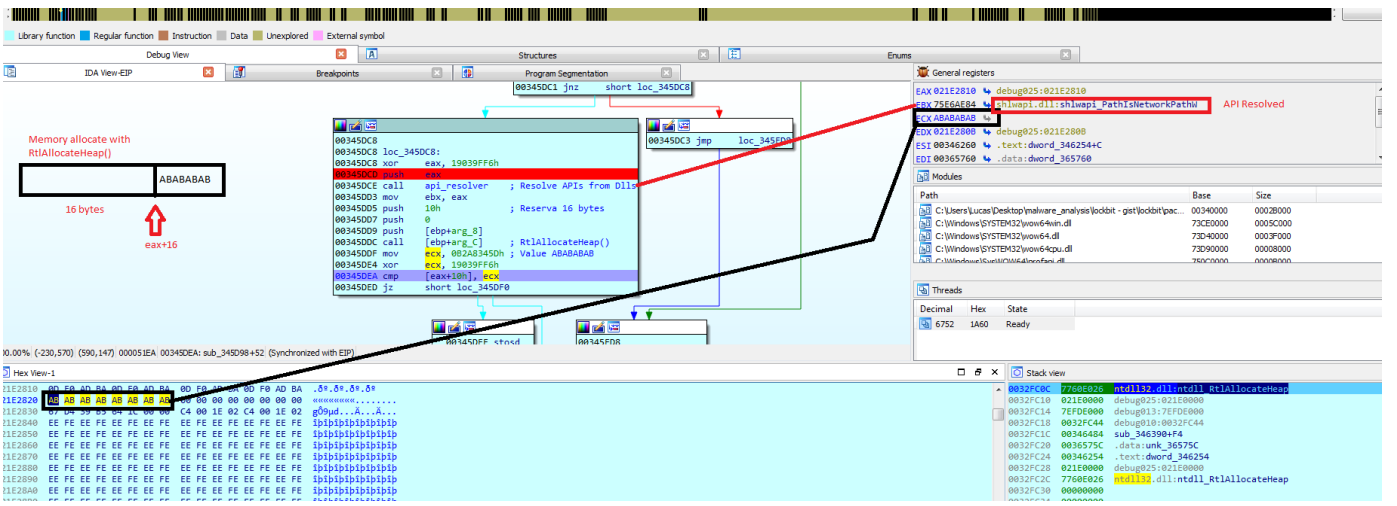
RtlAllocateHeap()
Lockbit 3.0

En este artículo analizamos cómo **Lockbit 3.0** utiliza RtlAllocateHeap() y después de llamar a esta API intenta saber si está siendo analizado mediante un debugger o no. Esta técnica también está explicada en un blog de analista Japonés [1]. Aquí lo encontrarás resumido por lo que si quieres leer el proceso de análisis completo te aconsejo que leas la entrada del investigador Japonés.

Para nuestras pruebas utilizamos el hash: A7782D8D55AE5FBFABBAAAEC367BE5BE
Esta muestra después de su ejecución despliega en memoria lo que es **Lockbit 3.0** sin token de acceso y al volcarla a disco el PE el hash es: E5A0136AC4FE028FEA827458B1A70124.

En la siguiente técnica antidebug si el malware detecta el debugger no se almacena la dirección correcta de la api que acaba de resolver y cuando se llame a esa dirección desde el propio malware como no se habrá correctamente almacenado lanzará una excepción. Es ingenioso ya que no falla directamente después de detectar si está en un entorno de debug o no, sino que falla cuando se invoca a la api.

La técnica con RtlAllocateHeap() funciona del siguiente modo:

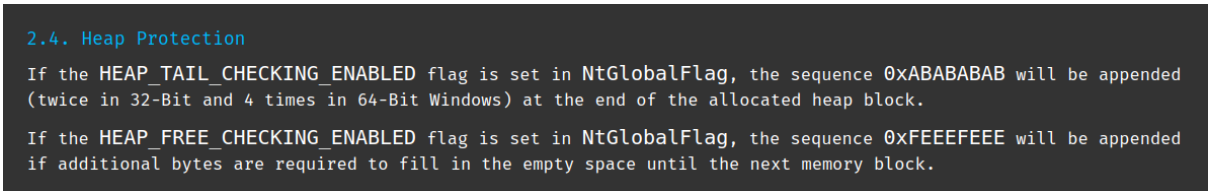


En la imagen anterior podemos ver cómo el malware realiza los siguientes pasos:

1. Llama a la función api_resolver() (renombrada por nosotros) para resolver la dirección de una api.
2. Después de resolver una función y tener su dirección, llama a RtlAllocateHeap(ins, call [ebp+arg_C]) y le pasa como parámetro el tamaño 10h (16 bytes). Por tanto va a crear un buffer para almacenar 16 bytes.
3. Ahora descifrar una cadena con la clave XOR y le da como resultado ABABABAB.

4. Y finalmente compara ABABABAB con los bytes después del byte 10h (16 bytes) que se había reservado.

La pregunta es ¿por qué compara con ABABABAB a partir del final del espacio reservado?. La respuesta la encontramos en el blog de CheckPoint [2], donde nos dice que si el HEAP_TAIL_CHECKING_ENABLED está activado (Flag que se puede activar cuando estamos en un debugger), la secuencia 0xABABABAB se añade al final del buffer reservado en el Heap.



2.4. Heap Protection

If the HEAP_TAIL_CHECKING_ENABLED flag is set in NtGlobalFlag, the sequence 0xABABABAB will be appended (twice in 32-Bit and 4 times in 64-Bit Windows) at the end of the allocated heap block.

If the HEAP_FREE_CHECKING_ENABLED flag is set in NtGlobalFlag, the sequence 0xFEEEFEEE will be appended if additional bytes are required to fill in the empty space until the next memory block.

[1] https://ameblo.jp/reverse-eg-mal-memo/entry-12773724929.html
[2] https://anti-debug.checkpoint.com/techniques/debug-flags.html#manual-checks-heap-flags