



# IV. Reversing

---

Carlos Alonso, Alejandro Cruz, Andrea Oliva, Sergio Pérez y Rubén Santos

# Índice

1. ¿Qué es reversing?
2. Análisis de código fuente
3. Aprende a compilar y dar permisos a un archivo
4. Conoce a los hermanos TRACE
5. Tu viejo amigo
6. Análisis ELF
7. Análisis .NET



# ¿QUÉ ES REVERSING?

---

# ¿QUÉ ES REVERSING?

**Es obtener información sobre el funcionamiento interno de un archivo sin tener los datos reales sobre su estructura. Por ejemplo, a partir de un .exe**

## ¿Cómo?

Desensamblando y/o decompilando

**Para ello nos ayudaremos de:**

- Herramientas como Ghidra, DnSpy, cutter, DotPeek
- Páginas web como CyberChef o hexed.it
- Comandos como file, strings, strace o ltrace



Alto Nivel → Bajo Nivel → Máquina  
Compilación -- Ensamblado

# ¿QUÉ ES REVERSING?

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int variable=0;
7
8     while(variable!=13)
9     {
10         printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11         scanf("%d",&variable);
12     }
13     printf("Correcto! URJC{Correctp}");
14
15     return 0;
16 }
17
```

Sigue la secuencia: 1-1-2-3-5-8-...

# ¿QUÉ ES REVERSING?

```

Sigue la secuencia: 1-1-2-3-5-8-...3
Sigue la secuencia: 1-1-2-3-5-8-...4
Sigue la secuencia: 1-1-2-3-5-8-...4
Sigue la secuencia: 1-1-2-3-5-8-...5
Sigue la secuencia: 1-1-2-3-5-8-...6
Sigue la secuencia: 1-1-2-3-5-8-...7
Sigue la secuencia: 1-1-2-3-5-8-...8
Sigue la secuencia: 1-1-2-3-5-8-...9
Sigue la secuencia: 1-1-2-3-5-8-...0
Sigue la secuencia: 1-1-2-3-5-8-...1
Sigue la secuencia: 1-1-2-3-5-8-...2
Sigue la secuencia: 1-1-2-3-5-8-...
  
```

```

*****
*                               *
*                               *
*****
int __fastcall __main(void)
EAX:4    <RETURN>
__main                                       XREF[3]:    __tmainCRTStartup:004013b6(c),
                                                main:00401538(c), 00405108(*)
                                                = ??

00402110 8b 05 2a    MOV     EAX,dword ptr [initialized]
                                                52 00 00

00402116 85 c0    TEST    EAX,EAX
00402118 74 06    JZ      LAB_00402120
0040211a f3 c3    RET
0040211c 0f      ??      0Fh
0040211d 1f      ??      1Fh
0040211e 40      ??      40h  @
0040211f 00      ??      00h

LAB_00402120
00402120 c7 05 16    MOV     dword ptr [initialized],0x1
                                                52 00 00
                                                01 00 00 00
0040212a eb 84    JMP     __do_global_ctors
                                                XREF[1]:    00402118(j)
                                                = ??

DAT_0040212c
0040212c 90      ??      90h
0040212d 90      ??      90h
0040212e 90      ??      90h
0040212f 90      ??      90h
                                                XREF[1]:    0040510c(*)

*****
*                               *
*                               *
*****
void __fastcall __security_init_cookie(void)
  
```

# ¿QUÉ ES REVERSING?

¿Se parecen?



```
1
2 int main(int _Argc, char **_Argv, char **_Env)
3
4 {
5     int local_c;
6
7     __main();
8     local_c = 0;
9     while (local_c != 0xd) {
10         printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11         scanf("%d", &local_c);
12     }
13     printf("Correcto! URJC{Correctp}");
14     return 0;
15 }
16
```

```
1
2 #include <stdio.h>
3
4 int main()
5 {
6     int variable=0;
7
8     while(variable!=13)
9     {
10         printf("Sigue la secuencia: 1-1-2-3-5-8-...");
11         scanf("%d",&variable);
12     }
13     printf("Correcto! URJC{Correctp}");
14
15     return 0;
16 }
17
```



# ANÁLISIS DE CÓDIGO FUENTE

---



# ANÁLISIS DE CÓDIGO FUENTE

¿Podremos  
sacar la flag  
analizando  
el código?

```
import java.util.*;

class Reversing {
    public static void main(String args[]) {
        Reversing reversing = new Reversing();
        Scanner scanner = new Scanner(System.in);
        System.out.print("Introduzca la contraseña: ");
        String userInput = scanner.next();
        String input = userInput.substring("URJC{".length(),userInput.length()-1);
        if (reversing.Password(input)) {
            System.out.println("Acceso permitido.");
        } else {
            System.out.println("Acceso denegado!");
        }
    }

    public boolean Password(String password) {
        return password.equals("Buscando en el código");
    }
}
```



# ANÁLISIS DE CÓDIGO FUENTE

## Código ofuscado

```
var _0x2da72b=_0x23ae;function _0x23ae(_0x384bcb,_0x4d943a){var
_0x56dc43=_0x56dc();return
_0x23ae=function(_0x23ae43,_0x4d79f2){_0x23ae43=_0x23ae43-0x143;var
_0x157b9f=_0x56dc43[_0x23ae43];return
_0x157b9f;},_0x23ae(_0x384bcb,_0x4d943a);}function _0x56dc(){var
_0x2de2d7=['8101179LDLJdd','charAt','81306hLMcQW','65oKTNMq','10Cil
RRV','1431062rTFndF','5934680LMAxTh','URJC{Desofuscando}','709227QTs
rkv','3969945atijhi','1030616LcvUIK','9YcTpzQ'];_0x56dc=function(){return
_0x2de2d7;};return _0x56dc();}(function(_0x2820fc,_0x53eb87){var
_0x2064a7=_0x23ae,_0x37c6cf=_0x2820fc;while(![]){try{var
_0x4c24c9=-parseInt(_0x2064a7(0x144))/0x1+-
parseInt(_0x2064a7(0x14d))/0x2+-parseInt(_0x2064a7(0x147))/0x3*(-
parseInt(_0x2064a7(0x146))/0x4)+parseInt(_0x2064a7(0x14b))/0x5*(parseI
nt(_0x2064a7(0x14a))/0x6)+-
parseInt(_0x2064a7(0x145))/0x7+parseInt(_0x2064a7(0x14e))/0x8+parseInt
(_0x2064a7(0x148))/0x9*(parseInt(_0x2064a7(0x14c))/0xa);if(_0x4c24c9==
_0x53eb87)break;else
_0x37c6cf['push'](_0x37c6cf['shift']());}catch(_0x22350c){_0x37c6cf['push'](
_0x37c6cf['shift']());}}(_0x56dc,0x9249e));var
numero=0x1,flag=_0x2da72b(0x143);while(numero>=0x0){numero=prompt
('introduce\x20un\x20número',''),secreta(flag,numero);}function
secreta(_0x18b49d,_0x34c2bb){var
_0x3bc3be=_0x2da72b;alert(_0x18b49d[_0x3bc3be(0x149)](_0x34c2bb));}
```

## ¿Qué podemos hacer?



<https://beautifier.io/>

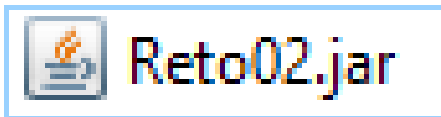
```
var numero = 0x1,
flag = _0x2da72b(0x143);
while (numero >= 0x0) {
    numero = prompt('introduce\x20un\x20número', ''),
    secreta(flag, numero);
}

function secreta(_0x18b49d, _0x34c2bb) {
    var _0x3bc3be = _0x2da72b;
    alert(_0x18b49d[_0x3bc3be(0x149)](_0x34c2bb));
}
```

# ANÁLISIS DE CÓDIGO FUENTE

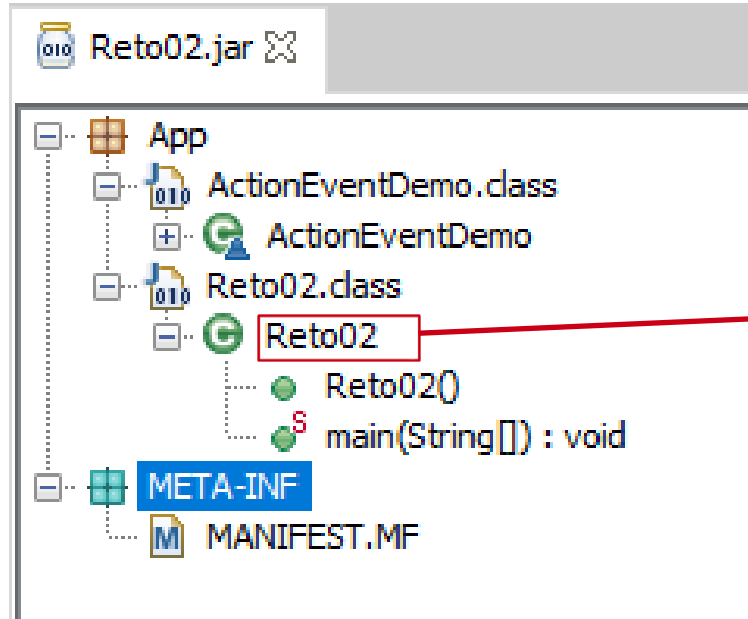
## ¿Y si nos dan una apk?

También analizamos el código fuente, pero primero miramos que es lo que se ejecuta!



# ANÁLISIS DE CÓDIGO FUENTE

## APK



```
package App;  
  
public class Reto02 {  
    public static void main(String[] args) {}  
}
```

# ANÁLISIS DE CÓDIGO FUENTE

## APK

```
package App;

import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.datatransfer.Clipboard;
import java.awt.datatransfer.StringSelection;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

class ActionEventDemo implements ActionListener {
    JFrame frame = new JFrame();
    JButton send = new JButton("Enviar");
    JTextField tf = new JTextField(10);
    JLabel result = new JLabel("", 0);

    ActionEventDemo() {
        prepareGUI();
        buttonProperties();
    }

    public void prepareGUI() {
        this.frame.setTitle("Jnic 2021 - Reto 02");
        this.frame.setVisible(true);
        Dimension dimension = Toolkit.getDefaultToolkit().getScreenSize();
        int x = (int)((dimension.getWidth() - this.frame.getWidth()) / 2.00);
        int y = (int)((dimension.getHeight() - this.frame.getHeight()) / 2.00);
        this.frame.setBounds(x - 150, y - 75, 300, 150);
        this.frame.setDefaultCloseOperation(3);
    }

    public void buttonProperties() {
        JPanel panel = new JPanel();
        this.send.setActionCommand("send");
        this.send.addActionListener(this);
        JLabel label = new JLabel("Clave");
        panel.add(label);
        panel.add(this.tf);
        panel.add(this.send);
        this.frame.getContentPane().add("North", panel);
        this.frame.getContentPane().add("Center", this.result);
        this.frame.setVisible(true);
    }

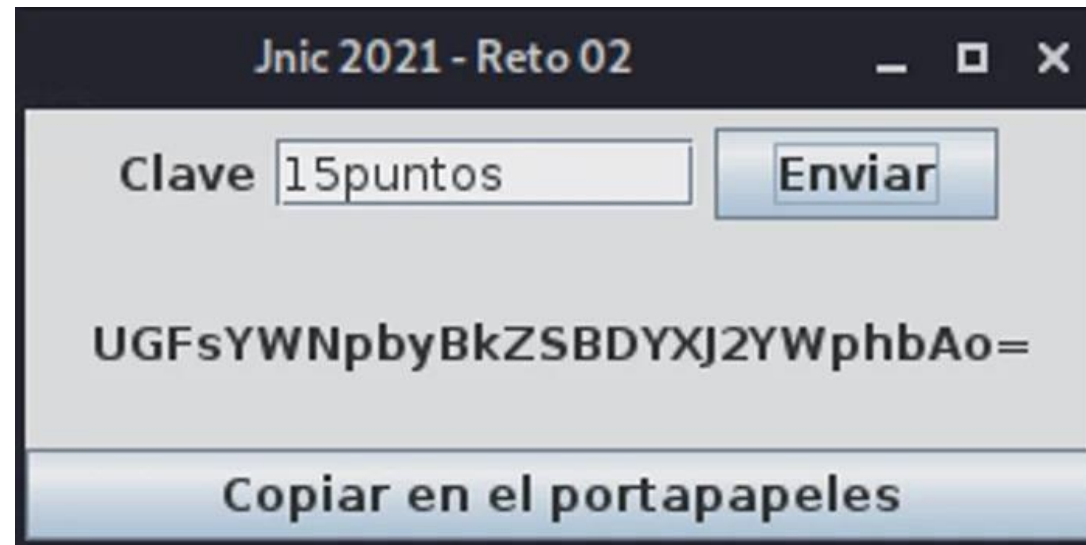
    public void actionPerformed(ActionEvent e) {
        if (e.getActionCommand().equals("send")) {
            if (this.tf.getText().equals("15puntos")) {
                JButton copy = new JButton("Copiar en el portapapeles");
                copy.setActionCommand("copy");
                copy.addActionListener(this);
                this.frame.getContentPane().add("South", copy);
                this.result.setText("UGFsYWNpbyBkZSBDYXJ2YWphbAo=");
            } else {
                this.result.setText("Vuelve a intentarlo!!");
            }
        } else if (e.getActionCommand().equals("copy")) {
            StringSelection stringSelection = new StringSelection(this.result.getText());
            Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
            clipboard.setContents(stringSelection, null);
            this.result.setText("Copiado correctamente");
        }
    }
}
```



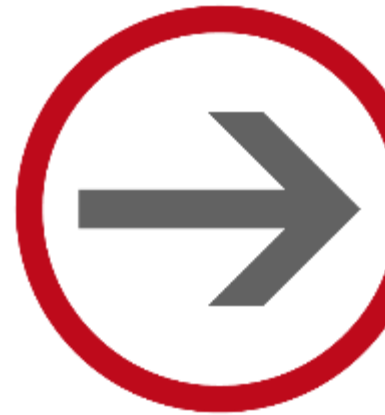
```
public void actionPerformed(ActionEvent e) {
    if (e.getActionCommand().equals("send")) {
        if (this.tf.getText().equals("15puntos")) {
            JButton copy = new JButton("Copiar en el portapapeles");
            copy.setActionCommand("copy");
            copy.addActionListener(this);
            this.frame.getContentPane().add("South", copy);
            this.result.setText("UGFsYWNpbyBkZSBDYXJ2YWphbAo=");
        } else {
            this.result.setText("Vuelve a intentarlo!!");
        }
    } else if (e.getActionCommand().equals("copy")) {
        StringSelection stringSelection = new StringSelection(this.result.getText());
        Clipboard clipboard = Toolkit.getDefaultToolkit().getSystemClipboard();
        clipboard.setContents(stringSelection, null);
        this.result.setText("Copiado correctamente");
    }
}
```

# ANÁLISIS DE CÓDIGO FUENTE

## APK



**¡Conseguido!**



# APRENDE A COMPILAR Y DAR PERMISOS A UN ARCHIVO

---

# ¿QUÉ ES UN CÓDIGO FUENTE?

El código fuente es aquel que escribe el programador y es **legible** para los humanos. Sin embargo, no siempre **se puede ejecutar directamente**, pues tiene que pasar antes por un proceso de compilación, como es el caso del código escrito en C (lenguaje compilado). En caso de que se pueda, es lenguaje interpretado (Python).

Un ejemplo de código fuente es:

```
$ cat holamundo.c
#include <stdio.h>

int main()
{
    printf("Esto es un código de ejemplo de \"Hola mundo\"");
    return 0;
}
```



# COMPILACIÓN DE UN PROGRAMA EN C



## EL COMPILADOR

Instalación: `sudo apt install gcc`

```
(kali㉿kali)-[~]  
$ gcc holamundo.c  
  
(kali㉿kali)-[~]  
$ ls  
a.out Desktop Documents
```

`gcc holamundo.c` → Genera un programa ejecutable llamado `a.out`

`gcc holamundo.c -o holamundo` → Genera un programa ejecutable llamado `holamundo`

```
(kali㉿kali)-[~]  
$ gcc holamundo.c -o holamundo  
  
(kali㉿kali)-[~]  
$ ls  
Desktop Documents Downloads holamundo
```

```
(kali㉿kali)-[~]  
└─$ ./holamundo  
Esto es un código de ejemplo de "Hola mundo"
```

18

**Carlos Alonso, Alejandro Cruz, Andrea Oliva, Sergio Pérez y Rubén Santos**

# GESTIÓN DE PERMISOS DE UN ARCHIVO

Los permisos se dividen en tres clases:

- Propietario (**u**)
- Grupo (**g**)
- Resto de usuarios (**o**)

Con el comando `ls -l` se lista mucha información de los ficheros, entre ellos los permisos:

```
(kali㉿kali)-[~]  
$ ls -l holamundo  
-rwxr-xr-x 1 kali kali 16616 Nov 26 05:23 holamundo
```

Ese primer “-” indica que se trata de un **archivo**. Si en su lugar fuera una “d”, indicaría que es un directorio (y no se puede ejecutar)

# GESTIÓN DE PERMISOS DE UN ARCHIVO

Para ejecutar un archivo será necesario tener permisos de ejecución, de lo contrario:

```
(kali@kali)-[~]  
$ ls -l ejecutable  
-rw-rw-rw- 1 kali kali 16616 Nov 26 07:02 ejecutable  
  
(kali@kali)-[~]  
$ ./ejecutable  
zsh: permission denied: ./ejecutable
```

La forma más sencilla de dar permisos es con el comando  
*chmod u+x <nombre\_archivo>*

```
(kali@kali)-[~]  
$ chmod u+x ejecutable  
  
(kali@kali)-[~]  
$ ls -l ejecutable  
-rwxrw-rw- 1 kali kali 16616 Nov 26 07:02 ejecutable  
  
(kali@kali)-[~]  
$ ./ejecutable  
Esto es un código de ejemplo de "Hola mundo"
```



# CONOCE LOS HERMANOS TRACE

---

# LOS HERMANOS

## STRACE

Ejecuta el programa hasta que termina

Intercepta las llamadas **al sistema**

También intercepta las señales que  
recibe el programa

```
> strace ./a.out
```

## LTRACE

Ejecuta el programa hasta que termina

Intercepta las llamadas **dinámicas a  
librerías**

También intercepta las señales que  
recibe el programa

```
> ltrace ./a.out
```

# EJEMPLO HELLO WORLD!

hello.c

File: hello.c

```
1  #include <stdio.h>
2
3  int main(){
4      printf("%s\n","Hello world!");
5  }
```

Programa fácil en C

Veamos las diferencias entre  
strace y ltrace...

# STRACE

```
execve("./hello", ["/.hello"], 0x7fff33825ad0 /* 40 vars */) = 0
brk(NULL) = 0x5571c4413000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=44313, ...}) = 0
mmap(NULL, 44313, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7eff26fe3000
close(3) = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\200\177\2\0\0\0\0\0"... , 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839168, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7eff26fe1000
mmap(NULL, 1852480, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7eff26e1c000
mprotect(0x7eff26e42000, 1658880, PROT_NONE) = 0
mmap(0x7eff26e42000, 1347584, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x26000) = 0x7eff26e42000
mmap(0x7eff26f8b000, 307200, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x16f000) = 0x7eff26f8b000
mmap(0x7eff26fd7000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7eff26fd7000
mmap(0x7eff26fdd000, 13376, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7eff26fdd000
close(3) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7eff26e1a000
arch_prctl(ARCH_SET_FS, 0x7eff26fe2540) = 0
mprotect(0x7eff26fd7000, 12288, PROT_READ) = 0
mprotect(0x5571c2d08000, 4096, PROT_READ) = 0
mprotect(0x7eff27018000, 4096, PROT_READ) = 0
munmap(0x7eff26fe3000, 44313) = 0
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x2), ...}) = 0
brk(NULL) = 0x5571c4413000
brk(0x5571c4434000) = 0x5571c4434000
write(1, "Hello world!\n", 13Hello world!
) = 13
exit_group(0) = ?
+++ exited with 0 +++
```

24

# LTRACE

```
puts("Hello world!" "Hello world!")
)
+++ exited (status 0) +++
```

$$= 13$$



```
1  #include <stdio.h>
2
3  int main() {
4      char frase[50];
5
6      printf("%s", "Introduce la password: ");
7      scanf("%s", &frase);
8
9      if (strcmp(frase, "impossible_password") == 0){
10         printf("%s\n", "Enhorabuena campeón");
11     } else {
12         printf("%s\n", "Oh! que pena... no era esa");
13     }
14 }
```

```
> ltrace ./pass
printf("%s", "Introduce la password: ") = 23
__isoc99_scanf(0x558199c6701c, 0x7fffb7c64e10, 0, 0Introduce la password: medaigual
) = 1
strcmp("medaigual", "impossible_password") = 4
puts("Oh! que pena... no era esa"0h! que pena... no era esa
) = 27
+++ exited (status 0) +++
```

- El argumento "-e" sirve para especificar el nombre de una llamada a librería/sistema. De esta manera solo se recogería el output de dicha llamada.

```
> ltrace -e strcmp ./pass
Introduce la password: prueba
pass->strcmp("prueba", "impossible_password")      = 7
Oh! que pena... no era esa
+++ exited (status 0) +++
```

- El argumento "-i" imprime a su vez la dirección de la instrucción que se está ejecutando

```
> ltrace -i -e strcmp ./pass
Introduce la password: prueba
[0x55bc9cd071b0] pass->strcmp("prueba", "impossible_password")      = 7
Oh! que pena... no era esa
[0xffffffffffffffff] +++ exited (status 0) +++
```

- El argumento "--output=<fichero>" sirve para especificar la ruta del output en caso de que queramos que se guarde.

```
> ltrace -i -e strcmp --output=mi_output ./pass
Introduce la password: prueba
Oh! que pena... no era esa
> cat mi_output
```

	File: mi_output
1	[0x560369fbf1b0] pass->strcmp("prueba", "impossible_password")      = 7
2	[0xffffffffffffffff] +++ exited (status 0) +++



# TU VIEJO AMIGO

---

# COMANDO FILE

## ¿Para qué sirve?

Devuelve el tipo de archivo con el que estás trabajando

```
> file hello
hello: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=55e52702e9ee717c03dec425df6da9623ff5531, for GNU/Linux 3.2.0, not stripped
```

## ¿Cómo funciona?

- Primeramente, con la llamada al sistema `stat` comprueba si es un fichero, directorio, link...
- Seguidamente, hace uso de los magic bytes (¿os suenan?) para analizar qué tipo de archivo es `/usr/share/misc/magic.mgc`
- Finalmente, realiza un análisis de la sintaxis del fichero para terminar de determinar el tipo



# ANÁLISIS ELF

---

# ALGUNAS HERRAMIENTAS



IDA Pro



30



# EJECUTAR CUTTER

```
~  
> cd Desktop  
~/Desktop  
> chmod +x Cutter.AppImage  
~/Desktop  
> ./Cutter.AppImage  
"0.3.0" "0.3.0"  
Setting PYTHONHOME = "/tmp/.mount_Cutter  
PYTHONHOME = "/tmp/.mount_CutterRNWJtu/u  
Setting Rizin prefix = "/tmp/.mount_Cutt  
Setting Rizin plugins dir = "/tmp/.mount  
Plugins are loaded from "/home/rsgbengi/  
Loaded 0 plugin(s).  
Plugins are loaded from "/home/rsgbengi/  
Plugins are loaded from "/var/lib/flatpa  
Plugins are loaded from "/tmp/.mount_Cut  
Plugins are loaded from "/tmp/.mount_Cut  
Loaded 1 plugin(s).  
Plugins are loaded from "/var/lib/snapd/
```

## ¿Cómo se inicia Cutter?

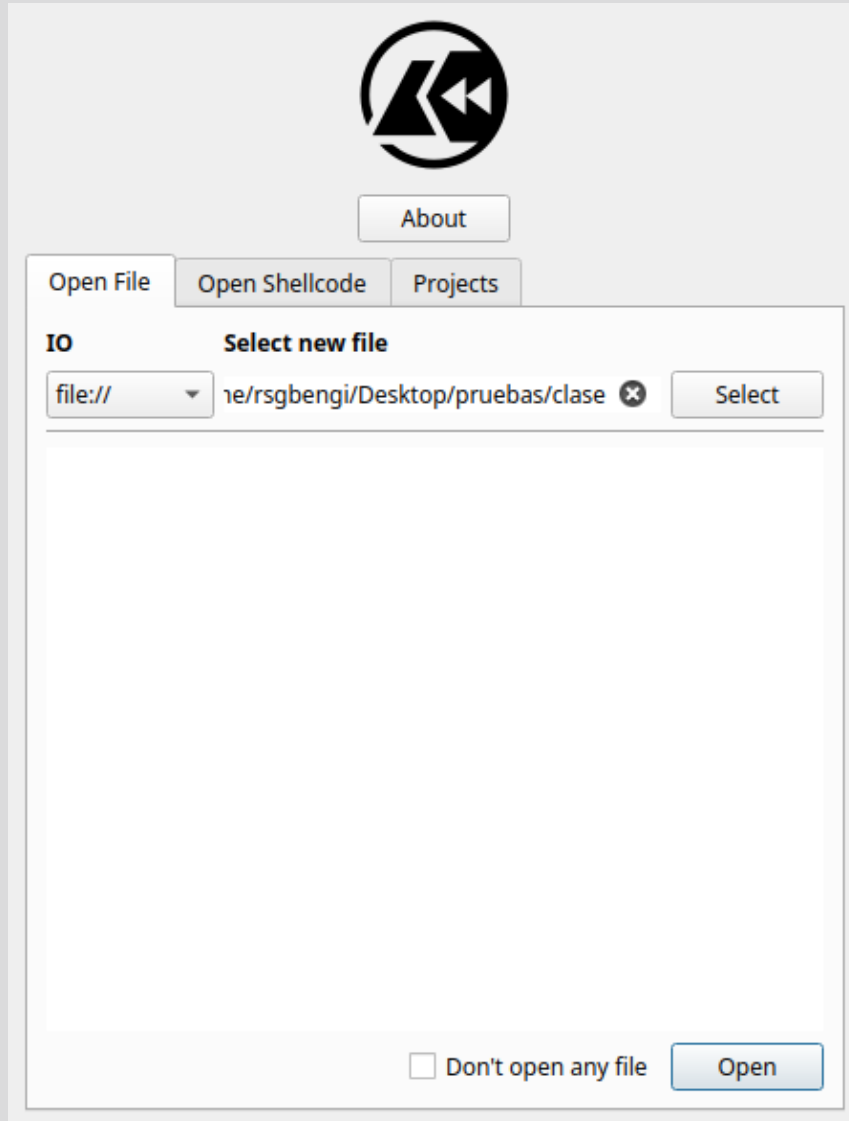
Nos dirigimos a la ruta donde hayamos descargado el archivo, damos permisos y lo ejecutamos:

- `chmod +x Cutter.AppImage`
- `./Cutter.AppImage`

# ¿CÓMO SE CARGA UN FICHERO?

## Carga de fichero

- Pulsamos select → binario que se va a analizar
- En este caso el binario es "clase"



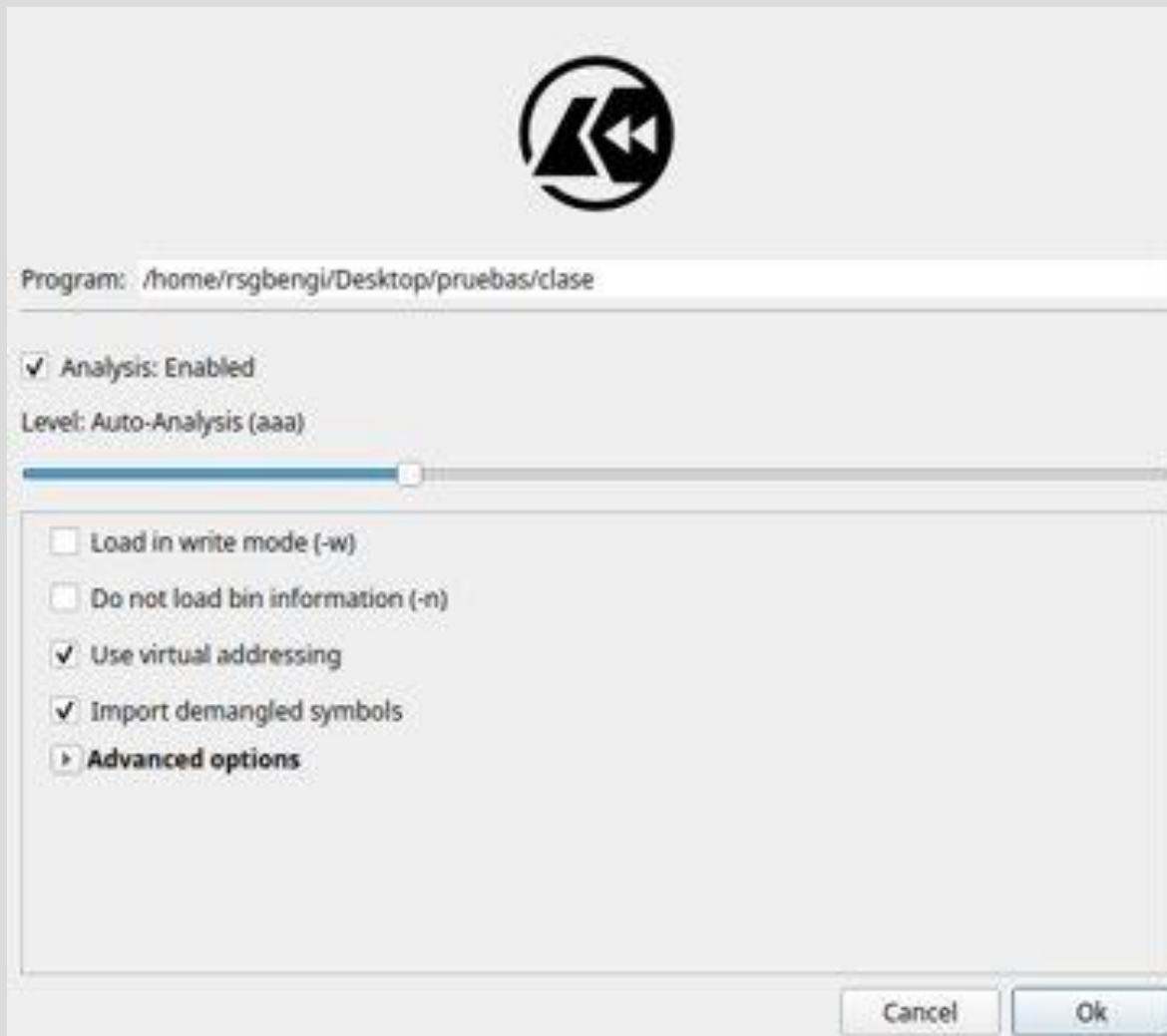


# OPCIONES DE ANÁLISIS

## Análisis

En este apartado se muestran las diferentes opciones de análisis.

En nuestro caso lo dejamos por defecto.



# FUNCIONES Y DASHBOARD

entry.init0	6	false
entry0	46	false
loc..annobin_lto_1	5	false
<b>main</b>	<b>150</b>	<b>false</b>
sym.__do_global_dtors_aux	33	false
sym._libc_csu_fini	5	false
sym._libc_csu_init	101	false
sym._fini	13	false
sym._init	27	false
sym.deregister_tm_clones	33	false
sym.imp.__isoc99_scanf	6	true
sym.imp.printf	6	true
sym.imp.puts	6	true
sym.register_tm_clones	49	false
sym.secret	22	false

Por un lado, tenemos las funciones que ha detectado

Por otro lado, en la sección "dashboard" tenemos información sobre el binario

Dashboard

### OVERVIEW

/home/rsgbengi/Desktop/prueb	FD:	3	Archite
elf64	Base addr:	0x00400000	Machin
64	Virtual addr:	True	OS:
ELF64	Canary:	False	Subsys
r-x	Crypto:	False	Strippe
23.9 kB	NX bit:	N/A	Relocs:
N/A	PIC:	N/A	Endian
C	Static:	False	Compil
	Relro:	Partial	Compil

Certificates Version info

### es

Libr

54e254d43f9a622eef3e6e1ad53f366b

375801eb5f25811c70864cc16dc89a5897d2a12c

0914ccf7d7a12c7976da1a599c414511feae9695698f380795821ef2f92fdf44

octal, w=32bit, q=64bit | pz[?] [len] print zoom view (see pz? for help)

# STRINGS

Address	String
0x00000034	@8\r@
0x00400318	/lib64/ld-linux-x86-64.so.2
0x00400439	puts
0x0040043e	printf
0x00400445	__libc_start_main
0x00400457	libc.so.6
0x00400461	GLIBC_2.2.5
0x0040046d	__gmon_start__
0x00401095	H=0@@
0x00401172	p/ @
0x00401209	\b[]A\VA]A^A_
0x00402010	No tengo amigos T.T
0x00402024	Hola mundo
0x0040202f	hola %d\n
0x00402038	This is my super secret:
0x00402052	Fuera :(

En esta sección podemos localizar cadenas de texto en el binario.

Normalmente es lo primero que vamos a hacer en un CTF para intentar encontrar cosas como **password**, **user**, **flag**...

Debajo hay un filtro para buscar la cadena que queramos

# IMPORTS

Tiene las funciones de librerías que usa el binario

0x00000000	NOTYPE	__gmon_start__	[30] ---- section size 300 named .shstrtab
0x00401050	FUNC	__isoc99_scanf	
0x00000000	FUNC	__libc_start_main	[30] ---- section size 300 named .shstrtab
0x00401040	FUNC	printf	
0x00401030	FUNC	puts	

# TRAUMA (DESENSAMBLADOR)

## Código en ensamblador

```

int main (int argc, char **argv, char **envp);
; var int64_t var_ch @ rbp-0xc
; var int64_t var_8h @ rbp-0x8
; var int var_4h @ rbp-0x4
0x0040115c      push rbp
0x0040115d      mov rbp, rsp
0x00401160      sub rsp, 0x10
0x00401164      mov edi, str.Hola_mundo ; 0x402024 ; const c
0x00401169      call puts          ; sym.imp.puts ; int puts
0x0040116e      mov dword [var_ch], 0
0x00401175      mov dword [var_8h], 4
0x0040117c      mov dword [var_4h], 0
0x00401183      jmp 0x40119d
0x00401185      mov eax, dword [var_4h]
0x00401188      mov esi, eax
0x0040118a      mov edi, str.hola__d ; 0x40202f ; const char
0x0040118f      mov eax, 0
0x00401194      call printf        ; sym.imp.printf ; int pr
0x00401199      add dword [var_4h], 1
0x0040119d      cmp dword [var_4h], 9
0x004011a1      jle 0x401185
0x004011a3      mov edi, str.Introduzca_un_numero_para_saber
0x004011a8      call puts          ; sym.imp.puts ; int puts
0x004011ad      lea rax, [var_ch]
0x004011b1      mov rsi, rax
0x004011b4      mov edi, 0x402066 ; 'f @' ; const char *for
0x004011b8      mov rax, 0

```

Normalmente los retos de reversing se basan en leer código en ensamblador y ver cómo se va ejecutando el programa

## Código en ensamblador visualización en modo grafo

Para visualizar el flujo de ejecución de una manera más "amigable"

```
[0x0040115c]
int main (int argc, char **argv, char **envp);
; var int64_t var_ch @ rbp-0xc
; var int64_t var_8h @ rbp-0x8
; var int var_4h @ rbp-0x4
push    rbp
mov     rbp, rsp
sub     rsp, 0x10
mov     edi, str.Hola_mundo      ; 0x402024 ; const char *s
call    puts                    ; sym.imp.puts ; int puts(const char *s)
mov     dword [var_ch], 0
mov     dword [var_8h], 4
mov     dword [var_4h], 0
jmp     0x40119d
```

```
[0x0040119d]
cmp     dword [var_4h], 9
jle     0x401185
```

```
[0x00401185]
mov     eax, dword [var_4h]
mov     esi, eax
mov     edi, str.hola__d
mov     eax, 0
call    printf
add     dword [var_4h], 1
```

```
ara_saber_mi_secreto... ; 0x402038 ; const char *s
i.imp.puts ; int puts(const char *s)

@' ; const char *format
i.imp.__isoc99_scanf ; int scanf(const char *format)
```

# LO QUE HARÁ REVERSING MÁS FÁCIL: DECOMPILADOR

```
// WARNING: Could not reconcile some variable overlaps  
// WARNING: [rz-ghidra] Detected overlap for variable var_8h  
// WARNING: [rz-ghidra] Detected overlap for variable var_4h
```

```
undefined8 main(void)  
{  
    int64_t var_ch;  
    int32_t var_4h;  
  
    puts("Hola mundo");  
    var_ch._0_4_ = 0;  
    var_ch._4_4_ = 4;  
    for (var_4h = 0; var_4h < 10; var_4h = var_4h + 1) {  
        printf("hola %d\n", var_4h);  
    }  
    puts("Introduzca un numero para saber mi secreto...");  
    __isoc99_scanf(0x402066, &var_ch);  
    if ((int32_t)var_ch == 5) {  
        puts("This is my super secret: ");  
        secret();  
    } else {  
        puts("Fuera :( ");  
    }  
    return 0;  
}
```

## Decompilador

El decompilador de Ghidra  
pasa de código en  
ensamblador a una especie de  
pseudocódigo en C/C++

Esto hace que analizar el  
binario sea menos trambólico

```
undefined8 main(void)
{
    int64_t entrada_usuario;
    int32_t contador;

    puts("Hola mundo");
    entrada_usuario._0_4_ = 0;
    entrada_usuario._4_4_ = 4;
    for (contador = 0; contador < 10; contador = contador + 1)
        printf("hola %d\n", contador);
    }
    puts("Introduzca un numero para saber mi secreto...")
    __isoc99_scanf(0x402066, &entrada_usuario);
    if ((int32_t)entrada_usuario == 5) {
        puts("This is my super secret: ");
        secret();
    } else {
        puts("Fuera :( ");
    }
    return 0;
}
```

## Decompilador

Click derecho sobre la variable  
→ Rename para cambiarla de  
nombre cuando sepamos más  
o menos que hace

Esto hará que el análisis sea  
más sencillo



# EJECUTAR EL BINARIO

## Decompilador

```
> ./clase  
Hola mundo  
hola 0  
hola 1  
hola 2  
hola 3  
hola 4  
hola 5  
hola 6  
hola 7  
hola 8  
hola 9  
Introduzca un numero para saber mi secreto...  
3  
Fuera :(
```

Al ejecutarlo vemos que hace:

- Muestra "hola mundo"
- Realiza el "for" tal y como habíamos visto
- Pide entrada de usuario
- Al fallar el número nos devuelve "Fuera :("

# ¿SOLUCIÓN?

```
nota 5
Introduzca un numero para saber mi secreto...
5
This is my super secret:
No tengo amigos T.T%
```

```
if ((int32_t)entrada_usuario == 5) {
    puts("This is my super secret: ");
    secret();
} else {
    puts("Fuera :( ");
}
return 0;
```

## Introducir el número 5

Simplemente  
introduciendo el número 5  
podríamos acceder dentro  
de una condición que no  
deberíamos poder

# HEXDUMP

## Hexdump

En muchas ocasiones  
puede ser de utilidad  
analizar el código en  
hexadecimal

```

0x000000000000401000 f3 0f 1e fa 48 83 ec 08 48 8b 05 e9 2f 00 00 48 85 c0 74 02 ff d0 48 83 c4 08 c3 00 00 00 00 00
0x000000000000401020 ff 35 e2 2f 00 00 ff 25 e4 2f 00 00 0f 1f 40 00 ff 25 e2 2f 00 00 68 00 00 00 e9 e0 ff ff ff
0x000000000000401040 ff 25 da 2f 00 00 68 01 00 00 e9 d0 ff ff ff ff 25 d2 2f 00 00 68 02 00 00 e9 c0 ff ff ff
0x000000000000401060 f3 0f 1e fa 31 ed 49 89 d1 5e 48 89 e2 48 83 e4 f0 50 54 49 c7 c0 70 12 40 00 48 c7 c1 00 12 40
0x000000000000401080 00 48 c7 c7 5c 11 40 00 ff 15 62 2f 00 00 f4 90 f3 0f 1e fa c3 66 2e 0f 1f 84 00 00 00 00 00 90
0x0000000000004010a0 b8 38 40 40 00 48 3d 38 40 40 00 74 13 b8 00 00 00 48 85 c0 74 09 bf 38 40 40 00 ff e0 66 90
0x0000000000004010c0 c3 66 66 2e 0f 1f 84 00 00 00 00 0f 1f 40 00 be 38 40 40 00 48 81 ee 38 40 40 00 48 89 f0 48
0x0000000000004010e0 c1 ee 3f 48 c1 f8 03 48 01 c6 48 d1 fe 74 11 b8 00 00 00 48 85 c0 74 07 bf 38 40 40 00 ff e0
0x000000000000401100 c3 66 66 2e 0f 1f 84 00 00 00 00 0f 1f 40 00 f3 0f 1e fa 80 3d 19 2f 00 00 75 13 55 48 89
0x000000000000401120 e5 e8 7a ff ff ff c6 05 07 2f 00 00 01 5d c3 90 c3 66 66 2e 0f 1f 84 00 00 00 00 0f 1f 40 00
0x000000000000401140 f3 0f 1e fa eb 8a 55 48 89 e5 bf 10 20 40 00 b8 00 00 00 e8 e7 fe ff ff 90 5d c3 5 48 89 e5
0x000000000000401160 48 83 ec 10 bf 24 20 40 00 e8 c2 fe ff ff c7 45 f4 00 00 00 c7 45 f8 04 00 00 c7 45 fc 00
0x000000000000401180 00 00 00 eb 18 8b 45 fc 89 c6 bf 2f 20 40 00 b8 00 00 00 e8 a7 fe ff ff 83 45 fc 01 83 7d fc
0x0000000000004011a0 09 7e e2 bf 38 20 40 00 e8 83 fe ff ff 48 8d 45 f4 48 89 c6 bf 66 20 40 00 b8 00 00 00 e8 8d
0x0000000000004011c0 fe ff ff 8b 45 f4 83 f8 05 75 16 bf 69 20 40 00 e8 5b fe ff ff b8 00 00 00 e8 67 ff ff ff eb
0x0000000000004011e0 0a bf 83 20 40 00 e8 45 fe ff ff b8 00 00 00 c9 c3 66 2e 0f 1f 84 00 00 00 00 0f 1f 40 00
0x000000000000401200 f3 0f 1e fa 41 57 4c 8d 3d 03 2c 00 00 41 56 49 89 d6 41 55 49 89 f5 41 54 41 89 fc 55 48 8d 2d
0x000000000000401220 f4 2b 00 00 53 4c 29 fd 48 83 ec 08 e8 cf fd ff ff 48 c1 fd 03 74 1f 31 db 0f 1f 80 00 00 00 00
0x000000000000401240 4c 89 f2 4c 89 ee 44 89 e7 41 ff 14 df 48 83 c3 01 48 39 dd 75 ea 48 83 c4 08 5b 5d 41 5c 41 5d
0x000000000000401260 41 5e 41 5f c3 66 66 2e 0f 1f 84 00 00 00 00 f3 0f 1e fa c3 00 00 00 f3 0f 1e fa 48 83 ec 08
0x000000000000401280 48 83 c4 08 c3 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x0000000000004012a0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x0000000000004012c0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
0x0000000000004012e0 ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff

```



# ANÁLISIS .NET

---



## Diferencias EXE/DLL y ELF

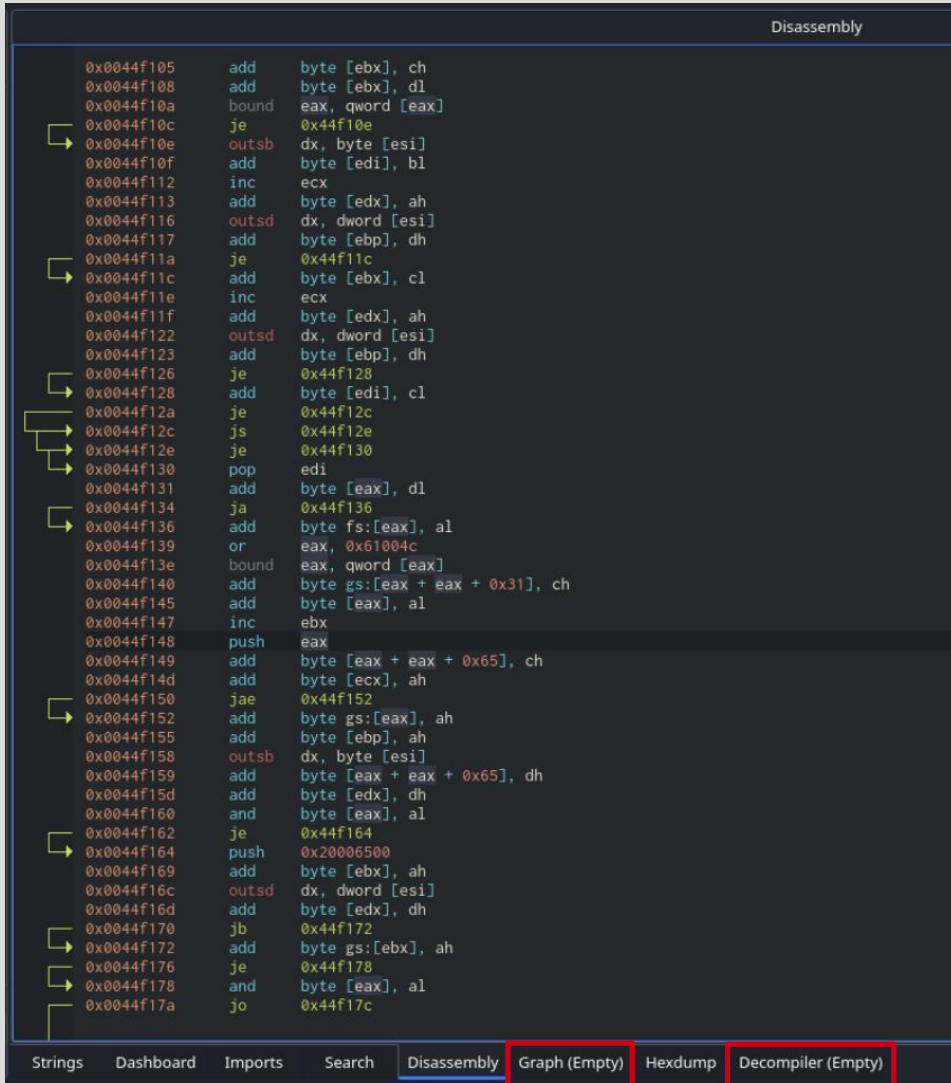
- Arquitectura diferente de código
- El código ensamblador que se genera es diferente (CIL)
- Similar al concepto de Java y su JVM
- Necesitamos herramientas específicas para decompilar y analizar este tipo de binarios

## ¿Cómo lo detectamos?

- Comando file:  
probablemente nos muestre algo similar a la imagen
- Pistas: cadena .NET assembly
- PE32, PE64... for MS Windows

```
(kali@kali)-[~/Downloads]
$ file CrackMe\#1-InfoSecInstitute-dotNET-Reversing.exe
CrackMe\#1-InfoSecInstitute-dotNET-Reversing.exe: PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows

(kali@kali)-[~/Downloads]
$
```



```
0x0044f105 add byte [ebx], ch
0x0044f108 add byte [ebx], dl
0x0044f10a bound eax, qword [eax]
0x0044f10c je 0x44f10e
0x0044f10e outsb dx, byte [esi]
0x0044f10f add byte [edi], bl
0x0044f112 inc ecx
0x0044f113 add byte [edx], ah
0x0044f116 outsd dx, dword [esi]
0x0044f117 add byte [ebp], dh
0x0044f11a je 0x44f11c
0x0044f11c add byte [ebx], cl
0x0044f11e inc ecx
0x0044f11f add byte [edx], ah
0x0044f122 outsd dx, dword [esi]
0x0044f123 add byte [ebp], dh
0x0044f126 je 0x44f128
0x0044f128 add byte [edi], cl
0x0044f12a je 0x44f12c
0x0044f12c js 0x44f12e
0x0044f12e je 0x44f130
0x0044f130 pop edi
0x0044f131 add byte [eax], dl
0x0044f134 ja 0x44f136
0x0044f136 add byte fs:[eax], al
0x0044f139 or eax, 0x61004c
0x0044f13e bound eax, qword [eax]
0x0044f140 add byte gs:[eax + eax + 0x31], ch
0x0044f145 add byte [eax], al
0x0044f147 inc ebx
0x0044f148 push eax
0x0044f149 add byte [eax + eax + 0x65], ch
0x0044f14d add byte [ecx], ah
0x0044f150 jae 0x44f152
0x0044f152 add byte gs:[eax], ah
0x0044f155 add byte [ebp], ah
0x0044f158 outsb dx, byte [esi]
0x0044f159 add byte [eax + eax + 0x65], dh
0x0044f15d add byte [edx], dh
0x0044f160 and byte [eax], al
0x0044f162 je 0x44f164
0x0044f164 push 0x20006500
0x0044f169 add byte [ebx], ah
0x0044f16c outsd dx, dword [esi]
0x0044f16d add byte [edx], dh
0x0044f170 jnb 0x44f172
0x0044f172 add byte gs:[ebx], ah
0x0044f176 je 0x44f178
0x0044f178 and byte [eax], al
0x0044f17a jo 0x44f17c
```

## ¿Cómo lo analizamos?

Para Linux no existen demasiadas herramientas potentes para un análisis exhaustivo

Podemos utilizar alguna de las vistas anteriormente para echarle un primer vistazo al binario

Nos mudamos a Windows



## Herramientas específicas

En Windows existen herramientas más específicas para este tipo de binarios

Algunas de ellas son:

- DnSpy (<https://github.com/dnSpy/dnSpy>)
- ILSpy (<https://github.com/icsharpcode/ILSpy>)
- DotPeek (<https://www.jetbrains.com/es-es/decompiler/>)

Podéis elegir aquella con la que os sintáis más cómodos



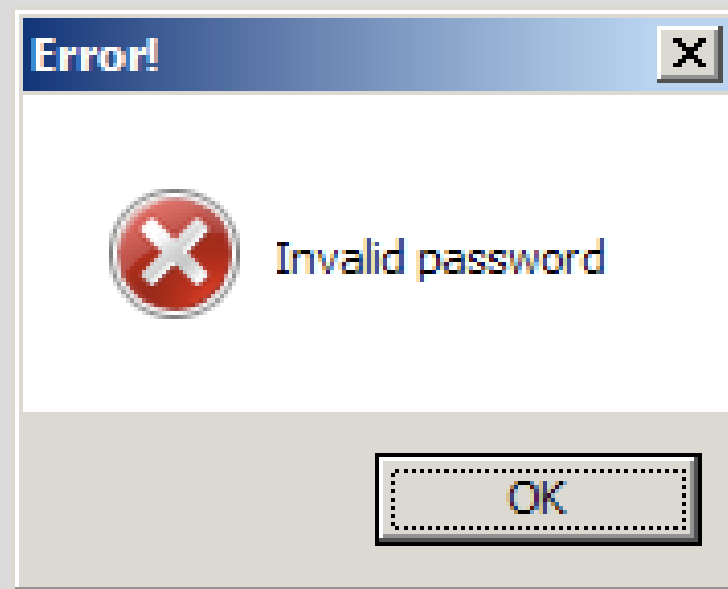
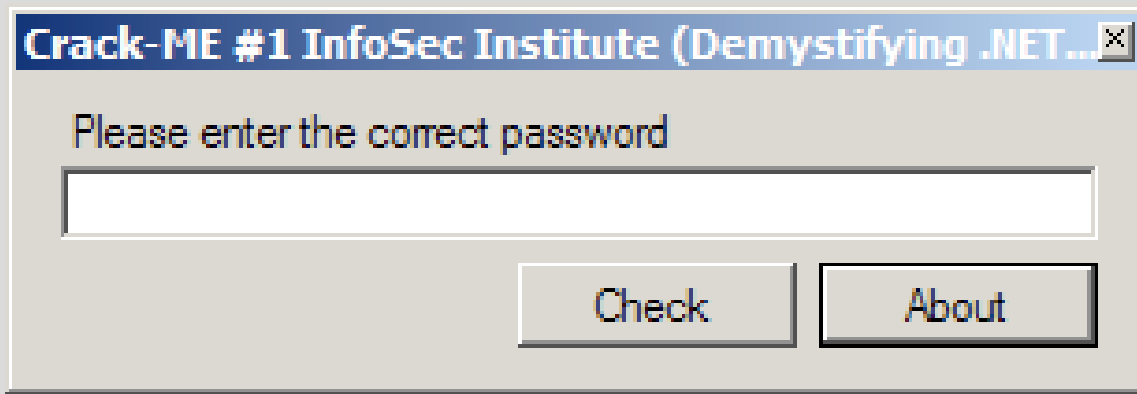
# ANÁLISIS .NET - EJEMPLO

## Inicio del análisis

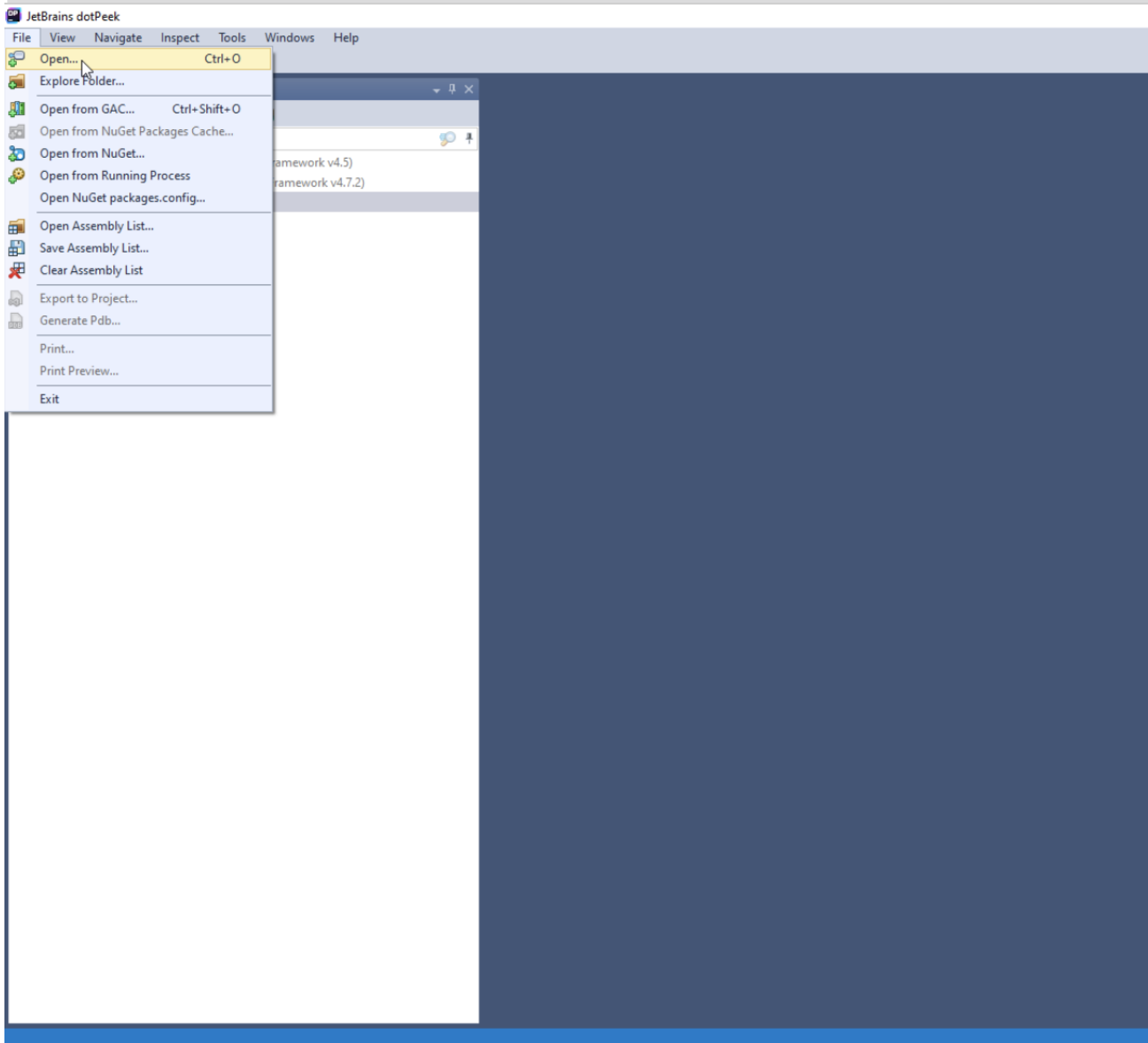
Analizaremos una aplicación sencilla creada por InfoSecInstitute (<https://www.infosecinstitute.com/>)

El primer paso, como siempre, es intentar ver el comportamiento del binario ejecutándolo

**¡OJO! SIEMPRE EN UNA MÁQUINA VIRTUAL, POR LO QUE PUEDA PASAR**



# ANÁLISIS .NET - EJEMPLO



## Decompilando ando

Ahora que tenemos una idea de cómo funciona, nos vamos al decompilador que más nos guste

Lo primero que hacemos es cargar el fichero al decompilador

- File > Open > Selección de nuestro ejecutable

# ANÁLISIS .NET - EJEMPLO

## Decompilando ando

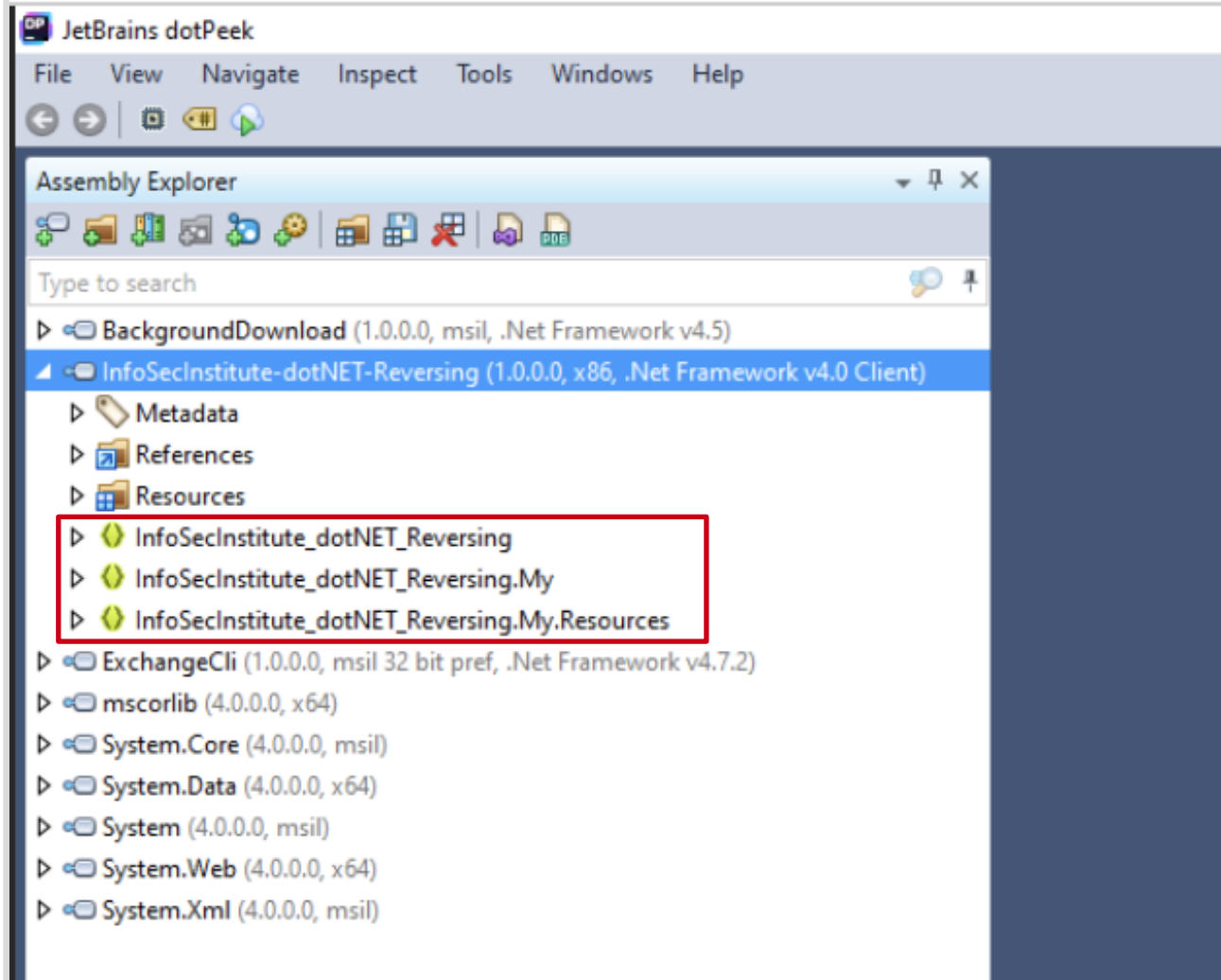
Una vez cargado nuestro ejecutable, desplegamos sus elementos y podemos ver diferentes secciones

**Metadata:** Contiene metadatos del ejecutable. Puede que encontremos aquí la flag si no está cifrada o muy escondida

**References:** Nos da una idea de cómo está construido el programa

**Resources:** Iconos, formularios... que contiene la aplicación

**Bloques de código:** la parte que realmente nos interesa



# ANÁLISIS .NET - EJEMPLO

The screenshot displays the Visual Studio IDE with the Assembly Explorer on the left and the source code of `InfoSecInstitute_dotNET_Reversing.Form1.resx` on the right. The Assembly Explorer shows the project structure, including references to `Microsoft.VisualBasic`, `mscorlib`, `System`, `System.Drawing`, and `System.Windows.Forms`, and resources like `Win32 resources` and `InfoSecInstitute_dotNET_Reversing.Form1.resources`. The source code on the right shows the `Form1` class with properties `txt_Pwd` and `Label1`, and two event handlers: `btn_Chk_Click` and `btn_About_Click`. The `btn_Chk_Click` method is highlighted with a red box.

```
EventHandler eventHandler = new EventHandler(this.btn_About_Click);
if (this._btn_About != null)
    this._btn_About.Click -= eventHandler;
this._btn_About = value;
if (this._btn_About == null)
    return;
this._btn_About.Click += eventHandler;
}

internal virtual TextBox txt_Pwd
{
    get => this._txt_Pwd;
    [MethodImpl(MethodImplOptions.Synchronized)] set => this._txt_Pwd = value;
}

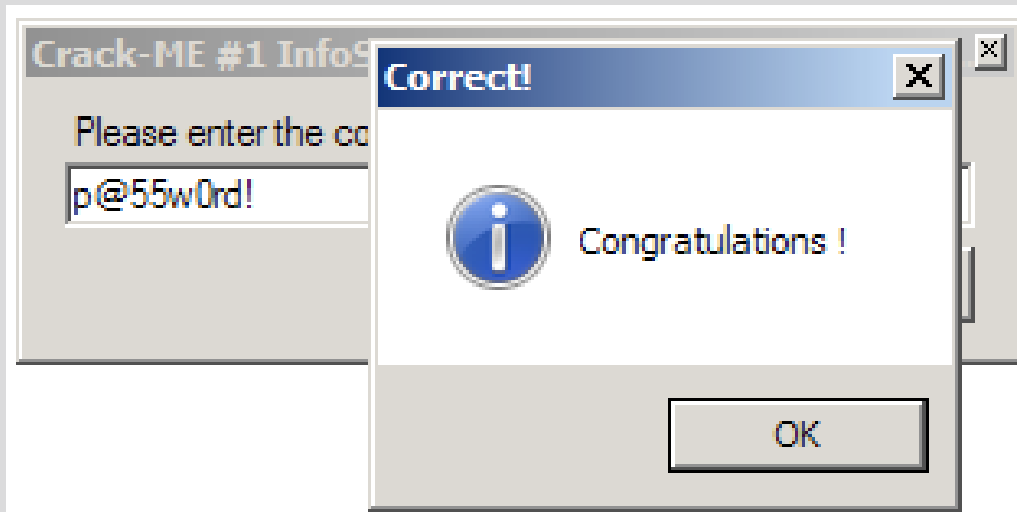
internal virtual Label Label1
{
    get => this._Label1;
    [MethodImpl(MethodImplOptions.Synchronized)] set => this._Label1 = value;
}

private void btn_Chk_Click(object sender, EventArgs e)
{
    if (Operators.CompareString(this.txt_Pwd.Text, "p@55w@rd!", false) == 0)
    {
        int num1 = (int) Interaction.MsgBox((object) "Congratulations !", MsgBoxStyle.Information, (object) "Correct!");
    }
    else
    {
        int num2 = (int) Interaction.MsgBox((object) "Invalid password", MsgBoxStyle.Critical, (object) "Error!");
    }
}

private void btn_About_Click(object sender, EventArgs e)
{
    int num = (int) Interaction.MsgBox((object) "Crack Me #1 for \"Demystifying dot NET reverse engineering\" series on InfoSec
}
```

# ANÁLISIS .NET - EJEMPLO

## Obtenemos la flag



- Este es un ejemplo muy simple para que el proceso se entienda
- Habitualmente en un CTF no encontraremos la flag de forma tan sencilla
- Probablemente esté oculta de alguna forma (XOR, AES...) y tengamos que hacernos con ella



# IV. Reversing

---

Carlos Alonso, Alejandro Cruz, Andrea Oliva, Sergio Pérez y Rubén Santos