《Python 程序设计高阶》

(2023-2024 学年第 1 学期)

作业报告

学号:2022337621188 姓名: 徐赫 班级: 计算机全英班
学号:2022337621030 姓名: 杨子豫 班级: 计科（1）班
学号:2022337621189 姓名: 徐歆诚 班级: 计科（3）班

§1 选题：路由器固件分析

路由器的固件本质上是一个嵌入式 Linux 系统，把整个 Linux 文件系统和路由器的应用程序全部封装在一个 rom 文件中，在路由器上电时，将这个文件在内存解开，加载各类服务，从而实现路由器的功能。下图就是用固件升级路由器的截图。路由器的配置有时候还是比较复杂的，很多时候，路由器一旦忘记管理员密码，那么路由器的管理界面就没办法访问了。有人说，那直接捅一下 Reset 孔，恢复出厂设置就行。但恢复了，配置都没了，有没有其他办法呢? 办法还是有的，但并不一定很方便。找一个热风机，把 Flash 芯片吹下来，然后，用一个编程器，把 Flash 芯片里的数据全部读到一个 Rom 文件里，然后，再分析这个 Rom 文件，找到里面的 password 文件，把对应用户名的密码 hash 改了，比方说改成 123456 的 hash 值，再把改过的 Rom 文件，用编程器烧写回去。怕写坏的话，再买片一样的。然后再焊上去，大功告成。这里面最关键的，就是能够解析固件的文件。我给你一个路由器，请你用 Python 写一个分析器，按上述操作实践一下。希望你能够成功。

§2 解答

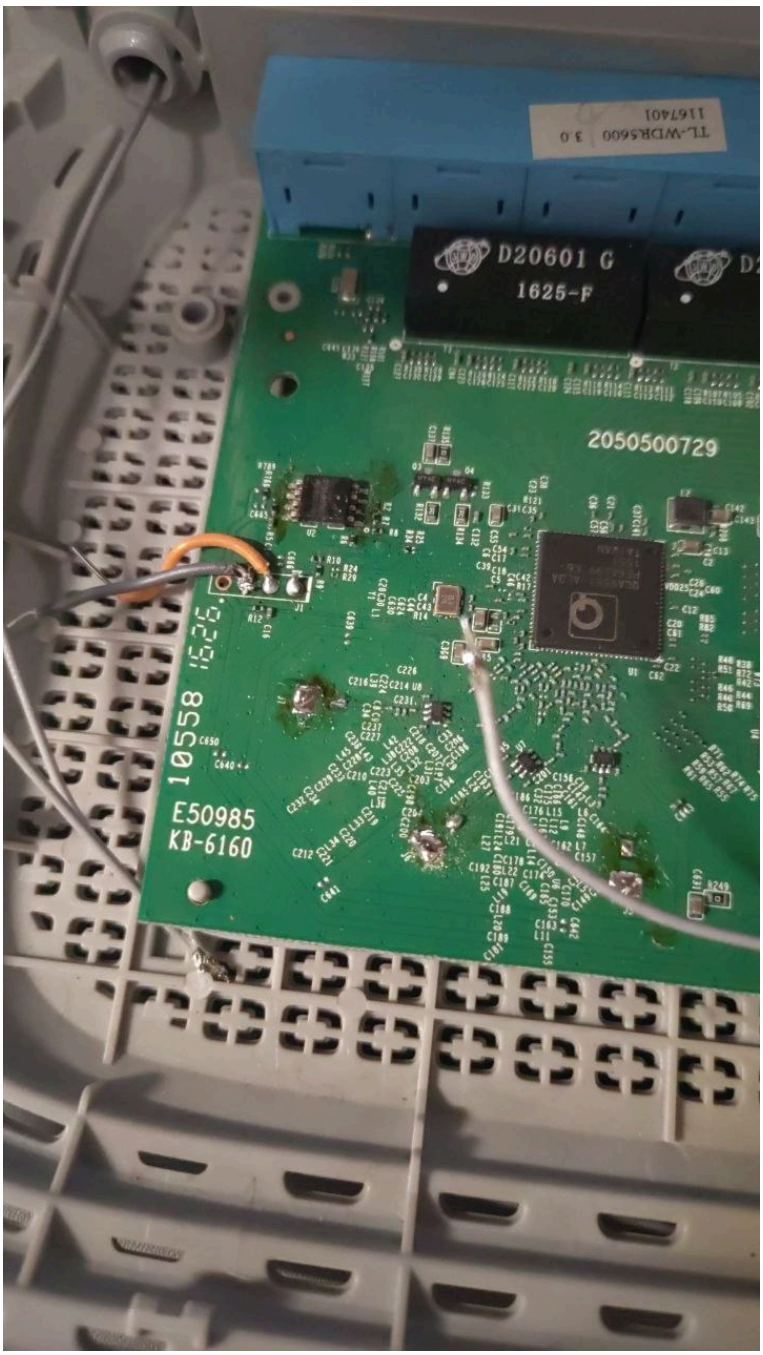
§2.1 硬件上的尝试

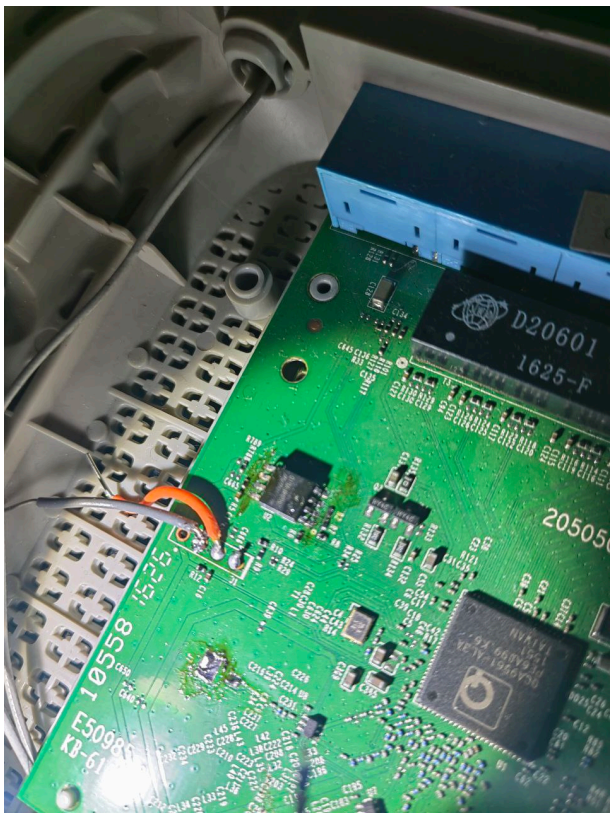
尝试从路由器的 FLASH 芯片中提取文件可以使用焊一个 UART 上去, 也可以使用架子架上去刷

这里我们先测试了使用架子配合 UBOOT, 但是, 当想要更新新的 UBOOT 的时候, 使用 neoprogrammer 的时候出现了写错偏移值的情况

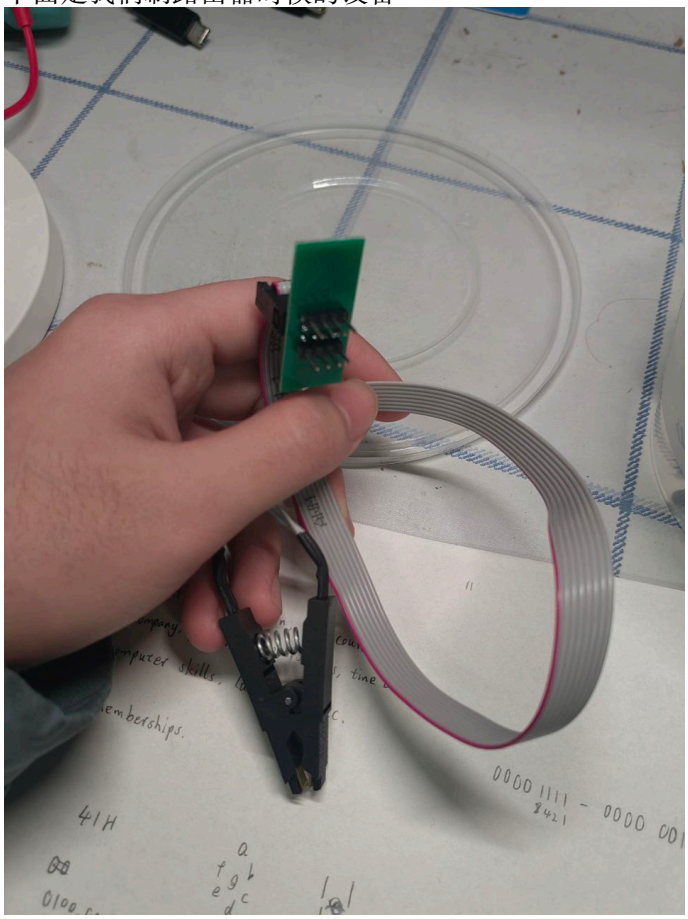
导致少了一个 0, 覆盖了不应该覆盖的内容

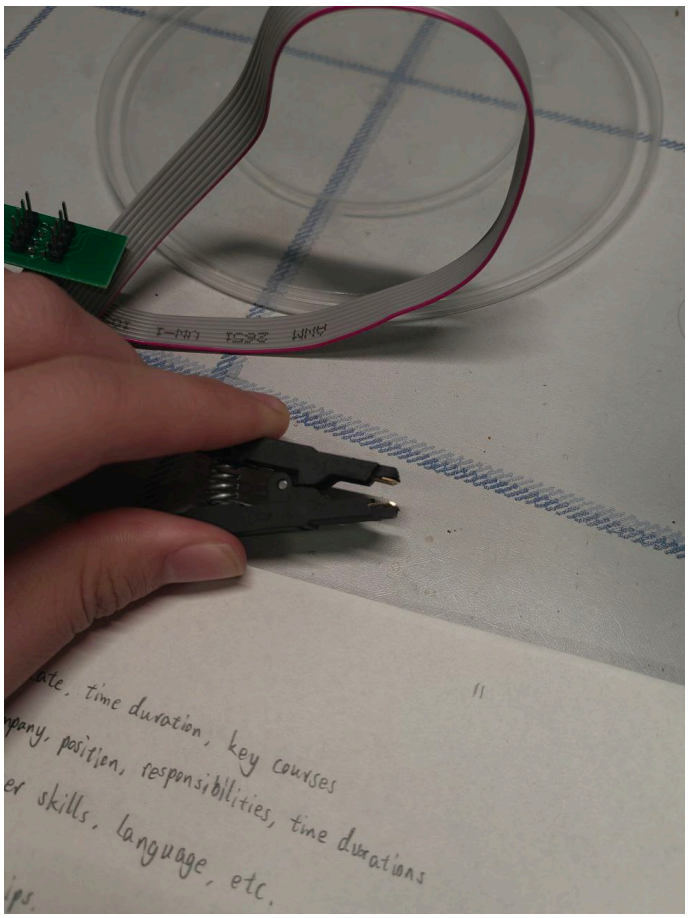
以下是路由器状态


TL-WDR5600 3.0
116740I
D20601 G
1625-F
2050500729
10558
E50985
KB-6160

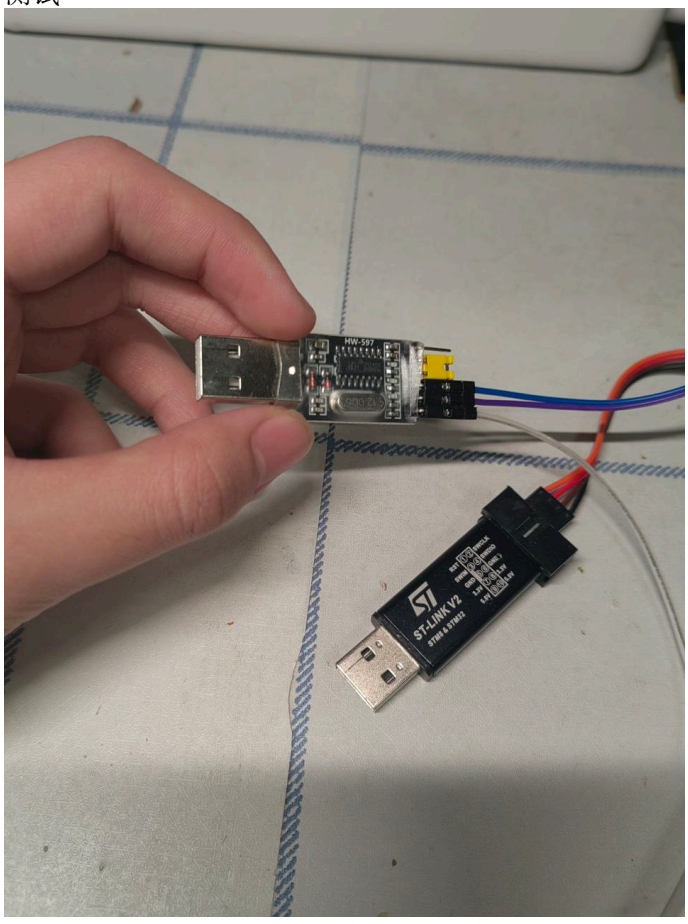


下面是我们刷路由器时候的设备





失败之后, 我们使用万用表抢救设备, 但是怀疑是边上的电路存在干扰, 我们打算使用 UART 焊上去进行测试



不幸的是, 在我们偏移量写错之后不知道刷坏了什么内容, 这个 FLASH 不能用了, 我有 STM32 用的 FLASH 但是有引脚不能正常焊上去

所以, 我们打算直接使用本来我刷进去的固件, 直接去解析 OPENWRT 的固件内容, 我们打算直接解析 squashfs 的内容, 但是我们使用 PySquashfsImage 库

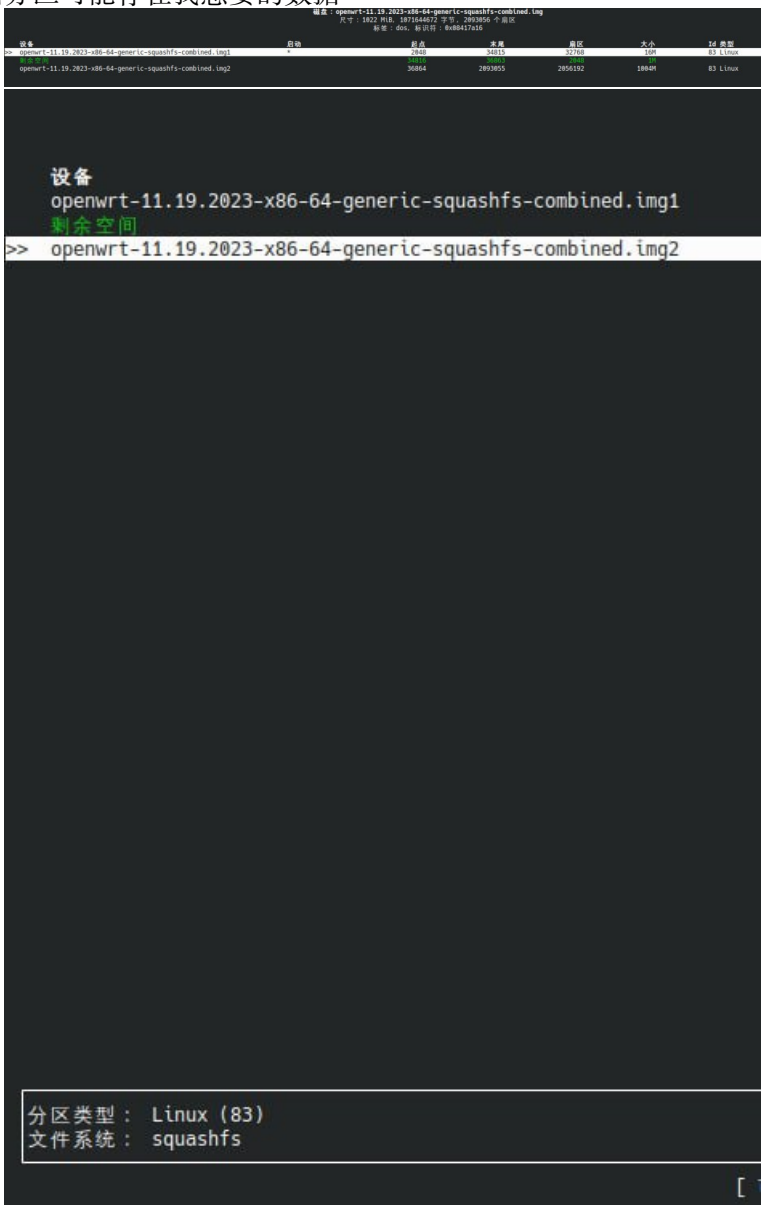
```
from PySquashfsImage import SquashFsImage

image = SquashFsImage.from_file('./openwrt-11.19.2023-x86-64-generic-squashfs-combined
for item in image:
    print(item.name)
image.close()
```

然后他发出报错, 显示这不是一个 squashfs 4.0 的文件

为了证明结果, 我们又使用了 squashfs-tools 的工具, 在我们自己的 debian 服务器上进行操作

使用了 file 命令查看文件信息, 检查之后发现是一个整盘的镜像, 对于这种镜像, 前面作为开机程序, 后面分区可能存在我想要的数据

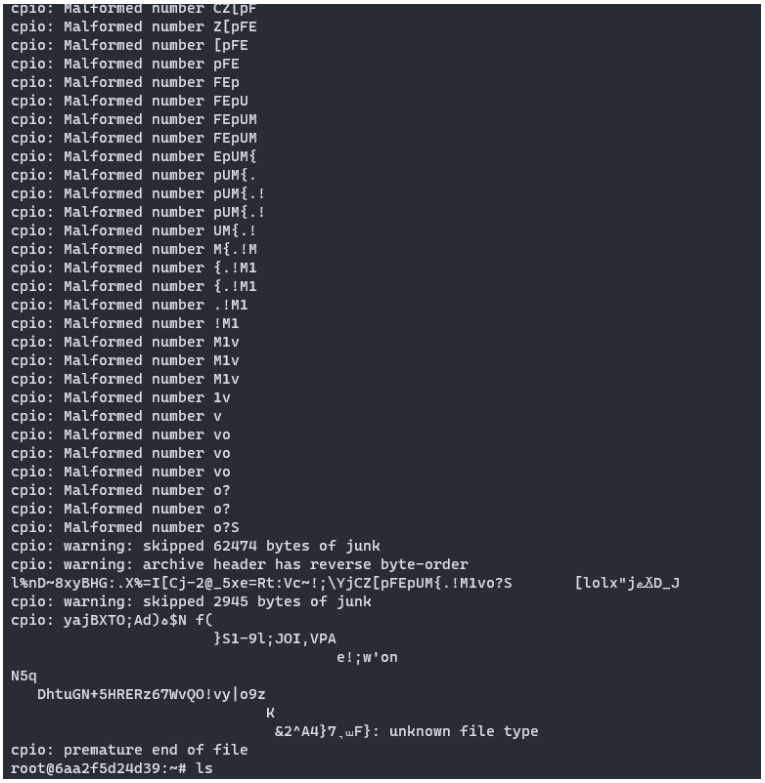


发现最后一个分区才是它的 squashfs 文件, 那我们需要使用一系列的方式, 比如 dd 把它从那个位置截断拿出来, 对于我们这种小白来说, 很难实现, 所以我们选择了一些直接解析一整个镜像的方式, 使用例

如 gpio 的工具的方式实现

使用 cpio 工具实现解包 rootfs.img

cpio -ivmd < rootfs.img



出错了

好吧, 最后我们准备使用 binwalk 解析文件, 在这之前让我们先看下 MBR 分区, 说不定能截断出需要的内容

结果, 失败

我们尝试使用 `Python` 的 `struct` 库来读取硬盘的主引导记录 (MBR) 并解析分区表，以获取扩展分区和逻辑分区的信息。我们分析每个分区表项的结构，以获取有关扩展分区和逻辑分区的信息，包括起始扇区和扇区数。特别关注了扩展分区的类型代码为 0x05 或 0x0F，以便正确识别扩展分区的存在。

§2.2 软件上的尝试

在软件层面上，我们尝试使用 `Python` 的文件操作方法来读取硬盘设备的数据，并通过第三方库进行特定类型文件的处理，比如 `squashfs` 文件的解压缩。通过这些尝试，成功地对硬盘数据进行了解析和特定类型文件的处理，从而获得了有关分区结构和存储内容的深入信息。

接下来我们要使用 binwalk 库来实现解析固件文件了:

binwalk 会分析二进制文件中可能的固件头或者文件系统，然后输出识别出的每个部分以及对应的偏移量，-e 参数，自动把固件镜像中的所有文件都解出来。

在获取了签名扫描的结果后，我通过对 Binwalk 模块化结果的遍历和处理，提取了关键信息，深入分析固件文件的内部组成。包括文件头、数据偏移、以及嵌套结构的处理。

我们用 python 在高层级中遍历二进制文件中的所有字节，并对镜像文件中的特定结构进行解析，将偏移地址，文件头信息，文件版本号，错误校正值等信息输出。

binwalk.scan 函数调用:

binwalk.scan 是 Binwalk 库提供的函数，用于对文件进行分析。通过传递文件路径 input_file_path、signature=True（表示执行签名模块）、quiet=True（抑制正常输出）等参数，执行固件文件的签名分析。Module 对象的遍历：

得到的 module 对象代表执行的模块。通过 binwalk.scan 返回的结果，遍历每个执行的模块。

Result 对象的遍历：

对于每个模块，module.results 包含了该模块执行的结果列表。遍历 result 对象，它包含了分析结果的详细信息，如偏移量、描述等。处理结果信息的输出：

提取 result 中的文件名、偏移量等信息。通过 format_result 函数对描述信息进行格式化处理，将每一项信息独立成一行。

在调用时我们发现因为固件文件的差异性，会存在文件信息不匹配的情况。为此，我们加入了异常处理机制，以确保程序在面对各种固件文件变种和异常情况时能够保持鲁棒性。

同时我们发现处理的结果每个信息都会包含文件路径，而这占据了大量的输出空间，对我们的后续处理造成了一定的麻烦，因此我们通过 os.path.basename 方法来获取文件名，避免显示重复路径

通过实验此方法成功提取到了如下信息：

uImage header:

文件名: openwrt-11.19.2023-ramips-mt7621-linksys_e5600-squashfs-factory.bin

偏移地址: 0x00000000

描述: 匹配到 uImage 头部，表明该固件文件包含 uImage 格式的内核镜像。提供了头部大小、头部 CRC、创建时间、镜像大小、数据地址、入口点、数据 CRC 等信息。

header size: 64 bytes,

header CRC: 0x43BC676,

created: 2023-11-18 19:37:08,

image size: 3022052 bytes,

Data Address: 0x80001000,

Entry Point: 0x80001000,

data CRC: 0x66350186,

OS: Linux,

CPU: MIPS,

image type: OS Kernel Image,

compression type: none,

image name: ”MIPS OpenWrt Linux-5.15.137”,

Copyright string:

偏移地址: 0x000015BC

描述: 匹配到版权信息字符串，其中包含了版权声明及相关作者信息。

”Copyright (C) 2011 Gabor Juhos <juhosg@openwrt.org>”,

LZMA compressed data:

偏移地址: 0x0000168C 描述: 匹配到 LZMA 压缩数据，表示固件中存在经 LZMA 压缩的数据块。提供了压缩属性、字典大小和解压后大小等信息。

properties: 0x6D,

dictionary size: 8388608 bytes,

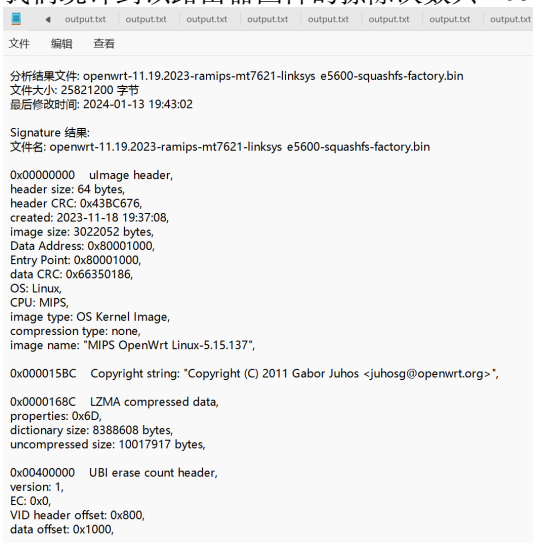
uncompressed size: 10017917 bytes,

UBI erase count header:

偏移地址: 0x00400000, 0x00420000, 0x00440000.....

描述: 匹配到 UBI（Unsorted Block Images）擦除计数头部，表明该固件文件中包含 UBI 擦除计数信息。提供了版本、EC（erase counter）值、VID header 偏移、数据偏移等信息。

我们统计到该路由器固件的擦除次数共 160 次



只展示部分代码，还有一些代码用于测试不同格式的固件文件，以及分析里面内容，还有样例代码

check_mbr.py:

```
import struct

# 打开硬盘设备
disk = open(r'./openwrt-11.19.2023-x86-64-generic-squashfs-combined.img', 'rb')

# 读取MBR
mbr = disk.read(512)

# 解析MBR中的分区表
def parse_mbr(mbr):
    # 分区表从第446字节开始，每个表项占16字节，共4个表项
    part_table = mbr[446:510]
    # 扩展分区的类型代码为0x05或0x0F
    ext_types = [0x05, 0x0F]
    # 遍历分区表，找到扩展分区
    for i in range(4):
        # 每个表项的结构为：1字节状态 + 3字节起始磁头/柱面/扇区 + 1字节类型 + 3字节结
        part_entry = part_table[i*16:(i+1)*16]
        # 获取分区类型
        part_type = part_entry[4]
        # 如果是扩展分区，返回其起始扇区和扇区数
        if part_type in ext_types:
            part_start = struct.unpack('<I', part_entry[8:12])[0]
            part_size = struct.unpack('<I', part_entry[12:16])[0]
```

```
            return part_start, part_size
    # 如果没有找到扩展分区，返回None
    return None, None

# 读取扩展分区中的逻辑分区信息
def read_ext(disk, ext_start, ext_size):
    # 扩展分区的起始地址
    ext_base = ext_start
    # 逻辑分区列表
    log_parts = []
    # 循环读取每个逻辑分区
    while True:
        # 读取逻辑分区的引导扇区
        disk.seek(ext_start * 512)
        log_boot = disk.read(512)
        # 逻辑分区的分区表从第446字节开始，每个表项占16字节，只有前两个表项有效
        log_table = log_boot[446:462]
        # 第一个表项描述了当前逻辑分区的信息
        log_entry = log_table[:16]
        # 获取逻辑分区的起始扇区和扇区数，相对于当前引导扇区
        log_start = struct.unpack('<I', log_entry[8:12])[0]
        log_size = struct.unpack('<I', log_entry[12:16])[0]
        # 计算逻辑分区的绝对起始扇区，相对于硬盘
        log_start += ext_start
        # 将逻辑分区的信息添加到列表中
        log_parts.append((log_start, log_size))
        # 第二个表项描述了下一个逻辑分区的引导扇区的位置，相对于扩展分区
        ext_entry = log_table[16:32]
        # 获取下一个逻辑分区的引导扇区的起始扇区和扇区数
        ext_start = struct.unpack('<I', ext_entry[8:12])[0]
        ext_size = struct.unpack('<I', ext_entry[12:16])[0]
        # 如果下一个逻辑分区的引导扇区的起始扇区为0，表示没有更多的逻辑分区，退出循环
        if ext_start == 0:
            break
        # 否则，计算下一个逻辑分区的引导扇区的绝对起始扇区，相对于硬盘
        ext_start += ext_base
    # 返回逻辑分区列表
    return log_parts

# 判断一个分区是否包含squashfs文件
def is_squashfs(disk, part_start, part_size):
    # 读取分区的前4个字节，squashfs文件的魔术字为'h' 's' 'q' 's'
    disk.seek(part_start * 512)
    magic = disk.read(4)
```

```
    # 如果是squashfs文件，返回True，否则返回False
    return magic == b'hsqs'

# 读取一个分区中的squashfs文件
def read_squashfs(disk, part_start, part_size):
    # 读取分区的所有数据，返回一个字节串
    disk.seek(part_start * 512)
    data = disk.read(part_size * 512)
    return data

# 解压缩一个squashfs文件
def unzip_squashfs(sqsh):
    # 这里可以使用第三方库或者调用系统命令来解压缩squashfs文件，例如使用squashfs-tools
    import squashfs
    # 创建一个SquashFS对象，传入squashfs文件的字节串
    fs = squashfs.SquashFS(sqsh)
    # 遍历squashfs文件中的所有文件和目录
    for entry in fs.list():
        # 打印文件或目录的名称和大小
        print(entry.name, entry.size)
        # 如果是文件，可以读取其内容，例如
        if entry.is_file():
            content = fs.read(entry)
            # 对文件内容进行处理，例如保存到本地或者显示出来
            # ...

# 获取扩展分区的起始地址和大小
ext_start, ext_size = parse_mbr(mbr)

# 如果存在扩展分区，进入扩展分区
if ext_start and ext_size:
    # 读取扩展分区中的逻辑分区信息
    log_parts = read_ext(disk, ext_start, ext_size)
    # 遍历每个逻辑分区
    for part_start, part_size in log_parts:
        # 判断是否包含squashfs文件
        if is_squashfs(disk, part_start, part_size):
            # 读取squashfs文件
            sqsh = read_squashfs(disk, part_start, part_size)
            # 解压缩squashfs文件
            unzip_squashfs(sqsh)

# 关闭硬盘设备
disk.close()
```

test.py:

```
import binwalk
import os
import datetime

def format_result(result):
    # 格式化结果，确保每一行都以逗号和换行符结尾
    formatted_result = ""
    for line in result.split(",␣"):
        formatted_result += line.strip() + ",\n"
    return formatted_result

def get_file_info(file_path):
    # 获取文件信息，包括大小和最后修改时间
    file_size = os.path.getsize(file_path)
    last_modified_time = datetime.datetime.fromtimestamp(os.path.getmtime(file_path)).
    return file_size, last_modified_time

def process_firmware_file(input_file_path, output_file_path):
    try:
        # 检查输入文件路径是否存在
        if not os.path.exists(input_file_path):
            raise FileNotFoundError(f"文件␣{input_file_path}␣不存在.")

        # 获取文件信息
        file_size, last_modified_time = get_file_info(input_file_path)

        # 等同于执行 'binwalk --signature firmware1.bin firmware2.bin'。
        # 注意使用 'quiet=True' 来抑制正常的 binwalk 输出。
        with open(output_file_path, 'w') as output_file:
            # 输出文件信息
            output_file.write(f"分析结果文件:␣{os.path.basename(input_file_path)}\n")
            output_file.write(f"文件大小:␣{file_size}␣字节\n")
            output_file.write(f"最后修改时间:␣{last_modified_time}\n\n")

            # 统计擦除次数的变量
            erase_count = 0

            for module in binwalk.scan(input_file_path, signature=True, quiet=True):

                # binwalk.scan 返回每个已执行模块的模块对象;
                output_file.write("%s␣结果:\n" % module.name)
```

```
                # 初始化上一个结果的文件路径为 None
                prev_file_path = None

                # 每个模块都有一个结果对象列表，描述执行模块返回的结果。
                for result in module.results:
                    # 使用 os.path.basename 获取文件名，避免显示完整路径
                    file_name = os.path.basename(result.file.name)

                    # 检查是否与上一个结果的文件路径相同，避免重复输出文件路径。
                    if file_name != prev_file_path:
                        # 输出文件名
                        output_file.write("文件名: " + "%s\n" % file_name + "\n")
                        # 更新上一个结果的文件路径
                        prev_file_path = file_name

                    # 输出该文件下的结果，每个信息都换行
                    formatted_result = format_result(result.description)
                    output_file.write("0x%.8X    %s\n" % (result.offset, formatted_re

                    # 检查描述信息中是否包含擦除计数相关的内容
                    if "UBI erase count" in result.description:
                        erase_count += 1

            # 输出擦除次数统计结果
            output_file.write(f"\n总擦除次数: {erase_count}\n")

        print(f"结果已写入 {output_file_path}")

    except FileNotFoundError as file_not_found_error:
        # 如果文件未找到，捕获 FileNotFoundError 异常并输出相应的错误信息
        print(f"文件未找到错误: {file_not_found_error}")

    except Exception as e:
        # 捕获其他异常，并输出相应的错误信息
        print(f"发生错误: {str(e)}")

# 自定义信息
firmware_input_path = 'C:/Users/27719/Downloads/python_course_design-master/python_co
output_result_path = 'output.txt'
process_firmware_file(firmware_input_path, output_result_path)
```

display_results.py:

```
import tkinter as tk
from tkinter import Scrollbar

# 创建GUI窗口
root = tk.Tk()
root.title("分析结果")

# 从文本文件中读取分析结果
file_path = 'C:/Users/admin/Downloads/python_course_design-master/python_course_design
with open(file_path, "r") as file:
    result_text = file.read()

# 创建文本框来显示分析结果
text_box = tk.Text(root, wrap="word")
text_box.insert("1.0", result_text)
text_box.pack(side="left", fill="both", expand=True)

# 创建一个垂直滚动条并绑定到文本框
scrollbar = Scrollbar(root, command=text_box.yview)
scrollbar.pack(side="right", fill="y")
text_box.config(yscrollcommand=scrollbar.set)

# 运行GUI
root.mainloop()
```

§3 总结

§3.1 分工情况

1. 小组成员徐赫：负责硬件部分,LINUX 操作, 部分代码编写, 阅读解析固件的库文档
2. 小组成员杨子豫：负责 binwalk 解析固件文件, 分析里面内容, 阅读解析固件的库文档
3. 小组成员徐歆诚：负责编写 GUI 结果展示程序, 文档编写,PPT 制作, 阅读解析固件的库文档。

§3.2 解决方案评价

详细见上方图片和实践内容

check_mbr 程序首先读取 MBR（主引导记录），解析分区表，找到扩展分区的起始地址和大小。然后读取扩展分区中的逻辑分区信息，检查每个逻辑分区是否包含 SquashFS 文件系统，如果包含则读取并解压缩。

test 程序用于对给定的固件文件执行 binwalk 扫描，并将扫描结果输出到文件中。程序首先检查输入的固件文件路径是否存在，然后获取文件信息，包括文件大小和最后修改时间。接下来使用 binwalk 库执行扫描，并将结果输出到指定的输出文件中。在处理扫描结果时，程序会遍历每个已执行模块（在这个例

子中主要是签名模块），然后遍历每个模块的结果对象列表，输出文件名、偏移位置和格式化后的扫描结果。程序还定义了一些辅助函数，如格式化扫描结果和获取文件信息的函数。

display_results 程序是用 Python 的 tkinter 库创建了一个简单的 GUI 窗口，用于显示之前生成的分析结果文本。程序首先使用 tkinter 创建了一个窗口，设置窗口的标题为“分析结果”。然后从文本文件中读取分析结果，并将结果文本显示在一个文本框中。另外，程序还创建了一个垂直滚动条，并将其绑定到文本框，以便在文本内容过长时可以进行滚动查看。

§3.3 实践心得

以用坏了一个可以刷软路由的路由器为代价，我们明白了偏移量的重要性

学习使用 UBOOT 调试内容, 如何飞线焊板子

使用万用表测试电路, 排除故障, 查找固件 superwrt 和 openwrt

学习了 LINUX 嵌入式知识, 包括 LINUX 分区,squashfs 格式,LINUX 启动流程

我们可以把固件里面的 rootfs 内容作为虚拟文件系统, 挂载到自己的 LINUX 服务器上, 实现对里面文件的访问

在编写 check_mbr 程序的过程中，我们学到了如何使用 Python 的 struct 库来进行二进制数据的解析，从而读取和分析硬盘的 MBR 和分区信息。另外，我们也了解了如何读取和解析硬盘中的扩展分区和逻辑分区信息。通过这段程序，我们还学习了如何判断分区中是否包含特定类型的文件（这里是判断是否包含 squashfs 文件），以及如何读取和解压缩这些特定类型的文件。在实际编写解析和处理硬盘分区的程序时，我们深刻认识到了数据的存储和组织形式对于程序设计的重要性。同时，通过这段程序，我们还学会了如何利用第三方库（比如 squashfs-tools 库）来处理特定类型的文件，这拓展了我的解决问题的思路和方法。

test.py 程序是用于处理固件文件的分析和输出结果，我学到了如何使用 Python 的 os、binwalk 和 datetime 等库来获取文件信息并执行文件扫描。这段程序对文件处理和异常处理有了更深入的理解。通过对二进制文件的解析，我们更深入的了解了路由器 flash 芯片中存储的信息，同时我们对于其存储格式和存储顺序有了进一步的认识。

display_results 程序则是使用 tkinter 库创建 GUI 窗口，展示了如何将程序的结果以图形界面的形式呈现给用户。通过这部分的编写，我了解了 tkinter 库的使用，学习了如何创建窗口、文本框和滚动条，以及如何将文本文件中的内容展示出来。

我们认为在编程的过程中，了解库的使用很重要。对于处理文件、创建 GUI 等常见需求，库的使用可以大大简化程序的编写和提高效率。另外，编写这些程序还锻炼了我的问题解决能力和错误处理能力，特别是异常处理方面。总的来说，这个经历让我对 Python 编程的实际应用有了更深刻的认识，也为我以后的编程学习提供了更好的基础。通过不断练习和尝试，我们相信未来会越来越熟练地运用 Python 来解决实际问题。

§3.4 特别鸣谢

也非常感谢我们的指导老师，沈炜老师 对我们的指导与鞭策，有幸能够在碌碌无为的大二就对嵌入式的相关知识有了初步的了解。从最开始的一无所知，到一步步的尝试，或许结果并不完美，也并未得到我们想要提取的所有信息，但这在很大程度上强化了我们的动手能力和对新知识的学习和探索能力。因为时间紧迫，部分工作或许略显糙粗，我们也希望在后续工作中继续完善。

参考文献

https://github.com/matteomattei/PySquashfsImage PySquashfsImage 项目的开源地址

https://github.com/plougher/squashfs-tools squashfs-tools 项目地址

debian 官方文档
https://juejin.cn/s/
https://blog.csdn.net/hcu5555/article/details/115005114 博客内容
https://github.com/ReFirmLabs/binwalk binwalk 项目地址
https://www.dwdvb.com/neoprogrammer-new-update-v2-2-0-10/ neoprogrammer
https://openwrt.ai/ 来自 GITHUB 开源项目的 OPENWRT 固件 (已经编译好的)
https://www.superwrt.com/ superwrt 固件地址