Windows 10 Pool Party

BAYET Corentin

ARMATURE TECHNOLOGIES

1.1

2017

Windows 10 Pool Party

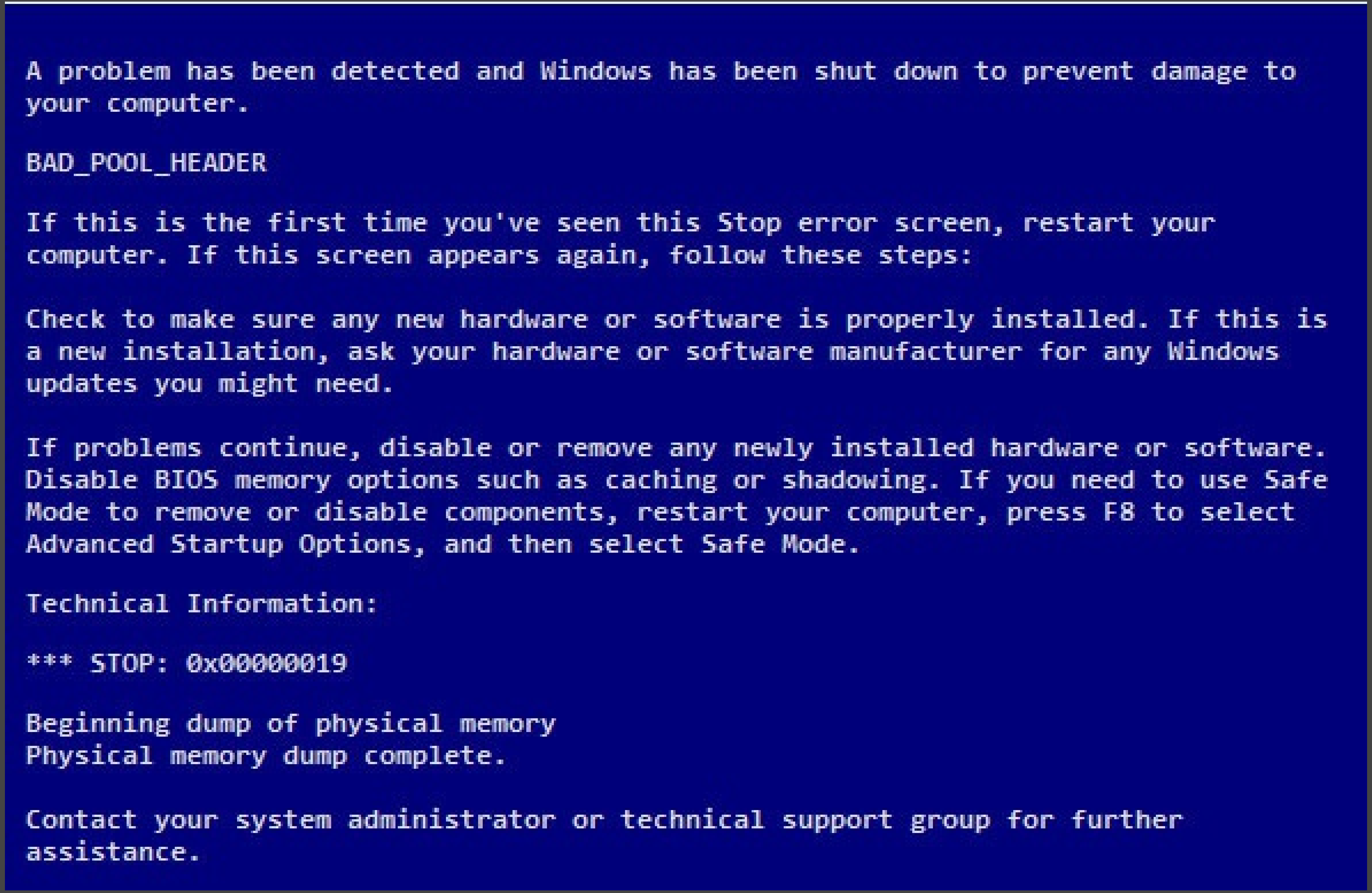
What we WILL talk about

- Exploitation in the NonPagedPool
- Exploitation at medium integrity level
- Attacking drivers and IOCTLs
- Tools and methods to attack pool

What we WONT talk about

- Win32k.sys, GDI / USER objects
- Exploitation at low integrity level

First crash



```
A problem has been detected and Windows has been shut down to prevent damage to
your computer.

BAD_POOL_HEADER

If this is the first time you've seen this Stop error screen, restart your
computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is
a new installation, ask your hardware or software manufacturer for any Windows
updates you might need.

If problems continue, disable or remove any newly installed hardware or software.
Disable BIOS memory options such as caching or shadowing. If you need to use Safe
Mode to remove or disable components, restart your computer, press F8 to select
Advanced Startup Options, and then select Safe Mode.

Technical Information:

*** STOP: 0x00000019

Beginning dump of physical memory
Physical memory dump complete.

Contact your system administrator or technical support group for further
assistance.
```

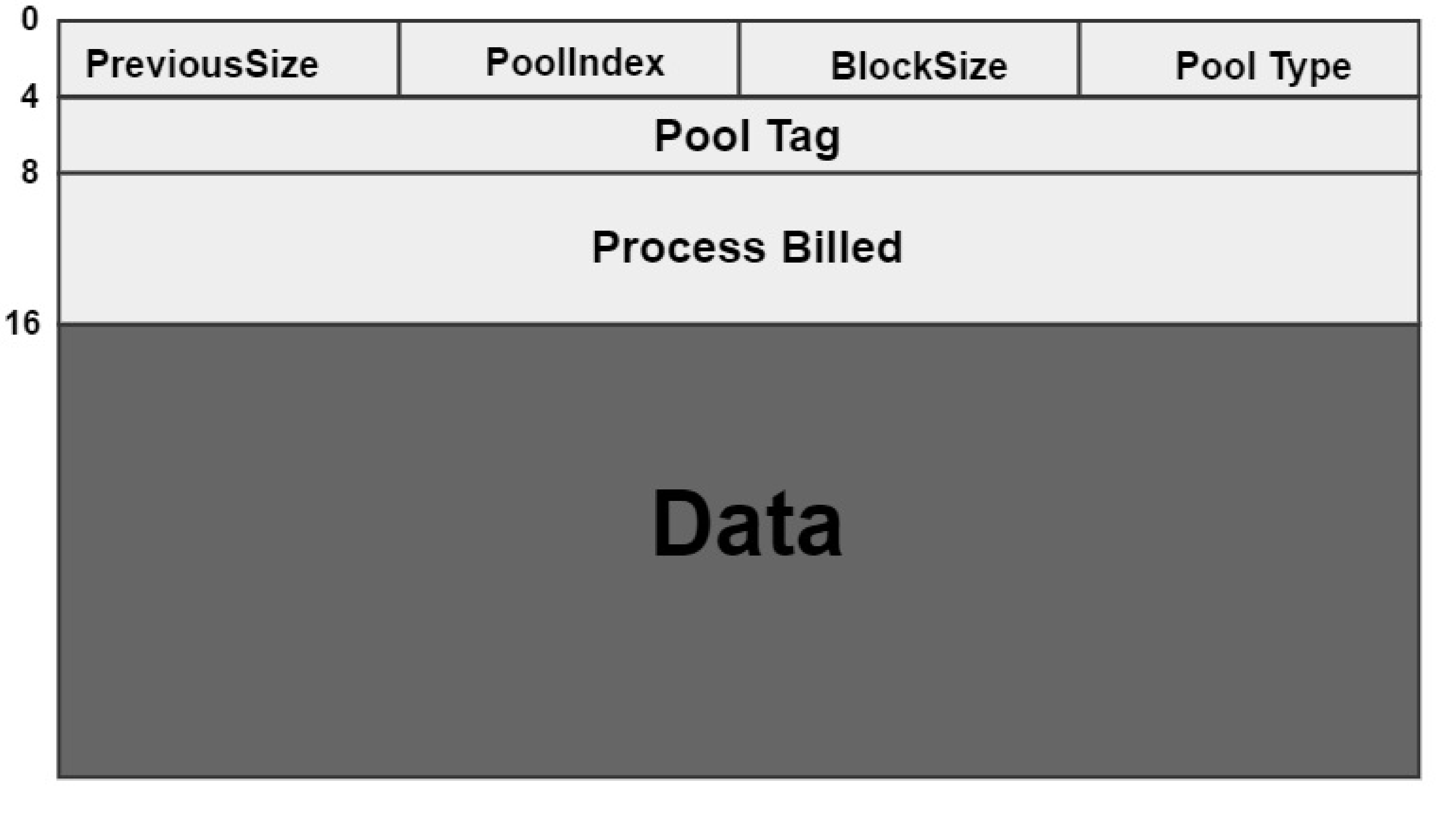
What is the kernel pool ?

- Place for every allocation in the windows kernel
- Common for every drivers
- Specific allocator and structures
- Several types:
  - NonPagedPool
  - PagedPool
  - ....

Basically, a list of pages fragmented in chunks !

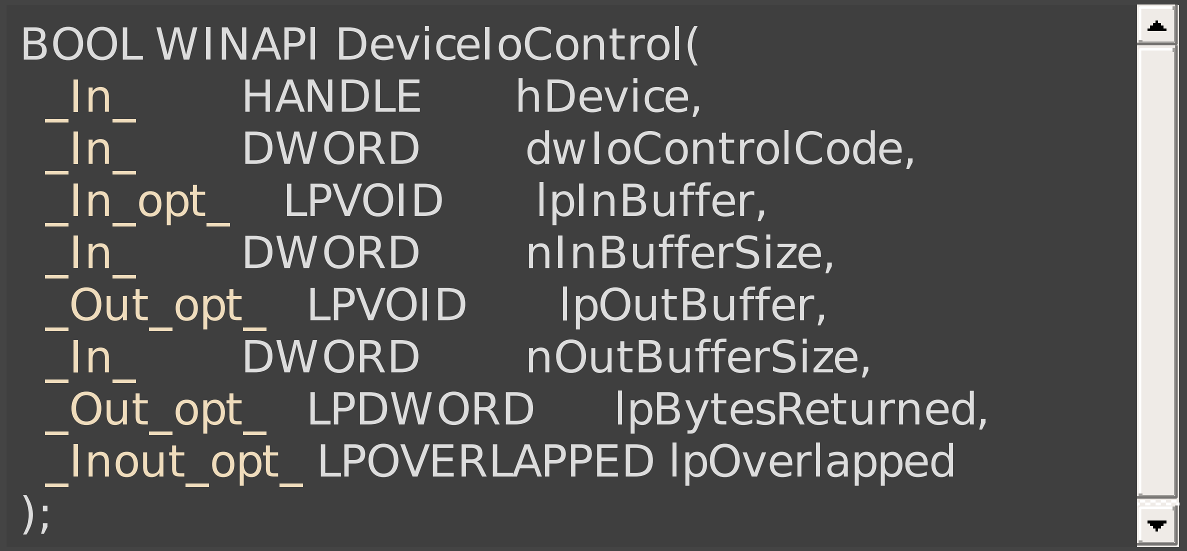
ARMATURE
TECHNOLOGIES

A pool chunk


0
PreviousSize
PoolIndex
BlockSize
Pool Type
4
Pool Tag
8
Process Billed
16
Data

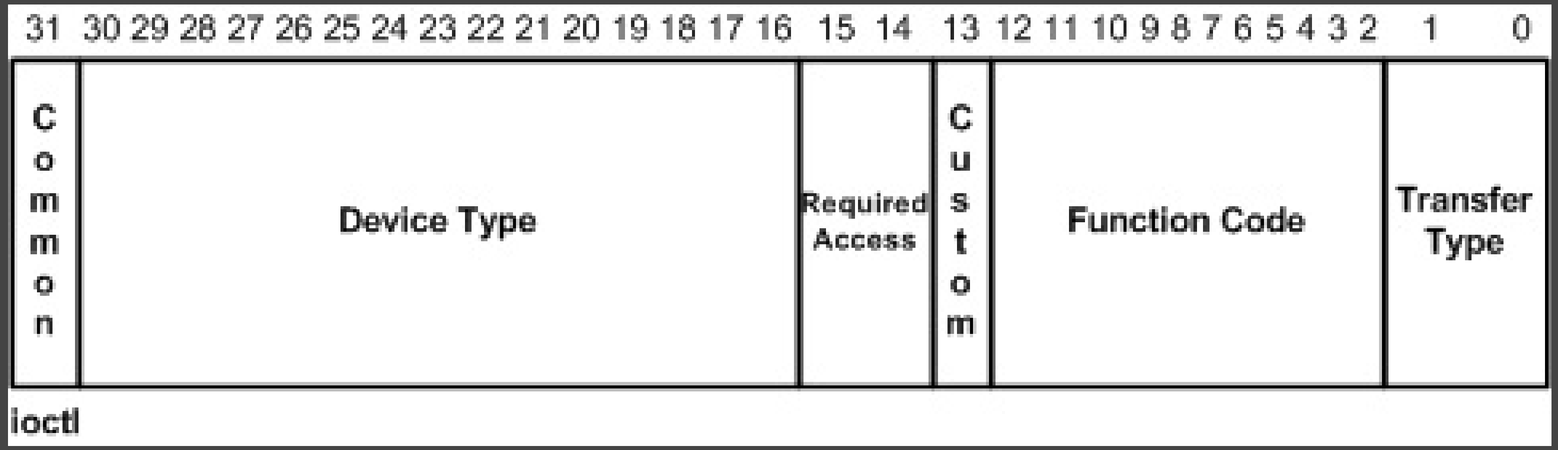
First crash

IOCTL: Input/Ouput Control



```
BOOL WINAPI DeviceIoControl(
  _In_        HANDLE       hDevice,
  _In_        DWORD        dwIoControlCode,
  _In_opt_    LPVOID       lpInBuffer,
  _In_        DWORD        nInBufferSize,
  _Out_opt_   LPVOID       lpOutBuffer,
  _In_        DWORD        nOutBufferSize,
  _Out_opt_   LPDWORD      lpBytesReturned,
  _Inout_opt_ LPOVERLAPPED lpOverlapped
);
```

I/O Control Code



31	30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14	13	12 11 10 9 8 7 6 5 4 3 2	1 0
Common	Device Type	Required Access	Custom	Function Code	Transfer Type

ioctl

First crash

METHOD_BUFFERED:

1. The I/O Manager allocates a buffer in the NonPaged Pool with the biggest size provided: it's the SystemBuffer
2. The I/O Manager copies the InputBuffer in the SystemBuffer and pass it to the driver
3. The driver handles the IOCTL, and writes the return in the SystemBuffer by overwriting the input. The driver must also tell to the I/O Manager how much he has written.
4. The I/O Manager copies the content of the SystemBuffer in the OutputBuffer using the size provided by the driver.

So we control the size of the buffer used for input and ouput in drivers... Great Attack Vector !

The vulnerability

About CVE-2017-6008

A memcpy is called with following arguments:

- **Dest:** The SystemBuffer (we control the size)
- **Src:** A full controlled buffer (from our Input Buffer)
- **Size:** the size of src

Classic **Buffer Overflow**... But in the NonPagedPool !

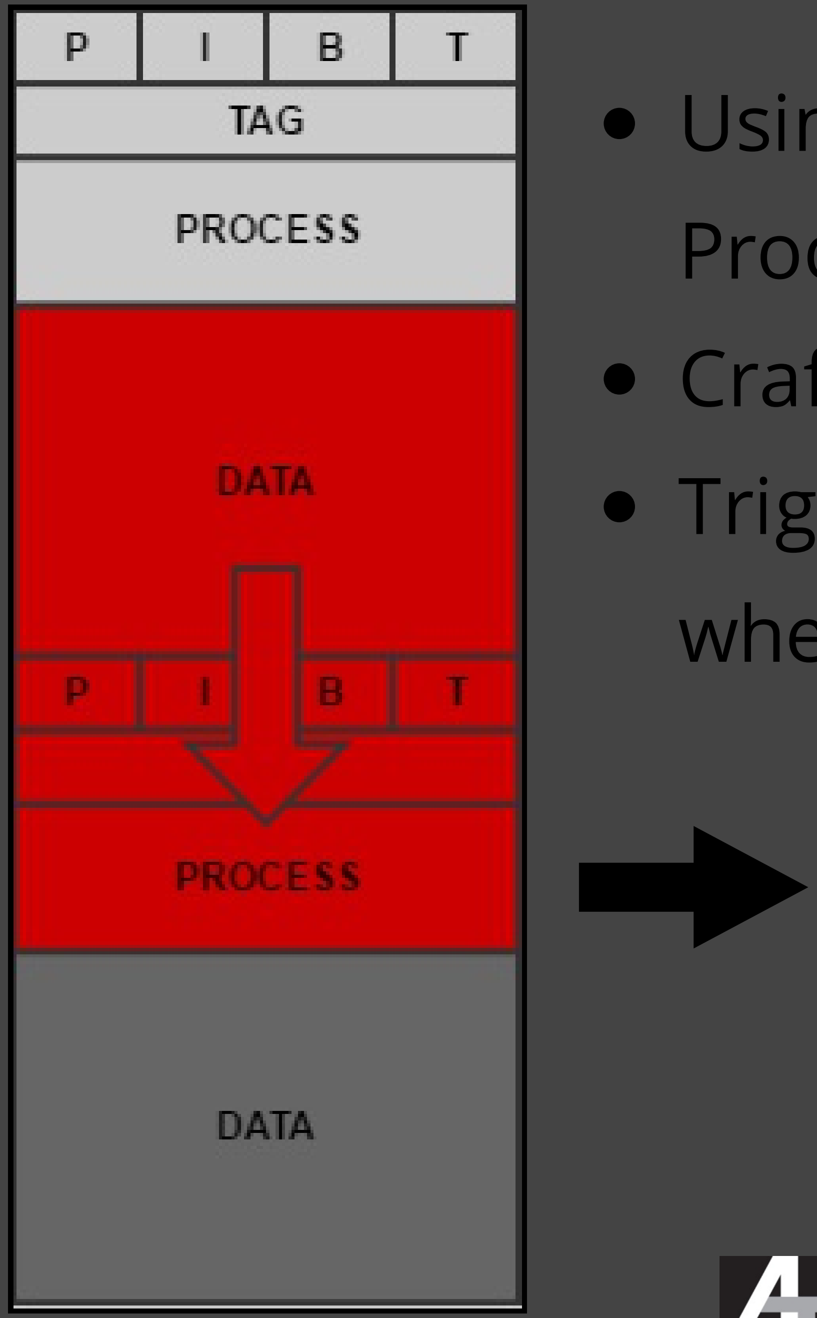
Pool History

Tarjei Mandt :

« Kernel Pool Exploitation on Windows 7 »

- **Deobfuscate Pool Internals**
- **Presents severals generic attacks**

Quota Process Pointer Overflow



- Using a pool buffer overflow to overwrite Process pointer
- Craft a fake EPROCESS structure
- Triggers an arbitrary decrementation when the overflowed chunk is free

Points to data controlled by attacker

ARMATURE TECHNOLOGIES

DEMO

History of the Pool

Windows 8 Introduced a lot of mitigations:

- REAL safe linking/unlinking
- Pool Index validation
- SMEP
- MIN_MAP_ADDR (reverted on windows 7 and vista x64)
- NonPagedPoolNx (DEP)

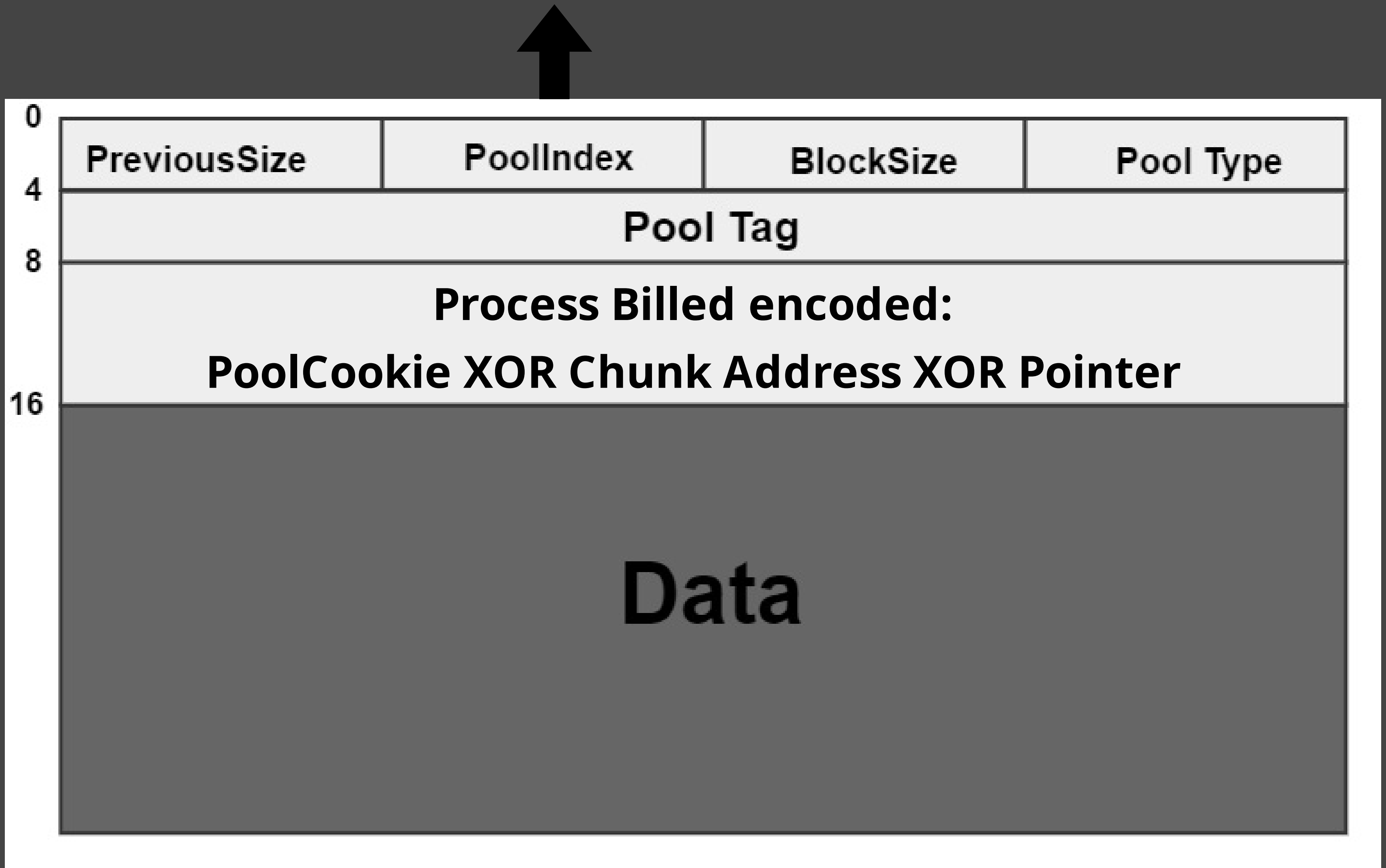
About the attack we used:

- Process Billed encoded with a cookie
- The free algorithms checks if the pointer is in kernel-land

ARMATURE TECHNOLOGIES

Nowadays Pool Chunk

Checked before use



Process Billed encoded:

PoolCookie XOR Chunk Address XOR Pointer

Today

- Exploiting vulnerabilities in the Pool is pretty hard
- No generic attacks

Goal: exploit the very same pool buffer overflow on Windows 10

What do we need

Quota Process Pointer Overwrite:

- The Pool Cookie
- The address of the overflowed chunk
- Arbitrary data in kernel-land at known address

Seems impossible...

ARMATURE TECHNOLOGIES

Pool Spraying

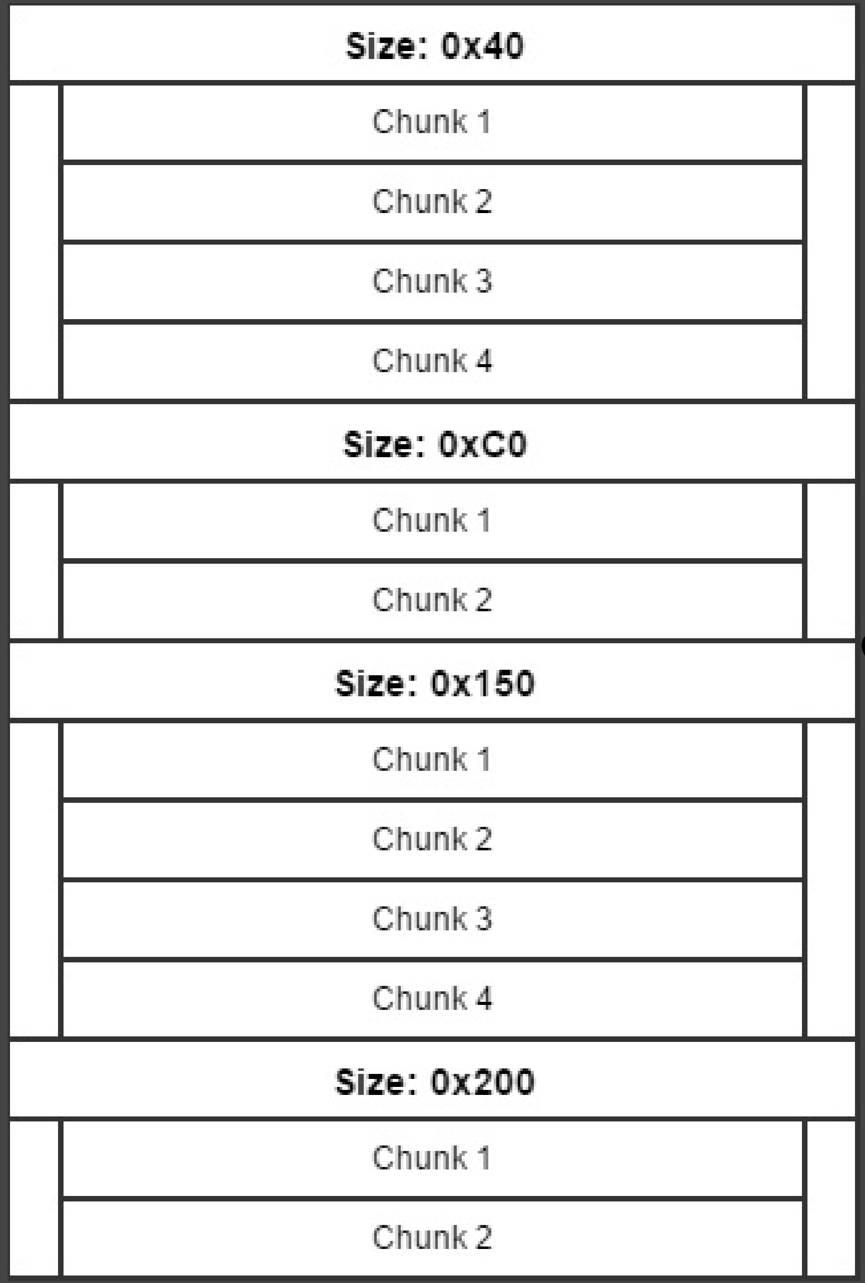
- Spraying is the art of making the further allocations predictible using the allocator behavior
- Provides you knowledge and control

Allocator Behavior

Two lists of free chunks :

- Lookaside list (for chunks with a size <= 0x200)
- ListHeads list

Lookaside List


Size: 0x40

Chunk 1

Chunk 2

Chunk 3

Chunk 4

Size: 0xC0

Chunk 1

Chunk 2

Size: 0x150

Chunk 1

Chunk 2

Chunk 3

Chunk 4

Size: 0x200

Chunk 1

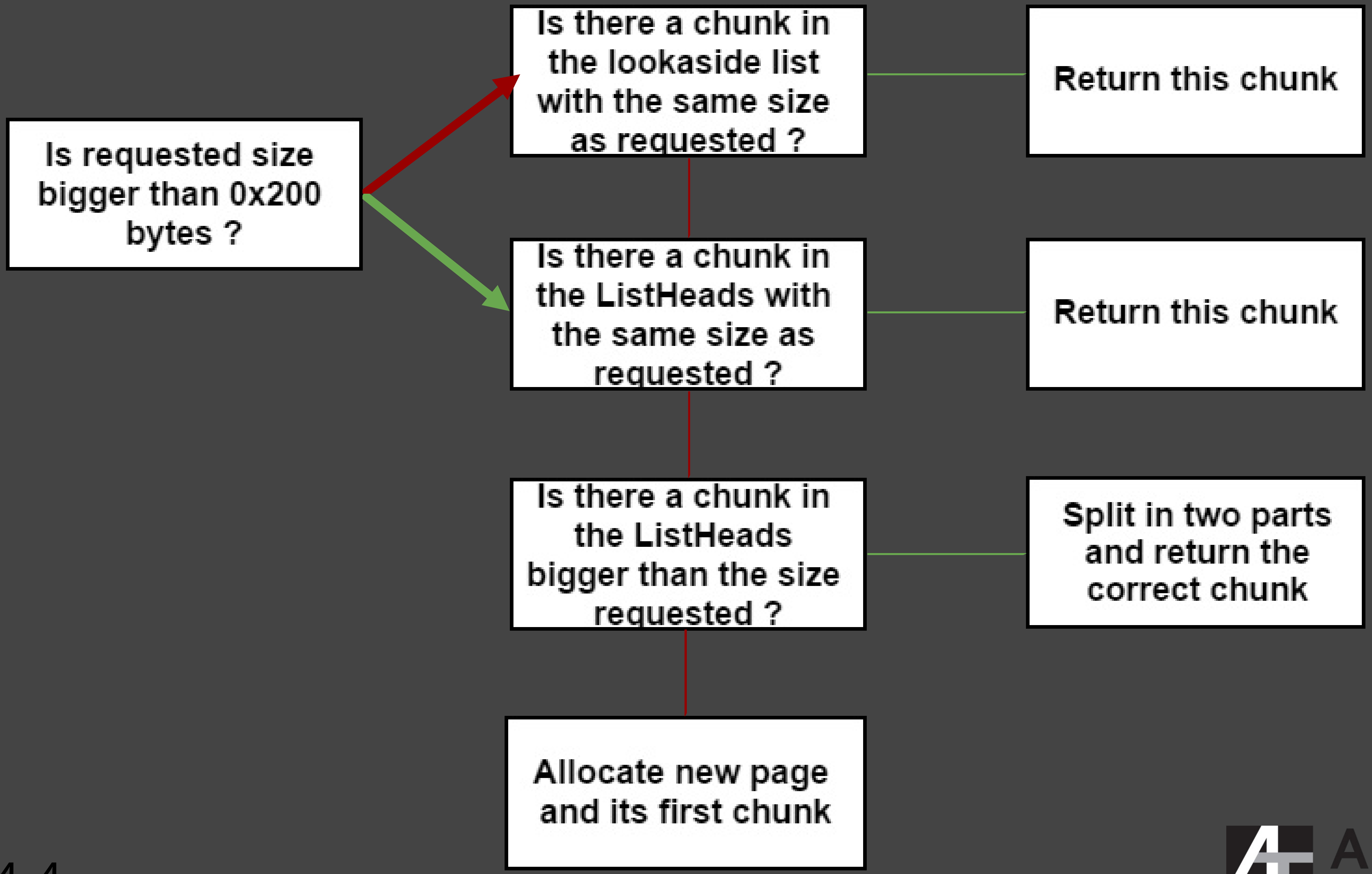
Chunk 2

- Contains chunk with a size ≤ 0x200 bytes

- Can contains only 255 chunks of the same size

Allocator behavior

Allocation algorithm



Is requested size bigger than 0x200 bytes ?

Is there a chunk in the lookaside list with the same size as requested ?

Return this chunk

Is there a chunk in the ListHeads with the same size as requested ?

Return this chunk

Is there a chunk in the ListHeads bigger than the size requested ?

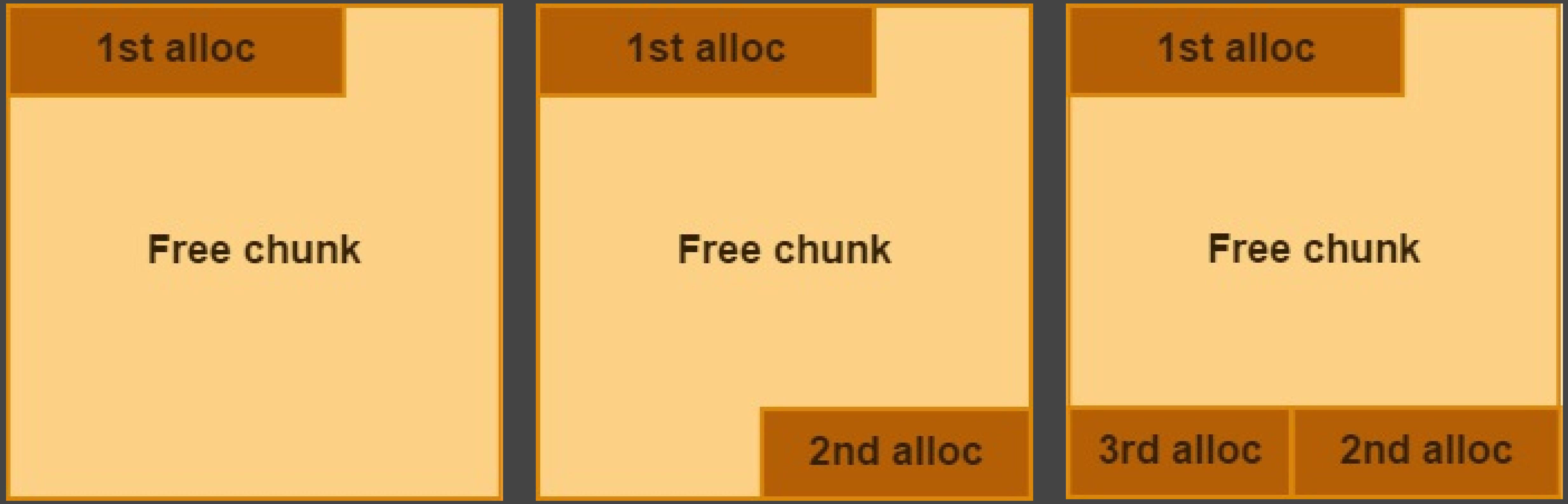
Split in two parts and return the correct chunk

Allocate new page and its first chunk

ARMATURE TECHNOLOGIES

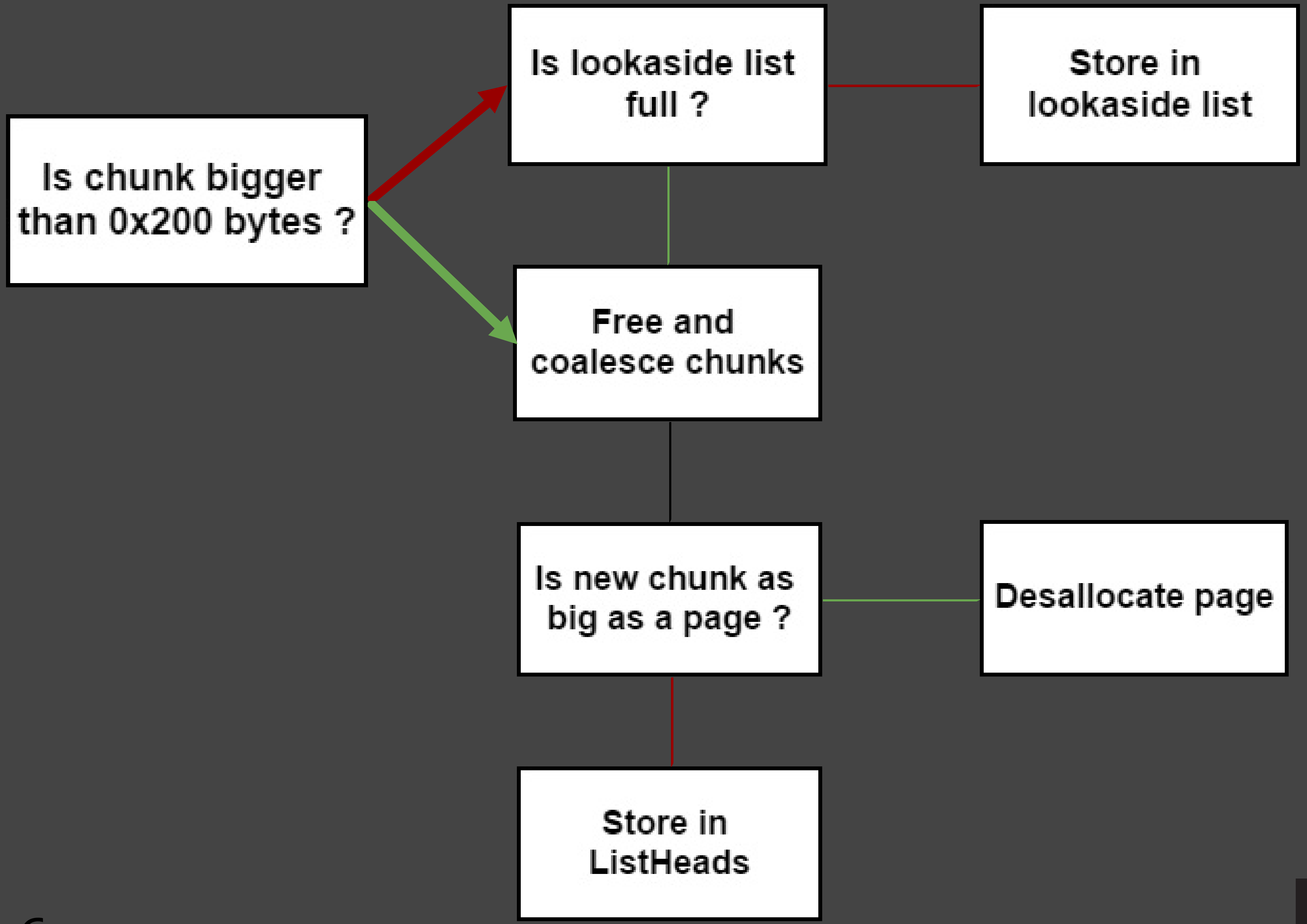
Allocator behavior

Allocation of a new page



Allocator behavior

Free algorithm



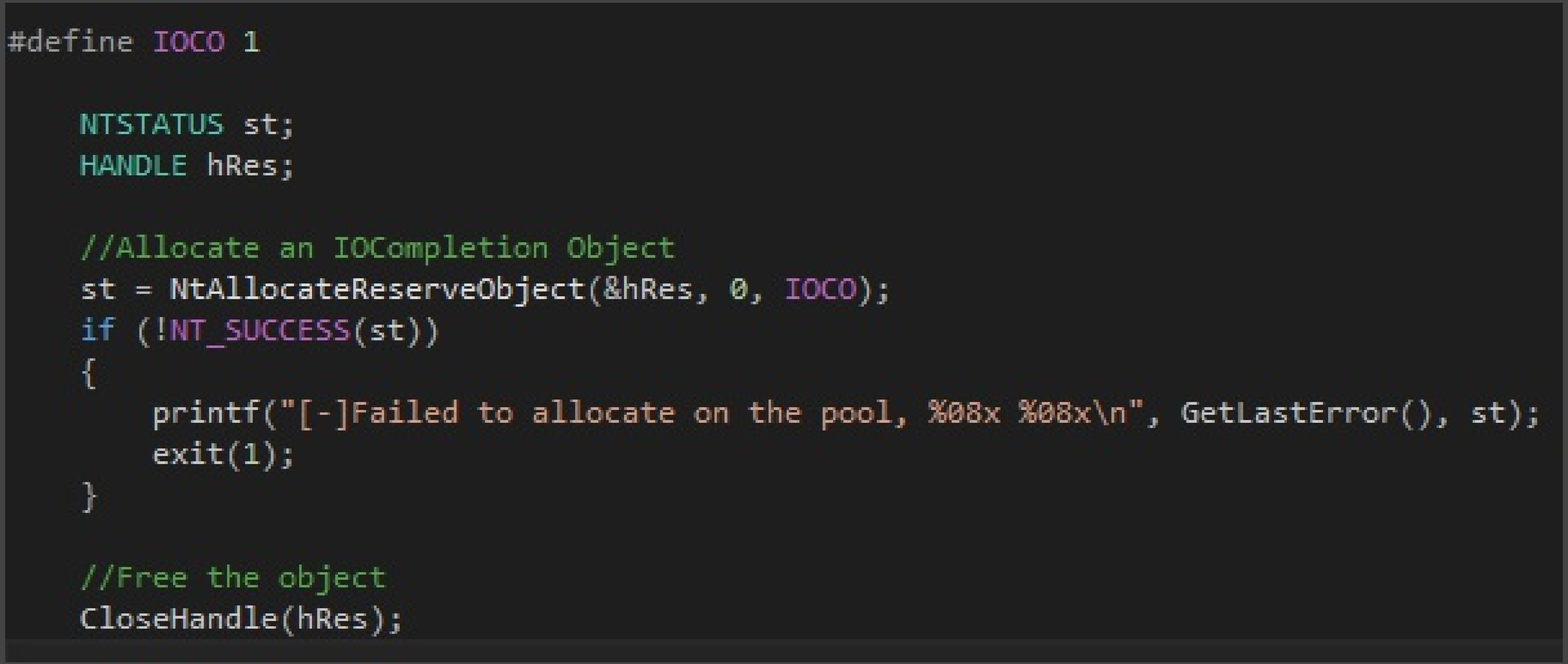
ARMATURE TECHNOLOGIES

Windows API tools

Windows named objects :

- A lot of different objects:
  - Reserved Objects
  - Semaphores
  - Processes
  - Register keys
  - Files
  - ...
- With various size
- Allocated in differents pools (Paged, NonPaged...)

Windows API tools



```
#define IOCO 1

    NTSTATUS st;
    HANDLE hRes;

    //Allocate an IOCompletion Object
    st = NtAllocateReserveObject(&hRes, 0, IOCO);
    if (!NT_SUCCESS(st))
    {
        printf("[-]Failed to allocate on the pool, %08x %08x\n", GetLastError(), st);
        exit(1);
    }

    //Free the object
    CloseHandle(hRes);
```

In userland, use a handle to interact with the object !

Basic Pool Spraying

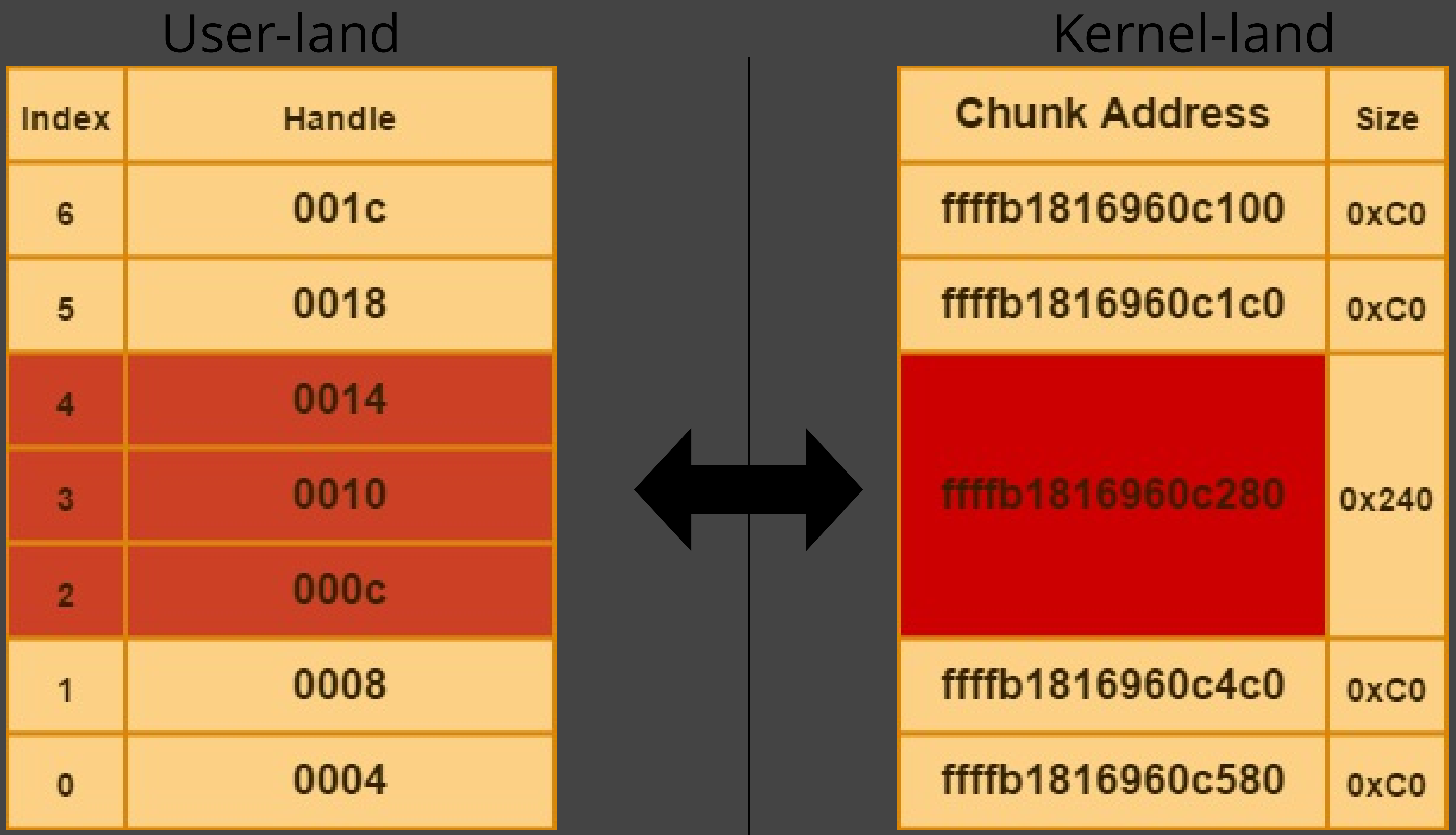
Step 1: Derandomize the pool

AKA : Massively allocate chunks

- Empty the Lookaside List
- Empty the ListHead List
- Create pages filled of our object

Basic Pool Spraying

Step 2: Create Gaps



User-land

Index	Handle
6	001c
5	0018
4	0014
3	0010
2	000c
1	0008
0	0004

Kernel-land

Chunk Address	Size
ffffb1816960c100	0xC0
ffffb1816960c1c0	0xC0
ffffb1816960c280	0x240
ffffb1816960c4c0	0xC0
ffffb1816960c580	0xC0

CloseHandle()

Chunks are freed and coalesced

Basic Pool Spraying

Problems:

- We can't predict allocations with a size <= 0x200 bytes
  - Or we need an object with the exact same size of the gap we want...
- Even if it's very likely, we're not sure the gaps we created actually exists
- We don't know the kernel addresses of our gaps

We can fix this

Another windows tool

Well known leak

NtQuerySystemInformation

└─ SystemExtendedHandleInformation

Retrieve any object's kernel address using its handle

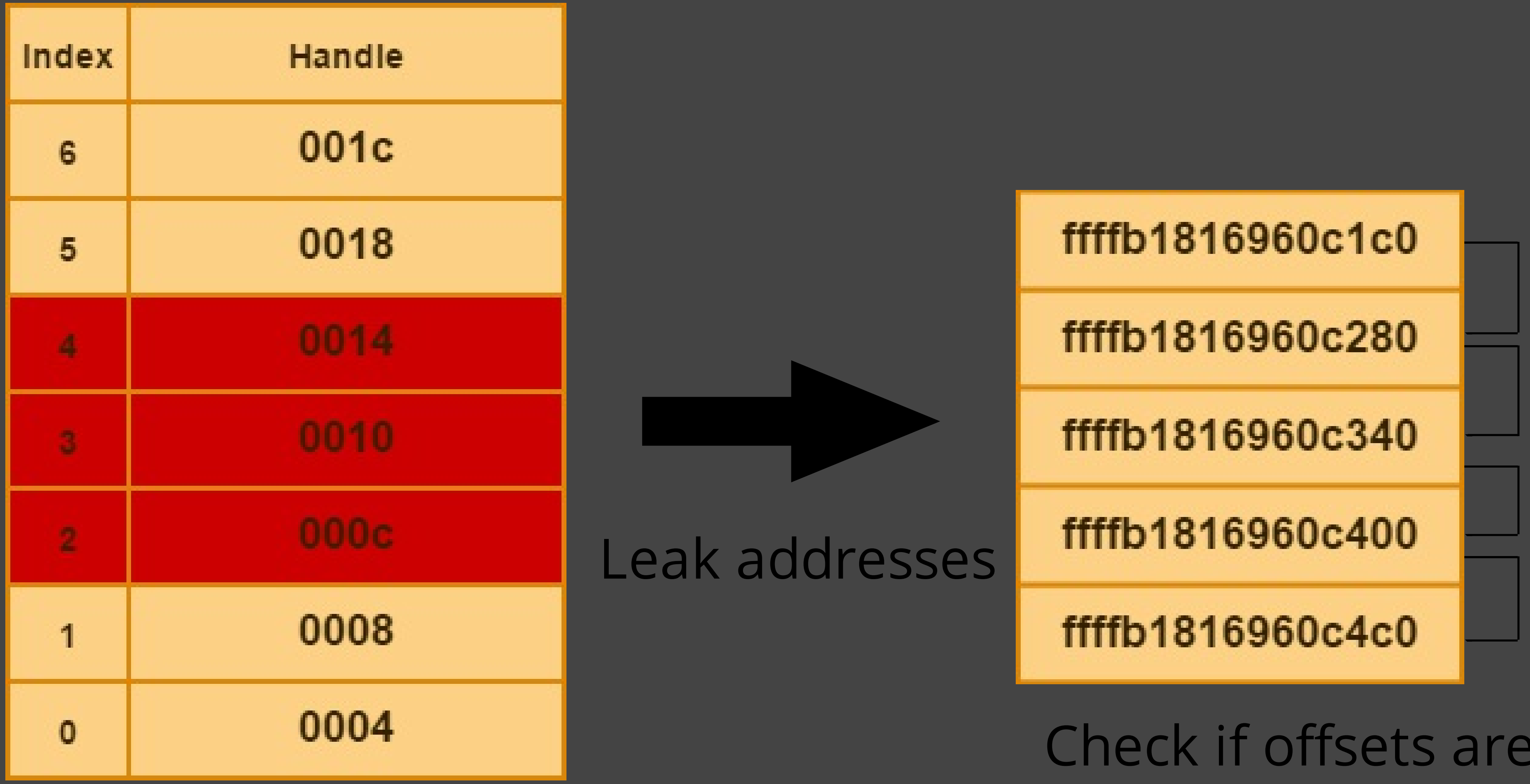
Advanced Pool Spraying

Step 1 : Derandomize the Pool

Step 2 : Find the perfect gap

Advanced Pool Spraying

Step 2 : Find the perfect gap



Index	Handle
6	001c
5	0018
4	0014
3	0010
2	000c
1	0008
0	0004

Leak addresses

- ffffb1816960c1c0
- ffffb1816960c280
- ffffb1816960c340
- ffffb1816960c400
- ffffb1816960c4c0

Check if offsets are correct

7.2

Advanced Pool Spraying

Step 3 : Enjoy your gaps !

- We can predict a future allocation at 100%
- And we know its kernel address
- Just Windows, only Windows

Time to start having fun !

What do we need

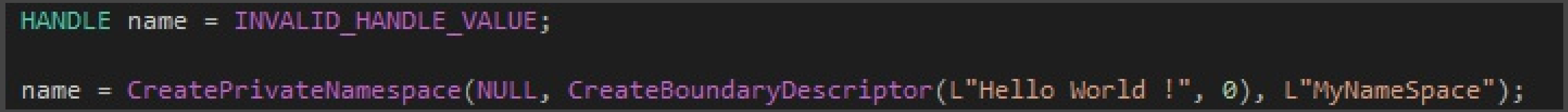
Quota Process Pointer Overwrite:

- The Pool Cookie
- ~~The address of the overflowed chunk~~
- Arbitrary data in kernel-land at known address

ARMATURE TECHNOLOGIES

Arbitrary data in kernel-land at known address

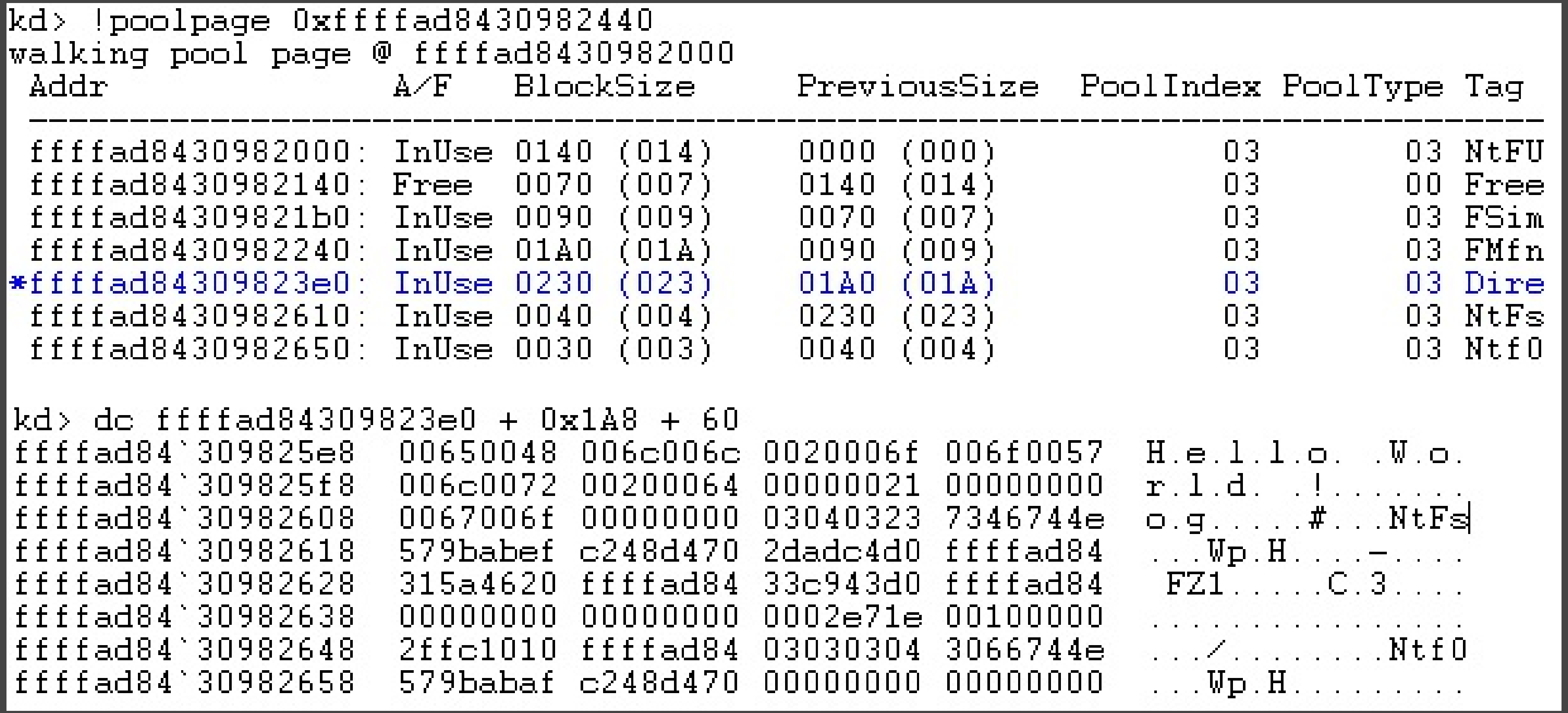
CreatePrivateNamespace() Function:



```
HANDLE name = INVALID_HANDLE_VALUE;

name = CreatePrivateNamespace(NULL, CreateBoundaryDescriptor(L"Hello World !", 0), L"MyNameSpace");
```

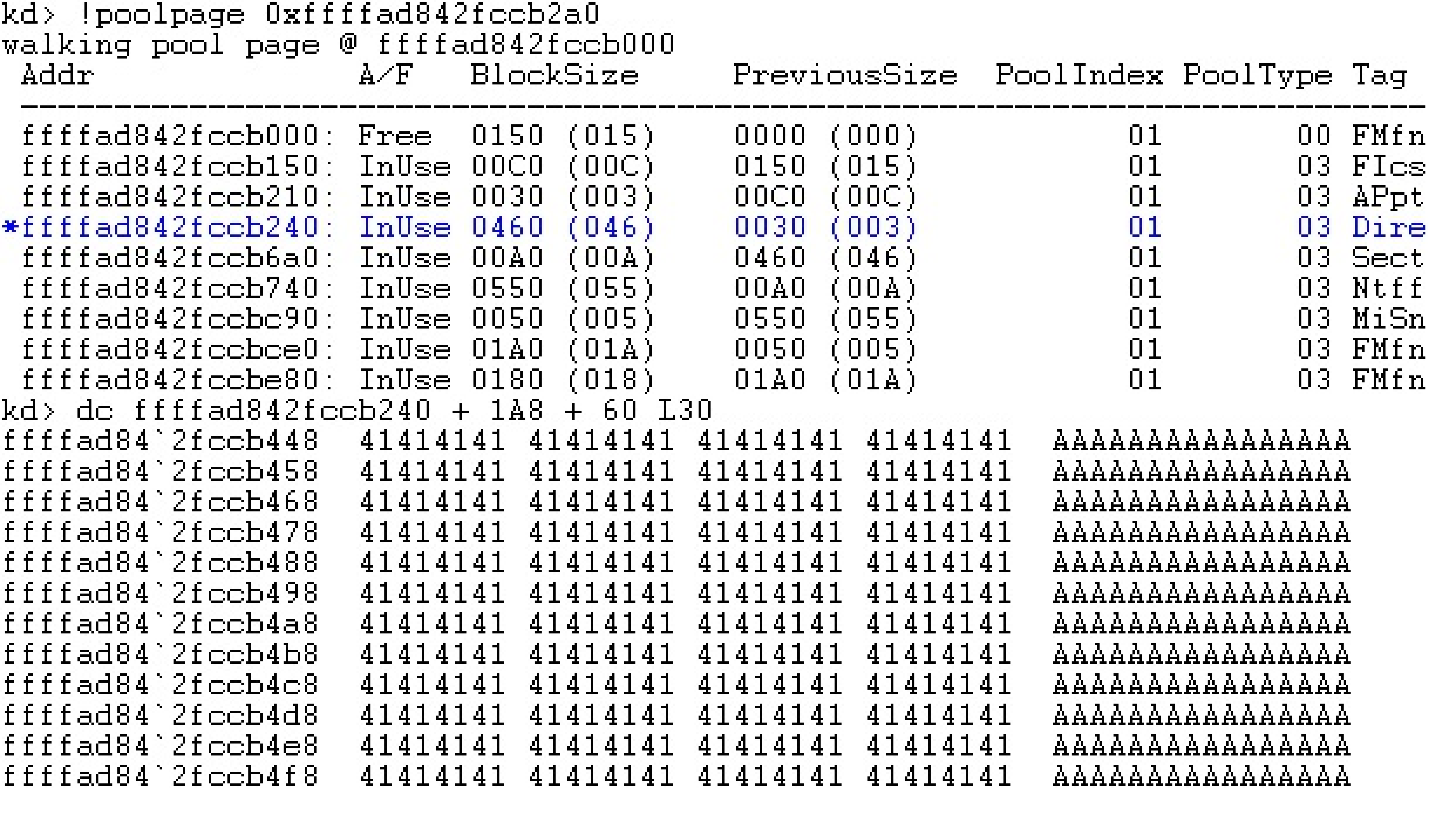
In paged pool, in the chunk of the object allocated



```
kd> !poolpage 0xffffad8430982440
walking pool page @ ffffad8430982000
 Addr                A/F   BlockSize      PreviousSize  PoolIndex PoolType Tag
 ------------------------------------------------------------------------------
 ffffad8430982000: InUse 0140 (014)    0000 (000)         03       03 NtFU
 ffffad8430982140: Free  0070 (007)    0140 (014)         03       00 Free
 ffffad84309821b0: InUse 0090 (009)    0070 (007)         03       03 FSim
 ffffad8430982240: InUse 01A0 (01A)    0090 (009)         03       03 FMfn
*ffffad84309823e0: InUse 0230 (023)    01A0 (01A)         03       03 Dire
 ffffad8430982610: InUse 0040 (004)    0230 (023)         03       03 NtFs
 ffffad8430982650: InUse 0030 (003)    0040 (004)         03       03 Ntf0

kd> dc ffffad84309823e0 + 0x1A8 + 60
ffffad84`309825e8  00650048 006c006c 0020006f 006f0057  H.e.l.l.o. .W.o.
ffffad84`309825f8  006c0072 00200064 00000021 00000000  r.l.d. .!.......
ffffad84`30982608  0067006f 00000000 03040323 7346744e  o.g.....#...NtFs
ffffad84`30982618  579babef c248d470 2dadc4d0 ffffad84  ...Wp.H....-....
ffffad84`30982628  315a4620 ffffad84 33c943d0 ffffad84   FZ1.....C.3....
ffffad84`30982638  00000000 00000000 0002e71e 00100000  ................
ffffad84`30982648  2ffc1010 ffffad84 03030304 3066744e  .../........Ntf0
ffffad84`30982658  579babaf c248d470 00000000 00000000  ...Wp.H.........
```

Arbitrary data in kernel-land at known address



```
kd> !poolpage 0xffffad842fccb2a0
walking pool page @ ffffad842fccb000
 Addr               A/F   BlockSize     PreviousSize  PoolIndex PoolType Tag
 ---------------------------------------------------------------------------
 ffffad842fccb000: Free  0150 (015)    0000 (000)           01        00 FMfn
 ffffad842fccb150: InUse 00C0 (00C)    0150 (015)           01        03 FIcs
 ffffad842fccb210: InUse 0030 (003)    00C0 (00C)           01        03 APpt
*ffffad842fccb240: InUse 0460 (046)    0030 (003)           01        03 Dire
 ffffad842fccb6a0: InUse 00A0 (00A)    0460 (046)           01        03 Sect
 ffffad842fccb740: InUse 0550 (055)    00A0 (00A)           01        03 Ntff
 ffffad842fccbc90: InUse 0050 (005)    0550 (055)           01        03 MiSn
 ffffad842fccbce0: InUse 01A0 (01A)    0050 (005)           01        03 FMfn
 ffffad842fccbe80: InUse 0180 (018)    01A0 (01A)           01        03 FMfn
kd> dc ffffad842fccb240 + 1A8 + 60 L30
ffffad84`2fccb448  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb458  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb468  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb478  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb488  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb498  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4a8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4b8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4c8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4d8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4e8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
ffffad84`2fccb4f8  41414141 41414141 41414141 41414141  AAAAAAAAAAAAAAAA
```

What do we need

Quota Process Pointer Overwrite:

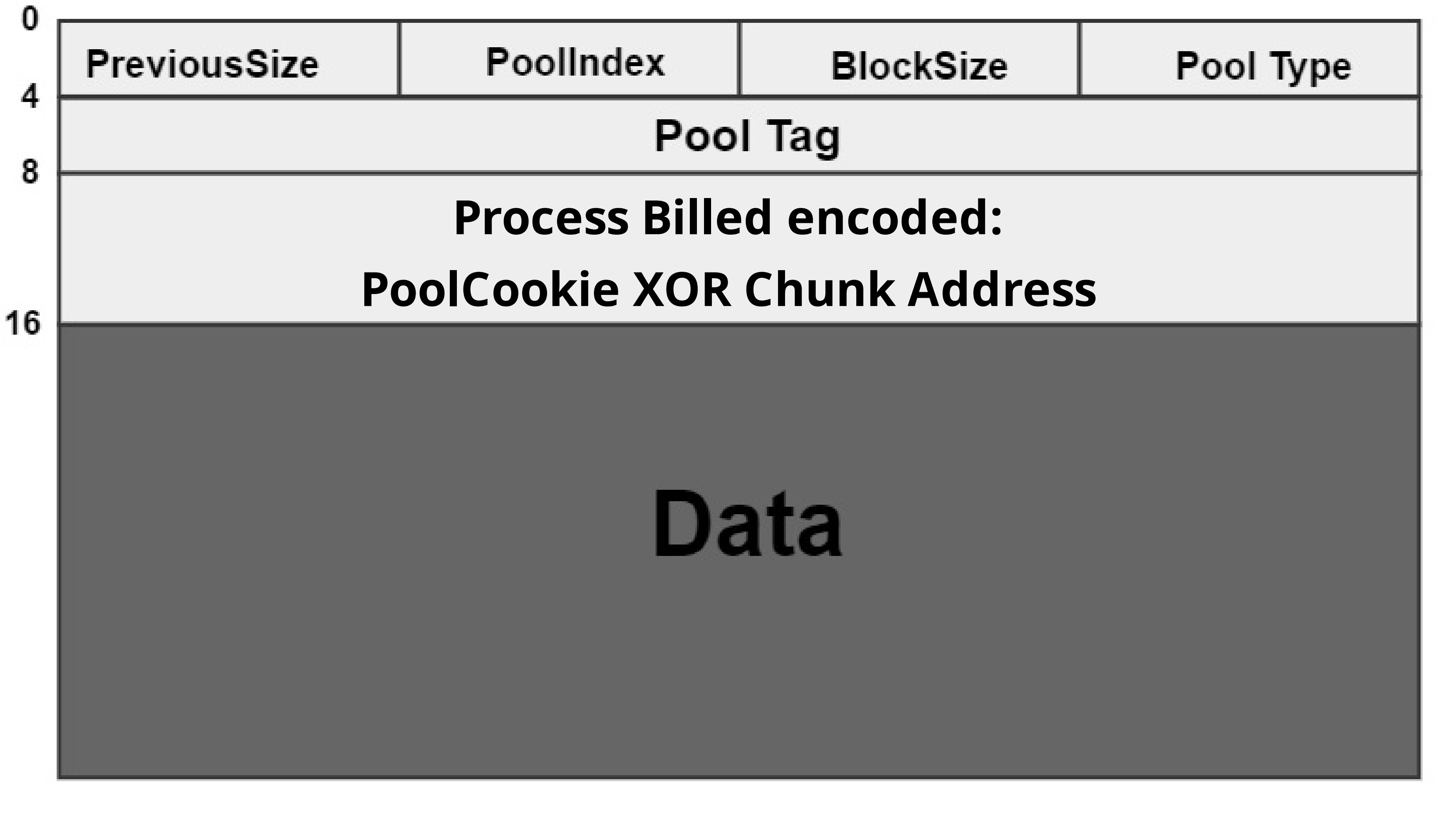
- The Pool Cookie
- ~~The address of the overflowed chunk~~
- ~~Arbitrary data in kernel-land~~

The Pool Cookie

- Symbol: nt!ExpPoolQuotaCookie
- Generated at boot
- Good enthropy

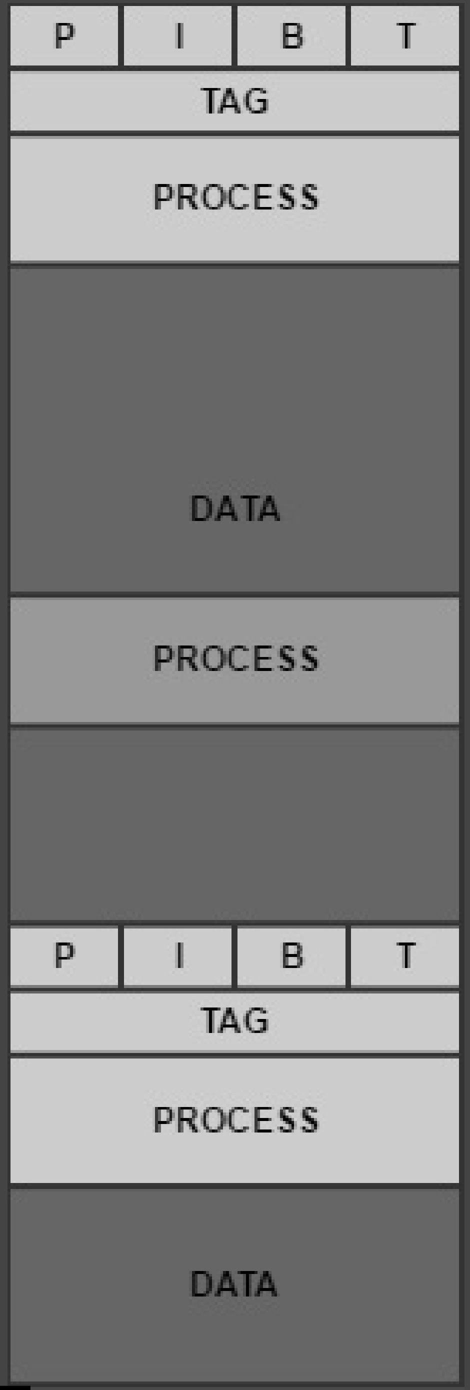
The Pool Cookie

Free chunk



Offset				
0	PreviousSize	PoolIndex	BlockSize	Pool Type
4	Pool Tag			
8	Process Billed encoded: PoolCookie XOR Chunk Address			
16	Data			

The Pool Cookie



1. Spray the pool in order to have controllable chunks
2. Free a chunk
3. Free the chunk just before
4. Reallocate a chunk with the size of the gap
5. The data is not overwritten... With a correct IOCTL, you might be able to read the old headers... containing the PoolCookie XORED with old chunk address

ARMATURE TECHNOLOGIES

The Pool Cookie

About CVE-2017-7441

- Use our input to call the function RtlLookupElementGenericTableAvl
- Write the result in the SystemBuffer for return but doesn't wipe the whole buffer
- Because of unicode and bad calculation, specify a wrong number to the IOManager: the driver write *n* bytes and tell *n+2* to the driver
- 2 bytes Out-Of-Bounds read
- It's enough to leak the PoolCookie !

What do we need

Quota Process Pointer Overwrite:

- ~~The Pool Cookie~~
- ~~The address of the overflowed chunk~~
- ~~Arbitrary data in kernel-land at known address~~

Let's exploit !

DEMO

Conclusion

Drivers are still a great attack vector:

- A buffer is used for input/output and we control its size...
- A buffer overflow is exploitable !

Be careful when writing a driver...

- You're dealing with user input in kernel land...
- The tyniest mistake becomes a critical vulnerability

Completely remediate the NtQuerySystemInformation leak !

QUESTIONS ?

ARMATURE
TECHNOLOGIES

Thanks for listening !

- A library for Pool Spraying : https://github.com/cbayet/PoolSprayer
- Source code of the exploits : https://github.com/cbayet/Exploit-CVE-2017-6008
- Full paper on Pool Spraying : https://trackwatch.com/windows-kernel-pool-spraying
- Full paper on exploits : https://trackwatch.com
- My twitter: https://twitter.com/OnlyTheDuck

I'm interested in job offers !