Módulo IV: Ingeniería Inversa & Exploiting

Pablo Pastor, Diego Palacios & David Billdhart

Universidad
Rey Juan Carlos

¿Qué es la Ingeniería Inversa?

Universidad
Rey Juan Carlos

Ingeniería Inversa

Realizar **ingeniería inversa** permite obtener información sobre el funcionamiento interno de un archivo o programa.

¿Cómo?

- Leyendo el código fuente (en caso de tenerlo)
- Desensamblando y decompilando
- Depurando
- Ejecución simbólica

“El reversing es más importante que el Inglés”

Análisis de código fuente

Universidad
Rey Juan Carlos

Análisis de código fuente

El **análisis de código** fuente se basa en la lectura del código fuente original de la aplicación.

Python3

```python
def main():
    l4 = [70, 123, 100, 53, 123, 58, 105, 109, 2, 108, 116, 21, 67, 69, 238, 47, 102, 110, 114, 84, 83, 68, 113, 72, 112, 54, 121, 104, 103, 41, 124]
    flag_length = 31
    mod = 256
    seed = 1000
    random.seed(seed)
    l = []
    inp = [1] * flag_length
    n = len(inp)
    l2 = [0] * n
    l3 = [0] * n

    for i in range(n):
        l.append(random.randint(6, 420))

    l3[0] = l2[0] % mod

    for i in range(1, n):
        l3[i] = (l2[i] ^ ((l[i] & l3[i - 1] + (l3[i - 1] * l[i]) % mod) // 2)) % mod

    result = combine(lst)

    print("flag is:", decrypt(result))
    l2[0] += int(recur(l2[1:]) * 50)

main()

```

Análisis de código fuente

El **análisis de código** fuente se basa en la lectura del código fuente original de la aplicación.

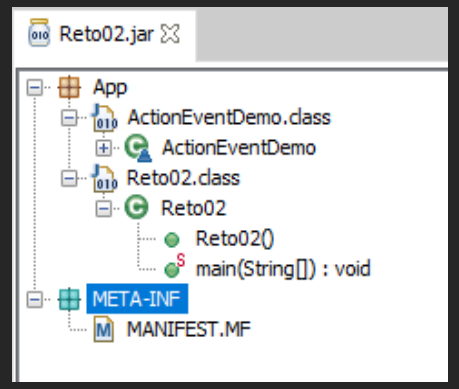
JavaScript

```javascript
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

El **análisis de código** fuente se basa en la lectura del código fuente original de la aplicación.



Java APKs

```
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");
  }
}
}
```

Ofuscación

Muchos retos de este tipo tendrán su código fuente ofuscado, lo que dificultará su comprensión

lambda.py

```python
#!/usr/bin/env python3.10

import sys
sys.setrecursionlimit(10000000)

(lambda _0: _0(37))(lambda _1: (lambda _2: _2(lambda _3: lambda _4: _3 == _4))(lambda _5: (lambda _6: _6(lambda _7: lambda _8: _7 + _8))(lambda _9: (
    if _54(_67)
    else _58(_67)(_66(_64(_67)))))(lambda _68: (lambda _69: _69(_5))(lambda _70: (lambda _71: _71(0))(lambda _72: (lambda _73: _73(_70(_72)))(lambda _
    if _74(_97)
    else _78(_96(_97 - 1))([_80(_97)(_94(_97))])))(lambda _98: (lambda _99: _99(_5))(lambda _100: (lambda _101: _101(0))(lambda _102: (lambda _103: _
    if _104(_106(_114))
    else _110(_112(_113)(_114[1: ]))([_113[_114[0]]])))(lambda _115: (lambda _116: _116(_5))(lambda _117: (lambda _118: _118(0))(lambda _119: (lambda
    if _121(_123(_131))
    else _127([_130(_131[0])])(_129(_130)(_131[1: ]))))(lambda _132: (lambda _133: _133(_5))(lambda _134: (lambda _135: _135(0))(lambda _136: (lambda
    if _138(_140(_148))
    else _144([_148[: _147]])(_146(_147)(_148[_147: ]))))(lambda _149: (lambda _150: _150(lambda _151: lambda _152: _151 + _17('X')(_13(_9(_152)(-len
    if _159(_161(_176))
    else _165(_167(_176[0]))(_173(_175(_176[1: ])))))(lambda _177: (lambda _178: _178(print))(lambda _179: (lambda _180: _180('Give me the flag...'))
    if _349
    else _353)))(lambda _355: lambda _356: _356)))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))))
```

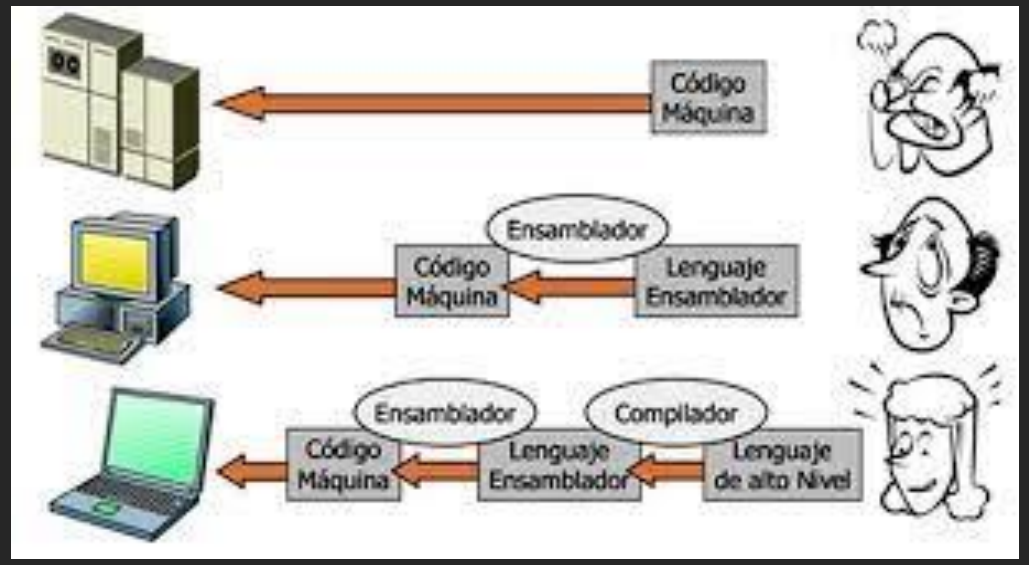
https://codebeautify.org/

Análisis de Binarios

Universidad
Rey Juan Carlos

Repaso

Un **binario** es el resultado final del proceso de compilación, que toma el código fuente legible por humanos y lo transforma en un formato binario que la computadora puede ejecutar directamente.



Comprobaciones previas

File

El comando **file** devuelve el tipo de archivo con el que estamos 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[s
ha1]=55e52702e9ee717c03decb425df6da9623ff5531, for GNU/Linux 3.2.0, not stripped
```

Strings

El comando **strings** busca y muestra todas las cadenas de caracteres imprimibles del binario

```
user@gh0st:~/cursoCTF/intro/compiled$ strings compiled
/lib64/ld-linux-x86-64.so.2
__cxa_finalize
__libc_start_main
puts
libc.so.6
GLIBC_2.2.5
GLIBC_2.34
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
Strings is the best CTF tool
:*3$"
GCC: (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Scrt1.o
__abi_tag
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_array_entry
frame_dummy
```

strace & ltrace

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
```

DEMO

File: hello.c

```c
#include <stdio.h>

int main(){
    printf("%s\n","Hello world!");
}
```

Argumentos

- 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) +++
```

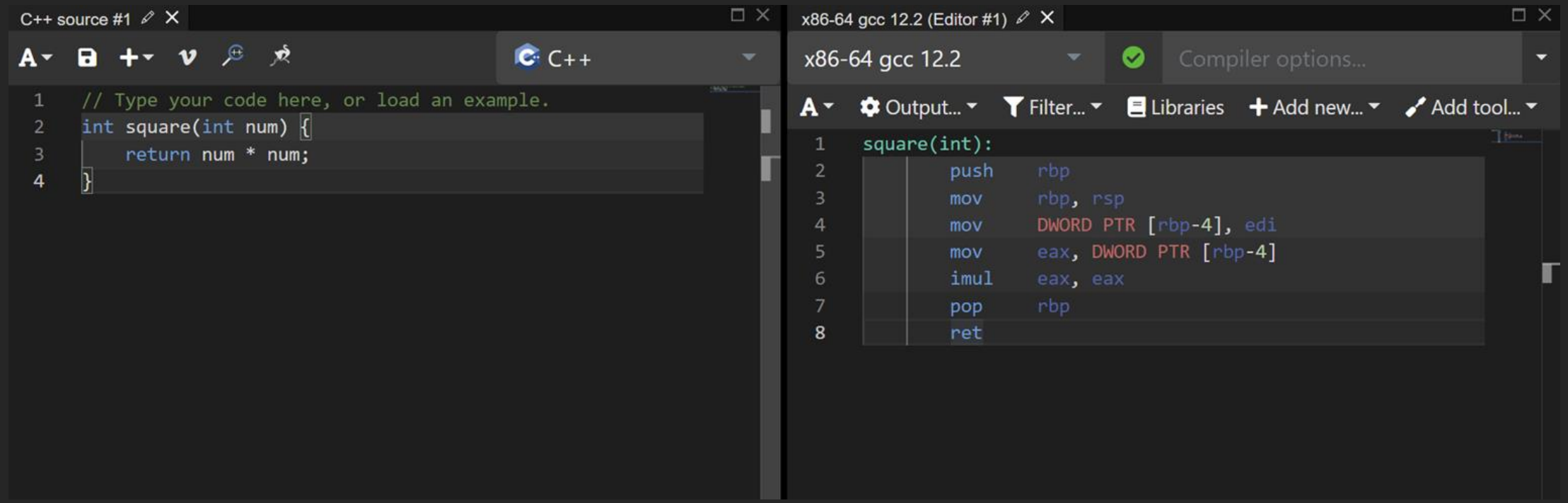
- El argumento "--output=<fichero>" sirve para especificar la ruta del output en caso de que queramos que se guarde.

```
❯ 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) +++
```

GodBolt

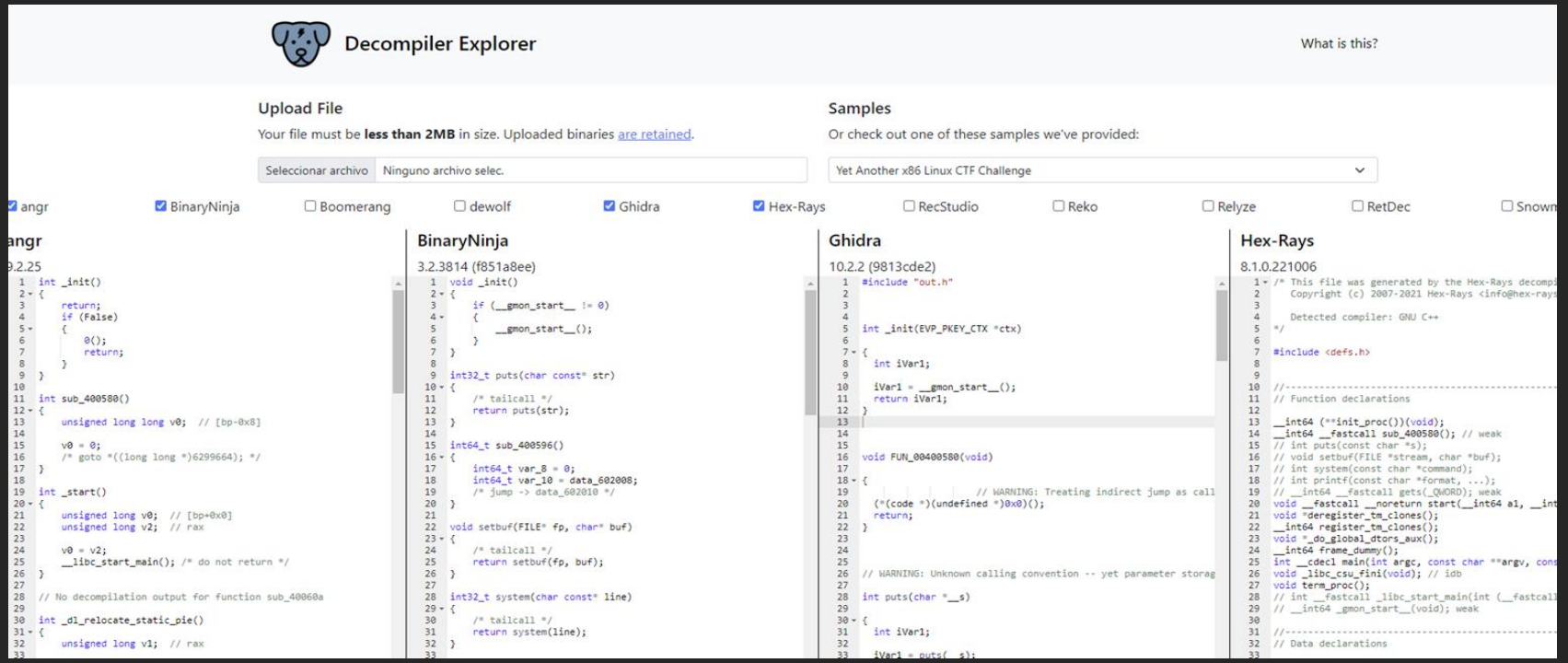
Godbolt AKA “Compiler Explorer” es una herramienta que permite introducir funciones en muchísimos lenguajes (C/C++, Go, Java, Assembly, Ada, C#....) y nos muestra la salida del ensamblador/bytecode resultante.



https://godbolt.org/

DogBolt

DogBolt AKA “Decompiler Explorer” cumple la función contraria a GodBolt. En esta página, se puede subir un binario (por ejemplo uno de los dados en un CTF) y comparar la salida del código decompilado por varios motores (por ejemplo Ghidra vs IDA vs BinaryNinja).



https://dogbolt.org/

Decompiladores

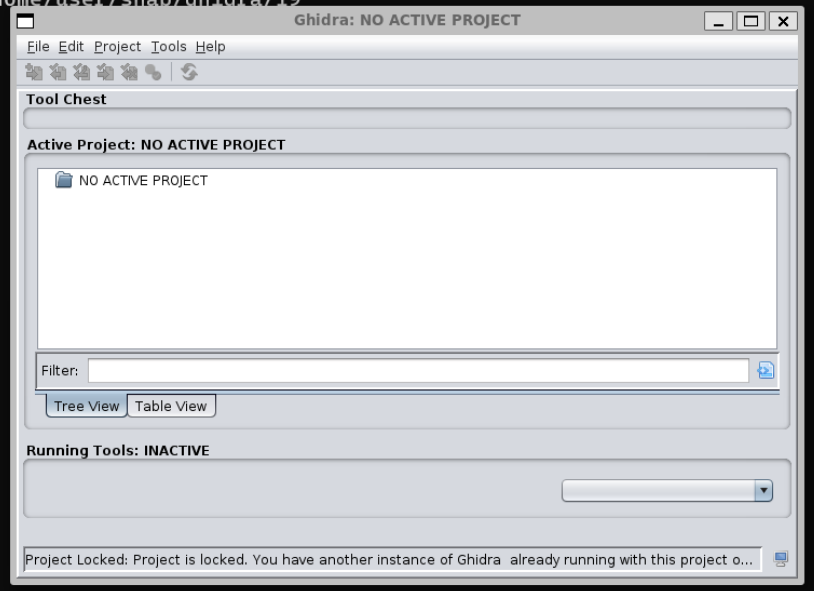
Universidad
Rey Juan Carlos

Decompiladores

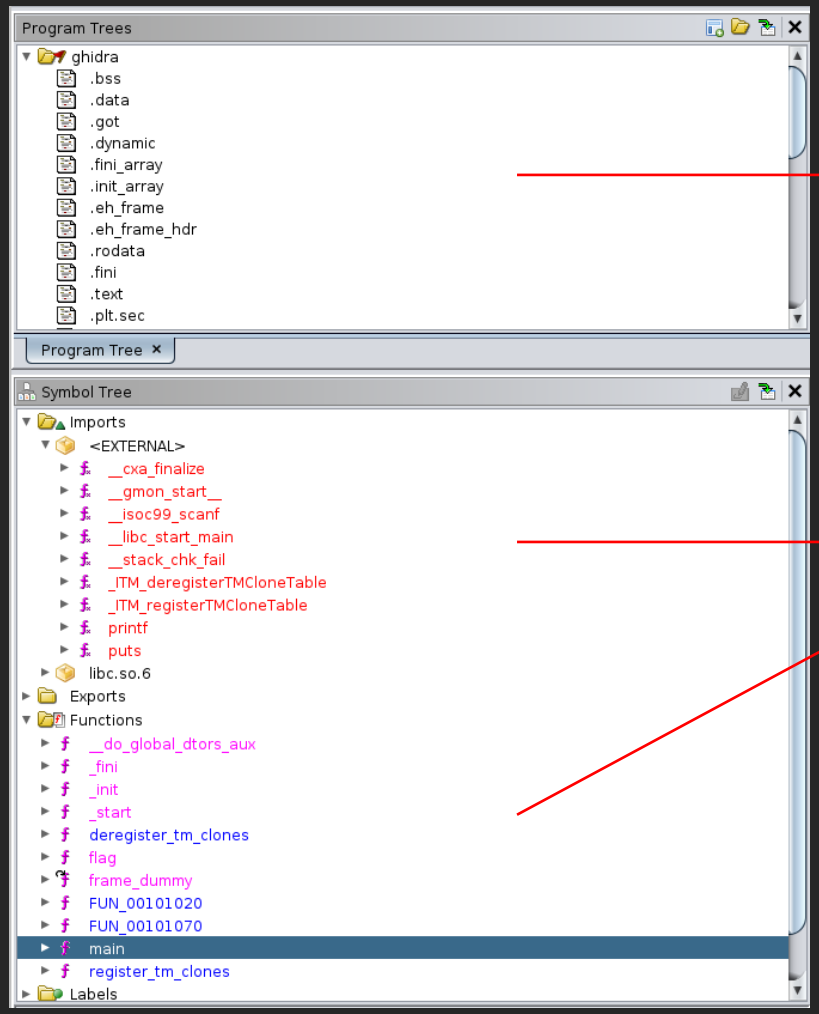
Instalación y Ejecución Ghidra

```
user@gh0st:~/cursoCTF$ sudo snap install ghidra
```

```
user@gh0st:~/cursoCTF/rev/compiled$ ghidra
2023/11/17 01:36:31.454431 cmd_run.go:1055: WARNING: cannot start document portal: dial unix /run/user/1000/bus: connect
: no such file or directory
Picked up _JAVA_OPTIONS: -Duser.home="/home/user/snap/ghidra/19"
Picked up _JAVA_OPTIONS: -Duser.home="/home/user/snap/ghidra/19"
user@gh0st:~/cursoCTF/rev/compiled$
```



Secciones, Funciones e Imports

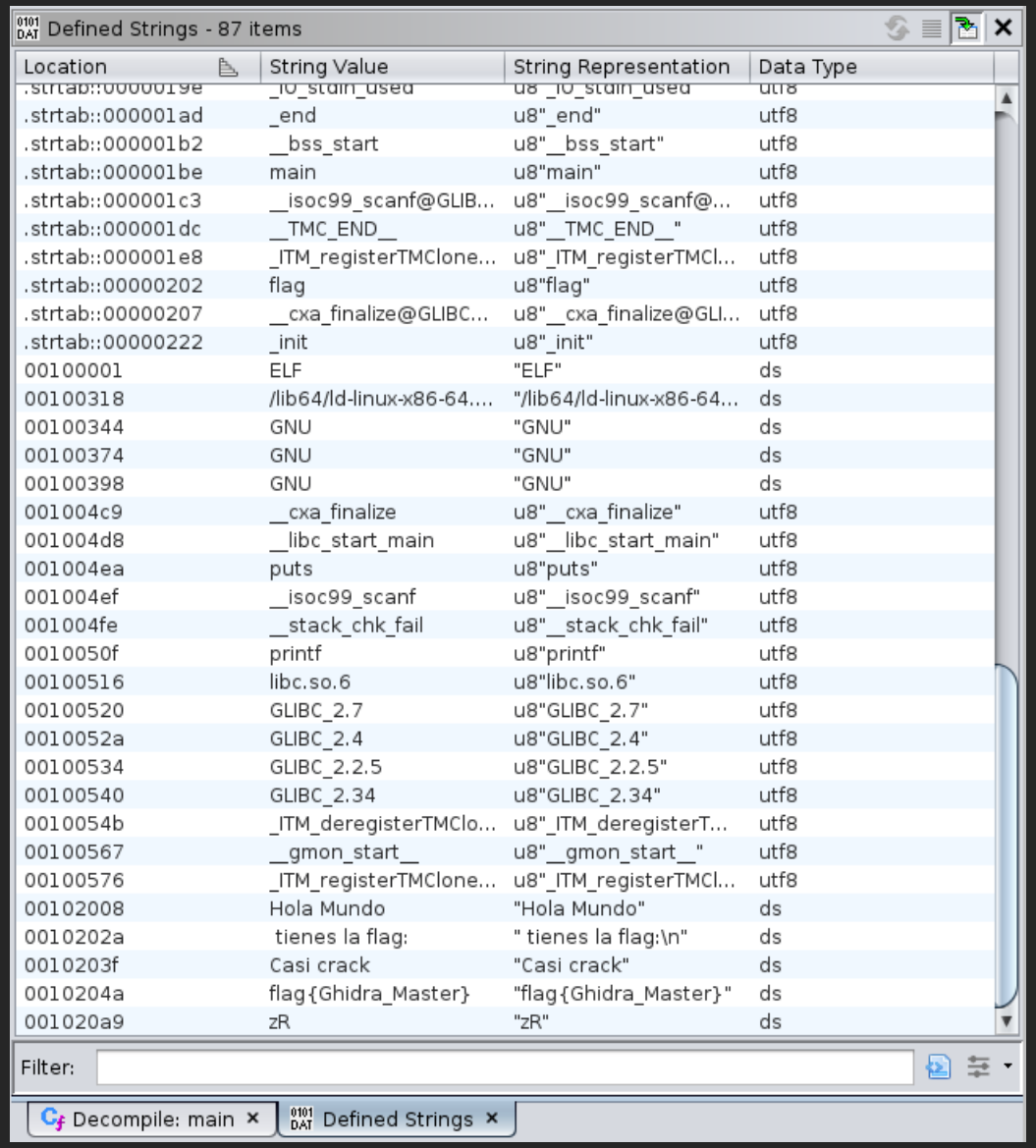


En la parte izquierda, encontramos 2 paneles:

- **Program Trees**: Contiene información sobre las secciones del binario

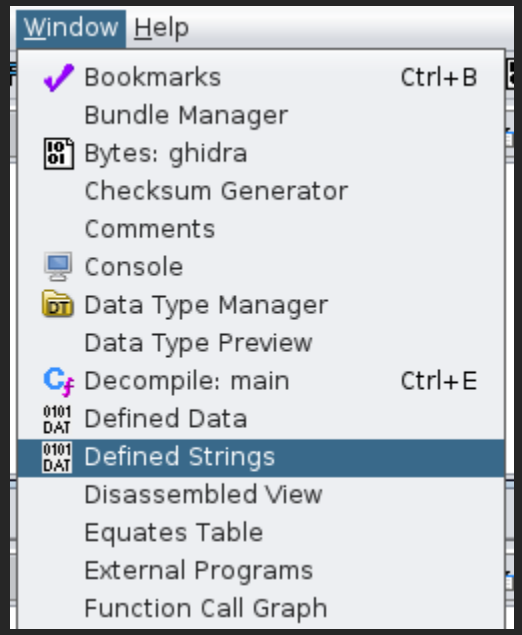
- **Symbol Tree**: Contiene las funciones e imports que se han detectado.

Strings



Para que Ghidra muestre los strings del binario deberemos irnos a:

Window -> Defined Strings

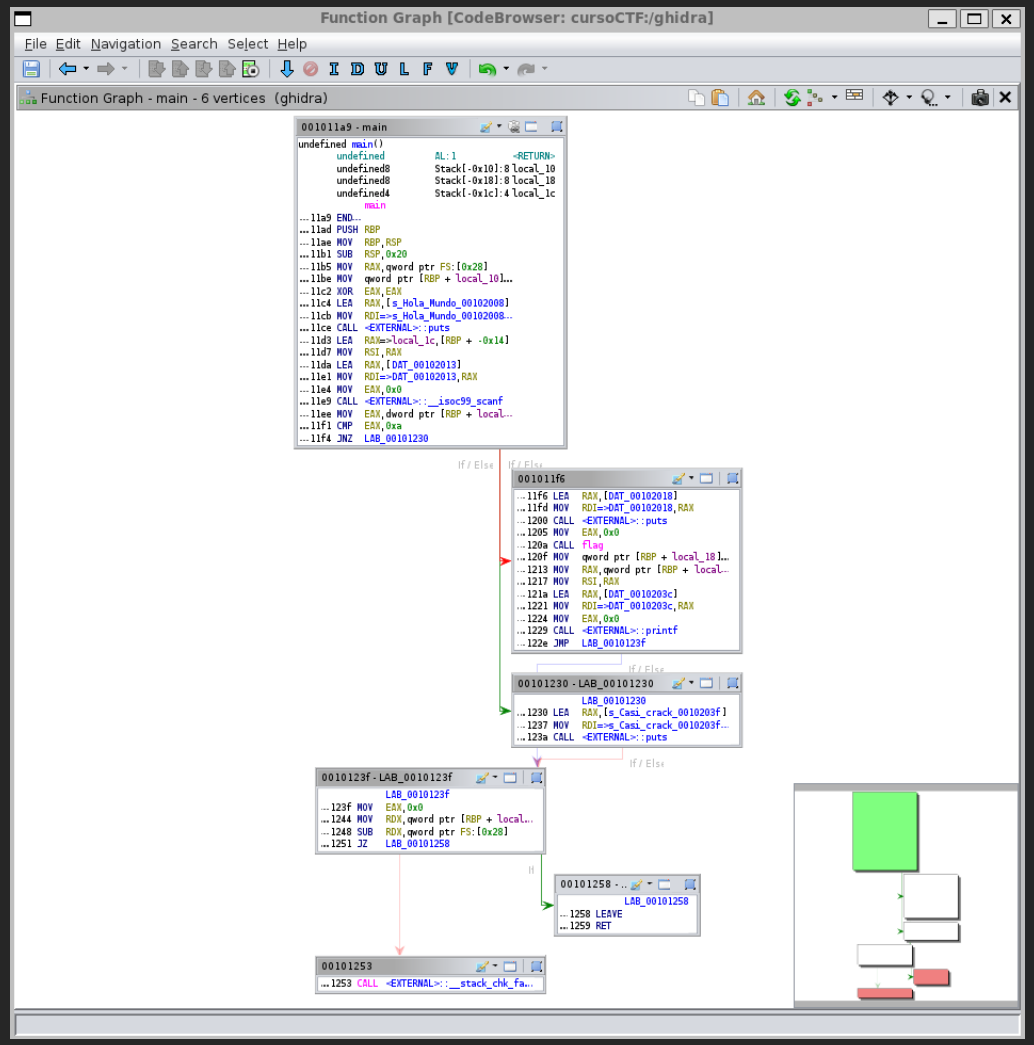


Desensamblado

```
                        main                                            XREF
001011a9 f3 0f 1e fa     ENDBR64
001011ad 55              PUSH        RBP
001011ae 48 89 e5        MOV         RBP,RSP
001011b1 48 83 ec 20     SUB         RSP,0x20
001011b5 64 48 8b        MOV         RAX,qword ptr FS:[0x28]
         04 25 28
         00 00 00
001011be 48 89 45 f8     MOV         qword ptr [RBP + local_10],RAX
001011c2 31 c0           XOR         EAX,EAX
001011c4 48 8d 05        LEA         RAX,[s_Hola_Mundo_00102008]
         3d 0e 00 00
001011cb 48 89 c7        MOV         RDI=>s_Hola_Mundo_00102008,RAX
001011ce e8 ad fe        CALL        <EXTERNAL>::puts
         ff ff
001011d3 48 8d 45 ec     LEA         RAX=>local_1c,[RBP + -0x14]
001011d7 48 89 c6        MOV         RSI,RAX
001011da 48 8d 05        LEA         RAX,[DAT_00102013]
         32 0e 00 00
001011e1 48 89 c7        MOV         RDI=>DAT_00102013,RAX
001011e4 b8 00 00        MOV         EAX,0x0
         00 00
001011e9 e8 c2 fe        CALL        <EXTERNAL>::__isoc99_scanf
         ff ff
001011ee 8b 45 ec        MOV         EAX,dword ptr [RBP + local_1c]
001011f1 83 f8 0a        CMP         EAX,0xa
001011f4 75 3a           JNZ         LAB_00101230
001011f6 48 8d 05        LEA         RAX,[DAT_00102018]
         1b 0e 00 00
001011fd 48 89 c7        MOV         RDI=>DAT_00102018,RAX
00101200 e8 7b fe        CALL        <EXTERNAL>::puts
         ff ff
00101205 b8 00 00        MOV         EAX,0x0
         00 00
0010120a e8 4b 00        CALL        flag
         00 00
0010120f 48 89 45 f0     MOV         qword ptr [RBP + local_18],RAX
00101213 48 8b 45 f0     MOV         RAX,qword ptr [RBP + local_18]
00101217 48 89 c6        MOV         RSI,RAX
0010121a 48 8d 05        LEA         RAX,[DAT_0010203c]
         1b 0e 00 00
00101221 48 89 c7        MOV         RDI=>DAT_0010203c,RAX
00101224 b8 00 00        MOV         EAX,0x0
         00 00
00101229 e8 72 fe        CALL        <EXTERNAL>::printf
         ff ff
0010122e eb 0f           JMP         LAB_0010123f
```

Para hacer ingeniería inversa al binario deberíamos leer el código en ensamblador y ver cómo se va ejecutando el programa.

CFG



En el código en assembly se producen diferente tipo de saltos.

El CFG “Control Flow Graph” nos permite ver de forma gráfica los posibles caminos de ejecución del binario.

Decompilado

Decompile: main - (ghidra)

```
1
2  undefined8 main(void)
3
4  {
5    long in_FS_OFFSET;
6    int local_1c;
7    undefined8 local_18;
8    long local_10;
9
10   local_10 = *(long *)(in_FS_OFFSET + 0x28);
11   puts("Hola Mundo");
12   __isoc99_scanf(&DAT_00102013,&local_1c);
13   if (local_1c == 10) {
14     puts(&DAT_00102018);
15     local_18 = flag();
16     printf("%s",local_18);
17   }
18   else {
19     puts("Casi crack");
20   }
21   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
22                     /* WARNING: Subroutine does not return */
23     __stack_chk_fail();
24   }
25   return 0;
26 }
27
```

Gracias a programas como Ghidra, no solo somos capaces de leer el desensamblado del binario, sino también su código decompilado.

Este código decompilado suele tener una sintaxis parecida a la de C (pseudocódigo).

Esto hace que realizar el análisis del binario sea más asequible.

Consejito

```
1
2  undefined8 main(void)
3
4  {
5    long in_FS_OFFSET;
6    int user_input;
7    undefined8 res_flag;
8    long local_10;
9
10   local_10 = *(long *)(in_FS_OFFSET + 0x28);
11   puts("Hola Mundo");
12   __isoc99_scanf(&DAT_00102013,&user_input);
13   if (user_input == 10) {
14     puts(&DAT_00102018);
15     res_flag = flag();
16     printf("%s",res_flag);
17   }
18   else {
19     puts("Casi crack");
20   }
21   if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
22                     /* WARNING: Subroutine does not return */
23     __stack_chk_fail();
24   }
25   return 0;
26 }
27
```

Renaming

En Ghidra, si tenemos seleccionada una variable y pulsamos la tecla “L” podremos renombrar una variable.

Esto es verdaderamente útil para no perderse en binarios de mayor complejidad

Depuradores (Windows)

Universidad
Rey Juan Carlos

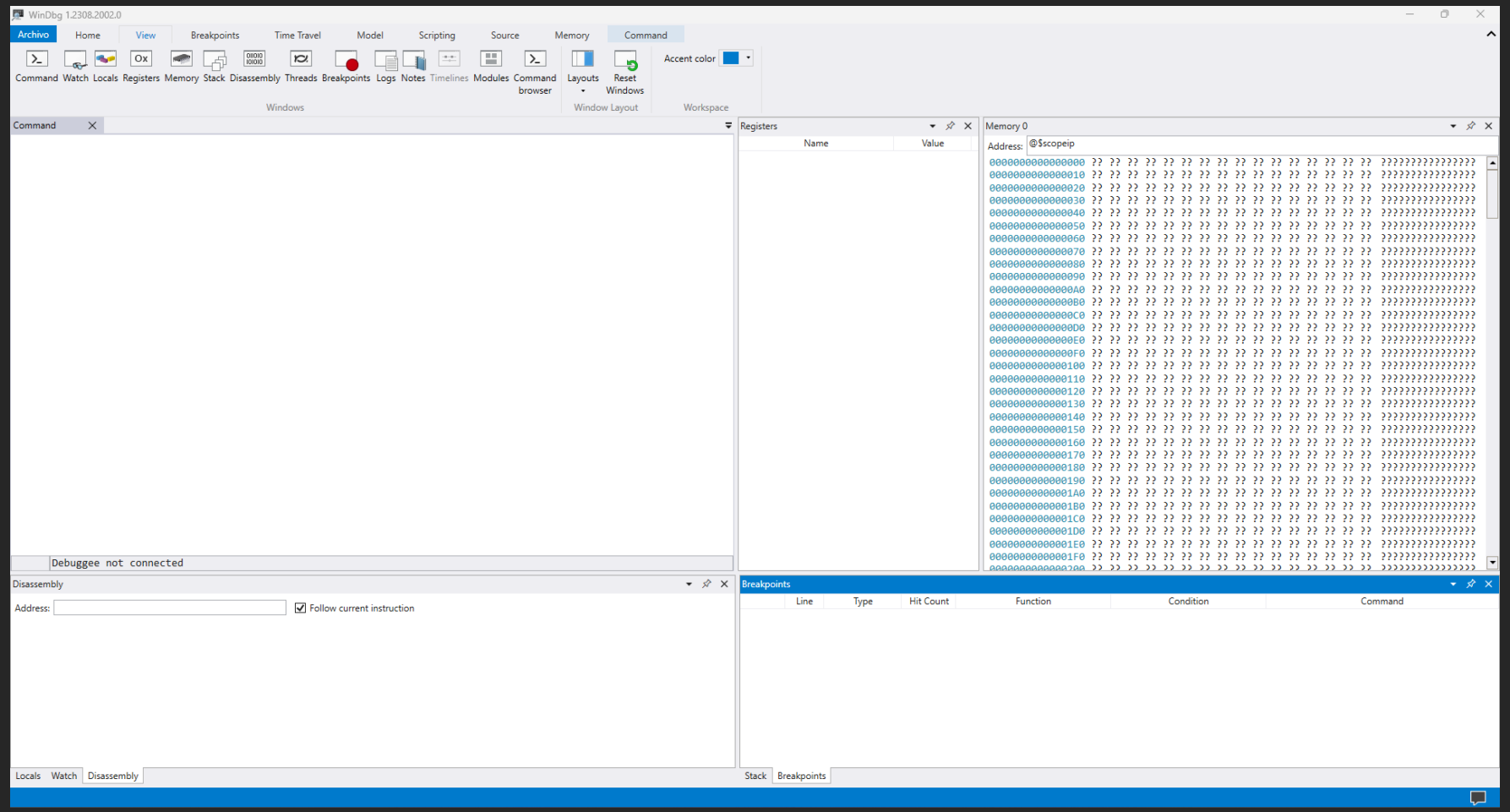
WinDBG

Cuando realizamos Ingeniería Inversa, los **depuradores** son utilizados para consultar el estado interno de un **proceso en tiempo de ejecución.**

Características principales:

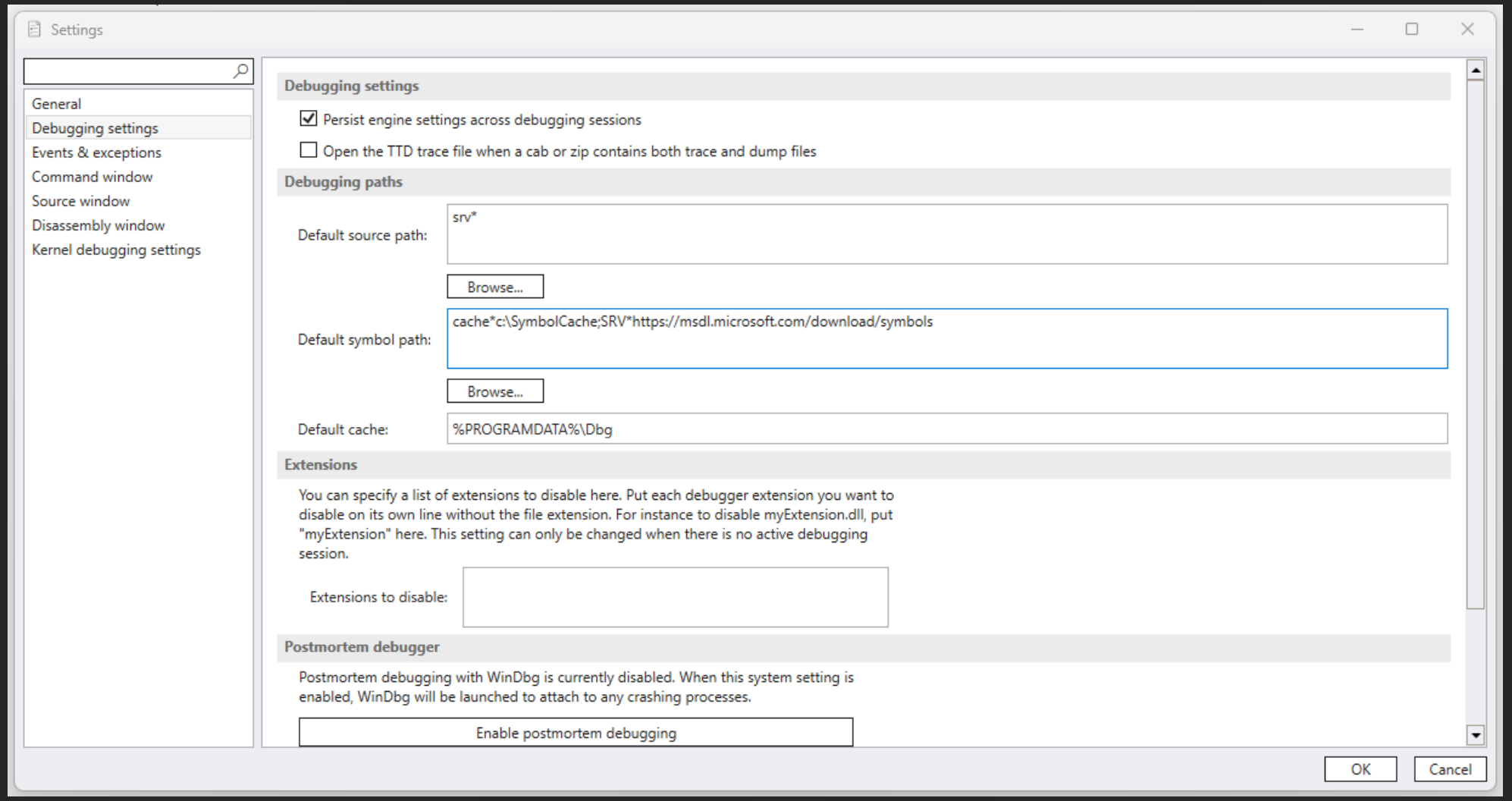
- Desensamblado
- Breakpoints
- Modificación dinámica
- Análisis de memoria

Configuración Inicial WinDBG




```
cache*c:\SymbolCache;SRV*https://msdl.microsoft.com/download/symbols
```

Configuración Inicial WinDBG

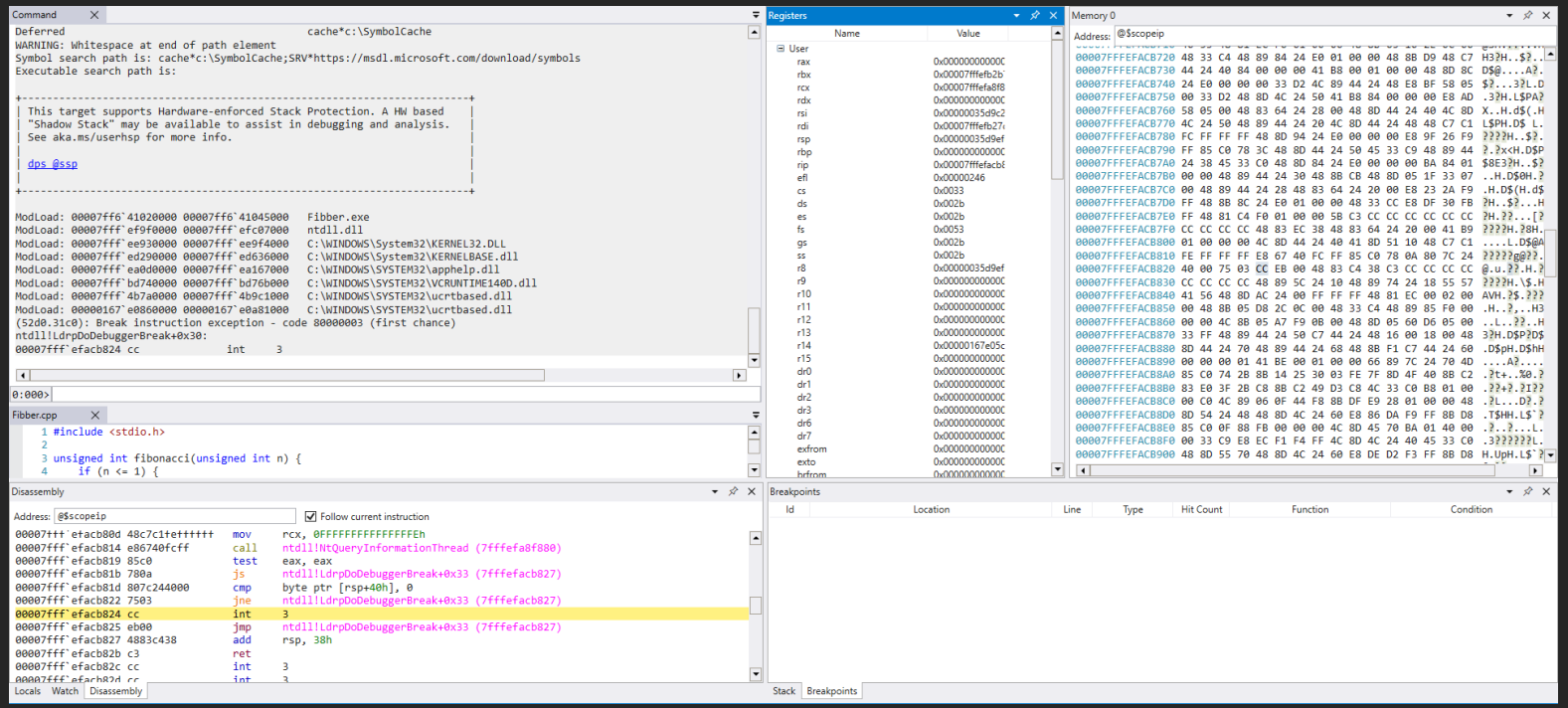



```
cache*c:\SymbolCache;SRV*https://msdl.microsoft.com/download/symbols
```

Carga y ejecución de un binario

Para cargar un binario en WinDBG deberemos:

- Archivo -> Launch executable
- Ctrl + e



Comandos

Para recargar los módulos de un binario podemos utilizar el comando **.reload**.

```
0:000> .reload
Reloading current modules
.......
*** WARNING: Unable to verify checksum for Fibber.exe
```

Para recargar la ejecución del binario, utilizaremos el comando **.restart**.

```
0:000> .restart
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
NatVis script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\Visu
JavaScript script unloaded from 'C:\Program Files\WindowsApps\Microsoft.WinDbg_1.2308.2002.0_x64__8wekyb3d8bbwe\amd64\

************* Preparing the environment for Debugger Extensions Gallery repositories **************
   ExtensionRepository : Implicit
   UseExperimentalFeatureForNugetShare : true
   AllowNugetExeUpdate : true
   AllowNugetMSCredentialProviderInstall : true
   AllowParallelInitializationOfLocalRepositories : true

   -- Configuring repositories
      ----> Repository : LocalInstalled, Enabled: true
      ----> Repository : UserExtensions, Enabled: true

>>>>>>>>>>>>> Preparing the environment for Debugger Extensions Gallery repositories completed, duration 0.000 seconds

************* Waiting for Debugger Extensions Gallery to Initialize **************
```

Comandos

Para desensamblar una parte del binario utilizaremos el comando **u <dirección_memoria>** (Unassemble):

```
0:000> u $exentry
Fibber!ILT+625(mainCRTStartup):
00007ff6`41031276 e975110000      jmp     Fibber!mainCRTStartup (00007ff6`410323f0)
Fibber!ILT+630(__scrt_is_managed_app):
00007ff6`4103127b e9602f0000      jmp     Fibber!__scrt_is_managed_app (00007ff6`410341e0)
Fibber!ILT+635(__castguard_set_user_handler):
00007ff6`41031280 e92b340000      jmp     Fibber!__castguard_set_user_handler (00007ff6`410346b0)
Fibber!ILT+640(__acrt_uninitialize_critical):
00007ff6`41031285 e9a6480000      jmp     Fibber!__scrt_stub_for_acrt_uninitialize_critical (00007ff6`41035b30)
Fibber!ILT+645(__security_init_cookie):
00007ff6`4103128a e9612a0000      jmp     Fibber!__security_init_cookie (00007ff6`41033cf0)
Fibber!ILT+650(__castguard_slow_path_check_os_handled):
00007ff6`4103128f e91c350000      jmp     Fibber!__castguard_slow_path_check_os_handled (00007ff6`410347b0)
Fibber!ILT+655(__acrt_initialize):
00007ff6`41031294 e957480000      jmp     Fibber!__scrt_stub_for_acrt_initialize (00007ff6`41035af0)
Fibber!ILT+660(WideCharToMultiByte):
00007ff6`41031299 e9c2470000      jmp     Fibber!WideCharToMultiByte (00007ff6`41035a60)
```

Comandos

Para desensamblar una función del binario utilizaremos el comando **uf <dirección_memoria>** (Unassemble Function):

```
0:000> uf 00007ff6`41031910
Fibber!main [C:\Users\pablo\source\repos\Fibber\Fibber\Fibber.cpp @ 12]:
   12 00007ff6`41031910 4055            push    rbp
   12 00007ff6`41031912 57              push    rdi
   12 00007ff6`41031913 4881ec28010000  sub     rsp,128h
   12 00007ff6`4103191a 488d6c2420      lea     rbp,[rsp+20h]
15732480 00007ff6`4103191f 488d0de2f60000  lea     rcx,[Fibber!_NULL_IMPORT_DESCRIPTOR <PERF> (Fibber+0x21008) (00007
15732480 00007ff6`41031926 e845faffff      call    Fibber!ILT+875(__CheckForDebuggerJustMyCode) (00007ff6`41031370)
   13 00007ff6`4103192b c745040a000000  mov     dword ptr [rbp+4],0Ah
   15 00007ff6`41031932 8b5504          mov     edx,dword ptr [rbp+4]
   15 00007ff6`41031935 488d0dec820000  lea     rcx,[Fibber!`string' (00007ff6`41039c28)]
   15 00007ff6`4103193c e859f8ffff      call    Fibber!ILT+405(printf) (00007ff6`4103119a)
   17 00007ff6`41031941 c7452400000000  mov     dword ptr [rbp+24h],0
   17 00007ff6`41031948 eb08            jmp     Fibber!main+0x42 (00007ff6`41031952)  Branch

Fibber!main+0x3a [C:\Users\pablo\source\repos\Fibber\Fibber\Fibber.cpp @ 17]:
   17 00007ff6`4103194a 8b4524          mov     eax,dword ptr [rbp+24h]
   17 00007ff6`4103194d ffc0            inc     eax
   17 00007ff6`4103194f 894524          mov     dword ptr [rbp+24h],eax

Fibber!main+0x42 [C:\Users\pablo\source\repos\Fibber\Fibber\Fibber.cpp @ 17]:
   17 00007ff6`41031952 8b4504          mov     eax,dword ptr [rbp+4]
```

Comandos

Para continuar con la ejecución del binario utilizaremos el comando **g** (Go):

```
ModLoad: 00007ff6`41020000 00007ff6`41045000   Fibber.exe
ModLoad: 00007fff`ef9f0000 00007fff`efc07000   ntdll.dll
ModLoad: 00007fff`ee930000 00007fff`ee9f4000   C:\WINDOWS\System32\KERNEL32.DLL
ModLoad: 00007fff`ed290000 00007fff`ed636000   C:\WINDOWS\System32\KERNELBASE.dll
ModLoad: 00007fff`ea0d0000 00007fff`ea167000   C:\WINDOWS\SYSTEM32\apphelp.dll
ModLoad: 00007fff`bd740000 00007fff`bd76b000   C:\WINDOWS\SYSTEM32\VCRUNTIME140D.dll
ModLoad: 00007fff`4b7a0000 00007fff`4b9c1000   C:\WINDOWS\SYSTEM32\ucrtbased.dll
ModLoad: 00000167`e0860000 00000167`e0a81000   C:\WINDOWS\SYSTEM32\ucrtbased.dll
(52d0.31c0): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007fff`efacb824 cc              int     3
0:000> g
ModLoad: 00007fff`ebe00000 00007fff`ebe18000   C:\WINDOWS\SYSTEM32\kernel.appcore.dll
ModLoad: 00007fff`edcd0000 00007fff`edd77000   C:\WINDOWS\System32\msvcrt.dll
ntdll!NtTerminateProcess+0x14:
00007fff`efa8f974 c3              ret
```

Breakpoints

Para interactuar con los breakpoints tenemos diferentes comandos:

- bp <dirección_memoria> “Breakpoint”
- bl “Breakpoint List”
- bc “Breakpoint Clear”

```
0:000> bp Fibber!main
0:000> bl
     0 e Disable Clear  00007ff6`41032330  [D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 77]
     1 e Disable Clear  00007ff6`41031910  [C:\Users\pablo\source\repos\Fibber\Fibber\Fibber.cpp @ 12]     0001 (0001)
0:000> bc 1
0:000> bl
     0 e Disable Clear  00007ff6`41032330  [D:\a\_work\1\s\src\vctools\crt\vcstartup\src\startup\exe_common.inl @ 77]
```

Encontrando el main

Para encontrar el main deberemos mirar el valor de $exentry e ir bajando en la pila de llamadas hasta que se salta al main().

u $exentry -> uf mainCRTStartup -> uf __scrt_common_main -> uf

__scrt_common_main_seh -> uf invoke_main -> uf (main)

Registros

Para la **lectura** de los registros utilizaremos el comando **r** seguido de los registros que queremos leer separados por una “,”

```
0:000> r rax, rsp, bl, r11d

rax=0000000000000002 rsp=000000000014fe08 bl=0 r11d=14fdb0
```

Para la **escritura** en los registros, utilizaremos el comando **r** seguido de los registros que queremos modificar y el valor a insertar en estos, separados por una “,”

```
0:000> r ax = 0xf00d, rbx = 0xdeadfacebeefd00d, bl = 0x0f

0:000> r ax, rbx, bl

ax=f00d rbx=deadfacebeefd00f bl=f
```

Memoria

Para la **lectura** de memoria utilizaremos el comando **d* <dirección_memoria> L<nº_bytes>**

En función de este "*" que sigue a la d, se nos mostrará una cantidad de información u otra:

- db "Display Bytes"
- dd "Display Double"
- dq "Display Quad"
- da "Display Ascii"

```
0:000> db rsp L6
00000000`0014fe08  ef be ad de 01 00
0:000> dd rsp L6
00000000`0014fe08  deadbeef 11100001 00000001 0000693d
00000000`0014fe18  4b054299 00007fff
0:000> dq rsp L6
00000000`0014fe08  11100001`deadbeef 0000693d`00000001
00000000`0014fe18  00007fff`4b054299 00000000`00000000
00000000`0014fe28  00000001`40001a4d 00000000`00000001
0:000> da 00000001`40002220
00000001`40002220  "First %d elements of the Fibbona"
00000001`40002240  "cci sequence: "
```

Attaching en UserSpace

Muchas veces no querremos abrir el proceso desde el debugger, sino conectarnos a un proceso que se encuentra corriendo.

Para ello deberemos irnos a Archivo -> “Attach to Process” o pulsar la tecla F6.

Ejecución Simbólica

Universidad
Rey Juan Carlos

¿Qué es la ejecución simbólica?

```
# A simple guessing game.
user_input = raw_input('Enter the password: ')
if user_input == 'hunter2':
  print 'Success.'
else:
  print 'Try again.'
```

Encontraremos muchas soluciones:

- Usar strings para encontrar “hunter2”
- Utilizar ltrace para encontrar la comparación.
- Utilizar windbg para inspeccionar la memoria y leer donde se encuentra guardada la string “hunter2”
- …

¿Qué es la ejecución simbólica?

```
# A complex guessing game. Don't bother to figure out what the code does.
def encrypt(string, amount):
  for i in range(0, len(string)):
    string[i] += amount

user_input = raw_input('Enter the password: ')
if encrypt(user_input, amount=1) == encrypt('hunter2', amount=2):
  print 'Success.'
else:
  print 'Try again.'
```

Ya no encontramos tantas soluciones:

- ~~Usar strings para encontrar "hunter2"~~
- ~~Utilizar ltrace para encontrar la comparación.~~
- ~~Utilizar windbg para inspeccionar la memoria y leer donde se encuentra guardada la string "hunter2"~~
- Reversearse la función "encrypt". En este caso es simple, pero se *puede complicar en otros casos*

Símbolos

$$x^2 + 2x + 3 = 4$$

Pensemos que un símbolo es como x, pero que es una variable del programa

El valor de x dependerá de la ecuación que lo restringe

El valor de un símbolo dependerá del camino de ejecución

Camino de ejecución

```
user_input = raw_input('Enter the password: ')
if user_input == 'hunter2':
  print 'Success.'
else:
  print 'Try again.'
```

En este caso Podemos observar que cláramente encontramos 2 posibles caminos de ejecución

Path 1: if user_input equals 'hunter2'

```
user_input = raw_input('Enter the password: ')
```

```
if user_input == 'hunter2'   # it is equal!
```

```
print 'Success.'
```

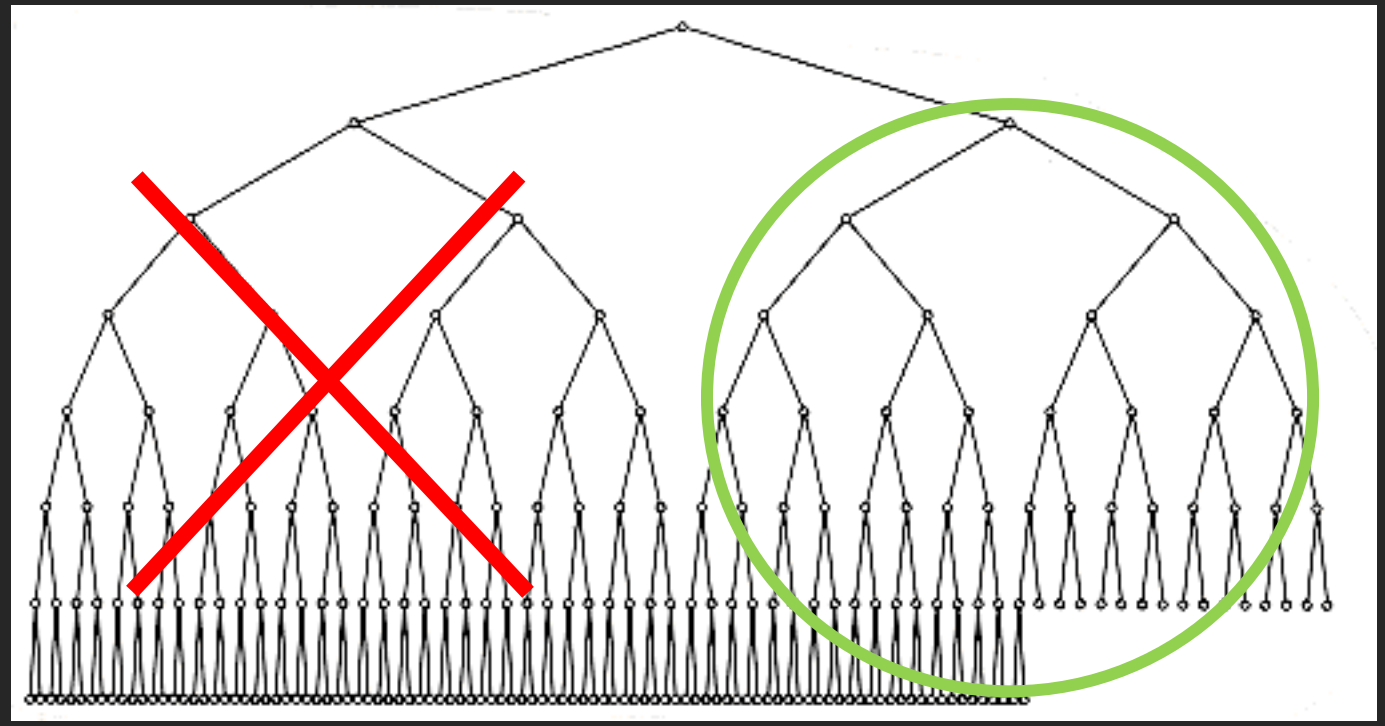
Path 2: if user_input not equals 'hunter2'

```
user_input = raw_input('Enter the password: ')
```

```
if user_input == 'hunter2'   # it is not equal
```

```
print Try again.'
```

Camino de ejecución



Resolviendo para λ

```
user_input = λ
if user_input == 'hunter2':
  print 'Success.'
else:
  print 'Try again.'
```

Se asigna un símbolo a nuestra variable, y se define el camino de ejecución que queremos alcanzar.

Una vez se ha seleccionado este camino, resolvemos la ecuación para nuestro símbolo.

Angr

Angr es un motor de ejecución simbólica que permite:

- Seguir cualquier rama del binario
- Buscar el estado deseado de un programa definiendo unos criterios
- Resolución de variable simbólicas dadas unas restricciones

Angr en sí es una librería de python3 muy extensa con multitude de funciones que nos permiten personalizar la exploración de los estados del binario.

Angr-management es su versión con UI.

Angr

Todos los scripts de angr comparten características comunes:

- Inicialización del Proyecto
- Definición de restricciones
- Definición de la exploración

Para encontrar la dirección de memoria definida en el script, debemos tener en cuenta el uso de Ghidra.

0x400000 + offset

```python
import angr
import sys

def main(argv):
  path_to_binary = argv[1]  # :string
  project = angr.Project(path_to_binary)
  initial_state = project.factory.entry_state(
    add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY,
                    angr.options.SYMBOL_FILL_UNCONSTRAINED_REGISTERS}
  )

  simulation = project.factory.simgr(initial_state)

  print_good_address = 0x804868c
  simulation.explore(find=print_good_address)


  if simulation.found:
    solution_state = simulation.found[0]
    print(solution_state.posix.dumps(sys.stdin.fileno()).decode())

  else:

    raise Exception('Could not find the solution')

if __name__ == '__main__':
  main(sys.argv)
```

Angr

Todos los scripts de angr comparten características comunes:

- Inicialización del Proyecto
- Definición de restricciones
- Definición de la exploración

```
1  import angr
2  import sys
3
4  def main(argv):
5    path_to_binary = argv[1]  # :string
6    project = angr.Project(path_to_binary)
7    initial_state = project.factory.entry_state(
8      add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY,
9                      angr.options.SYMBOL_FILL_UNCONSTRAINED_REGISTERS}
10   )
11
12   simulation = project.factory.simgr(initial_state)
13
14   print_good_address = 0x804868c
15   simulation.explore(find=print_good_address)
16
17
18   if simulation.found:
19     solution_state = simulation.found[0]
20     print(solution_state.posix.dumps(sys.stdin.fileno()).decode())
21
22   else:
23
24     raise Exception('Could not find the solution')
25
26 if __name__ == '__main__':
27   main(sys.argv)
28
```

Para encontrar la dirección de memoria definida en el script, debemos tener en cuenta el uso de Ghidra.

0x400000 + offset

```
001011cf 48 8d 05 31 0e 00 00   LEA   RAX,[s_Success!!_00102007]
001011d6 48 89 c7               MOV   RDI=>s_Success!!_00102007,RAX
001011d9 e8 92 fe ff ff         CALL  <EXTERNAL>::puts
```

```
print_good_address = 0x400000 + 0x000011cf
```

Angr

Todos los scripts de angr comparten características comunes:

- Inicialización del Proyecto
- Definición de restricciones
- Definición de la exploración

Path al binario, inicalización del proyecto y el estado inicial

```
def main(argv):
  path_to_binary = argv[1]
  project = angr.Project(path_to_binary)

  initial_state = project.factory.entry_state(
    add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY,
                    angr.options.SYMBOL_FILL_UNCONSTRAINED_REGISTERS}
  )
```

Definición de la simulación

```
simulation = project.factory.simgr(initial_state)
```

Restricciones y exploración

```
def is_successful(state):

  puts_address = 0x8048370
  if state.addr == puts_address:

    return check_puts(state)
  else:

    return False

simulation.explore(find=is_successful)
```

Angr

Todos los scripts de angr comparten características comunes:

- Inicialización del Proyecto
- Definición de restricciones
- Definición de la exploración

Path al binario, inicalización del proyecto y el estado inicial

```
def main(argv):
  path_to_binary = argv[1]
  project = angr.Project(path_to_binary)

  initial_state = project.factory.entry_state(
    add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY,
                    angr.options.SYMBOL_FILL_UNCONSTRAINED_REGISTERS}
  )
```

Definición de la simulación

```
simulation = project.factory.simgr(initial_state)
```

Restricciones y exploración

```
def is_successful(state):

  puts_address = 0x8048370
  if state.addr == puts_address:

    return check_puts(state)
  else:

    return False

simulation.explore(find=is_successful)
```

Big S/O


Guardia Civil
GUARDIA CIVIL
175 AÑOS A TU LADO
LIGA DE RETOS
EN EL CIBERESPACIO

Bibliografía

Todos los scripts de angr comparten características comunes:

- Inicialización del Proyecto
- Definición de restricciones
- Definición de la exploración

Path al binario, inicalización del proyecto y el estado inicial

```
def main(argv):
  path_to_binary = argv[1]
  project = angr.Project(path_to_binary)

  initial_state = project.factory.entry_state(
    add_options = { angr.options.SYMBOL_FILL_UNCONSTRAINED_MEMORY,
                    angr.options.SYMBOL_FILL_UNCONSTRAINED_REGISTERS}
  )
```

Definición de la simulación

```
simulation = project.factory.simgr(initial_state)
```

Restricciones y exploración

```
def is_successful(state):

  puts_address = 0x8048370
  if state.addr == puts_address:

    return check_puts(state)
  else:

    return False

simulation.explore(find=is_successful)
```

Módulo IV: Ingeniería Inversa & Exploiting

Pablo Pastor, Diego Palacios & David Billdhart

Universidad
Rey Juan Carlos