Sociedad Andaluza para el Desarrollo de las Telecomunicaciones S.A.
CONSEJERÍA DE ECONOMÍA, INNOVACIÓN, CIENCIA Y EMPLEO

seguridad + Y CONFIANZA DIGITAL | AndalucíaCERT CENTRO DE SEGURIDAD TIC

Informe de divulgación

Laboratorio para el análsis de malware (II): Análisis manual - Análisis estático

Tipo de documento:	*Informe*
Autor del documento:	*AndalucíaCERT*
Código del Documento:	*CERT-IF-7893-150603*
Edición:	*0*
Categoría	*Público*
Fecha de elaboración:	*03/06/2015*
N° de Páginas	*1 de 17*

1 TABLA DE CONTENIDOS

2 OBJETIVO

Este es el segundo de una serie de documentos que describen la implementación de un laboratorio para el análisis de malware. Este laboratorio permitirá estudiar una muestra en un entorno controlado, y que cuente con las herramientas para obtener gran cantidad de información.

3 ALCANCE

Este documento va dirigido al personal de la Junta de Andalucía y al público en general. Pretende aportar las nociones necesarias para entender y tener conocimiento sobre las técnicas que se suelen usar para combatir las amenazas de malware.

En este documento se describen algunas de las herramientas que se pueden usar a la hora de realizar un análisis estático sobre muestras en pruebas manuales.

4 ANÁLISIS MANUAL

En el análisis manual se busca obtener información en profundidad del comportamiento de una muestra con herramientas manuales. Suele aportar más información que el análisis automático ya que el investigador puede indicar opciones específicas que las herramientas automáticas no tienen en cuenta. El análisis manual en profundidad utilizando técnicas de ingeniería inversa es la única manera de conocer el comportamiento completo de una pieza de malware.

Existen dos métodos principales a la hora de analizar malware:

- **Análisis estático:** se recoge información de la muestra sin ejecutarla en el sistema.
- **Análisis dinámico:** se recoge información del malware ejecutando la muestra en el sistema.

4.1 ANÁLISIS ESTÁTICO

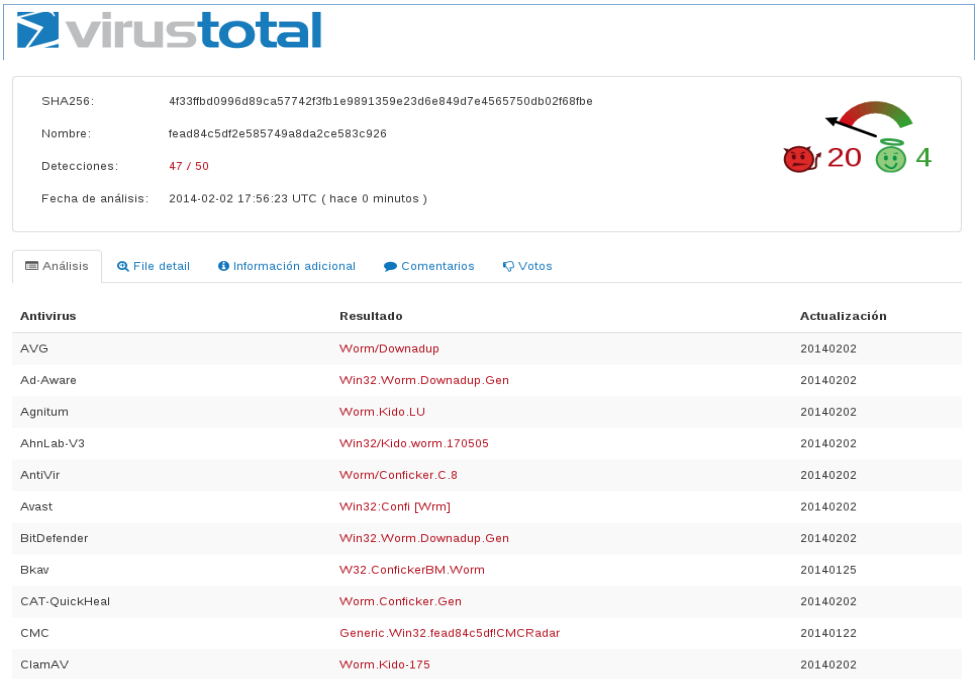
Se describen herramientas que se pueden usar para obtener información de una muestra de malware sin ejecutarla en el sistema. Se analiza el tipo de fichero, el código y la estructura del programa, si está empaquetado, etc. Estos datos pueden proporcionar información sobre las funcionalidades del malware y ayudar a generar firmas de detección.

4.1.1 Antivirus

Uno de los primeros pasos que se recomiendan dar en un análisis es pasar la muestra por un sistema antivirus. Los antivirus detectan código malicioso principalmente en base a firmas, cadenas de código sospechoso conocido, y en base a comportamiento (heurística). Diferentes antivirus tienen bases de datos de firmas y algoritmos heurísticos diferentes, los cuales puede que ya identifiquen el fichero como malicioso y nos aporte información sobre sus características. Es por ello que es útil pasar varios antivirus diferentes contra la misma muestra de malware.

La herramienta VirusTotal (www.virustotal.com) permite subir un fichero y escanearlo por varios motores antivirus. La herramienta generará un reporte en el que proporcionará el total de motores que detectan la muestra como maliciosa, el nombre del malware, y alguna otra información de interés.

Por ejemplo, se puede ver el resultado de analizar una muestra del gusano Conficker. Como se puede comprobar lo detectan como malicioso un total de 47 motores de antivirus:



virustotal

SHA256: 4f33ffbd0996d89ca57742f3fb1e9891359e23d6e849d7e4565750db02f68fbe
Nombre: fead84c5df2e585749a8da2ce583c926
Detecciones: 47 / 50
Fecha de análisis: 2014-02-02 17:56:23 UTC (hace 0 minutos)

20 4

Análisis | File detail | Información adicional | Comentarios | Votos

Antivirus	Resultado	Actualización
AVG	Worm/Downadup	20140202
Ad-Aware	Win32.Worm.Downadup.Gen	20140202
Agnitum	Worm.Kido.LU	20140202
AhnLab-V3	Win32/Kido.worm.170505	20140202
AntiVir	Worm/Conficker.C.8	20140202
Avast	Win32:Confi [Wrm]	20140202
BitDefender	Win32.Worm.Downadup.Gen	20140202
Bkav	W32.ConfickerBM.Worm	20140125
CAT-QuickHeal	Worm.Conficker.Gen	20140202
CMC	Generic.Win32.fead84c5df!CMCRadar	20140122
ClamAV	Worm.Kido-175	20140202

4.1.2 Strings

El programa "strings" muestra las secuencia de caracteres que contiene un programa. Estas cadenas pueden ser mensajes de error, URLs, direcciones IP, claves de registro, funciones, etc. Las strings de un fichero pueden dar información sobre sus funcionalidades.

El programa "strings" suele venir de serie en los sistemas Linux en el paquete binutils junto con otras herramientas del sistema. Podemos ver la ayuda del programa ejecutando strings -h:

```
# strings -h
Usage: strings [option(s)] [file(s)]
 Display printable strings in [file(s)] (stdin by default)
 The options are:
  -a - --all                Scan the entire file, not just the data section
  -f --print-file-name      Print the name of the file before each string
  -n --bytes=[number]       Locate & print any NUL-terminated sequence of at
  -<number>                   least [number] characters (default 4).
  -t --radix={o,d,x}        Print the location of the string in base 8, 10 or 16
  -o                        An alias for --radix=o
  -T --target=<BFDNAME>     Specify the binary file format
  -e --encoding={s,S,b,l,B,L} Select character size and endianness:
```

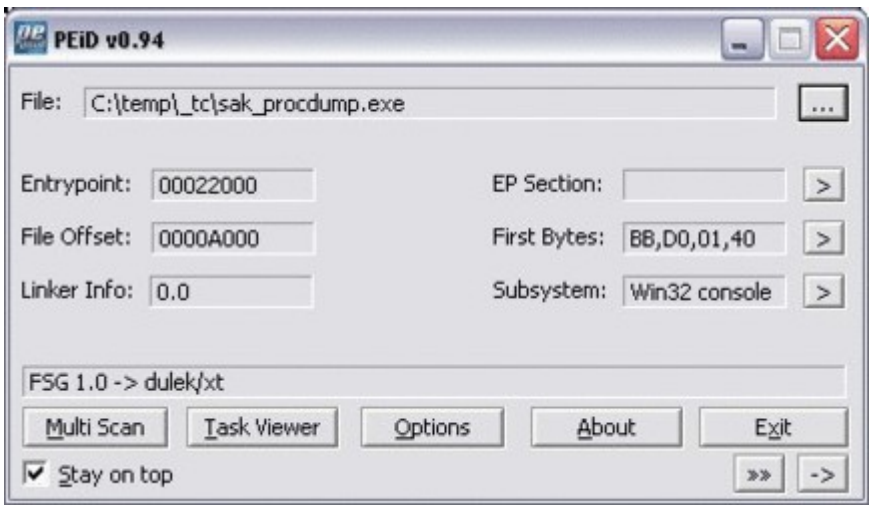
```
                          s = 7-bit, S = 8-bit, {b,l} = 16-bit, {B,L} = 32-bit
  @<file>                 Read options from <file>
  -h --help               Display this information
  -v -V --version         Print the program's version number
strings: supported targets: elf64-x86-64 elf32-i386 elf32-x86-64 a.out-i386-linux
pei-i386 pei-x86-64 elf64-l1om elf64-k1om elf64-little elf64-big elf32-little elf32-
big plugin srec symbolsrec verilog tekhex binary ihex
Report bugs to <http://www.sourceware.org/bugzilla/>
```

Para ver las cadenas de un fichero simplemente ejecutamos el programa strings con el fichero como parámetro:

```
$ strings ficheroMuestra
AUT]
t      WVS
NWVS
u7WPS
u&WVS
_^[]
IsBadWritePtr
IsBadStringPtrA
GetCurrentThreadId
SetLastError
Sleep
GetACP
IsDBCSLeadByte
LoadLibraryA
...
```

4.1.3 PeiD

PeiD es una herramienta gráfica para usar con los ficheros ejecutables de Windows (Portable Executable). La principal funcionalidad de PeiD es la detección del empaquetador o compilador utilizado para construir una aplicación, facilitando el análisis del fichero empaquetado.



Cuando un programa está empaquetado es necesario desempaquetarlo primero para examinar en más detalle el fichero. Por ejemplo, el programa strings no sería capaz de ver las secuencias de caracteres de un programa empaquetado.

Aunque el proyecto está descontinuado, esta herramientas es de las mejores que hay disponibles para la detección de empaquetadores y compiladores. Actualmente detecta más de 470 firmas diferentes en archivos PE.

Puede descargarse de la siguiente URL:

- http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml

Una vez descargada hay que actualizar las firmas, ya que inicialmente el fichero de éstas se encuentra vacío. Es necesario reemplazar el fichero userdb.txt por unos de los siguientes:

- http://handlers.sans.org/jclausing/userdb.txt
- http://reverse-engineering-scripts.googlecode.com/files/UserDB.TXT

Otras funcionalidades de PeiD:

- Visor de secciones de un fichero.
- Desensamblador de archivos PE.
- Detalles de un archivo PE, imports, exports y visores TLS.
- Analizador criptográfico.
- Visor y controlador de tareas.
- Escáner y detección de archivos no reconocidos.
- Escaneo de múltiples ficheros y directorios con recursión.
- Plugins.
- Opciones de escaneo heurístico.

4.1.4 Pedump

Pedump es una implementación en el lenguaje ruby de la herramienta de volcado de ficheros PE win32.

Soporta los siguientes formatos:

- DOS MZ EXE.
- win16 NE.
- win32 PE.
- win64 PE.

Puede volcar la siguiente información de un fichero:

- Las cabeceras MZ/NE/PE.
- Cabecera.
- Secciones de un fichero.
- Recursos.
- Strings.
- Imports y exports.
- Estructura de información de versionado (VS_VERSIONINFO).
- Detección del empaquetador y compilador.

Para instalarlo se puede usar el gestor de gemas de ruby:

```
# gem install pedump

Fetching: multipart-post-1.1.5.gem (100%)
Fetching: progressbar-0.21.0.gem (100%)
Fetching: awesome_print-1.2.0.gem (100%)
Fetching: iostruct-0.0.4.gem (100%)
Fetching: zhexdump-0.0.2.gem (100%)
Fetching: pedump-0.5.0.gem (100%)
Successfully installed multipart-post-1.1.5
Successfully installed progressbar-0.21.0
Successfully installed awesome_print-1.2.0
Successfully installed iostruct-0.0.4
Successfully installed zhexdump-0.0.2
Successfully installed pedump-0.5.0
6 gems installed
Installing ri documentation for multipart-post-1.1.5...
Installing ri documentation for progressbar-0.21.0...
Installing ri documentation for awesome_print-1.2.0...
Installing ri documentation for iostruct-0.0.4...
Installing ri documentation for zhexdump-0.0.2...
```

```
Installing ri documentation for pedump-0.5.0...
Installing RDoc documentation for multipart-post-1.1.5...
Installing RDoc documentation for progressbar-0.21.0...
Installing RDoc documentation for awesome_print-1.2.0...
Installing RDoc documentation for iostruct-0.0.4...
Installing RDoc documentation for zhexdump-0.0.2...
Installing RDoc documentation for pedump-0.5.0...
```

O clonar el repositorio del proyecto en github.com:

```
git clone http://github.com/zed-0xff/pedump
```

Para ejecutarlo simplemente ejecutamos pedump seguido del fichero a analizar:

```
pedump filename.exe
```

Las opciones disponibles son las siguientes:

```
# pedump -h
Usage: pedump [options]
        --version                    Print version information and exit
    -v, --verbose                    Run verbosely
                                     (can be used multiple times)
    -q, --quiet                      Silent any warnings
                                     (can be used multiple times)
    -F, --force                      Try to dump by all means
                                     (can cause exceptions & heavy wounds)
    -f, --format FORMAT              Output format:
bin,c,dump,hex,inspect,table,yaml
                                     (default: table)
        --mz
        --dos-stub
        --rich
        --pe
        --ne
        --data-directory
    -S, --sections
        --tls
        --security
    -s, --strings
    -R, --resources
        --resource-directory
    -I, --imports
    -E, --exports
    -V, --version-info
        --packer
        --deep                       packer deep scan, significantly slower
    -P, --packer-only                packer/compiler detect only,
                                     mimics 'file' command output
    -r, --recursive                  recurse dirs in packer detect
        --all                        Dump all but resource-directory (default)
        --va2file VA                 Convert RVA to file offset

    -W, --web                        Uploads files to a http://pedump.me
```

```
                                    for a nice HTML tables with image previews,
                                    candies & stuff
  -C, --console                   opens IRB console with specified file loaded
```

También hay disponible una versión web en la siguiente URL:

- http://pedump.me

Permite subir un fichero y realizar un análisis a través de la aplicación web.

filename	a7ac0003d2f36841dc217fe78fcdeeed_spyeye
size	413696 (0x65000)
md5	a7ac0003d2f36841dc217fe78fcdeeed
type	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed
mimetype	application/x-dosexec
clamav	Trojan.Spy.SpyEyes-8 FOUND
virustotal	→ scan with virustotal.com

Packer / Compiler

MASM/TASM - sig2(h)

Sections

name	va	vsize	raw size	flags
.text	0x1000	0x2520	0x2600	R-X CODE
.rdata	0x4000	0x6b4	0x800	R-- IDATA
.data	0x5000	0x4b59c	0x4b600	RW- IDATA
.rsrc	0x51000	0x128fc	0x12a00	RWX IDATA
.reloc	0x64000	0x4e4	0x600	R-- IDATA DISCARDABLE

4.1.5 Peframe

Peframe es una herramienta para realizar un análisis estático de archivos de malware PE (Portable Executables).

Las características principales son las siguientes:

- Hash MD5 y SHA1.
- Atributos del fichero PE.
- Información de la versión y metadatos.
- Firma de identificación del PE.
- Detección de técnicas Anti Virtual Machine.
- Detección de técnicas Anti Debug.
- Analizador de secciones.
- DLLs importadas y funciones API.
- Búsqueda de secciones y APIs sospechosas (Anti Debug).
- Volcado de toda la información del fichero.
- Extracción de strings.
- Extracción del nombre del fichero y URLs.

La página del proyecto es la siguiente:

- https://github.com/guelfoweb/peframe

En la antigua wiki del proyecto aparecen algunos ejemplos de uso:

- https://code.google.com/p/peframe/wiki/Example

Las opciones que presenta son las siguientes:

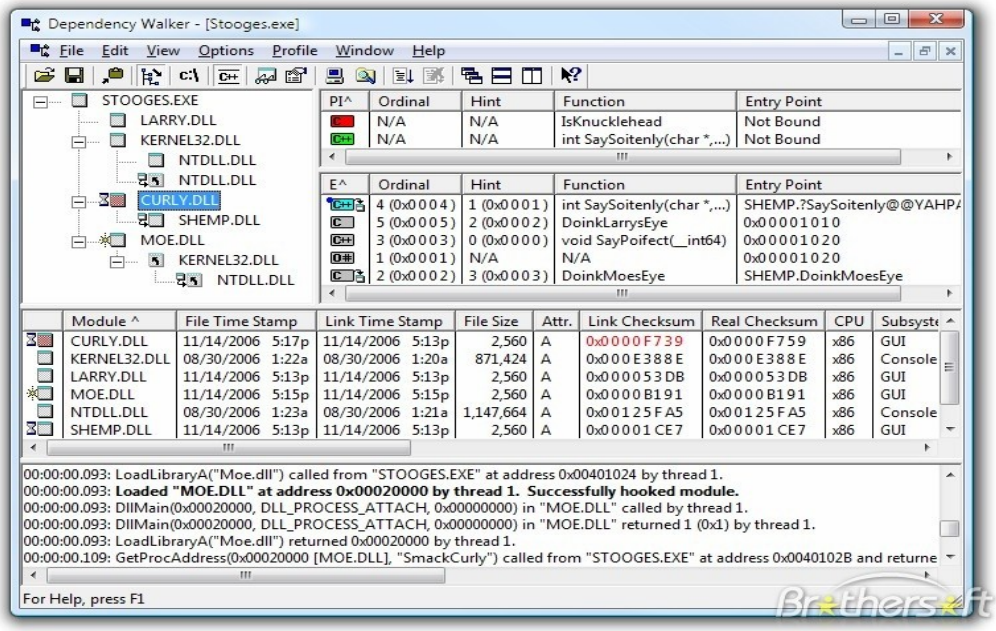
```
# python peframe.py -h
peframe 0.4.1 by Gianni 'guelfoweb' Amato
http://code.google.com/p/peframe/

USAGE:
       peframe <opt> <file>

OPTIONS:
       -h     --help        This help
       -a     --auto        Show Auto analysis
```

```
-i      --info        PE file attributes
        --hash        Hash MD5 & SHA1
        --meta        Version info & metadata
        --peid        PE Identifier Signature
        --antivm           Anti Virtual Machine
        --antidbg          Anti Debug | Disassembler
        --sections         Section analyzer
        --functionsImported DLLs & API functions
        --suspicious Search for suspicious API & sections
        --dump      Dumping all the information
        --strings          Extract all the string
        --file-url         Extract File Name and Url
        --file-verbose      Discover potential file name
        --hexdump          Reverse Hex dump
        --import           List Entry Import instances
        --export           List Entry Export instances
        --resource         List Entry Resource instances
        --debug            List Entry DebugData instances
```

4.1.6 Dependency Walker

Dependency Walker es una herramienta que escanea módulos Windows de 32 y 64 bits (exe, dll, ocs, sys, etc.) y construye un diagrama jerárquico con todos los módulos dependientes. Para cada módulo encontrado, lista todas las funciones que son exportadas por ese módulo, y aquellas funciones que están siendo llamadas por otros módulos.

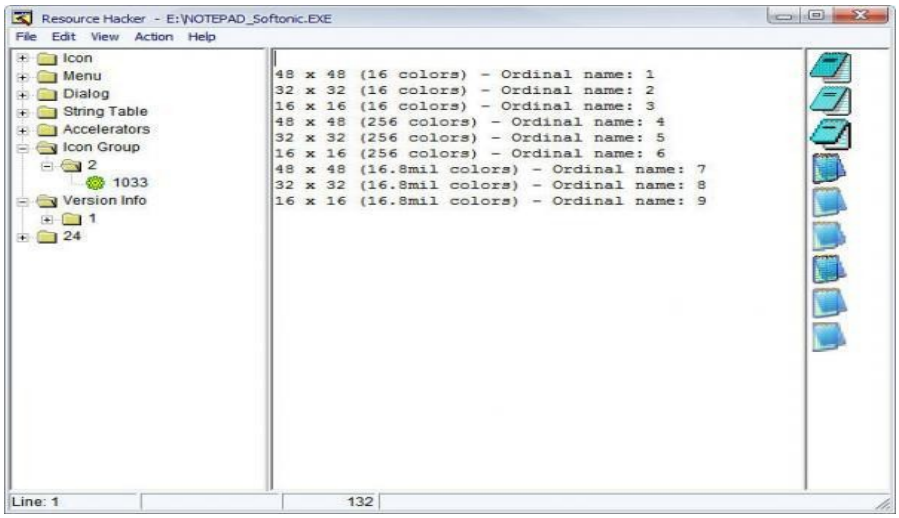
Se puede descargar desde la página del proyecto: http://www.dependencywalker.com/



4.1.7 Resource Hacker

Resource Hacker es una utilidad para ver, modificar, renombrar, añadir, borrar y extraer recursos gráficos en ejecutables y archivos de recursos de Windows 32 y 64 bits. Visualizar las imágenes de un fichero sin ejecutarlo puede darnos una idea que lo que puede hacer la aplicación.

Se puede descargar de la siguiente página: http://www.angusj.com/resourcehacker/



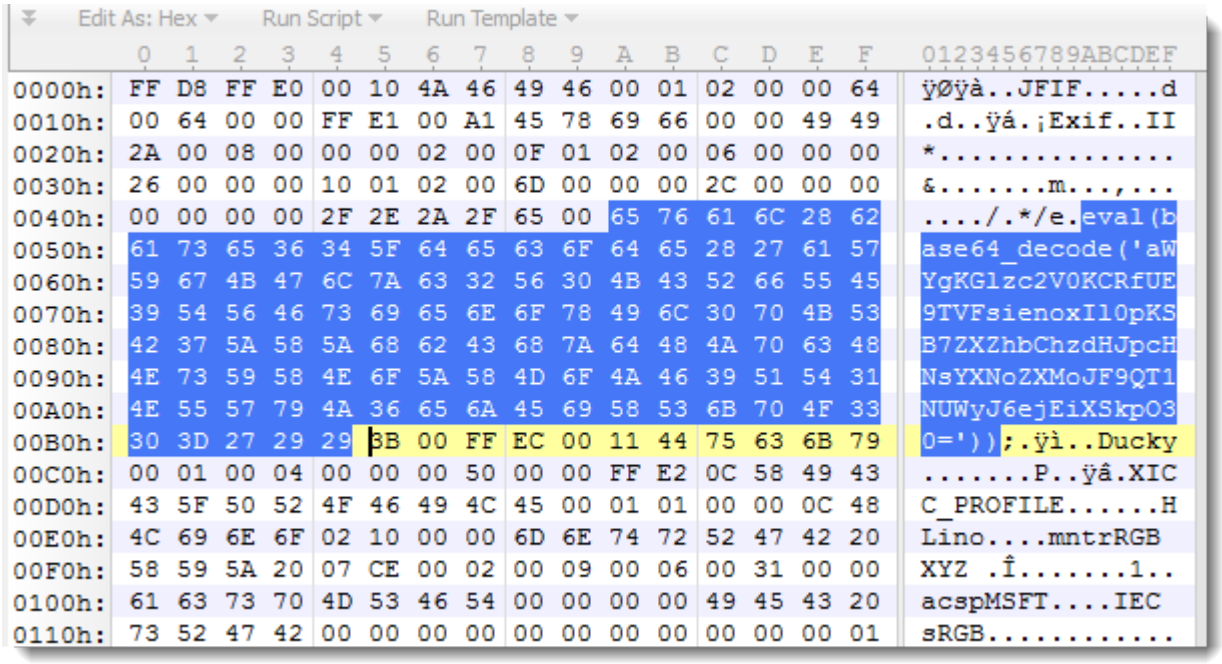
4.1.8 Yara

Yara es una herramienta multiplataforma que permite identificar y clasificar muestras de código malicioso en base a unos patrones textuales o binarios presentes en el malware.

Mediante la definición de rules podemos indicarle al motor de Yara qué debe buscar y cómo lo debe clasificar. Veamos un ejemplo:

```
rule JPEG_EXIF_Contains_eval
{
    meta:
        author = "Didier Stevens"
        description = "Detect eval function inside JPG EXIF header"
        method = "Detect JPEG file: EXIF header ($a) and eval function ($b)"
    strings:
        $a = {FF E1 ?? ?? 45 78 69 66 00}
        $b = /\Weval\s*\(/
    condition:
        uint16be(0x00) == 0xFFD8 and $a and $b in (@a + 0x12 .. @a + 0x02 +
uint16be(@a + 0x02) - 0x06)
}
```

La anterior regla, creada por el investigador de seguridad Didier Stevens, indica a Yara cómo encontrar ficheros de imagen tipo JPEG, que incluyan en sus metadatos EXIF una expresión de tipo eval(), es decir, imágenes que sean potencialmente maliciosas. Se puede observar un ejemplo en la siguiente imagen:



Se puede encontrar más información sobre Yara en la siguiente URL:

- https://plusvic.github.io/yara/

4.1.9 Desofuscación

Para conseguir esquivar las medidas de protección implementadas por los usuarios y engañar a los sistemas antivirus, es muy común que el malware venga ofuscado, tratando de ocultar lo que realmente hace.

Existen multitud de herramientas que permiten desofuscar el código para un posterior análisis. Se mencionan aquí las siguientes:

- NoMoreXOR
- XORBruteForce
- unxor
- Balbuzard

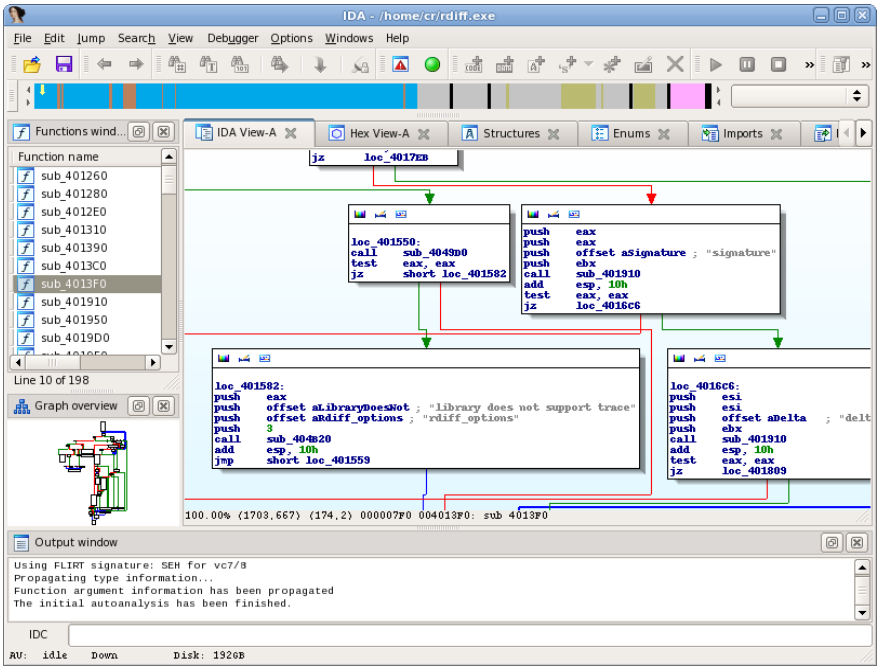
Se puede consultar un listado más completo, incluyendo enlaces a los sitios webs de cada proyecto en la siguiente URL: https://github.com/rshipp/awesome-malware-analysis#deobfuscation

4.1.10 Debugging, Reversing y Desensamblado

Para profundizar aún más en el análisis estático de una muestra es necesario hacer uso del desensamblado. Se usa un programa desensamblador para obtener el código ensamblador de un binario. Este código se lee y se analiza, con el objetivo de entender cómo funciona el programa. Un malware almacenado en disco está normalmente en formato binario a nivel de código máquina. El código máquina es una forma de código que el ordenador puede ejecutar rápida y eficientemente. Cuando se desensambla un binario, se toma el binario como entrada y se genera código ensamblador como salida.

Algunos de los desensambladores más conocidos son los siguientes:

- IDA Pro
- Hopper Disassembler
- OBJ2ASM
- PE Explorer
- W32DASM
- OllyDbg
- Objconv
- BORG Dissasembler
- objdump
- gdb



Se puede consultar un listado más amplio de desensambladores en el siguiente enlace:

- http://en.wikibooks.org/wiki/X86_Disassembly/Disassemblers_and_Decompilers

5 CONCLUSIONES

El presente documento es la continuación de la serie iniciada en comunicados anteriores y donde se exponía cómo el análisis de malware nos puede proporcionar la información necesaria para detectar infecciones de equipos y en la red, determinando el impacto que pueden ocasionar y las acciones que realizan. De esta forma se pueden determinar las medidas necesarias para evitar que se produzcan incidentes de seguridad o reducir su impacto en caso de producirse.

Dado que la ejecución de muestras de malware supone un riesgo para el equipo en el que se ejecuta y para la red, a través de los conocimientos expuestos en el presente documento se detalla la metodología para analizar código potencialmente malicioso sin necesidad de ejecutarlo, detectando ciertos indicadores que nos permitan identificar ese código como malware.

Por tanto, el documento se centra en las técnicas de análisis estático para un proceso de análisis manual de malware, destacando algunas de las herramientas más usadas por los investigadores de seguridad.

En futuros informes de seguridad se mostrará cómo continuar analizando un fichero sospechoso mediante un análisis dinámico, esto es, ejecutándolo en un entorno controlado.

6 REFERENCIAS

[1] Sikorski, Michael & Honig, Andrew. *Practical Malware Analysis. The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012

[2] Kendall, Kris & McMillan, Chad. *Practical Malware Analysis*. Conferencia Black Hat, 2007