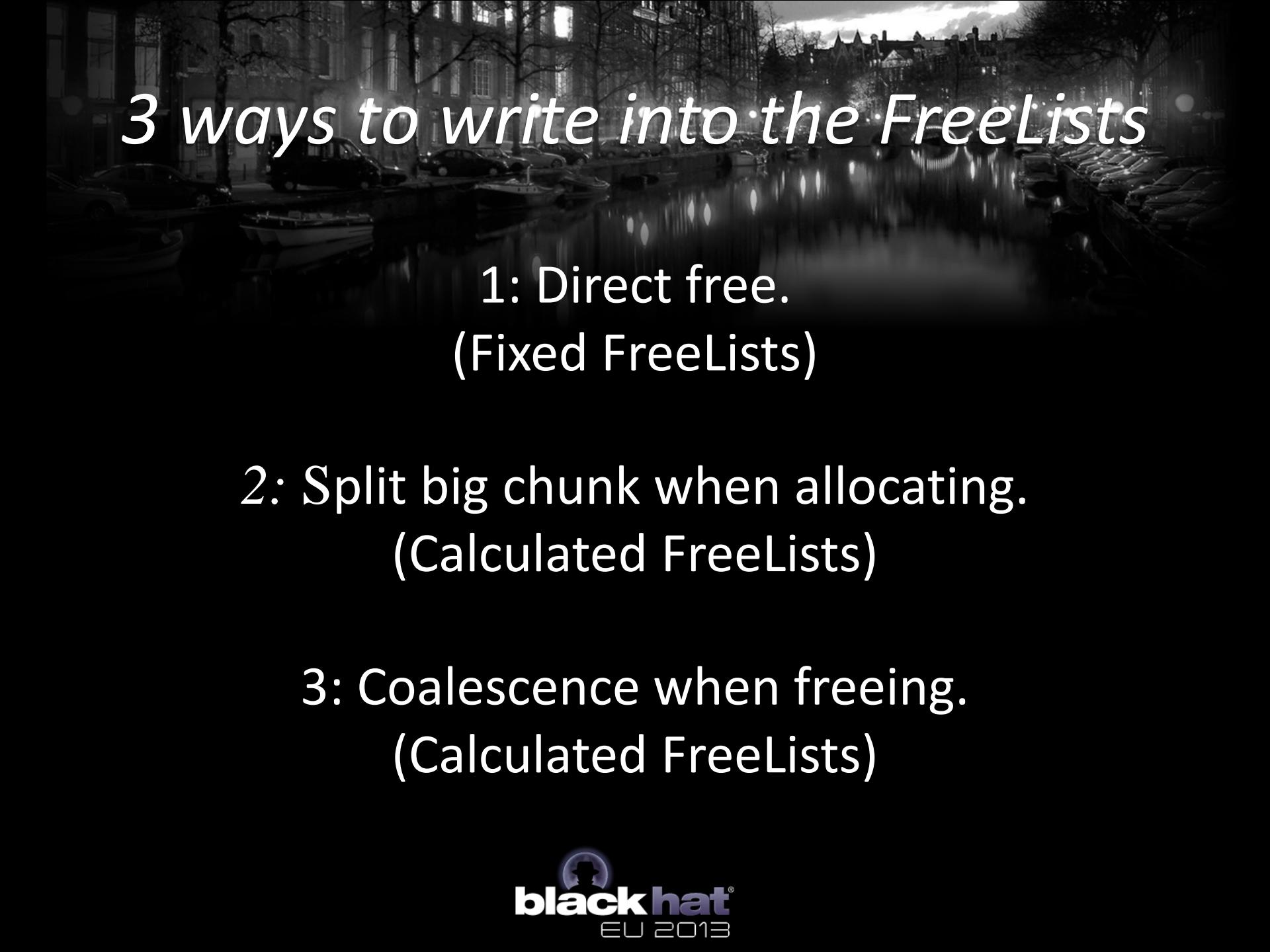
*B: **0** allocation entropy*

FreeLists are still been used in both kernel pool and user heap as of Windows 8.

Control the FreeLists



<http://sushibandit.com/wp-content/uploads/2010/04/belt.jpg>



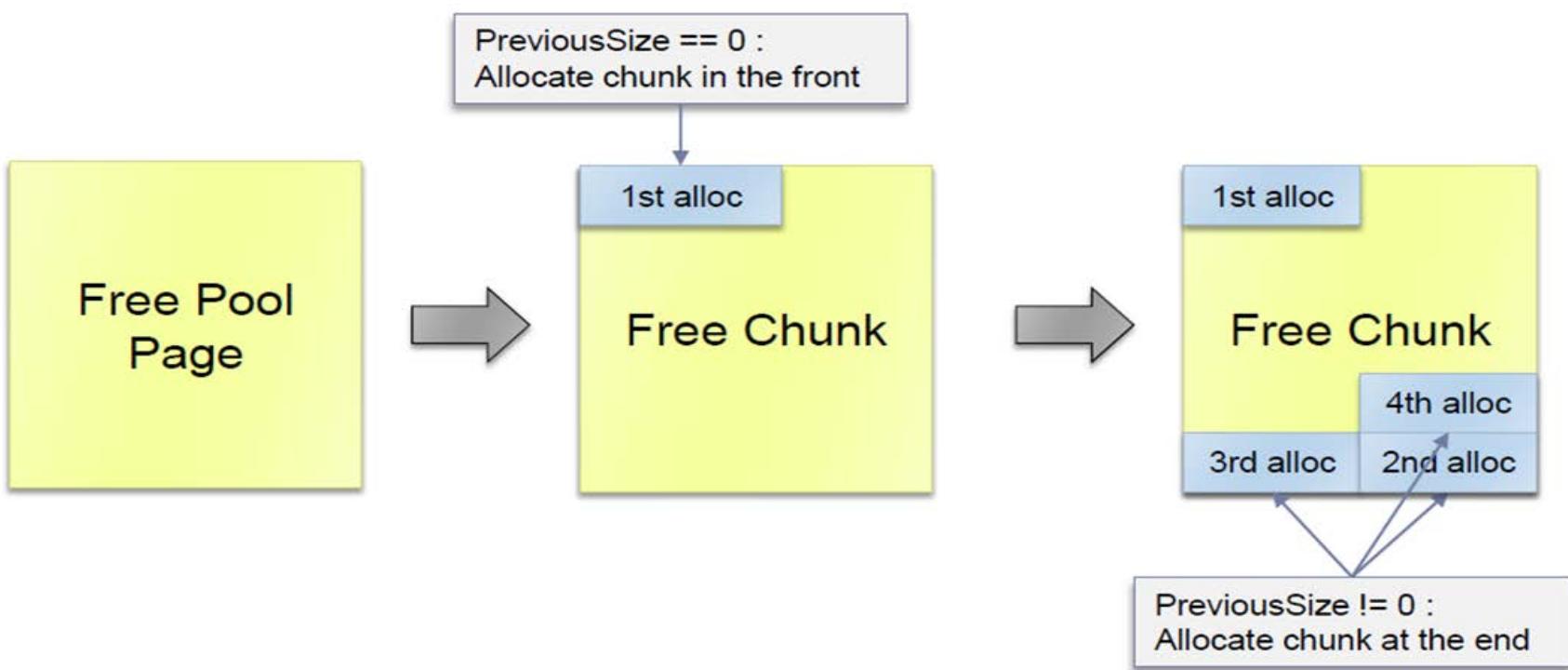
3 ways to write into the FreeLists

1: Direct free.
(Fixed FreeLists)

2: Split big chunk when allocating.
(Calculated FreeLists)

3: Coalescence when freeing.
(Calculated FreeLists)

Splitting Pool Chunks process



The Mandatory Search Technique

- Force the FreeLists searching process to take place.
 - Force the searching result greater than requested.
- *To control the Freelisters dynamically when allocating*

The Mandatory Search Technique

ExAllocatePoolWithTag

Size?

Lookaside
Searching

Small Pool

Medium
Pool

Large Pool

Success?

Evaluation
N

FreeList
Searching

Success?

N

expand the pool using
MiAllocatePoolPages
and split

Y

Y

Y

The Mandatory Search Technique

ExAllocatePoolWithTag

Lookaside
Searching

Success?

Evaluation
N

FreeList
Searching

Success?

Y

black hat®
EU 2013

Small Pool

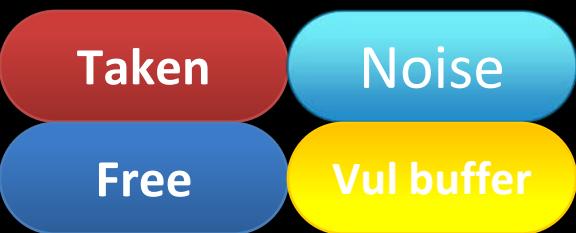
Medium

Size?

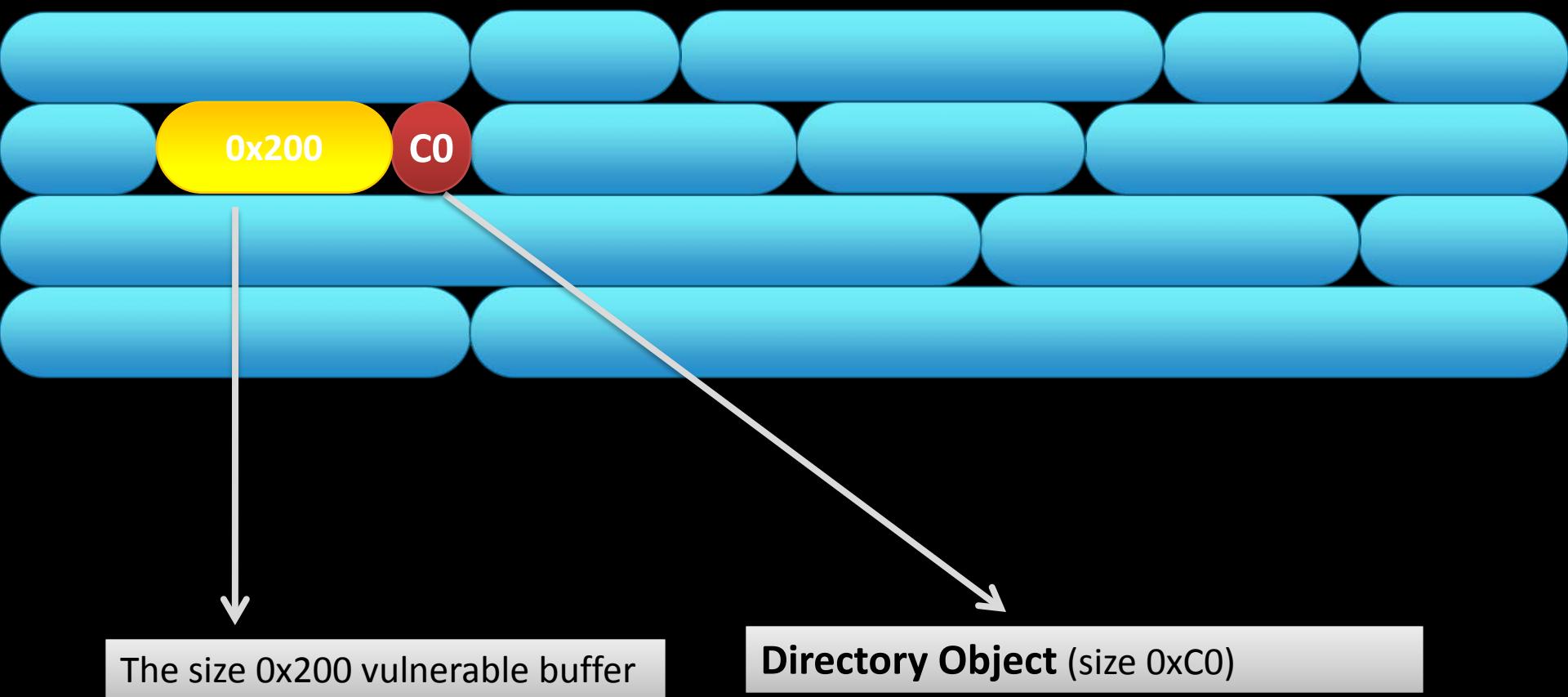
```
P RtIpFindEntry();  
RtIpHeapRemoveListEntry();  
// FreeListEntry is controlled
```

```
if (CommitSize < FreeListEntry ->Size){  
// Force the CommitSize smaller than  
// the FreeListEntry ->Size  
    RtIpCreateSplitBlock();  
}  
return Chunk
```

Return



The target



Taken

Noise

Free

0x01: Initial status

0x1000

0x1000

0x1000

0x1000

0x1000

Taken

Noise

Free

0x02: Alloc 0x808 block

0x808

0x7F8

0x808

0x7F8

0x808

0x7F8

0x808

0x7F8

0x808

0x7F8

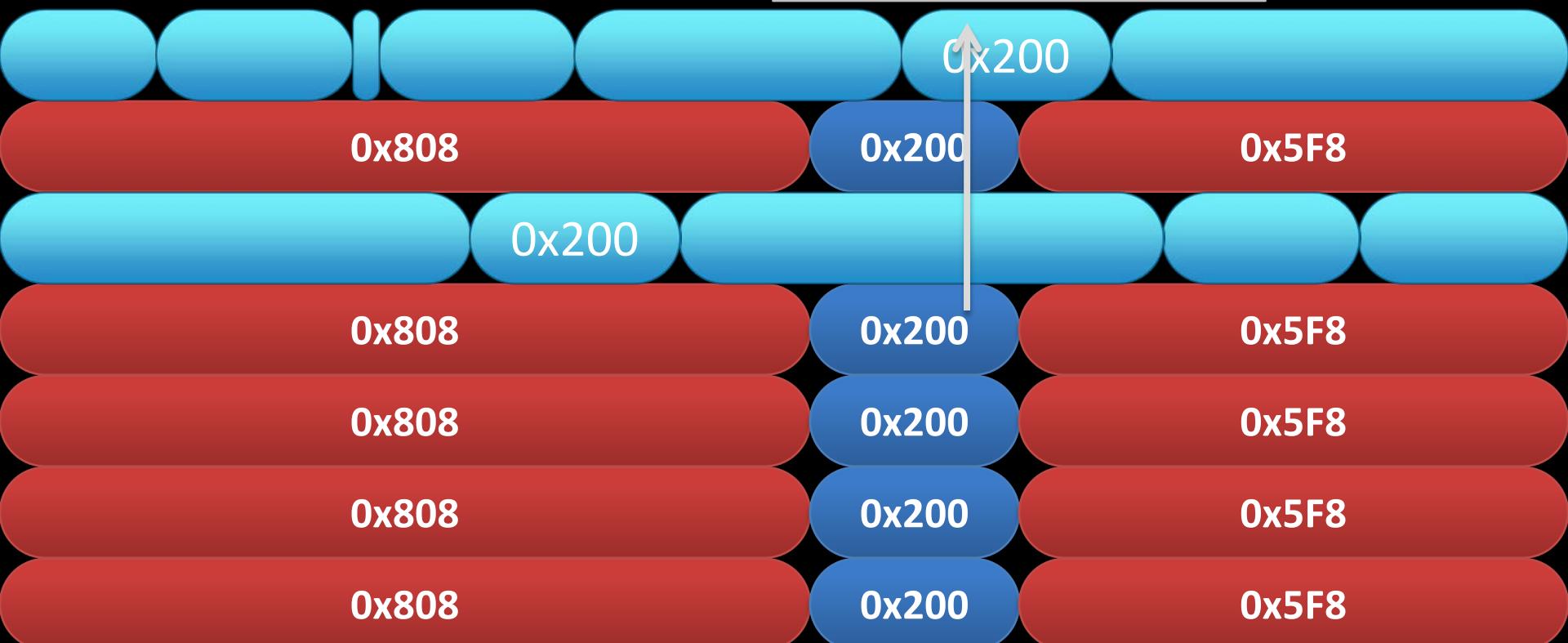
Taken

Noise

Free

0x03: Alloc 0x5F8 block and make 0x200 hole

The same size as of
vulnerable buffer



Taken

Noise

Free

0x04: Alloc 0x200 block



Taken

Noise

Free

0x05: Free 0x5F8 block



Taken

Free

Noise

0x06: Alloc 0x538 block and make 0xC0 hole



Taken

Noise

Free

0x07: Alloc 0xC0 block

Data structure we want
corruption to



Taken

Noise

Free

Vul buffer

0x08: Make 0x200 Holes



Taken

Noise

Free

Vul buffer

Trigger the vulnerability:
vulnerable buffer will fall
into one of the holes
eventually





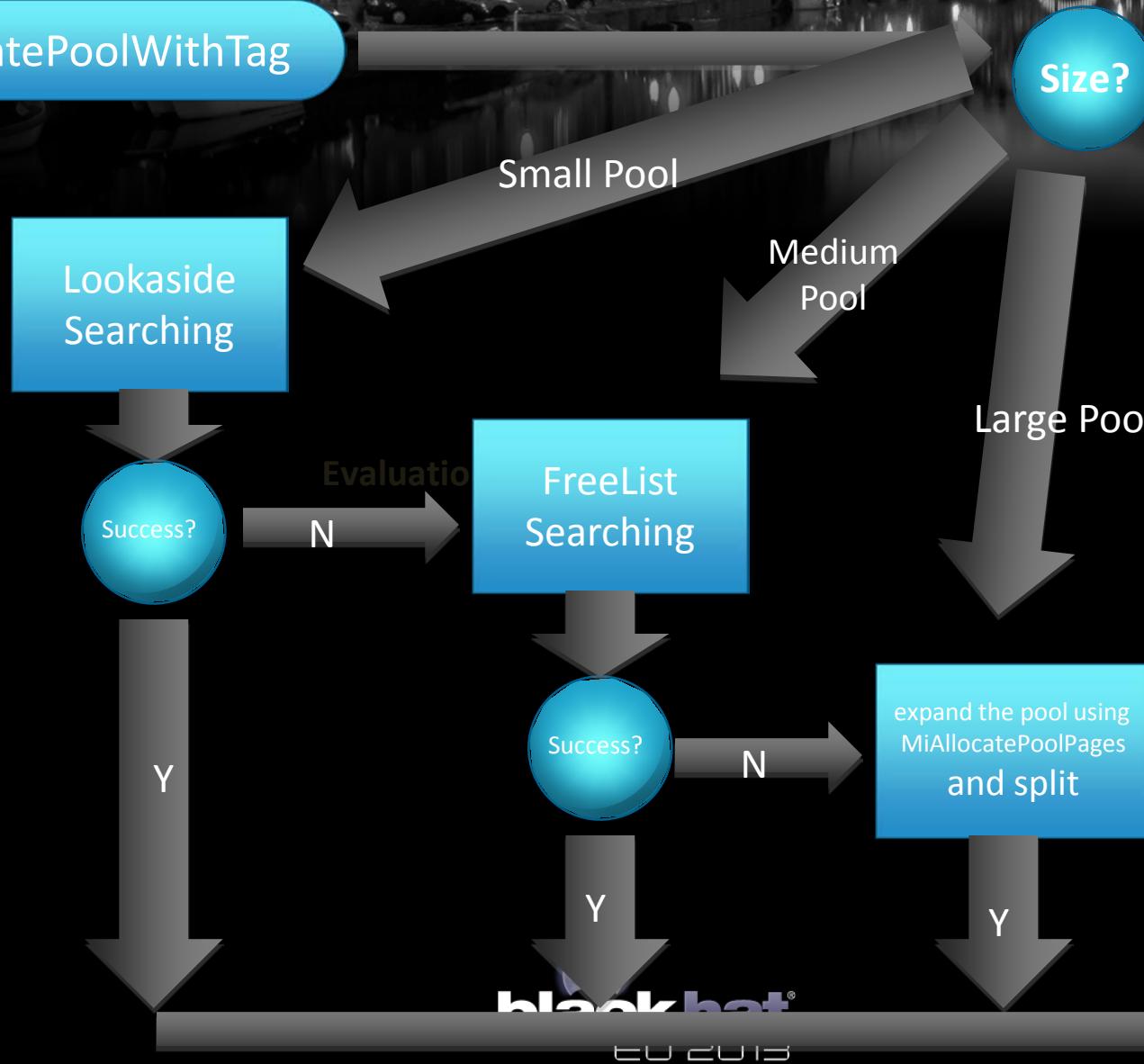
Demo of this section

0x02:

Implementation in Kernel Pool

Allocation Algorithm pre-view

ExAllocatePoolWithTag





Prerequisites

- Allocate Buffer of Arbitrary Size
- Free Buffer of Arbitrary Size
- Control Allocations and Frees using user code.

Example Alloc Proxy

Alloc (paged)

```
HANDLE UserAlloc(int size){  
    HANDLE LinkHandle;  
    std::wstring s((size - 2) / 2, 'a');  
    UNICODE_STRING TargetName;  
    MyRtlInitUnicodeString (&TargetName, s.c_str());  
    OBJECT_ATTRIBUTES Test1;  
    InitializeObjectAttributes(&Test1, NULL, 0, NULL, NULL);  
  
    int Status = MyCreateSymbolicLinkObject(&LinkHandle,  
                                           1,  
                                           &Test1,  
                                           &TargetName);  
  
    return LinkHandle;  
}
```

Example Free Proxy

Free

```
void UserFree(HANDLE Handle){  
    if (Handle){  
        CloseHandle(Handle);  
    }  
}
```

Massage the Kernel Pool

ExAllocatePoolWithTag()

When FreeList search failed, allocation will come from a new page.

```
82928443 bf00100000    mov    edi,1000h  
82928448 57            push   edi  
82928449 ff742424      push   dword ptr [esp+24h]  
8292844d e8b3ebffff    call   nt!MiAllocatePoolPages (82927005)
```

As 1000h is hard coded which leads to allocation aligned by 0x1000
(Paged , NonPaged, NonPagedNX,)

Kernel Virtual Address Space Allocation

nt!MiAllocatePoolPages

-- RtlFindClearBitsAndSet

-- MiObtainSystemVa

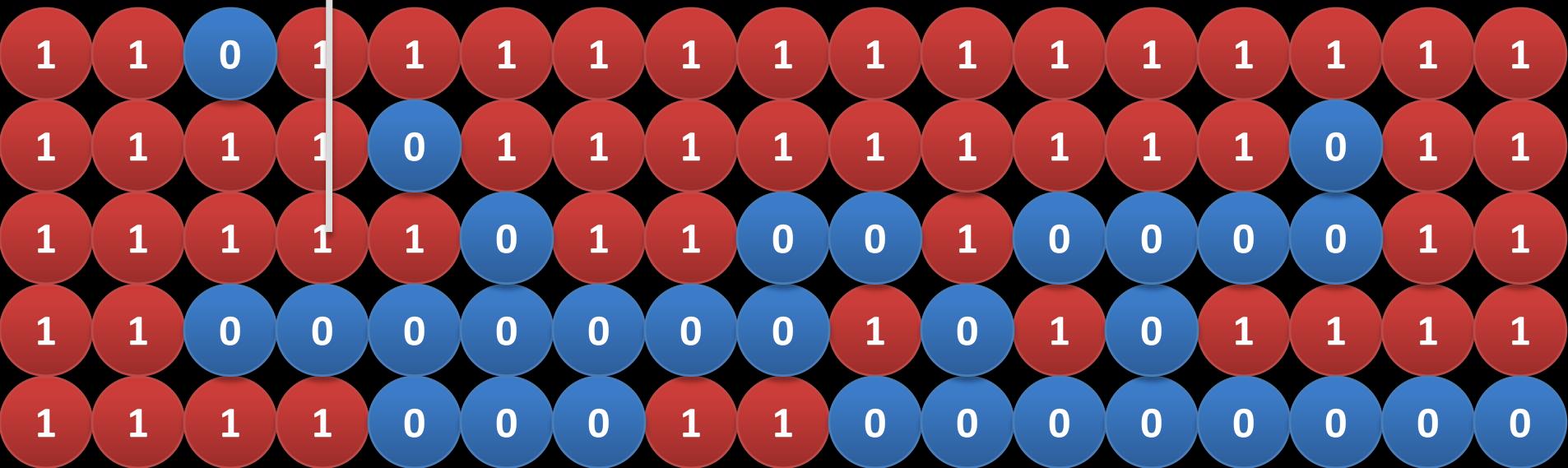
- kd> dt ntkrpamp!_RTL_BITMAP 827a1194
 - +0x000 SizeOfBitMap : 0x7fc00
 - +0x004 Buffer : 0x80731000 -> 0xffffffff

Kernel Pool Layout and Bitmap

0 : Free

1 : Used

Current Index

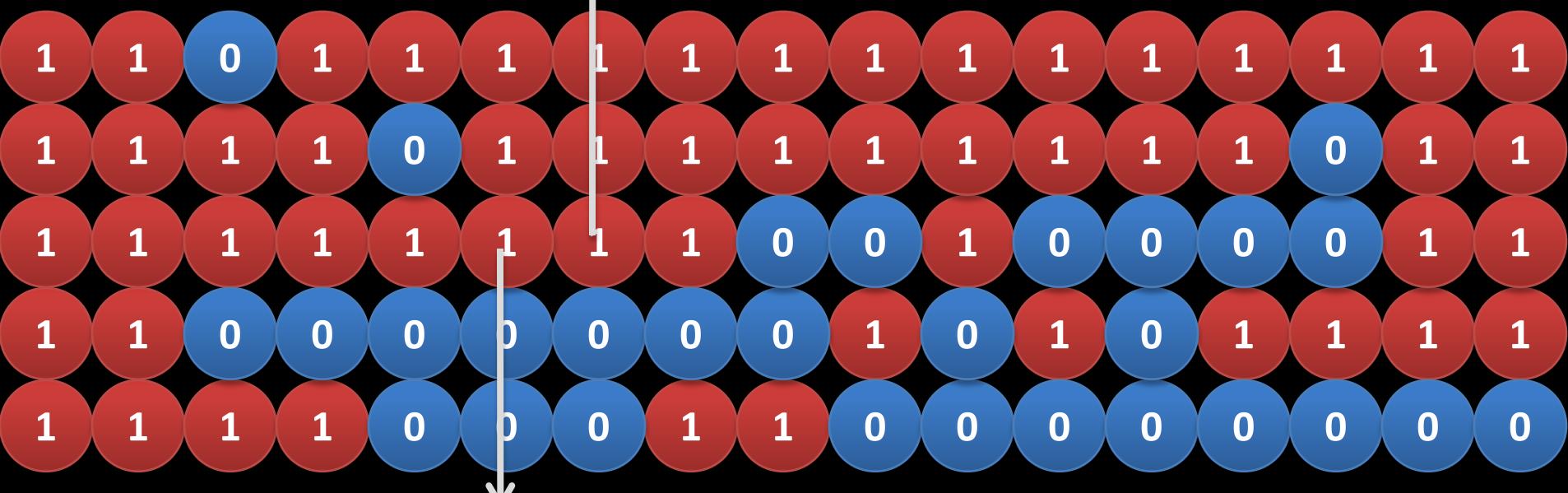


Request For 1 block

0 : Free

1 : Used

Current Index

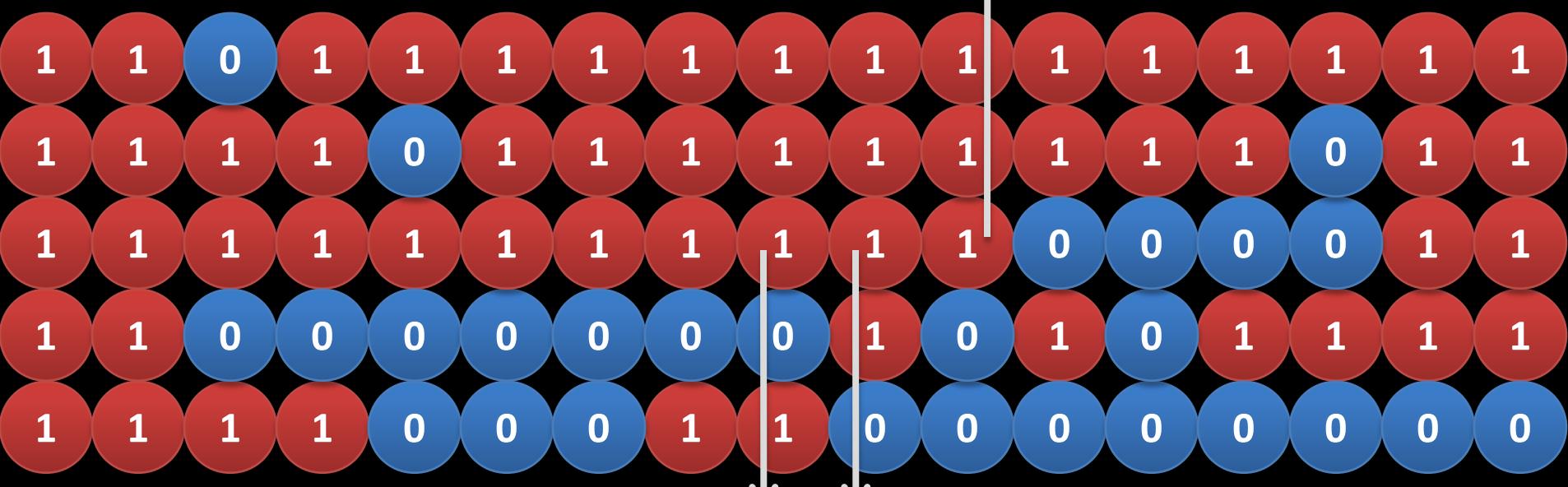


nt!RtlFindClearBitsAndSet

Request For 2 blocks

0 : Free

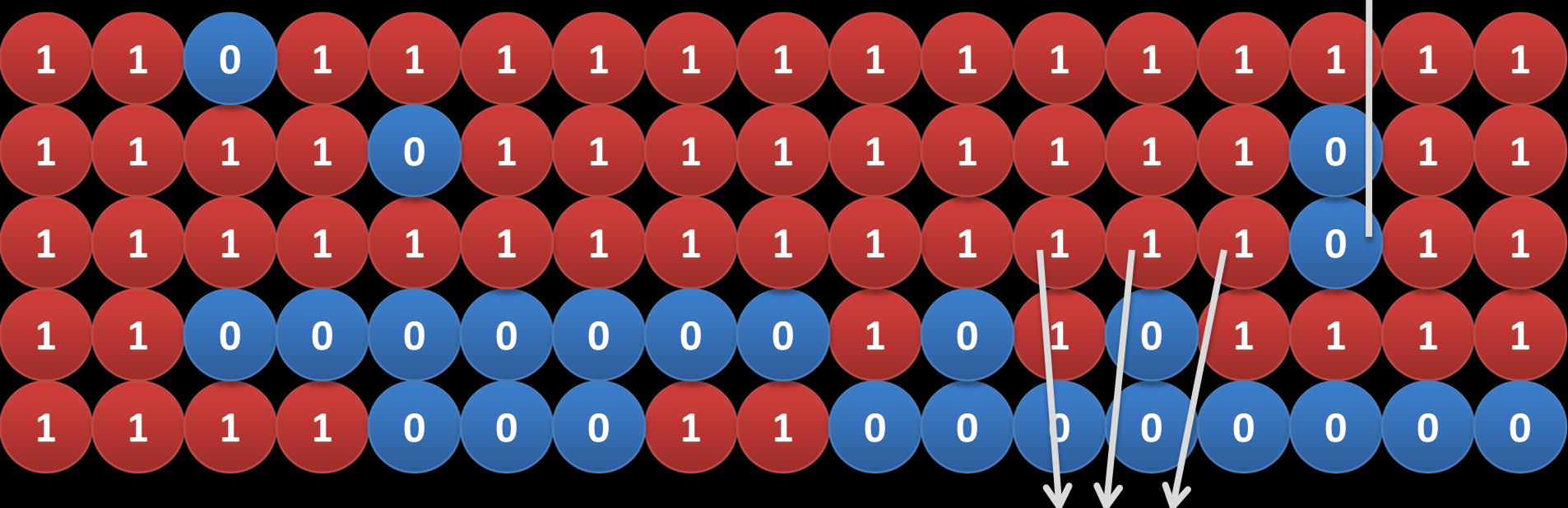
1 : Used



Request For 3 blocks

0 : Free

1 : Used



nt!RtlFindClearBitsAndSet

If all searches failed

0 : Free

1 : Used

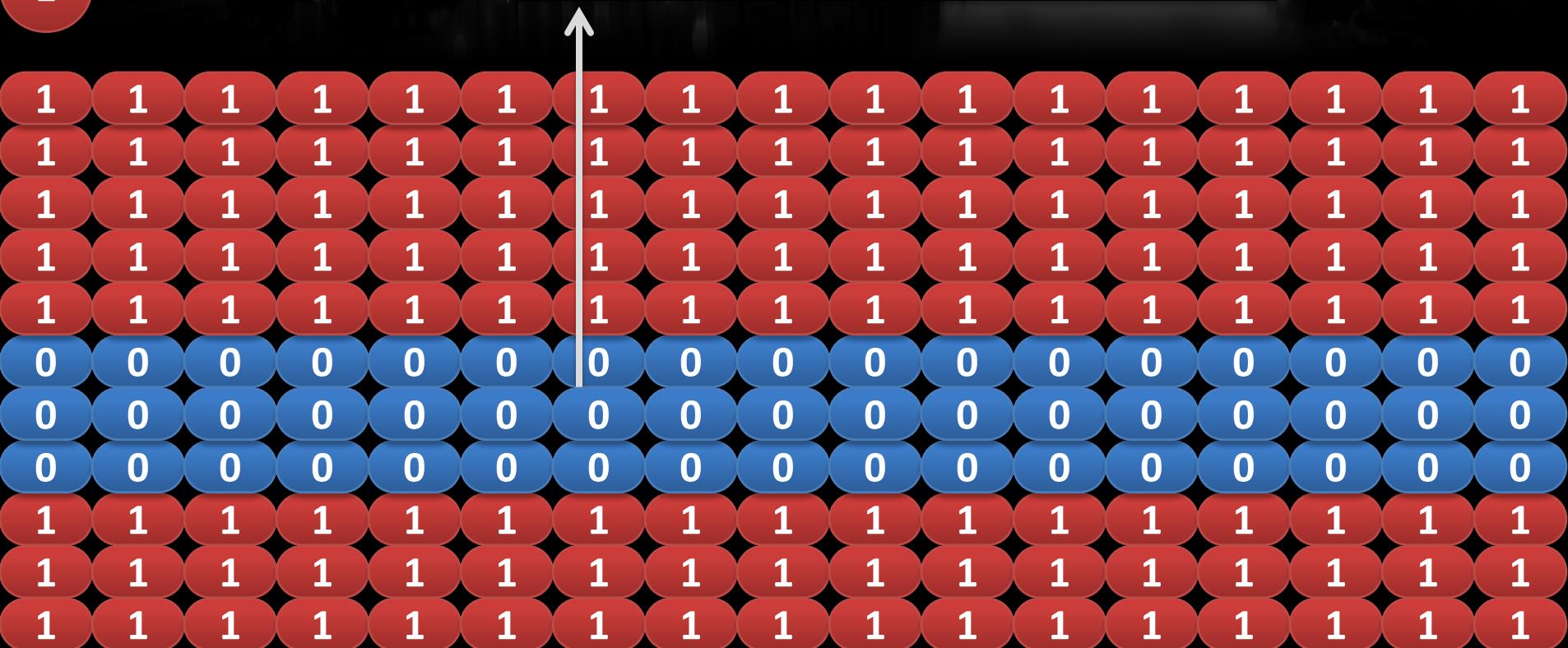
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Kernel VA dynamic allocate will taken (32bit)

0 : Free

1 : Used

MiObtainSystemVa is used to dynamically
allocate VA range



Interesting picking sequence

An empty page:

0x1000

Interesting picking sequence

1st allocation picked from front:

0x1000

0x808

0x7F8

Interesting picking sequence

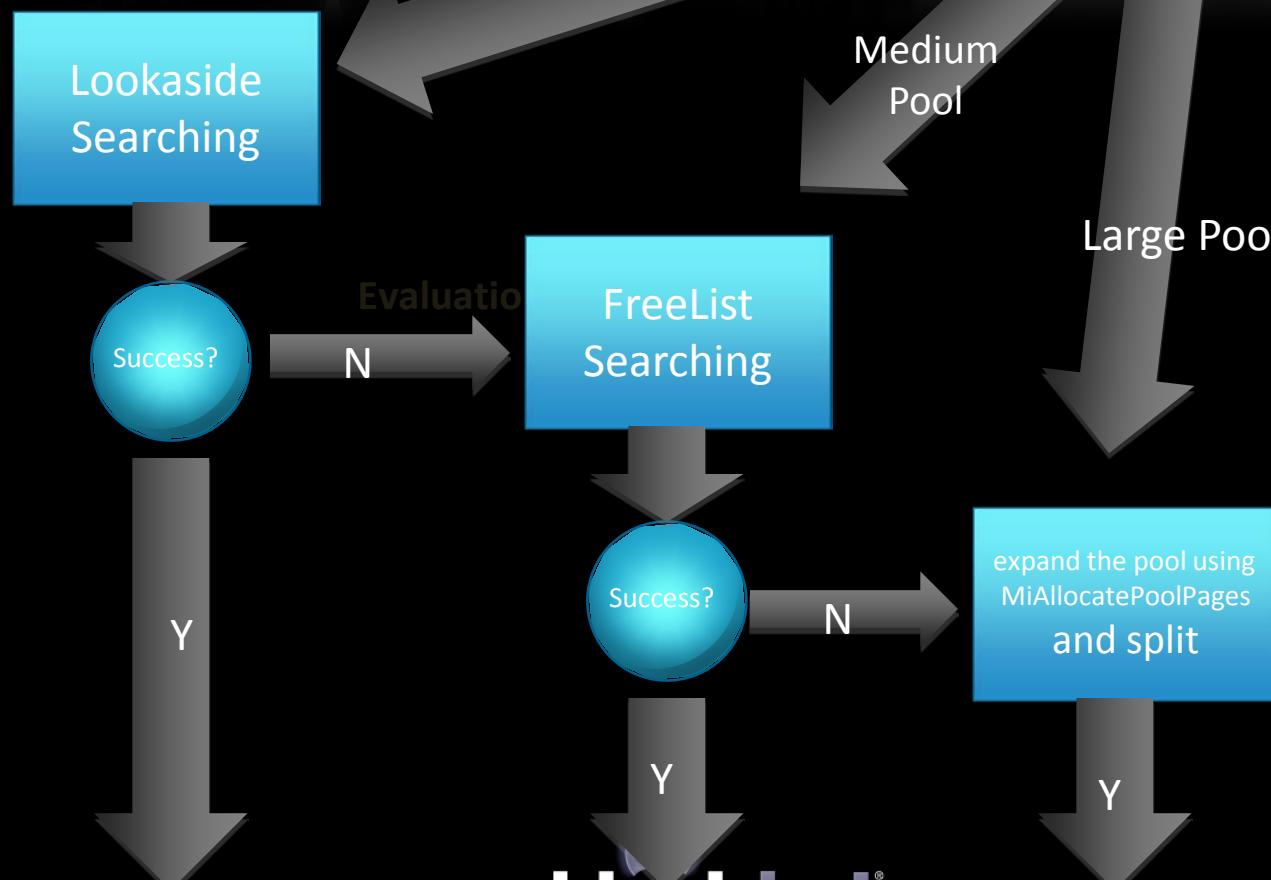
2nd allocation picked from end:



Our controlled way (small)

ExAllocatePoolWithTag

Size?



Our controlled way (small)

ExAllocatePoolWithTag

Size?

Lookaside
Searching

FreeList
Searching

Success?

Evaluation
N

Large Pool

Medium
Pool

Success?

N

expand the pool using
MiAllocatePoolPages
and split

Y

Y

Y

Y

Our controlled way (small)

ExAllocatePoolWithTag

Size?

Lookaside
Searching

Small Pool

Medium
Pool

Large Pool

Success?

Evaluation
N

FreeList
Searching

Success?

N

expand the pool using
MiAllocatePoolPages
and split

Y

Y

Y

Return

Our controlled way (small)

ExAllocatePoolWithTag

Lookaside
Searching

Success?

Y

Evaluation
N

FreeList
Searching

Success?

Y

Small Pool

Medium

Size?

```
P RtIpFindEntry();  
RtIpHeapRemoveListEntry();  
// FreeListEntry is controlled
```

```
if (CommitSize < FreeListEntry ->Size){  
// Force the CommitSize smaller than  
// the FreeListEntry ->Size  
    RtIpCreateSplitBlock();  
}  
return Chunk
```

Our controlled way (small)

ExAllocatePoolWithTag

Size?

Lookaside
Searching

Small Pool

Medium
Pool

Large Pool

Success?

Evaluation
N

FreeList
Searching

Success?

N

expand the pool using
MiAllocatePoolPages
and split

Split Chunks

Y

Y

Or this way (Medium)

ExAllocatePoolWithTag

Size?

Lookaside
Searching

Small Pool

Medium
Pool

Large Pool

FreeList
Searching

Success?

Evaluation
N

Success?

N

expand the pool using
MiAllocatePoolPages
and split

Split Chunks

Y

Return

What about size > 0xFF0?

Daniel: Yes it will. There's always a way out...

-Quotes from Stargate SG-1 "Abyss"

- A: if (size_t < 0x400)
- B: if ((size_t >= 0x400) & (size_t < 0x800))
- C: if ((size_t >= 0x800) & (size_t < 0xFF0))
- D: if (size_t >= 0xFF0)

A: if (size_t < 0x400)

Make holes on size 0x1000 chopping board



B: if ((size_t < 0x400) & (size_t < 0x800))

Make holes on size 0x2000 chopping board

0x1000

0x1000

0x1010

0x9F8

0xC0

0x538

C: if ((size_t > 0x800) & (size_t < 0xFF0))

Make holes on size 0x3000 chopping board



D: if (size_t > 0xFF0)

Vulnerable buf will be allocated by MiAllocatePoolPages directly

0x1000

0x1000

0x1010

0xC0

0xF30



Demo of this section

2.01:

*Windows Objects in
Kernel Vulnerability Exploitation*

Exploitation in Windows 7 (Bonus)

- kd> dt nt!_OBJECT_HEADER
 - +0x000 PointerCount : Int4B
 - +0x004 HandleCount : Int4B
 - +0x004 NextToFree : Ptr32 Void
 - +0x008 Lock : _EX_PUSH_LOCK
 - +0x00c TypeIndex : Uchar** // used to be a Ptr in XP
 - +0x00d TraceFlags : UChar
 - +0x00e InfoMask : UChar
 - +0x00f Flags : UChar
 - +0x010 ObjectCreateInfo : Ptr32 _OBJECT_CREATE_INFORMATION
 - +0x010 QuotaBlockCharged : Ptr32 Void
 - +0x014 SecurityDescriptor : Ptr32 Void
 - +0x018 Body : _QUAD**

Exploitation in Windows 7 (Bonus)

*0x01: InitTrampoline:
Mapping VA 0x0 through NtAllocateVirtualMemory*

0x02: Modify TypeIndex

Then..

0x03: Jump into shellcode when CloseHandle()

```
; Attributes: bp-based frame
; int __stdcall ObpCloseHandleTableEntry(int, int, int, ULONG_PTR BugCheckParameter1, int, char)
_ObpCloseHandleTableEntry@24 proc near

var_25= byte ptr -25h
var_24= dword ptr -24h
var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_18= dword ptr -18h
arg_0= dword ptr 8
arg_4= dword ptr 0Ch
arg_8= dword ptr 10h
BugCheckParameter1= dword ptr 14h
arg_10= dword ptr 18h
arg_14= byte ptr 1Ch

mov    edi, edi
push   ebp
mov    ebp, esp
and   esp, 0FFFFFFF8h
sub   esp, 2Ch
mov    eax, [ebp+arg_4]
push   ebx
push   esi
mov    esi, [eax]
and   esi, 0FFFFFFF8h
movzx  ecx, byte ptr [esi+8Ch]
mov    ebx, _ObTypeIndexTable[ecx*4]
push   edi
mov    edi, large fs:124h
cmp   dword ptr [ebx+74h], 0
lea    ecx, [esi+18h]
mov    [esp+38h+var_24], ecx
mov    [esp+38h+var_25], 0
jz    loc_82881C30

    mov    ecx, large fs:124h
    mov    eax, [ebp+arg_8]
    cmp    [ecx+50h], eax
    jz    short loc_82881BB8

lea    ecx, [esp+38h+var_18]
push   ecx      ; int
push   eax      ; BugCheckParameter1
call   KeStackAttachProcess@8 ; KeStackAttachProcess(x,x)
mov    [esp+38h+var_25], 1

loc_82881BB8:
push   [ebp+arg_18]
push   [ebp+BugCheckParameter1]
push   [esp+40h+var_24]
push   [ebp+arg_8]
call   dword ptr [ebx+74h]
test   al, al
jnz   short loc_82881C2D
```

mov ebx, _ObTypeIndexTable[ecx*4]
// ecx is TypeIndex

...

call dword ptr [ebx+74h]

Exploitation in Windows 8 (Mateusz 'j00ru' Jurczyk way)

- kd> dt nt!_KTIMER 84247538
 - +0x000 Header : _DISPATCHER_HEADER
 - +0x010 DueTime : _ULARGE_INTEGER
0x4`9b8e6360
 - +0x018 TimerListEntry : _LIST_ENTRY [
0x85360160 - 0x82765ce4]
 - +0x020 Dpc : 0x84247590 _KDPC**
 - +0x024 Period : 0x7d0

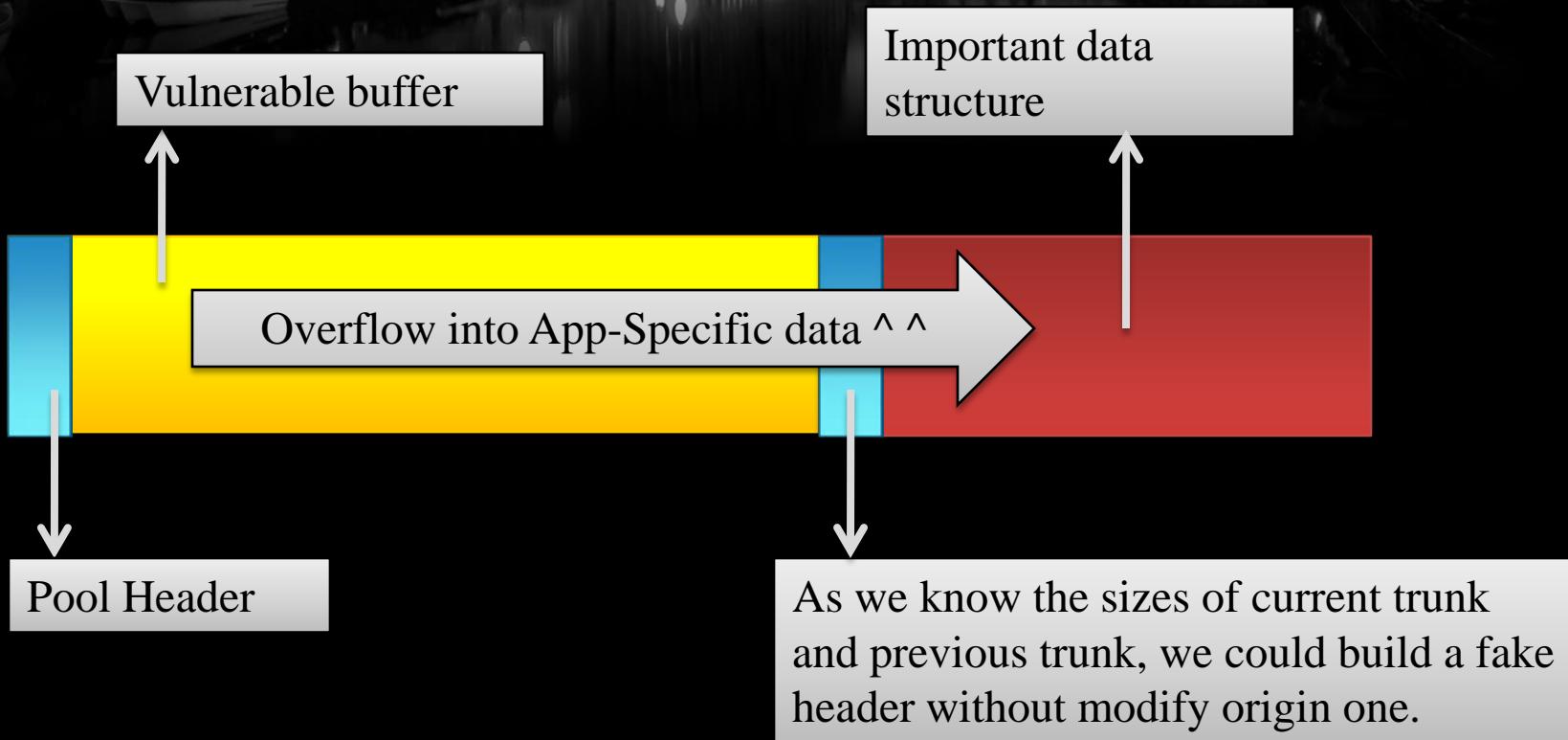
Exploitation in Windows 8 (Mateusz 'j00ru' Jurczyk way)

- kd> dt nt!_KDPC
 - +0x000 Type : UChar
 - +0x001 Importance : UChar
 - +0x002 Number : UInt2B
 - +0x004 DpcListEntry : _LIST_ENTRY
 - +0x00c DeferredRoutine : Ptr32 void**
 - +0x010 DeferredContext : Ptr32 Void
 - +0x014 SystemArgument1 : Ptr32 Void
 - +0x018 SystemArgument2 : Ptr32 Void
 - +0x01c DpcData : Ptr32 Void

2.02:

Practical exploiting kernel pool Overflow / Corruption

Exploiting Kernel Pool Overflow / Corruption



2.03:

Practical Exploiting

write-what-where vulnerability

Place object at a predictable address

0x9e51e000

(a relative high address, supposed be reached only through heap spray)

0x1000

Place object at a predictable address

0x9e51e000



0x1000

0x900

0x700

0x9e51e900



Place object at a predictable address

0x9e51e000



0x1000

0x900

0x700

0x900

0x48

0x6B8

0x9e51e900 + 0x1c: TypeIndex

Demo

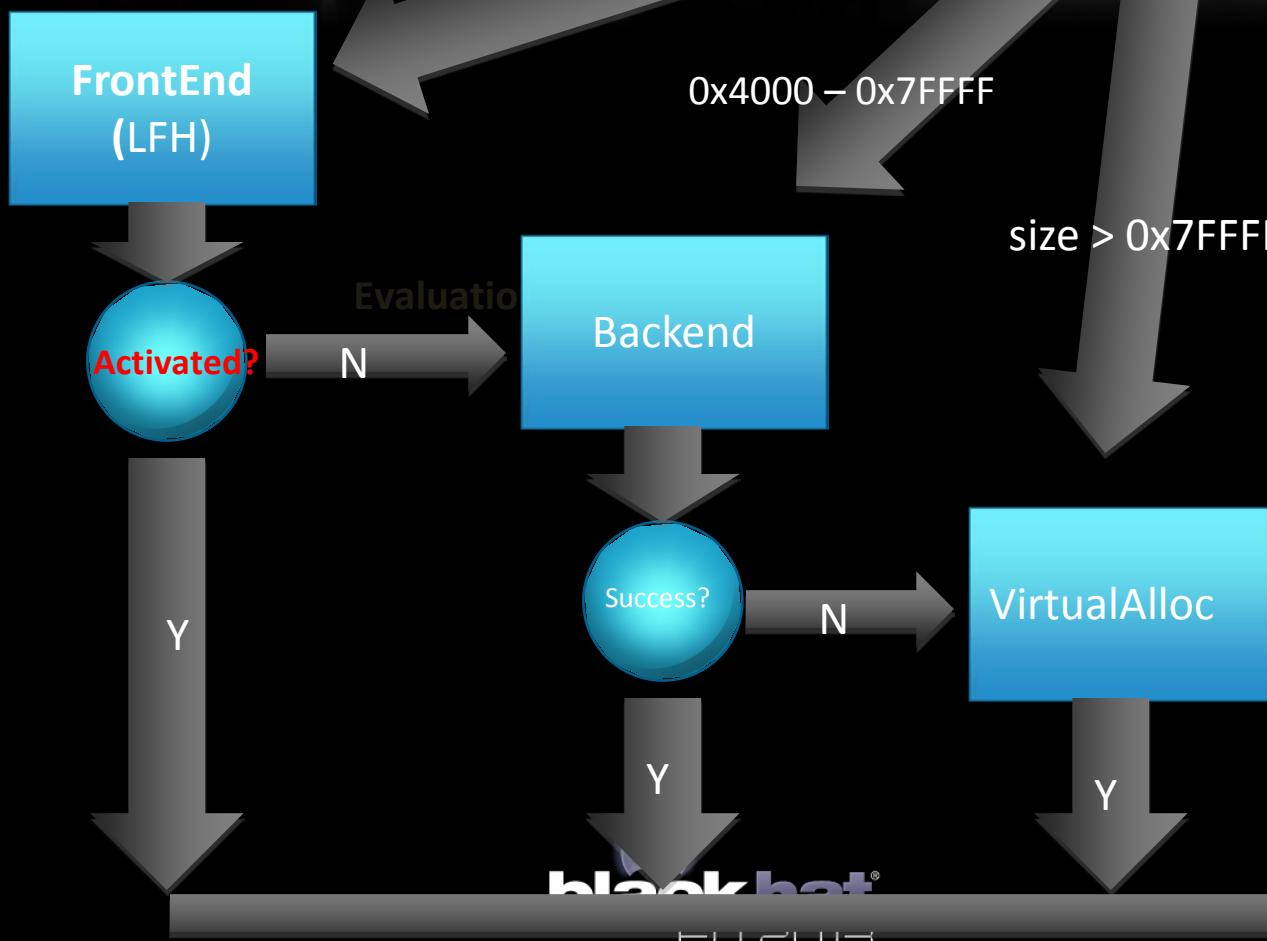
0x03:

Implementation in User Heap

Allocation Algorithm pre-view

HeapAlloc(x, x, size)

Size?



3.01:

Practical Attacking

_HEAP_USERDATA_HEADER

HEAP USERDATA HEADER

- Idea brought by Chris Valasek
- Chunk = UserBlocks + RandIndex * BlockStride + FirstAllocationOffset



Two Challenges

- 18 times of allocations will trigger LFH
- 400 times of allocations will trigger guard pages.



GP

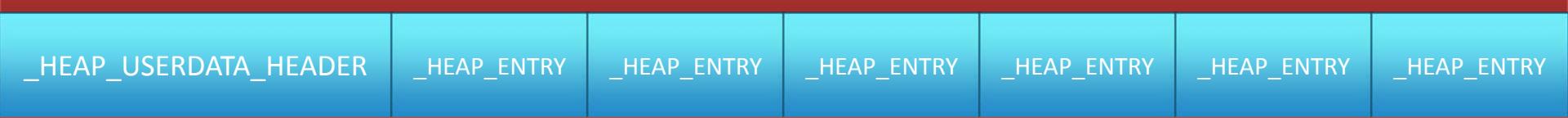
LFH

Vul buffer

LFH & Guard Pages



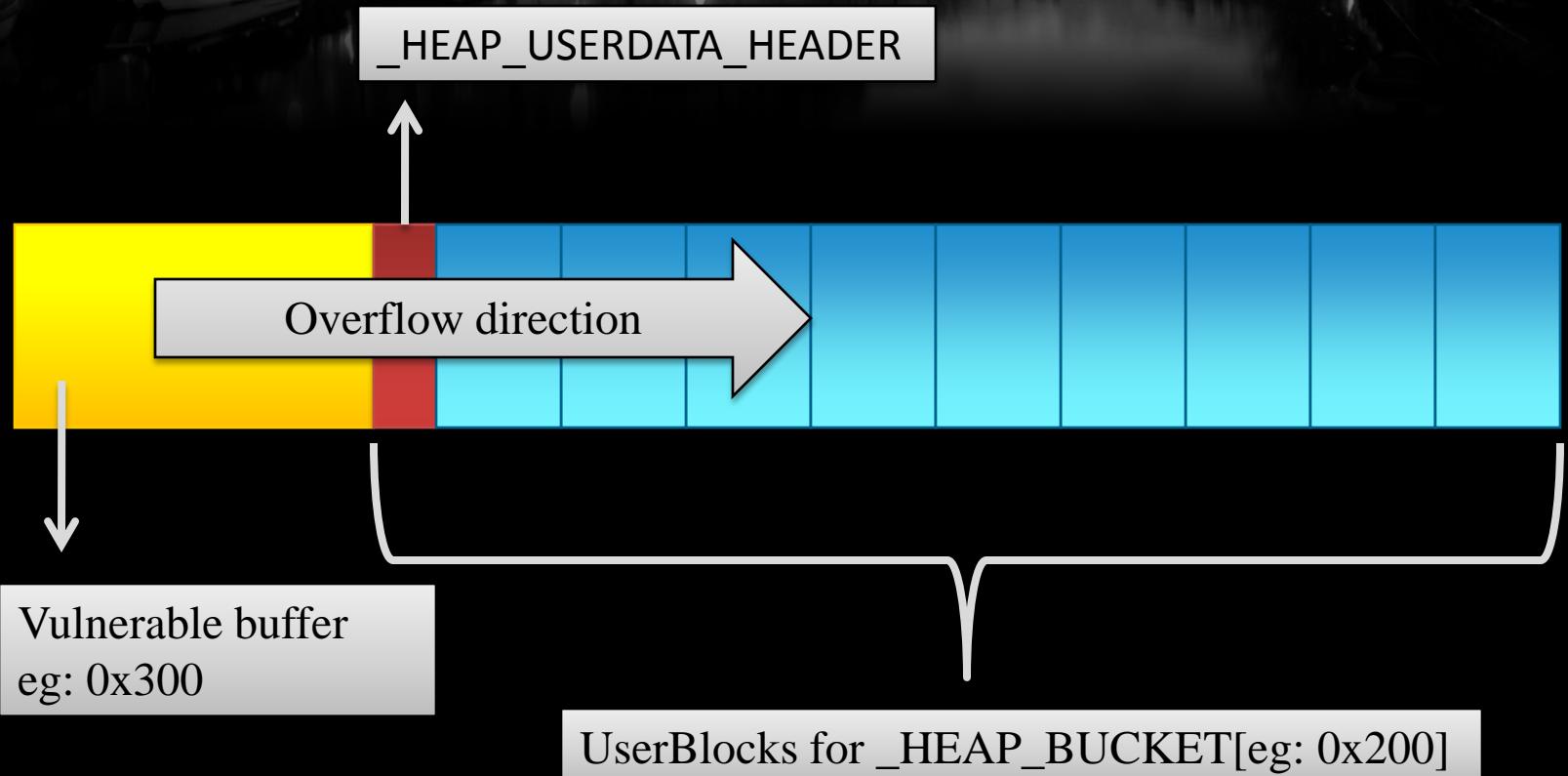
GP – PAGE_NOACCESS



GP – PAGE_NOACCESS



The target



to position the vulnerable buffer just BEFORE an important structure.
Like: `_HEAP_USERDATA_HEADER` structure



Mandatory Search in Action

- Defragment using chunk 0x4000 - 0xFFFF.
- Freeing (0x70100) --> Allocating (0x70000)
Could make 0x100 hole.
Hey, get out of my way -- LFH
- The size of UserBlocks (total size) is fixed.

Taken

Noise

Free

0x01: Defragment

0x8000

0x8000

0x8000

0x8000

0x8000

Taken

Noise

Free

0x02: Freeing

0x8000

0x8000

0x8000

0x8000

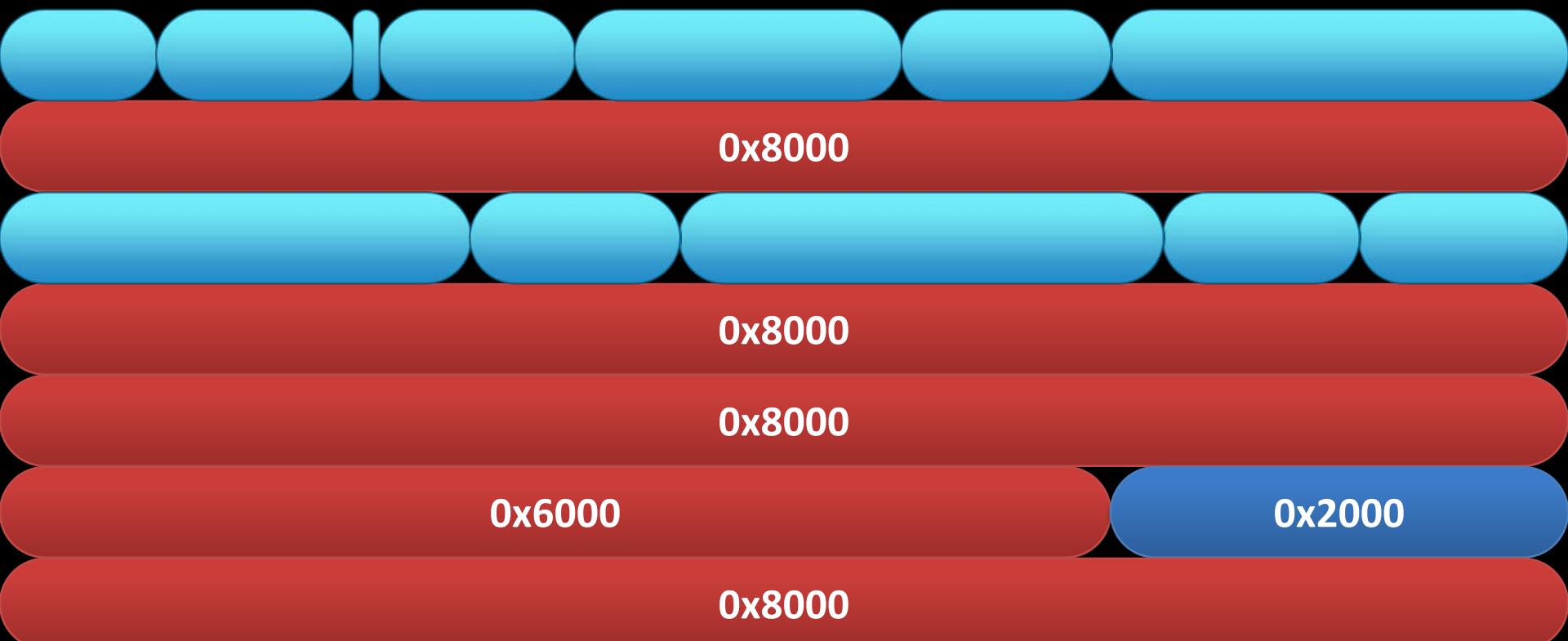
0x8000

Taken

Noise

Free

0x03: Alloc 0x6000 block and make 0x2000 hole

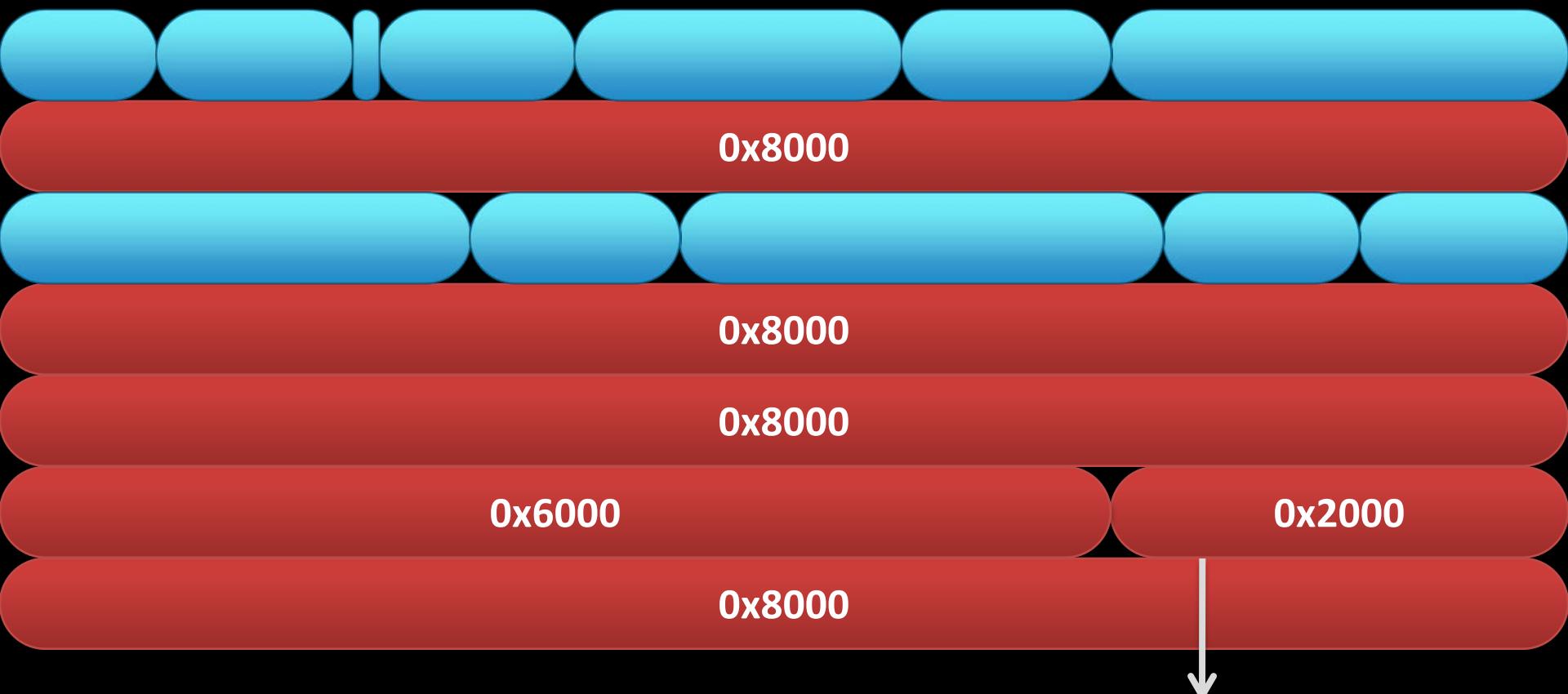


Taken

Noise

Free

0x04: Trigger LFH (0x200)



Taken

LFH

Free

Take a closer look at

0x6000

_HEAP_USERDATA_HEADER		_HEAP_ENTRY						
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY



UserBlocks for _HEAP_BUCKET[0x200]

Taken

LFH

Free

Free 0x6000 block

0x6000

_HEAP_USERDATA_HEADER		_HEAP_ENTRY						
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY

Taken

Free

Alloc 0x5D00 block and make 0x300 hole

0x6000 – 0x300

0x300

_HEAP_USERDATA_HEADER		_HEAP_ENTRY						
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY

Taken

Vul buffer

Free

Alloc vulnerable buffer

0x6000 – 0x300

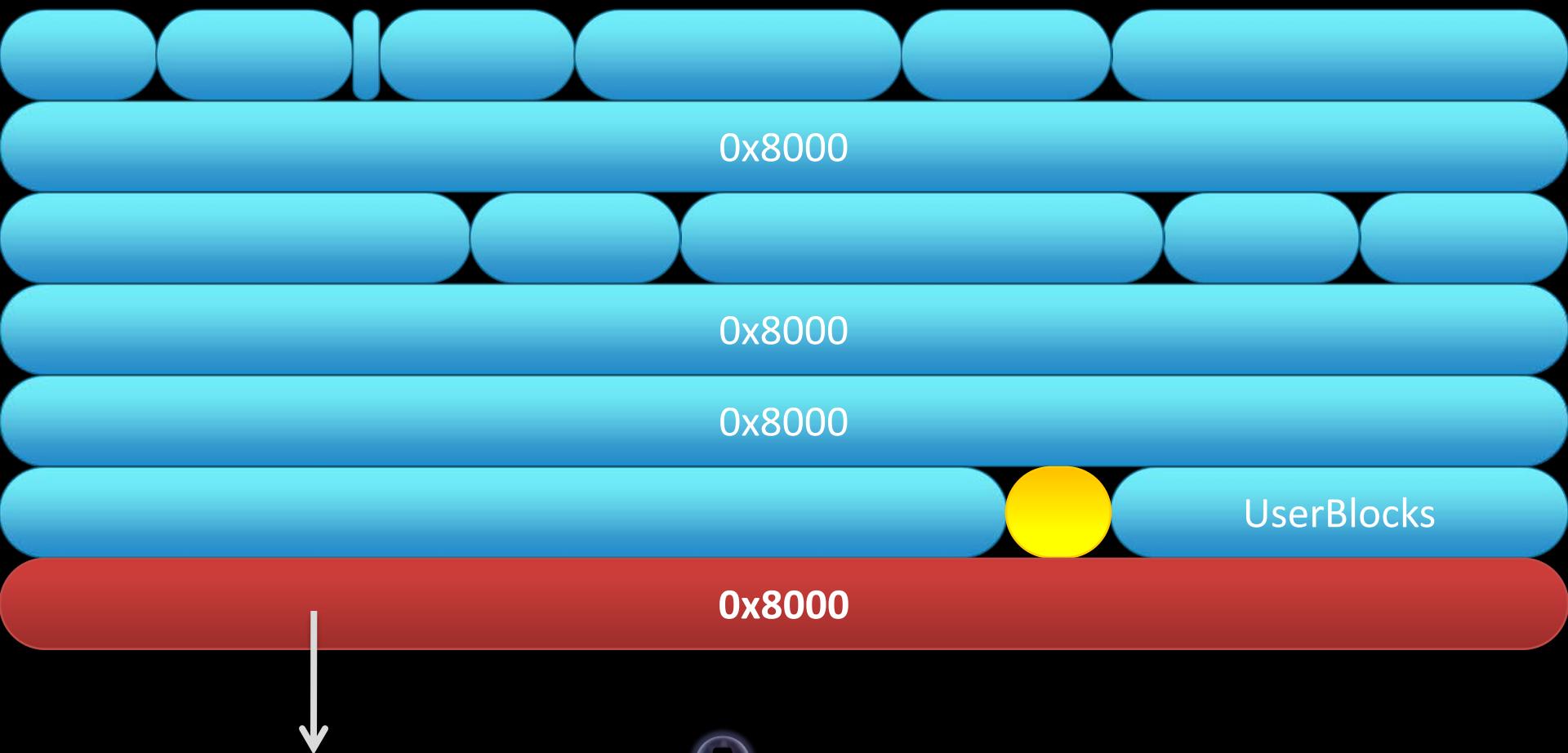
0x300

_HEAP_USERDATA_HEADER		_HEAP_ENTRY						
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY
_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY	_HEAP_ENTRY

Vul buffer

Controlled

Future allocation will get controlled after overflow



Applicable circumstance (Prerequisites)

- The LFH of the certain bin size has not been activated by the time of allocation.
(no 16 consecutive allocations of the vulnerable buffer's size)
- Allocate Buffer of Arbitrary Size w/ Arbitrary Content
- Free Buffer of Arbitrary Size
- Programmatic Control of Allocations and Frees



The exploitation process:

Step 0: Figure out the vulnerability

Step 1: Heap Feng Shui.

Step 2: Trigger the overflow, modify "FirstAllocationOffset"

Step 3: Allocate new objects with proper size.

Step 4: Modify new object's content.

Step 5: Control the EIP.

3.02:

Practical Heap Determining in IE 10

Conclusion

Questions?