paper

摘要

通过捕获到的状态信息，逐步生成复杂有效的SQL查询。另外利用之前查询执行的错误信息去过滤掉无效的测试案例以进一步提高生成的查询有效性，将上述思想通过一个DynSQL全自动模糊测试框架来实现。在6个DBMS上进行评估获取更好的效果。

简介

DBMSs和Fuzzing（生成SQL语句进行测试检测漏洞），这里的Fuzzing会应用到一个生成种子等技术，DBMS模糊器会在复杂性和有效性之间做出权衡。有效性和复杂性之间的矛盾（原因：这些模糊其严重依赖预定义的语法模型和DBMS的固定知识，没有动态的去获取查询信息和状态变化并依此进行生成查询）

主要实现内容和技术创新点

开发出新的有状态模糊方法（合并查询生成和查询执行，并通过错误反馈提高生成查询的有效性），根据方法实现DynSQL，评估效果。

背景和动机

SQL处理（解析：检查正确性、优化、评估、执行）、增加复杂性会使保证有效性难度变大、生成复杂而有效的SQL查询来检测DBMS中的深层错误是非常重要的。
现有模糊器局限性：**不能准确捕获实时的DBMS状态。**其中主流有两种方法第一种：无状态的生成一个复杂语句，避免对状态进行分析（例子：SQLsmith利用预定义的AST进行复杂查询生成，但是是无状态）；第二种：结合多个简单查询，可以轻松判断状态（例子：SQUIRREL，使用IR维护查询结构，可以判断状态，但在判断复杂语句状态的时候准确度低，生成的无效查询太多），因此掌握生成复杂而有效的查询从而检测深层错误是很有必要的。
有状态的DBMS模糊测试：与目标DBMS进行动态交互来获取实时状态，核心是动态查询交互（合并查询生成和查询执行），同时利用错误反馈过滤掉种子池中无效的测试案例。
动态查询互动：

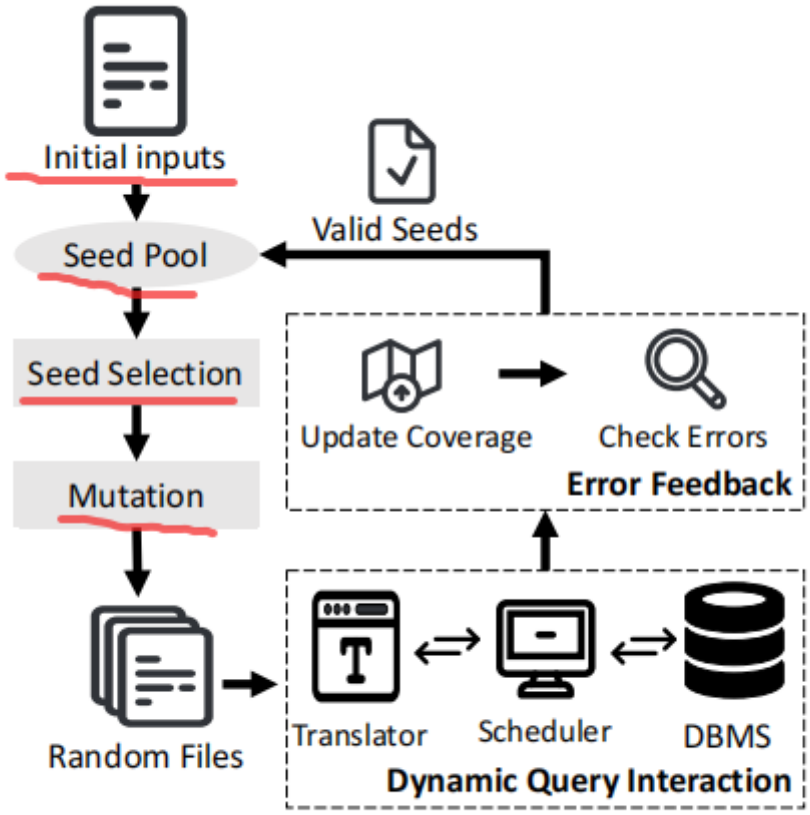

Figure 2: Overview of stateful DBMS fuzzing.

对上图的解释：初始投入放入种子池中，从种子池中选取种子进行变异形成随机文件，然后是translator，translator在接收到scheduler的参数之后会去读取随机文件中未被读取的部分存到临时文件中，然后translator会使用内置的StmtGenerator根据提供的schema和临时文件生成一个SQL查询，然后translator会返回这个SQL查询并更新StmtGenerator读取过的字节，这里的StmtGenerator使用AST模型进行生成，但特别的是，他利用临时文件作为随机种子（具体来说就是通过临时文件的值去在遍历抽象语法树的时候去选择路径）然后讲解scheduler，它首先会初始化使用的变量（文件size、目标DBMS、读取字节rb、查询、覆盖率），然后开始循环读取字节进行查询，获取数据库模式，然后将这些参数发送给translator，然后translator会将上文说到的SQL查询和更新的rb返回到scheduler，然后scheduler把SQL查询发送到DBMS执行，执行完后scheduler收集覆盖分支、查询语句处理状态信息，如果语句崩溃或者有错误，scheduler会退出循环。当循环结束时，scheduler会返回生成的查询、代码覆盖率和查询处理的末状态。这里还会根据覆盖率和错误信息选取更有效的种子和过滤掉无效种子。

举例：SQL语句生成过程，分为三个阶段：读取未访问的输入文件，根据数据库模式和读取的字节生成SQL查询，返回查询并记录读取到的字节。例如对论文中图1的第三个SQL查询，当translator把未读取的输入文件传入stmtgenerator后，stmtgenerator会首先确定生成语句的类型，然后根据数据库模式和这个文件遍历AST生成SQL查询，例如读取到了一个值为5的字节，就要用CTE进行select，读取到1，就要用cross join去构造CTE，根据读取的值3去选择cross join中用右表作为现有的表。然后现在有v1和t1两个表作为候选表，根据读取到的值9，9mod2=1，所以用第二个v1作为进行查询的表，后面过程类似。

状态检查：scheduler在语句输入DBMS进行执行的时候会检查DBMS的状态，如果存在错误或者

崩溃，对于错误会进一步的检查错误的原因等，无论是错误还是崩溃，只要出现了，scheduler都会终止交互。

举例：论文图片1中的整个查询生成和检测过程：测试开始，首先scheduler会查询DBMS的模式，初始是空模式，然后将空模式发给translator，然后translator用内置的stmtgenerator根据空模式和从文件读取的值遍历AST model去生成了一个create table语句，然后把这个语句传给scheduler，然后scheduler发给DBMS执行，然后scheduler获取模式，这是包含一个表的模式，再次发给tranlator，translator首先更新文件（删掉之前被读取的部分，继续读取未被读取的部分），然后根据新的数据库模式和更新后的文件生成了一个引用t1表的create view语句，然后发送给scheduler，scheduler发送给DBMS进行执行同时收集信息，但是这次也不会发生异常，于是把收集到的数据库模式再次返回给translator，现在的模式中有一个t1表，有一个有两列属性的v1视图，于是translator再次使用内置的stmtgenerator去生成了一个select语句，然后发送给scheduler，然后scheduler发送给DBMS执行，但是这次执行触发了崩溃，scheduler终止交互，并报告整个过程中生成的SQL查询。

错误反馈机制：完成想要有效的输入文件的目标，这里要解决的问题是：当一个无效查询触发一个语法或语义错误的时候，代码覆盖率增加，但是基于这个生成的变异查询可能会遇到相同的的错误，这样会使覆盖率停滞不前。于是利用错误反馈来过滤掉种子池中无效查询的输入文件。实现过程是对于每个增加覆盖率的SQL查询的输入文件，会进一步去检查在DBMS中执行的时候是否出现异常，如果有异常这个输入文件会被放弃用于种子池，这样可以保证种子的有效性。

框架和实施

基于上述的方法，开发出DynSQL框架，通过生成复杂有效的查询来检测DBMS中的深层错误。用Clang来编译和检测DBMS，共六个模块，框架图如下：

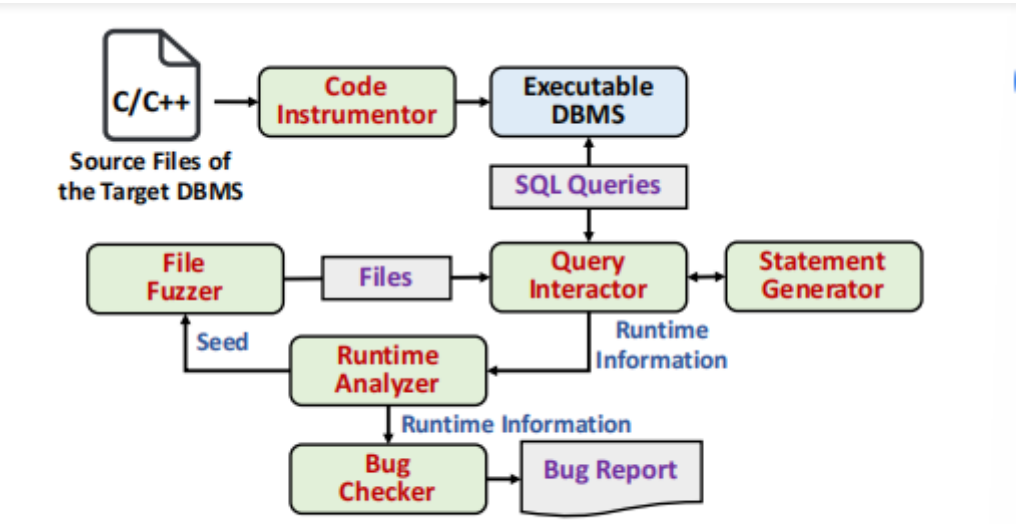

Figure 6: Overall architecture of DynSQL.

首先是Code Instrumentor，编译分析DBMS代码，生成可执行程序来接收和执行SQL查询；Query interactor：接受输入文件，执行动态查询互动来生成复杂有效查询，同时也会收集必要的信息来便于动态分析；Statement generator：利用ASTmodel生成查询，只引用数据库模式信息中有的数据；Runtime analyzer：分析收集的运算时间信息，并根据错误反馈来过滤筛选下次模糊测试的种子；File fuzzer：类似传统文件模糊，基于给定种子生成文件，这个模块是用AFL实

现的；Bug checker：根据收集的运行时间信息检测错误并生成相应的错误bug报告。
DynSQL重要细节的介绍：
DBMS支持：提供相当多的API用来外部程序对DBMS进行测试。
SQL查询生成：基于ASR模型实现，参考了SQLsimth的实现，并格外添加了一个SQL特性（group by、union等），并固定了SQL特性的通用部分，将有针对性的SQL特性作为可选特性，在针对不同DBMS进行测试的时候，只需要根据对应的DBMS官方文档进行对相应的SQL特性启用即可。
bug检测：用ASan作为检测器来检测内存错误，另外利用动态收集到的SQL执行异常信息检测导致DBMS报错的错误。
查询最小化：对那些出发新错误的SQL查询进行最小化处理，以便于复现和定位DBMS的错误，这里的实现参考了APOLLO和C-Reduce，有时候还需要开发者的帮助。（这里所为最小化的意思就是简化目标SQl查询的结构，在保持具有相同错误出发能力的同时，尽可能简化其结构）

评估测试

主要针对以下几个方面进行测试：
1.是否可以生成复杂有效的查询
2.DynSQL检测出的bug的安全影响程度
3.新开发的动态查询交互和错误反馈机制的效果
4.对比其他最先进的DBMS模糊器的效果
对象：6个主流DBMS：SQLite,、MySQL、MariaDB、PostgreSQL、MonetDB、ClickHouse。
运行时间测试：
对每个DBMS进行五次模糊测试并计算平均值来获取结果，将模糊测试超时时间设置为24小时，观察到24小时之后的分支覆盖率和发现bug基本收敛和SQUIRREL观察结果一致。
paper中的表三含义
生成的查询和语句：每个查询平均包含的语句是8.6个，其中无效语句的原因是使用了不符合数据类型约束或者完整性约束的复杂表达式。图7说明了触发错误查询所要用到的语句个数。
错误触发查询的大小，字节数递增发现错误递增，具体数据见图8。
出发错误的语句中包含特定语句的百分比，见图9.
安全影响：18个是空指针解引用错误，可以被利用来进行拒绝服务攻击；7个严重内存错误（2个使用后释放错误，2个堆栈缓冲区溢出错误，1个整数溢出错误，2个堆缓冲区溢出错误），可以进行权限提升和信息泄露；6个断言失败错误；还有9个语义错误，其中19个被分配了CVE ID。
三个案例：
1.MariaDB中的整数溢出。这个错误被识别为严重漏洞，它允许攻击者在内存空间中写入和读取任意数据，图10展示部分代码。过程是计算select项目数时发生整数溢出，于是分配了远远不够的数组，进一步出发堆缓冲区溢出，可以进行dword shoot攻击。修复bug：扩大表达范围，并利用断言增加判断机制。
2.在MariaDB中的使用后释放错误，产生原因是错误使用rollback机制引起的，MariaDB服务器会将语句引起的更改存在一个例表中，成功则全部删除，会对列表进行检查，如果没空代表执行失

败需要进行rollback。在bug出发过程中，服务器释放了更改内容但是没删除，导致错误回滚，该bug被利用来使用已释放的数据覆盖任意地址上的数据。
3.MonetDB中的子查询结果缺失错误，触发bug的查询包含一个CREATE TABLE语句和一个带有CTE的SELECT语句，原因是由于错误的优化引起的。
敏感性分析：为了理解动态查询交互和错误反馈的贡献，进行了敏感性分析。设计了DynSQL!DQI、DynSQL!EF和DynSQL!DQI!EF。在DynSQL!DQI中，我们只禁用了动态查询交互；在DynSQL!EF中，我们只禁用了错误反馈；在DynSQL!DQI!EF中，我们同时禁用了动态查询交互和错误反馈。表5列出了测试数据。
运行开销：错误反馈的开销很小，但是动态查询交互可能会引起开销，但是由于DBMS一般会提供有效方法所以开销很小。这部分的对比数据在表6，总体来说超过95%的时间用于查询执行。
代码覆盖率：错误反馈和动态查询交互会提高覆盖率。
错误检测：错误检测可以帮助模糊器生成更多有效查询，但是不会影响查询复杂性，动态查询交互提高了复杂性和有效性。
与现有的DBMS模糊器比较：SQLsmith和SQUIRREL，其中SQLsmith生成的查询都是单条语句，不能生成多个语句的查询，SQUIRREL可生成多状态的查询，但是无效查询很多。这个方面，DynSQL可以生成更多的包含多个语句的查询。
覆盖率：观察图14
错误检测：DynSQL发现了所有另外两个模糊器发现的bug还多发现了20个bug。
查询的复杂性：DynSQL可以生成另外两个所有错误出发查询，但是DynSQL还能另外生成更加复杂语句查询，出发更深层错误。

局限与未来

无效查询：主要由于违反约束条件造成的，数据类型约束，数据范围完整性约束。准备利用SAT来生成表达式
其他bug：逻辑错误（返回错误结果）和性能错误（减慢性能）
AST规则构建：计划利用机器学习技术从有效查询中自动提取AST规则

相关工作

DBMS测试特定bug：SQLlancer检测逻辑错误等等，这些产生具有特定模式的测试用例，限制了检测其他类型错误的可能性。
DBMS常见bug：RAGS随机生成，送入多个DBMS，比对返回结果，不同则报错。有缺点：DBMS支持的SQL功能共享集很少，而且生成随机。
通用的模糊测试：Angoral使用污点分析跟踪然后利用梯度下降搜索满足约束的合适值。QSYM使用符号执行，降低符号执行开销。但不行，没有针对性，无效太多。
DBMS的模糊测试：略上文有。

总结

在本论文中，我们开发了一个实用的DBMS模糊测试框架，名为DynSQL，可以有效地生成复杂且有效的SQL查询，以检测DBMS中的深层次错误。DynSQL集成了一种新颖的技术，即动态查询交互，用于捕获准确的DBMS状态信息并提升查询生成效果。此外，DynSQL还使用错误反馈来进一步提高生成查询的有效性。我们已经在6个广泛使用的DBMS上对DynSQL进行了评估，并发现了40个独特的错误。我们还将DynSQL与最先进的DBMS模糊测试工具进行了比较，结果表明DynSQL在具有更高代码覆盖率的DBMS中发现了更多的错误。