答辩讲稿

刘志威

大家下午好，本组进行汇报的论文为“DynSQL: Stateful Fuzzing for Database Management Systems with Complex and Valid SQL Query Generation”。**（翻页）**

以下是本组成员的分工情况。**（翻页）**

我们将从目标与动机、方法与效果、思考与未来工作三个方面来完成我们的汇报。**（翻页）**

首先是目标与动机。**（翻页）**

如图显示了一个恶意查询，它会使MariaDB服务器崩溃，并启用拒绝服务(DoS)攻击。这是一个经过简化后，并被认为是最小的测试用例，但依旧可以看到这个查询语句是十分复杂的。我们使用现有的DBMS工具进行模糊测试，很难生成类似的查询。**（翻页）**

因为现有的DBMS模糊器严重依赖于它们预定义的语法模型和关于DBMS的固定知识，不能准确捕获由生成的语句引起的DBMS状态更改。它们要么忽略状态变化例如SQLsmith，要么静态地推断相应的状态，例如SQUIRREL，但会受到可靠性和完整性问题的影响。**（翻页）**

没有准确的DBMS状态信息，现有的模糊器在生成复杂和有效的查询以广泛测试DBMS方面受到限制，导致许多深层错误仍然存在。因此，提出解决这些限制的实际解决方案对于DBMS模糊测试是必要和重要的。而这篇文章提出了一种新的有状态模糊方法来解决现有DBMS模糊器的局限性。该方法执行动态查询交互，合并查询生成和查询执行，以有效地生成复杂而有效的SQL查询。此外，它来使用错误反馈来提高生成查询的有效性。接下来由我们组的另一位同学来进行更详细的方法与效果的汇报。**（翻页）**

陆皓喆

下面由我来为大家介绍此论文的方法与效果部分。**（翻页）**

本文所使用的方法有很多，概括来说，就是在DynSQL算法总流程中引入状态模糊测试来对文件进行分析，在状态模糊测试中包含了两个关键步骤——动态查询交互和错误反馈，下面我来具体介绍一下这些算法流程。

首先是状态模糊测试。首先，我们将初始的输入传送到我们的种子池中，从种子池中筛选出符合要求的种子，进行种子变异，并生成对应的随机文件。然后我们进入核心步骤：动态查询交互，该部分中Translator和Scheduler起到了关键作用。下面的一步——错误反馈帮助我们过滤掉了部分无效测试用例来提升我们的查询有效性。我们的反馈器会将那些系统认为有效的种子返回到池中供下一轮使用。**（翻页）**

然后是动态查询交互。该算法主要涉及到三个函数，Scheduler函数通过输入file和DBMS，利用Translator函数来生成SQL语句并执行，最后再通过CheckStatus函数来检查当前的状态。Translator函数的主要功能就是生成对应的SQL语句，涉及到AST抽象语法树，此处不再赘述。CheckStatus函数则是用于实时检查查询状态，根据不同的情况来输入true或者false。**（翻页）**

然后是错误反馈。由于我们不需要那些无效的查询步骤，所以我们需要一种过滤机制来帮助我们进行筛选，也就是作者引入的错误反馈机制。首先我们检查代码覆盖率是否上升，如果没有上升就直接丢弃；如果上升了就再进一步检查有没有触发错误，若没有触发，就放入种子池备用。该算法确保了我们池中的查询都是有效的。**（翻页）**

最后我们简单看一下DynSQL的总流程。首先，我们输入目标DBMS的源文件，进行编译得到可执行程序，Fuzzer模块执行传统的文件模糊测试（如AFL）以基于给定的种子生成文件Files，传输到动态交互查询器中进行交互查询。Generator模块负责利用AST模型来提供正确的SQL语句。交互查询后，将所得信息提供给Bug checker输出bug报告，同时通过错误反馈机制更新池中的种子。**（翻页）**

下面我们来介绍一下实验效果部分。作者共对四个问题进行了实验验证，均获得了理想的结果。**（翻页）**

实验发现，DySQL在bug的查找和查询有效性、语句有效性中均表现优异，说明该算法可以生成有效且复杂的SQL语句去挖掘DBMS中的bug。**（翻页）**

其中，挖掘到的bug大部分都是由2-3个语句，甚至4个语句触发的；触发bug的查询大小也是相对较大的，有部分已经达到了1000字节以上。这些特征对于当下主流模糊器都是难以做到的。**（翻页）**

有关于40个bug的分布如下所示。可以看出CREATE TABLE和SELECT语句占到了大部分。**（翻页）**

发现的bug共有40个，其中空指针取消引用就占到了45%。有许多发现的bug都获得了CVE漏洞的标签，此处不再赘述。**（翻页）**

为了探究动态查询交互和错误反馈在bug查询中的作用，作者分别禁用两种技术来进行漏洞挖掘，发现从全部禁用到启用的过程中，bug数量和代码覆盖率都逐步上升，且我认为动态查询交互起到了关键作用。**（翻页）**

左图是代码覆盖率的实验效果图，可以看出通过这两个步骤很大地提升了我们的代码覆盖率。当然两者各有自己的优势，在bug查询中都起到了很大的作用。**（翻页）**

与其他模糊器相比，DynSQL比SQUIRREL生成的语句更复杂；DynSQL比SQLsmith生成的语句长度更长；所以代码覆盖率最广，发现了更多的bug。**（翻页）**

SQUIRREL和SQLsmith各有自己的优点，但是也有不足之处，比如说SQLsmith 在每个查询中只能生成一个语句；SQUIRREL 无法生成逻辑复杂的语句，这就导致了两种查询结果各有遗漏。我们的DynSQL不仅发现了另外两者发现的所有bug，还找出20个通过复杂语句生成的bug，这对于SQUIRREL和SQLsmith来说几乎不可能完成。**（翻页）**

侯博文