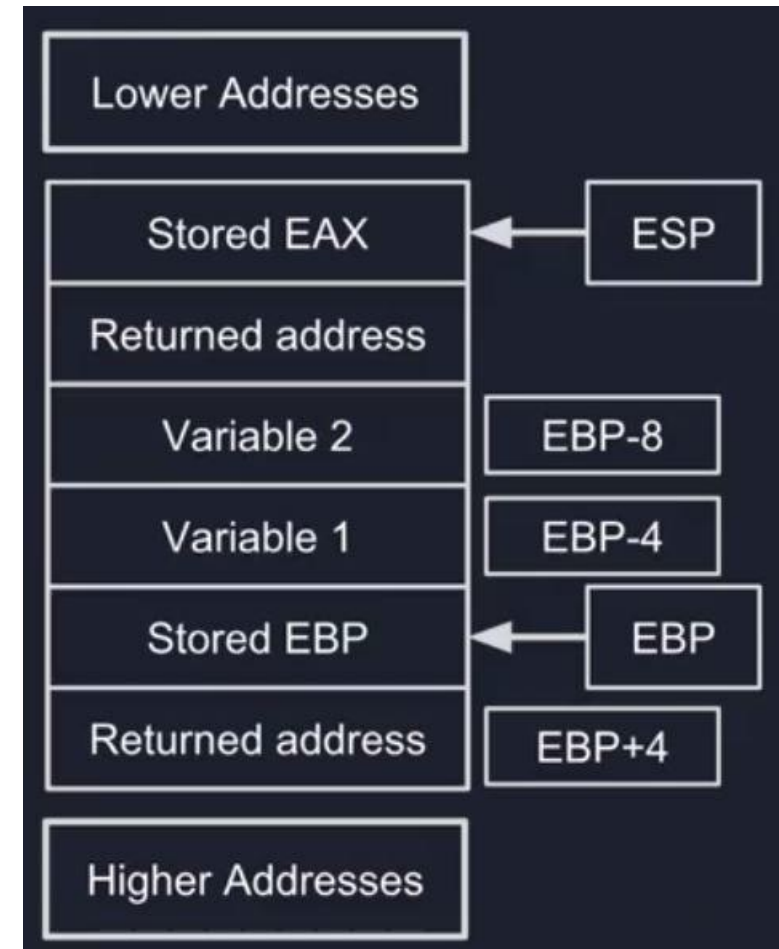


Assembly Language Basics

For Malware Analysis of Native Exe

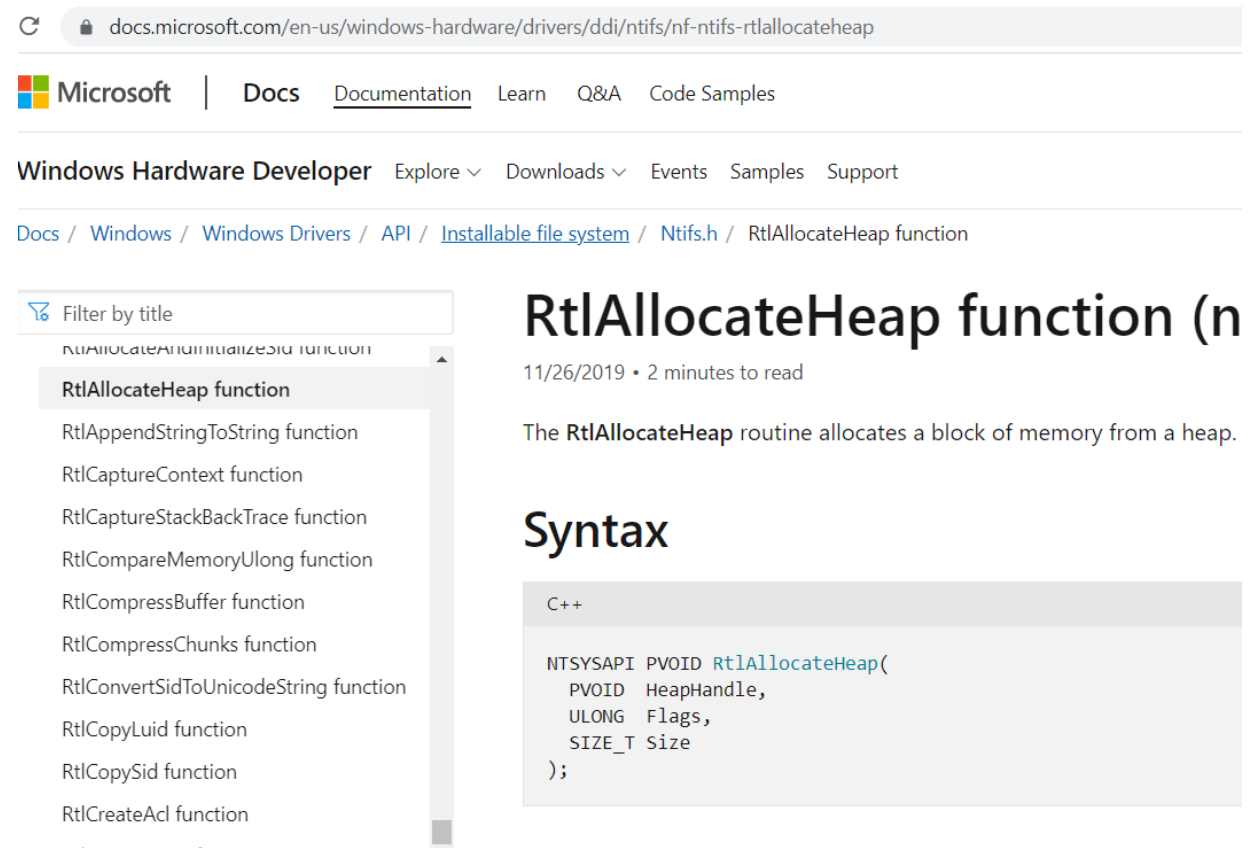
What is the Stack?

- LIFO (Last In First Out) Data Structure
- Stores local variables, and return addresses for functions
- Accessed through push, pop, call and ret
- RAM memory layout:
 - Starts at higher addresses and as more values are pushed, smaller addresses are used



What is the Heap?

- Globally stored memory
- All functions can access it
- Typically stored in the Data Section of a program
- RtlAllocateHeap can be used to create a Heap
- Malware use heap as storage area for anything it is going to use



The screenshot shows a web browser displaying the Microsoft Docs page for the `RtlAllocateHeap` function. The browser's address bar shows the URL: `docs.microsoft.com/en-us/windows-hardware/drivers/ddi/ntifs/nf-ntifs-rtlallocateheap`. The page header includes the Microsoft logo and navigation links for Docs, Documentation, Learn, Q&A, and Code Samples. Below the header, there are navigation links for Windows Hardware Developer, Explore, Downloads, Events, Samples, and Support. The breadcrumb trail indicates the path: Docs / Windows / Windows Drivers / API / Installable file system / Ntifs.h / RtlAllocateHeap function. A search filter box is visible, with a list of functions including `RtlAllocateHeap`, which is highlighted. To the right of the search results, the main heading reads "RtlAllocateHeap function (ntifs.h)", followed by the date "11/26/2019" and "2 minutes to read". A brief description states: "The `RtlAllocateHeap` routine allocates a block of memory from a heap." Below this, the "Syntax" section shows the C++ function signature:

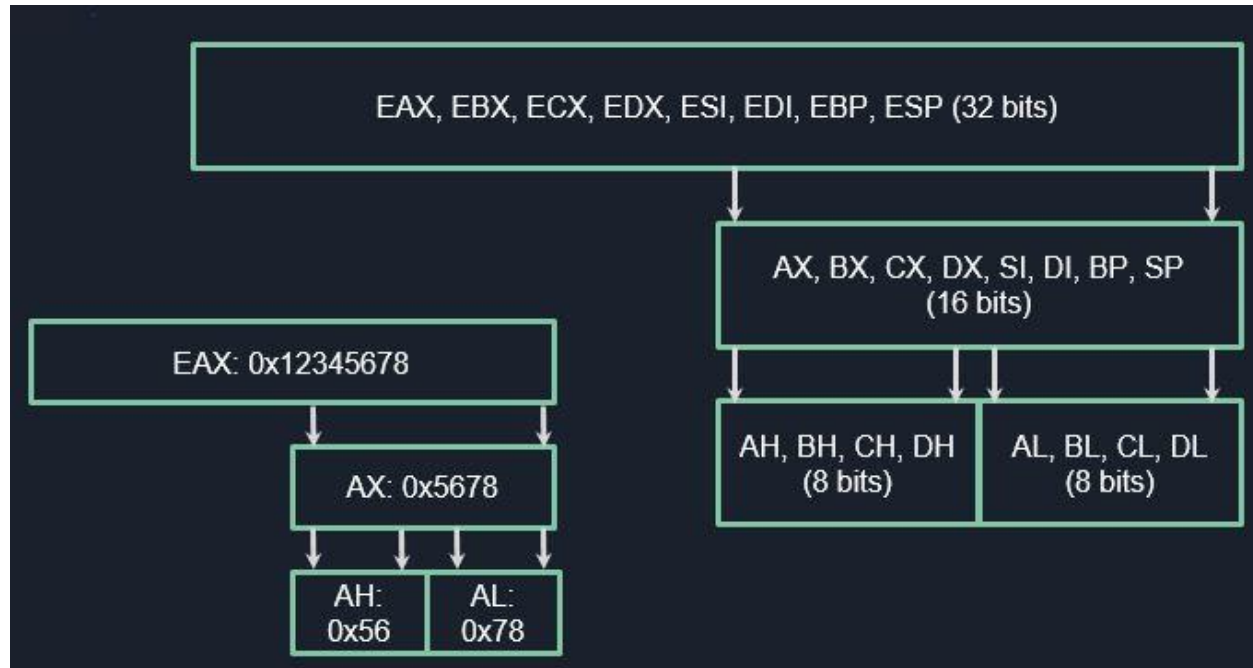
```
NTSYSAPI PVOID RtlAllocateHeap(
    PVOID HeapHandle,
    ULONG Flags,
    SIZE_T Size
);
```

CPU Registers

Registers	Purpose
EAX	Accumulator (Arithmetic)
EBX	Base (Pointer to Data)
ECX	Counter (Shift/Rotate instructions + loops)
EDX	Data (Arithmetic and I/O)
ESI	Source Index (Pointer to Source in stream operations)
EDI	Destination Index (Pointer to Destination in stream operations)
EBP	Base Pointer (Pointer to Base of Stack)
ESP	Stack Pointer (Pointer to top of Stack)
EIP	Instruction Pointer (Address of next instruction to exec)

Segment Registers	
SS	Stack Pointer
CS	Code Pointer
DS	Data Pointer
ES	Extra Data Pointer
FS	Extra Data Pointer
GS	Extra Data Pointer

Accessing parts of a register



dword = 4 bytes (32 bits), word = 2 bytes (16 bits), byte = 8 bits

Flags Register

- register where each bit acts as flag, containing a 1 or a 0

Flags	Purpose
CF	Carry Flag - Set when the result of an operation is too large for the destination operand
ZF	Zero Flag - Set when the result of an operation is equal to zero
SF	Sign Flag - Set if the result of an operation is negative
TF	Trap Flag - Set if step by step debugging - only one instruction will be executed at a time

Assembly Language Instructions

- Three main categories:
 - Data transfer (mov)
 - Control Flow (push, call, jmp ...)
 - Arithmetic/Logic (xor, or, and, mul, add ...)

Examples of Data Transfer Instructions

Instruction	Purpose	Format	Example
mov	Move	<i>mov dest, src</i>	mov eax, [edx]
movzx	Move-Zero-Extended	<i>movzx dest, src</i>	movzx eax, 0x123
lea	Load Effective Address	<i>lea dest, src</i>	lea edx, [ebp-0x40]
xchg	Exchange (Swap)	<i>xchg dest, src</i>	xchg eax, ebx

Examples of Control Flow Instructions (function calls)

Instruction	Purpose	Format	Example
call	Execute function	<i>call function</i>	call sub_3B18C0
push	Push value to stack	<i>push value</i>	push ecx
pop	Pop value off stack	<i>pop register</i>	pop ebx
ret	Return from function	<i>ret</i>	ret

Examples of Control Flow Instructions (Jumps)

Instruction	Purpose	Format	Example
<code>jmp</code>	Unconditional Jump	<i>jmp address</i>	<code>jmp [eax]</code>
<code>je</code>	Jump if Equal (ZF = 1)	<i>je address</i>	<code>je loc_</code>
<code>jnz</code>	Jump if Not Zero (ZF = 0)	<i>jnz address</i>	<code>jnz loc_3B162F</code>
<code>jnb</code>	Jump if Not Below (CF= 0)	<i>jnb address</i>	<code>jnb [edx]</code>

Each jump is preceded by either a test or a `cmp` instructions. However, `jmp` is an unconditional jump and not preceded by anything test or `cmp`.

Examples of Arithmetic Instructions

Instruction	Purpose	Format	Example
add	Add <i>src</i> to <i>dest</i>	<i>add dest, src</i>	add eax, 0x10
sub	Subtract <i>src</i> from <i>dest</i>	<i>sub dest, src</i>	sub eax, ebx
imul	Multiply <i>src</i> by <i>val</i> and store in <i>dest</i>	<i>imul dest, src, val</i>	imul ebx, eax, 5
inc	Increment register by 1	<i>inc register</i>	inc ecx

Examples of Logic Instructions

Instruction	Purpose	Format	Example
xor	Performs Bitwise XOR	<i>xor dest, src</i>	xor eax, eax (Zeroes) xor eax, ebx (xor's)
shl	Shift <i>dest</i> left by <i>src</i> bits	<i>shl dest, src</i>	shl ebx, ecx
and	Performs Bitwise AND	<i>and dest, src</i>	and edx, eax
ror	Rotate <i>dest</i> right by <i>src</i> bits	<i>ror dest, src</i>	ror ecx, edx

test and cmp instructions

Instruction	Purpose	Format	Example
test	Performs a Bitwise AND on the two operands If result is 0, ZF is set Often used with conditional jumps, though less than cmp	<i>test arg1, arg2</i>	test eax, edx
cmp	Compares first operand with second operand by subtraction	<i>cmp arg1, arg2</i>	cmp eax, 0

Jump instructions always come immediate after a test or cmp

Return Values

- EAX register is used to hold the return value of a function call
- The return value could be an integer, eg 0 or 1 or -1 (FFFFFFFF), or, even an address eg, 0x3FA593D3

Thank you