Centro de
Investigación en
Computación

Reverse Engineering

Ingeniería Inversa

Teoría y práctica

Course

Análisis y Detección de Malware

Instructor

Acosta Bermejo Raúl

Lecture notes

2024-B
22 de octubre del 2024
Última actualizacioón

Instituto
Politécnico
Nacional

Table of contents (outline)

Tabla de contenido

1. Introducción
2. Análisis de Archivos
    i. Formato ELF
    ii. Formato PE
    iii. Formato Mach-O
3. Des-ensamblador de ejecutables
    1. Lista de ejemplos

Introduction

Introducción

Temario

Introduction

Ingeniería Inversa

- El **holismo** es una posición metodológica y epistemológica que postula cómo los sistemas y sus propiedades, deben ser analizados en su conjunto y no solo a través de las partes que los componen.
- Pero aún consideradas éstas separadamente, analiza y observa el sistema como un *todo* integrado y global que en definitiva determina cómo se comportan las partes, mientras que un mero análisis de éstas no puede explicar por completo el funcionamiento del **todo**.

- La ingeniería inversa (II) estudia o analiza un producto disponible en el mercado (software, dispositivo electrónico, pieza mecánica, estructura, etc.) con el fin de conocer detalles de su diseño, construcción y operación.
- La II usada como una forma de producir una versión mejorada del producto y no con el objetivo de producir una copia normalmente ilegal o alterar su funcionamiento original (cracking de contraseñas).
- La II se ha incluido en algunos currículos de los programas de ingeniería ya que se requiere una metodología con la cual se adquieren algunas habilidades como son:

Entender de manera holística un producto de ingeniería, plantearse hipótesis, revisar los conceptos físicos y realizar experimentación para validar las hipótesis, y proponer nuevas mejoras sobre el diseño, construcción y operación del producto

Reverse engineering

Ingeniería Inversa

- Se ha utilizado en el desarrollo de la ingeniería, como es el caso de:
  - Muchas empresas de producción que iniciaron copiando un producto, y posteriormente optimizándolo.
  - Las industrias militares que mejoraron su tecnología a partir del material de guerra incautado a los ejércitos enemigos.
- Otro uso de la II (caso Corea) que la usó como herramienta de innovación para conservar la independencia de los países desarrollados por lo que primero promovió el flujo de la tecnología en el país para luego no pagar licencias de productos que podían obtener a través de la II.

Introduction

Ingeniería Inversa

Hacer II puede requerir varias actividades que dependen del área (Hardware/Software) como son:

- Elaborar un modelo de caja negra y validarlo:
  - Identificar entradas y salidas.
  - Identificar y agrupan los subsistemas, componentes, funciones, interacciones y flujos.
- Identificar el protocolo de comunicación utilizado:
  - Mensajes, protocolos de red, identificadores/llaves, etc.
- Identificar los componentes internos
  - Para malware
  - Des-ensamblar/Decompilar un archivo ejecutable.

Introduction

Ingeniería Inversa

El análisis de un programa se puede hacer de 3 formas:

- Caja negra: no se sabe nada.
- Caja blanca: se tiene acceso al código fuente y la documentación
- Caja gris: se tiene información parcial.

Hay algo similar en:

1. El desarrollo de software:
   Pruebas de software de caja negra
2. El hackeo ético (pentesting)

Introduction

Conceptos básicos

SysAdmin, Audit, Network and Security
Organismo privado de EU con fines de lucro: entrenaimento
https://www.sans.org/

Definiciones

- Es el estudio o proceso para determinar la funcionalidad, el origen y el impacto potencial de una muestra determinada de malware, como un virus, un gusano, un troyano, un rootkit o una puerta trasera [Wikipedia].
- El **SANS** define Objetivos, Tipos, Herramientas, Metodología, Adquisición, y Detección, todo dentro del marco de la Respuesta a Incidentes de Seguridad

  https://www.sans.org/reading-room/whitepapers/malicious/malware-analysis-introduction-2103
- El examen de software malicioso implica varias etapas, que incluyen, entre otras, las siguientes:
  1. Realizar Ingeniería Inversa de forma manual al código.
  2. Análisis de comportamiento Interactivo
  3. Análisis de propiedades estáticas.
  4. Análisis totalmente automatizado

Introduction

Tipos de análisis

El análisis y la detección de malware se puede hacer de dos formas:

- Análisis estático

  Estudio del malware sin tener que ejecutarlo.

  Se revisa su estructura: código binario.

- Análisis dinámico

  Se ejecuta el malware y se obtiene su comportamiento, es decir, se identificas todas sus actividades, por ej. leer, modificar o borrar archivos.

Introduction

Tipos de análisis

En ambos tipos de análisis se puede procesar la muestra de malware de dos formas:

- Código Fuente

  Depende del tipo de lenguaje: compilado e interpretado.

  Para los interpretados siempre está disponible (PHP, Bash, etc.).

- Código Binario

  Requiere saber como fue generado:

  1. Conocer el compilador.
  2. Conocer el procesador.

Análisis de archivos

Conceptos

Herramientas

Análisis de archivos

Conceptos

- Analizadores de ejecutables
  - Saben leer e interpretar la estructura de un archivo ejecutable.
  - Depende del sistema operativo:
    - PE (*Portable Executable*) para Windows.
    - ELF (*Executable and Linkable Format*) para sistemas Like-UNIX (Linux, BSD, etc).
    - Mach-O para Mac OS X.
  - Una gran cantidad de herramientas:
    - PE: pev (http://pev.sourceforge.net/).
    - ELF: GUI (ELFparser), CLI (readelf).

Análisis de archivos

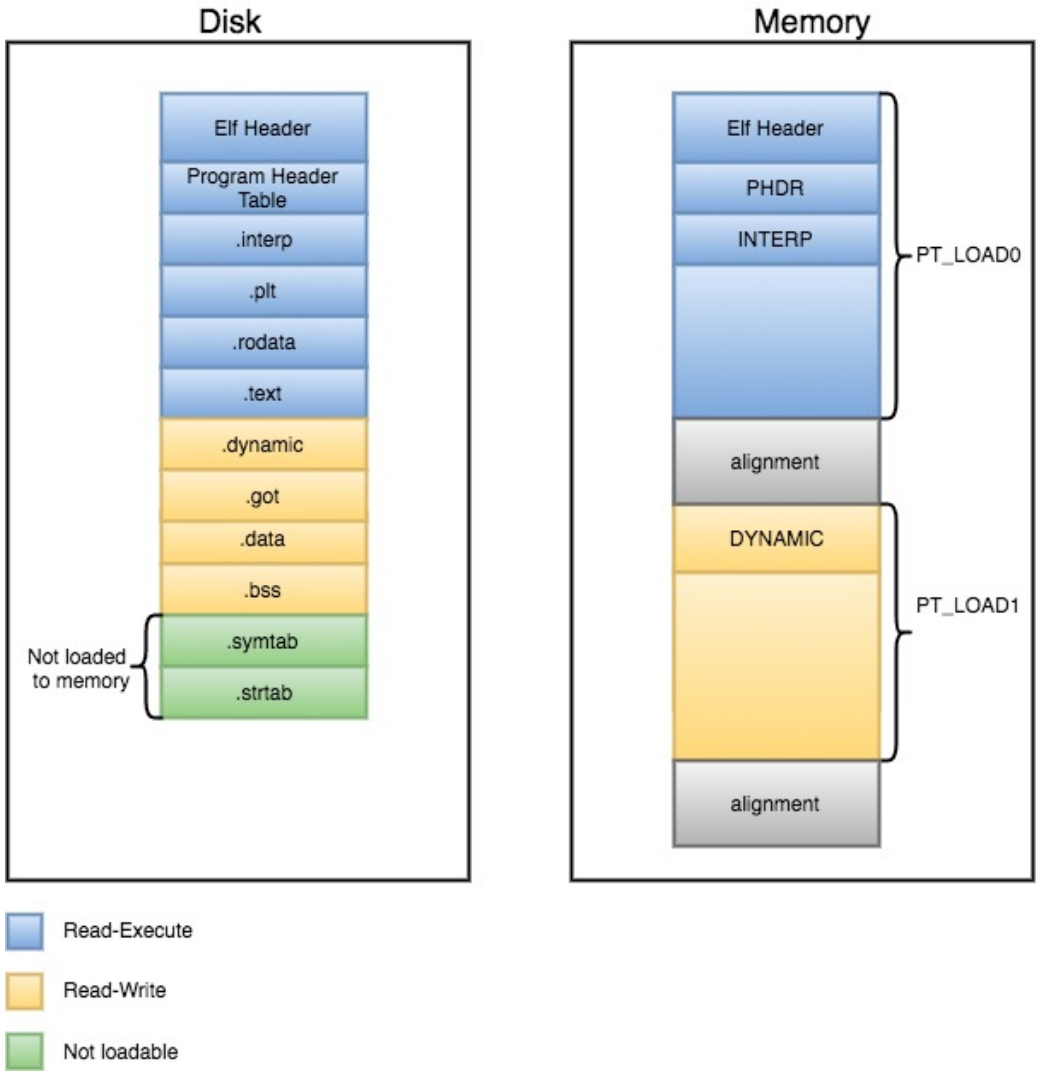
Proceso = Programa en Ejecución (esta en la RAM + registros)

Código	+	Datos	+	Pila
Instrucciones		Variables globales estáticas y dinámicas		Funciones parámetros var locales

- Un archivo EXE no tiene todo.
- Además no es una secuencia de datos, de bytes
  - Sino una secuencia de bloques de datos: segmentos, secciones.

Análisis de archivos

- **Segments** are divided into sections, each section has an utility for the ELF file.
- **Sections** per se, are not useful at runtime, so they are only useful at link time.
- Segments are used for creating a block of memory, with some specific permissions and store there some content.
- Sections gather all needed information to link a given object file and build an executable, while Program Headers split the executable into segments with different attributes, which will eventually be loaded into memory.



Formato ELF

Para sistemas UNIX

Intro

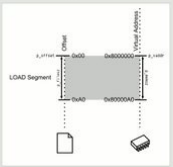
ELF

UNIX

Loading process

1 Header
the ELF header is parsed
the Program Header is parsed
(Sections are not used)

2 Mapping
the file is mapped in memory
according to its segment(s)

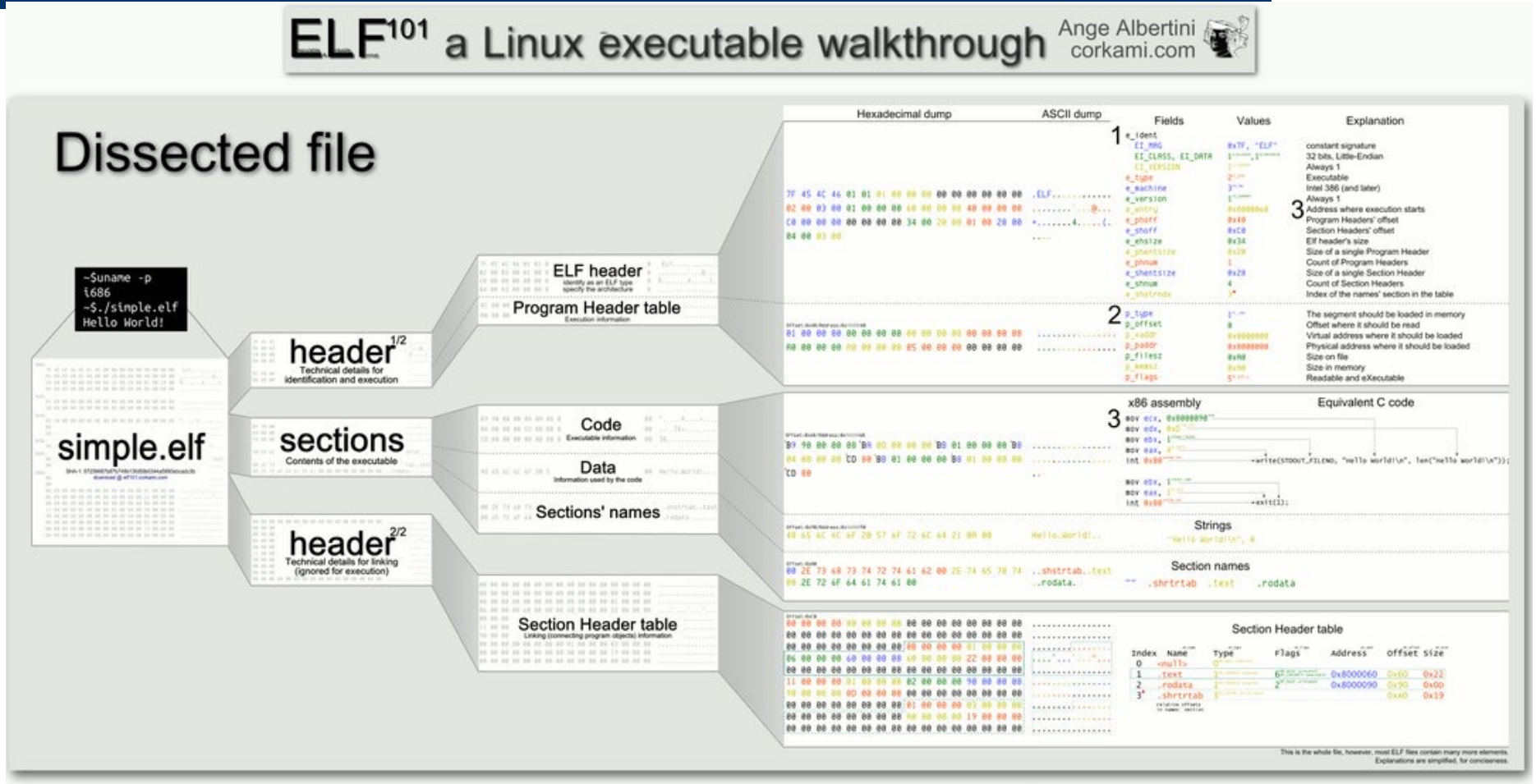


3 Execution
Entry is called
Syscalls are accessed via:
- Syscall number in the EAX register
- calling Interrupt 0x80

Trivia

The ELF was first specified by U.S. L. and U.I. for UNIX System V, in 1989

The ELF is used, among others, in:
- Linux, Android, *BSD, Solaris, BeOS
- PSP, Playstation 2-4, Dreamcast, GameCube, Wii
- various OSes made by Samsung, Ericsson, Nokia,
- Microcontrollers from Atmel, Texas Instruments



ELF

UNIX

Referencias

- http://www.skyfree.org/linux/references/ELF_Format.pdf
- https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/
- https://gist.github.com/DtxdF/e6d940271e0efca7e0e2977723aec360

Comentarios

- El formato original fue el a.out.

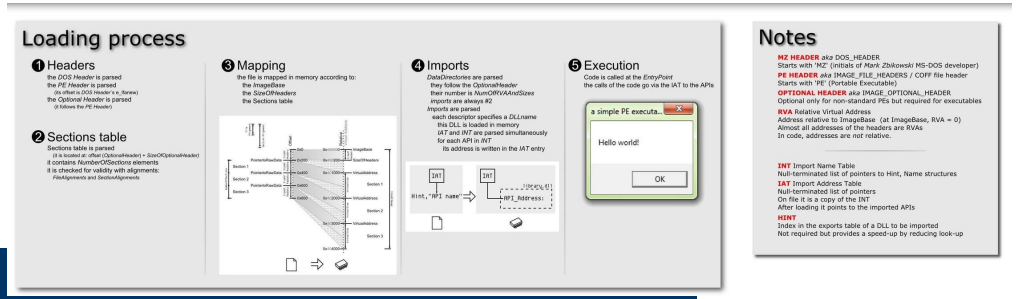
Formato PE

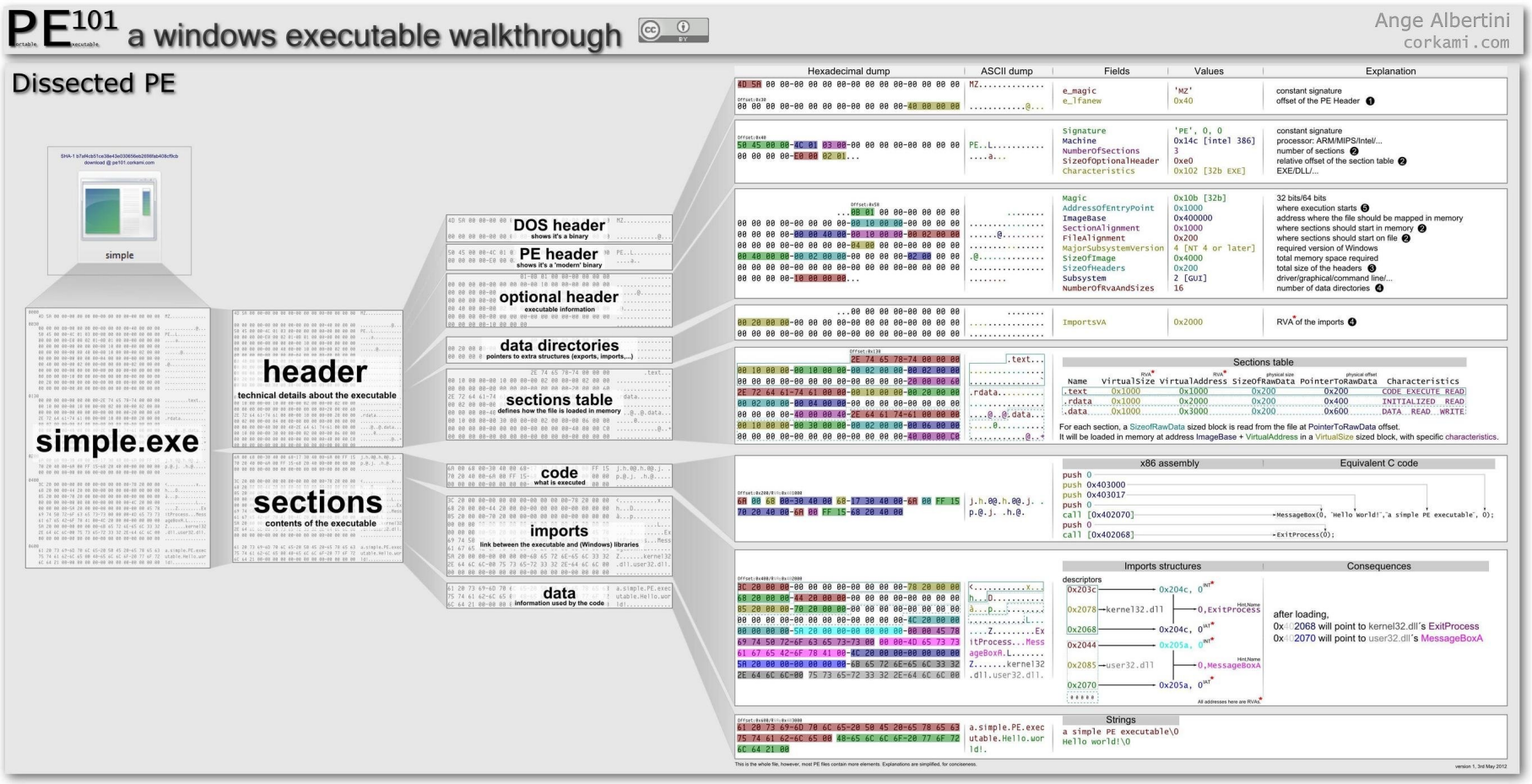
Para Windows Microsoft

Intro

PE

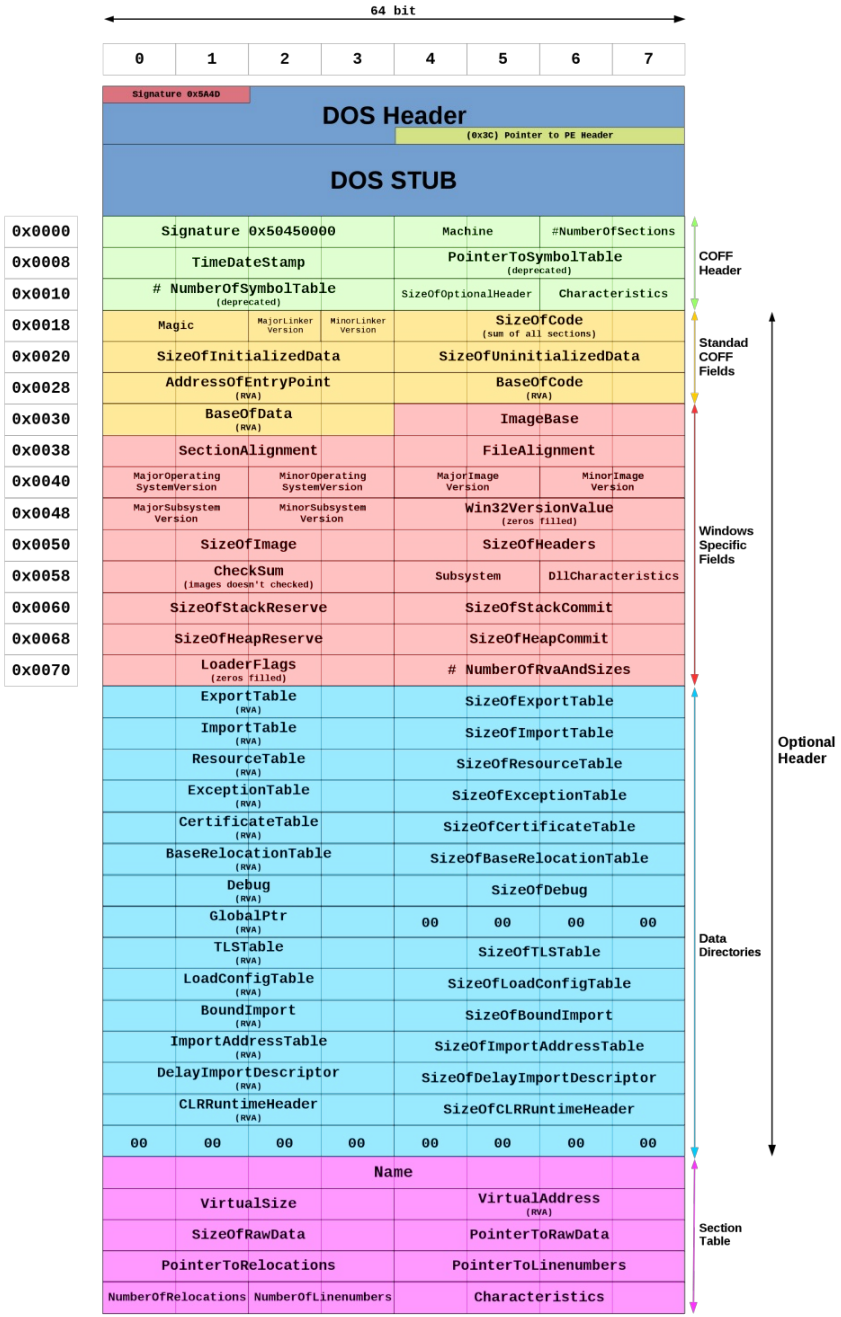
Windows


Loading process
1 Headers
the DOS Header is parsed
the PE Header is parsed
the Optional Header is parsed
2 Sections table
3 Mapping
the file is mapped in memory according to:
the ImageBase
the SizeOfHeaders
the Sections table
4 Imports
imports are always #2
5 Execution
Code is called at the EntryPoint
a simple PE executa...
Hello world!
OK
Notes
MZ HEADER aka DOS_HEADER
Starts with 'MZ' (initials of Mark Zbikowski MS-DOS developer)
PE HEADER aka IMAGE_FILE_HEADERS / COFF file header
Starts with 'PE' (Portable Executable)
OPTIONAL HEADER aka IMAGE_OPTIONAL_HEADER
Optional only for non-standard PEs but required for executables
RVA Relative Virtual Address
Address relative to ImageBase (at ImageBase, RVA = 0)
Almost all addresses of the headers are RVAs
In code, addresses are not relative.
INT Import Name Table
Null-terminated list of pointers to Hint, Name structures
IAT Import Address Table
Null-terminated list of pointers
On file it is a copy of the INT
After loading it points to the imported APIs
HINT
Index in the exports table of a DLL to be imported
Not required but provides a speed-up by reducing look-up


PE101 a windows executable walkthrough
Ange Albertini
corkami.com
Dissected PE
simple
simple.exe
header
technical details about the executable
sections
contents of the executable
DOS header
shows it's a binary
PE header
shows it's a 'modern' binary
optional header
executable information
data directories
pointers to extra structures (exports, imports,...)
sections table
defines how the file is loaded in memory
code
what is executed
imports
link between the executable and (Windows) libraries
data
information used by the code
Hexadecimal dump
ASCII dump
Fields
Values
Explanation
e_magic
'MZ'
constant signature
e_lfanew
0x40
offset of the PE Header
Signature
'PE', 0, 0
constant signature
Machine
0x14c [intel 386]
processor: ARM/MIPS/Intel/...
NumberOfSections
3
number of sections
SizeOfOptionalHeader
0xe0
relative offset of the section table
Characteristics
0x102 [32b EXE]
EXE/DLL/...
Magic
0x10b [32b]
32 bits/64 bits
AddressOfEntryPoint
0x1000
where execution starts
ImageBase
0x400000
address where the file should be mapped in memory
SectionAlignment
0x1000
where sections should start in memory
FileAlignment
0x200
where sections should start on file
MajorSubsystemVersion
4 [NT 4 or later]
required version of Windows
SizeOfImage
0x4000
total memory space required
SizeOfHeaders
0x200
total size of the headers
Subsystem
2 [GUI]
driver/graphical/command line/...
NumberOfRvaAndSizes
16
number of data directories
ImportsVA
0x2000
RVA of the imports
Sections table
Name
VirtualSize
VirtualAddress
SizeOfRawData
PointerToRawData
Characteristics
.text 0x1000 0x1000 0x200 0x200 CODE_EXECUTE_READ
.rdata 0x1000 0x2000 0x200 0x400 INITIALIZED_READ
.data 0x1000 0x3000 0x200 0x600 DATA_READ_WRITE
For each section, a SizeofRawData sized block is read from the file at PointerToRawData offset.
It will be loaded in memory at address ImageBase + VirtualAddress in a VirtualSize sized block, with specific characteristics.
x86 assembly
Equivalent C code
push 0
push 0x403000
push 0x403017
push 0
call [0x402070]
push 0
call [0x402068]
MessageBox(0, "Hello World!","a simple PE executable", 0);
ExitProcess(0);
Imports structures
Consequences
descriptors
0x203c
0x204c, 0
0x2078
kernel32.dll
0,ExitProcess
0x2068
0x204c, 0
0x2044
0x205a, 0
0x2085
user32.dll
0,MessageBoxA
0x2070
0x205a, 0
after loading,
0x2068 will point to kernel32.dll's ExitProcess
0x2070 will point to user32.dll's MessageBoxA
Strings
a simple PE executable\0
Hello world!\0
This is the whole file, however, most PE files contain more elements. Explanations are simplified, for conciseness.
version 1, 3rd May 2012

PE

Windows


64 bit
0 1 2 3 4 5 6 7
Signature 0x5A4D
DOS Header
(0x3C) Pointer to PE Header
DOS STUB
0x0000 Signature 0x50450000 | Machine | #NumberOfSections
0x0008 TimeDateStamp | PointerToSymbolTable (deprecated)
0x0010 # NumberOfSymbolTable (deprecated) | SizeOfOptionalHeader | Characteristics
0x0018 Magic | MajorLinker Version | MinorLinker Version | SizeOfCode (sum of all sections)
0x0020 SizeOfInitializedData | SizeOfUninitializedData
0x0028 AddressOfEntryPoint (RVA) | BaseOfCode (RVA)
0x0030 BaseOfData (RVA) | ImageBase
0x0038 SectionAlignment | FileAlignment
0x0040 MajorOperating SystemVersion | MinorOperating SystemVersion | MajorImage Version | MinorImage Version
0x0048 MajorSubsystem Version | MinorSubsystem Version | Win32VersionValue (zeros filled)
0x0050 SizeOfImage | SizeOfHeaders
0x0058 CheckSum (images doesn't checked) | Subsystem | DllCharacteristics
0x0060 SizeOfStackReserve | SizeOfStackCommit
0x0068 SizeOfHeapReserve | SizeOfHeapCommit
0x0070 LoaderFlags (zeros filled) | # NumberOfRvaAndSizes
ExportTable (RVA) | SizeOfExportTable
ImportTable (RVA) | SizeOfImportTable
ResourceTable (RVA) | SizeOfResourceTable
ExceptionTable (RVA) | SizeOfExceptionTable
CertificateTable (RVA) | SizeOfCertificateTable
BaseRelocationTable (RVA) | SizeOfBaseRelocationTable
Debug (RVA) | SizeOfDebug
GlobalPtr (RVA) | 00 00 00 00
TLSTable (RVA) | SizeOfTLSTable
LoadConfigTable (RVA) | SizeOfLoadConfigTable
BoundImport (RVA) | SizeOfBoundImport
ImportAddressTable (RVA) | SizeOfImportAddressTable
DelayImportDescriptor (RVA) | SizeOfDelayImportDescriptor
CLRRuntimeHeader (RVA) | SizeOfCLRRuntimeHeader
00 00 00 00 00 00 00 00
Name
VirtualSize | VirtualAddress (RVA)
SizeOfRawData | PointerToRawData
PointerToRelocations | PointerToLinenumbers
NumberOfRelocations | NumberOfLinenumbers | Characteristics
COFF Header
Standad COFF Fields
Windows Specific Fields
Optional Header
Data Directories
Section Table

PE

Windows

Referencias

Oficial

- https://learn.microsoft.com/en-us/windows/win32/debug/pe-format

Varias

- https://wiki.osdev.org/PE

Comentarios

- Se usa a partir de Win95 y Win NT.
- Se basa en el formato COFF *(Common Object File Format)*
- Para .NET es un contenedor de Cli y Mono.

Formato Mach-O

Para Apple, macOS e iOS

Intro

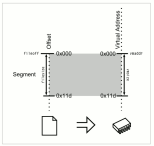
Mach-O

Apple

LOADING PROCESS

1 HEADER
THE MACH-O HEADER IS PARSED
THE COMMANDS ARE PARSED

2 MAPPING
THE FILE IS MAPPED IN MEMORY ACCORDING TO ITS SEGMENT AND COMMANDS



3 EXECUTION
THE ENTRY POINT IS CALLED

SYSCALLS[OS] ARE ACCESSED VIA:
- SYSCALL NUMBER AND CLASS, IN THE RAX REGISTER
- CALLING INSTRUCTION SYSCALL

TRIVIA

THE MACH-O FORMAT COMES FROM THE MACH KERNEL, CREATED AT C'M'U, IN 1985

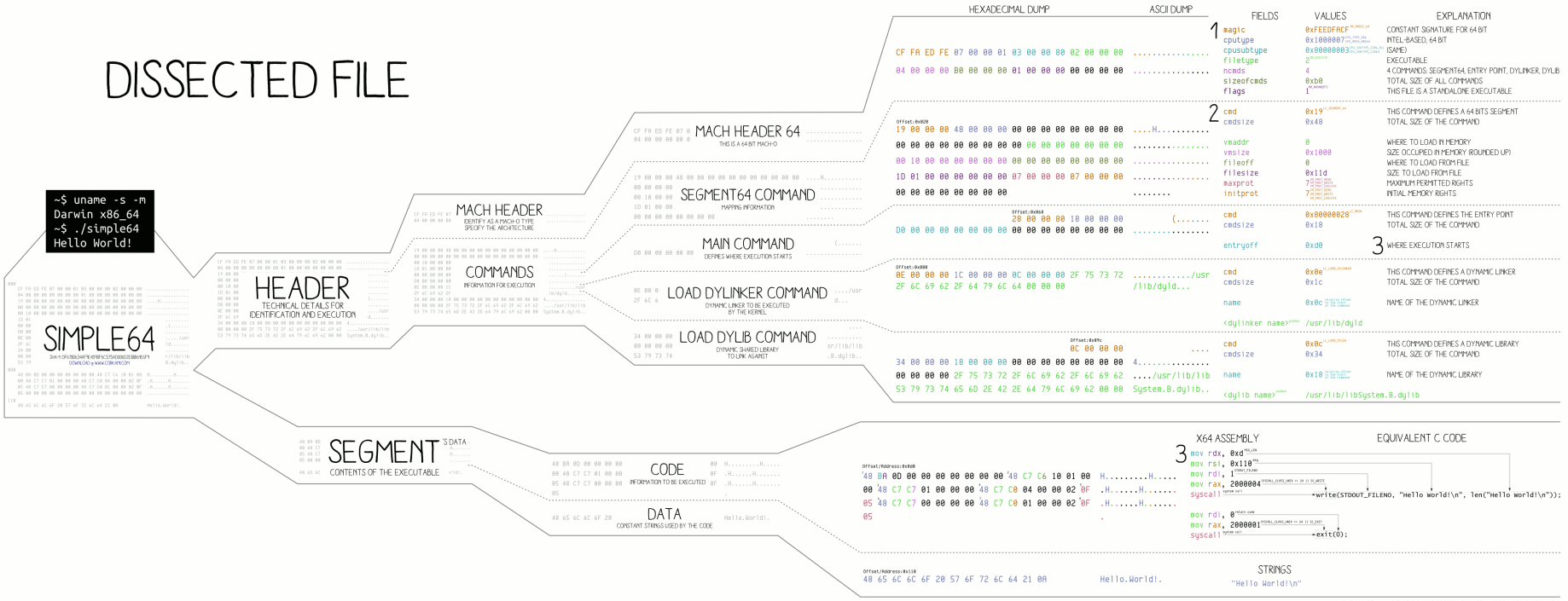
IT'S USED AMONG OTHERS, BY OS X, IOS, "STEP... ON IPHONES, IPODS, MACS...

VERSION 1.00
2014/01/02

MACH-O[101] a 64b OS X executable walkthrough

ANGE ALBERTINI
CORKAMI.COM

DISSECTED FILE



THIS KIND OF MACH-O FILES IS ONLY SUPPORTED SINCE OS X VERSION 10.8 MOUNTAIN LION.
THIS IS THE WHOLE FILE. HOWEVER, MOST MACH-O FILES CONTAIN MANY MORE ELEMENTS.
EXPLANATIONS ARE SIMPLIFIED, FOR CONCISENESS.

Des-ensambladores

De binarios / Ejecutables

Conceptos y
Herramientas

Des-ensamblador

Concepto

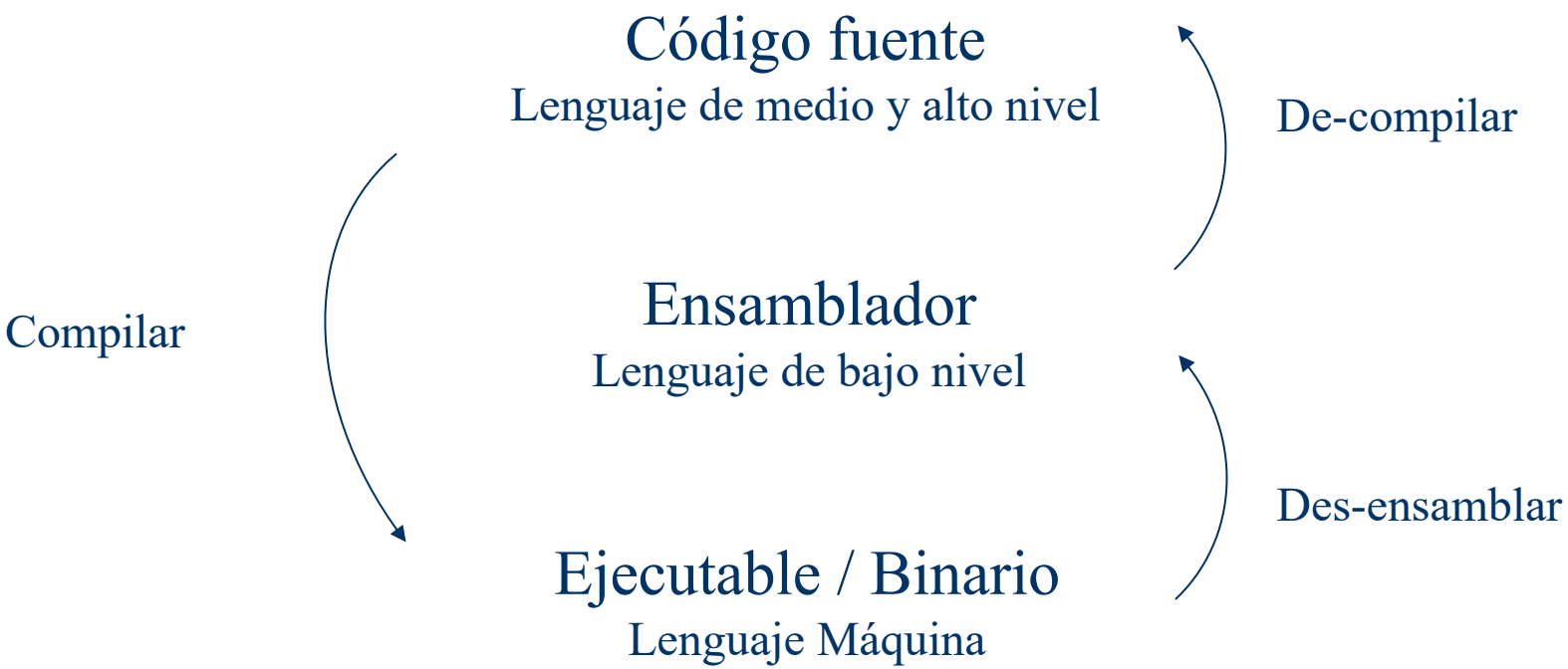
- Un **desensamblador** es un programa que traduce el lenguaje de máquina a lenguaje ensamblador, la operación inversa de la que hace el ensamblador.
- Un desensamblador difiere de un **decompilador**, en que éste tiene como objetivo un lenguaje de alto nivel en vez de al lenguaje ensamblador.
- La salida de un desensamblador, es a menudo formateada para la legibilidad humana en vez de ser adecuada para la entrada a un ensamblador, haciendo que éste sea principalmente una herramienta de ingeniería inversa.
- Algunos desensambladores hacen uso de la información de depuración simbólica presente en los archivos objeto para facilitar la comprensión.

Links

- https://en.wikipedia.org/wiki/Disassembler.

Conceptos

Flujo



Des-ensamblador

Concepto

Problemas

- El desensamblado no es una ciencia exacta.
  - En las plataformas CISC con instrucciones de ancho variable, o en presencia de código automodificable, es posible para un simple programa tener dos o más desensamblados razonables.
- El código fuente en lenguaje ensamblador generalmente permite el uso de constantes y comentarios del programador.
  - Estos son generalmente removidos, por el ensamblador, del código ensamblado a código de máquina.
  - Por lo anterior el resultado de un desensamblador carece de esta información lo que hace que la salida sea más difícil de ser interpretada por un humano.
- No es capaz de distinguir:
  - Las estructuras de control de flujo (if, for, while).
  - Tampoco los nombres de las variables, así que suele inventar nombres como VAR_0001, VAR_0002, etc.

Des-ensamblador (comparison table)

Tabla comparativa

1. Analysis
    1. Call/syscall recognition
    2. Cross-references
    3. DWARF support
    4. dSym support
    5. Signature recognition
    6. Custom structures definition
    7. Type support
    8. Diffing
2. Architectures
    1. ARM
    2. CSR
    3. Java/Dalvik
    4. TMS320C55x+
    5. V850
    6. x86 and x64.
3. Bindings
    1. Javascript
    2. Lua
    3. Python
4. Debugger
    1. Breakpoints
    2. Process attaching
    3. Remote debugging
    4. Tracing
    5. Emulation
5. Platforms
    1. Android
    2. BSD
    3. iPhone
    4. Linux
    5. OSX
    6. Windows
6. Decompilation
    1. ARM
    2. X64
    3. x86
7. User interface
    1. CLI
    2. GUI
    3. Themes support
    4. Web interface
    5. Graphs
    6. Ascii graphs

Y muchas más características.

Des-ensambladores

Descripción

Práctica

- Para familiarizarse con las variantes de archivos ejecutables que existen, se recomienda compilar y analizar varios archivos binarios.
- Las variantes más importantes son:

Formato	Sistema Operativo	Procesadores	Aplicación
PE	Cygwin, mingw32, Win	ia64, x86, x64	ls, bash
ELF	Linux, NetBSD, OpenBSD	PowerPC	
Mach-O		ARM	

- URL
  - https://github.com/JonathanSalwan/binary-samples

Des-ensambladores

Descripción

Ver slides de Ensamblador

Disassembler

- It is the exact opposite of an assembler. Where an assembler converts code written in an assembly language into binary machine code, a disassembler reverses the process and attempts to recreate the assembly code from the binary machine code.
- Since most assembly languages have a one-to-one correspondence with underlying machine instructions, the process of disassembly is relatively straight-forward, and a basic disassembler can often be implemented simply by reading in bytes, and performing a table lookup.
- Of course, disassembly has its own problems and pitfalls:
  - Separing code from data
  - Lost information (variable names)

Des-ensambladores

Descripción

- Akin to Disassembly, **decompilers** take the process a step further and actually try to reproduce the code in a high level language. Frequently, this high level language is C, because C is simple and primitive enough to facilitate the decompilation process.
- Decompilation does have its drawbacks, because lots of data and readability constructs are lost during the original compilation process, and they cannot be reproduced.

Des-ensambladores

Lista

Hay realmente muchos des-ensambladores y unos pocos menos de-compiladores:

1. IDA (Interactive DisAssembler) Comercial, Limitations-Freeware
2. Hopper Comercial, Limitations
3. x64dbg Open Source
4. Immunity Debugger
5. PE Explorer's disassembler Comercial 129 dol canadienses
6. Hiew Comercial
7. Radare2 Open Source
8. ODA desensamblador en línea.

Links

- https://en.wikibooks.org/wiki/X86_Disassembly/Disassemblers_and_Decompilers
- http://reverseengineering.stackexchange.com/questions/1817/is-there-any-disassembler-to-rival-ida-pro

Herramientas de análisis

Con GUI o Terminal

Ejemplos

Herramientas de análisis

Introducción

- Examinadores de archivos:
  - Ejecutables de los sistemas operativos: PE, ELF, Mach-O.
  - Des-ensambladores y de-compiladores.
- Depuradores
  - Ejecutan el malware a bajo nivel (ensamblador) mediante un control fino (paso a paso) y con posibilidad de examinar y alterar las variables (zonas de memoria).
- Entornos controlados:
  - Ambientes seguros para ejecutar el malware.
  - No se propaga ni daña información real.

Herramientas

De análisis

Veremos la lista detalla en los slides
Herramientas_analisis

Volcado de memoria

Toda o por proceso

Conceptos, herramientas.

Volcado de memoria

Introducción

- Definición:
  - Es un registro no estructurado del contenido de la memoria en un momento concreto, generalmente utilizado para depurar un programa que ha finalizado su ejecución incorrectamente.
  - Actualmente se refiere a un archivo que contiene una imagen en memoria de un proceso determinado.
- Herramientas:
  - Gratuitas: Volatility, Rekall
  - Comerciales: Varias usadas en:

    DFIR (*Digital Forensics and Incident Response*)

Links

- https://en.wikipedia.org/wiki/Core_dump.
- https://github.com/digitalisx/awesome-memory-forensics

Courses

Cursos

Gratuitos

Courses

Cursos

Existen muchos en internet y dentro de los más conocidos:

- Ricardo Narvaja
  - Argentino que pertenece al grupo CRACKSLATINOS.
- Cursos centrados en Windows
  - Windows reversing:

    Function calls & Syscalls, DLLs, Structures.
  - https://tryhackme.com/room/windowsreversingintro
- Cursos con repaso de todo lo necesario:
  - Notación en hexadecimal, ensamblador (registros), etc.
  - https://0xinfection.github.io/reversing/

Courses

Cursos

En sus videos comenta la evolución del tema y como se ha ido actualizando Radare, Ghidra

Ricardo Narvaja

- Todo el material es gratuito.
  - Herramientas principalmente: IDA y OllyDbg.
  - Empieza desde cero con cursos de C y Sistemas Operativos.
  - Se centra principalmente en arquitectura x86.
  - Tienes varias prácticas cuyo nivel se incrementa poco a poco.
  - Material de Youtube, documentos en Word y muchos archivos de ejemplos para las prácticas.
- Referencias
  - https://ricardonarvaja.info/
  - Algunos navegadores detectan posible malware en su sitio por el tipo de material que publica pero no es cierto.

Prácticas

Análisis estático

Ejercicios

Práctica

Análisis Estático

Analizar un binario

- Con varias herramientas: GUI, CLI y API
- Obtener información diversa
- Analizar en su conjunto toda la información
  - Tipo de archivo y cadenas
  - Encabezado
  - Librerías que usa y tipo
  - Ensamblador
  - Etc.

Práctica

Análisis Estático

Ejemplo

- Usaremos el comando ls de Linux.
- Tipo de archivo

  $ file /bin/ls

  ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=bccb4c17516c6a9ad59c3ec19b347c83236c04c2, stripped

Formato: ELF
Procesador: x86-64, 64 bits
SO: GNU/Linux
ABI: SYSV
Opciones: stripped (has no debugging symbols)

Práctica

Análisis Estático

- Encabezado

$ readelf -e /bin/ls

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x5e50
  Start of program headers:          64 (bytes into file)
  Start of section headers:          141240 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         31
  Section header string table index: 30
```


Práctica

Análisis Estático

- Encabezado

$ readelf -S /bin/ls

There are 31 section headers, starting at offset 0x227b8

Section Headers:

[Nr]	Name	Type	Address	Offset	Size	EntSize	Flags	Link	Info	Align
[0]		NULL	0000000000000000	00000000	0000000000000000	0000000000000000		0	0	0
...										
[30]	.shstrtab	STRTAB	0000000000000000	00022674	000000000000013e	0000000000000000		0	0	1

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), I (large)

I (info), L (link order), G (group), T (TLS), E (exclude), x (unknown)

O (extra OS processing required) o (OS specific), p (processor specific)

$ readelf -s /bin/ls

Symbol table '.dynsym' contains 135 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
...							
134:	0000000000016680	104	FUNC	GLOBAL	DEFAULT	15	_obstack_free

Práctica

Análisis Estático

- Información general

$ exiftool /bin/ls

```
ExifTool Version Number         : 11.11
File Name                       : ls
Directory                       : /bin
File Size                       : 183 kB
Resource Fork Size              : 36 kB
File Modification Date/Time     : 2022:02:06 15:22:11-06:00
File Access Date/Time           : 2022:02:06 15:22:11-06:00
File Inode Change Date/Time     : 2022:02:06 15:22:11-06:00
File Permissions                : rwxr-xr-x
File Type                       : Mach-O fat binary executable
File Type Extension             :
MIME Type                       : application/octet-stream
CPU Count                       : 2
CPU Type                        : x86 64-bit, ARM 64-bit
CPU Subtype                     : i386 (all) 64-bit, ARM A500 64-bit
Object File Type                : Demand paged executable
```

Práctica

Análisis Estático

```
$ strings ls | head
/lib64/ld-linux-x86-64.so.2
libselinux.so.1
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
...
```

Ejemplo

- Firma hash

  ```
  $ sha1sum /bin/ls
  68d3a4e823b804d4aeeea94bd49ad7fad3264475  ls
  ```

- Cadenas (any sequence of 4 -the default- or more printing characters)

  ```
  $ strings /bin/ls
  68d3a4e823b804d4aeeea94bd49ad7fad3264475  ls
  $ strings ls | wc –l
  1620
  $ strings ls | awk '{ print length(), $0 | "sort -n" }'
  4 $E1
  ...
  80      --sort=WORD          sort by WORD instead of name: none (-U), size (-S),
  ```

Lista ordenada por tamaño

Práctica

Análisis Estático

Generar el ensamblador

- Con GUI
  - IDA
  - Ghidra
- Con CLI
  - Ejecutan el malware a bajo nivel (ensamblador) mediante un control fino (paso a paso) y con posibilidad de examinar y alterar las variables (zonas de memoria).
- Con API
  - Ambientes seguros para ejecutar el malware.
  - No se propaga ni daña información real.

Práctica

Análisis Estático

$ objdump -d ls | wc -l
20139

$ objdump -d ls | grep ">:"
0000000000003588 <.init>:
00000000000035b0 <.plt>:
0000000000003cc0 <.plt.sec>:
00000000000043c0 <.text>:
0000000000016500 <_obstack_begin>:
0000000000016520 <_obstack_begin_1>:
0000000000016540 <_obstack_newchunk>:
0000000000016640 <_obstack_allocated_p>:
0000000000016680 <_obstack_free>:
00000000000166f0 <_obstack_memory_used>:
0000000000016fd4 <.fini>:

Ejemplo

- Ensamblador

$ objdump -d ls | head

ls: file format elf64-x86-64

Disassembly of section .init:

0000000000003588 <.init>:

3588:	f3 0f 1e fa	endbr64	
358c:	48 83 ec 08	subq	$8, %rsp
3590:	48 8b 05 19 da 21 00	movq	2218521(%rip), %rax # 220fb0

$ objdump -d ls | grep “Disassembly of section”

Disassembly of section .init:
Disassembly of section .plt:
Disassembly of section .plt.sec:
Disassembly of section .text:
Disassembly of section .fini:

Práctica

Análisis Estático

Radare2

```
$ r2 ls.bin
 -- Use the 'id' command to see the source line related to the current seek
[0x00005e50]> aaa
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Check for objc references
[x] Check for vtables
[x] Type matching analysis for all functions (aaft)
[x] Propagate noreturn information
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00005e50]> pd $s > ls.radare2.asm
```

Práctica

Conclusión

Análisis

- Mucha información puede ser extraída del ejecutable.
- Esta información puede ser usada para detectar malware?
  - **Si**, la reflexión sería: es necesaria toda? Como se procesa?
- Queremos encontrar un patrón que se repita en todos los mlwrs.
- Como se construyen los patrones?
  - Estáticamente: por ejemplo una cadena o secuencia de bytes.
  - Estadísticamente: contar elementos.
  - Hibrido: usar varias características.
- Procedimiento
  - Analizar varios archivos
  - Proponer y encontrar el patrón
  - Verificar la propuesta

The end

Contacto

Raúl Acosta Bermejo

http:www.cic.ipn.mx
http://www.ciseg.cic.ipn.mx/

racostab@ipn.mx
racosta@cic.ipn.mx

57-29-60-00
Ext. 56652