Bingo

flag

```
FLAG{THIS_challenge_is_too_easy_QQ}
```

參與解題的人

b04902053 鄭淵仁

b04902113 陳伯叡

scripts

- `Bingo.py` ：可以拿到 shell 的 `python3` script。
- `rand.c` ：可以生成題目要的 random 的 16 個數字。

write-up

程式的漏洞

- 這支程式會讓人輸入 16 個數字後，也 random 取 16 個數字出來跟使用者輸入的 16 個數字做比對，有 8 個以上的數字相同的話，使用者就可以寫入 `0x18` 個 bytes 進去從第 16 個數字以後，而且這支程式沒有開啟 `canary` 保護，所以這裡就有 stack overflow。
- 另外這支程式在 `rand` 數字之前，會先設定 `srand(0)` 。所以其實會取的數字是固定好的，我就先寫一個 `.c` 把 random 的數字都生出來，就保證一定可以到後面的 overflow 的地方了。
- 還有這支程式的 `stack` 段的位置是可以執行的，所以可以把 shell code 放到 stack 上在跳到那裡。

接下來就必須思考怎麼操作上述這些問題來取得 `shell` 。

利用第 16 個數字 leak 出 stack 的 address

首先，把第 16 個數字塞好塞滿不要留 `\0` 或 `\n` 之後， `printf` 的 `"%s"` 就會把第 16 個數字之後下一個 address 的值也 print 出來，而這個值正好是指向 stack 上的，所以只要用這個值就可以推算出其他 stack 上的其他 address。

寫 shell code `read()` + `jmp` + `execve("/bin/sh", 0, 0)`

首先由於可以 overflow 的 buffer 大小只有 18 bytes，其中 0 ~ 3 byte 是存第 16 個數字的地方，4~11 byte 是會被 pop 到 rbp 的地方，12 ~ 17 byte 則是會被當作 return address 的地方。

所以只有 12 個 byte 可以寫入值。就算再加上第 16 個數字也放 shell code ，也只有 16 個 byte ，不夠放執行 `system("/bin/sh")` 的 shell code 進去。

所以我就先放一個 `read` 的 shell code 進去，把 `execve("/bin/sh", 0, 0)` 的 shell code 讀到 `rsi` 指向的 address 上，再跳到 `rsi` 指向的 address 上，就可以執行 `execve("/bin/sh", 0, 0)` 並成功拿到 shell 了。