



BLIND ROP

Эксплуатация уязвимостей с
помощью техники Blind ROP

sp0ctf

@B1N4R9



INTRO

*Неожиданный сюрприз,
где можем встретить*



ИHTPO

Challenge

0 Solves



PWN
300

Show me what you gou!

```
nc 10.10.10.10 31337
```

Flag

Submit

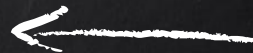
Directed by
ROBERT B. WEIDE



Области применения

Проприетарный проект

Нет бинаря, ноль информации



Библиотека с открытым кодом

Бинаря нет, опенсорс библиотека
используется в проекте

Проект с открытым кодом

Бинаря нет, есть сорцы





Выход?

Пывним
вслепую!





ONCE AGAIN

*Stack overflow, Shellcode,
NX, ROP, ASLR, Canary, PLT*



Stack overflow

SOME DATA REGION:
0x4141414141414141:
???????

Segmentation fault

RIP=0x4141414141414141

1

2

3

7777

4

AAAAAAAA

5

AAAAAAAA

6

AAAAAAAA

7

AAAAAAAA

8

old ebp|AAAAAAAA

9

ret from main|AAAAAAAA

.stack ; segment with stack



Stack overflow

pwner@PWN:~# *cat flag.txt*

spbctf{r3d_kks_far_c4d3ts}

RIP=0x7fffffff890 shellcode - execve("/bin/bash",0,0)	
1	
2	
3	7777
4	AAAAAAAA
5	AAAAAAAA
6	AAAAAAAA
7	AAAAAAAA
8	old ebp AAAAAAAA
9	ret from main shellcode addr
.stack ; segment with stack	

NX bit



RIP= 0x7fffffff890
shellcode - execve("/bin/bash",0,0)

1

2

3

7777

4

AAAAAAAA

5

AAAAAAAA

6

AAAAAAAA

7

AAAAAAAA

8

old ebp|AAAAAAAA

9

ret from main| shellcode addr

.stack ; segment with stack

SOME DATA REGION:
0x7fffffff890 : ???????

Segmentation fault



Return Oriented Programming

`mov esi, 0x1337; push esi; ret`

`xor eax, eax; ret`

`pop rdi; ret`

`inc eax; ret`

`syscall`

`ret`





ROP

RIP= 0x7fffffff890

ROP - `execve("/bin/bash",0,0)`

1	AAAAAAAA
2	old ebp AAAAAAAA
3	ret from main 0x40072e (g1)
4	0x40189b ("/bin/sh")
5	0x400582 (g2)
6	0x3b
7	0x400601 (g3)

→ pop rdi; *ret*

→ pop rax; *ret*

→ *syscall*

.stack ; segment with stack



Canary

*** *stack smashing detected* ***:

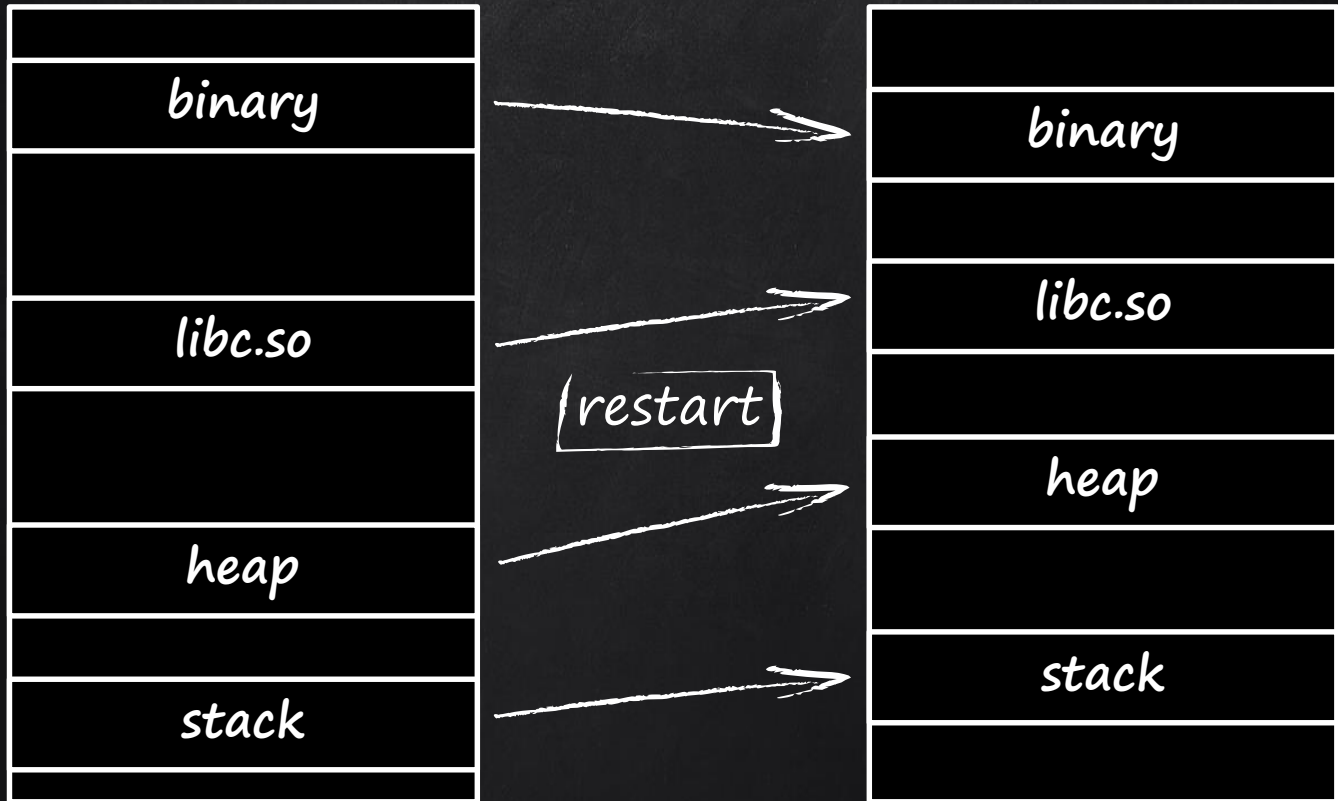
terminated



RIP=0x7f98b3bea5e0 (stask_chk_fail)	
1	
2	
3	7777
4	AAAAAAAA
5	AAAAAAAA
6	AAAAAAAA
7	canary AAAAAAAA
8	old ebp AAAAAAAA
9	ret from main shellcode addr
.stack ; segment with stack	

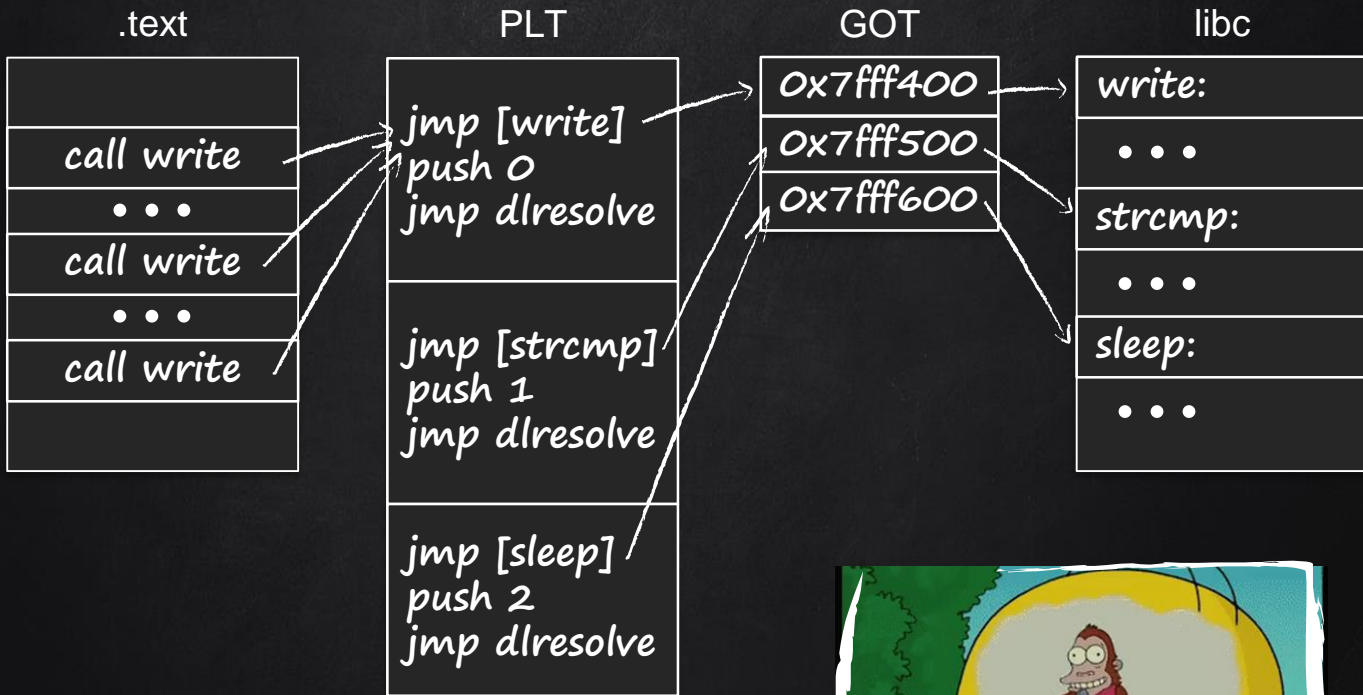


Address space layout randomization





PLT





BLIND ROP

Условия, ограничения, фазы,
stop/trap/syscall гаджеты,
BROR гаджет,
поиск PLT и остальных гаджетов



Еще раз зачем BROF



Нет бинаря



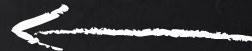
Можно строить
универсальные
эксплойты
под все системы



Условия

Уязвимость

Должно существовать
переполнение стека и известно
как его переполнить



Ревут после падения

Сервер перезапускает
приложение после его падения





Дерево вариантов

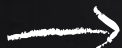




Фазы атаки



Чтение стека: лик канарейки и адреса возврата для обхода ASLR



Blind ROP: найти достаточно гаджетов для выполнения write и контроля его аргументов



Построение эксплоита:
сдамнить достаточную часть бинаря для поиска остальных гаджетов, построить ROP и запустить финальный эксплоит



Blind ROP



pop rdi; ret (сокет)



pop rsi; ret (буфер)



pop rdx; ret (длина)



write (функция из PLT)





Этапы – легкий путь



Найти BROP гаджет




Найти PLT

- write
- strcmp



STOP и TRAP гаджеты

Поведение программы:

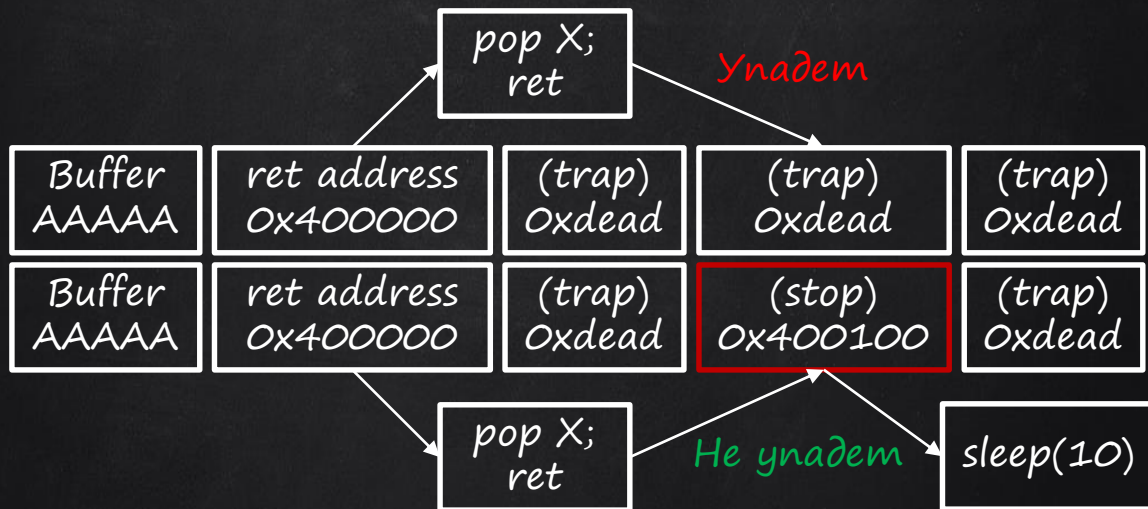
- упадет (trap)
- оставит коннект (stop) 

Виды “stop” гаджетов:

- sleep(10)
- вывод чего-либо
- остановка для чтения





Поиск pop гаджетов






Поиск гаджетов (probe)

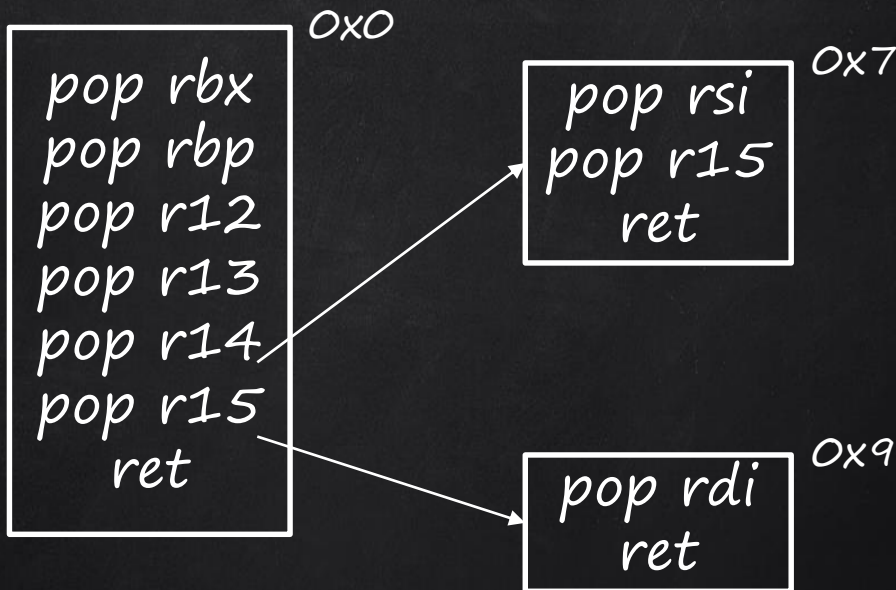
 `probe, stop, traps` – не делают “pop”
 (“ret”, “xor rax, rax; ret”)

 `probe, trap, stop, traps` –
 делают один “pop”
 (“pop rdi; ret”, “pop rax; ret”)

 `probe, trap(X6), stop, traps` –
 делают шесть “pop”
 (BR0P гаджет)

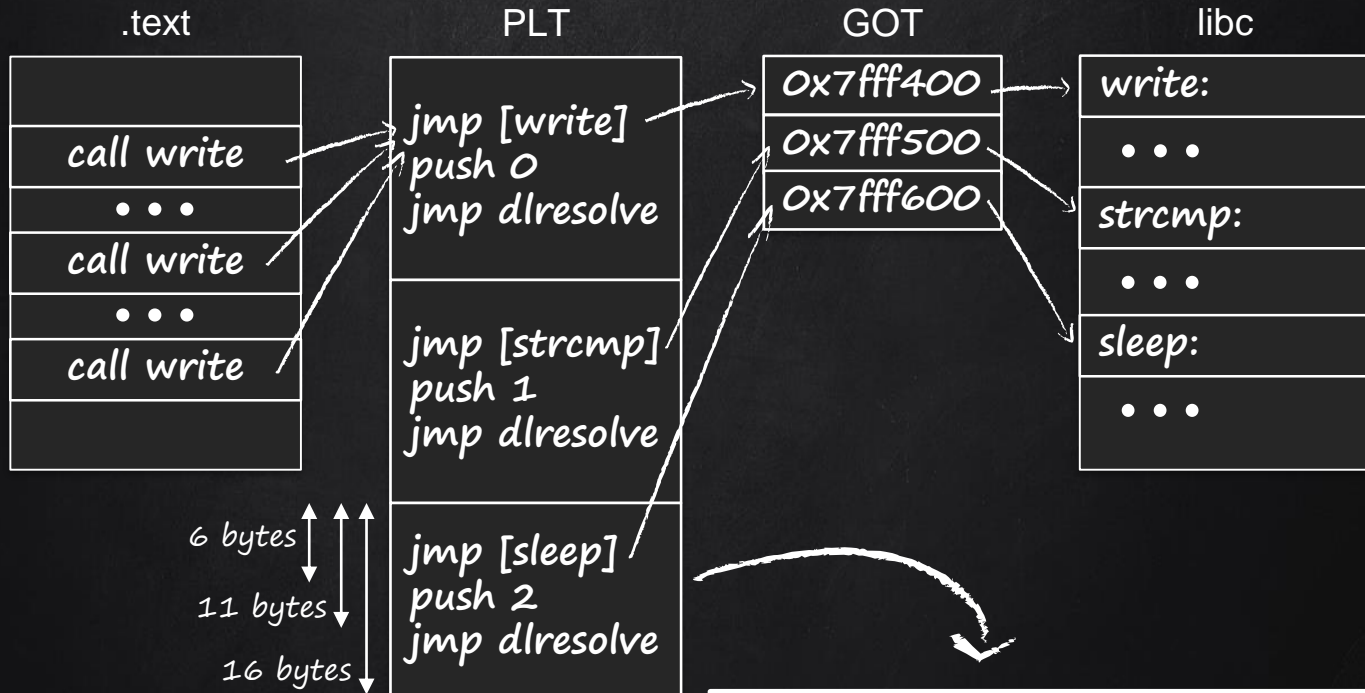


BROP за́жет для “pop rsi/rdi”





Пуск PLT



Пуск:
probe, stop, trap

При неправильных параметрах
вместо crash -> EFAULT.
Можем вызвать как plt_i и plt_i+6

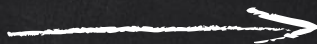


Контроль rdx через strcmp

```
int strcmp(const char *s1, const char *s2)
```

stack

pop rdi; ret
address 1
pop rsi; ret
address 2
strcmp



$rdx = X > 0$

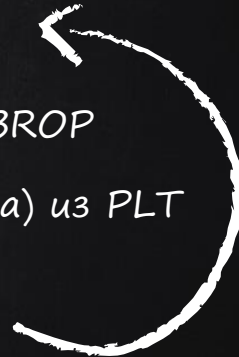
- ❧ strcmp(bad, bad): *ynaдем*
- ❧ strcmp(bad, readable): *ynaдем*
- ❧ strcmp(readable, bad): *ynaдем*
- ❧ strcmp(readable, readable): *не yнадем*



Дамним бинарь



- ⚙ `pop rdi; ret` (сокет) из BROP
- ⚙ `pop rsi; ret` (адрес дампа) из BROP
- ⚙ `strcmp` с параметрами (длина) из PLT
- ⚙ `write` из PLT





Усложняем





Blind ROP



pop rdi; ret (сокет)



pop rsi; ret (буфер)



pop rdx; ret (длина)



pop rax; ret
(номер сискола для write)



syscall





Этапы – трудный путь



Найти гаджеты:

- `pop rax`
- `pop rdi`
- `pop rsi`
- `pop rdx`



Найти `syscall`



Пуск “pop rax” и “syscall”



Цель – выполнить pause()

stack

pop rax; ret
0x22
syscall



Как – перебор всех “pop X; ret” + 0x22 + probe

pop X1; ret, 0x22, probe, pop X2; ret, 0x22, probe,
pop X3; ret, 0x22, probe, pop X4; ret, 0x22, probe,
pop X5; ret, 0x22, probe, pop X6; ret, 0x22, probe,

• • •

• • •

• • •



Пуск “pop rdi”



Цель – выполнить `nanosleep()`

stack

<code>pop rdi; ret</code>
<code>seconds</code>
<code>pop rax; ret</code>
<code>0x23</code>
<code>syscall</code>



Как – “`pop X; ret`” + `seconds` +
+ “`pop rax; ret`” + `0x23` + `syscall`



Пуск "pop rsi"

📌 Цель – выполнить kill()

stack

pop rdi; ret
sig
pop rax; ret
0x3e
syscall

📌 Как – "pop X; ret" + sig +
+ "pop rax; ret" + 0x3e + syscall



Пуск "pop rdx"

📌 Цель – выполнить `clock_nanosleep()`

stack

<code>pop rdx; ret</code>
<code>seconds</code>
<code>pop rax; ret</code>
<code>0xe6</code>
<code>syscall</code>

📌 Как – "`pop X; ret`" + `seconds` +
+ "`pop rax; ret`" + `0xe6` + `syscall`



Дампим бинарь



- ⚙ `pop rdi; ret` (сокет) через `nanosleep`
- ⚙ `pop rsi; ret` (адрес дампа) через `kill`
- ⚙ `pop rdx; ret` (длина) через `clock_nanosleep`
- ⚙ `pop rax; ret` (номер сокета write) через `pause`
- ⚙ `syscall` (вызов write) через `pause`



Викторина!





\$ 7

Что означает буква "В" в BROP?

A

Binary

B

Bind

C

Bear

D

Blind



\$ 37

На вызов какой функции нацелена атака BROP?

A print_flag

B read

C write

D system



\$ 337

Где будет лежать BROP заджет?

A

PLT

B

__libc_csu_init

C

libc

D

main



\$ 1337

Через что можно контролировать
регистр RDX?

A `execve`

B `strcat`

C `strlen`

D `strcmp`



PWN TIME