

WEB:

1. 今天早上研究 git, 下午理发店门口排队的时候在看国外大佬介绍 git 用法的时候提到了 git 的 config 泄露啥的, 我由于从比赛第一天就一直在苦苦思考版本管理工具, git, github 和博客历史版本之间的关系, 所以查了一堆资料, 结果完全没头绪, 因为网上根本没人提到这档事。我于是这两天把思路放回 git 上, 觉得只要我全面一点了解 git, 就能找出头绪。果然功夫不负有心人, 突然看到在 url 后面加上 .git/config 再搜索就可以追寻回版本目录啥的。因为早上我刚好理解过了 .git 是 repo 的重要目录, 所以马上认定, 这一定就是这道题的解法。回家以后我就马上实施, 果然发现没错, 然后又看到一堆评论在下面感慨。但我先是把 commit 内容当十六进制捣鼓了一番没结果, 然后突然看到 base64 解码的提示, 感觉整了人都智障了。



The screenshot shows a web browser at the URL `cosmos.hgame.n3ko.co/.git/config`. The content of the `.git/config` file is displayed as follows:

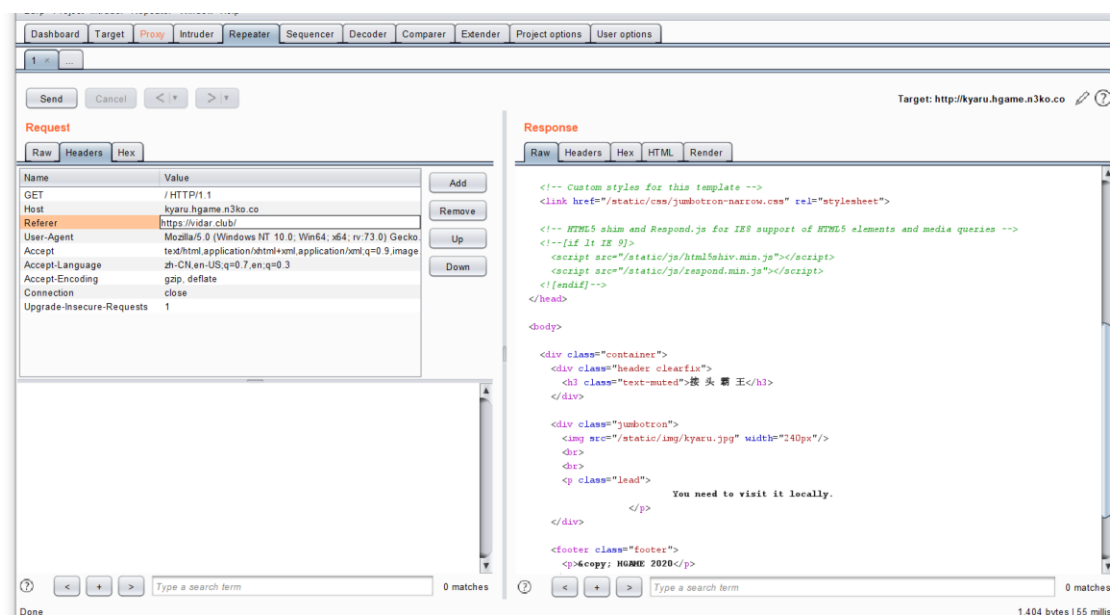
```
[core]
  repositoryformatversion = 0
  filemode = true
  bare = false
  logallrefupdates = true
[remote "origin"]
  url = https://github.com/FeYcYodhrPDJSru/8LTUKCL83VLhXbc
  fetch = +refs/heads/*:refs/remotes/origin/*
```

Below the file content, a commit message is shown for commit `f79171d` by `FeYcYodhrPDJSru`, committed 13 days ago. The commit message is:

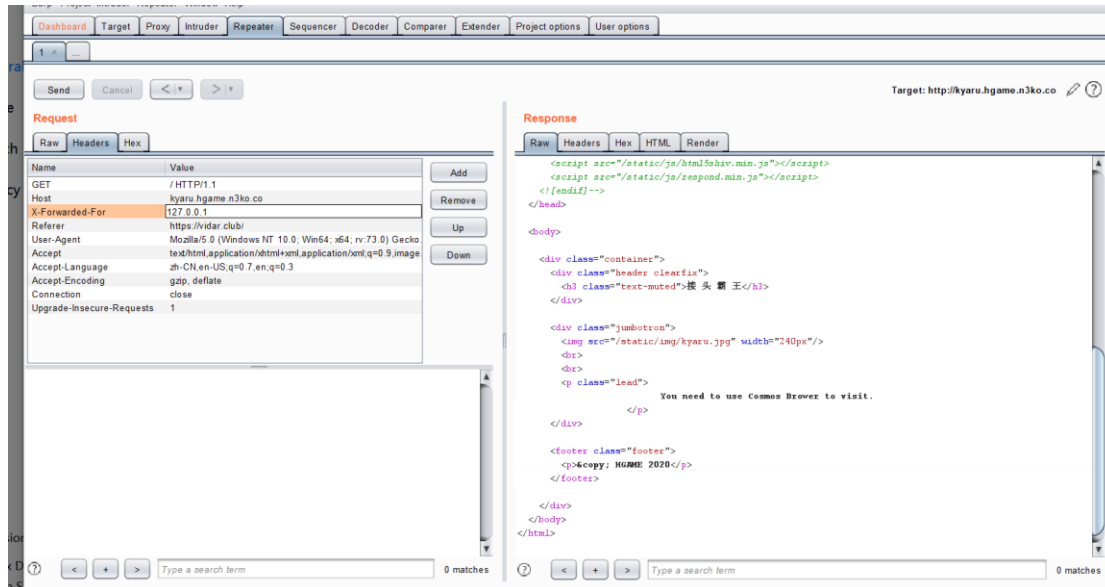
```
@@ -0,0 +1 @@
+ base64 解码: aGdhbWV7ZzF0X2x1Q0t0f0XNfZGFuZ2VYMHVzXyEhISF9
```

The commit is identified as `on commit f79171d`.

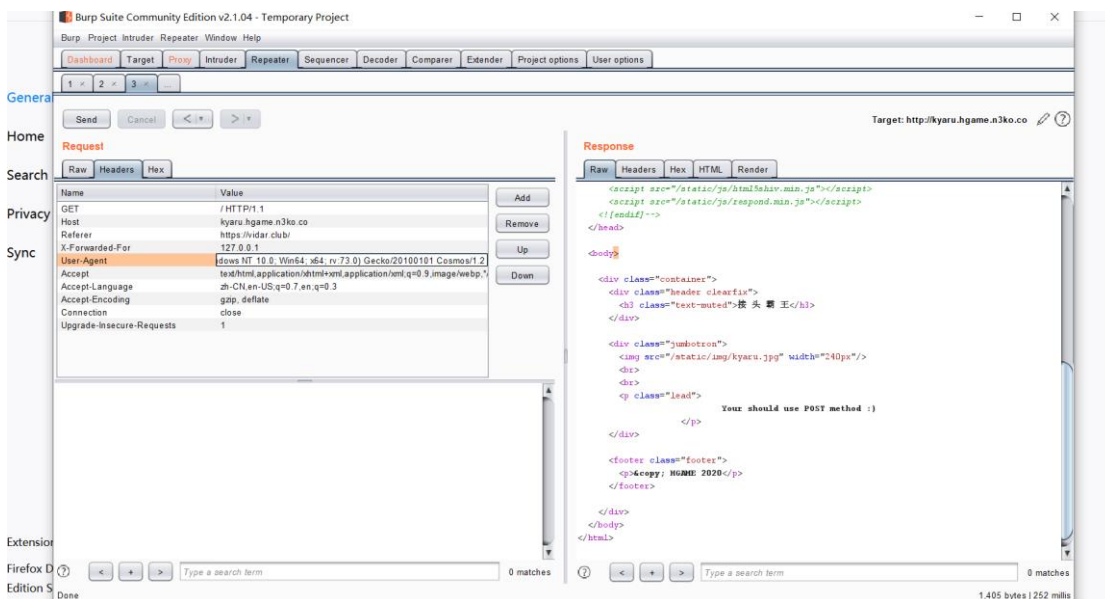
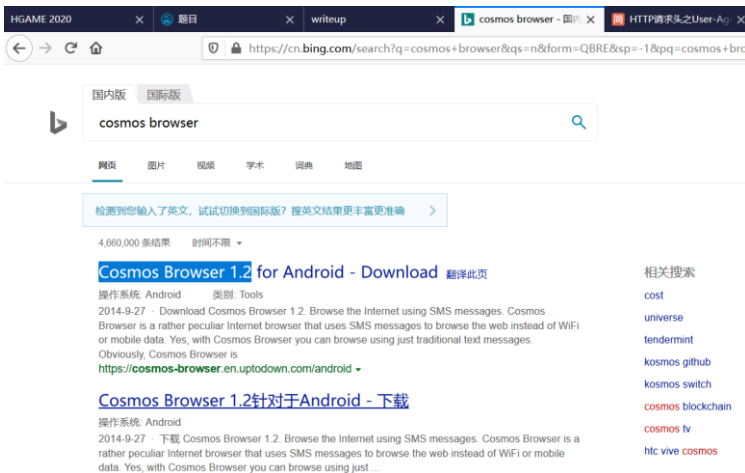
2. 由于是要从 <http://vidar.club/> 来访问该网站, 于是想到把 Referer 伪造改成该网址, 由于本来没有所以就添加一个。

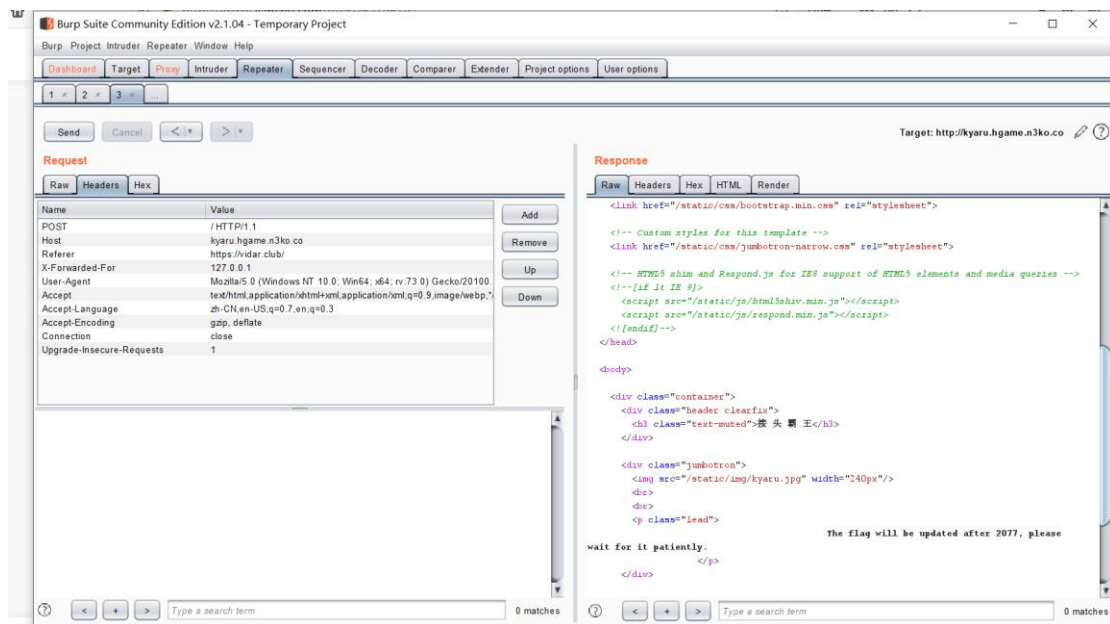


新访问的网站又说要 locally, 想想就知道伪造 ip 地址说明自己是从自己本地 ip 来访问的, 于是把 X-Forwarded-For 改成 127.0.0.1 即可。

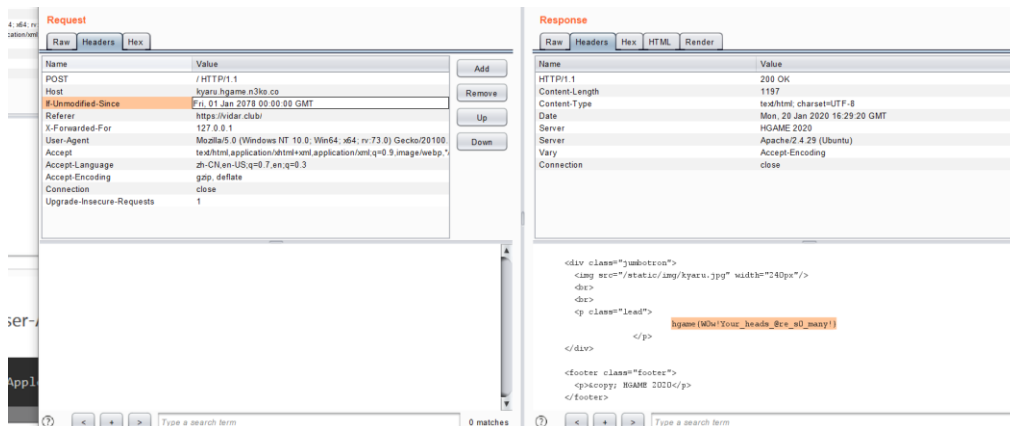
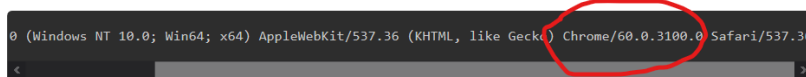


新网站又说要用 cosmos 浏览器浏览，查了一番资料后明白了这跟 user-agent 有关，于是把 user-agent 的最后一个参数按原来的格式改成 cosmos，而且还要注意版本号 and cosmos 之间要加上/，因为原来的中间就有。





我用的Chrome浏览器，查看User-Agent的结果：



哭死了。。。。。。最后一个试了好久好久的 date，各种思路，想说是改变请求时间，还是改变页面本身的不同时间下的版本，怎么都不对，整整浪费了快一个小时，最后又回去看这表

Content-Type	请求的与实体对应的MIME信息	Content-Type: application/x-www-form-urlencoded
Date	请求发送的日期和时间	Date: Tue, 15 Nov 2010 08:12:31 GMT
Expect	请求的特定的服务器行为	Expect: 100-continue
From	发出请求的用户邮箱	From: user@email.com
Host	指定请求的服务器的域名和端口号	Host: www.zcmhi.com
IF-Match	只有请求内容与实体匹配时才有	IF-Match: "737060cd8c284d8af7ad3082f209582d"
IF-Modified-Since	如果请求的部分在指定时间之后被修改，请求成功，未被修改则返回304代码	IF-Modified-Since: Sat, 29 Oct 2010 19:43:31 GMT
IF-None-Match	如果内容未改变返回304代码，参数为服务器发回的Etag，与服务器发回的Etag以判断是否改变	IF-None-Match: "737060cd8c284d8af7ad3082f209582d"
IF-Range	如果实体未改变，服务器发送客户端丢失的部分，否则发送整个实体，参数也为Etag	IF-Range: "737060cd8c284d8af7ad3082f209582d"
IF-Unmodified-Since	只在实体在指定时间之后未被修改才请求成功	IF-Unmodified-Since: Sat, 29 Oct 2010 19:43:31 GMT
Max-Forwards	限制信息通过代理和网关传递的时间	Max-Forwards: 10
Pragma	用来包含实现特定的指令	Pragma: no-cache
Proxy-Authenticate	代理认证所需的认证信息	Proxy-Authenticate: Basic

觉得下面这个好像也跟时间有关，要不随便试一试算了，结果一试居然就对了，再回头认真

想一想，好像还真的是这个理。。。泪流满面

3. 进行晕头转向的 js 代码审计，发现 game class 下的 gameOver 这个函数有猫腻，会向外发送一个 url，里面还包含 globalScore，一下子断定要通过修改 url 来获得 flag（其实这道题我从第一天就开始想到周三，是周三下午才突然发现这个函数的猫腻的）。一开始的思路是自己伪造出一个符合条件的 url，使得 score 的值是比 30000 大的数，然而查了不少资料后还是写不出来。。。最后突然灵光一闪想起可以先用 burpsuite 抓包，然后修改再发送达成目的。

The image shows a code editor window with a JavaScript function named `gameOver()`. The function calculates a score based on the current `globalScore` and a timestamp, then sends an alert with the score. Below the code editor, there is a screenshot of Burp Suite showing the raw HTTP request and response for a POST to `/submit`. The request includes a `Cookie` and a `score` parameter. The response shows the server's status and headers.

```
gameOver() {  
  let po = 'ejIy';  
  let rt = po + 'LmWj';  
  let rou = 'L3N1Ym';  
  let sche = 'aHR0c';  
  let k = 'c2Nw';  
  let me = sche + 'DovL2N';  
  clearInterval(this.timer)  
  this.context.clearRect(0, 0, this.canvas.width, this.canvas.height)  
  let stamp = md5(Date.parse(new Date()) / 1000);  
  this.globalScore = this.globalScore + this.storageScore;  
  this.context.font = '32px Microsoft YaHei'  
  this.context.fillStyle = '#000'  
  this.context.fillText('CXK, 你球掉了! 得分: ' + this.globalScore, 404, 226)  
  $('#ballspeedset').removeAttr('disabled');  
  let s = this.globalScore;  
  (function () {  
    let getU = me + '4ay5oZ';  
    let r1 = getU + '2FtZ553';  
    let te = rou + '1pdA';  
    let ey = k + 'cmU-';  
    $.post(atob(r1 + rt + te), atob(ey) + '=' + s + '|' + stamp, function (data) {  
      alert(data);  
    })  
  }) ();  
  this.globalScore = 0;  
}
```

Raw Params Headers Hex

POST /submit HTTP/1.1
Host: cxk.hgame.wz22.cc
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:73.0) Gecko/20100101
Firefox/73.0
Accept: */*
Accept-Language: zh-CN,en-US;q=0.7,en;q=0.3
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
X-Requested-With: XMLHttpRequest
Content-Length: 44
Origin: http://cxk.hgame.wz22.cc
Connection: close
Referer: http://cxk.hgame.wz22.cc/
Cookie: __cfduid=db8a553860c02387d7c79ea6d799f33721579181930
score=40000|ff2cba3f4cedf30ae56e9433b7fdf6

Raw Headers Hex Render

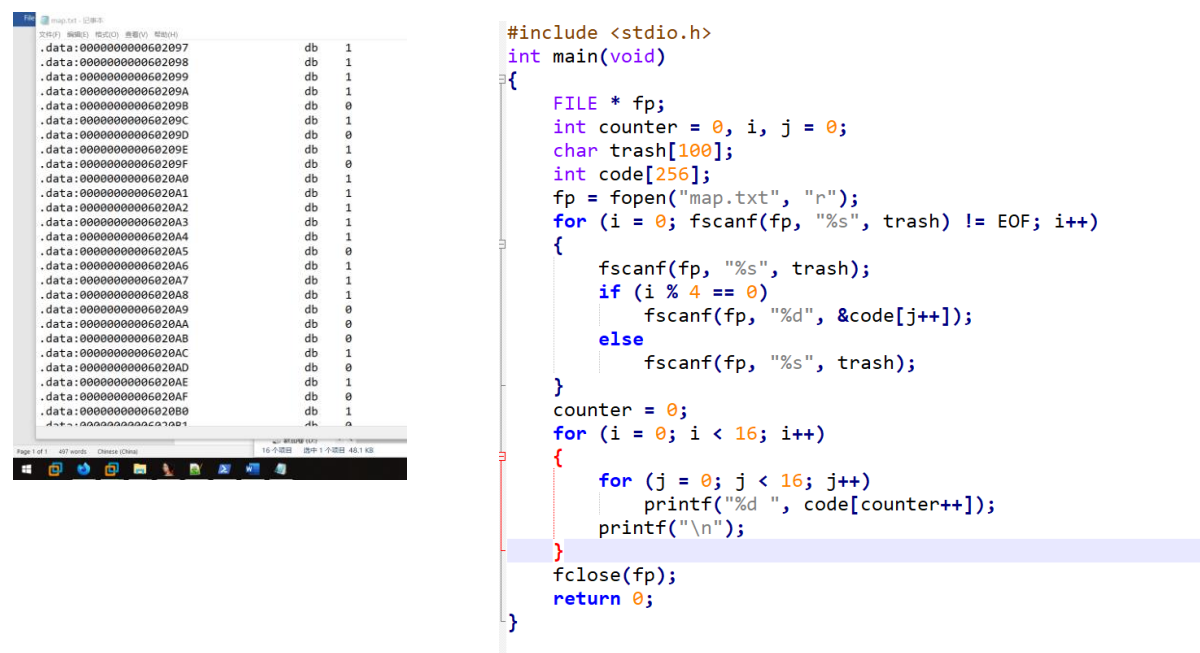
HTTP/1.1 200 OK
Date: Wed, 22 Jan 2020 12:11:57 GMT
Content-Type: text/plain; charset=utf-8
Content-Length: 49
Connection: close
Access-Control-Allow-Origin: http://cxk.hgame.wz22.cc
Vary: Origin
Vary: Accept-Encoding
CF-Cache-Status: DYNAMIC
Alt-Svc: h2=":443"; ma=60
Server: cloudflare
CF-RAY: 55916ef16fdf9cf4-AMS
hgame(j4vAScript_will_tell_yOu_somethin9_u5eful?!)

卑鄙地拦截下 http 的 post，并将 score 改为 40000，然后再发送，搞定。

REVERSE:

1. 由题目知道是迷宫类型的题目，由于先前在攻防世界见过一道迷宫题，所以一看到好多的 label 马上就挂上勾了，又发现好多 if else 语句，认真一看发现是分析比较 v5 的值的，一查 ASCII 码发现刚好对应打游戏的上下左右，wsad。

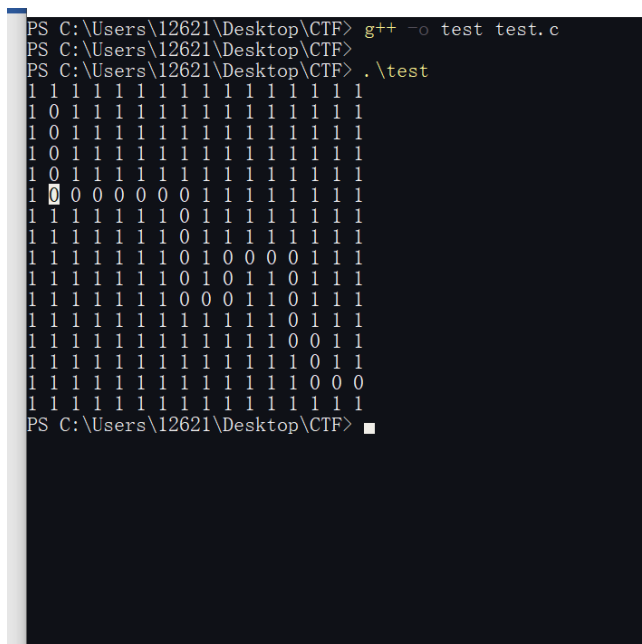
由此处分析发现 v5 必须大于 602080 且 60247C，而且 v5 指向的值不能等于 0。再回头看看前面对 v5 的交代，发现 v5 是一个字符型指针，且一开始指向 6020C4，最终胜利时候是指向 60243C 的。说实话一开始我因为攻防世界里面提供的思路是，数据块里面会存放迷宫的地图，我只要拼接起来就好了，结果在数据块里面来回找，只发现了 kali 里面运行程序就出来的图像，和另一个不可名状的图像，由此苦恼了好久。后来突然一下子灵光一闪，发现控制 v5 移动相当于加减指针值，只要我把 v5 指针通过一系列移动就可以移到目的地了。再一想，要是只是单纯移动指针，那上下或者左右移动不是就抵消了，而且左右有什么意义呢？又看了对 v5 加减的值后，我突然明白了一点东西，再一看内存每个地址上都定义了 0 或 1，而在 602080~60247C 范围之外的内存上却没有定义，我直到这个时候才大体上理清了思路。于是我把这段数据复制到记事本，因为数据实在太多只好写 c 脚本处理 (python 掌握不好，不如写 c 来的快，以后一定会好好学)，如下：



The image shows a Notepad++ window with two tabs. The left tab, titled 'map.txt - 记事本', contains a memory dump with addresses from 00000000 to 0000000F and values 0 or 1. The right tab contains a C++ program that reads the data from 'map.txt' and processes it into a 16x16 grid.

```
#include <stdio.h>
int main(void)
{
    FILE * fp;
    int counter = 0, i, j = 0;
    char trash[100];
    int code[256];
    fp = fopen("map.txt", "r");
    for (i = 0; fscanf(fp, "%s", trash) != EOF; i++)
    {
        fscanf(fp, "%s", trash);
        if (i % 4 == 0)
            fscanf(fp, "%d", &code[j++]);
        else
            fscanf(fp, "%s", trash);
    }
    counter = 0;
    for (i = 0; i < 16; i++)
    {
        for (j = 0; j < 16; j++)
            printf("%d ", code[counter++]);
        printf("\n");
    }
    fclose(fp);
    return 0;
}
```

作用是从记事本每 4 个提取一个数出来，这样做我其实是分析了好久好久 (哭死)。。。因为起始点和终点的十六进制换算成十进制后，再除以 4，都能被整除，而移动的步数都是 4 的



The image shows a Windows command prompt with the following commands and output:

```
PS C:\Users\12621\Desktop\CTF> g++ -o test test.c
PS C:\Users\12621\Desktop\CTF>
PS C:\Users\12621\Desktop\CTF> .\test
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 0 1 0 0 0 0 1 1
1 1 1 1 1 1 1 1 0 1 0 1 1 0 1 1
1 1 1 1 1 1 1 1 0 0 0 1 1 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
PS C:\Users\12621\Desktop\CTF>
```

倍数，故位置一定也是 4 的倍数。更加巧的是，64 除以 4 后刚好等于 16，而地图总长度是 255 还是 256 来着，反正睁一只眼闭一只眼，就当作 256 个了，刚好是 16x16，所以上移下移的加减 16 (因为这时我已经全部缩小 4 倍了，这样好分析多了)，刚好就表示上移下移一格。这下就一切都明明白白了。说实话一开始我是从下往上走的，结果一直提交一直错；后来突然发现应该从上往下走，再提交，还是错的 (心态炸裂)，好在最后发现是第一步下来的 4 个 s 我打成 5 个了 (吐血)。

PWN:

1. 这道题最后没做出来。。。但具体思路是有的:

```
for ( j = 0; j <= 7; ++j )
{
    *((_BYTE *)&v14 + j) ^= *(&v6 + j);
    if ( *((_BYTE *)&v14 + j) != byte_602050[j] )
    {
        puts("sry, wrong flag");
        exit(0);
    }
}
```

由这块内容推断出 v14 的内容。

其中 byte_602050 为 e4sy_Re_

```
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);
    }
}
```

再由这块内容退出 v16 的内容。

```
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;
}
```

由这块内容推断出位运算前 v14 和 v16 的内容

```
sub_400616((__int64)&v14, (__int64)&v24);
sub_400616((__int64)&v16, (__int64)&v25);
```

由这两个函数推断出输入的花括号之间的 32 个字符。

1LL 一下载往后移了 8 个字节

花括号之间一共 32 个字符

v24 解决了前 16 个

v25 解决了后 16 个

同时使得 v16 和 v14 分别对应的 8 个字符被赋值。

CRYPTO:

1. 已知:

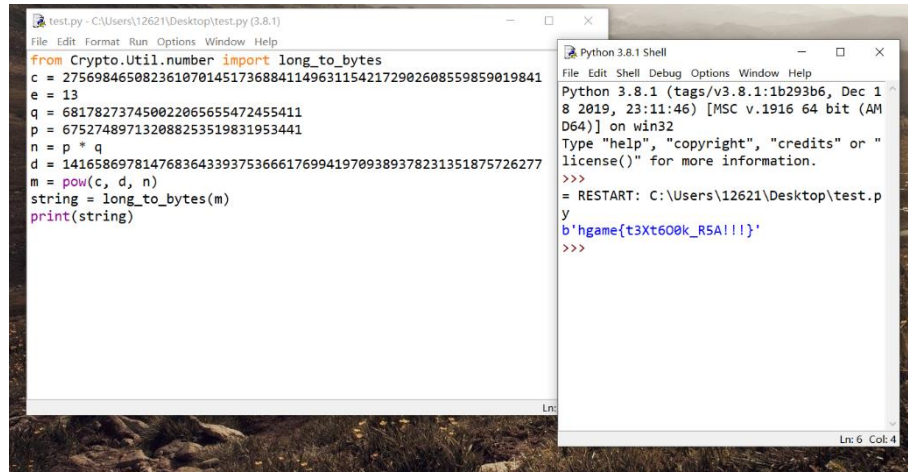
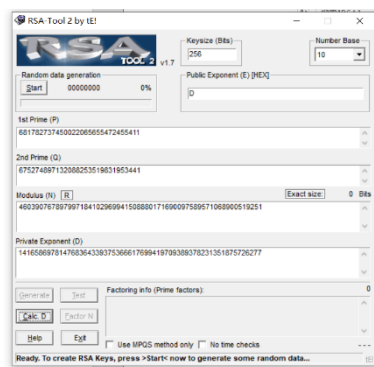
第一个质数 $p = 681782737450022065655472455411$;

第二个质数 $q = 675274897132088253519831953441$;

密钥 $e = 13$;

公钥 $n = p * q = 460390767897997184102969941508880171690097589571068900519251$;

利用 RSA-Tool 2 by tE! 计算出 d



2. 先制作了一个转换器根据对应关系，输入字母输出对应索引

```
#include <stdio.h>
#include <string.h>
int main(void) {
    char ch;
    int index;
    char table[] = "zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM";
    while (scanf("%c", &ch) == 1) {
        for (index = 0; index < strlen(table); index++) {
            if (table[index] == ch)
                break;
        }
        printf("%d\n", index);
    }
}
```

$$(12A + B) \% 62 = 46$$

$$(11A + B) \% 62 = 33$$

$$(7A + B) \% 62 = 43$$

$$(6A + B) \% 62 = 30$$

$$(18A + B) \% 62 = 0$$

解得：

$$A = 13 + 62k_1$$

$$B = 14 + 62k_2$$

验证发现能使得所有五个式子都成立

于是取 $A = 13$, $B = 14$

写程序解码，即可得 flag。

```

#include <stdio.h>
#include <string.h>
int main(void) {
    int i, j;
    char table[] = "zxcvbnmasdfghjklqwertyuiop1234567890QWERTYUIOPASDFGHJKLZXCVBNM";
    char code[] = "xr1AJ7havGTPH410";
    char words[20];
    for (i = 0; i < strlen(code); i++) {
        for (j = 0; j < strlen(table); j++) {
            if (code[i] == table[(13 * j + 14) % 62]) {
                words[i] = table[j];
                break;
            }
        }
    }
    words[i] = '\0';
    puts(words);
    return 0;
}

```

```

PS C:\Users\12621\Desktop\CTF> .\decode
M4thu5EdiNcRYpt0
PS C:\Users\12621\Desktop\CTF>

```

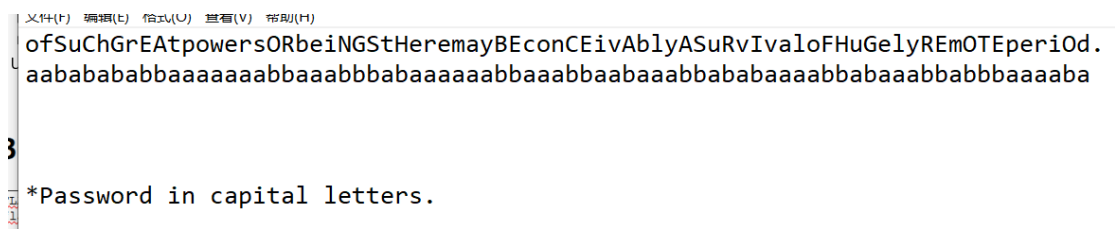
MISC:

1. 先 base64 转换，得到莫尔斯密码，然后再转换得到英文字符串，最后不知道为啥还要在单词间加上下划线才能通过。hgame{W3LC0ME_TO_2020_HGAM3}
2. 先用 foremost 解密 jpg，得到需要密钥的 txt 文件，再将压缩包放入 winhex 中发现如下，于是上 p 站查找该画师找到该图片，查看图片地址，根据后面的一串数字得出 id，

00 00 00 00 21 00 00 00 00 00 00 00 20 00 00 00	flag.txt
00 00 00 00 66 6C 61 67 2E 74 78 74 0A 00 20 00	E"@=ÓÆÕ
00 00 00 00 01 00 18 00 45 A8 40 3D D3 C6 D5 01	E"@=ÓÆÕ ç, I ÓÆÕ
5 A8 40 3D D3 C6 D5 01 E7 B8 9C 1C D3 C6 D5 01	PK Z
0 4B 05 06 00 00 00 00 09 00 01 00 5A 00 00 00	v Password i
6 00 00 00 17 00 50 61 73 73 77 6F 72 64 20 69	s picture ID.
3 20 70 69 63 74 75 72 65 20 49 44 2E 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	

打开文件发现是 unicode 编码，解码即可得到

把空格去掉，令小写为 a，大写为 b 翻译一遍得：



3. 没做出来, 只发现 bacon 文档用培根解密后得到密码 flaghiddenindoc, 然而尝试解密文档发现并不是密码。。。。。

Bugku | 培根密码加解密

