

# REVERSE

## ezASM

整个程序的核心功能，都在 Check Flag 段实现。以下是每条汇编语句的基本行为：

```
ezASM(Z) - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

; Check flag

xor esi, esi (将寄存器 esi 清零)

check_flag (循环) :
mov al, byte [flag + esi] (将数组中的第一个字节加载到寄存器al)
xor al, 0x22 (将al中的值与 0x22 进行异或)
cmp al, byte [c + esi] (将al 中的值与 c 数组中的第一个字节比较)
jne failure_check (直接报错)

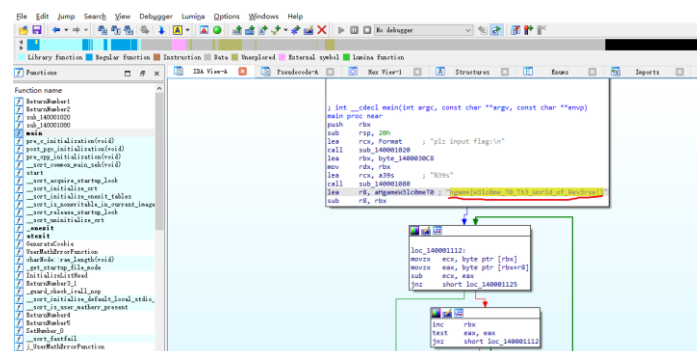
inc esi (寄存器 esi 的值加1, 即移动到下一个字节)
cmp esi, 33 (检查 esi 的值是否达到了 33,
jne check_flag 如果没有, 则跳转到标签 check_flag 继续进行下一轮的比较)
```

整体思路就是：对于用户输入，每次拿一位与 0x22 异或，结果要与数组 c 中的数一一对应。由此，可用 python 进行反向 print flag：

```
python 3.11
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type 'help', 'copyright', 'credits' or 'license()' for more information.
>>> ascii_array = [104, 103, 97, 109, 101, 123, 65, 83, 77, 95, 73, 115, 95, 73, 109, 112, 48, 114, 116, 52, 110, 116, 9
5, 52, 95, 82, 101, 118, 51, 114, 115, 51, 125, 0]
>>>
>>> # 将ASCII码数组转换为字符串
>>> result = ''.join(chr(i) for i in ascii_array)
>>>
>>> print(result)
hgme(ASM_ls_imp0rt4nt_4_Rev3rs3)
>>>
```

## ezIDA

放进 IDA 里，flag 就在 main 函数中显示：



至于后面的题，作为逆向萌新，只能说能力太次了，做不起……第三题想放到 OD 里脱 upx 壳。但是，从网上下了好几个版本的 OD，要不就是没证书不开调试功能，要不就拖入文件后显示只能调试 32 位，就黔驴技穷了；第四题，就是单纯搞不出来。

## PWN

ezSignIn

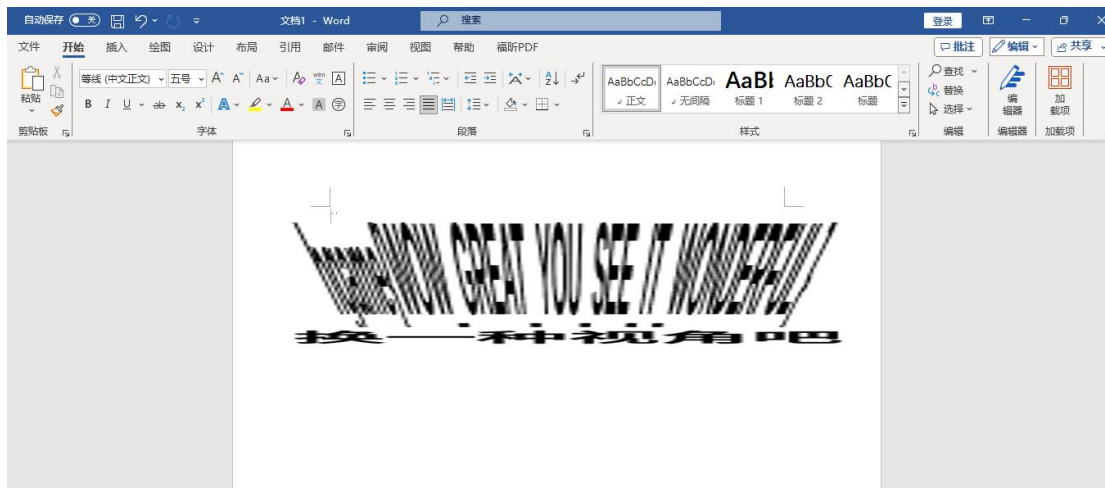
本来只准备看逆向题目的，但看到其他方向的某些题，答题量挺高的，所以就来讨点分了。  
这题扔到 kali 里，就直接返回 flag 了：

```
(kali㉿kali)-[~]  
$ nc 47.100.245.185 31017  
hgame{I_HATE_PWN}
```

## MISC

SignIn:

第一眼看上去就感觉图片被拉长了，但手上专业的工具一个没下，所以就寒酸地扔进 word 里拉长，勉强看出来了：



签到：

以目前的能力，感觉只有这道题适合自己做。