Reverse Engineering I

CC5325 - Taller de Hacking Competitivo
Diego Vargas

Contenidos

- Intro a Reversing
- Desensamblación y Decompilación
- Herramientas
- Demo

Intro a Reversing

¿Qué es reversing?

Proceso de analizar *algo* para entender cómo está hecho y su funcionamiento interno. Estudiaremos su aplicación en el área de computación, en donde ese *algo* es un programa o un ejecutable.

El trabajo de reversing es muy involucrado y se requiere amplio conocimiento del ambiente y de la plataforma en la que se ejecuta el objeto analizado.

Para este curso:
Los problemas no requerirán conocer profundamente el lenguaje, ni aplicación que se está reversando.

¿Para qué sirve?

El proceso de reversing sirve para entender cómo funciona el programa o ejecutable que se está analizando, cuando no se tiene acceso al código fuente.

Ejemplos:

- Aplicaciones móviles
- Aplicaciones web
- Ejecutables binarios
- Cualquier tipo de código compilado

Desensamblación y Decompilación

¿Cuál es la diferencia?

Desensamblación

Convierte el lenguaje de máquina en código Assembler. La transformación es directa, pues cada instrucción de Assembler está asociada a una instrucción en lenguaje de máquina.

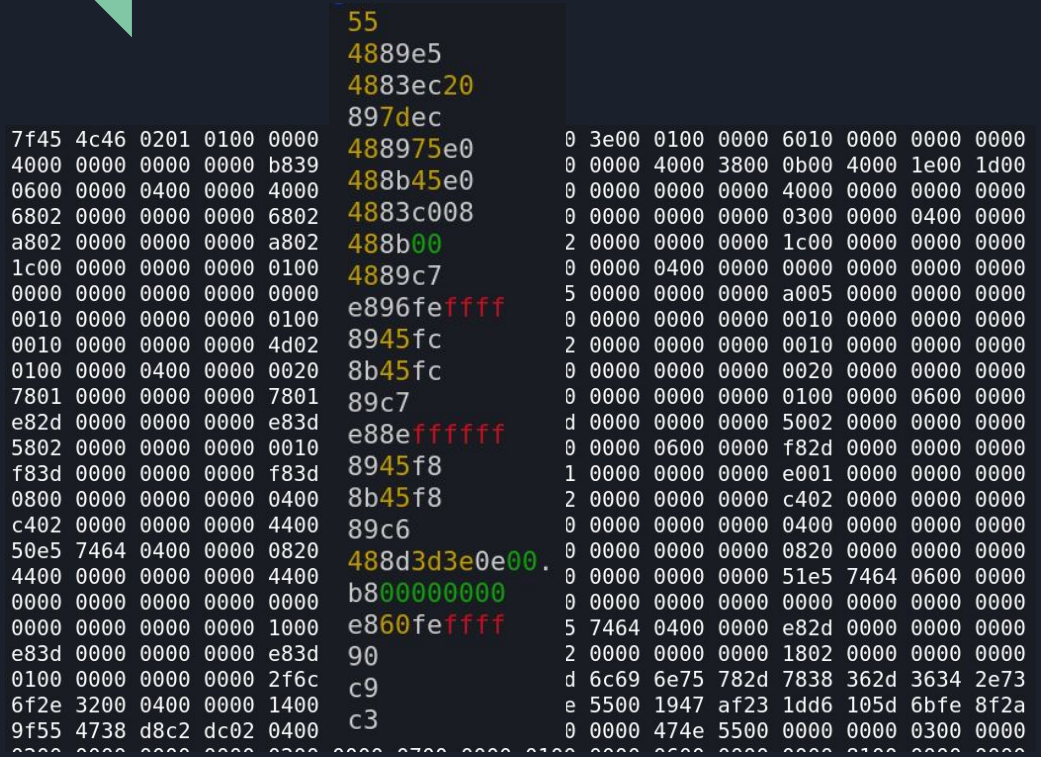
Se suele utilizar con lenguajes de bajo nivel: C/C++, Assembler, Fortran, Go.

Decompilación

Proceso inverso a la compilación: intenta obtener algo similar al código fuente. Requiere el uso de heurísticas que pretenden *“adivinar”* la estructura del código original.

Se suele utilizar con lenguajes de alto nivel: Python, PHP, Java, .NET.

Desensamblación: Código de máquina




```
push rbp
mov rbp, rsp
sub rsp, 0x20
mov dword [var_14h], edi
mov qword [str], rsi
mov rax, qword [str]
add rax, 8
mov rax, qword [rax]
mov rdi, rax
call sym.imp.atoi
mov dword [var_4h], eax
mov eax, dword [var_4h]
mov edi, eax
call sym.fibonacci
mov dword [var_8h], eax
mov eax, dword [var_8h]
mov esi, eax
lea rdi, [0x00002004]
mov eax, 0
call sym.imp.printf
nop
leave
ret
```

Desensamblación: Assembler

Es un lenguaje de muy bajo nivel, que se transforma directamente en lenguaje de máquina. Varía dependiendo de la arquitectura de la CPU. Las más comunes son ARM y x86, pero todas tienen elementos en común.

Cada instrucción se compone de 1 operador y 0 o más argumentos:

[op] [arg1 arg2 ...]

Los operadores pueden: copiar registros de memoria, realizar operaciones aritméticas, preguntar por ciertas condiciones, saltar a una dirección de memoria, insertar y quitar datos de la pila, llamar a subrutinas, etc.

Los argumentos pueden ser del tipo: constantes, direcciones de memoria, registros de la CPU, labels, etc.

Desensamblación: Ejemplo

```
int ex(int n) {
        while (n < 100) {
                n = n * 2;
        }
        return n;
}

int main() {
        ex(1);
        return 0;
}
```

Desensamblación: Proceso

Se utiliza herramientas para desensamblar el código. Las herramientas usualmente son del tipo dinámicas, por lo que ayudan ejecutando el código instrucción a instrucción.

La dificultad suele estar en entender la estructura básica del ejecutable. Con este conocimiento ya es más fácil buscar lo que se necesita para resolver el problema.

Decompilación: Bytecode

El bytecode es una especie de lenguaje intermedio entre el código fuente y el código de máquina. Se utiliza para optimizar el funcionamiento del código, sin tener que compilarlo completamente, lo cual es dependiente de la plataforma.

Distintos lenguajes tienen distintos tipos de bytecode, cada uno de los cuales puede ser o no legible para los humanos. Sin embargo, todos tienen en común que al decompilarlos se obtiene algo muy parecido al código original, pues se pierde mucha menos información que al compilar y ensamblar a código de máquina.

Decompilación: Ejemplo

```
00000000: 03f3 0d0a d0da a760 6300 0000 0000 0000  .......`c.......
00000010: 0003 0000 0040 0000 0073 4100 0000 6400  .....@...sA...d.
00000020: 0064 0100 6c00 005a 0000 6402 0084 0000  .d..l..Z..d.....
00000030: 5a01 0065 0200 6500 006a 0300 6403 0019  Z..e..e..j..d...
00000040: 8301 005a 0400 6501 0065 0400 8301 005a  ...Z..e..e.....Z
00000050: 0500 6404 0047 6505 0047 4864 0100 5328  ..d..Ge..GHd..S(
00000060: 0500 0000 69ff ffff ff4e 6301 0000 0004  ....i....Nc.....
00000070: 0000 0004 0000 0043 0000 0073 6000 0000  .......C...s`...
00000080: 6401 007d 0100 6402 007d 0200 784d 0074  d..}..d..}..xM.t
00000090: 0000 7c00 0083 0100 445d 3f00 7d03 007c  ..|.....D]?.}..|
000000a0: 0300 6403 0016 6404 006b 0200 7240 007c  ..d...d..k..r@.|
000000b0: 0100 6405 007c 0200 1537 7d01 006e 0e00  ..d..|...7}..n..
000000c0: 7c01 0064 0500 7c02 0015 387d 0100 7c02  |..d..|...8}..|.
000000d0: 0064 0300 377d 0200 7119 0057 7c01 0053  .d..7}..q..W|..S
000000e0: 2806 0000 004e 6700 0000 0000 0000 0067  (....Ng........g
000000f0: 0000 0000 0000 f03f 6902 0000 0069 0000  .......?i....i..
00000100: 0000 6904 0000 0028 0100 0000 7405 0000  ..i....(....t...
00000110: 0072 616e 6765 2804 0000 0074 0100 0000  .range(....t....
00000120: 6e74 0100 0000 7374 0100 0000 6b74 0100  nt....st....kt..
00000130: 0000 6928 0000 0000 2800 0000 0073 0500  ..i(....(....s..
00000140: 0000 7069 2e70 7974 0700 0000 4c65 6962  ..pi.pyt....Leib
00000150: 6e69 7a03 0000 0073 1000 0000 0001 0601  niz....s........
00000160: 0601 1301 1001 1102 0e01 0e01 6901 0000  ............i...
00000170: 0073 0500 0000 5049 203d 2028 0600 0000  .s....PI = (....
00000180: 7403 0000 0073 7973 5205 0000 0074 0300  t....sysR....t..
00000190: 0000 696e 7474 0400 0000 6172 6776 5201  ..intt....argvR.
000001a0: 0000 0074 0200 0000 7069 2800 0000 0028  ...t....pi(....(
000001b0: 0000 0000 2800 0000 0073 0500 0000 7069  ....(....s....pi
000001c0: 2e70 7974 0800 0000 3c6d 6f64 756c 653e  .pyt....<module>
000001d0: 0100 0000 7308 0000 000c 0209 0b13 010c  ....s...........
000001e0: 01                                       .
```

```python
import sys

def Leibniz(n):
    s = 0.0
    k = 1.0
    for i in range(n):
        if i % 2 == 0:
            s += 4 / k
        else:
            s -= 4 / k
        k += 2
    return s

n = int(sys.argv[1])
pi = Leibniz(n)
print 'PI = ', pi
```

Decompilación: Proceso

Al igual que en la desensamblación, para decompilar se utiliza herramientas. Sin embargo estas herramientas toman el bytecode y lo transforman en algo similar al código original.

Lo importante luego es leer el código y entender la lógica general, la cual puede ser bastante compleja.

Herramientas

Desensamblación

- Radare2
- GDB
- Ghidra
- IDA
- OllyDbg
- Binary Ninja

Decompilación

- JD-GUI
- Decompyle3
- Uncompyle6
- dnspy
- UnPHP

Demo

Demo

Herramientas

- Radare2
- Uncompyle6

Challenges

- Impossible Password de HTB
- Ejemplo en Python