Lab 0：基础知识及技能

> 3250101697冯梓航

> 由于最近任务略有点多，这个报告很简陋，请见谅。期末之后如果可以重新提交的话我会尽可能补齐！

1. Prerequisite

1.1 Challenge 1：Linux环境的搭建与简单使用

1. 在实验报告中给出任意 4 个 shell 命令的用法介绍以及在 Linux 环境下的实操截图

(a) ls 命令

```
ls                      # 简单列出文件
ls -l                   # 详细列表（权限、大小、修改时间）
ls -a                   # 显示隐藏文件
ls -lh                  # 显示文件大小（KB/MB）
```

(b) cd 命令

```
cd /home                # 进入 /home 目录
cd ..                   # 返回上一级目录
cd ~                    # 回到自己的家目录
cd -                    # 回到上一次所在的目录
```

(c) cp 命令

```
cp a.txt b.txt          # 复制 a.txt 为 b.txt
cp -r dir1 dir2         # 复制整个目录 dir1 到 dir2
cp -i a.txt /tmp        # 覆盖前提醒是否确认
```

(d) grep 命令

```
grep "hello" a.txt      # 在 a.txt 里找包含 hello 的行
grep -i "error" log     # 不区分大小写查找 error
grep -n "test" a.txt    # 显示匹配内容所在行号
```

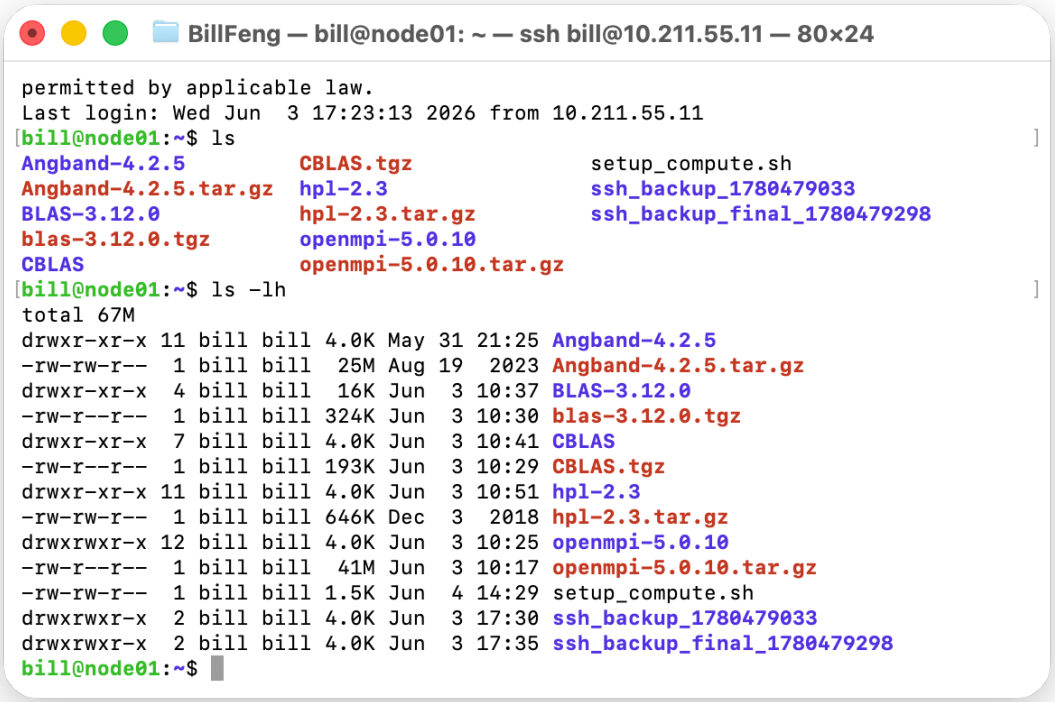


Image 1

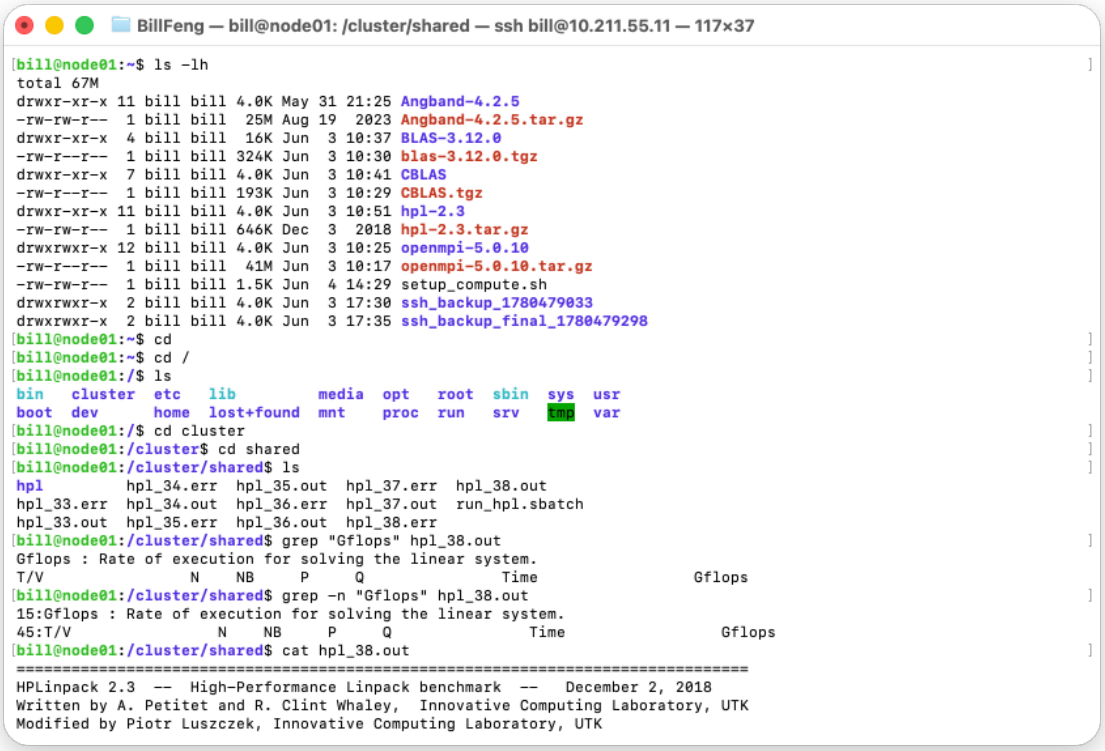
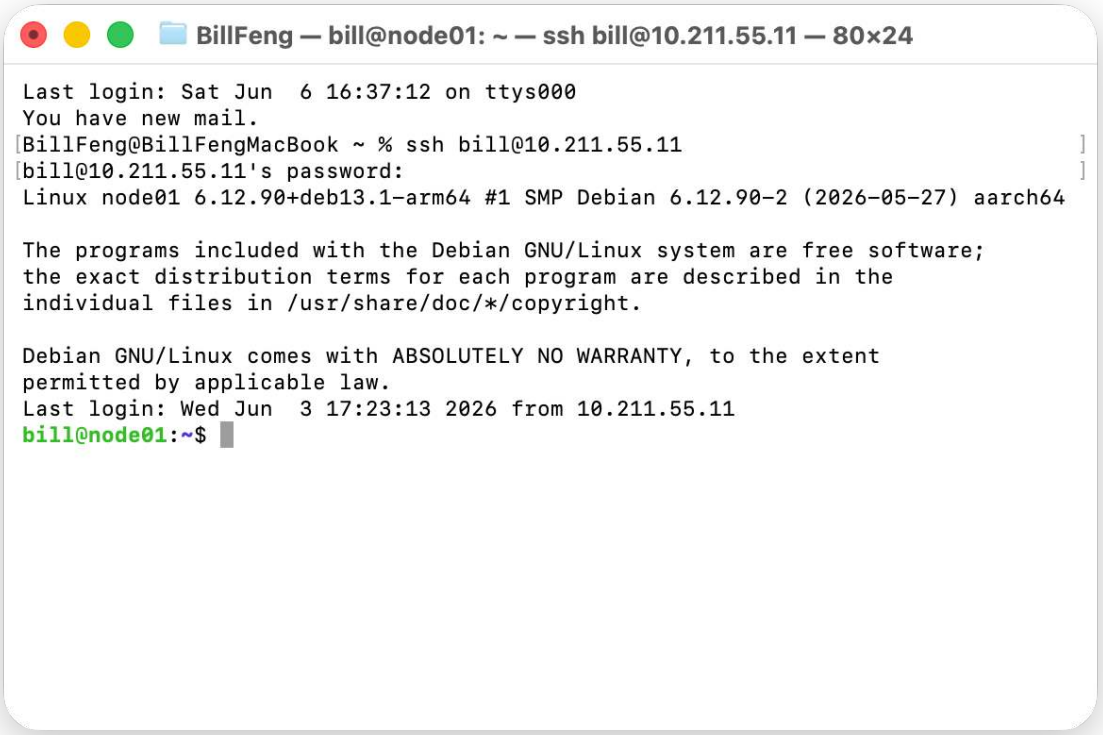


Image 2

2. 在本机中通过 ssh 远程连接到你的 Linux 环境



3. 完成 SadServers 上的题目 "Saint John": what is writing to this log file? 给出解答以及通过截图。

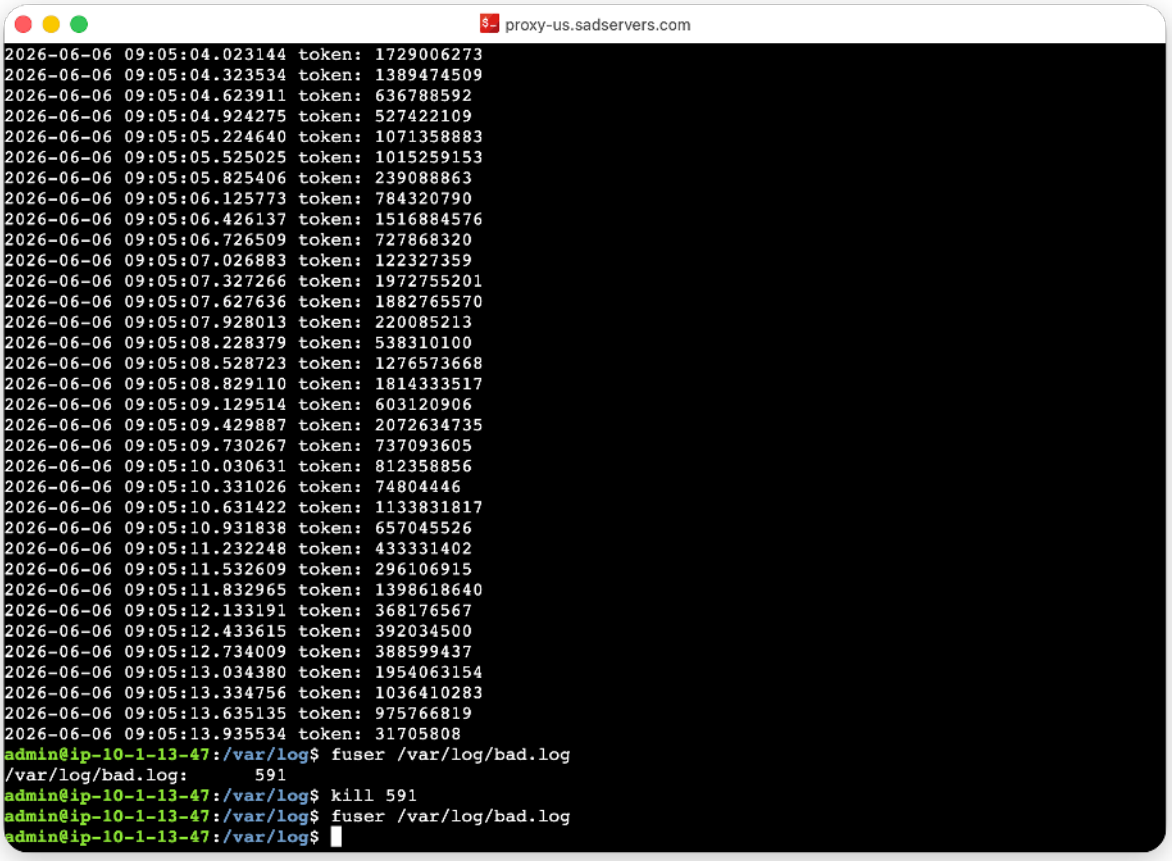


Image 3

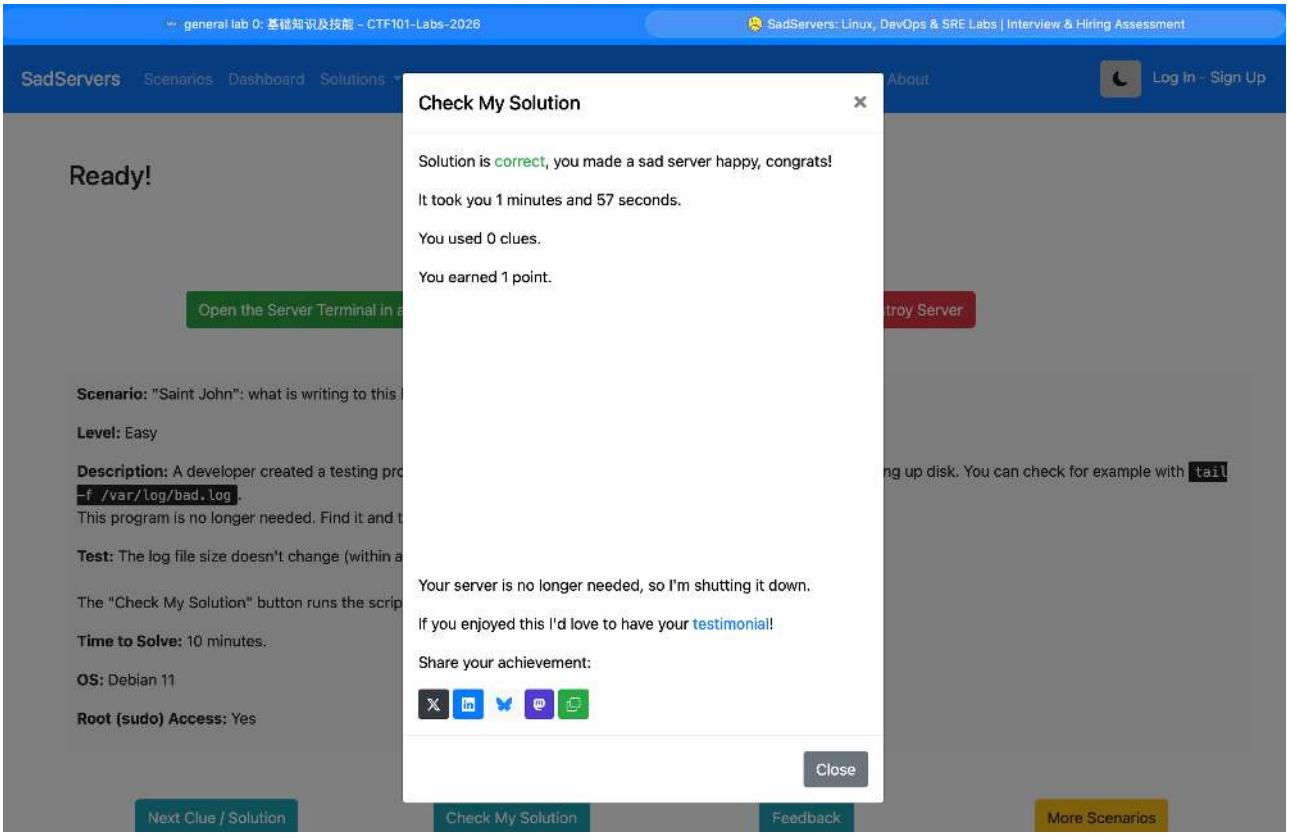


Image 4

1.2 Challenge 2：基础的Python编程

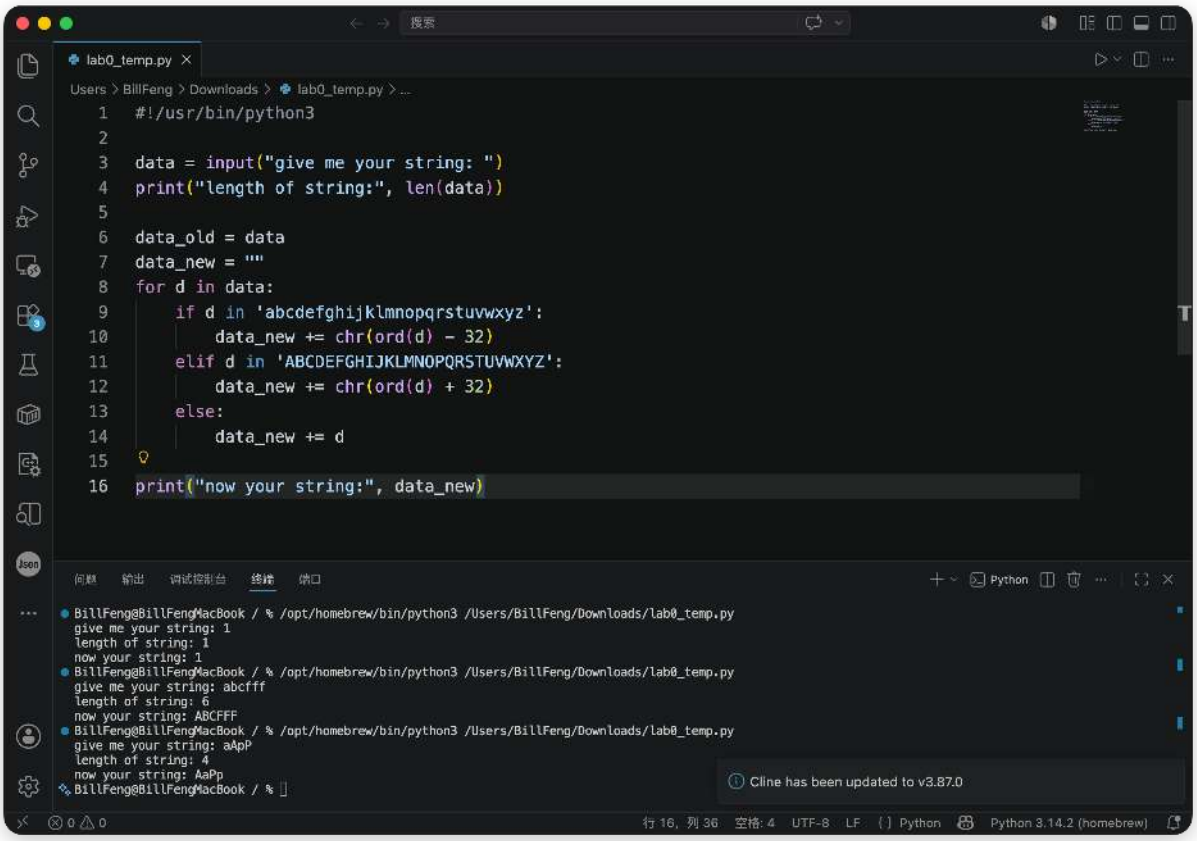


Image 5

- 实现了字符串长度测量和大小写替换。

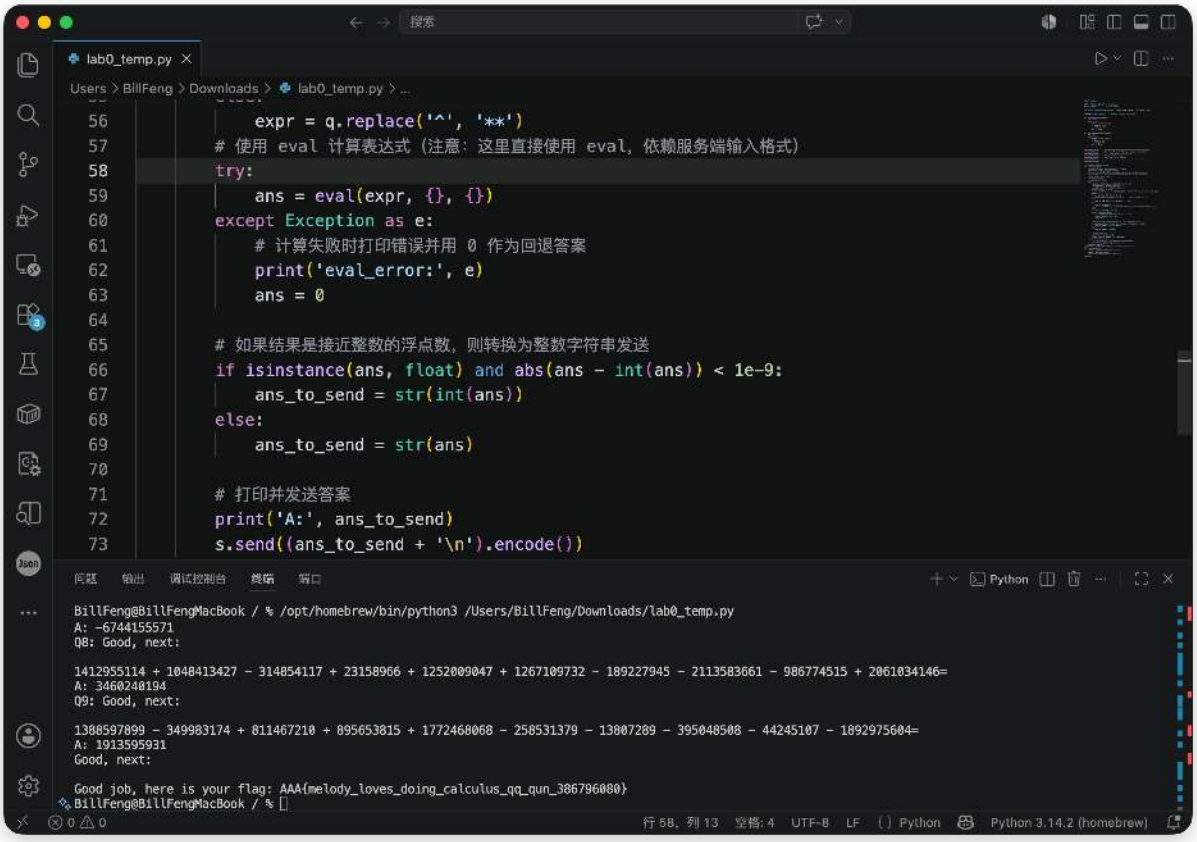


Image 6

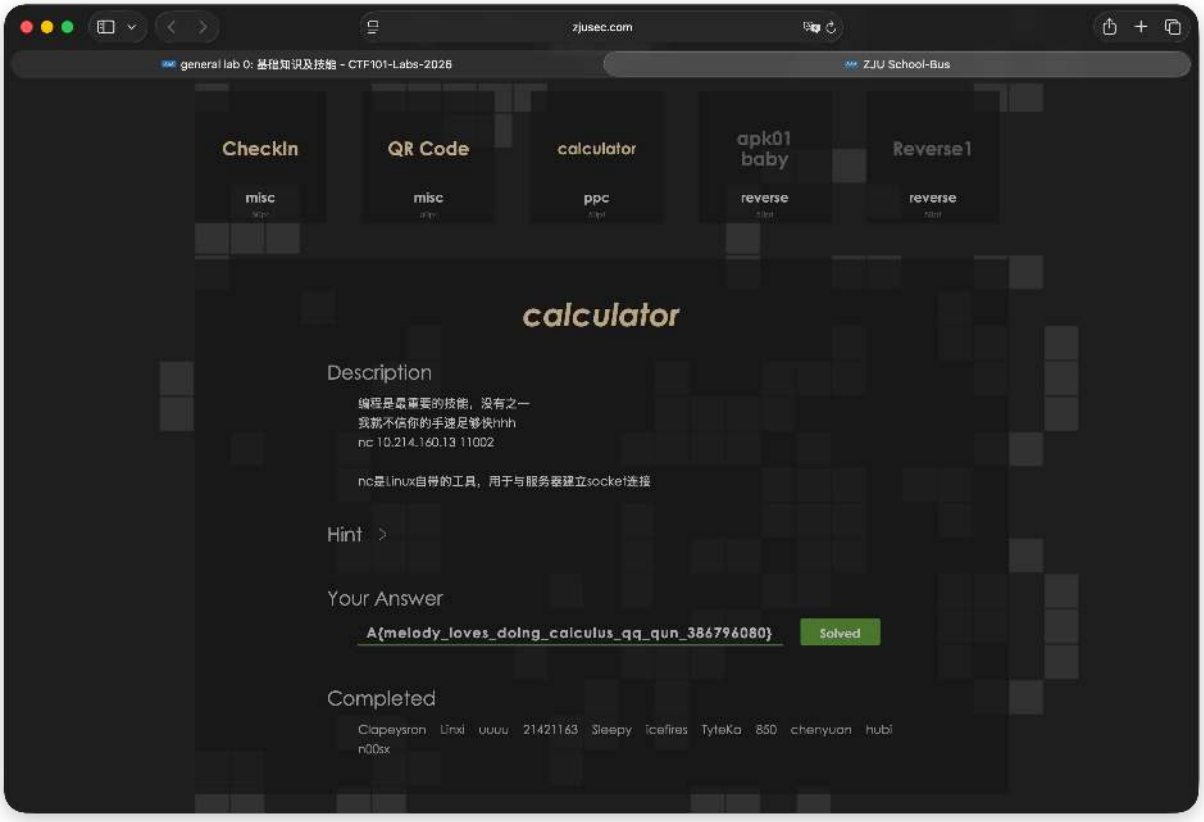


Image 7

1.3 Challenge 3

- asm_tour: `ACTF{We1com3_7o_R3_00_00_0000_0e74_0e4f_030f}`

2. Web

大概只做了第一题

2.1 Challenge 1

首先用浏览器打开网页，我发现按钮会在鼠标移动到对应位置时消失。通过控制台可以看出有一个 `getflag()` 函数，直接调用会向 `/flag.php?token=xxx` 发请求并弹窗显示 "One more time! 1 / 1337"，说明需要累计调用 1337 次才能拿到 flag。

此外每次请求需要从页面重新提取 token（token 一次性使用），且必须保持同一 Session 才能使计数器累加。每次先请求 `lab0.php` 提取 token 再请求 `flag.php`，最终在第 1337 次成功获取 flag: **`flag{56297ad00e70449a16700a77bf24b071}`**。

- ai搓出来的脚本...

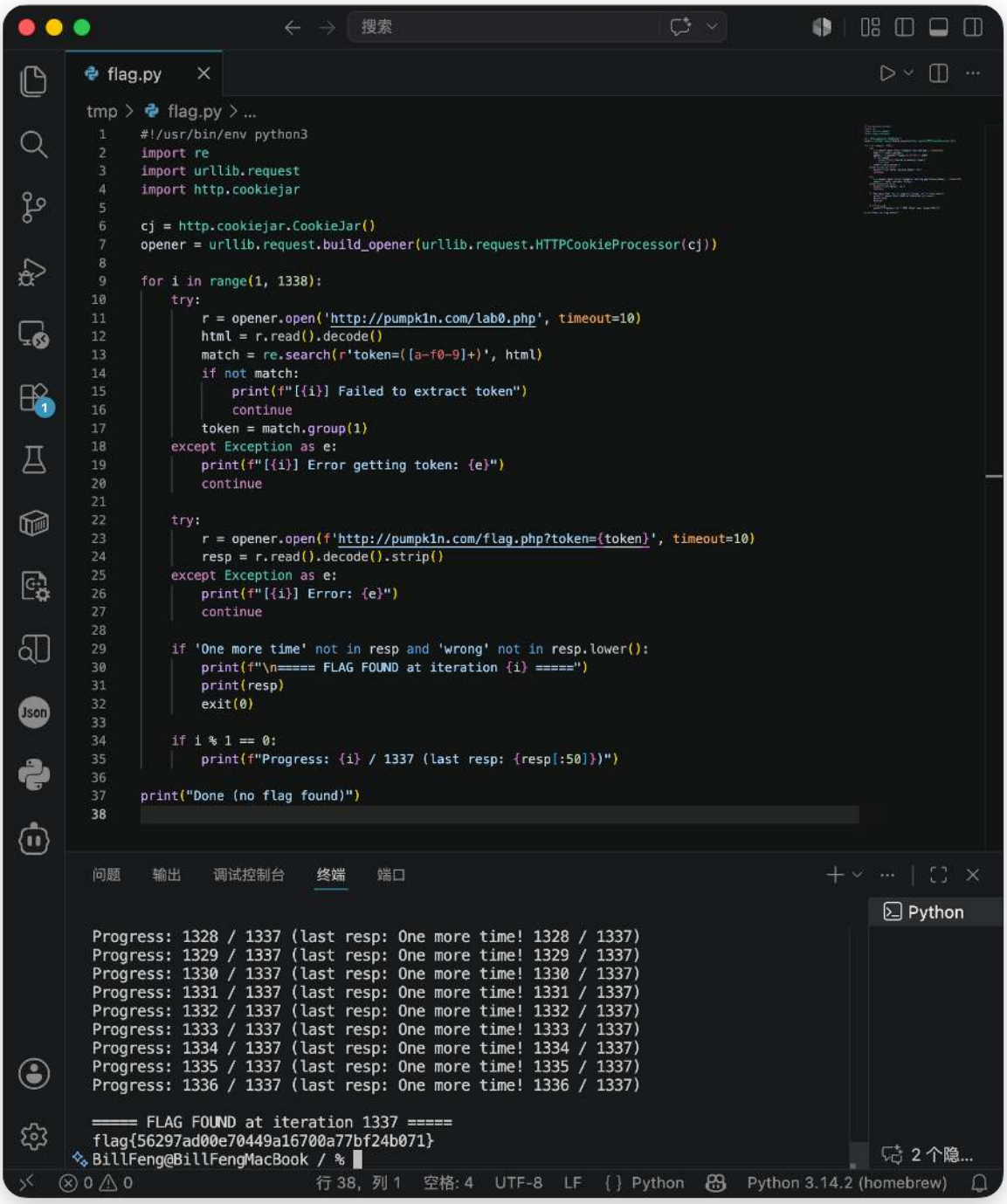


Image 8

3. Reverse

本学期学习了80x86汇编，然而打开ida的时候还是眼花缭乱，学习了一些快捷键后在`string`窗口找到了`Access Granted`。随后定位到此处并生成反汇编代码。通过规则判断出flag，随后在Linux环境中测试flag的正确性。

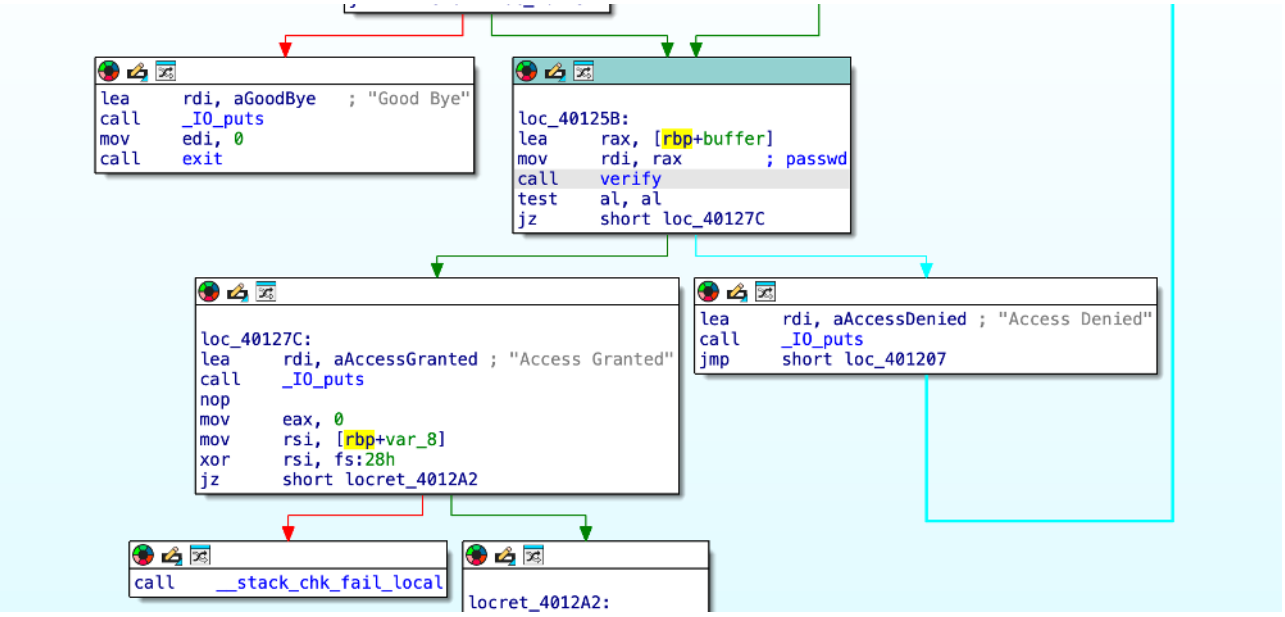


Image 9

```
1  int __fastcall main(int argc, const char **argv, const char **envp)
2  {
3    char buffer[264]; // [rsp+10h] [rbp-110h] BYREF
4    unsigned __int64 v5; // [rsp+118h] [rbp-8h]
5
6    v5 = __readfsqword(0x28u);
7    memset(buffer, 0, 0x100u);
8    while ( 1 )
9    {
10     banner();
11     IO_gets(buffer);
12     if ( j_strlen(buffer) == 1 && buffer[0] == 113 )
13     {
14       IO_puts("Good Bye");
15       exit(0);
16     }
17     if ( !verify(buffer) )
18       break;
19     IO_puts("Access Denied");
20   }
21   IO_puts("Access Granted");
22   return 0;
23 }
```

Image 10

```
bool __cdecl verify(char *passwd)
{
  int v2; // ecx
  int v3; // r8d
  int v4; // r9d
  char v5; // [rsp+0h] [rbp-F0h]
  int i; // [rsp+18h] [rbp-D8h]
  char *table[14]; // [rsp+20h] [rbp-D0h]
  char tmp[64]; // [rsp+90h] [rbp-60h] BYREF
  unsigned __int64 v9; // [rsp+D8h] [rbp-18h]

  v9 = __readfsqword(0x28u);
  table[0] = "1040";
  table[1] = "1040";
  table[2] = "1040";
  table[3] = "1968";
  table[4] = "1152";
  table[5] = "1680";
  table[6] = "1312";
  table[7] = "1616";
  table[8] = "1888";
  table[9] = "1616";
  table[10] = "1824";
  table[11] = "1840";
  table[12] = "1616";
  table[13] = "2000";
  if ( j_strlen(passwd) != 14 )
    return 1;
  memset(tmp, 0, sizeof(tmp));
  for ( i = 0; i < (unsigned __int64)j_strlen(passwd); ++i )
  {
    sprintf((unsigned int)tmp, (unsigned int)"%d", 16 * passwd[i], v2, v3, v4, v5);
    if ( (unsigned int)j_strcmp(tmp, table[i]) )
      return 1;
  }
  return 0;
}
```

Image 11

```
parallels@ubuntu-linux-2404:~$ '/media/psf/homework/CTF/lab0/crackme'
Enter Password (or q to quit): AAA{HiReverse}
Access Granted
parallels@ubuntu-linux-2404:~$
```

Image 12

4. Misc

4.1 Challenge 1

- Magic!自动解决了。

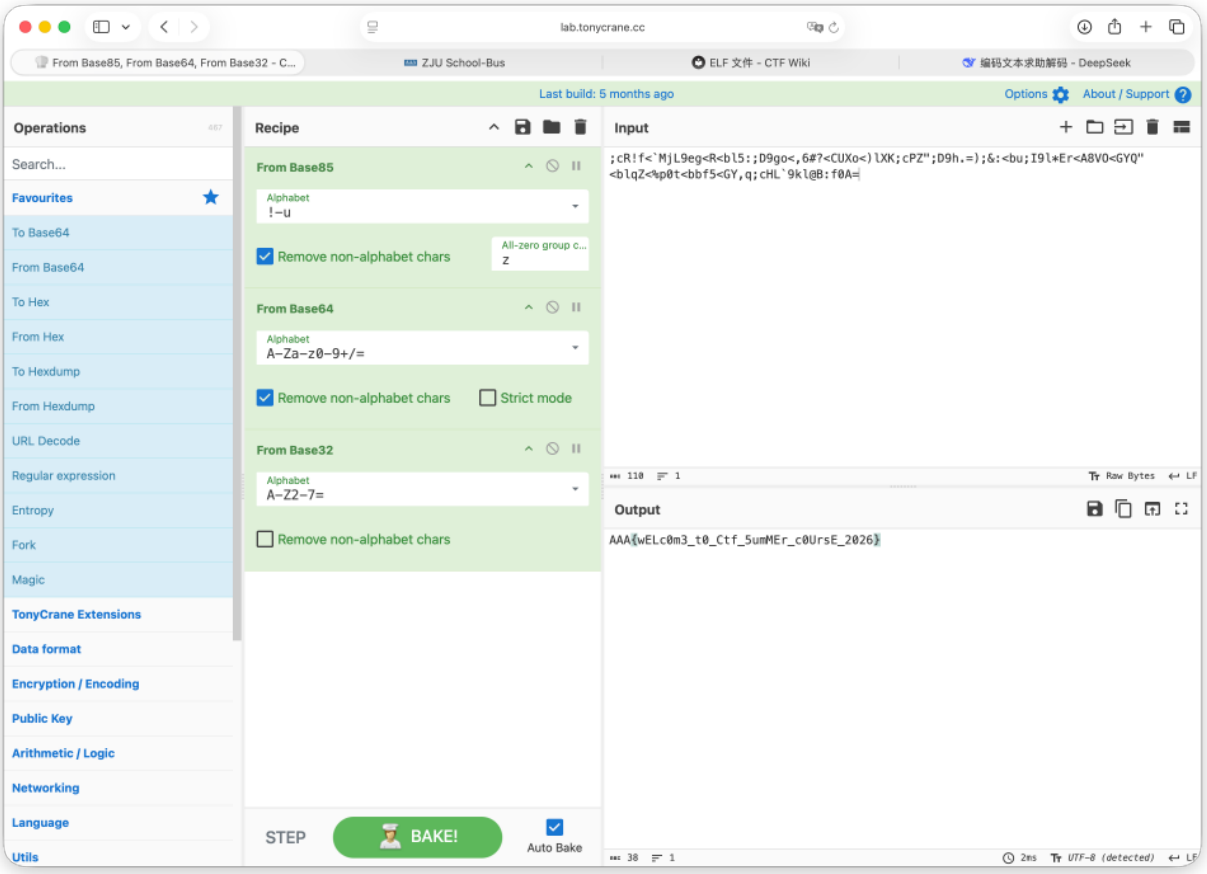


Image 13

4.2 Challenge 2

看来是肉眼难以发现的色差。将整体二值化，背景变黑其余变白即可找到flag的前半段。

2. 关键是：这不是传统的 LSB 隐写！

那些 LSB 变化是因为白色文字经过抗锯齿渲染后，边缘像素的 RGB 值发生了微小变化，导致 LSB 被动改变 —— 而不是故意在 LSB 中编码数据。文字本身就是信息载体。

3. 提取方法：阈值化 + OCR

```
# 把背景色替换为黑，其他像素替换为白，做成高对比度图
new_pixels = []
for p in pixels:
    if (p[0], p[1], p[2]) == (50, 119, 194):  # 背景色
        new_pixels.append((0, 0, 0, 255))      # 变黑色
    else:
        new_pixels.append((255, 255, 255, 255)) # 变白色（文字）
out = Image.new('RGBA', (w, h))
out.putdata(new_pixels)
out.save('text_revealed.png')
```

4. OCR 识别

```
tesseract text_clean.png stdout --psm 3
# 输出：AAA{gr3@t_JQ8!_let'5_
```

核心思路就是：这不是 LSB 比特编码，而是视觉上的文字隐写 —— 文字颜色和背景非常接近（肉眼难辨），但通过阈值化分离后就清晰可见了。

Image 14

Image 15

`AAA{gr3@t_J08!_1et'5_`

后半段比较简单，直接查看十六进制(`hex`)即可。

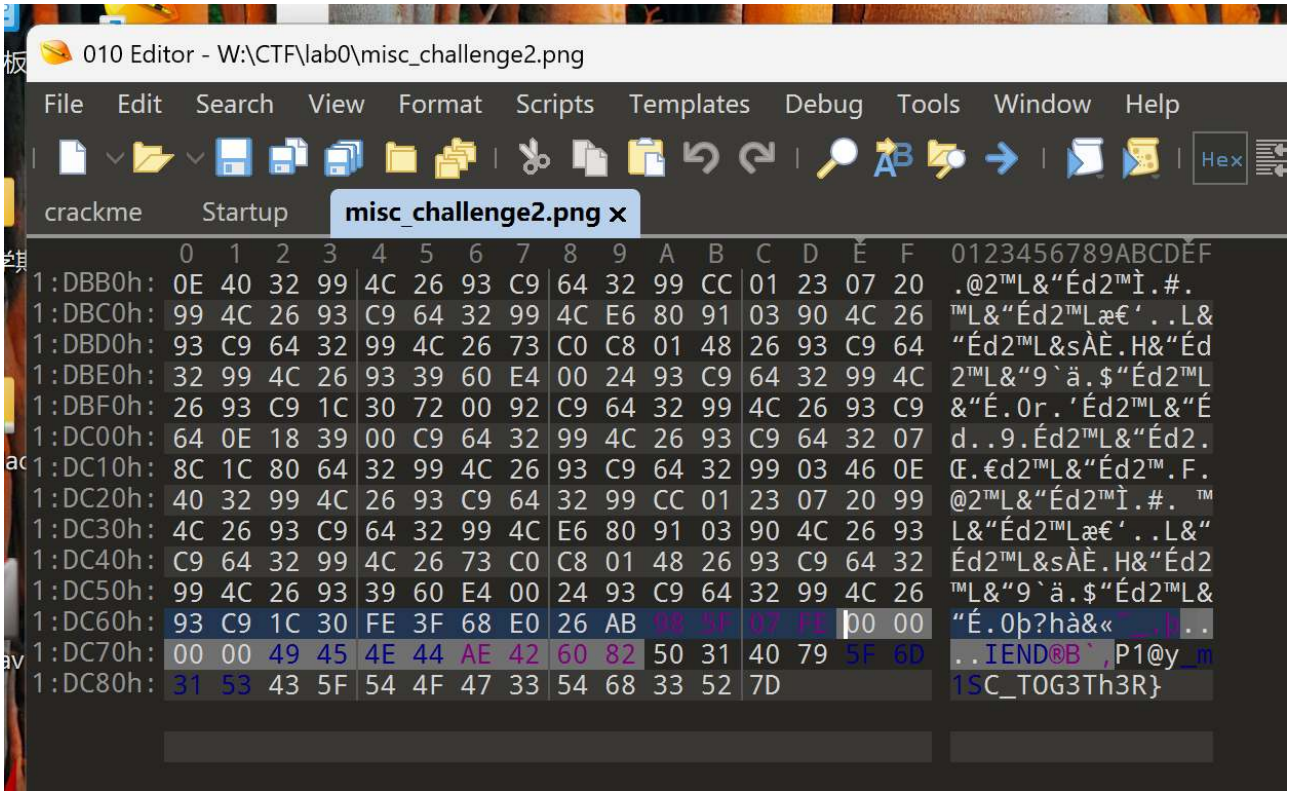


Image 16

`P1@y_m1SC_TOG3Th3R}`

连起来就是`AAA{gr3@t_J08!_1et'5_P1@y_m1SC_TOG3Th3R}`