卧槽，sql注入竟然把我们的系统搞挂了

Original 因为热爱所以坚持ing 苏三说技术 2/6

收录于话题
#安全 1 #线上问题 3

前言

最近我在整理安全漏洞相关问题，准备在公司做一次分享。恰好，这段时间团队发现了一个sql注入漏洞：在一个公共的分页功能中，排序字段作为入参，前端页面可以自定义。在分页sql的 `mybatis mapper.xml` 中， `order by` 字段后面使用 `$` 符号动态接收计算后的排序参数，这样可以实现动态排序的功能。

但是，如果入参传入：

```
id; select 1 --
```

最终执行的sql会变成：

```
select * from user order by id; select 1 -- limit 1,20
```

`--` 会把后面的 `limit` 语句注释掉，导致分页条件失效，返回了所有数据。攻击者可以通过这个漏洞一次性获取所有数据。

动态排序这个功能原本的想法是好的，但是却有 `sql注入的风险` 。值得庆幸的是，这次我们及时发现了问题，并且及时解决了，没有造成什么损失。

但是，几年前在老东家的时候，就没那么幸运了。

一次sql注入直接把我们支付服务搞挂了。

1. 还原事故现场

有一天运营小姐姐跑过来跟我说，有很多用户支付不了。这个支付服务是一个老系统，转手了3个人了，一直很稳定没有出过啥问题。

我二话不说开始定位问题了，先看服务器日志，发现了很多报数据库连接过多的异常。因为支付功能太重要了，当时为了保证支付功能快速恢复，先找运维把支付服务2个节点重启了。

5分钟后暂时恢复了正常。

我再继续定位原因，据我当时的经验判断一般出现数据库连接过多，可能是因为 连接忘了关闭 导致。但是仔细排查代码没有发现问题，我们当时用的数据库连接池，它会自动回收空闲连接的， 排除了这种可能 。

过了会儿，又有一个节点出现了数据库连接过多的问题。

但此时，还没查到原因，逼于无奈，只能让运维再重启服务，不过这次把数据库 最大连接数调大 了，默认是100，我们当时设置的500，后面调成了1000。（其实现在大部分公司会将这个参数设置成 `1000` ）

使用命令：

```
setGLOBAL max_connections=500;
```

能及时生效，不需要重启mysql服务。

这次给我争取了更多的时间，找dba帮忙一起排查原因。

使用 `show processlist;` 命令查看当前线程执行情况：

Id	User	Host	db	Command	Time	State	Info
5	event_scheduler	localhost	(NULL)	Daemon	4588946	Waiting on empty queue	(NULL)
218	root	localhost:49637	test	Sleep	2660		(NULL)
219	root	localhost:52787	test	Query	0	starting	show processlist
220	root	localhost:52807	test	Query	2	executing	select * from test1 t1
221	root	localhost:54917	test	Sleep	27		(NULL)
222	root	localhost:54997	mysql	Sleep	369		(NULL)

还可以查看当前的连接状态帮助识别出有问题的查询语句。（需要特别说明的是上图只是我给的一个例子，线上真实的结果不是这样的）

- id 线程id
- User 执行sql的账号
- Host 执行sql的数据库的ip和端号
- db 数据库名称
- Command 执行命令，包括：Daemon、Query、Sleep等。
- Time 执行sql所消耗的时间
- State 执行状态
- info 执行信息，里面可能包含sql信息。

果然，发现了一条不寻常的查询sql，执行了差不多1个小时还没有执行完。

dba把那条sql复制出来，发给我了。然后 `kill -9` 杀掉了那条执行耗时非常长的sql线程。

后面，数据库连接过多的问题就没再出现了。

我拿到那条sql仔细分析了一下，发现一条订单查询语句被攻击者注入了很长的一段sql，肯定是高手写的，有些语法我都没见过。

但可以确认无误，被人sql注入了。

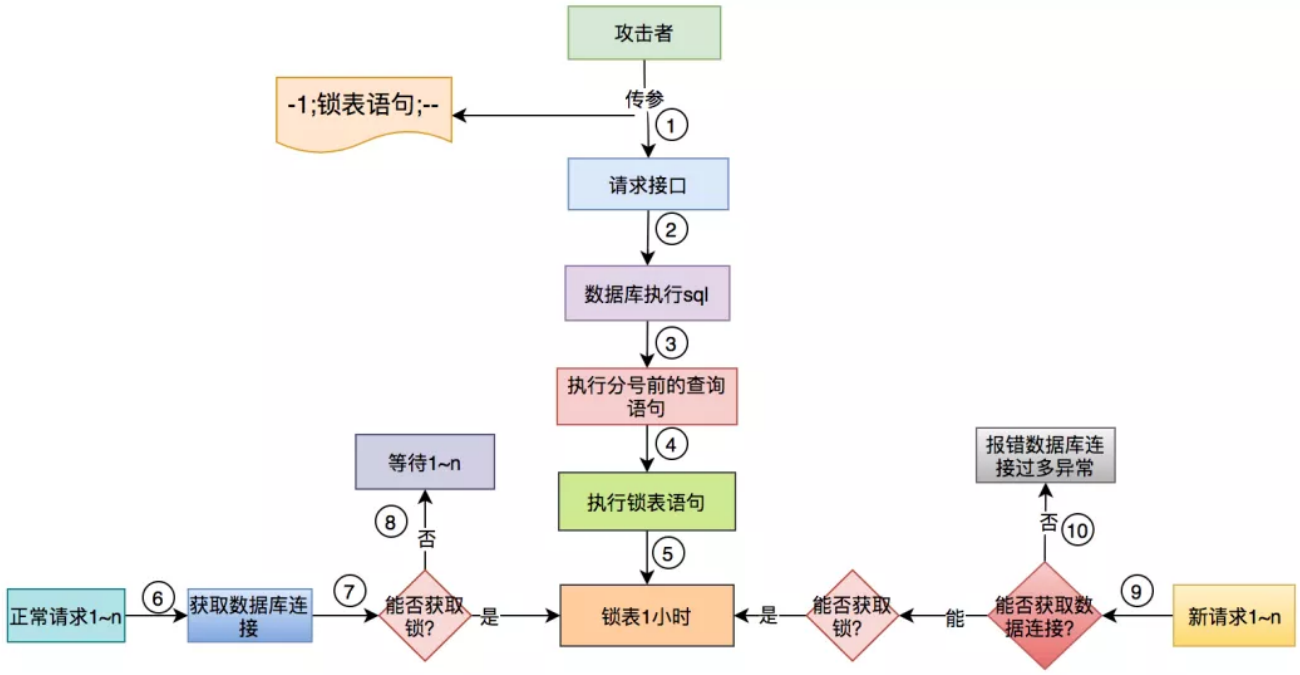
通过那条sql中的信息，我很快找到了相关代码，查询数据时入参竟然用的 `Statment` ，而非 `PrepareStatement` 预编译机制。

知道原因就好处理了，将查询数据的地方改成 `preparestatement` 预编译机制后问题得以最终解决。

2.为什么会导致数据库连接过多？

我相信很多同学看到这里，都会有一个疑问：sql注入为何会导致数据库连接过多？

我下面用一张图，给大家解释一下：



1. 攻击者sql注入了类似这样的参数： `-1;锁表语句--` 。
2. 其中 `;` 前面的查询语句先执行了。
3. 由于 `--` 后面的语句会被注释，接下来只会执行锁表语句，把表锁住。
4. 正常业务请求从数据库连接池成功获取连接后，需要操作表的时候，尝试获取表锁，但一直获取不到，直到超时。注意，这里可能会累计大量的数据库连接被占用，没有及时归还。
5. 数据库连接池不够用，没有空闲连接。
6. 新的业务请求从数据库连接池获取不到连接，报数据库连接过多异常。

sql注入导致数据库连接过多问题，最根本的原因是长时间锁表。

3.预编译为什么能防sql注入？

`preparestatement` 预编译机制会在sql语句执行前，对其进行语法分析、编译和优化，其中参数位置使用占位符 `?` 代替了。

当真正运行时，传过来的参数会被看作是一个纯文本，不会重新编译，不会被当做sql指令。

这样，即使入参传入sql注入指令如：

```
id; select 1 --
```

最终执行的sql会变成：

```
select * from user order by 'id; select 1 --' limit 1,20
```

这样就不会出现sql注入问题了。

4.预编译就一定安全？

不知道你在查询数据时有没有用过like语句，比如：查询名字中带有“苏”字的用户，就可能会用类似这样的语句查询：

```
select * from user where name like '%苏%';
```

正常情况下是没有问题的。

但有些场景下要求传入的条件是必填的，比如：name是必填的，如果注入了： `%` ，最后执行的sql会变成这样的：

```
select * from user where name like '%%%';
```

这种情况预编译机制是正常通过的，但sql的执行结果不会返回包含 % 的用户，而是返回了所有用户。

name字段必填变得没啥用了，攻击者同样可以获取用户表所有数据。

为什么会出现这个问题呢？

% 在mysql中是关键字，如果使用 like '%%%' ，该like条件会失效。

如何解决呢？

需要对 % 进行转义： \% 。

转义后的sql变成：

```
select * from user where name like '%\%%';
```

只会返回包含 % 的用户。

5.有些特殊的场景怎么办？

在java中如果使用 mybatis 作为持久化框架，在 mapper.xml 文件中，如果入参使用 # 传值，会使用预编译机制。

一般我们是这样用的：

```
<sql id="query">
   select * fromuser
   <where>
     name = #{name}
   </where>
</sql>
```

绝大多数情况下，鼓励大家使用 # 这种方式传参，更安全，效率更高。

但是有时有些特殊情况，比如：

```
<sql id="orderBy">
   order by ${sortString}
</sql>
```

sortString字段的内容是一个方法中动态计算出来的，这种情况是没法用 # ，代替 $ 的，这样程序会报错。

使用 $ 的情况就有sql注入的风险。

那么这种情况该怎办呢？

1. 自己写个util工具过滤掉所有的注入关键字，动态计算时调用该工具。
2. 如果数据源用的阿里的druid的话，可以开启filter中的wall（防火墙），它包含了防止sql注入的功能。但是有个问题，就是它默认不允许多语句同时操作，对批量更新操作也会拦截，这就需要我们自定义filter了。

6.表信息是如何泄露的？

有些细心的同学，可能会提出一个问题：在上面锁表的例子中，攻击者是如何拿到表信息的？

方法1：盲猜

就是攻击者根据常识猜测可能存在的表名称。

假设我们有这样的查询条件：

```
select * from t_order where id = ${id};
```

传入参数： `-1;select * from user`

最终执行sql变成：

```
select * from t_order where id = -1; select * from user;
```

如果该sql有数据返回，说明user表存在，被猜中了。

建议表名不要起得过于简单，可以带上适当的前缀，比如：t_user。这样可以增加盲猜的难度。

方法2：通过系统表

其实mysql有些系统表，可以查到我们自定义的数据库和表的信息。

假设我们还是以这条sql为例：

```
select code,name from t_order where id = ${id};
```

第一步，获取数据库和账号名。

传参为： `-1 union select database(),user()#`

最终执行sql变成：

```
select code,name from t_order where id = -1 union select database(),user()#
```

会返回当前 数据库名称： `sue` 和 账号名称： `root@localhost` 。

code	name
sue	root@localhost

第二步，获取表名。

传参改成： `-1 union select table_name,table_schema from information_schema.tables where table_schema='sue'#` 最终执行sql变成：

```
select code,name from t_order where id = -1 union select table_name,table_schema from informat
```

会返回数据库 `sue` 下面所有表名。

code	name
jump_log	sue
t_order	sue

建议在生成环境程序访问的数据库账号，要跟管理员账号分开，一定要控制权限，不能访问系统表。

7.sql注入到底有哪些危害？

1. 核心数据泄露

大部分攻击者的目的是为了赚钱，说白了就是获取到有价值的信息拿出去卖钱，比如：用户账号、密码、手机号、身份证信息、银行卡号、地址等敏感信息。

他们可以注入类似这样的语句：

```
-1; select * from user; --
```

就能轻松把用户表中所有信息都获取到。

所以，建议大家对这些敏感信息加密存储，可以使用 `AES` 对称加密。

2. 删库跑路

也不乏有些攻击者不按常理出牌，sql注入后直接把系统的表或者数据库都删了。

他们可以注入类似这样的语句：

```
-1; delete from user; --
```

以上语句会删掉user表中所有数据。

```
-1; drop database test; --
```

以上语句会把整个test数据库所有内容都删掉。

正常情况下，我们需要控制线上账号的权限，只允许DML（data manipulation language）数据操纵语言语句，包括：select、update、insert、delete等。

不允许DDL（data definition language）数据库定义语言语句，包含：create、alter、drop等。

也不允许DCL（Data Control Language）数据库控制语言语句，包含：grant,deny,revoke等。

DDL和DCL语句只有dba的管理员账号才能操作。

顺便提一句：如果被删表或删库了，其实还有补救措施，就是从备份文件中恢复，可能只会丢失少量实时的数据，所以一定有备份机制。

3. 把系统搞挂

有些攻击者甚至可以直接把我们的服务搞挂了，在老东家的时候就是这种情况。

他们可以注入类似这样的语句：

```
-1;锁表语句;--
```

把表长时间锁住后，可能会导致数据库连接耗尽。

这时，我们需要对数据库线程做监控，如果某条sql执行时间太长，要邮件预警。此外，合理设置数据库连接的超时时间，也能稍微缓解一下这类问题。

从上面三个方面，能看出sql注入问题的危害真的挺大的，我们一定要避免该类问题的发生，不要存着侥幸的心理。如果遇到一些不按常理出票的攻击者，一旦被攻击了，你可能会损失惨重。

8. 如何防止sql注入？

1. 使用预编译机制

尽量用预编译机制，少用字符串拼接的方式传参，它是sql注入问题的根源。

2. 要对特殊字符转义

有些特殊字符，比如： `%` 作为 `like` 语句中的参数时，要对其进行转义处理。

3. 要捕获异常

需要对所有的异常情况进行捕获，切记接口直接返回异常信息，因为有些异常信息中包含了sql信息，包括：库名，表名，字段名等。攻击者拿着这些信息，就能通过sql注入随心所欲的攻击你的数据库了。目前比较主流的做法是，有个专门的网关服务，它统一暴露对外接口。用户请求接口时先经过它，再由它将请求转发给业务服务。这样做的好处是：能统一封装返回数据的

返回体，并且如果出现异常，能返回统一的异常信息，隐藏敏感信息。此外还能做限流和权限控制。

4. 使用代码检测工具

使用sqlMap等代码检测工具，它能检测sql注入漏洞。

5. 要有监控

需要对数据库sql的执行情况进行监控，有异常情况，及时邮件或短信提醒。

6. 数据库账号需控制权限

对生产环境的数据库建立单独的账号，只分配 `DML` 相关权限，且不能访问系统表。切勿在程序中直接使用管理员账号。

7. 代码review

建立代码review机制，能找出部分隐藏的问题，提升代码质量。

8. 使用其他手段处理

对于不能使用预编译传参时，要么开启 `druid` 的 `filter` 防火墙，要么自己写代码逻辑过滤掉所有可能的注入关键字。

最后说一句(求关注，别白嫖我)

如果这篇文章对您有所帮助，或者有所启发的话，帮忙扫描下发二维码关注一下，您的支持是我坚持写作最大的动力。

求一键三连：点赞、转发、在看。

关注公众号：【苏三说技术】，在公众号中回复：面试、代码神器、开发手册、时间管理有超赞的粉丝福利，另外回复：加群，可以跟很多BAT大厂的前辈交流和学习。

个人公众号

个人微信

Modified on 2021/02/19

People who liked this content also liked

如何设计百万人抽奖系统……
苏三说技术

【热点】张昆，不要自责，你已经很棒了！
中国普法

专一的人都有哪些特征？
有趣青年