

La máquina dentro de la máquina

Al ejecutar el binario:

```
L$ ./Ultra-Pensador-VM-hard
Ultra-Pensador-VM v2
Nada interesante por aqui ...
```

El programa no solicita entrada ni imprime información sensible, lo que indica que la lógica real del reto está oculta y requiere análisis estático.

Al usar strings, no aparece ninguna flag ni texto sospechoso.

Esto sugiere que la flag no está en texto plano.

El análisis de la función main revela que el programa únicamente imprime dos mensajes y finaliza su ejecución. No existe ninguna llamada directa a funciones que procesen los datos observados en .rodata.

Sin embargo, se identifica una instrucción lea que carga la dirección 0x1149 en una variable local, sin que dicha dirección sea ejecutada mediante una instrucción call. Esto indica la posible existencia de código relevante que nunca es invocado durante la ejecución normal del programa.

```
[0x00001060]> pdf@main
      ; DATA XREF from entry0 @ 0x1074(r)
56: int main (int argc, char **argv, char **envp);
afv: vars(1:sp[0x10 .. 0x10])
    0x00001236      55          push rbp
    0x00001237      4889e5      mov rbp, rsp
    0x0000123a      4883ec10   sub rsp, 0x10
    0x0000123e      488d05540e .. lea rax, str.Ultra_Pensador_VM_v2 ; 0x2099 ; "Ultra-Pensador-VM v2"
    0x00001245      4889c7      mov rdi, rax           ; const char *s
    0x00001248      e8f3fdffff  call sym.imp.puts     ; int puts(const char *s)
    0x0000124d      488d055a0e .. lea rax, str.Nada_interesante_por_aqui... ; 0x20ae ; "Nada interesante p
or aqui ..."
    0x00001254      4889c7      mov rdi, rax           ; const char *s
    0x00001257      e8e4fdffff  call sym.imp.puts     ; int puts(const char *s)
    0x0000125c      488d05e6fe .. lea rax, [0x00001149]
    0x00001263      488945f8      mov qword [var_8h], rax
    0x00001267      b800000000  mov eax, 0
    0x0000126c      c9          leave
    0x0000126d      c3          ret
```

Al analizar el código ubicado en la dirección 0x1149, se observa una función completa que no es invocada desde main. Esta función accede directamente al bloque de datos ubicado en la sección .rodata, descifra cada byte aplicando una operación XOR con 0x5A y ejecuta un bucle de interpretación.

Las comparaciones contra los valores 0x01, 0x05 y 0xFF indican la implementación de instrucciones personalizadas, confirmando que se trata de una máquina virtual encargada de ejecutar el bytecode previamente identificado.

```
[0x00001060]> s 0x1149
[0x00001149]> pdf
ERROR: Cannot find function at 0x00001149
[0x00001149]> pd 20
    ; DATA XREF from main @ 0x125c(r)
0x00001149      55          push rbp
0x0000114a      489e5       mov rbp, rsp
0x0000114d      4883ec20   sub rsp, 0x20
0x00001151      c745e00000 .. mov dword [rbp - 0x20], 0
0x00001158      48c745e800 .. mov qword [rbp - 0x18], 0
    ; CODE XREF from entry.init0 @ +0xeb(x)
0x00001160      488b45e8   mov rax, qword [rbp - 0x18]
0x00001164      488d5001   lea rdx, [rax + 1]
0x00001168      488955e8   mov qword [rbp - 0x18], rdx
0x0000116c      488d15ad0e .. lea rdx, [0x00002020] ; "[Z\x1c_Z[Z\x13_Z[Z\Z_t_Z[Z!_Z[Z\Z\f_Z[Z\x17_Z
[Z\x05_Z[Z(Z Zi_Z[Z, Z[Z\x05_Z[Z2_Z[Zn_Z[Z\Z[Z>_Z[Z\x05_Z[Z8_Z[Z/_Z[Z.Z\x05_Z[Z<_Z[Z/Z\Z[Z\x05Ultra-Pen
sador-VM v2"
0x00001173      0fb60410  movzx eax, byte [rax + rdx]
0x00001177      83f05a    xor eax, 0x5a
0x0000117a      8845ff    mov byte [rbp - 1], al
0x0000117d      0fb645ff  movzx eax, byte [rbp - 1]
0x00001181      3dff000000 cmp eax, 0xff
0x00001186      0f84a400000 je 0x1230
0x0000118c      3dff000000 cmp eax, 0xff
0x00001191      0f8f9c00000 jg 0x1233
0x00001197      83f801    cmp eax, 1
0x0000119a      740a      je 0x11a6
0x0000119c      83f805    cmp eax, 5
[0x00001149]> █
```

Se observa la sección:

```
.rodata vaddr=0x00002000 size=0xcb
```

Nos posicionamos en dicha sección:

```
[0x00002000]> s 0x2000
[0x00002000]> s
0x2000
[0x00002000]> px
- offset -  0 1 2 3 4 5 6 7 8 9  A B  C D  E F  0123456789ABCDEF
0x00002000  0100 0200 0000 0000 0000 0000 0000 0000  ..... .
0x00002010  0000 0000 0000 0000 0000 0000 0000 0000  ..... .
0x00002020  5b5a 1c5f 5a5b 5a13 5f5a 5b5a 095f 5a5b  [Z._Z[Z._Z[Z._Z[
0x00002030  5a21 5f5a 5b5a 0c5f 5a5b 5a17 5f5a 5b5a  Z!_Z[Z._Z[Z._Z[Z
0x00002040  055f 5a5b 5a28 5f5a 5b5a 695f 5a5b 5a2c  ._Z[Z(_Zi_Z[Z,
0x00002050  5f5a 5b5a 055f 5a5b 5a32 5f5a 5b5a 6e5f  _Z[Z._Z[Z2_Z[Zn_
0x00002060  5a5b 5a28 5f5a 5b5a 3e5f 5a5b 5a05 5f5a  Z[Z(_Z[Z>_Z[Z._Z
0x00002070  5b5a 385f 5a5b 5a2f 5f5a 5b5a 2e5f 5a5b  [Z8_Z[Z/_Z[Z._Z[
0x00002080  5a05 5f5a 5b5a 3c5f 5a5b 5a2f 5f5a 5b5a  Z._Z[Z<_Z[Z/_Z[Z
0x00002090  345f 5a5b 5a27 5f5a a555 6c74 7261 2d50  4_Z[Z'.Z.Ultra-P
0x000020a0  656e 7361 646f 722d 564d 2076 3200 4e61  ensador-VM v2.Na
0x000020b0  6461 2069 6e74 6572 6573 616e 7465 2070  da interesante p
0x000020c0  6f72 2061 7175 692e 2e2e 0000 011b 033b  or aqui ... ... ;
0x000020d0  3000 0000 0500 0000 54ef ffff 7c00 0000  0 ... . . T. . | ...
0x000020e0  84ef ffff a400 0000 94ef ffff 4c00 0000  .. . . . . L ...
0x000020f0  7df0 ffff bc00 0000 6af1 ffff dc00 0000  }. . . . j. . . .
[0x00002000]> s 0x2020
```

Una vez identificado el bloque de datos no ASCII dentro de la sección .rodata, se procede a extraer el bytecode cifrado para su posterior análisis y ejecución fuera del binario.

A partir del análisis previo se determinó que el bytecode comienza en la dirección virtual 0x2020. Utilizando radare2, se extrae dicho bloque directamente a un archivo binario:

```
s 0x2020
```

```
wtf enc.bin 0x79
```

El bytecode extraído se encuentra ofuscado mediante una operación XOR simple. Cada byte del programa virtual es descifrado aplicando la siguiente operación:

```
byte_descifrado = byte_cifrado ^ 0x5A
```

Para obtener la flag, se implementó un pequeño intérprete en Python que replica el comportamiento de la máquina virtual identificada en el binario.

```
KEY = 0x5A

def run_vm(enc):
    regs = [0, 0, 0, 0]
    pc = 0
    out = []

    while pc < len(enc):
        op = enc[pc] ^ KEY
        pc += 1

        if op == 0x01: # LOADI
            if pc + 1 >= len(enc):
                break
            r = (enc[pc] ^ KEY) & 0x03
            pc += 1
            imm = enc[pc] ^ KEY
            pc += 1
            regs[r] = imm

        elif op == 0x05: # PRINT
            if pc >= len(enc):
                break
            r = (enc[pc] ^ KEY) & 0x03
            pc += 1
            out.append(chr(regs[r]))
```

```
elif op == 0xFF: # HALT
break
else:
break

return "".join(out)

if __name__ == "__main__":
with open("enc.bin", "rb") as f:
enc = f.read()

print("Flag:", run_vm(enc))
```

```
└$ python3 solucion.py
Flag: FIS{VM_r3v_h4rd_but_fun}
```