Windows Heap and Heap overflow

Group 7
Wenjun Li, Xiaohang Yu, Chao Lei

Introduction

This presentation will

explain windows heap data structure and algorithm

examine how to exploit heap based buffer overflows on the Windows platform.

Heap overflows have been well documented on *nix platforms, but they’ve not been well documented on Windows

What is heap

The heap is an area of memory used for storage of dynamic data. Every process has a default process heap but a developer can create their own private heaps. Space is allocated from the heap and freed when finished with.

Background

- Windows heap is a mysterious, intriguing and chaotic place
- Technical details of Windows heap is not open source
- All the knowledge about it come from personal research of hackers, geeks

Background(cont)

Halvar Flake

Top 10 sexy info sec geeks for 2011 :)
“Third Generation Exploitation” Black Hat

David Litchfield

“Windows Heap overflows” Black Hat

Matt Conover

“XP SP2 Heap Exploitation” Black Hat

I think it’s unfair, because they are extremely SMART and HANDSOME :(

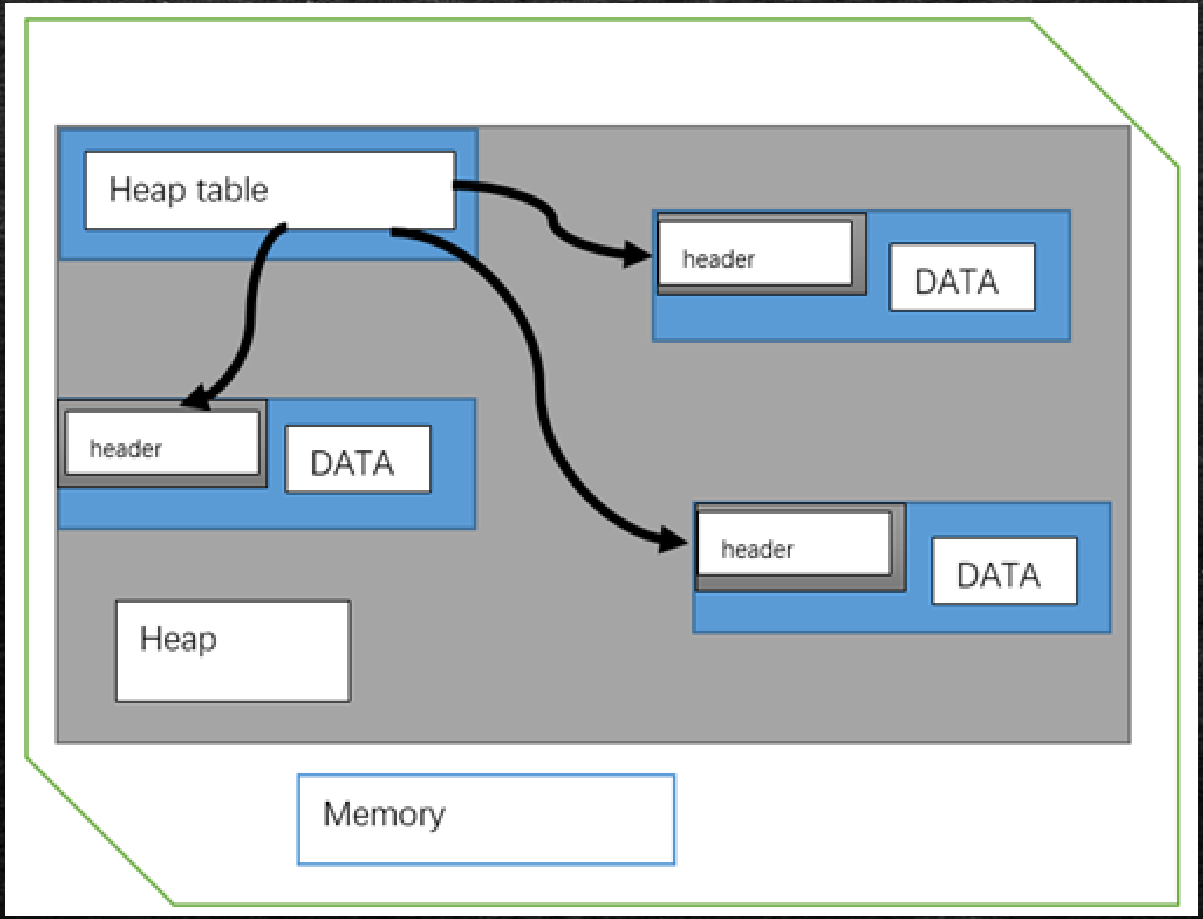
History of Windows Heap

- Modern operating system’ s heap management can balance the efficient utilization of memory, speed of decision-making, robustness and security.
- Windows 2000 – windows XP SP1
  - No security policy
- Windows XP2 – Windows 2003
  - Add some security factors
- Windows Vista – Windows 7
  - Milestone in Windows heap history

Data structure and management strategy

- When using heap:
  - For programmers: request it, use it, free it.
  - For system: **identify** free memory on **chaotic** heap find an **appropriate** space and return to user.
- Two data structure:
  - Heap block
  - Heap table

Heap memory organization



Two important heap tables

- In windows operating system, occupied heap blocks are indexed by programs that use it, however, heap table only index all the idle state heap blocks.
- Two important heap tables:
  - Freelist and lookside

Windows Heap Design

Each heap starts with a structure. This structure, amongst other data, contains an array of 128 LIST_ENTRY structures. Each LIST_ENTRY structure contains two pointers – see winnt.h. This array can be found at 0x178 bytes into the heap structure – call it the FreeList array.

Windows Heap Design (cont)

When a heap is first created there are two pointers that point to the first free block set in FreeList[0]. Assuming the heap base address is 0x00350000 then first available block can be found at 0x00350688.

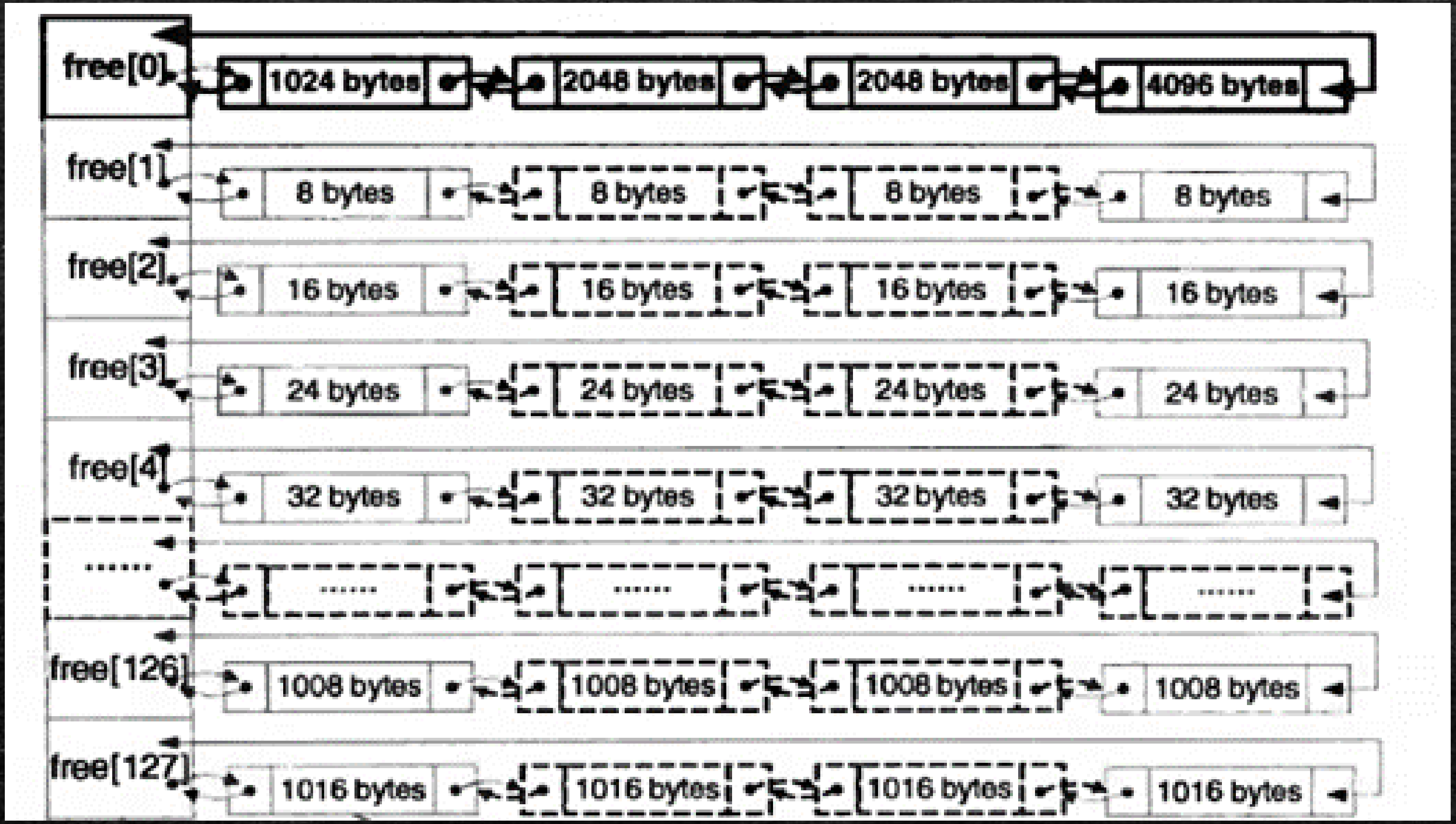
Windows Heap design(cont)

0x00350178 (FreeList[0].Flink) = 0x00350688 (First Free Block)
0x0035017C (FreeList[0].Blink) = 0x00350688 (First Free Block)

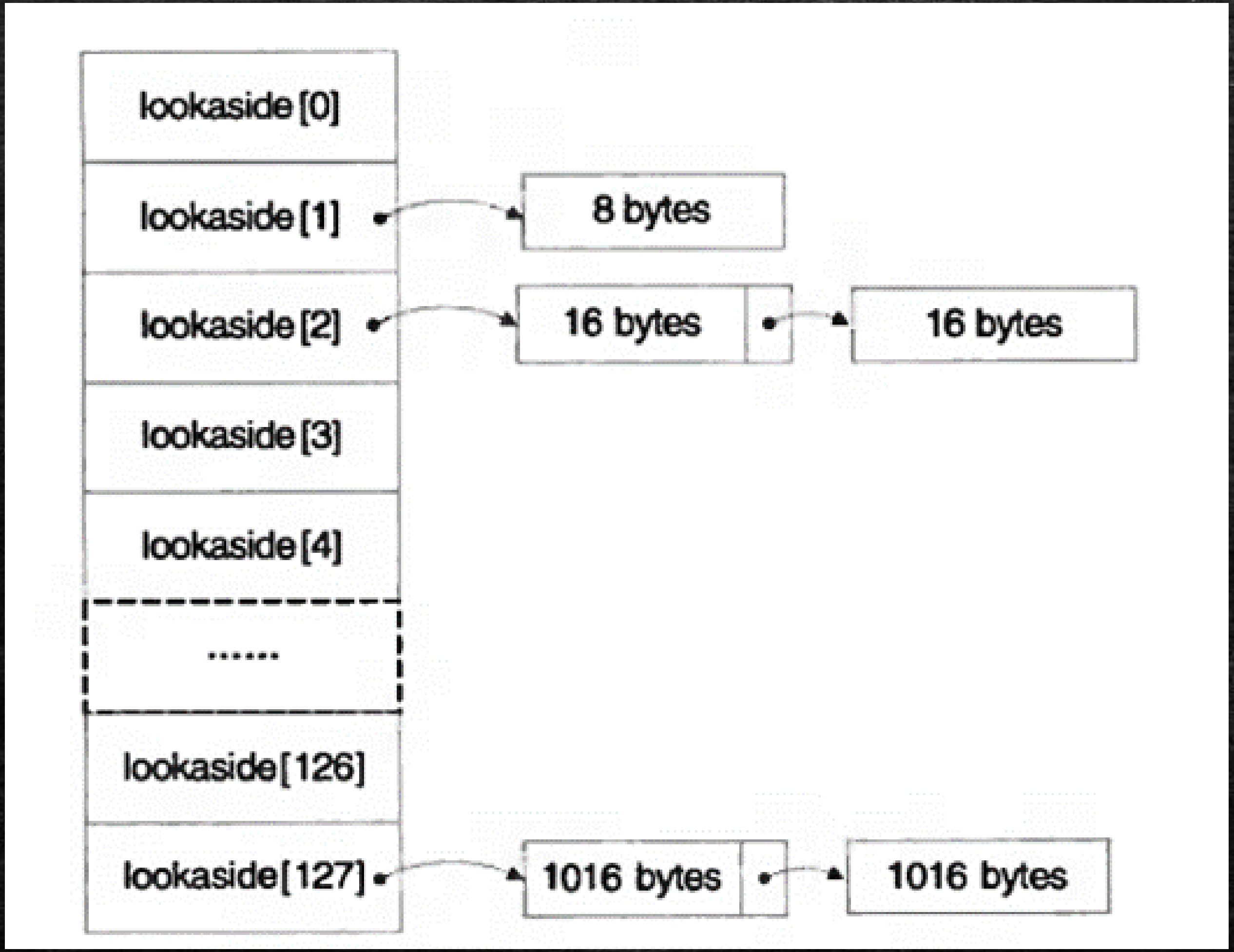
0x00350688 (First Free Block) = 0x00350178 (FreeList[0])
0x0035068C (First Free Block+4) = 0x00350178 (FreeList[0])

When an allocation occurs these pointers are updated accordingly. As more allocations and frees occur these pointers are continually updated and in this fashion allocated blocks are tracked in a doubly linked list.

Two important heap table(Freelist)



Two important heap table(Lookside)



Windows Heap operation

- Allocation
- Free
- Coalesce

Windows Heap operation (allocation)

- Allocation: 3 types
  - Lookside
    - allocation for lookside is pretty simple, it includes find the idle heap block which has the fitted size, modify its state to occupied, unload it from heap tables and return a pointer that point to heap block body to program.
  - Freelist (expect freelist[0])
    - first search most optimal idle chunk, if failed, then find the next best idle chunk(smallest idle block that meet the requirement)
  - Freelist[0]
    - dle chunks are link based on ascending order, thus during the allocation management program will first search the last chunk(the biggest one), to see whether it meet the requirement, if successes, check the penult until find the smallest one that meet the requirement

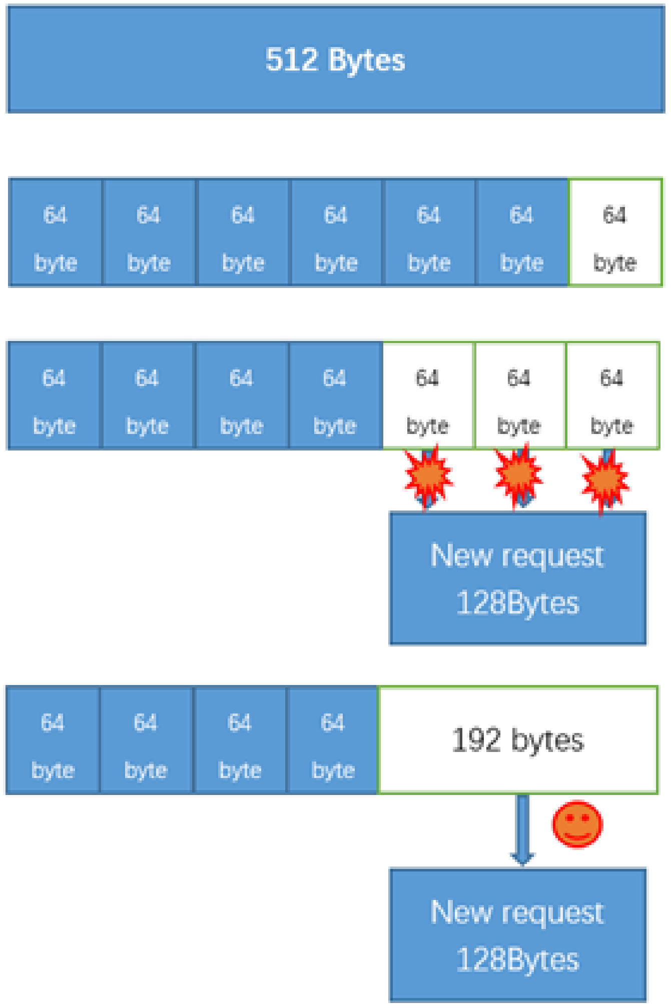
Windows Heap operation(free)

To release a heap block, we just need to modify the states of block state to idle, and link the block into heap table, all the blocks released will link to the end of heap table. When allocate, management system will find the free block from the end of table.

Windows Heap operation(coalesce)

- After many times of requests and release, there will be a lot of memory fragments on heap, then heap management system will combine all these fragments.
- When management system finds two adjacent idle chunk, it will combine them. Actually there is another operation which name is shrink the compact, executed by RtlCompactHeap, its consequence is similar to disk defragmentation

Windows Heap operation(shrink the compact)



1. There is an idle chunk in memory, size is 512 byte

2. Program A request 64 bytes 6 times.

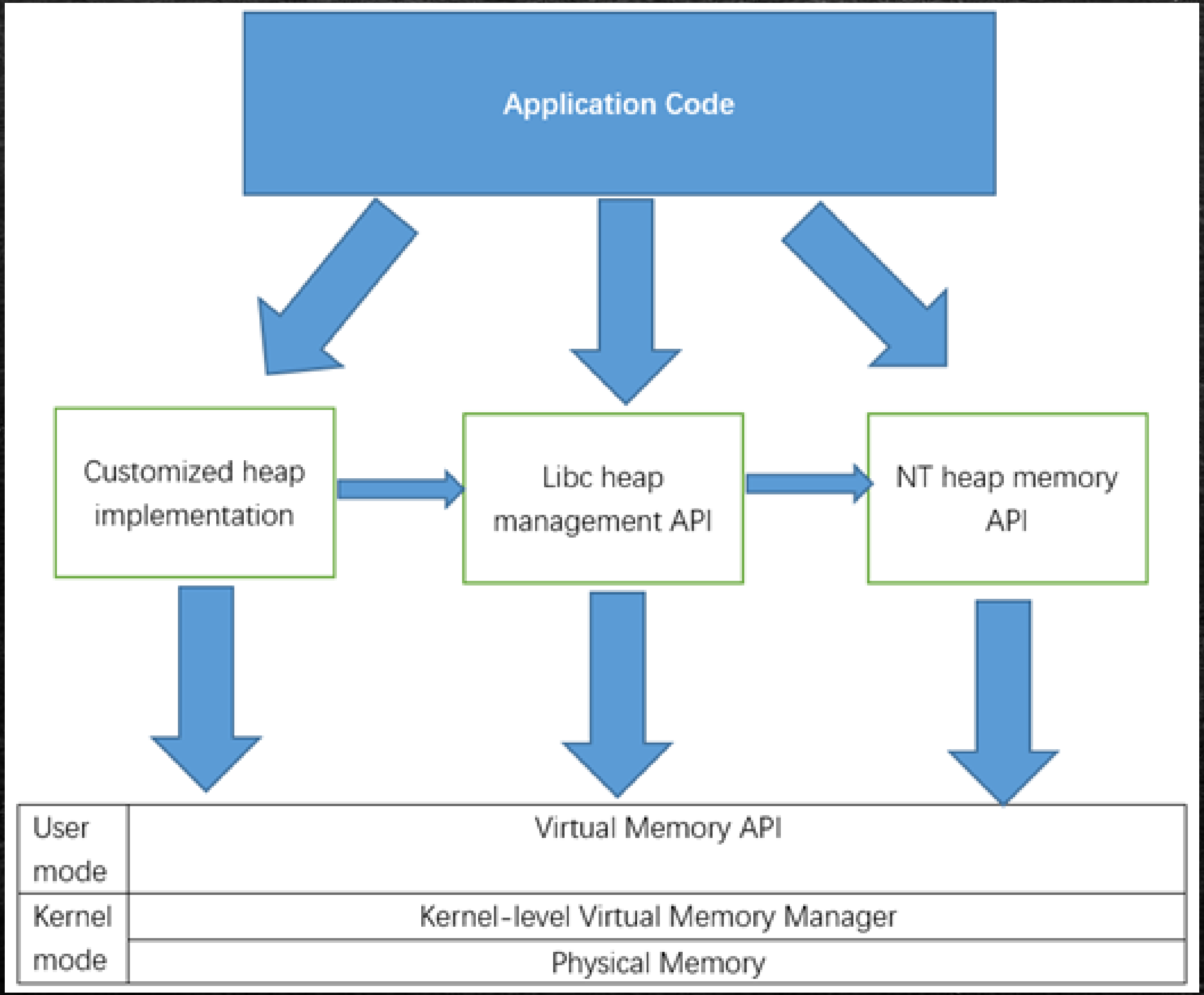
3. During the usage, 2 chunk are freed, then program A request 128 Bytes, although there are 192 bytes free on heap, but theses idle chunks are not adjacent, in another word, they are fragments.

4. If we combine small chunks into a big one, then we can allocate it

Windows Heap operation(allocation and free)

	Allocation	Release
Small chunk	1. Lookside allocation 2. if 1 failed, freelist allocation 3. if 2 failed, use heap cache allocation 4. if 3 failed, try free[0] allocation 5. if 4 failed, try shrink the compact, then do 1 6. if 5 failed, then return NULL	1. first try to link the chunk into lookside 2. if lookside is full, then link chunk into related freelist
Big chunk	1.use heap cache alloction 2.if 1 failed, then use big chunk in free[0] to allocate it.	1. first try to release into heap cache 2. if 1 failed, then will link into freelist[0]
Giant chunk	Very rare, relate to virtual memory allocation	Release directly, no heap table operation

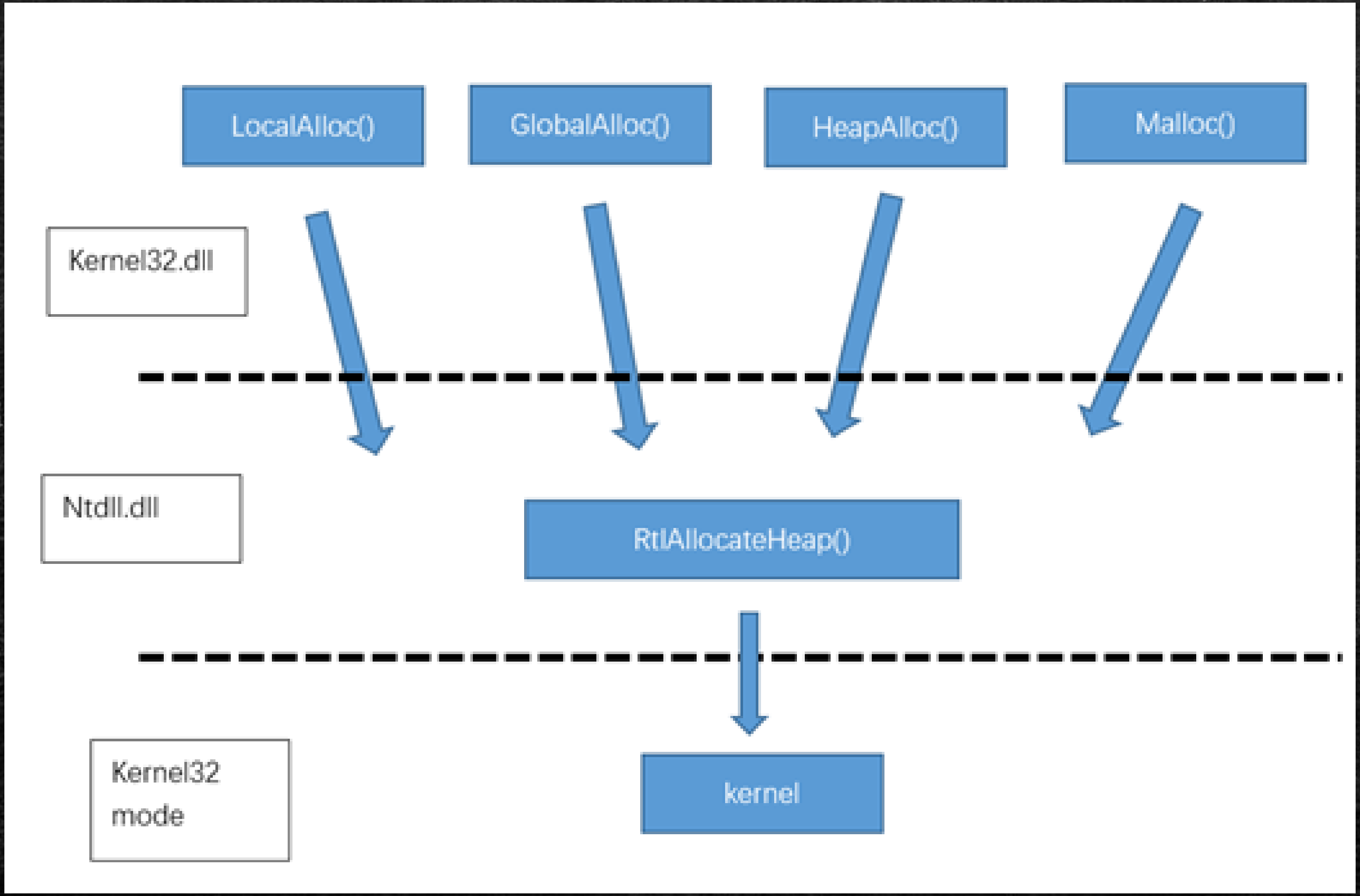
Heap functions



Heap functions

- Windows provides varies type of heap allocation functions. But all the functions will be distributed by RtlAllocateHeap() function in ntdll.dll, this function is the lowest functions that users can see in windows operating system.

Heap functions



Time to see some code :-)

	Environment
Operating system	Windows 2000 virtual machine
VMware version	VMware workstation Pro 12
Compiler	VC++ 6.0
Build option	Default
Build version	Release

Heap Structure

- Start at 0x00420000
- Segment List
- Virtual Allocation list (all 0)
- Freelist usage bitmap (32 bytes)
- Range of Freelist: offset 0x178 – 0x584
- Tail chunk: offset 0x680

Data Structure of Free Heap Block

Block head (free)	Self Size	Previous chunk Size	Segme nt index	Flags	Unused bytes	Tag index
	Flink in freelist		Blink in Freelist			
Block body	DATA…..					

Data Structure of Busy Heap Block

Self size	Previous chunks size	Segment index	Flags	Unused bytes	Tag index (Debug)	Block head
DATA....						Block body

FLAGS

0x01 (bin:00000001)- Busy
0x02 (bin:00000010)- Extra present
0x04 (bin:00000100)- Fill pattern
0x08 (bin:00001000)- Virtual Alloc
0x10 (bin:00010000)- Last entry
0x20 (bin:00100000)- FFU1
0x40 (bin:01000000)- FFU2
0x80 (bin:10000000)- No coalesce

Reference

1. https://www.blackhat.com/presentations/win.../halvarflake-winsec02.ppt
2. https://www.blackhat.com/presentations/win.../bh-win-04-litchfield.ppt
3. www.blackhat.com/.../BHUSA09-McDonald-WindowsHeap-PAPER.pdf
4. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366723(v=vs.85).aspx
5. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366574(v=vs.85).aspx
6. https://msdn.microsoft.com/en-us/library/windows/desktop/aa366597(v=vs.85).aspx
7. https://msdn.microsoft.com/en-us/library/6ewkz86d.aspx
8. https://msdn.microsoft.com/en-us/library/windows/hardware/ff552108(v=vs.85).aspx

Thank you