

web

Cosmos 的博客

提示是“修改很多次”“版本”“github”

前往 github 搜索 Cosmos，搜索到这题目的源码仓库，查看以前的版本
得到 base64 密文，解码，完成

接头霸王

这道题跟去年的 week1 某道题很像吧。

根据提示修改来源 Referer: <https://vidar.club/>

本地访问: X-Forwarded-For: 127.0.0.1

使用 Cosmos 浏览器: 我用的 chrome，于是把 http 包中的 Chrome 改成 Cosmos

提示要 2077 年后才能得到内容，修改 If-Unmodified-Since 到 2077 以后

最终的修改后的数据包为

GET / HTTP/1.1

Host: kyaruhgame.n3ko.co

Referer:https://vidar.club/

X-Forwarded-For:127.0.0.1

If-Unmodified-Since:Fri, 01 Jan 2087 00:00:00 GMT

User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Cosmos/60.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

Accept-Language: zh-CN,en-US;q=0.7,en;q=0.3

Accept-Encoding: gzip, deflate

Connection: close

Upgrade-Insecure-Requests: 1

Cache-Control: max-age=0

Code World

访问跳出 403，查看网页源码发现提示“302 跳转”，

于是用 burpsuite 拦截 302 跳转并拦截 response，访

问 <http://codeworld.hgame.day-day.work>抓包（注意把 url 中的 new.php 去掉（这是
跳转后的 page，所以得去掉），得到提示 405，

405 的意思是请求方式不正确，常见的请求无非 GET, POST。这里尝试把 GET 改成 POST
(√)

得到提示要在 url 上用参数 a 提交一个两数相加恒为 10 的东西

经过 n 次尝试，试出解法<http://codeworld.hgame.day-day.work/?a=1%2B9>

因为字符“+”会在发送时被转换为空格，用%2B 则会转化为“+”

cxk 打篮球

直接在浏览器上修改 main.js 文件

self.scene=new Scene(9)//直接跳到最后一关

修改 scene.js 文件

```
let s = { x: _main.score_x, y: _main.score_y, text: '分数: ', textLv: '关卡: ', score: 40000,
allScore: 40000, blockList: _main.blockList, blockListLen: _main.blockList.length, lv: 9, }
```

```
computeScore(){
```

```

let num=1
let allNum=this.blockListLen
num=1;
this.allScore=this.score*num
console.log("score:"+this.allScore);S
}}

```

RE Maze

拖进 ida7.0,找到 main () 函数后 F5 反汇编, 解读伪代码

```

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);
11    sub_4006A6();
12    __isoc99_scanf("%40s", s);
13    HIDWORD(v4) = strlen(s);
14    LODWORD(v4) = 0;
15    v5 = &unk_6020C4;
16    while ( v4 < SHIDWORD(v4) )
17    {
18        v3 = s[v4];
19        if ( v3 == 100 )
20        {
21            v5 += 4;
22        }
23        else if ( v3 > 100 )
24        {
25            if ( v3 == 115 )
26                // 115--s
27                //
28                v5 += 64;
29        }
30        else
31        {
32            v5 += 64;
33        }
34    }
35    puts("Illegal input!");
36    exit(0);
37    v5 -= 64;
38    // 119--w
39    }
40    else
41    {
42        if ( v3 != 97 )
43            // 输入错误
44            goto LABEL_12;
45        v5 -= 4;
46        // 97--a
47    }
48    if ( v5 < &unk_602080 || v5 > &unk_60247C || *v5 & 1 )// 小于602080或者大于60247C时错误 (这应该是地图的起始和终止处)
49        goto LABEL_12;
50    LODWORD(v4) = v4 + 1;
51    // 自增
52    if ( v5 == &unk_60243C )
53        // 出口
54        {
55            sprintf(&v7, "hgame{%s}", s, v4);
56            puts("You win!");
57            printf("Flag is: ");
58            puts(&v7);
59            exit(0);
60        }
61    }
62    }
63    }
64    }
65    }
66    }
67    }
68    }
69    }
70    }
71    }
72    }
73    }
74    }
75    }
76    }
77    }
78    }
79    }
80    }
81    }
82    }
83    }
84    }
85    }
86    }
87    }
88    }
89    }
90    }
91    }
92    }
93    }
94    }
95    }
96    }
97    }
98    }
99    }
100   }
101   }
102   }
103   }
104   }
105   }
106   }
107   }
108   }
109   }
110   }
111   }
112   }
113   }
114   }
115   }
116   }
117   }
118   }
119   }
120   }
121   }
122   }
123   }
124   }
125   }
126   }
127   }
128   }
129   }
130   }
131   }
132   }
133   }
134   }
135   }
136   }
137   }
138   }
139   }
140   }
141   }
142   }
143   }
144   }
145   }
146   }
147   }
148   }
149   }
150   }
151   }
152   }
153   }
154   }
155   }
156   }
157   }
158   }
159   }
160   }
161   }
162   }
163   }
164   }
165   }
166   }
167   }
168   }
169   }
170   }
171   }
172   }
173   }
174   }
175   }
176   }
177   }
178   }
179   }
180   }
181   }
182   }
183   }
184   }
185   }
186   }
187   }
188   }
189   }
190   }
191   }
192   }
193   }
194   }
195   }
196   }
197   }
198   }
199   }
200   }
201   }
202   }
203   }
204   }
205   }
206   }
207   }
208   }
209   }
210   }
211   }
212   }
213   }
214   }
215   }
216   }
217   }
218   }
219   }
220   }
221   }
222   }
223   }
224   }
225   }
226   }
227   }
228   }
229   }
230   }
231   }
232   }
233   }
234   }
235   }
236   }
237   }
238   }
239   }
240   }
241   }
242   }
243   }
244   }
245   }
246   }
247   }
248   }
249   }
250   }
251   }
252   }
253   }
254   }
255   }
256   }
257   }
258   }
259   }
260   }
261   }
262   }
263   }
264   }
265   }
266   }
267   }
268   }
269   }
270   }
271   }
272   }
273   }
274   }
275   }
276   }
277   }
278   }
279   }
280   }
281   }
282   }
283   }
284   }
285   }
286   }
287   }
288   }
289   }
290   }
291   }
292   }
293   }
294   }
295   }
296   }
297   }
298   }
299   }
300   }
301   }
302   }
303   }
304   }
305   }
306   }
307   }
308   }
309   }
310   }
311   }
312   }
313   }
314   }
315   }
316   }
317   }
318   }
319   }
320   }
321   }
322   }
323   }
324   }
325   }
326   }
327   }
328   }
329   }
330   }
331   }
332   }
333   }
334   }
335   }
336   }
337   }
338   }
339   }
340   }
341   }
342   }
343   }
344   }
345   }
346   }
347   }
348   }
349   }
350   }
351   }
352   }
353   }
354   }
355   }
356   }
357   }
358   }
359   }
360   }
361   }
362   }
363   }
364   }
365   }
366   }
367   }
368   }
369   }
370   }
371   }
372   }
373   }
374   }
375   }
376   }
377   }
378   }
379   }
380   }
381   }
382   }
383   }
384   }
385   }
386   }
387   }
388   }
389   }
390   }
391   }
392   }
393   }
394   }
395   }
396   }
397   }
398   }
399   }
400   }
401   }
402   }
403   }
404   }
405   }
406   }
407   }
408   }
409   }
410   }
411   }
412   }
413   }
414   }
415   }
416   }
417   }
418   }
419   }
420   }
421   }
422   }
423   }
424   }
425   }
426   }
427   }
428   }
429   }
430   }
431   }
432   }
433   }
434   }
435   }
436   }
437   }
438   }
439   }
440   }
441   }
442   }
443   }
444   }
445   }
446   }
447   }
448   }
449   }
450   }
451   }
452   }
453   }
454   }
455   }
456   }
457   }
458   }
459   }
460   }
461   }
462   }
463   }
464   }
465   }
466   }
467   }
468   }
469   }
470   }
471   }
472   }
473   }
474   }
475   }
476   }
477   }
478   }
479   }
480   }
481   }
482   }
483   }
484   }
485   }
486   }
487   }
488   }
489   }
490   }
491   }
492   }
493   }
494   }
495   }
496   }
497   }
498   }
499   }
500   }
501   }
502   }
503   }
504   }
505   }
506   }
507   }
508   }
509   }
510   }
511   }
512   }
513   }
514   }
515   }
516   }
517   }
518   }
519   }
520   }
521   }
522   }
523   }
524   }
525   }
526   }
527   }
528   }
529   }
530   }
531   }
532   }
533   }
534   }
535   }
536   }
537   }
538   }
539   }
540   }
541   }
542   }
543   }
544   }
545   }
546   }
547   }
548   }
549   }
550   }
551   }
552   }
553   }
554   }
555   }
556   }
557   }
558   }
559   }
560   }
561   }
562   }
563   }
564   }
565   }
566   }
567   }
568   }
569   }
570   }
571   }
572   }
573   }
574   }
575   }
576   }
577   }
578   }
579   }
580   }
581   }
582   }
583   }
584   }
585   }
586   }
587   }
588   }
589   }
590   }
591   }
592   }
593   }
594   }
595   }
596   }
597   }
598   }
599   }
600   }
601   }
602   }
603   }
604   }
605   }
606   }
607   }
608   }
609   }
610   }
611   }
612   }
613   }
614   }
615   }
616   }
617   }
618   }
619   }
620   }
621   }
622   }
623   }
624   }
625   }
626   }
627   }
628   }
629   }
630   }
631   }
632   }
633   }
634   }
635   }
636   }
637   }
638   }
639   }
640   }
641   }
642   }
643   }
644   }
645   }
646   }
647   }
648   }
649   }
650   }
651   }
652   }
653   }
654   }
655   }
656   }
657   }
658   }
659   }
660   }
661   }
662   }
663   }
664   }
665   }
666   }
667   }
668   }
669   }
670   }
671   }
672   }
673   }
674   }
675   }
676   }
677   }
678   }
679   }
680   }
681   }
682   }
683   }
684   }
685   }
686   }
687   }
688   }
689   }
690   }
691   }
692   }
693   }
694   }
695   }
696   }
697   }
698   }
699   }
700   }
701   }
702   }
703   }
704   }
705   }
706   }
707   }
708   }
709   }
710   }
711   }
712   }
713   }
714   }
715   }
716   }
717   }
718   }
719   }
720   }
721   }
722   }
723   }
724   }
725   }
726   }
727   }
728   }
729   }
730   }
731   }
732   }
733   }
734   }
735   }
736   }
737   }
738   }
739   }
740   }
741   }
742   }
743   }
744   }
745   }
746   }
747   }
748   }
749   }
750   }
751   }
752   }
753   }
754   }
755   }
756   }
757   }
758   }
759   }
760   }
761   }
762   }
763   }
764   }
765   }
766   }
767   }
768   }
769   }
770   }
771   }
772   }
773   }
774   }
775   }
776   }
777   }
778   }
779   }
780   }
781   }
782   }
783   }
784   }
785   }
786   }
787   }
788   }
789   }
790   }
791   }
792   }
793   }
794   }
795   }
796   }
797   }
798   }
799   }
800   }
801   }
802   }
803   }
804   }
805   }
806   }
807   }
808   }
809   }
810   }
811   }
812   }
813   }
814   }
815   }
816   }
817   }
818   }
819   }
820   }
821   }
822   }
823   }
824   }
825   }
826   }
827   }
828   }
829   }
830   }
831   }
832   }
833   }
834   }
835   }
836   }
837   }
838   }
839   }
840   }
841   }
842   }
843   }
844   }
845   }
846   }
847   }
848   }
849   }
850   }
851   }
852   }
853   }
854   }
855   }
856   }
857   }
858   }
859   }
860   }
861   }
862   }
863   }
864   }
865   }
866   }
867   }
868   }
869   }
870   }
871   }
872   }
873   }
874   }
875   }
876   }
877   }
878   }
879   }
880   }
881   }
882   }
883   }
884   }
885   }
886   }
887   }
888   }
889   }
890   }
891   }
892   }
893   }
894   }
895   }
896   }
897   }
898   }
899   }
900   }
901   }
902   }
903   }
904   }
905   }
906   }
907   }
908   }
909   }
910   }
911   }
912   }
913   }
914   }
915   }
916   }
917   }
918   }
919   }
920   }
921   }
922   }
923   }
924   }
925   }
926   }
927   }
928   }
929   }
930   }
931   }
932   }
933   }
934   }
935   }
936   }
937   }
938   }
939   }
940   }
941   }
942   }
943   }
944   }
945   }
946   }
947   }
948   }
949   }
950   }
951   }
952   }
953   }
954   }
955   }
956   }
957   }
958   }
959   }
960   }
961   }
962   }
963   }
964   }
965   }
966   }
967   }
968   }
969   }
970   }
971   }
972   }
973   }
974   }
975   }
976   }
977   }
978   }
979   }
980   }
981   }
982   }
983   }
984   }
985   }
986   }
987   }
988   }
989   }
990   }
991   }
992   }
993   }
994   }
995   }
996   }
997   }
998   }
999   }
1000  }

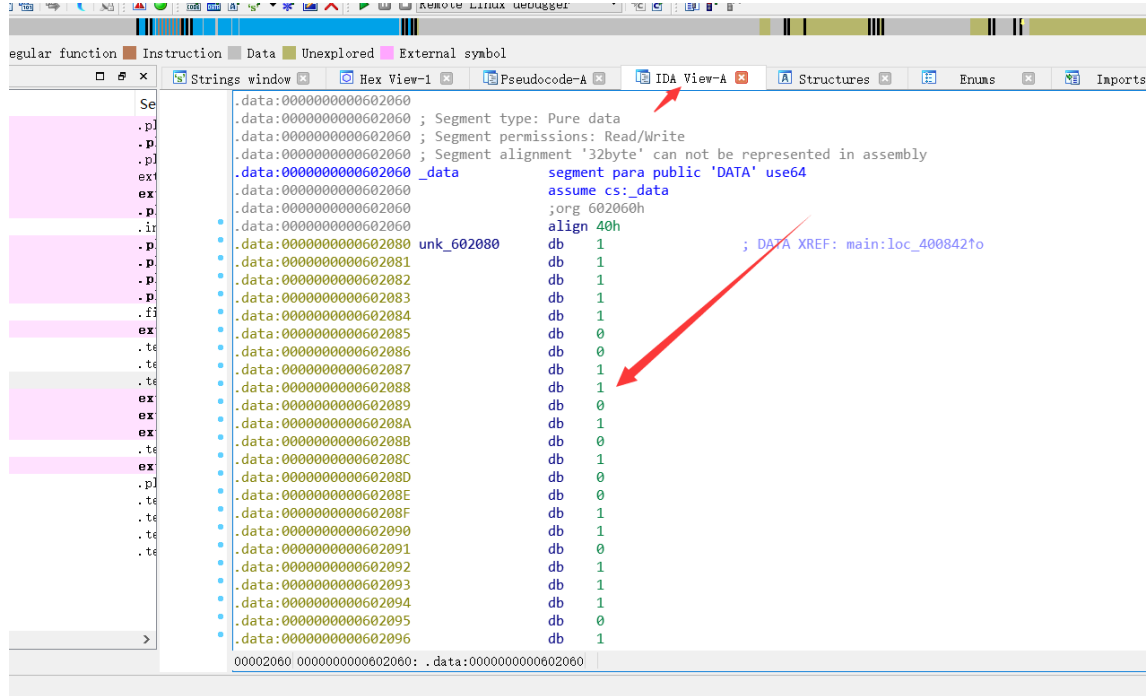
```

不难看出关系

W119 (-64)

A97 (-4) S115 (+64) D100 (+4)

至于地图，真正的地图需要自己找。发现 602080 到 60247C 之间有对应的 0, 1 值

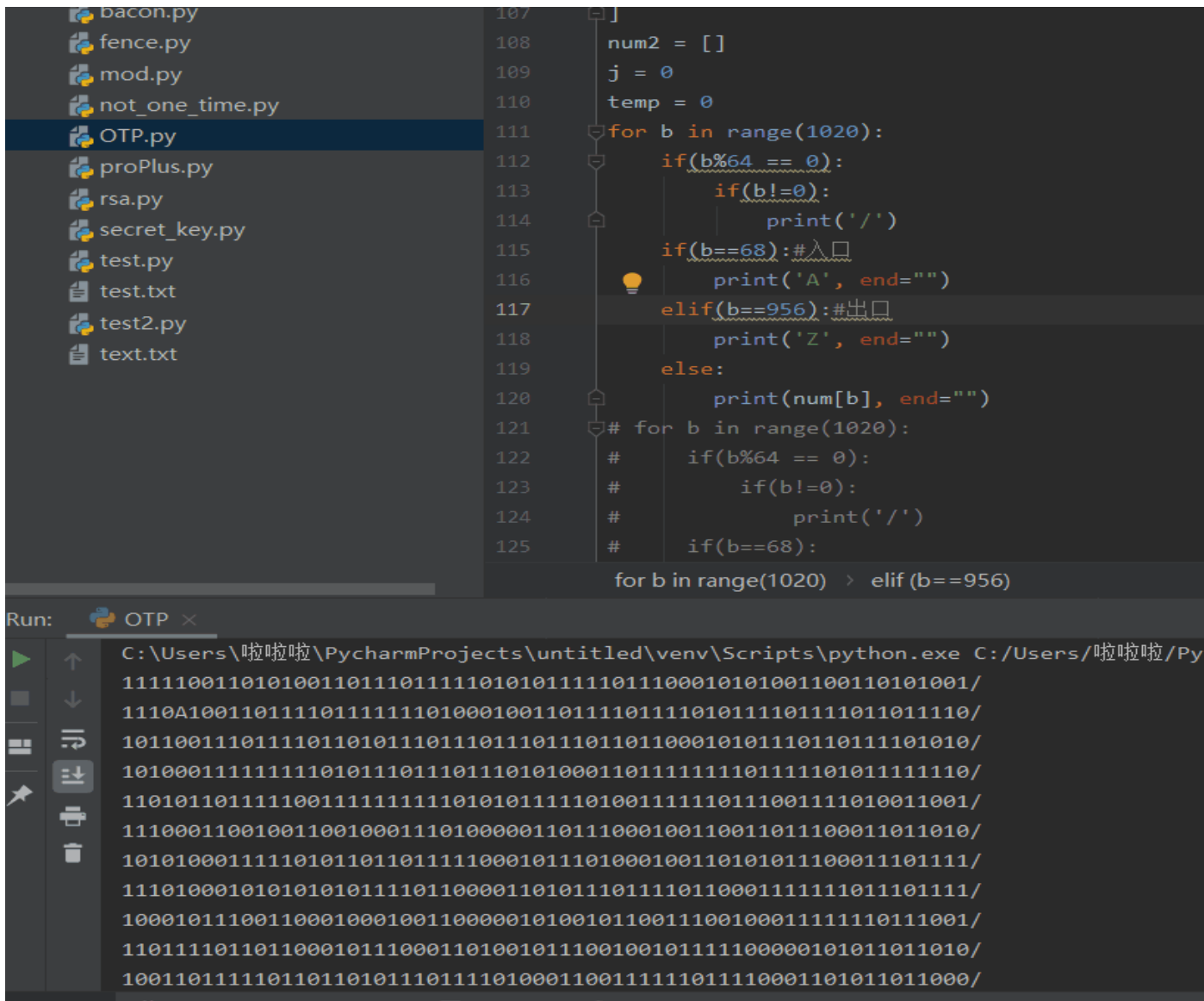


decompile, Ctrl-F5: decompile all.
t/Plugins menu for more information

由于方向 W, S 是加减 64, 那么可以猜出一行有 64 个 0, 1 值

A, D 加减 4, 那么左右行走的步长就是 4

将这 1024 个 0, 1 值取出, 并简单打印出来



The screenshot shows a PyCharm IDE with a file explorer on the left containing several Python files, with 'OTP.py' selected. The main editor displays the code for 'OTP.py', which is a 1020-length binary sequence. The code includes comments in Chinese and Python syntax for printing the sequence in groups of 64 bits, with special characters for entry ('A') and exit ('Z'). The bottom panel shows the output of the script, which is a 16x16 grid of binary digits (0s and 1s) with some characters replaced by '#' and 'o'.

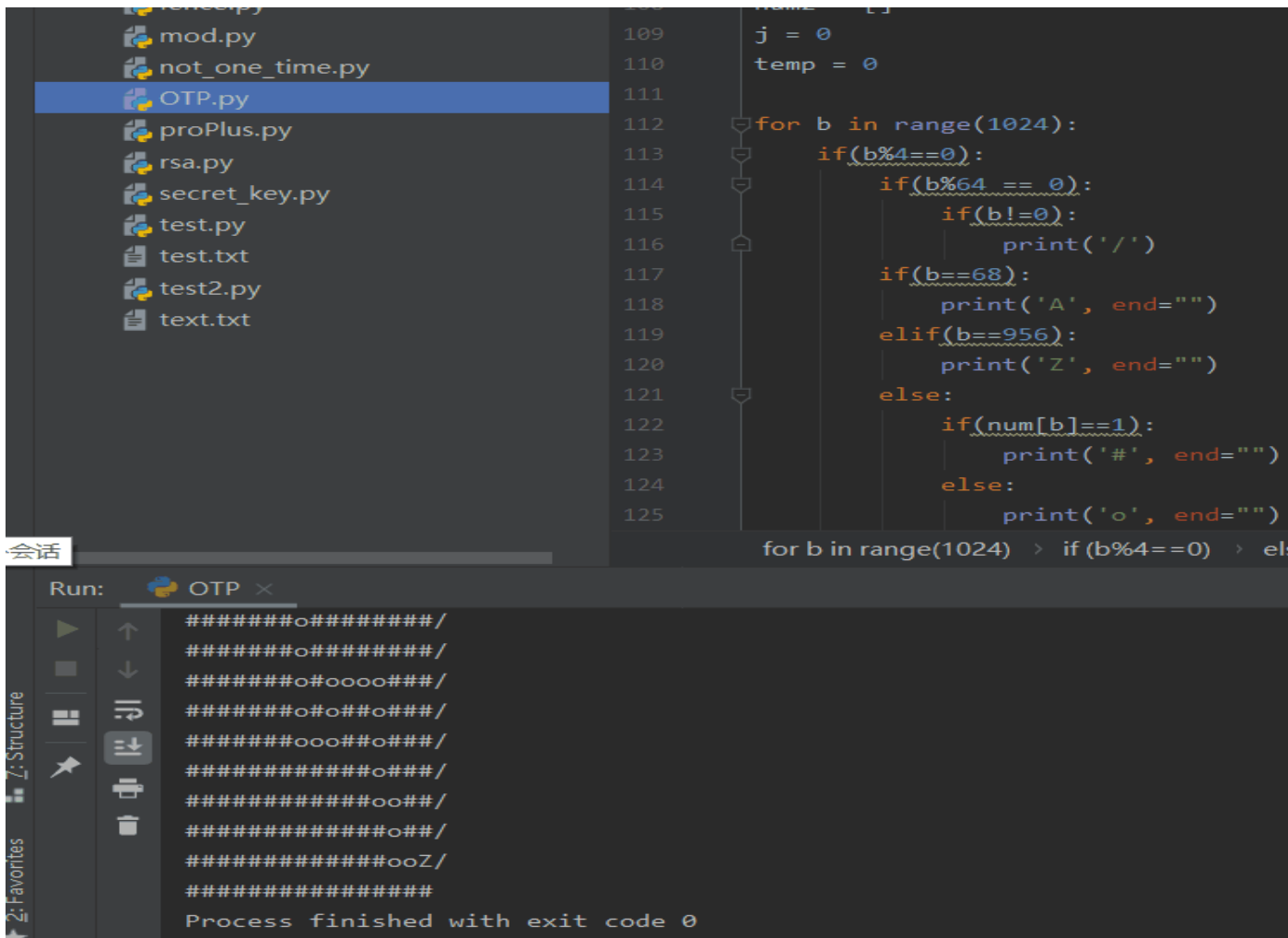
```
107 ]
108 num2 = []
109 j = 0
110 temp = 0
111 for b in range(1020):
112     if(b%64 == 0):
113         if(b!=0):
114             print('/')
115         if(b==68):#入口
116             print('A', end="")
117         elif(b==956):#出口
118             print('Z', end="")
119         else:
120             print(num[b], end="")
121     # for b in range(1020):
122     #     if(b%64 == 0):
123     #         if(b!=0):
124     #             print('/')
125     #         if(b==68):
126
for b in range(1020) > elif (b==956)
```

Run: OTP x

C:\Users\啦啦啦\PycharmProjects\untitled\venv\Scripts\python.exe C:/Users/啦啦啦/Py
11111001101010011011101111010101111011100010101001100110101001/
1110A1001101111011111101000100110111101111010111101111011110/
10110011101111011010111011101110111011101100010101110110111010/
10100011111111101011101110111010001101111111110111110101111110/
1101011011111001111111111101010111101001111110111001111010011001/
1110001100100110010001110100000110111000100110011011100011011010/
101010001111101011011011110001011101000100110101011100011101111/
1110100010101010101111011000011010111011110110001111111011101111/
1000101110011000100010011000001010010110011100100011111110111001/
1101111011011000101110001101001011100100101111100000101011011010/
10011011111011011010111011111010001100111111011110001101011011000/

刚好 64*16 矩阵

由于左右步长为 4，所以其实只要取出每行下标是 4 的倍数的项就可以了，这样就刚好是一个 16*16 的地图。为了方便观察，我用#和 o 代替 0，1，A 表示入口，Z 表示出口（图里 A 被挡住了）



```
108
109 j = 0
110 temp = 0
111
112 for b in range(1024):
113     if(b%4==0):
114         if(b%64 == 0):
115             if(b!=0):
116                 print('/')
117         if(b==68):
118             print('A', end="")
119         elif(b==956):
120             print('Z', end="")
121         else:
122             if(num[b]==1):
123                 print('#', end="")
124             else:
125                 print('o', end="")
126
127 for b in range(1024) > if (b%4==0) > el
```

Run: OTP ×

```
#####o#####/
#####o#####/
#####o#oooo###/
#####o#o##o###/
#####ooo##o###/
#####o###/
#####oo##/
#####o##/
#####ooZ/
#####
Process finished with exit code 0
```

然后在虚拟机里进行远程调试（ida 在 windows，动态调试），把代码跑起来，依次输入路径就 ok 了

Crypto

InfantRSA

```
用一段 python 解出答案
import libnum
import gmpy2
p = 681782737450022065655472455411
q = 675274897132088253519831953441
e = 13
c = 275698465082361070145173688411496311542172902608559859019841
n = p * q
phi_n = (p-1)*(q-1)
d = gmpy2.invert(e,phi_n)
print(d)
m = pow(c,d,n)
print("m=\n%s"%m)//得到一串数字
```


> abcdefghijklmnopqrstuvwxyz123456
apbjciilhfdkgenmoq6rzsy2xvtlwu435 的关系
可以逆推 hLgta\$5U{m+jemIp3}_TPA0TmeiuR!n! 得到 flag

Misc

签到题：略

壁纸

用记事本打开图片，在末尾得到提示：“Password is picture ID”
linux 下用 binwalk 跑一下图片 \$binwalk ~(图片).jpg，发现包含 zip 文件
用 dd 提取该 zip 文件，解压需要密码，提示“password is picture ID”
图片文件名是“Pixiv@純白可憐”，输入，密码错误。。。
Pixiv 是 P 站，找到画师“純白可憐”的作品“天意”（这张图的原名），输入，密码错误
发现浏览器地址栏上有图片 id，输入，密码（√）<（^ - ^）>
解压后得到 unicode 文段，解码，得到 flag

克苏鲁神话

得到压缩包和 Bacon.txt 文件。根据文件名可知是培根解密
先把 txt 中的英文句子根据大小写转化为培根密文
（由于不确定小写对应 A 还是大写对应 A，就把两种情况的结果都列出来，即下文代码的 m1 和 m2）

```
s='of SuCh GrEAt powers OR beiNGS tHere may BE conCEivAbly A SuRvIval oF HuGely  
REmOTE periOd.'  
m1=""  
m2=""  
for b in s:  
    if b.isupper():  
        m1+='A'  
        m2+='B'  
    elif b.islower():  
        m1+='B'  
        m2+='A'  
print(m1)  
print(m2)
```

得到明文，（后面用来解开 doc 文件加密）这里暂时告一段落
然后是发现压缩包中里面竟然也有一个一样的 Bacon.txt 文件
经过查询得知是要进行明文攻击了（在两个 Bacon.txt 都被压缩后，它们的 CRC 值相同时可用）
根据提示先将外部未加密的 Bacon.txt 文件用 7z 加密为 zip
（注意，不同软件加密算法有差别，题目已经提示这里用 7z 来处理了）
然后使用 ARCHPR 明文攻击，好慢。。。搞到一半停电了。。。
换成 pkcrack 来跑，半小时出结果。

```
./pkcrack -c "Bacon.txt" -p Bacon.txt -C ./test/Novel.zip -P ./test/Bacon  
.zip -d ./test/decrypt.zip  
//-C:要破解的目标文件(含路径)  
//-c:破解文件中的明文文件的名字(其路径不包括系统路径,从 zip 文件一层开始)  
//-P:压缩后的明文文件  
//-p:压缩的明文文件中明文文件的名字(也就是 readme.txt 在 readme.zip 中的位置)
```


//-d:指定文件名及所在的绝对路径，将解密后的 zip 文件输出

把 doc 文件放到 010editor 里。拉到末尾，得到“bacon is tasty! ”。。没啥用的样子
用 word 打开 doc 文件，真的在讲克苏鲁神话，在工具选项中把“隐藏文字”选项取消勾选

在文章末尾得到 flag

签到题 ProPlus

根据提示将前三行（包括空行和提示行）进行栅栏解密，栏数为 3；

再进行凯撒解密，偏移为 5。得到一句英文句子和压缩包密码

解开压缩包，得到 Brainfuck/Ook! 密文，

解密得到 base32 密文，进行 base32 解密

将上述解密结果再进行 base64 解密，简单观察可以发现是 png 图片的 ascii 码

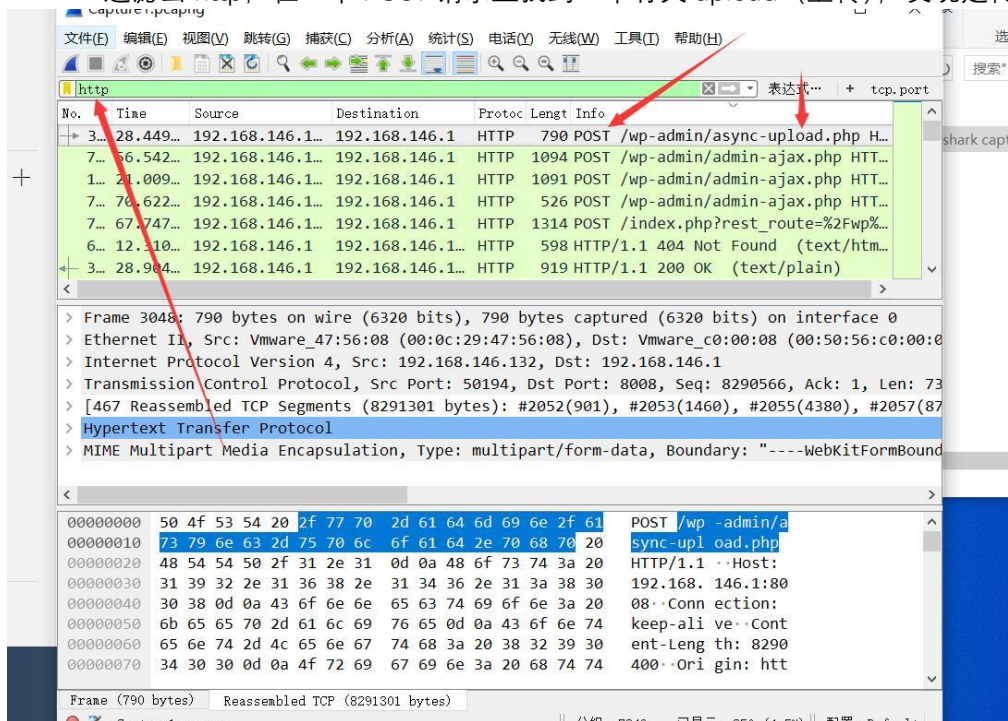
百度得知在浏览器地址栏输入 data:image/png;base64,<base64 密文>可以直接显示图片

得到一张二维码图片，微信扫码得到 flag

每日推荐

用 wireshark 打开。根据提示，猜测是有一个音乐文件

过滤出 http，在一个 POST 请求里找到一个有关 upload（上传），发现是传一首音乐！





把数据包导出，扔到 binwalk 里，发现有 zip 文件，分离出 zip
提示需要 6 位数字密码，用 Advanced Archive Password Recovery 暴力破解
得到 MP3 文件，用 Audacity 打开，把波形转化为频谱图，得到 flag