爬虫插件系统的开发—E-Mail收集插件实列

一、实验介绍

1.1 实验内容

基于上节课的爬虫，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，本节也会写个简单的email收集插件作为列子，后面也会讲到如何写各种基于爬虫的插件。

1.2 实验知识点

- 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

1.3 实验环境

- python2.7
- Xfce终端

1.4 适合人群

本课程难度为一般，属于初级级别课程，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/shiyanlouscan_2.zip
unzip shiyanlouscan_2.zip
```

二、实验原理

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。上节课我们创造了 script 这个目录，这个目录里面存放我们编写的python插件。

三、实验步骤

3.1 __import__ 函数

我们都知道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，我们可以动态去调用函数，如果我们知道模块的名称(字符串)的时候，我们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，我们就可以简单调用一个模块的函数了。

3.2 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

1. 获取插件，通过对一个目录里的以 .py 的文件扫描得到
2. 将插件目录加入到环境变量 sys.path
3. 爬虫将扫描好的**url**和网页源码传递给插件
4. 插件工作，工作完毕后主动权还给扫描器

3.3 插件系统代码

在 lib/core/plugin.py 中创建一个**spiderplus**类，实现满足我们要求的代码。

```
#!/usr/bin/env python
# __author__= 'w8ay'
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**，这个就是我们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)
```

我们定义插件必须使用 class spider 中的 run 方法调用。

3.4 扫描器中调用插件

这里我们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，所以我们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：

```
    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
                try:
                    if(sqlcheck.sqlcheck(new_url)):
                        print("url:%s sqlcheck is valueable"%new_url)
                except:
                    pass
                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"])
                disallow = ["sqlcheck"]
                _plugin = plugin.spiderplus("script",disallow)
                _plugin.work(_str["url"],_str["html"])
                self.urls.add_new_urls(new_urls)
```



```
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。因为我们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。

3.5 sql注入融入插件系统

其实非常简单，大家修改下 script/sqlcheck.py 为下面即可：

P.S. 关于 Download 模块，其实就是 Downloader 模块(写的时候不小心写混了)，大家把 Downloader.py 复制一份命名为 Download.py 就行。

```
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.*D
river", 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"Wa
rning.*\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，只需要我们实现了 `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，引用我们自己写的下载模块.然后注释掉我们原来在扫描器中调用**sql**注入的部分就可以了。

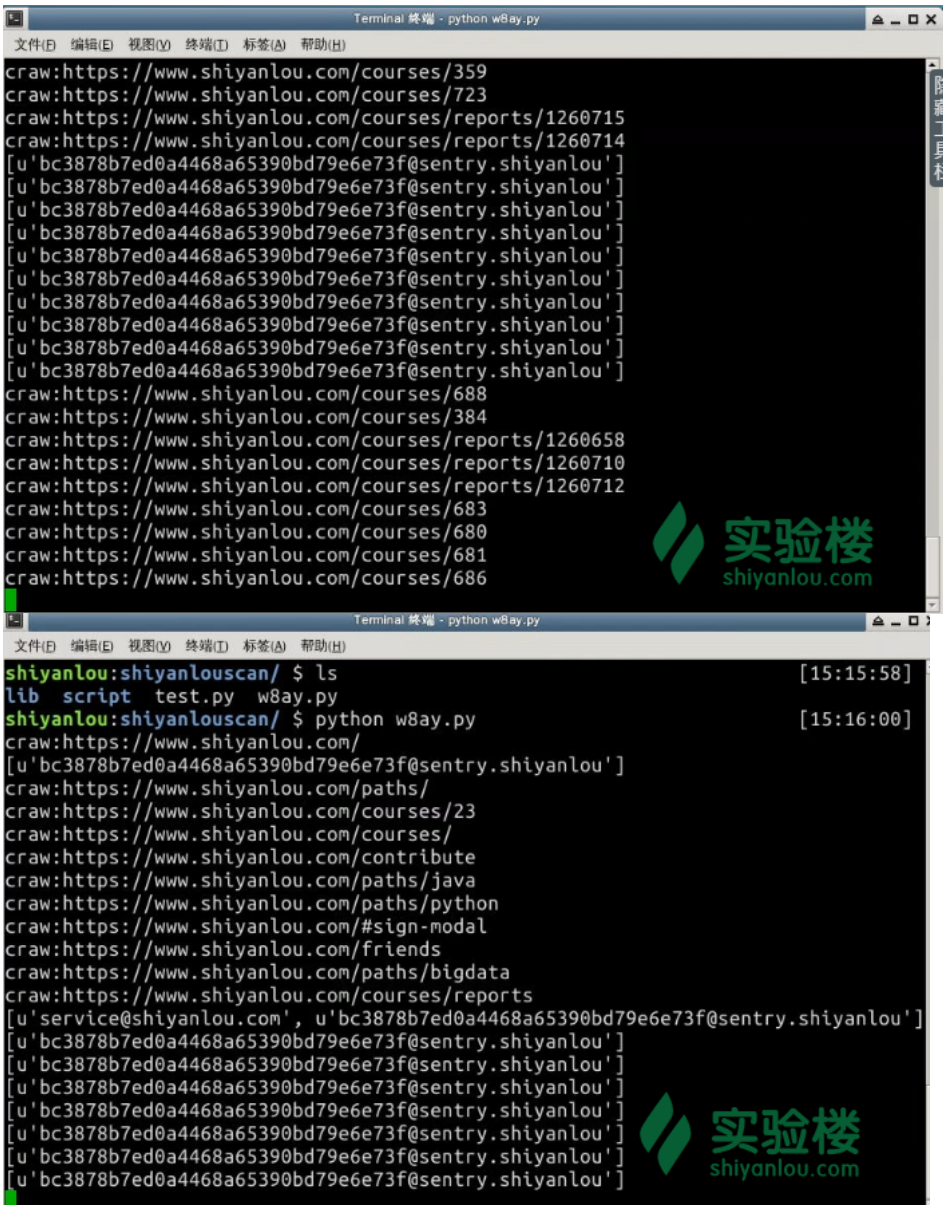
3.6 E-Mail搜索插件

然后我们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，我们用一个正则表达式 `([\w-]+@[\w-]+\.[\w-]+)+` 搜索出所有的邮件。

创建 `script/email_check.py` 文件

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

效果演示：



所有收集到的邮箱都被采集了。

四、关于插件系统踩过的坑

有一次写插件系统的时候，出现mothod no arrtibute "xxxx" 大概是这种情况，检查了好久，都找不到原因，后来找到了，是自己插件命名成email,而python有个类库就是email,所以并没有先调用我们的文件，而是先在python环境中搜寻，所以告诉我们，插件的命名最好复杂一些不要和环境中已有的冲突。

五、总结

这节我们简单实现了基于爬虫的插件系统，我们可以利用这个系统结合爬虫编写很多有趣的插件，来帮助我们进行扫描。

上一节 (/courses/761/labs/2552/document) 下一节 (/courses/761/labs/2648/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)