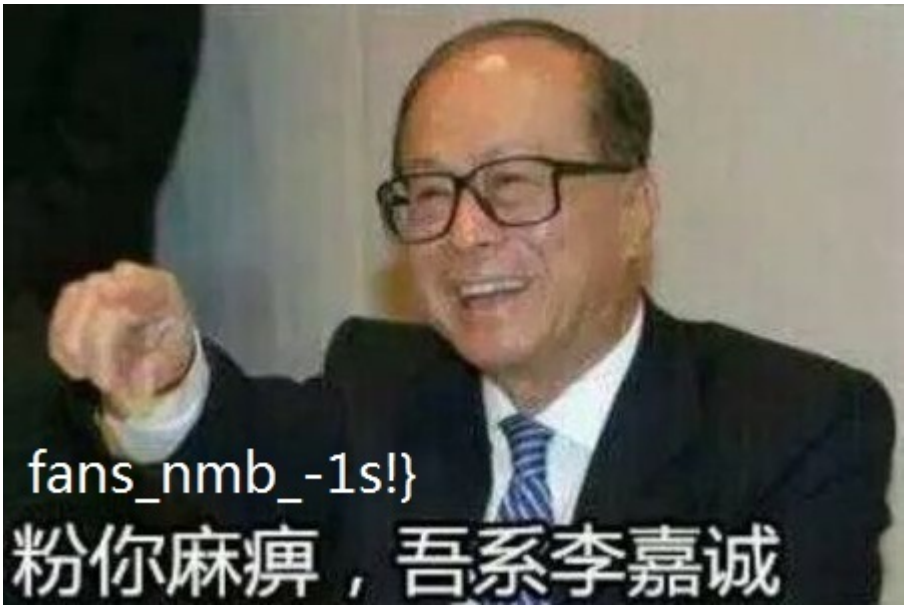
Misc Lab2 Report

Challenge 1

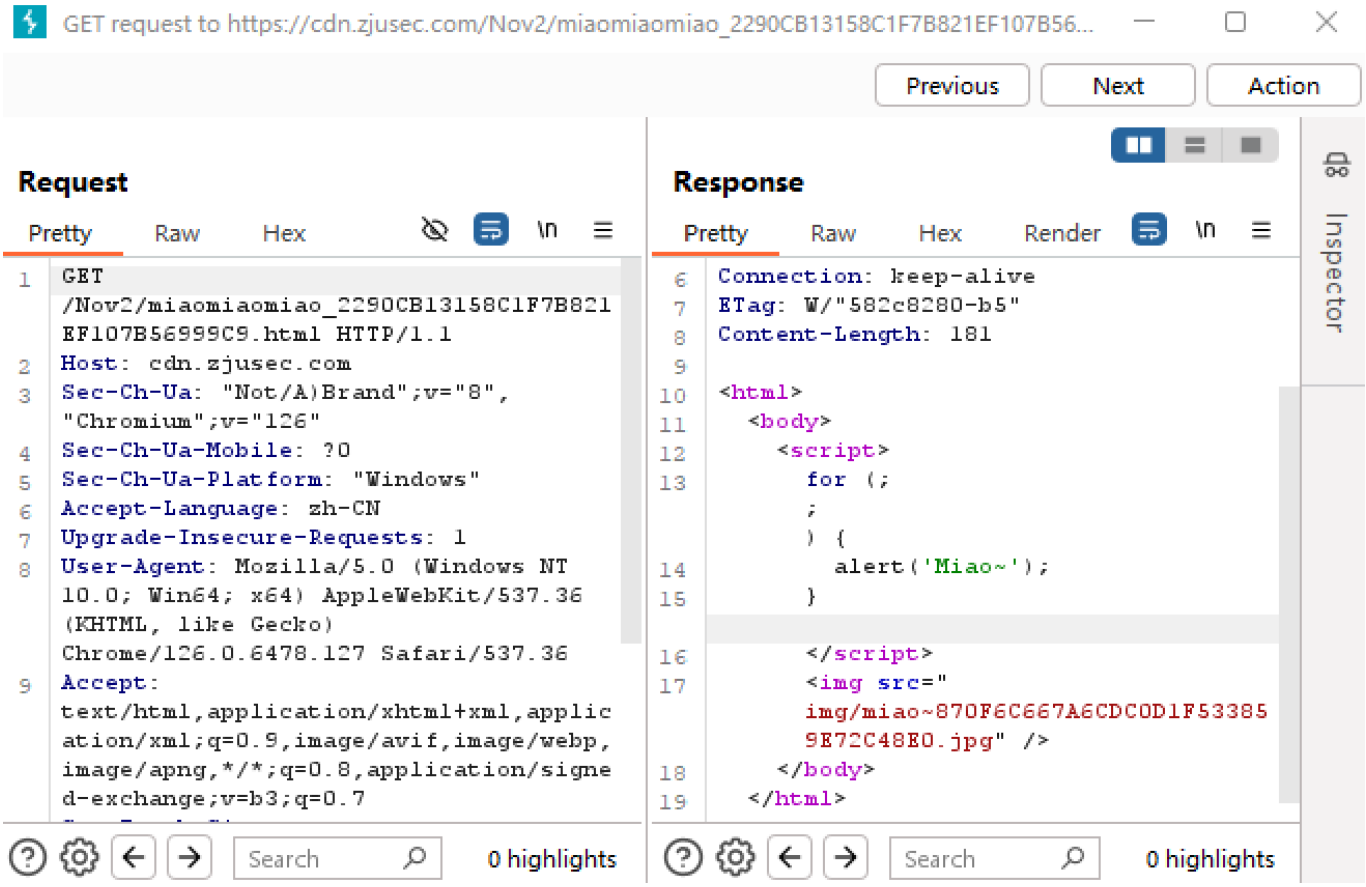
将原图放到各类软件中尝试，最终在foremost里面成功分离出两张图片，即可得到flag



AAA{the_true_fans_fans_nmb_-1s!}

Challenge 2

本题点进链接发现是一个html链接，但点击之后似乎没反应。所以先通过burp抓包，发现其中还是有一个图片的，之后将图片的名称直接输入url当中，即可获得图片。



而后将图片拖入010editor中查看，发现一个密码key:m1a0@888，所以需要某工具将其解密 将其和密码放入

steghide工具中解密，发现一串二进制编码，按照八位一组编码，并转换成ASCII码即可得到flag。

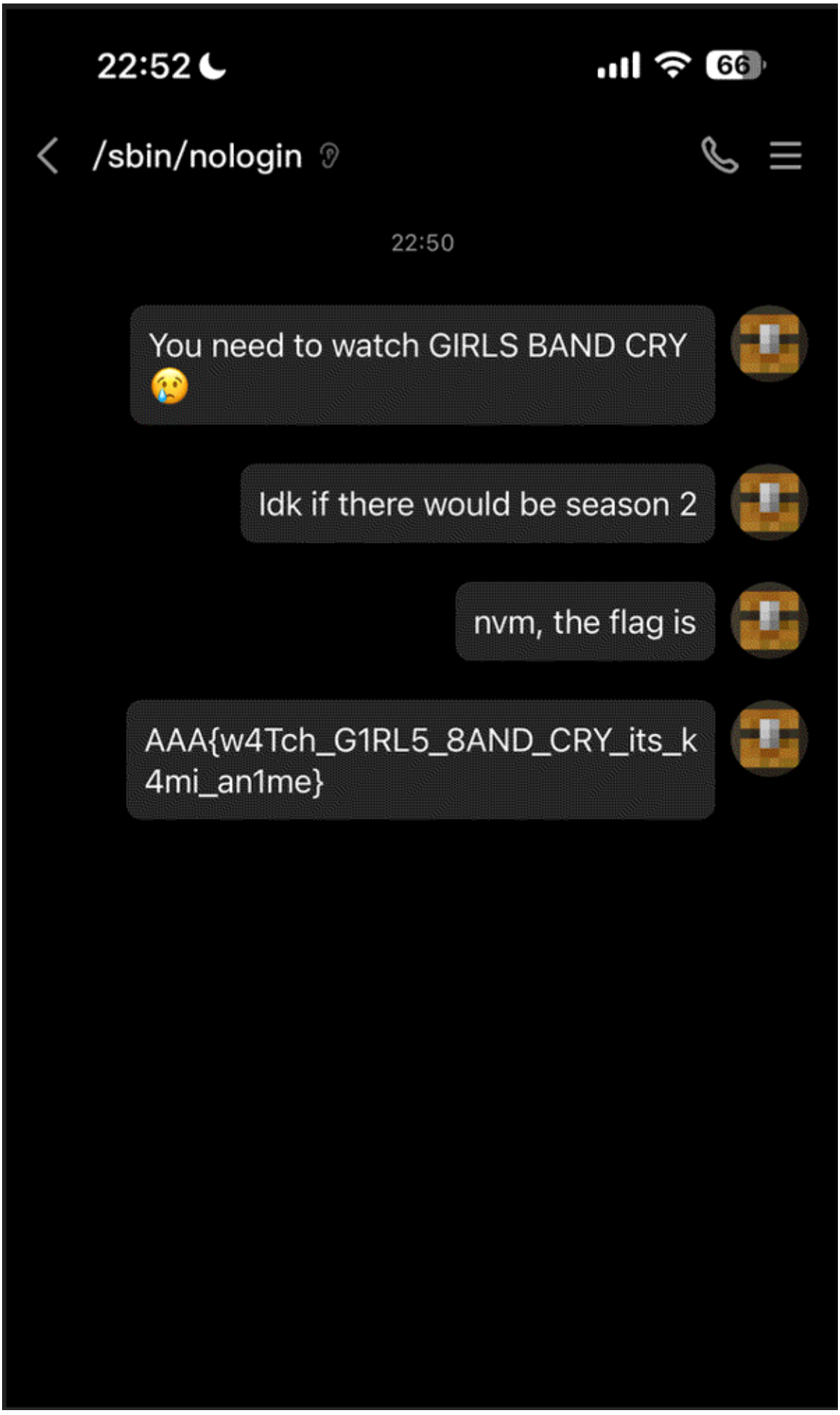
```
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/misc_lab2/miao# steghide extract -sf miao.jpg -p m1a0@888
wrote extracted data to "secret_file.txt".
```

AAA{D0_Y0u_L1ke_Ste9H1de_M1a0}

Challenge 3

首先将图片放到steg中查看，发现在绿色图层时有点诡异（bushi，然后在dataextract里选中绿色的0和LSB，发现文件头竟然变成了PNG，之后将生成的文件保存为png，发现如图

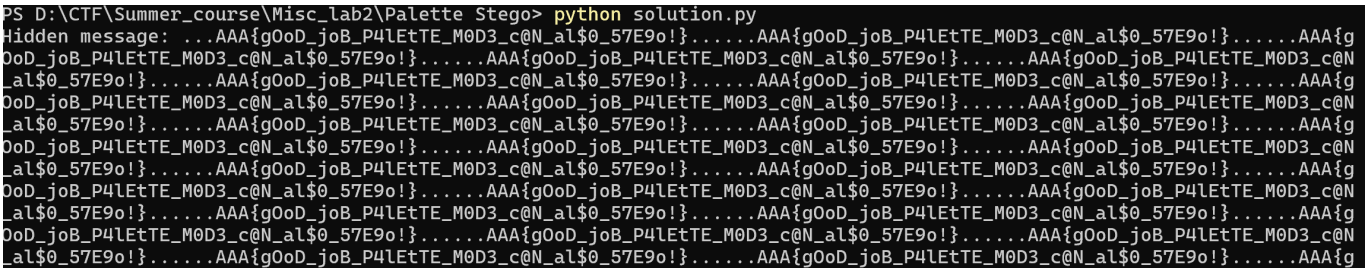
显然，需要我们更改图片的高度，在010editor中修改即可得到下面的flag


22:52
66
/sbin/nologin
22:50
You need to watch GIRLS BAND CRY 😢
Idk if there would be season 2
nvm, the flag is
AAA{w4Tch_G1RL5_8AND_CRY_its_k4mi_an1me}

AAA{w4Tch_G1RL5_8AND_CRY_its_k4mi_an1me}

Challenge A

PLTE chunk主要用于索引颜色图像,定义了图像中使用的所有颜色。 EZStego是一种基于调色板的隐写方法，通常将秘密信息嵌入图像的最低有效位（LSB） 所以我们需要编写一个python程序，获取图片的每一个像素值，同时获取该像素对应的调色板颜色值的最低有效位。最后再将得到的这串二进制数转换为ASCII码可见字符就能得到最终答案。当然需要注意获取每一个像素时需交换x和y的顺序。 代码见附件。

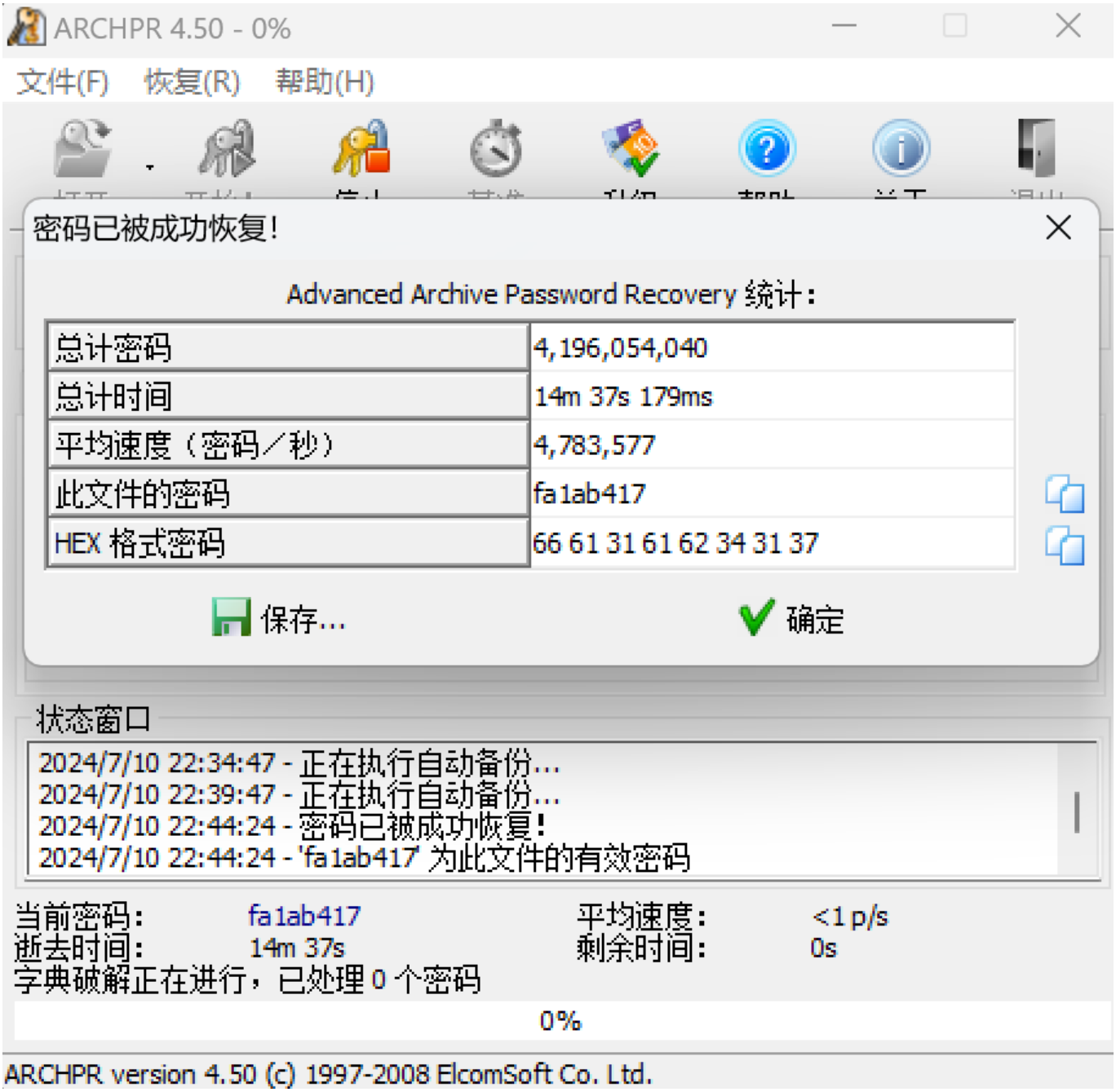


```
PS D:\CTF\Summer_course\Misc_lab2\Palette Stego> python solution.py
Hidden message: ...AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
OoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N
_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
OoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N
_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
OoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N
_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
OoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N
_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
OoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N
_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}......AAA{g
```

AAA{gOoD_joB_P4lEtTE_M0D3_c@N_al$0_57E9o!}

Challenge C

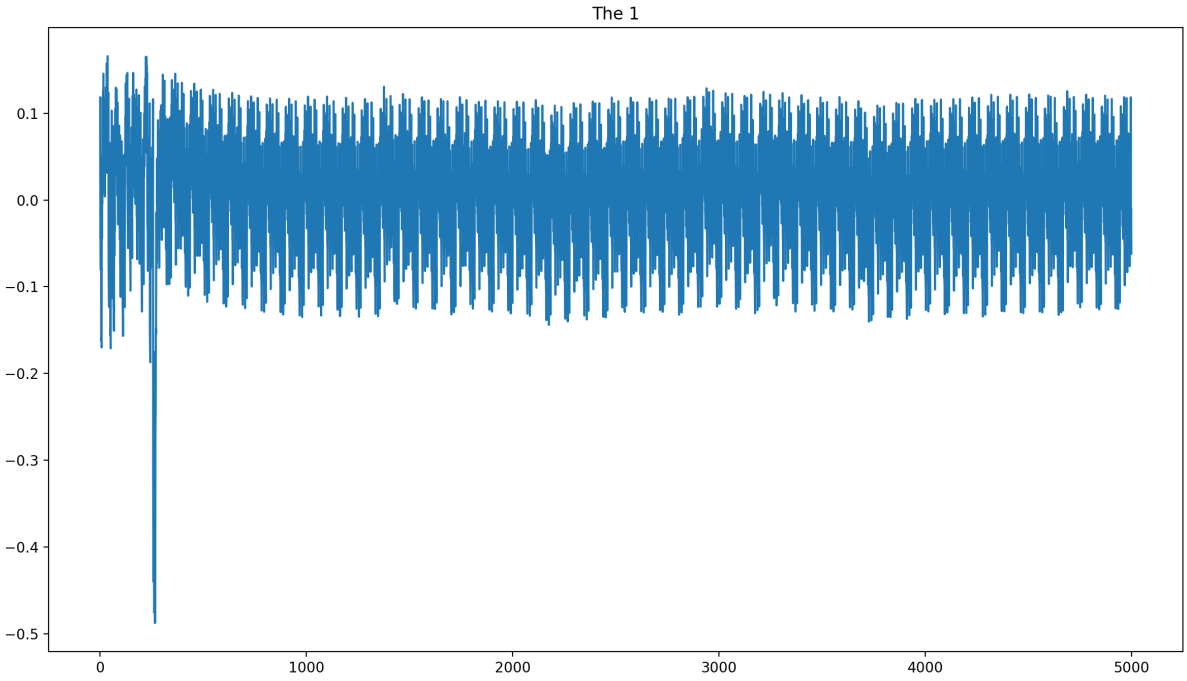
本题首先给了一个图片文件，用binwalk指令进行分离，得到一个压缩文件，发现是加密的，但这个显然是伪加密。将文件拖入010editor中，修改加密符号位，则文件就可以无需密码解压。之后在对下一个压缩文件进行上述操作，发现又可以得到一个文件，且文件的开头名字减一。所以我们需要编写脚本使上述过程重复一千次，才能得到最终的答案。 但重复一千次之后却又得到一个压缩包，文件名显示该压缩包的密码为八位16进制数。联想到前面999个文件的文件名中都含有16进制数，但是在想不出这些数之间有什么联系，于是只能爆破（），显然python并不适合爆破，所以先用c生成字典，再用软件进行爆破，耗时在可接受范围之内 压缩包密码为fa1ab417



ZJUCTF{R4w_Un3NcRyP73d_&_RaNd0MiZ3_U53I35SLy}

Challenge D

首先将给定的文件解压，得到四个文件，index0-12说明密码总共13位，input则为字符的合集，output为空，而trace则存放了大量数据，即为攻击时功率的变化。 所以，我们就可以使用python编写脚本呈现出13个密码爆破时的功率图像。图中横轴为尝试字母顺序，纵轴为功耗。



需要注意的是，trace文件中一行有520个数据，而520=13*40,13为密码长度，40则为所尝试的字符个数（a-z，0-9，以及四个特殊字符）。显然，当尝试的字符恰好为该位所对应的密码的字符时，图中的功率会变得最小，因此我们只需编写程序找寻功率最小值时所对应的字符即可。

```
PS D:\CTF\Summer_course\Misc_lab2\PowerTrajectoryDiagram\attachment> python3 analyze.py
_
c
i
s
c
n
_
2
0
2
4
_
a
_ciscn_2024_a
```

最后的flag为AAA{_ciscn_2024_a} 但提交了发现不对，所以尝试把最后一个看起来没什么意义的a去掉就对了。。。