从零开始制作一个Web蜜罐扫描器

想法的来源

在渗透的过程中，会遇到很多蜜罐，一旦不小心踩了蜜罐，就会被溯源，所以很可怕。

为了规避上面的现象，就需要把蜜罐筛出来。

使用场景是在前期资产收集的过程中，搞到了一堆子域名，先筛掉一批蜜罐，留下可以攻击的纯净资产。

同上得到的一份资产如下：

http://accounts.ccb.com
http://accounts1.ccb.com
http://accounts2.ccb.com
http://acesso2.br.ccb.com
http://ad3.icbkfs.com
http://ae.ccb.com
http://ae1.ccb.com
http://ae2.ccb.com
http://api.joying3.cmbchina.com
http://api.open.icbkfs.com
http://asahi.icbkfs.com
http://au.ccb.com
http://au2.ccb.com
http://autodiscover.cmbchina.biz
http://b.esgcc.com.cn
http://bbs.ccb.com
http://bbs.ccb.com/index.fhtml
http://bbuy.esgcc.com.cn
http://big5.cmbchina.com
http://big5.cmbchina.com/gate/big5/www.cmbchina.com
http://big5.cmbchina.com/gate/big5/www.cmbchina.com/
http://big5.icbc-ltd.com
http://big5.icbc-ltd.com/icbcltd/
http://big5.icbc.com.cn
http://biz.mall3.dccnet.com.cn
http://biz.mall3.dccnet.com.cn/main.jhtml
http://blog.ccb.com
http://blog.ccb.com/blog
http://blog.ccb.com/blog/
http://blog.ccb.com/blog/system-index.bhtml
http://bmerchant.esgcc.com.cn
http://br.ccb.com
http://buser.esgcc.com.cn
http://buy.esgcc.com.cn
http://ca.ccb.com

如上图所示，大量的域名如果靠人工来筛选蜜罐，费时费力，开发一个工具来减少工作量是很自然的想法。

实现的思路

要实现蜜罐扫描器，首先得搞清楚以下几个问题：

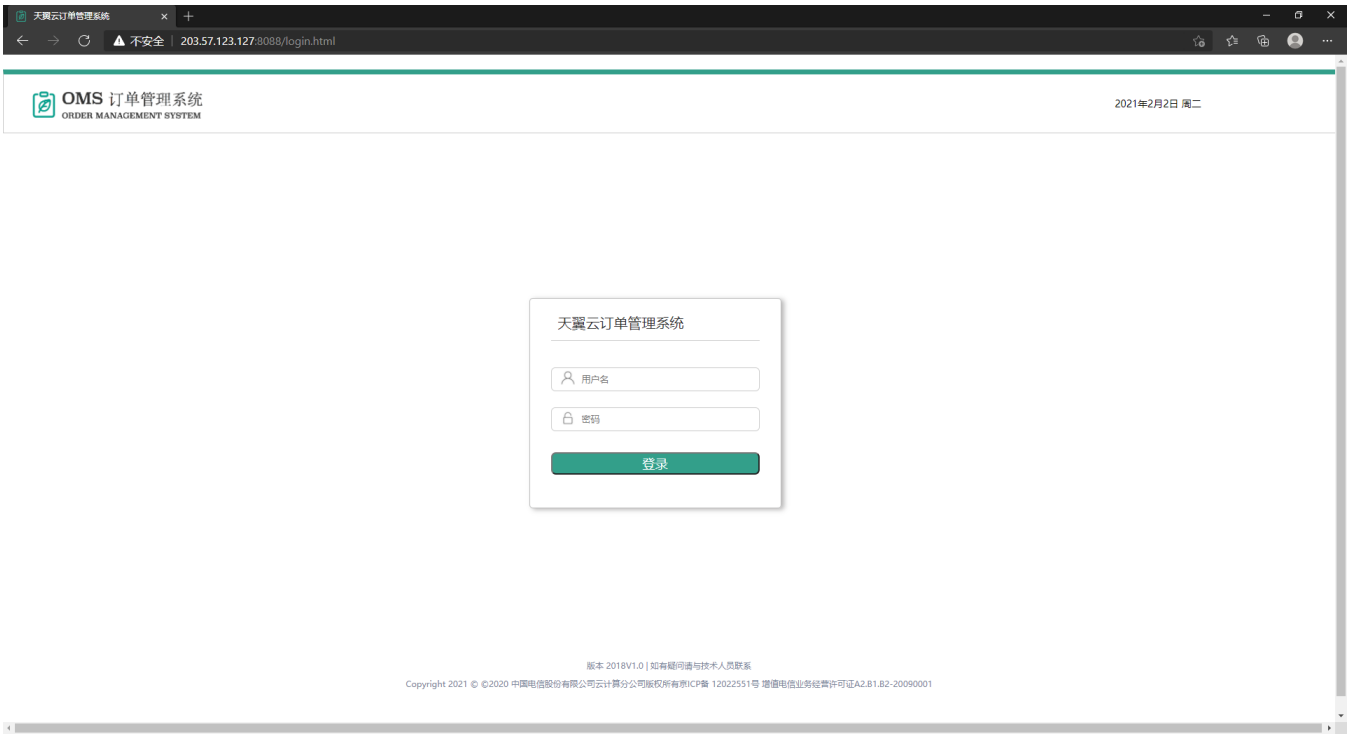
- **蜜罐是什么**
- **蜜罐有啥用**
- **蜜罐有什么特征**
- **怎么根据特征从一大批资产中找到蜜罐**
- **怎么快速准确的找到蜜罐**
- **怎么快速且准确且批量的找到蜜罐**
- **怎么快速且准确且批量的找到蜜罐然后还不会留下痕迹**

很明显，一个成功的扫描器，是能够成功解决上述问题的。

我的制作思路，也是跟着上面的步骤一步步走下来的，以下分点进行详细阐述。

- **蜜罐是什么**

蜜罐，英文名honeypot，蜜罐蜜罐，就是带蜜的罐子，进去吃了蜜，就出不来了，本质上就是一个陷阱。



上面是一个从网上扒下来的蜜罐，长得人模狗样。

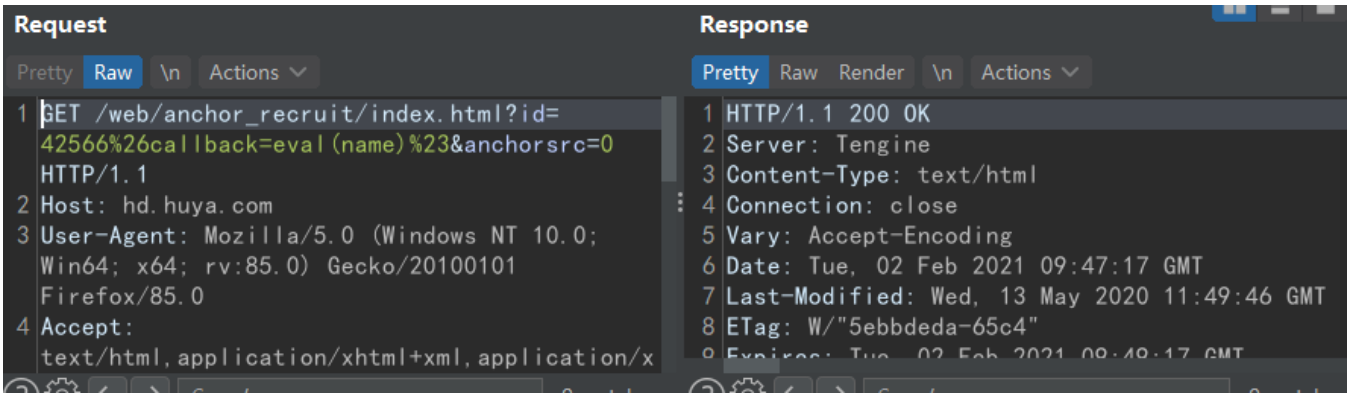
进一步查看前端js代码可以发现竟然直接使用js进行前端校验：

```
function login() {
  var username = $('#username').val().trim();
  var password = $('#password').val().trim();
  if (!validate(username, password)) return;
  var user = {
    "username": username,
    "password": password,
  }
  $.ajax({
    url: '/api/login',
    type: 'POST', // 请求的类型（GET 或 POST）
    xhrFields: {
      withCredentials: true
    },
    async: false, // 是否异步
    timeout: 30000, // 请求超时时间（以毫秒计）
    data: JSON.stringify(user),
    contentType: 'application/json',
    dataType: 'json', // 服务器响应的数据类型 json/xml/html/jsonp/text
    statusCode: { //对响应状态码进行拦截
      401: function () {
        modalFun('用户名或密码错误！');
      }
    },
    success: function (result, status, xhr) {
      if (result.code === 0) {
        modalFun('登录成功', 'success')
        setTimeout(function () {
          window.location.href = "index.html"
        }, 500)
      } else {
        modalFun("用户名或密码错误")
      }
    },
    error: function (xhr, status, error) {
      console.log(xhr.status);
    }
  });
}
```

可以看到这里如果code为0直接就可以进去了，显然是在耍我。

- **蜜罐有啥用**

蜜罐的用处就是利用jsonp捕获黑客社交网络信息。

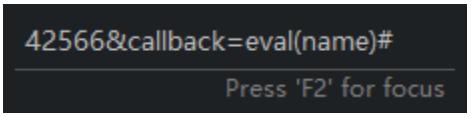


上图就请求了hd.huya.com这个网页。

注意看这段payload

/web/anchor_recruit/index.html?id=42566%26callback=eval(name)%23&anchorsrc=0

本质上就是



通过一个回调函数，巧妙的绕过了同源策略的限制，达到了跨域请求的目的。

通过跨域请求，获得了攻击者的社交网络信息，在这个例子中是huya的id信息

当然，这要页面存在jsonp漏洞才能派上用场。

如果不理解同源策略，关于同源策略以及绕过的方法可以看这篇文章：

https://blog.csdn.net/duzm200542901104/article/details/85870019

- **蜜罐有什么特征**

根据我们上面的描述，现在可以理解，蜜罐的功能是诱捕黑客。

诱捕黑客是利用的jsonp劫持技术来做。

jsonp劫持技术需要利用jsonp跨域请求很多其他的社交网站页面从而获得攻击者的相关信息。

因此蜜罐的特征就是会发送很多非正常的跨域请求，并且请求的都是一些常用的社交网站，如bilibili，baidu，huya，youku等。

所以鉴别方法就是抓包看是否存在很多跨域请求，这里直接用burp抓包即可。

417	http://my.zol.com.cn	GET	/public_new.php		200	463	script	php
416	http://chinaunix.net	GET	/site/loginstatusbar.html		200	668	script	html
415	https://www.cndns.com	POST	/cn/domain/search.aspx	✓	302	456	HTML	aspx
414	https://p.qiao.baidu.com	GET	/cps3/mobileChat?siteId=12740146...	✓	200	7900	HTML	
413	https://bbs.zhibo8.cc	GET	/user/userinfo?device=pc&_=15846...	✓	200	438	script	
412	https://hudong.vip.youku.com	GET	/act/mili/download.html?mobile=1&...	✓	200	1315	HTML	html
411	https://www.zbj.com	GET	/g/service/api/getUserPhone?&callb...	✓	200	740	script	
410	https://wap.sogou.com	GET	/passport?op=get_userinfo&_=154...	✓	502	973	HTML	
409	https://v2.sohu.com	GET	/user/info/web?&callback=jsonp_cal...	✓	401	107		
408	http://m.game.weibo.cn	GET	/notice/index/save_eventing?appkey...	✓	200	503	text	
407	https://yys.cbg.163.com	GET	/cgi/mweb/search/r/role?keyword=...	✓	200	8622	HTML	
406	https://analyze.pwnchain.cn	GET	/s/jquery.min.js?v=1612259319921	✓	200	145106	script	js
405	https://c.v.qq.com	GET	/vuserinfo?otype=json&callback=js...	✓	404	156	HTML	
404	http://passport.game.renren.c...	GET	/user/info?callback=jsonp_callback_...	✓	200	224	HTML	
403	http://mapp.jrj.com.cn	GET	/pc/content/getMqNews?vname=%...	✓	200	857	HTML	
402	https://www.iqiyi.com	GET	/intl/invite.html?lang=zh_cn&mod=...	✓	404	190817	HTML	html
401	https://iask.sina.com.cn	GET	/cas/logins?domain=iask.sina.com.c...	✓	200	14931	HTML	
400	http://account.itpub.net	GET	/login/sso?url=javascript%3Aeval%2...	✓	302	526	HTML	
399	https://hd.huya.com	GET	/web/anchor_recruit/index.html?id=...	✓	200	28715	HTML	html
398	https://m.ctrip.com	GET	/webapp/things-to-do/list?pagetype...	✓	200	12628	HTML	
397	https://u.faloo.com	GET	/regist/Login.aspx?txtUserID=%22...	✓	200	10874	HTML	aspx
396	https://webapi.ctfile.com	GET	/api.php?item=file_act&action=xt_d...	✓	200	2841	HTML	php
395	https://api.csdn.net	GET	/oauth/authorize?client_id=100000...	✓	200	5128	HTML	
394	http://databack.dangdang.com	GET	/dde.php?platform=pc&type=3&url...	✓	200	364	text	php
393	https://ajax.58pic.com	GET	/58pic/index.php?m=adManageSyst...	✓	200	1156	HTML	php

如上所示，请求了很多其他的社交网站api，这是一个很显著的特征，蜜罐本身使用了jsonp劫持的溯源技术，当进入这个网页的那一刻，他就开始溯源攻击者了。

如果对jsonp劫持攻击不熟悉可以看这篇文章，讲得很详细：

https://www.cnblogs.com/happystudyhuan/p/11583384.html

- **怎么根据特征从一大批资产中找到蜜罐**

这是一个很实际的问题。

如果用通过上面burp抓包一个个点开然后一个个看，其实也能做到，但是很慢。

在做攻击队项目的时候，如果还是用这种原始的方式，肯定是不行的。

因此开发一个自动化工具快速找到蜜罐的想法应运而生，下面说一下我开发的思路：

首先，通过上面的介绍我们可以了解到，无论什么蜜罐，无论哪个厂商的蜜罐，只要是web蜜罐，他就会去做jsonp劫持攻击，做了jsonp劫持，就会有request，所以我这里用爬虫去抓request就可以了。

抓到了request之后，会有两个数据，一个是request是什么，可能是baidu.com,github.com等等，还有一个就是request的量，通过上面的截图我们可以看到其实request的量是很大的，至少有大几十条。

由于我们前期抓到了一些蜜罐，其实这里可以把蜜罐中去请求的request提取出来作为一个字典，然后以这个字典作为基准去判断是否其他的网站是否存在同样的request请求。

上述判断方式具有一定的通用性，因为无论是哪家厂商的蜜罐，来来回回请求的api就那么几个。

思路理清楚了，那么就针对需求进行代码实现：

爬虫实现

爬虫这个东西，说难不难，说简单也不简单，一个高性能的爬虫其实也是很有技术含量的。

这里星光大佬直接推荐了crawlergo，附链接：

https://github.com/0Kee-Team/crawlergo

使用crawlergo进行爬取：

```
┌──(root💀kali)-[/home/kali/Desktop/bond7pypy]
└─# ./crawlergo -c /home/kali/Desktop/chrome-linux/chrome -t 10 -o json http://117.32.155.71:9002
```

crawlergo可以直接爬取一个ip的所有request，然后返回一个json文件，如下：

```
}],"all_domain_list":["epay.citicbank.com","getbootstrap.com","bootstrapvalidator.com","twitter.com","COOKIE",",Mb=","mg","ig","res.epay.citicbank.com","ecitic2.qdone.com.cn","resecitic2.qdone
```

在all_domain_list这个字段里面，存在请求这个ip后爬取到的所有request的domain。

这里通过crawlergo，就实现了抓取request的功能。

数据清洗和判断实现

爬取完毕后的数据需要进行清洗，才能拿来使用，那么如何清洗呢？

这里使用了python的re模块来进行清洗，思路如下：

首先匹配到all_domain_list

```
47      result1 = re.search('\"all_domain_list\"\:\[(.*?)\]',testiplist,re.S)
```

然后将匹配到的数据转换成一个大列表

```
48      a1 = result1.group(1).replace('"','').split(',')
```

接下逐个判断字典里是否有这个值，如果有，就设置一个计数器count++，如果没有，就不动，完整代码实现如下：

```
    for i in a1:
        if i in biglist:
            count = count + 1
    if count > 0 and count < 20 :
        print("possible honeypot...%s"%target)
        print("confirm times:%s"%count)
        savefile(target,resultfilename,count)
    if count >= 20:
        print("honeypot!!!---%s"%target)
        print("confirm times:%s"%count)
        savefile(target,resultfilename,count)
    else:
        print("nomal url---%s"%target)
        print("confirm times:%s"%count)
```

其实这样写有点麻烦，星光大佬提出了直接使用集合匹配交集的方法，有效减小了代码量，这里也贴出来：

```
def work(data):
    all_domain_list = data["all_domain_list"]
    local_domain_list = map(str.strip, load_file(LOCAL_DOMAIN))
    result = set(all_domain_list) & set(local_domain_list)
    return len(result) >= 20
```

文件读取和写入实现

上面的工作已经完成了逻辑判断的部分，下面还需要进一步完善一些旁支末节的部分。

因为爬虫生成的文件是一个json文件，那么操作这个文件还需要一个函数，同时对文件操作完毕之后需要将result写入一个文件以供最终审阅。

因此还需要添加文件读取和写入的模块。

读取模块实现如下：

```
#加载文件
def loadfile(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        content = f.read()
    return content
```

写入模块实现如下：

```python
#保存文件
def savefile(target,filename,count):
    with open(filename, 'a', encoding='utf-8') as f:
        f.write("count:" + str(count) + "---"+target)
        f.write("\n")
```

那么完成了上述步骤，其实一个简易的蜜罐扫描器雏形就搭建完成了，下面需要进行的是对扫描器进一步的优化。

- **怎么快速找到蜜罐**

通过初期对扫描器的测试，会发现这个扫描器扫的非常慢。

完成一个ip的扫描，平均需要十多秒的时间，这个速度十分的离谱。

因为一般在实战中，几个域名裂变成的子域名，如果是大的站，可能就会有成百上千个，子域名再裂变，那么数量就极为庞大了。

为了解决大批量扫描的问题，就必须提高扫描的性能。

由于爬虫是调取别人的，且crawlergo是闭源的工具，这里就只能进行本地的优化，这里的思路是利用python的多线程进行优化。

关于python多线程的介绍可以看这篇文章：

https://www.cnblogs.com/luyuze95/p/11289143.html

这里我们选择的是threadpool，需要pip额外安装，这里直接在python中pip install就好。

```
C:\Users\bh>pip3 install threadpool
Looking in indexes: https://pypi.tuna.tsinghua.edu.cn/simple
Requirement already satisfied: threadpool in c:\zzzzzzzz\python3.9\lib\site-packages (1.3.2)
```

可以看到我这里是已经安装完毕了。

然后在代码中定义线程的数量

```python
    pool = threadpool.ThreadPool(10)
```

我这里是直接写死的，看起来有点笨拙，那么可以换一种方法，这里需要再import一个模块argparse。

```python
import argparse
```

那么这里就可以写成这个样子：

```python
    pool = threadpool.ThreadPool(args.thread)
```

args.thread的值是从命令行中取得的 如下

```
root kali)-[/home/kali/Desktop/bond7pypy]
./crawlergo -c /home/kali/Desktop/chrome-linux/chrome -t 10 -o json http://117.32.155.71:9002
```

这里是设置了10个线程

那么通过上述多线程的方式，扫描的速度就得到了大大的提高，虽然python的多线程一直饱受诟病，但有总比没有强，并且针对这种强io读写的操作，使用线程的确要恰当一些。

那么完整代码如下：

```
71      pool = threadpool.ThreadPool(args.thread)
72      requests = threadpool.makeRequests(work, iplist)
73      for i in requests:
74          pool.putRequest(i)
75      pool.wait()
```

其中work是我们调取的处理函数。

通过上述代码，就实现了一个多线程扫描器，在10个线程的条件下，实测1-2s可以扫描一个ip，扫描效率得到了大幅度提升。

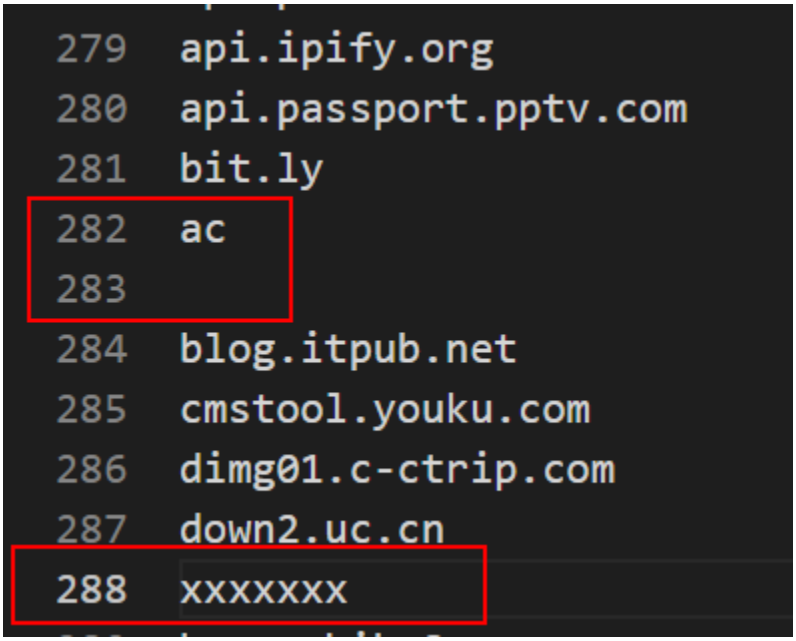
- **怎么快速准确的找到蜜罐**

快速的问题，通过多线程的方式得到了解决，下面还存在一个准确性的问题。

准不准确，来自于字典准不准确。

字典准不准确，在于对于字典的优化是否到位。

在初期爬取的时候，字典会有很多杂项，需要对字典去重&删除脏数据&删除空格。

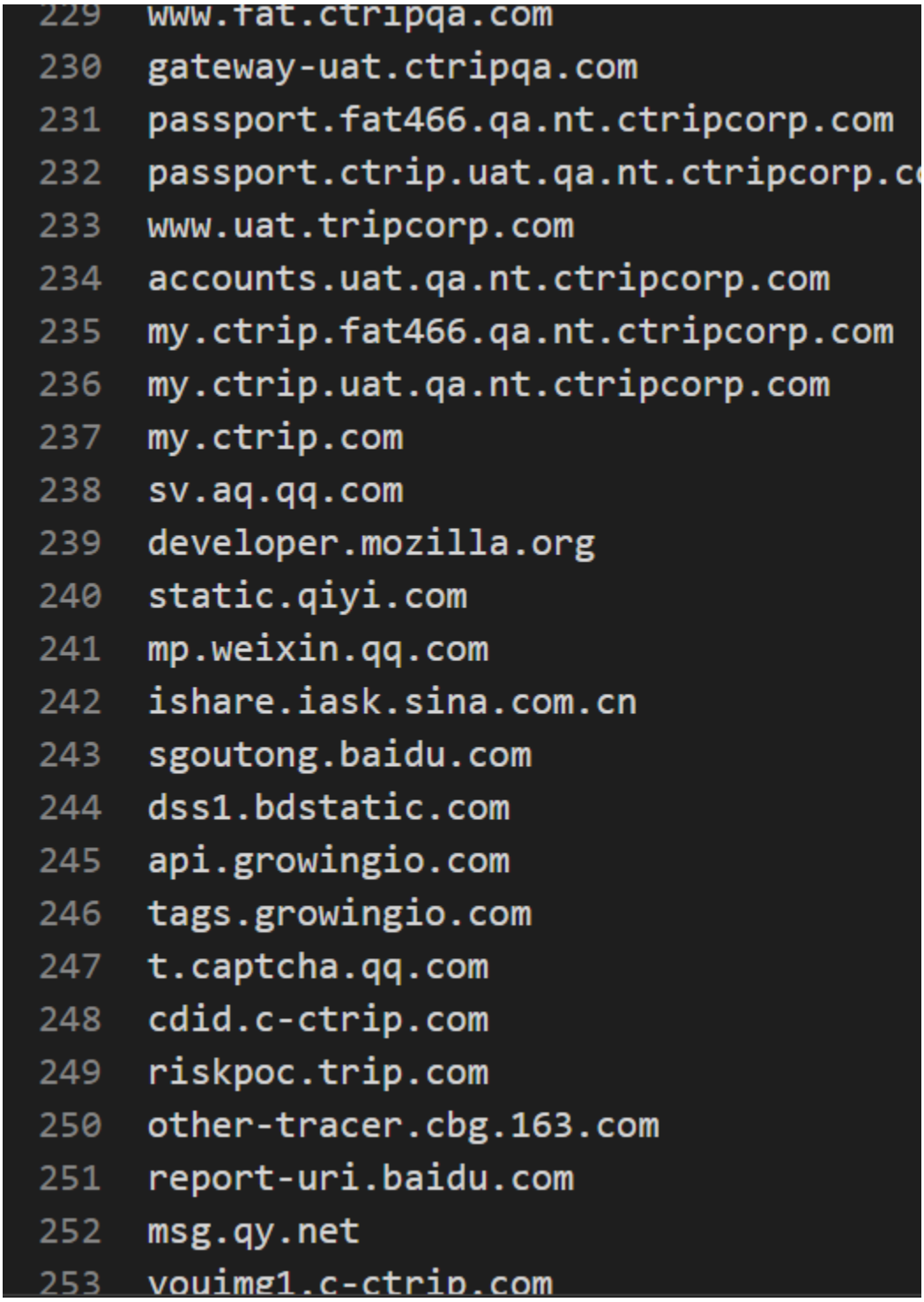


如上图所示，初期的字典包含了很多这样的数据，筛选和去重成为了一个必不可少的工作。

这里一分为二的来做这件事，首先将字典导入代码中筛掉空格，然后再人工的筛选出脏数据。

过程就不再赘述，因为其实也是相当简单的一件事。

那么经过字典的优化，最终就得到了一份ok的字典，如下所示：



基本上都是有效的api，那么字典清洗这一步到这里就算完成了。

有了一份好的字典，准确性的问题就迎刃而解。

- **怎么快速且准确且批量的找到蜜罐**

这里一个关键字就是批量

市面上很多好用的工具都是可以批量扫描的，这个东西不难实现，思路如下：

首先，把待检测的iplist写到txt的文件里。

读取待检测的txt文件

读取到了文件之后，把文件中的ip列成单独的一条条的形式。

把单独的ip传送给检测模块进行检测。

于是根据上面的思路，直接写一个解析模块就行：

```
21  #把txt里面的ip转换为list
22  def parseiplist(filename):
23      content = loadfile(filename).split('\n')
24      return content
```

```
16  def loadfile(filename):
17      with open(filename, 'r', encoding='utf-8') as f:
18          content = f.read()
19      return content
```

通过上面的解析模块，就可以完成iplist.txt文件的解析工作。

解析完毕后，会return一个list。

- **怎么快速且准确且批量的找到蜜罐然后还不会留下痕迹**

这里痕迹清除的思路可以类比其他的扫描器或者渗透过程中痕迹清除的思路。

在实际渗透的过程中或者使用漏扫的过程中，我们常常会使用代理。

使用代理一方面是为了防止被溯源到，另一方面常常主机ip会被ban，所以要不断的更换代理来保证工作的正常完成。

那么这里本质上就是需要搭建一个代理池。

代理池这个东西，如果展开来说，又是很大的篇幅，因此这里直接采用他人已经造好的轮子。

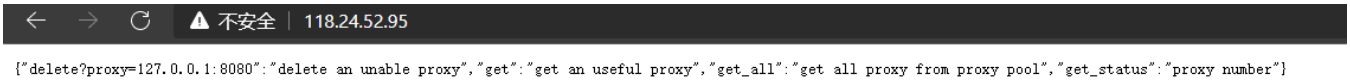
免费

如果选用免费代理，可以用这个proxypool：

https://github.com/jhao104/proxy_pool

这个代理池可以自动筛选出可用代理然后给与更换，不过由于都是免费的代理，所以可用度不是很高。

接口如图：



付费

如果资金充裕，那么就可以考虑付费版本。

付费版本是付费的代理，付费代理的花样就有很多了。

这里要介绍一种黑产常用的技术，叫做秒拨。

秒拨秒拨，本质上是一种拨号上网技术，现在被广泛的应用于黑产，一次断线重拨，就能能换一个ip，因此得名秒拨。

秒拨的详细说明可以看下面这篇文章：

https://blog.csdn.net/weixin_34268169/article/details/93024675

现在网上的秒拨池很多，这里不一一列举了，列出我常用的两个，云立方和极光代理，都可以配置代理池。

云立方：https://www.yunlifang.cn/

极光代理：http://www.jiguangdaili.com/

感兴趣可以试用，出了问题不要找我

通过代理池，其实这里根本就不用写是否被banip，因为他几秒一个ip，ip池资源完全够用，所以不用担心。

这里直接在电脑开启代理池，然后跑检测脚本即可，简单快捷。

由于过分简单，这里不再演示。

• 完整代码呈现

检测脚本代码链接：honeypotdetection.py

大字典：biglist.txt

crawlergo脚本：crawlergo

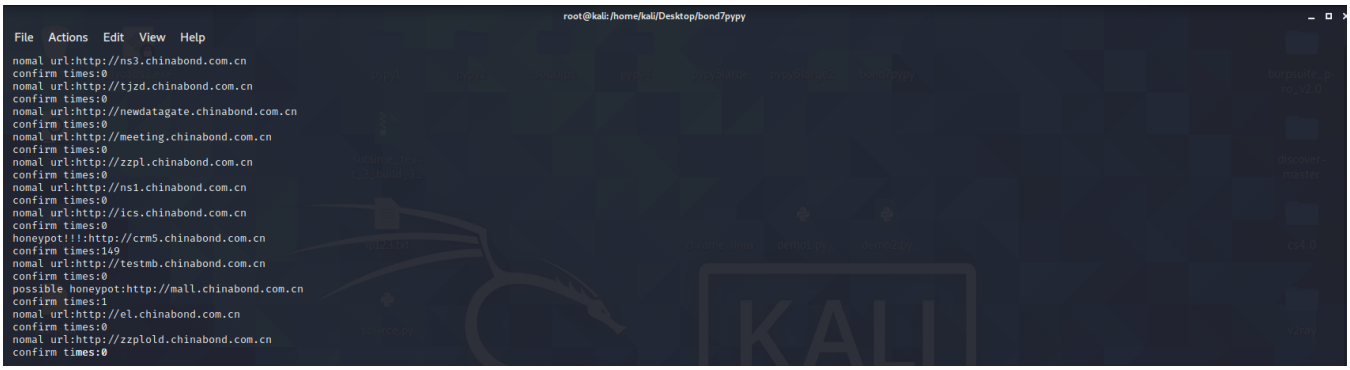
```python
#!/usr/bin/python3
# coding: utf-8
import subprocess
import re
from multiprocessing import Pool
import argparse
import threadpool
import uuid

parser = argparse.ArgumentParser(description="a honeypot detection program")
parser.add_argument('-f','--fileofiplist',type=str,default='iplist.txt',help='input your ipfile name(ex:-f iplist.txt)')
parser.add_argument('-t','--thread',type=int,default=5,help='thread numbers(ex:-t 10)')
args = parser.parse_args()

#加载文件
def loadfile(filename):
    with open(filename, 'r', encoding='utf-8') as f:
        content = f.read()
    return content

#把txt里面的ip转换为list
def parseiplist(filename):
    content = loadfile(filename).split('\n')
    return content

#保存文件
def savefile(target,filename,count):
    with open(filename, 'a', encoding='utf-8') as f:
        f.write("count:" + str(count) + "---"+target)
        f.write("\n")
```

```python
#爬取ip 生成该ip所有请求domain
def generatefile(target,jsonfilename):
    cmd = ["./crawlergo", "-m","5","-c", "/home/kali/Desktop/chrome-linux/chrome", "-o", "json", "--custom-headers","{\"User-Agent\": \"Mozilla/5.0
    rsp = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    output, error = rsp.communicate()
    # return jsonfilename

#处理文件
def work(target):
    # jsonfilename = str(uuid.uuid4())[0:4] + '.json'
    jsonfilename = 'xxx.json'
    generatefile(target,jsonfilename)
    resultfilename = "result.txt"
    biglist = loadfile("biglist.txt")
    testiplist = loadfile(jsonfilename)
    result1 = re.search('\"all_domain_list\"\:\[(.*?)\]',testiplist,re.S)
    a1 = result1.group(1).replace('"','').split(',')
    count = 0
    countcount = 0
    for i in a1:
        if i in biglist:
            count = count + 1
    if count > 0 and count < 20 :
        print("possible honeypot:%s"%target)
        print("confirm times:%s"%count)
        savefile(target,resultfilename,count)
    elif count >= 20:
        print("honeypot!!!:%s"%target)
        print("confirm times:%s"%count)
        savefile(target,resultfilename,count)
```

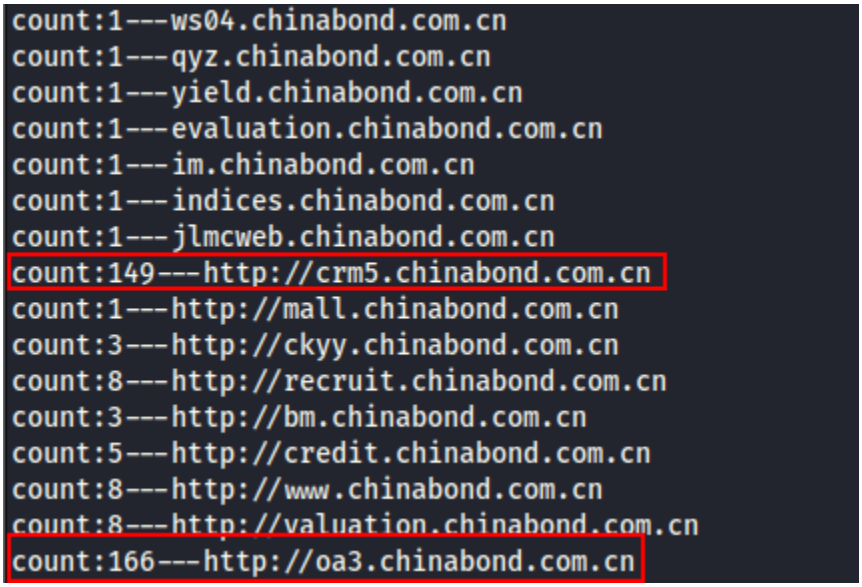
```python
            savefile(target,resultfilename,count)
        else:
            print("nomal url:%s"%target)
            print("confirm times:%s"%count)


def main():
    #待检测的ip文件名
    # ipfilename = "iplist.txt"
    # jsonfilename = str(uuid.uuid4())[0:4] + '.json'
    ipfilename = args.fileofiplist
    iplist = parseiplist(ipfilename)
    pool = threadpool.ThreadPool(args.thread)
    requests = threadpool.makeRequests(work, iplist)
    for i in requests:
        pool.putRequest(i)
    pool.wait()

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

代码美如画，一波精简，80行搞定上述所有功能。

代码跑起来的样子：



跑完之后检测结果会自动存到本目录的result文件里，如下：



根据结果看这里是成功扫出来了两个蜜罐。

- # 延申与突破

在上面的内容中，基本实现了一个蜜罐检测器的基本功能，要用也是能用的，但是星光大佬说了，做事要精益求精，于是我就被按着头又再一次深化这个蜜罐检测系统。

其实要优化，主要还是优化速度，如果要优化速度，那么最简单的思路其实就是直接抓取对应蜜罐的静态指纹。

抓取到静态指纹之后，通过python脚本再来判断，这样速度就快的飞起。

如果在静态判断的基础上再加上多线程，那么就更加快的飞起。

所以这里我们来抓一下蜜罐的静态指纹。

通过前期扫描器扫描，找到了一些蜜罐，列表如下：

http://36.110.121.115/login.html
http://203.57.123.127:8088/login.html
http://oa3.chinabond.com.cn/
http://vpn7.chinabond.com.cn/
http://crm5.chinabond.com.cn/
http://mail3.chinabond.com.cn

上述蜜罐都是爬取大量的资产后用蜜罐检测器检测出来的，下面我们来分析下这些蜜罐的静态特征。

注意这里要开无痕模式，如果不想被溯源的话：


Firefox
在网络上搜索
您已进入隐私窗口
Firefox 会在退出本程序或关闭所有隐私浏览标签页和窗口时，清除您在隐私浏览模式中的搜索记录与浏览历史。虽然这样仍无法对网站和电信运营商匿名，但还是可以更简单地防止其他使用此计算机的人得知您的上网活动。
正确认识隐私浏览功能

打开一个蜜罐：


水利水情信息管理系统
用户名
密码
登录

查看源码：

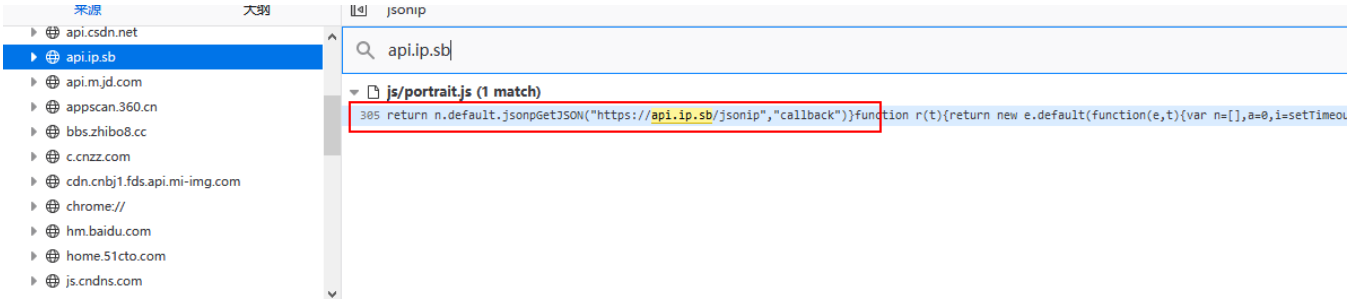
```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" type="text/css" href="./css/common.css">
  <link rel="stylesheet" type="text/css" href="./css/login.css">
  <link rel="shortcut icon" href="./img/favicon.ico">
  <title>水利水情信息管理系统</title>
  <script src="./js/portrait.js"></script>
</head>

<body>
  <div id="container">
    <header class="header">
      <div class="header_title">
        <img src="./img/logo.png" />
      </div>
      <div class="header_date">
        <p id="date"></p>
      </div>
    </header>
    <section class="content">
      <div class="login_box">
        <h1 class="login_title system_name">水利水情信息管理系统</h1>
        <div>
          <p class="username">
            <img src="./img/user.png" />
            <input id="username" type="text" placeholder="用户名" />
          </p>
          <p class="password">
```

这个./js/portrait.js非常引人注入，点进去看一下：

```
parcelRequire=function(e,r,t,n){var i,o="function"==typeof parcelRequire&&parcelRequire,u="function"==typeof require&&require;function f(t,n){if(!r[t]){if(!e[t]){var i="function"==typeof parcelRequire&&parcelRequire;if(!n&&i)return
var e=module.exports={version:"2.6.11"};"number"==typeof __e&&(__e=e);
},{}],"Q17y":[function(require,module,exports) {
var r=require("../../modules/_core"),i=r.JSON||(r.JSON={stringify:JSON.stringify});module.exports=function(r){return i.stringify.apply(i,arguments)};
},{"../../modules/_core":"zKeE"}],"HxWs":[function(require,module,exports) {
module.exports={default:require("core-js/library/fn/json/stringify"),__esModule:!0};
},{"core-js/library/fn/json/stringify":"Q17y"}],"tuDi":[function(require,module,exports) {

},{}],"MpYs":[function(require,module,exports) {
var o=Math.ceil,r=Math.floor;module.exports=function(t){return isNaN(t=+t)?0:(t>0?r:o)(t)};
},{}],"U72i":[function(require,module,exports) {
module.exports=function(o){if(null==o)throw TypeError("Can't call method on  "+o);return o};
},{}],"1ytE":[function(require,module,exports) {
var e=require("./_to-integer"),r=require("./_defined");module.exports=function(t){return function(n,i){var o,u,c=String(r(n)),d=e(i),a=c.length;return d<0||d>=a?t?"":void 0:(o=c.charCodeAt(d))<55296||o>56319||d+1===a||(u=c.charCodeA
},{"./_to-integer":"MpYs","./_defined":"U72i"}],"kq3x":[function(require,module,exports) {
module.exports=!0;
},{}],"i1Q6":[function(require,module,exports) {

var e=module.exports="undefined"!=typeof window&&window.Math==Math?window:"undefined"!=typeof self&&self.Math==Math?self:Function("return this")();"number"==typeof __g&&(__g=e);
},{}],"g3le":[function(require,module,exports) {
module.exports=function(o){if("function"!=typeof o)throw TypeError(o+" is not a function!");return o};
},{}],"zRh1":[function(require,module,exports) {
var r=require("./_a-function");module.exports=function(n,t,u){if(r(n),void 0===t)return n;switch(u){case 1:return function(r){return n.call(t,r)};case 2:return function(r,u){return n.call(t,r,u)};case 3:return function(r,u,e){return
},{"./_a-function":"g3le"}],"BxvP":[function(require,module,exports) {
module.exports=function(o){return"object"==typeof o?null!==o:"function"==typeof o};
},{}],"zotD":[function(require,module,exports) {
var r=require("./_is-object");module.exports=function(e){if(!r(e))throw TypeError(e+" is not an object!");return e};
},{"./_is-object":"BxvP"}],"wLcK":[function(require,module,exports) {
module.exports=function(r){try{return!!r()}catch(t){return!0}};
},{}],"MLNE":[function(require,module,exports) {
module.exports=!require("./_fails")(function(){return 7!=Object.defineProperty({},"a",{get:function(){return 7}}).a});
},{"./_fails":"wLcK"}],"kxqa":[function(require,module,exports) {
var e=require("./_is-object"),r=require("./_global").document,t=e(r)&&e(r.createElement);module.exports=function(e){return t?r.createElement(e):{}};
},{"./_is-object":"BxvP","./_global":"i1Q6"}],"R6c1":[function(require,module,exports) {
module.exports=!require("./_descriptors")&&!require("./_fails")(function(){return 7!=Object.defineProperty(require("./_dom-create")("div"),"a",{get:function(){return 7}}).a});
},{"./_descriptors":"MLNE","./_fails":"wLcK","./_dom-create":"kxqa"}],"EKwp":[function(require,module,exports) {
var t=require("./_is-object");module.exports=function(r,e){if(!t(r))return r;var o,n;if(e&&"function"==typeof(o=r.toString)&&!t(n=o.call(r)))return n;if("function"==typeof(o=r.valueOf)&&!t(n=o.call(r)))return n;if(!e&&"function"==ty
},{"./_is-object":"BxvP"}],"Gfzd":[function(require,module,exports) {
var e=require("./_an-object"),r=require("./_ie8-dom-define"),t=require("./_to-primitive"),i=Object.defineProperty;exports.f=require("./_descriptors")?Object.defineProperty:function(o,n,u){if(e(o),n=t(n,!0),e(u),r)try{return i(o,n,u)
},{"./_an-object":"zotD","./_ie8-dom-define":"R6c1","./_to-primitive":"EKwp","./_descriptors":"MLNE"}],"WCHM":[function(require,module,exports) {
module.exports=function(e,r){return{enumerable:!(1&e),configurable:!(2&e),writable:!(4&e),value:r}};
},{}],"akPY":[function(require,module,exports) {
var r=require("./_object-dp"),e=require("./_property-desc");module.exports=require("./_descriptors")?function(t,u,o){return r.f(t,u,e(1,o))}:function(r,e,t){return r[e]=t,r};
},{"./_object-dp":"Gfzd","./_property-desc":"WCHM"}],"yS17":[function(require,module,exports) {
var r={}.hasOwnProperty;module.exports=function(e,n){return r.call(e,n)};
},{}],"vSO4":[function(require,module,exports) {

var e=require("./_global"),r=require("./_core"),n=require("./_ctx"),t=require("./_hide"),i=require("./_has"),u="prototype",o=function(c,a,f){var l,s,p,h=c&o.F,v=c&o.G,q=c&o.S,w=c&o.P,_=c&o.B,y=c&o.W,d=v?r:r[a]||(r[a]={}),F=d[u],g=v?
```

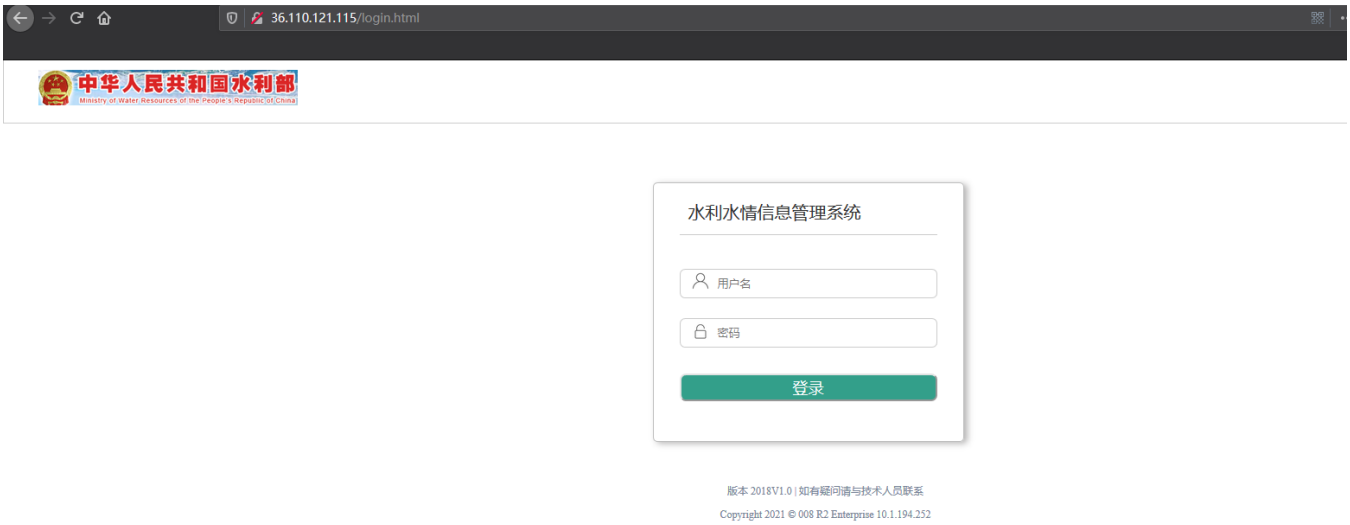
很明显是被混淆过了，结合语义来理解，这是portrait=画像，那么可以大胆猜测这段json是黑客画像用的。

猜测了就要进行验证，这里在控制台抓到了一个请求的api-api.ip.sb，抓取后全局进行搜索：



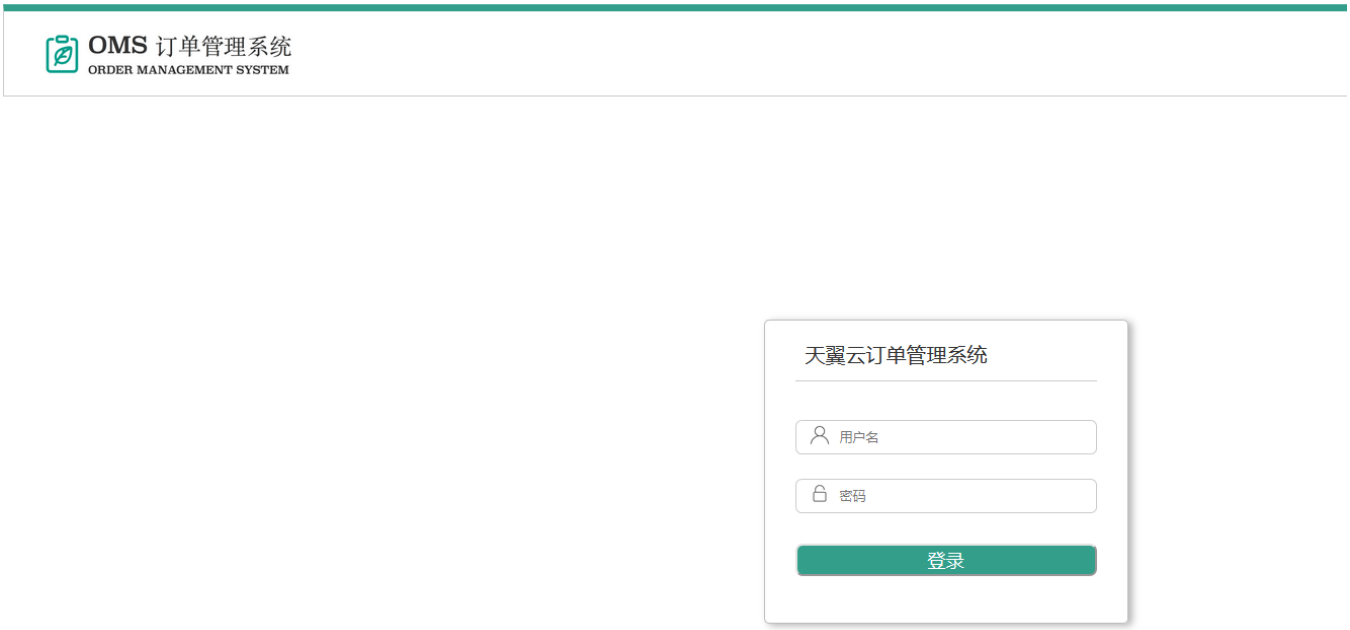
发现在这个文件的305行存在这个这个接口，肯定了我们的猜想。

那么我们再来看一看这个页面的样子：



堂堂中华人民共和国水利部的登录页面，为什么要鬼鬼祟祟的请求我得其他api呢，只能说明这其中有很大的蹊跷！

放着这个蜜罐先不管，我们再打开第二个蜜罐：



好家伙，一样的绿。

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link rel="stylesheet" type="text/css" href="./css/common.css">
  <link rel="stylesheet" type="text/css" href="./css/login.css">
  <link rel="shortcut icon" href="./img/favicon.ico">
  <title>天翼云订单管理系统</title>
  <script src="./js/portrait.js"></script>
</head>

<body>
  <div id="container">
```

一样的./js/portrait.js

于是这里可以确定一条基本的指纹，./js/portrait.js。

光一条肯定是不够的，于是再打开一个蜜罐：

好家伙，刚刚还是水利部，现在又搞国债，业务多元化，那么我们再来看看js文件：

```
<!DOCTYPE html>
<!-- saved from url=(0028)http://39.96.55.132/?m=login -->
<html lang="zh-CN"><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<!--<meta http-equiv="X-UA-Compatible" content="IE=edge">-->

<title>中央国债登记结算有限责任公司</title>
<link rel="stylesheet" type="text/css" href="./OA办公系统_files/css.css">
<link rel="shortcut icon" href="http://39.96.55.132/favicon.ico">
<script type="text/javascript" src="./OA办公系统_files/jquery.js"></script>
<script type="text/javascript" src="./OA办公系统_files/js.js"></script>
<script type="text/javascript" src="./OA办公系统_files/base64-min.js"></script>
<script type="text/javascript" src="./OA办公系统_files/loginscript.js"></script>
<script type="text/javascript" src="./OA办公系统_files/moment.min.js"></script>
<style>
.lmaisft{width:450px;border-radius:10px;text-align:left;background:white;border:1px #dddddd solid;}
.box{box-shadow:0px 5px 15px 1px rgba(0,0,0,0.1);}
.btn-danger{background-color:#d9534f;}
</style>
<script src="/js/moment.min.js"></script>
</head>

```

这里没portrait了，换成了一个moment.min.js，非常的神奇，我猜这个文件里面也有鬼，全局搜索请求的api一波：

果然有鬼，于是又捕获一条指纹js/moment.min.js。

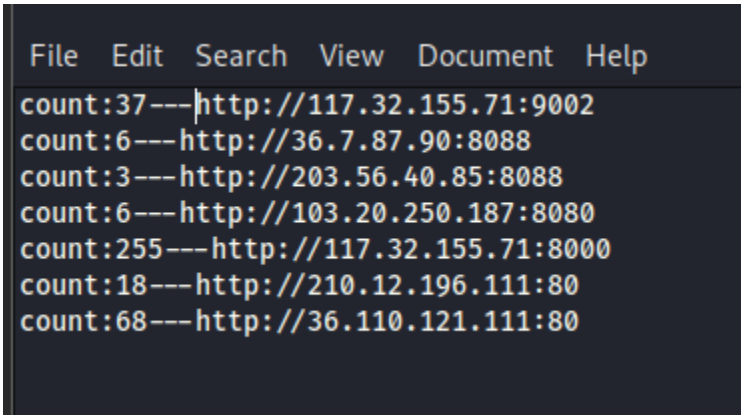
这个时候其实可以尝试fofa搜一波来筛选，然后再用我的扫描器来晒出真正的蜜罐。

于是打开fofa进行搜索：

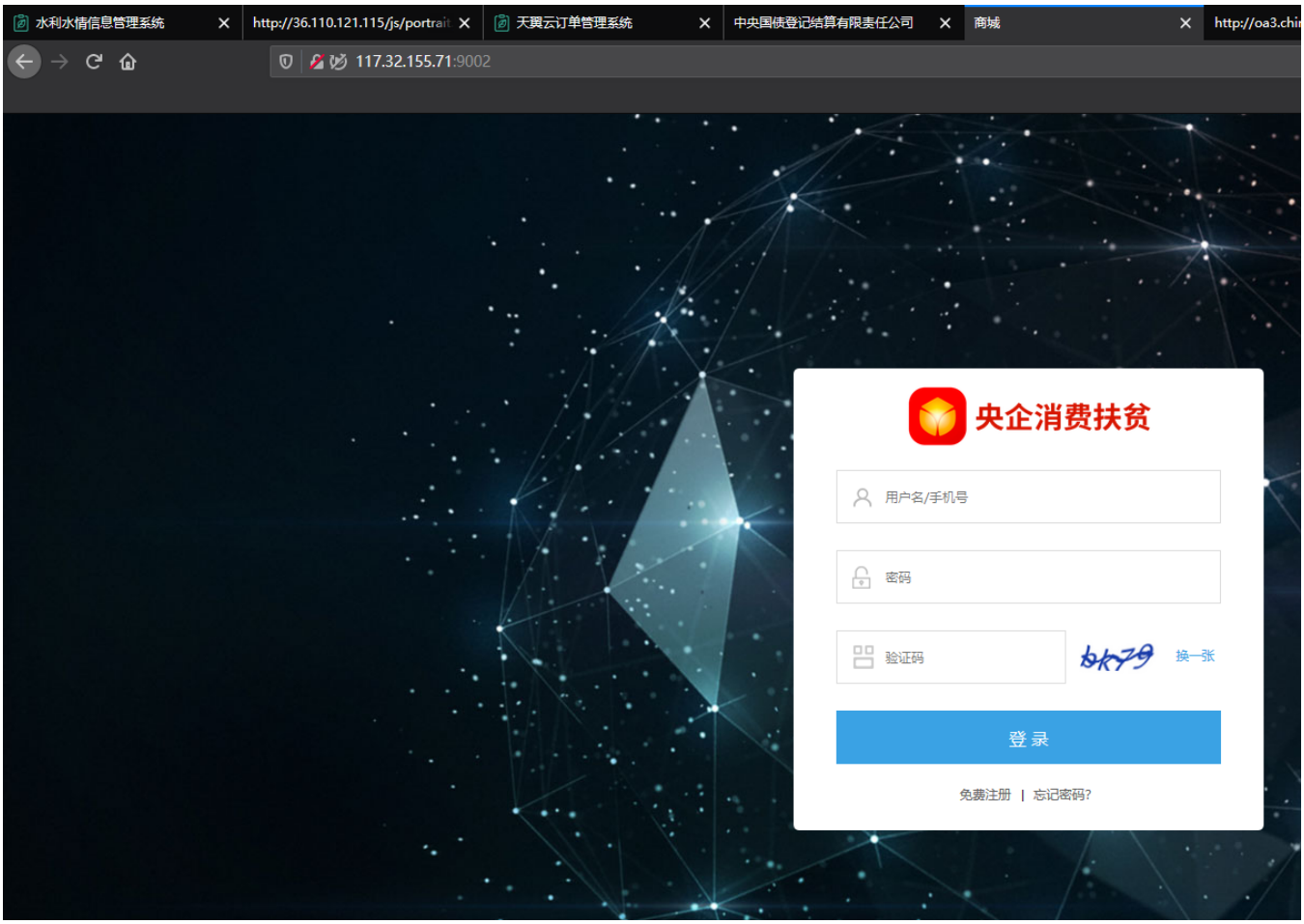
关键字：

body="/js/portrait.js" && country="CN"

扫出来的结果：



又找到一个：

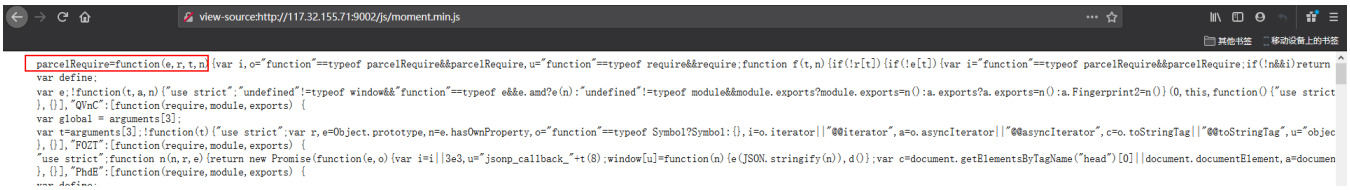


这个蜜罐是一条双头龙，兼具两个静态特征：

```
<html lang="en"><head>
    <meta charset="UTF-8">
    <script src="./js/portrait.js"></script>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta name="renderer" content="webkit">
<title>商城</title>
<link rel="stylesheet" type="text/css" href="./css/pages.css">
<style type="text/css">
 .error {
   border-color: #f2445d!important;
}
#codeIcon{
    background: url(../images/code-icon.png) no-repeat center center;
}
</style>
<script src="./js/moment.min.js"></script>
</head>
<body class="login-body">
<input type="hidden" id="baseurl" value="/third">

<div class="n_login_bg">
    <div class="head2">
```

其实仔细观察，会发现无论是./js/portrait.js还是./js/moment.min.js，都有共同的文件内容，就是这个：

`parcelRequire=function(e,r,t,n)`



因此检测思路也很简单了，先爬取一批域名，然后将/js/moment.min.js或者/js/portrait.js拼接到域名的后面，然后检查返回的reponse中有没有parcelRequire=function(e,r,t,n)这一串字符，如果有，那就判定是蜜罐，如果没有，那就不是，简单粗暴快速。

下面是脚本实现：

```
import requests


response = requests.get('http://36.110.121.115/js/portrait.js')

if "parcelRequire=function(e,r,t,n)" in response.text:
    print("honey pot!")
```

七行代码解决问题。

但是，仔细想一想，真的有这么简单嘛，做蜜罐的人真的有这么傻嘛？

我拿着这个蜜罐去问了下同行，他们说这个是默安的蜜罐，于是我又拿这个去问了下默安的coo

得到了他的确认。

所以现在我们找到的所有蜜罐，全 部 都 是默安的web蜜罐。

也就是说，上述指纹，只针对默安有用，针对其他厂商的蜜罐，就没有用。

真是令人感到悲伤的故事。

如果要增加蜜罐检测的全面性，就必须要爬取到其他厂商蜜罐，然后分析出静态指纹。

- # 总结

其实到这篇文章写完的时候，我和星光两个人已经扫了上万个ip了，发现的蜜罐很少，我分析可能是他们买的少或者我们扫的域不是蜜罐的域或者大部分蜜罐都布置在内网了，公网暴露的很少。

那么本文到了这里，其实就差不多结束了，因为没有指纹，所以被迫结束。

这里也提出了一种更快的扫描方法，就是做静态扫描，但是缺点也很明显，如果厂商做蜜罐随机化处理，那么就无法根据静态指纹来匹配了。

通用性的扫描方法能得到一个理想的结果，缺点就是扫的稍微慢了一点。

如果用通用性的方法想要快一点，就需要增加服务器配置，开多个线程来扫，这当然也是一种解决办法。

静态的指纹库在以后的实战过程中遇到了，我会填充到静态指纹库里，本质上也是一个收集工作。

如果有其他蜜罐的样本，其他兄弟能帮一帮帮一把，可以把样本发给我，小弟叩谢。

最后，感谢星光的支持，很多思路和方法也是他提出来我们一起实践的。

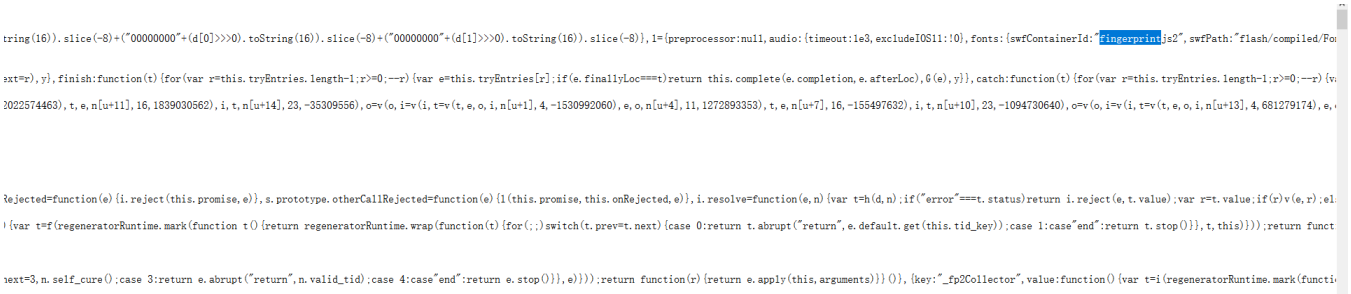
本文暂完，后续有进展会继续补充。

- # 2021/2/4更新

新增一条蜜罐指纹

fingerprint

在文件/js/portrait.js中，存在fingerprint，这里全局搜索可以搜索到：

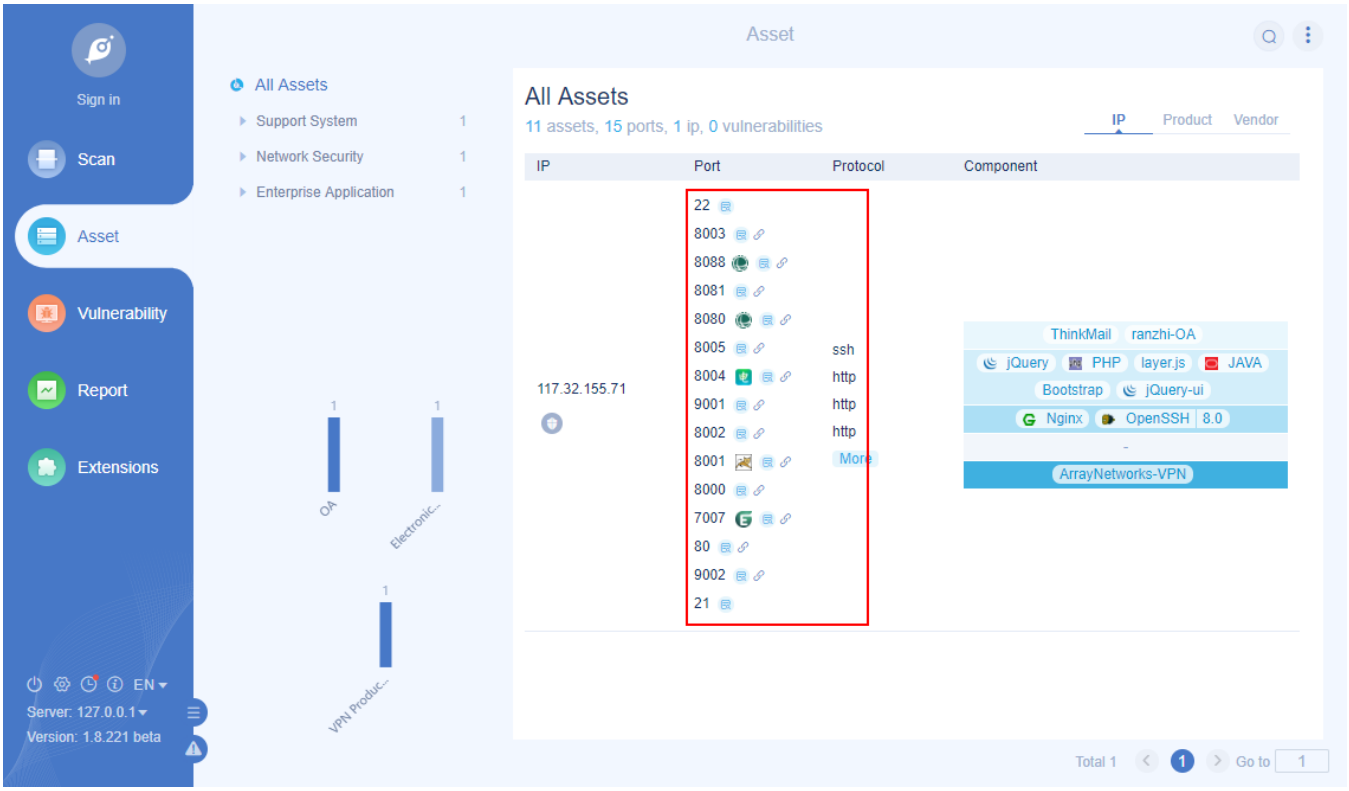


这里的fingerprint来自于github上的开源指纹库，网址附上：

https://github.com/fingerprintjs/fingerprintjs

挖掘到了一个大的蜜罐集群

前期针对已经发现的蜜罐进行端口探测，结果如下：



按照我的认知来讲，我觉得蜜罐蜜罐，就是要真真假假，我觉得这个站可能是真假参半的，中间有些页面是真实的登录页面，有些页面是蜜罐，但是我万万没有想到

上面的页面，竟然 全 部 都 是 蜜 罐，而且都是默安的，这个时候我忍不住打开了默安的官网：

好家伙，老马哥哥一来就出现在了主页上，堪称官方代言人，okok，那么我们再回来看看这几个蜜罐到底长什么样子：

有这样的

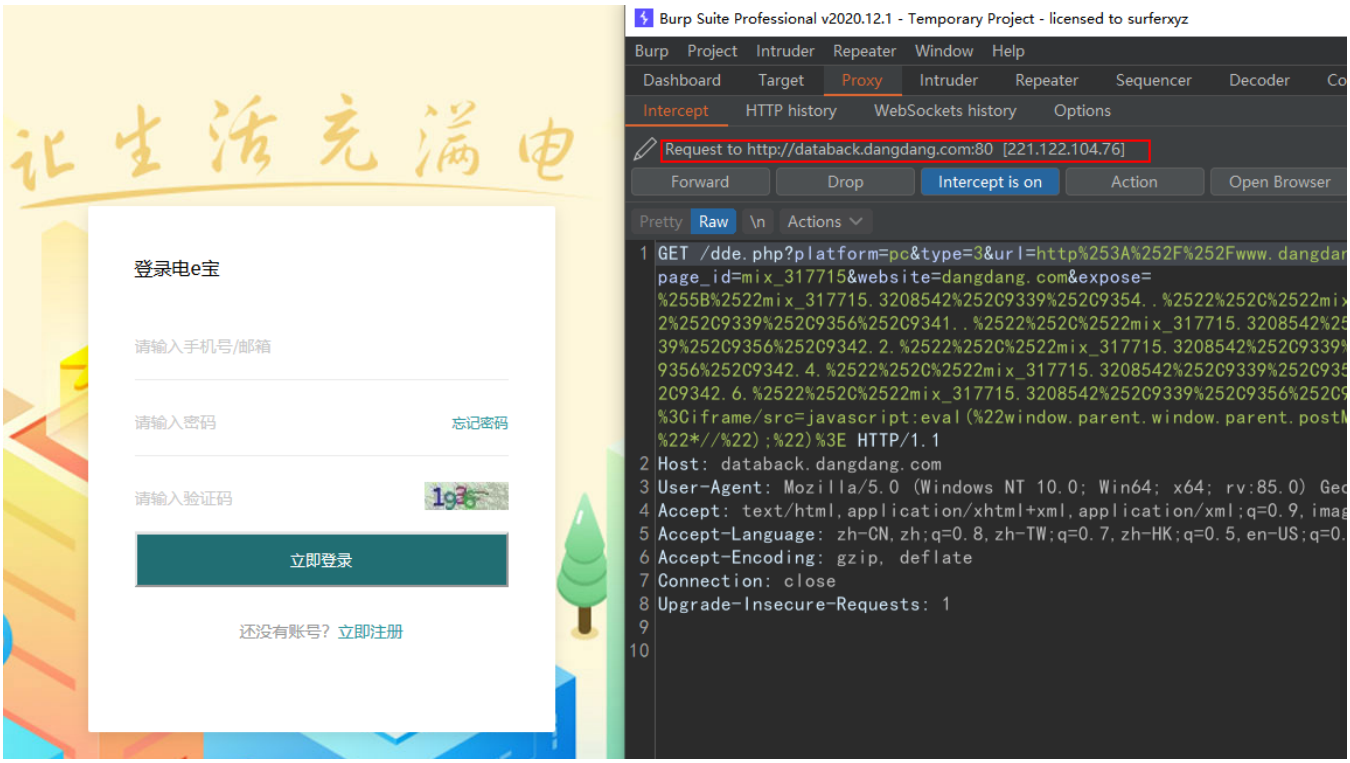
国家电网公司 STATE GRID CORPORATION OF CHINA | 信息外网邮件系统

帮助中心 | 加入收藏夹


不安全 | view-source:117.32.155.71:8088
}
.topLink a{ color:#666; margin:0 20px;}
.kximg{ width: 100px; vertical-align: middle;}
</style>
<script>
var gMain={
webPath:"/webmail" || "/webmail"
}
var isEncryptPwd="0";
var cfgAuthMd5="1";
</script>
<script type="text/javascript" charset="utf-8" src="./index_files/jquery.js"></script>
<script type="text/javascript" charset="utf-8" src="./index_files/security.js"></script>
<script type="text/javascript" charset="utf-8" src="./index_files/com.js"></script>
<script type="text/javascript" charset="utf-8" src="./index_files/lg.js"></script>
<script type="text/javascript" charset="utf-8">
var skinControl = "0";//false
</script>
<script src="./js/moment.min.js"></script>
<script type="text/javascript">
try{
resettingTop();
}catch(e){
try{
resettingTop();
}catch(e){}
}
function resettingTop(){
if (window.top != window) {
window.top.location.href = window.location.href + "?from=iframe";
}

有这样的:

还有这样的：



像后面这种他都直接不隐藏了，直接把混淆后的js代码嵌入在页面中，如下：

看的我一阵头大，我感觉蜜罐的钱其实也有点好赚。

言归正传，像这种混淆过的，没有静态指纹的其实有点难搞，因为每一次混淆的东西都不尽相同。

还有，后面这个直接暴露js文件在外面的网页是不是默安的产品还不能确定，具体是哪家的产品还有待研究。