Lab 1：Web 导论

3250101697 冯梓航

- `1.system_echo_solve.py`：PPT 18 页
- `2.jttp_solve.py`：PPT 54 页
- `3.zip_slop_solve.py`：PPT 71 页

1. Challenge 1：PPT 18 页

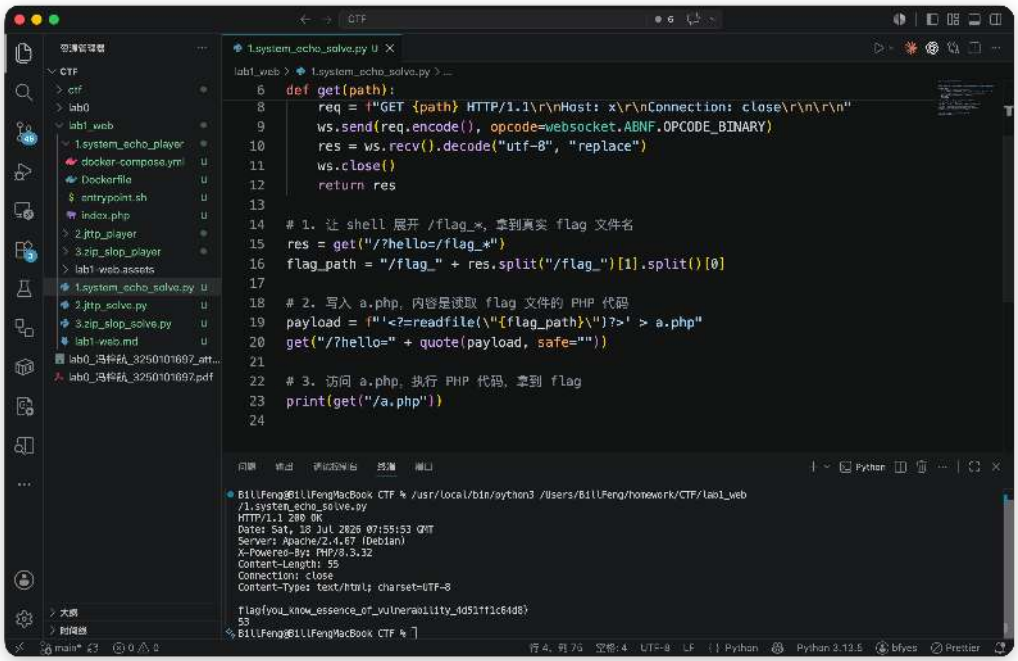
```php
if (preg_match('/[;|&`$\n\r\t]/', $_GET['hello'])) {
    exit('blocked');
}

system('echo '.$_GET['hello']);
```

这里虽然过滤了 `;`、`|`、`&`、反引号、`$` 等字符，但用户输入仍然被拼接进 shell 命令执行。

由于 `*` 通配符和 `>` 重定向没有被过滤，可以先通过 `/?hello=/flag_*` 让 shell 展开真实 flag 文件名，再构造：

```
echo '<?=readfile("/flag_xxx")?>' > a.php
```

写入一个可以读取 flag 的 PHP 文件，最后访问 `/a.php` 得到 flag。完整自动化过程见 `1.system_echo_solve.py`。



2. Challenge 2：PPT 54 页

前端代理会拦截直接访问 `/flag`：

```
if path == "/flag":
    self.reply(403, "blocked\n")
    return
```

但是代理随后手工拼接 JSON 请求发给后端：

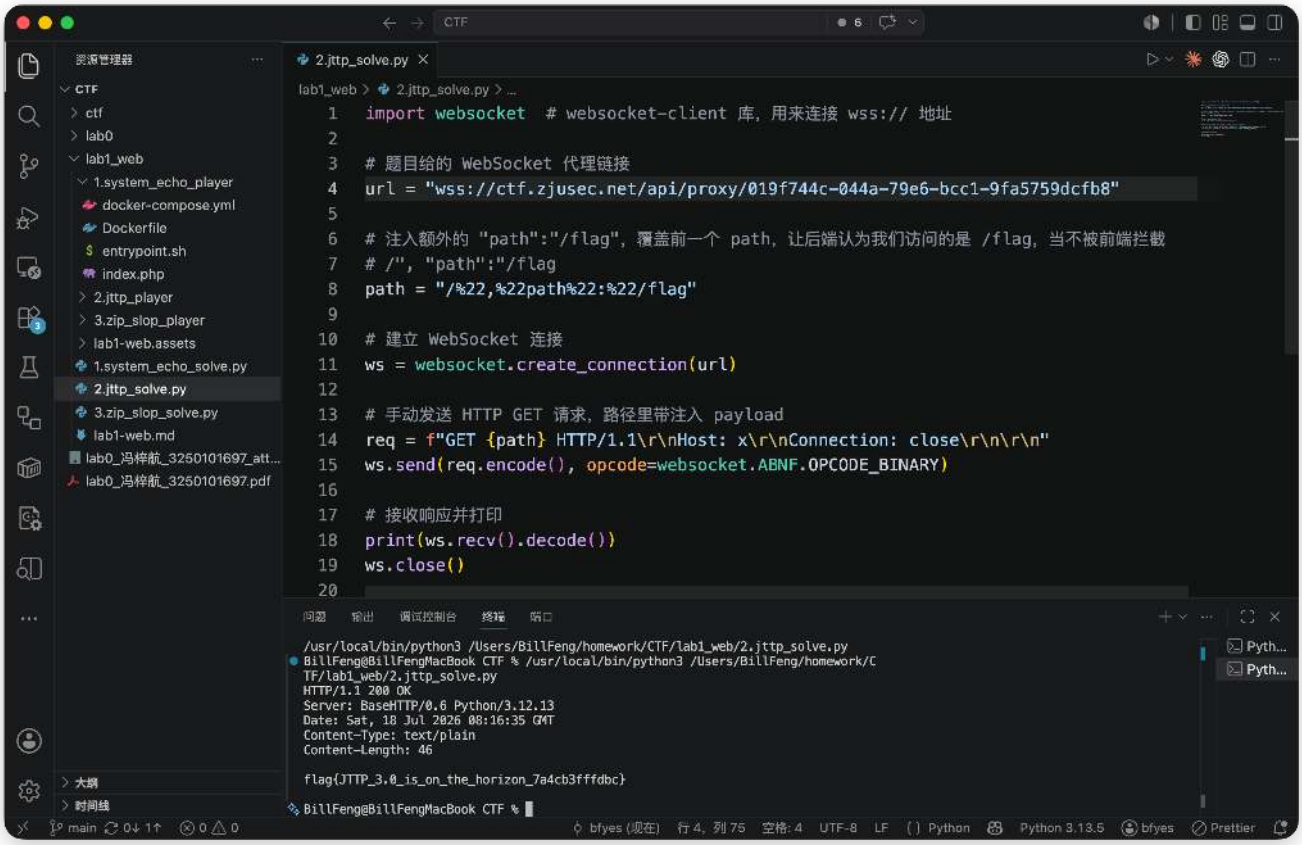
```
req = '{"method":"GET","path":"{path}","headers":"{headers}"}\n'
req = req.replace("{path}", path).replace("{headers}", headers).encode()
```

后端真正根据 JSON 里的 `path` 字段决定是否读取 flag。

利用点是 JSON 字符串没有做转义，路径中插入：

```
/", "path":"/flag
```

即可构造出重复的 `path` 字段。Python 的 `json.loads` 在遇到重复键时会保留后面的值，于是后端最终看到的 `path` 是 `/flag`，绕过了前端代理的检查。



3. Challenge 3：PPT 71 页

题服务端允许上传 ZIP，解压后会把指定的额外文件复制到临时目录，再重新打包返回。代码显式禁止直接选择 `/flag`：

```
if extra_path == "/flag" or os.path.realpath(extra_path) == "/flag":
    return page("/flag is blocked", 400)
```

但服务端没有检查上传 ZIP 解压后的文件类型。构造一个包含符号链接的 ZIP，使 `flag_link` 指向 `/flag`：

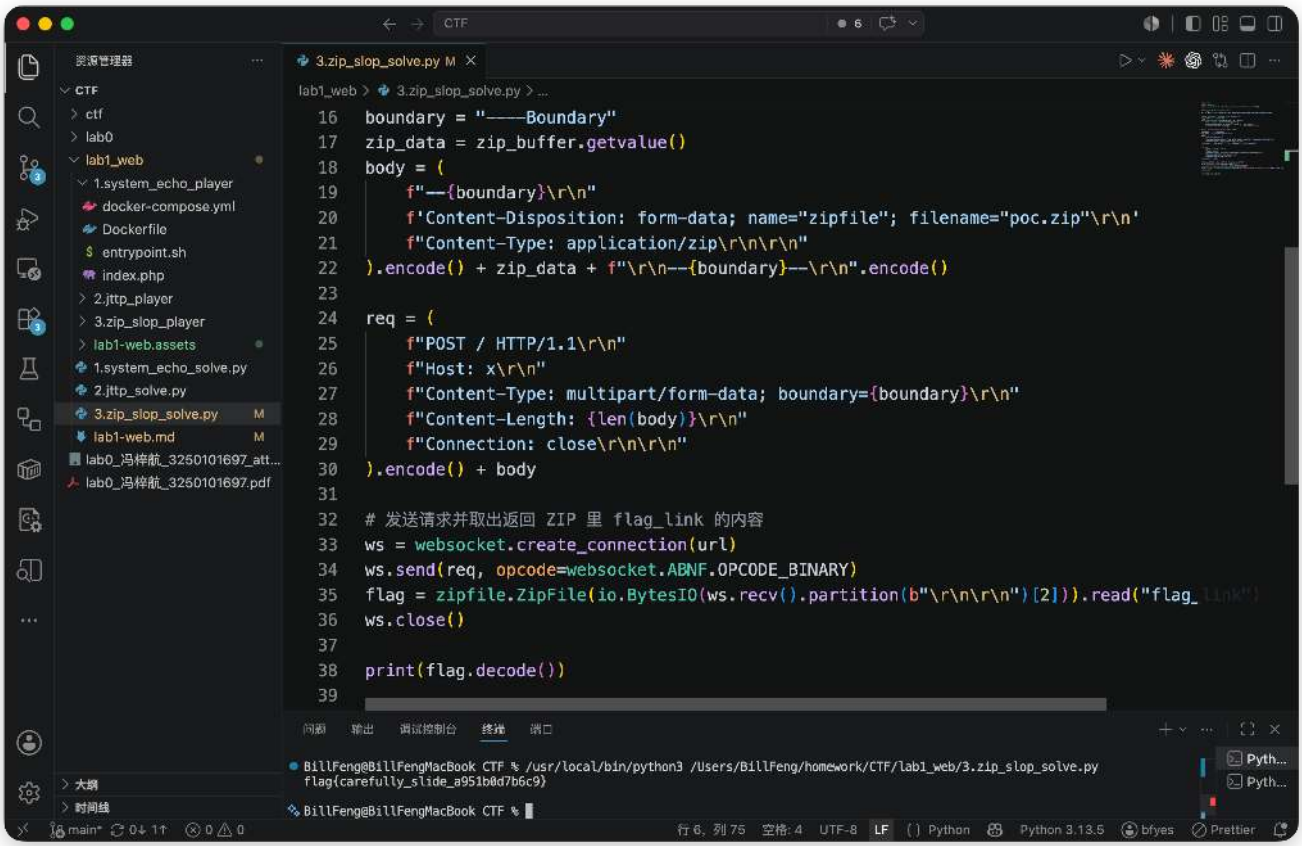
```
info = zipfile.ZipInfo("flag_link")
info.external_attr = 0o120777 << 16
zf.writestr(info, "/flag")
```

`unzip` 解压后会在临时目录中生成这个符号链接。

下面是Agent的对话：

```
我：
  0o120777 << 16 是什么意思呢?
AI Agent：
  0o120777 << 16 是在给 ZIP 条目设置 Unix 文件属性，让 unzip 解压时把它当作一个符号链接（symlink）。
1. 0o120777（八进制）
  这是 Unix 的 st_mode，由「文件类型」+「权限」两部分组合：
  0o120000 = S_IFLNK，即符号链接类型
  0o000777 = rwxrwxrwx，权限位
  二者按位或就是 0o120777（十进制 41471），含义是「这是一个权限 0777 的符号链接」。
2. << 16（左移 16 位）
  ZIP 格式里每个条目有一个 32 位的 external_attr 字段，它的高 16 位用来存 Unix 的 st_mode。把 0o120777 左移 16 位，就是把 mode 放到高 16 位上：
  external_attr = st_mode << 16
                = 0o120777 << 16
                = 0xA1FF0000
  unzip 解压时会读高 16 位，看到 S_IFLNK 标志，就会把该条目的数据当作链接目标路径（这里是 /flag），在磁盘上建出一个真正的符号链接，而不是写入一个普通文件。
```

后续 `zip -qr archive.zip .` 重新打包时会跟随符号链接读取 `/flag` 内容，并把它放入返回的压缩包中。脚本接收返回 ZIP 后读取其中的 `flag_link`，即可得到 flag。



4. ==AI生成部分==

Challenge 4：ACTF 2026 WEB 12307

题目地址：https://github.com/team-s2/ACTF-2026/tree/main/web/12307

> 同一份 JSON，程序“检查”的时候看到的是前一个值，“真正执行”的时候看到的是后一个值。

相关文件已整理到 `4.actf-12307/`，官方 exp 在 `4.actf-12307/exp/exp.py`，题目环境在 `4.actf-12307/env/`。

4.1 漏洞点

```
public_view = json.loads(payload_text, object_pairs_hook=first_wins_object)
render_view = json.loads(payload_text)
```

这里把同一段 JSON 解析了两次：

- `public_view`：重复字段保留第一次出现的值，用来做安全检查；
- `render_view`：重复字段保留最后一次出现的值，用来真正渲染打印内容。

所以 exp 构造了两组同名字段：

```
"printProfile":"counter-copy",
"printer":"thermal-standard",
"printProfile":"clearing-batch",
"printer":"line-printer",
"driverProgram":"/usr/bin/base64",
"driverArgument":"/flag"
```

检查时程序看到 `counter-copy / thermal-standard`，觉得没问题；执行时程序看到 `clearing-batch / line-printer`，就会进入更危险的打印流程。最后打印程序执行：

```
/usr/bin/base64 /flag
```

这样 flag 会被 base64 编码后放到返回的报表里。

官方 exp 的其他步骤主要是在补业务条件，比如创建订单、获取登车确认、触发结算报表等。看起来很长，但核心就是为了让程序走到上面这个打印流程。

4.2 最小修复

修复思路很简单：JSON 里不允许出现重复字段。只要发现重复 key，就直接拒绝。

修改 `4.actf-12307/env/services/receipt_signer/app.py`：

```python
def unique_object(pairs):
    result = {}
    for key, value in pairs:
        if key in result:
            raise ValueError(f"duplicate json key: {key}")
        result[key] = value
    return result
```

并将原来的双视图解析改为：

```
public_view = json.loads(payload_text, object_pairs_hook=unique_object)
render_view = public_view
```

这样检查和执行用的是同一个结果，exp 里的重复 `printProfile` 会直接被拦下来。