

Hoja de Trabajo: Análisis de Malware

Javier Mombiola

Carnet: 20067

23 de febrero del 2024

In []:

```
import pefile
```

Parte 1 – análisis estático

In []:

```
exe_path = './sample_vg655_25th.exe'
```

1. Utilice la herramienta pefile para examinar el PE header y obtenga las DLL y las APIs

In []:

```
def get_imports_info(exe_path):
    try:
        pe = pefile.PE(exe_path)
        print("DLLs importadas:")
        for entry in pe.DIRECTORY_ENTRY_IMPORT:
            print("\t", entry.dll.decode())

        print("\nAPIs importadas:")
        for entry in pe.DIRECTORY_ENTRY_IMPORT:
            print("\tDLL:", entry.dll.decode())
            for imp in entry.imports:
                print("\t\t", imp.name.decode())
    except Exception as e:
        print("Error:", e)

get_imports_info(exe_path)
```

DLLs importadas:

- KERNEL32.dll
- USER32.dll
- ADVAPI32.dll
- MSVCRT.dll

APIs importadas:

DLL: KERNEL32.dll

- GetFileAttributesW
- GetFileSizeEx
- CreateFileA
- InitializeCriticalSection
- DeleteCriticalSection
- ReadFile
- GetFileSize
- WriteFile
- LeaveCriticalSection
- EnterCriticalSection
- SetFileAttributesW
- SetCurrentDirectoryW
- CreateDirectoryW
- GetTempPathW
- GetWindowsDirectoryW
- GetFileAttributesA
- SizeofResource
- LockResource
- LoadResource
- MultiByteToWideChar
- Sleep
- OpenMutexA
- GetFullPathNameA
- CopyFileA
- GetModuleFileNameA
- VirtualAlloc
- VirtualFree
- FreeLibrary
- HeapAlloc
- GetProcessHeap
- GetModuleHandleA
- SetLastError
- VirtualProtect
- IsBadReadPtr
- HeapFree
- SystemTimeToFileTime
- LocalFileTimeToFileTime
- CreateDirectoryA
- GetStartupInfoA
- SetFilePointer
- SetFileTime
- GetComputerNameW
- GetCurrentDirectoryA
- SetCurrentDirectoryA
- GlobalAlloc
- LoadLibraryA
- GetProcAddress
- GlobalFree
- CreateProcessA
- CloseHandle
- WaitForSingleObject
- TerminateProcess
- GetExitCodeProcess
- FindResourceA

DLL: USER32.dll

- MsgPrintA

DLL: ADVAPI32.dll

- CreateServiceA
- OpenServiceA
- StartServiceA
- CloseServiceHandle
- CryptReleaseContext
- RegCreateKeyW
- RegSetValueExA
- RegQueryValueExA
- RegCloseKey
- OpenSCManagerA

DLL: MSVCRT.dll

- realloc
- fclose
- fwrite
- fread
- fopen
- sprintf
- rand
- srand
- strcpy
- memset
- strlen
- wscat
- wcslcn
- __CxxFrameHandler
- 7739YAXPAIXZ
- memcpy
- _except_handler3
- _local_unwind2
- wcschr
- swprintf
- 7729YAPAXIQZ
- memcpy
- strcmp
- strrchr
- __p_argv
- __p_argc
- _stricmp
- free
- malloc
- ??exception@QQAEABAV0@0Z
- ??exception@QQAEABQZ
- ??exception@QQAEABQ0QZ
- _CxxThrowException
- calloc
- strcat
- _mbstr
- ??type_info@QUAE@XZ
- _exit
- _XcptFilter
- exit
- _acmdln
- __getmainargs
- _initterm
- __setusermatherr
- _adjust_fdiv
- __p_commode
- __p_fmode
- __set_app_type
- _controlfp

• No hay indicios inmediatos de comportamiento sospechoso en la cantidad de DLLs importadas ni en las APIs llamadas. El número de DLLs importadas (cuatro) no es por sí mismo alarmante, y las APIs llamadas son funciones estándar del sistema operativo Windows.

2. Obtenga la información de las secciones del PE Header. ¿Qué significa que algunas secciones tengan como parte de su nombre "upx"? Realice el procedimiento de desempaquetado para obtener las llamadas completas de las APIs.

• Las secciones con nombres que contienen "upx" generalmente indican que el archivo ha sido comprimido o empaquetado utilizando la herramienta de compresión UPX (Ultimate Packer for eXecutables).

In []:

```
def get_sections_info(exe_path):
    try:
        pe = pefile.PE(exe_path)
        print("Secciones del PE Header:")
        for section in pe.sections:
            print("\tNombre:", section.Name.decode().strip('\x00'))
            print("\tVirtualAddress:", hex(section.VirtualAddress))
            print("\tVirtualSize:", hex(section.Misc.VirtualSize))
            print("\tSizeOfRawData:", hex(section.SizeOfRawData))
            print("\tPointerToRawData:", hex(section.PointerToRawData))
            print("\tCharacteristics:", hex(section.Characteristics))
            print()

            # Desempaquetar secciones con "upx" en el nombre
            if b"upx" in section.Name.lower():
                print("Desempaquetando sección:", section.Name.decode().strip('\x00'))
                unpacked_data = section.get_data()
                print("Sección desempaquetada:", unpacked_data)
    except Exception as e:
        print("Error:", e)

get_sections_info(exe_path)
```

Secciones del PE Header:

Nombre: .text

VirtualAddress: 0x1000

VirtualSize: 0x6000

SizeOfRawData: 0x7000

PointerToRawData: 0x1000

Characteristics: 0x60000020

Nombre: .rdata

VirtualAddress: 0x8000

VirtualSize: 0x570

SizeOfRawData: 0x600

PointerToRawData: 0x8000

Characteristics: 0x40000040

Nombre: .data

VirtualAddress: 0xe000

VirtualSize: 0x105

SizeOfRawData: 0x200

PointerToRawData: 0xe000

Characteristics: 0xc0000040

Nombre: .rsrc

VirtualAddress: 0x10000

VirtualSize: 0x349fa0

SizeOfRawData: 0x34a000

PointerToRawData: 0x10000

Characteristics: 0x40000040

3. Según el paper "Towards Understanding Malware Behaviour by the Extraction of API Calls", ¿en que categoría sospechosas pueden clasificarse estos ejemplos en base a algunas de las llamadas a las APIs que realizan? Muestre una tabla con las APIs sospechosas y la categoría de malware que el artículo propone

Categoría de Malware	API Sospechosa
Search files to infect	CreateProcessA
	LoadLibraryA
	TerminateProcess
	RegCreateKeyW
Copy/Delete files	CryptReleaseContext
	CopyFileA
	DeleteCriticalSection
	RemoveDirectoryA
Get file information	DeleteFileA
	GetFileAttributesW
	GetFileSizeEx
	GetFileAttributesA
Move Files	GetFileSize
	GetTempPathW
	GetWindowsDirectoryW
	GetModuleFileNameA
Read /Write files	GetComputerNameW
	GetCurrentDirectoryA
	GetFullPathNameA
	GetStartupInfoA
Change file attributes	MoveFileExA
	ReadFile
Move Files	WriteFile
	SetFileAttributesW
	SetCurrentDirectoryW
	SetFilePointer
Read /Write files	SetFileTime

4. Investigue algunas de las funciones y DLLs utilizadas por el ejecutable e indique su propósito.

DLLs

- KERNEL32.dll: Contiene funciones relacionadas con la gestión de memoria, administración de procesos y archivos, manipulación de subprocesos, y otras operaciones del sistema operativo Windows.
- USER32.dll: Proporciona funciones relacionadas con la creación y gestión de interfaces de usuario y ventanas, como la manipulación de mensajes, la creación de ventanas y controles, y la manipulación del cursor y del teclado.
- ADVAPI32.dll: Contiene funciones relacionadas con la administración de servicios, seguridad y registro del sistema operativo Windows.
- MSVCRT.dll: Biblioteca de tiempo de ejecución de Microsoft (Microsoft C Runtime Library) que contiene funciones y rutinas estándar para manipular cadenas, memoria, archivos y otros aspectos relacionados con la ejecución de programas escritos en lenguaje C o C++.

Funciones

- CreateProcessA: Esta API se utiliza para crear un nuevo proceso y su hilo primario. Es comúnmente utilizada por programas para ejecutar otros programas o procesos secundarios.
- ReadFile: La función ReadFile se utiliza para leer datos desde un archivo o dispositivo de entrada. Es parte de la API de manipulación de archivos de Windows y se utiliza para la lectura de archivos en modo síncrono o asíncrono.
- WriteFile: La función WriteFile se utiliza para escribir datos en un archivo o dispositivo de salida. Al igual que ReadFile, es parte de la API de manipulación de archivos de Windows y se utiliza para escribir datos en archivos en modo síncrono o asíncrono.
- RegCreateKeyW: Esta función se utiliza para crear una nueva subclave o abrir una clave existente en el registro de Windows. Es parte de la API de manipulación del registro de Windows y se utiliza para realizar operaciones relacionadas con la configuración y la persistencia de datos en el registro del sistema.
- Con la información recopilada hasta el momento, indique para el archivo "sample_vg655_25th.exe" si es sospechoso o no, y cual podría ser su propósito.

Basándonos en la información proporcionada hasta el momento, sabemos lo siguiente:

- El archivo sample_vg655_25th.exe importa DLLs estándar del sistema operativo Windows, como KERNEL32.dll, USER32.dll, ADVAPI32.dll y MSVCRT.dll. Además, hace uso de diversas APIs comunes para la gestión de archivos, procesos, registros y memoria.
- Las APIs utilizadas en sample_vg655_25th.exe están relacionadas principalmente con operaciones de gestión de recursos del sistema, lo que sugiere un comportamiento típico de software legítimo. Las DLLs importadas son bibliotecas estándar del sistema operativo Windows, lo cual refuerza la impresión de que el archivo podría ser legítimo.

Pero basándonos únicamente en esta información, no podemos determinar de forma concluyente si el archivo es sospechoso o no. Su propósito potencial podría ser el de una aplicación legítima.

6. Para el archivo "sample_vg655_25th.exe" obtenga el HASH usando el algoritmo SHA256.

In []:

```
import hashlib

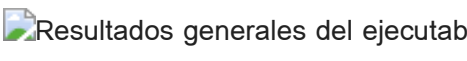
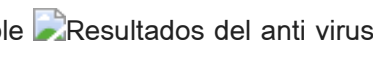
def calcular_hash_sha256(archivo):
    with open(archivo, 'rb') as f:
        contenido = f.read()
        hash_sha256 = hashlib.sha256(contenido).hexdigest()
    return hash_sha256

hash_sha256 = calcular_hash_sha256(exe_path)
print("El hash SHA256 del archivo {} es: {}".format(exe_path, hash_sha256))

El hash SHA256 del archivo ./sample_vg655_25th.exe es: e0d1ebfbc9eb5bea545af4d01bf5f1071661840480439c6e5babe8e08e41aa
```

Parte 2 – análisis dinámico

7. Utilice la plataforma de análisis dinámico <https://www.hybrid-analysis.com> y cargue el archivo "sample_vg655_25th.exe". ¿Se corresponde el HASH de la plataforma con el generado? Si encontró información, indique cual. Incluya posibles capturas de pantalla, etc. ¿Se corresponde la información encontrada con el análisis realizado en el punto 5

Resultados generales del ejecutable Resultados del anti virus

Como podemos observar con las dos imagenes anteriores, el ejecutable resulta ser un archivo sospechoso y malicioso, con un resultado de amenaza del 100%. Como habiamos visto anteriormente en el punto 5, esto no se habia podido confirmar, ya que las funciones que utilizaba eran normales y parecia que el proposito del archivo no era maligno. En conclusion, por este es que son necesarios analisis como estos, para poder descartar malwares aunque parezcan no ser malignos.