Rafael Rivera Tablada

“Análisis de PE con .NET”

- Que es el formato PE
- DOS HEADER
- DOS STUB
- Encabezado PE
- Image File Header
- Image Optional Header
- Tabla de secciones
- Cálculo de MD5, SHA1 y SHA256
- Que es el ImpHash y como se calcula
- Cálculo de entropía de las secciones
- Generando información como Virus Total

SECADMIN2019

“Que es el formato PE”

El formato PE es una estructura de datos que encapsula la información necesaria para el cargador de Windows y así administrar el código ejecutable que le envuelve.

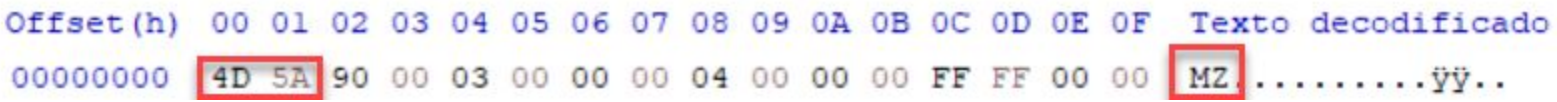
Esto incluye las referencias hacia las bibliotecas de enlace dinámico para el enlazado, la importación y exportación de las tablas de la API.

El Formato PE fue diseñado con la intención de poder generar archivos ejecutables compatibles con cualquier versión del sistema Windows que trabajen sobre procesadores de 32 y 64 bits.

“DOS HEADER”

Esta cabecera está incluida en los archivos tanto DOS como PE, actualmente se ha quedado obsoleta y solo tiene dos campos útiles.

e_magic: Este campo contiene en los primeros 2 Bytes del **“Offset” 0** los valores **0x4D5A**, equivalente a “**MZ**” en ASCII , esta combinación permite identificar este archivo como ejecutable, ya sea EXE o DLL, estas siglas son en honor a **Mark Zbikowsky** uno de los principales desarrolladores de MS-DOS, si un archivo ejecutable no tiene estas siglas no será ejecutado.



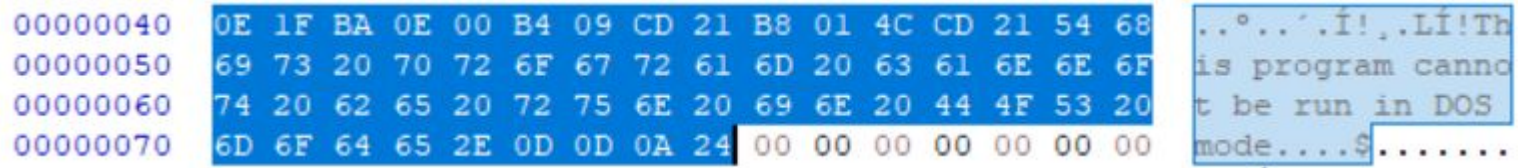
l_fanew: Dirección física o puntero hacia el **NT HEADER** el cual inicia por la firma **PE\x0\x0**. El valor de este campo suele encontrarse en el desplazamiento **0x3C**

“DOS STUB”

Los Bytes que corresponden a las direcciones **0x18** a **0x1B** representan el Offset en el cual encontraremos la información del encabezado MS-DOS ó “MS-DOS Stub”.

Hay que tener en cuenta que los procesadores Intel escriben los datos en orden inverso, a este tipo de codificación se le llama “**little-endian**” por lo que el valor actual de estas direcciones es **0x00000040**.

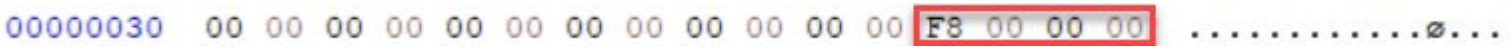
Esto significa que debemos desplazarnos hasta esta dirección para encontrar la siguiente sección del encabezado.



Este valor es una leyenda heredada que sólo indica que el archivo no puede ser ejecutado desde la línea de comandos de MS-DOS y existe en todos los archivos ejecutables PE.

“NT HEADER”

NT HEADER lo encontraremos siempre en la dirección donde apunta **l_fanew** y corresponde a lo que ya es formalmente el encabezado PE.



El contenido de **l_fanew** (**0X3C** a **0x3F**) nos indica que el encabezado PE se encuentra en el offset **0x000000F8** por lo que debemos saltar a esta dirección, la misma comienza siempre con una secuencia de 4 Bytes (32 bits) con valor **0x50450000** (**P E NULL NULL**)

El **NT HEADER** está presente tanto en archivos ejecutables como librerías de enlace dinámico. En caso de no encontrar la firma **PE\x0\x0** el archivo no se ejecutaría, lo mismo pasa con la firma MZ.

“Image File Header (0x00)”

Esta cabecera contiene información general acerca del archivo, se localiza 2 bytes después de **0x000000F8** entre sus campos tenemos:

- Machine
- NumberOfSections
- TimeDateStamp
- PointerToSymbolTable & NumberOfSymbols
- SizeOfOptionalHeader
- Characteristics

Machine: Este campo define el tipo de arquitectura u emulador en el cual se podrá ejecutar el programa. Este puede ser:

IMAGE_FILE_MACHINE_UNKNOWN	0x0	Applicable to any machine type
IMAGE_FILE_MACHINE_AM33	0x1d3	Matsushita AM33
IMAGE_FILE_MACHINE_AMD64	**0x8664**	x64
IMAGE_FILE_MACHINE_ARM	0x1c0	ARM little endian
IMAGE_FILE_MACHINE_ARMNT	0x1c4	ARMv7 (or higher) Thumb mode only

“Image File Header (Continuación)”

Machine: Este campo define el tipo de arquitectura u emulador en el cual se podrá ejecutar el programa. Este puede ser:

IMAGE_FILE_MACHINE_ARM64	0xaa64	ARMv8 in 64-bit mode
IMAGE_FILE_MACHINE_EBC	0xebc	EFI byte code
IMAGE_FILE_MACHINE_I386	0x14c	Intel 386 or later processors and compatible
IMAGE_FILE_MACHINE_IA64	0x200	Intel Itanium processor family
IMAGE_FILE_MACHINE_M32R	0x9041	Mitsubishi M32R little endian
IMAGE_FILE_MACHINE_MIPS16	0x266	MIPS16
IMAGE_FILE_MACHINE_MIPSFPU	0x366	MIPS with FPU
IMAGE_FILE_MACHINE_MIPSFPU16	0x466	MIPS16 with FPU
IMAGE_FILE_MACHINE_POWERPC	0x1f0	Power PC little endian
IMAGE_FILE_MACHINE_POWERPCFP	0x1f1	Power PC with floating point support
IMAGE_FILE_MACHINE_R4000	0x166	MIPS little endian
IMAGE_FILE_MACHINE_SH3	0x1a2	Hitachi SH3
IMAGE_FILE_MACHINE_SH3DSP	0x1a3	Hitachi SH3 DSP
IMAGE_FILE_MACHINE_SH4	0x1a6	Hitachi SH4
IMAGE_FILE_MACHINE_SH5	0x1a8	Hitachi SH5
IMAGE_FILE_MACHINE_THUMB	0x1c2	ARM or Thumb (“interworking”)
IMAGE_FILE_MACHINE_WCEMIPSV2	0x169	MIPS little-endian WCE v2

“Image File Header (Continuación)”

NumberOfSections: Este campo tiene un tamaño de 2 bytes y contiene el número de secciones que tienen el ejecutable teniendo en cuenta que el límite que impone el loader es de 96 secciones o al menos esa cantidad cargará como máximo.

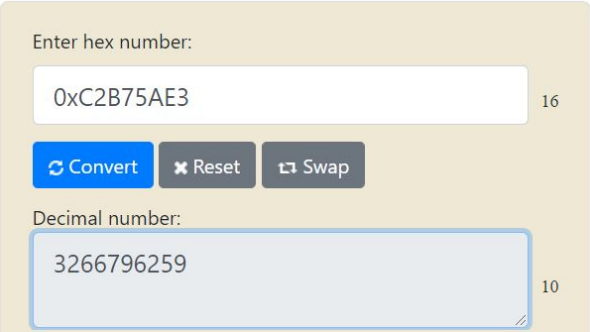
En la imágen anterior se puede observar que CALC.exe posee 6 secciones las cuales serían **.text** .**rdata**, **.data**, .**pdata, .rsrc y .reloc**

Podemos obtener más información en https://www.virustotal.com/

“Image File Header (Continuación)”

NumberOfSections: Este campo tiene un tamaño de 4 bytes e indica el número de segundos transcurridos a partir de 1 de Enero de 1970. Estos segundos nos indicarán la fecha de creación del archivo.

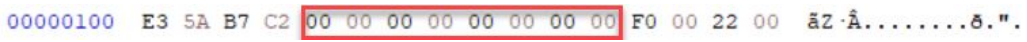
```
00000100  E3 5A B7 C2 00 00 00 00 00 00 00 00 F0 00 22 00  ãZ·Â........ð.".
```



“Image File Header (Continuación)”

PointerToSymbolTable & **NumberOfSymbols**: Estos ocho bytes corresponden a funciones de debug que generalmente están deshabilitados. Los primeros 4 equivalen al puntero para la tabla de símbolos y los 4 restantes al número de símbolos.

Estos campos son empleados por los archivos .obj o COFF FILES, en los archivos ejecutables predeterminadamente estarán en valor 0



“Image File Header (Continuación)”

SizeOfOptionalHeader: Los siguientes 2 Bytes corresponden al tamaño del encabezado opcional que en nuestro ejemplo corresponde a **0x00F0** (**240 Bytes**).

Este valor es utilizado para validar que la estructura actual del archivo permanecerá dentro de los límites definidos al subirlo en memoria.

El término opcional es relativo porque de hecho es obligatorio para todos los ejecutables, en archivos de objetos el valor debe ser cero.

```
00000100  E3 5A B7 C2 00 00 00 00 00 00 00 00 F0 00 22 00  ãZ·Â........ð.".
```

“Image File Header (Continuación)”

Characteristics: Este campo indica los distintos atributos del archivo mediante distintos valores. Cuando el archivo presenta varias características el valor final se obtiene mediante la suma de cada característica.

IMAGE_FILE_RELOCS_STRIPPED 0x0001 Image only, Windows CE, and Windows NT® and later. This indicates that the file does not contain base relocations and must therefore be loaded at its preferred base address. If the base address is not available, the loader reports an error. The default behavior of the linker is to strip base relocations from executable (EXE) files.

IMAGE_FILE_EXECUTABLE_IMAGE 0x0002 Image only. This indicates that the image file is valid and can be run. If this flag is not set, it indicates a linker error.

IMAGE_FILE_LINE_NUMS_STRIPPED 0x0004 COFF line numbers have been removed. This flag is deprecated and should be zero.

IMAGE_FILE_LOCAL_SYMS_STRIPPED 0x0008 COFF symbol table entries for local symbols have been removed. This flag is deprecated and should be zero.

IMAGE_FILE_AGGRESSIVE_WS_TRIM 0x0010 Obsolete. Aggressively trim working set. This flag is deprecated for Windows 2000 and later and must be zero.

IMAGE_FILE_LARGE_ADDRESS_AWARE 0x0020 Application can handle > 2-GB addresses.

“Image File Header (Continuación)”

Characteristics: Este campo indica los distintos atributos del archivo mediante distintos valores. Cuando el archivo presenta varias características el valor final se obtiene mediante la suma de cada característica.

	0x0040	This flag is reserved for future use.
IMAGE_FILE_BYTES_REVERSED_LO	0x0080	Little endian: the least significant bit (LSB) precedes the most significant bit (MSB) in memory. This flag is deprecated and should be zero.
IMAGE_FILE_32BIT_MACHINE	0x0100	Machine is based on a 32-bit-word architecture.
IMAGE_FILE_DEBUG_STRIPPED	0x0200	Debugging information is removed from the image file.
IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP	0x0400	If the image is on removable media, fully load it and copy it to the swap file.
IMAGE_FILE_NET_RUN_FROM_SWAP	0x0800	If the image is on network media, fully load it and copy it to the swap file.
IMAGE_FILE_SYSTEM	0x1000	The image file is a system file, not a user program.
IMAGE_FILE_DLL	0x2000	The image file is a dynamic-link library (DLL). Such files are considered executable files for almost all purposes, although they cannot be directly run.
IMAGE_FILE_UP_SYSTEM_ONLY	0x4000	The file should be run only on a uniprocessor machine.
IMAGE_FILE_BYTES_REVERSED_HI	0x8000	

“Image Optional Header (0x18)”

Esta cabecera proporciona información adicional al loader para que cargue correctamente el ejecutable. Esta información en los archivos COFF es opcional pero en los archivos ejecutables es obligatoria. El tamaño de esta estructura no es fijo, lo define el campo **SizeOfOptionalHeader** (**IMAGE FILE HEADER**).

La misma contiene los siguientes campos:

- **Magic**
- MajorLinkerVersion
- MinorLinkerVersion
- SizeOfCode
- SizeOfInitializedData
- SizeOfUninitializedData
- **AddressOfEntryPoint**
- **BaseOfCode**
- **BaseOfData**
- **ImageBase**
- SectionAlignment
- FileAlignment
- MajorOperatingSystemVersion
- MinorOperatingSystemVersion
- MajorImageVersion

“Image Optional Header (Continuación)”

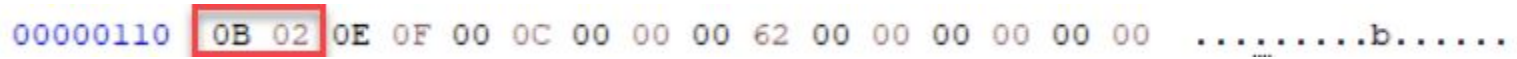
Esta cabecera proporciona información adicional al loader para que cargue correctamente el ejecutable. Esta información en los archivos COFF es opcional pero en los archivos ejecutables es obligatoria. El tamaño de esta estructura no es fijo, lo define el campo **SizeOfOptionalHeader** (**IMAGE FILE HEADER**).

La misma contiene los siguientes campos:

- MinorImageVersion
- MajorSubsystemVersion
- MinorSubsystemVersion
- Win32VersionValue
- SizeOfImage
- SizeOfHeaders
- CheckSum
- Subsystem
- DllCharacteristics
- SizeOfStackReserve
- SizeOfStackCommit
- SizeOfHeapReserve
- SizeOfHeapCommit
- LoaderFlags
- NumberOfRvaAndSizes

“Image Optional Header (Continuación)”

Esta cabecera proporciona información adicional al loader para que cargue correctamente el ejecutable. Esta información en los archivos COFF es opcional pero en los archivos ejecutables es obligatoria. El tamaño de esta estructura no es fijo, lo define el campo **SizeOfOptionalHeader** (**IMAGE FILE HEADER**).



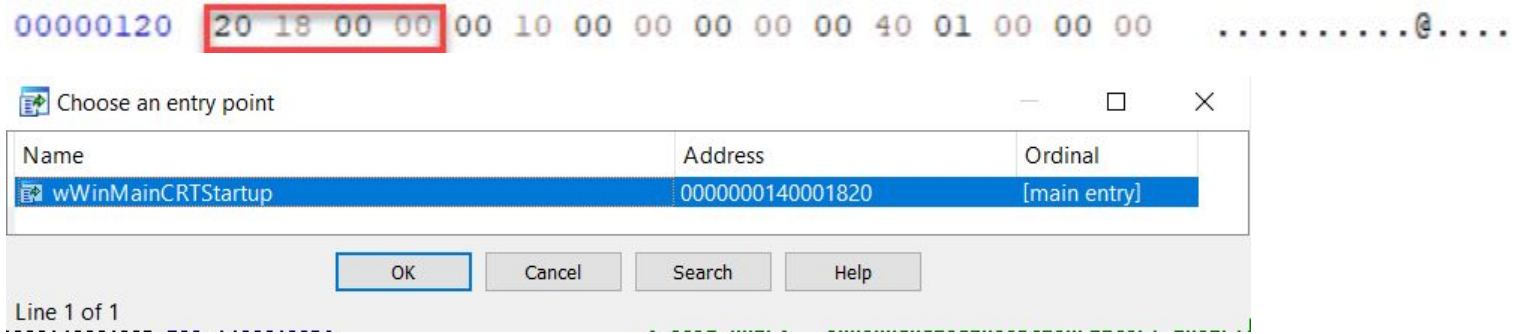
Magic: (No confundir con e_magic).
Este campo tiene un tamaño de 2 bytes y determina si el ejecutable es para sistemas x86 o x64. Las constantes que lo determinan son:

- 0x010B PE32 (x86)
- 0x020B PE32+ (x64)

“Image Optional Header (Continuación)”

AddressOfEntryPoint: Los siguientes 4 Bytes indican la dirección de entrada o EntryPoint (EP). Esta dirección es un RVA (Relative Virtual Address), significa que es una dirección relativa con respecto de la dirección base.

Para encontrar en memoria esta dirección debemos sumar su valor al valor de la dirección base. En nuestro ejemplo el EP es 0x00001820. Esta es una de las secciones más importantes ya que en archivos ejecutables indica cómo encontrar la primera instrucción de ejecución en memoria. En archivos DLL este parámetro es opcional y de no usarse su valor debe ser 0.



“Image Optional Header (Continuación)”

BaseOfCode & BaseOfData: Estos 4 Bytes proporcionan un RVA el cual indica o corresponde al inicio de las secciones de código y datos respectivamente.

```
00000120  20 18 00 00 00 10 00 00 00 00 00 40 01 00 00 00   ..........@....
```

BaseOfData: Los 4 Bytes de este campo indican la dirección relativa al inicio de los datos, es usada sólo para archivos PE32

ImageBase:Este campo tiene un tamaño de 4 bytes y proporciona la dirección de preferencia donde se cargará el ejecutable. Este campo debe ser múltiplo de 0x10000. Es muy común ver en DLL este campo con valor 0x10000000 y en ejecutables 0x00400000.

```
00000120  20 18 00 00 00 10 00 00 00 00 00 40 01 00 00 00   ..........@....
```

“Tabla de Secciones”

ImageBase:Esta sección sigue inmediatamente con el primer Byte después de terminar el encabezado opcional (de existir). Debido a que no existe un puntero que indique exactamente esta dirección, es necesario que calcularla utilizando la dirección de inicio de **OptionalHeaders** + **SizeOfOptionalHeaders (240 Bytes)**.

El número de secciones que siguen corresponde al valor de **NumberOfSections** (en nuestro ejemplo son 0x06). Las entradas en estas secciones se numeran comenzando con el valor “1”. Cada encabezado de sección contiene la siguiente información para un total de 40 Bytes por entrada.

“Cálculo de MD5, SHA1 y SHA256”

A veces cuando analizamos un PE es necesario comprobar que el mismo no ha sido alterado en su totalidad o que sus secciones no se han modificado, para ello deberíamos generar lo que llamamos Checksum (Suma de comprobación) o hashes. El Checksum es el resultado de aplicar los algoritmos MD5, SHA1, SHA256 entre otros a un archivo, produciendo una secuencia de letras y números de un tamaño fijo.

Estos hashes son utilizados por los antivirus para generar una huella digital que identifique el ejecutable en cuestión, cabe destacar que en algoritmos como MD5 pueden existir colisiones que den como resultado hashes iguales para archivos totalmente diferentes.

Cálculo de MD5 en C#

```
public static string GenerateSHA1(byte[] input)
{
    SHA1 sha1 = SHA1.Create();
    StringBuilder sb = new StringBuilder();
    byte[] hash = sha1.ComputeHash(input);

    for (int i = 0; i < hash.Length; i++)
        sb.Append(hash[i].ToString("x2"));

    return sb.ToString();
}
```

Cálculo de SHA-1 en C#

```
public static string GenerateSHA1(byte[] input)
{
    SHA1 sha1 = SHA1.Create();
    StringBuilder sb = new StringBuilder();
    byte[] hash = sha1.ComputeHash(input);

    for (int i = 0; i < hash.Length; i++)
        sb.Append(hash[i].ToString("x2"));

    return sb.ToString();
}
```

“Que es el ImpHash y como se calcula”

El 23 de Enero del 2014 la compañía MANDIANT especializada en ofrecer productos y servicios de seguridad informática, publica en su blog el artículo “Tracking Malware with Import Hashing” donde muestran uno de los métodos que ellos utilizan para identificar y clasificar malware.

Una forma única que Mandiant da seguimiento a los grupos específicos de amenazas es realizando un seguimiento de las importaciones de un fichero ejecutable PE.

Las importaciones son las funciones que una pieza de software utiliza para llamar a otros archivos (por lo general varias DLL que ofrecen funcionalidad para el sistema operativo Windows).

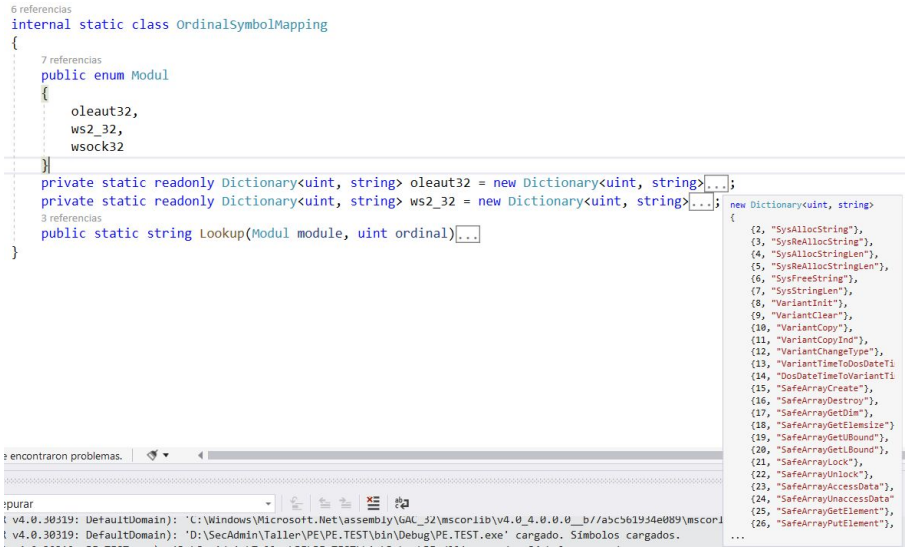
Para realizar un seguimiento de estas importaciones, Mandiant crea un hash basado en nombres de biblioteca / API y su orden específico dentro del ejecutable. Nos referimos a este convenio como un “Imphash” (de “importación de hash”). Debido a la forma en que se genera la tabla de importación de un PE (y, por tanto, cómo se calcula su Imphash), podemos usar el valor Imphash para identificar muestras de malware relacionados.

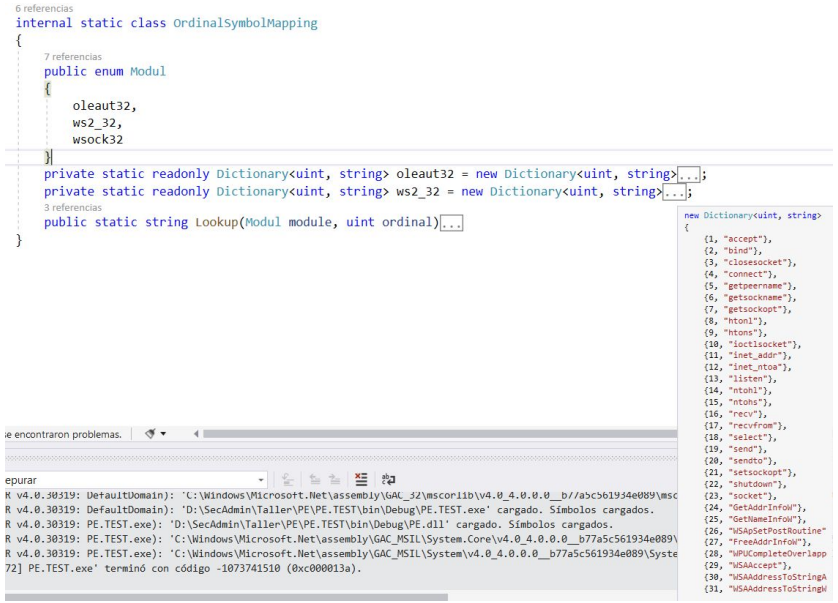
“Que es el ImpHash y como se calcula”

Para el cálculo del ImpHash se ha de seguir la siguiente convención:

- Resolver ordinales a nombres de funciones cuando aparecen
- Convertir nombres de DLL y nombres de funciones a minúsculas
- Eliminar las extensiones de archivo de los nombres de módulos importados
- Construir y almacenar la cadena en minúsculas. en una lista ordenada
- Generar el hash MD5 de la lista ordenada

“Que es el ImpHash y como se calcula”





SECADMIN2019

“Cálculo de entropía de las secciones”

En el ámbito de la teoría de la información, la entropía de la información y entropía de Shannon, mide la incertidumbre de una fuente de información.

Podríamos considerar la entropía como la cantidad de información promedio que contienen los símbolos usados, los símbolos con menor probabilidad son los que aportan mayor información.

- Una muestra completamente homogénea tiene tiene entropía 0
- Una muestra igualmente distribuida tiene entropía 1

En general la fórmula de la entropía es:

$$H = -\sum_{i=1}^{N} P(i) \log P(i)$$

Cálculo de entropía de secciones en C#

```
public static double GetEntropy(this byte[] input) {

    double Length = input.Length;
    var result = 0d;
    int[] _map = new int[256];
    _map.Fill(0);

    for (var i = 0; i < (int)Length; i++)
        _map[input[i]]++;

    foreach(var item in _map)
    {
        if (item != 0)
        {
            var frecuency = item / Length;
            result -= frecuency * Math.Log(frecuency, 2);
        }
    }
    return  double.IsNaN(result) || double.IsInfinity(result) ? 0 : Math.Round(result,2);
}
```

SECADMIN2019