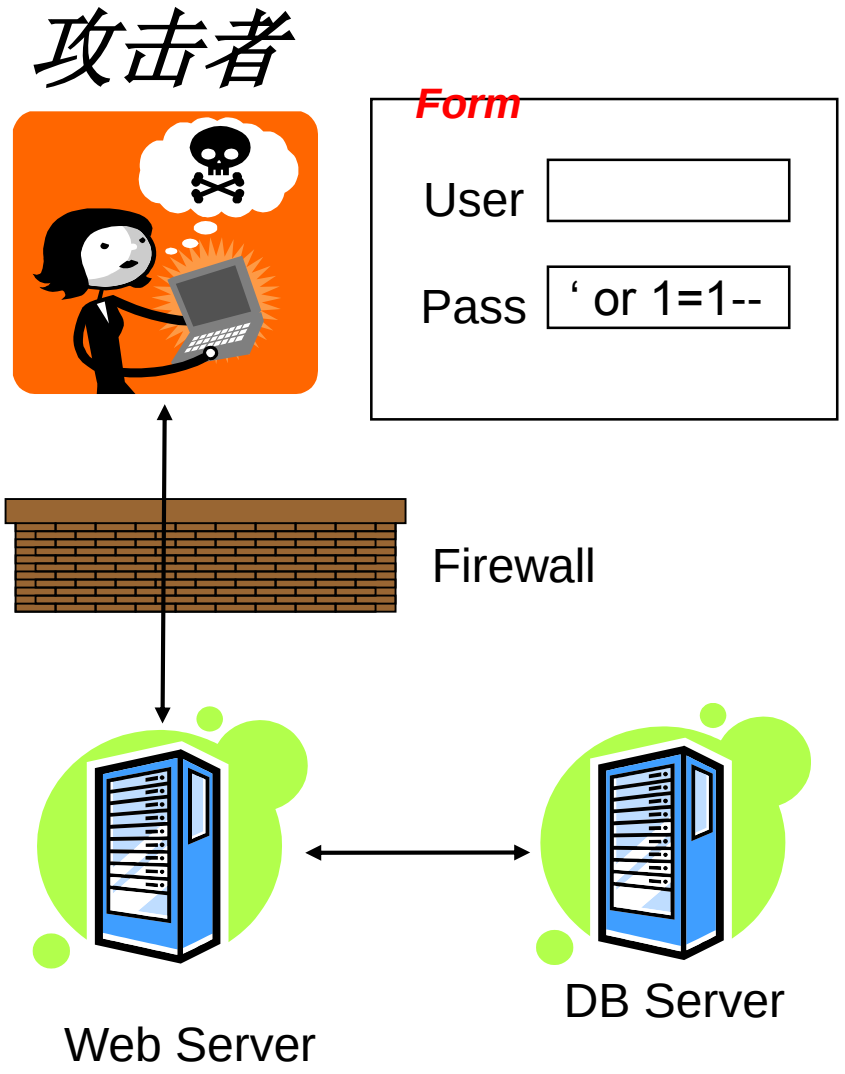
揭开SQL注入的神秘面纱

IT运维中心数据库管理部 宋沄剑

2015.5.7

1. APP将表格发送给用户.
2. 攻击者将带有SQL注入的参数发送给WEB服务器.
3. APP利用用户输入的数据构建SQL串.
4. APP将SQL发送给DB.
5. DB执行了被注入的SQL，返回结果给APP.
6. APP将数据返回给用户.



SQL注入？

可信部分

```
http://www.mysite.com/Widget?Id= 1

SELECT * FROM Widget WHERE ID =  1
```

不可信部分

It's not just for SQL

议程

- 什么是SQL注入
- SQL注入的种类
- SQL注入工具
- 如何防范SQL注入

什么是SQL注入

- SQL 注入是一种代码注入技术，用于攻击基于数据库的应用，基本原理是将SQL语句插入到参数位置执行。SQL 注入通过应用软件的安全漏洞（比如说用户输入的特殊字符没有被转义、或用户输入不是强类型导致意外执行）

- 本质是用户输入作为SQL命令被执行

SQL注入的危害

- 在数据库层面可以执行权限范围内的任意操作
- 绕过身份验证机制
- 读取本不应该读取的信息（传说中的开房库）（CSDN明文存用户密码）
- 修改数据库
- 数据丢失、数据库拒绝服务（由于没有经济利益，通常黑客不会这么做）
- 入侵数据库所在服务器（通过xp_cmdshell提权）
- 损害公司或者产品形象

SQL注入的现实例子-请假条

请假条

领导：

因 XX 事，需要请假 3 天。

请假人：XXX　　领导：XXX

请假条

领导：

因 XX 事，需要请假 3天，另外，因为 XX 事情，还需要额外请假 180天。

请假人：XXX　　领导：XXX

被注入的代码片段

请假条-防注入方法1-强类型（参数化）

请假条

请假理由（Varchar）	XX 事
请假天数（Int）	3

请假人：XXX

天数只能是Int类型，无法被注入

领导：XXX

请假条-防注入方法2-特殊字符过滤

请假条

领导：

因 XX 事，需要请假 3 天##另外##因为 XX 事情##还需要额外请假 180 天。

请假人：XXX　　领导：XXX

逗号被转义为##，语法不通，报错从而拒绝执行

议程

- 什么是SQL注入
- SQL注入的种类
- SQL注入工具
- 如何防范SQL注入

SELECT语句注入点

- 常见SQL注入点

  SELECT columns
  FROM table
  WHERE expression
  ORDER BY expression

- 注入点最可能位置
  - WHERE 之后
  - ORDER BY 之后
  - 表名称和列名称位置

Insert语句注入点

- 插入语句

  INSERT INTO *table* (*col1*, *col2*, ...)

  VALUES (*val1*, *val2*, ...)

- 要求
  - 插入列数必须和值一致
  - 值数据类型必须和指定列一致
- 手段：不断加输入值，直道不再报错

  foo’)--

  foo’ , 1)--

  foo’ , 1, 1)--

Update注入点

- Update语句

  UPDATE *table*

  SET *col1=val1, col2=val2, ...*

  WHERE *expression*

- 注入点
  - SET 子句
  - WHERE 子句

- 请小心Where子句注入，黑客在尝试过程就可能导致：
  - ’ OR 1=1 将会是灾难

基于批处理的SQL注入

par=1; SQL query;--

基于Union的注入（Union-Based Injection）

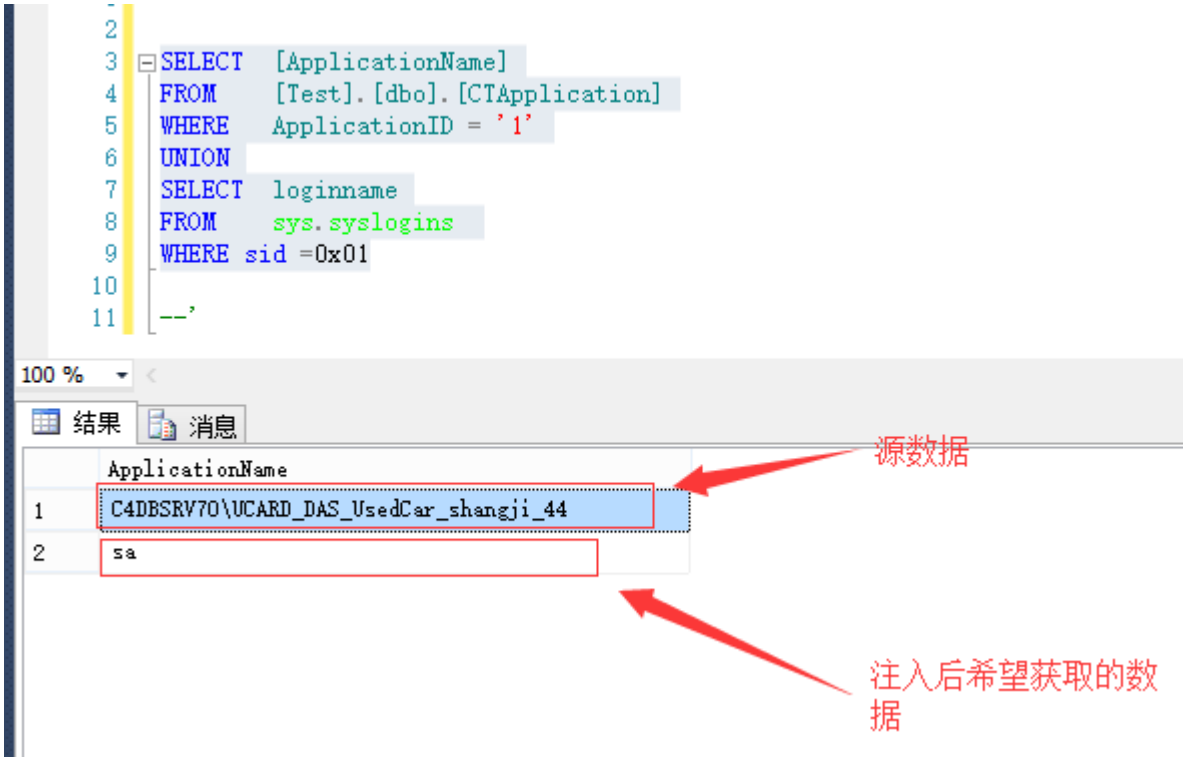
- 将Select合并为一个结果
  - SELECT cols FROM table WHERE expr
  - UNION
  - SELECT cols2 FROM table2 WHERE expr2
- 允许攻击者读取任意表
  - foo’ UNION SELECT number FROM cc--
- 要求
  - 结果集必须是同样的列数和同样数据类型
  - 攻击者必须知道表名称
  - DB返回的列名是第一个查询的列名称

基于Union的注入（Union-Based Injection）

- 使用NULL找到列数量
  - ‘ UNION SELECT NULL--
  - ‘ UNION SELECT NULL, NULL--
  - ‘ UNION SELECT NULL, NULL, NULL--
- 使用Order By找到列数量
  - ‘ ORDER BY 1--
  - ‘ ORDER BY 2--
  - ‘ ORDER BY 3--
- 找到哪一列是字符串类型列
  - ‘ UNION SELECT ‘a’, NULL, NULL—
  - ‘ UNION SELECT NULL, ‘a’, NULL--
  - ‘ UNION SELECT NULL, NULL, ‘a’ --

基于Union的注入（Union-Based Injection）

col1=1 UNION ALL SELECT query--



基于错误的注入（Error-Based Injection）

- http://localhost:12587/Default.aspx?id=2 or x=1

永远不要将数据库错误信息在生产环境中返回！！！500类错误重定向！

推理注入

- 基于布尔的注入和基于时间的注入统称为推理注入
- 这是由于应用程序可能不返回对应数据，所以需要借助这种比较麻烦的办法

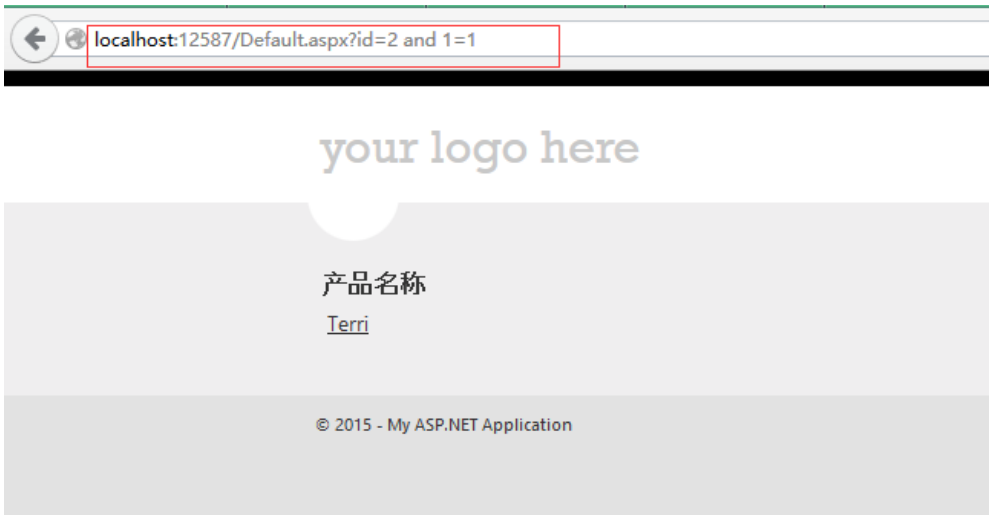
- 确定页面是否可注入
  - http://site/blog?message=5 AND 1=1
  - http://site/blog?message=5 AND 1=2

- 注入方式取决于想象力

盲目注入（Blind-Based Injection）

- 注入真假条件，根据返回内容的不同猜测结果
- 涉及大量的尝试，人工完成非常耗时，通常由工具进行注入
- 开多线程容易对服务器有损耗（一次某台易车DB被注入就是因为CPU报警后发现有人开大量线程注入）

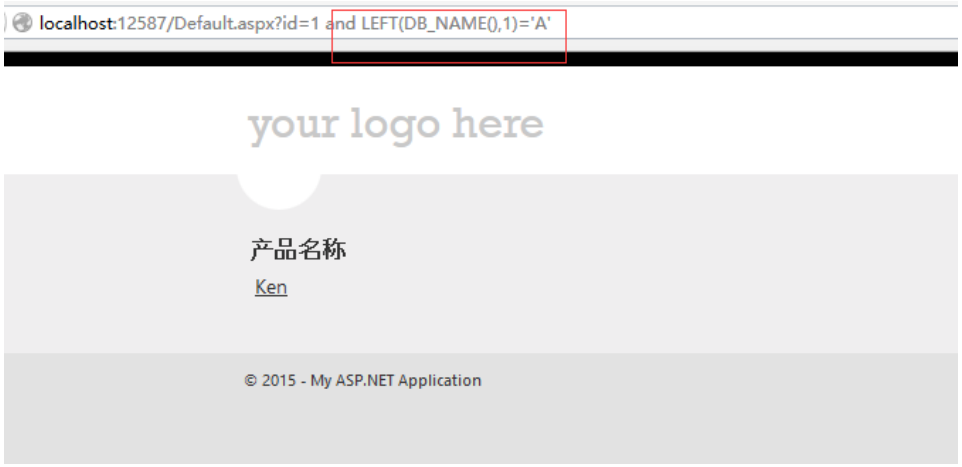
盲目注入（Blind-Based Injection）



根据输入条件返回内容完全不同！

盲目注入（Blind-Based Injection）

- 如何使用？



第一位为A,有返回值

前2位为Ad,有返回值

基于时间的注入（Time-based Injection）

- 在上述注入方式都无法奏效时，考虑使用基于时间的注入
- 基于时间的注入有时也被称为深度盲注
- 基本原理：

SELECT * FROM products WHERE id=1; WAIT FOR DELAY '00:00:15‘

SELECT * FROM products WHERE id=1; IF SYSTEM_USER='sa' WAIT FOR DELAY '00:00:15'

议程

- 什么是SQL注入
- SQL注入的种类
- SQL注入工具
- 如何防范SQL注入

注入工具SQLMAP介绍

- SqlMap
  - ***http://sql****map.org*
  - SQLMap是一个基于Python的开源测试工具，用于自动化检测和数据库控制工具。
- 使用说明：
  https://github.com/sqlmapproject/sqlmap/wiki/Usage

SQLMap基本注入方式介绍

- 帮助：sqlmap.py --help (Script guy必备技能)
- 基本Get方式注入
  - sqlmap.py -u “http://127.0.0.1/xx.aspx?category=1”
- 基本Post方式注入
  - sqlmap.py --data "username=xyz&password=xyz" -u http://127.0.0.1/xx.aspx
- 表单注入
  - sqlmap.py --forms -u "http://localhost:12587/Post.aspx"

SQLMap获取一些信息

- 当前用户：--current-user
- 当前数据库：--current-db
- 是否为管理员：--is-dba
- 数据库中所有用户：--users
- 列出所有数据库：--dbs
- 列出所有表：--tables
- 获取所有列：--columns

- 限制只扫描某库：***-D database_name***
- 限制只扫描某表：-T TableName

SQL注入方式选择

- --technique
  - B: Boolean-based blind
  - E: Error-based
  - U: Union query-based
  - S: Stacked queries
  - T: Time-based blind
  - Q: Inline queries
- 默认是所有
- 线程选择 --threads 5

议程

- 什么是SQL注入
- SQL注入的种类
- SQL注入工具
- 如何防范SQL注入

防范手段-简单检测

- 输入参数用单引号
  - 程序报错，则说明可注入
  - 程序未报错，则查看返回结果是否有变化
- 输入参数用两个单引号
  - 数据库中通常’ ’ 等同于一个’
  - 如果错误消失，则说明可注入

防范手段-减少Surface Area

- 数据库严禁暴漏在公网中（例：某银行信用卡系统）
- 应用程序从网络端限制不应该的请求

减少Surface Area？不仅仅是Web应用程序端…

任何录入数据库系统的数据（扫码枪、移动端、摄像头、物联网数据采集装置等等）都需要考虑对此进行防范。

防范手段-用户输入过滤1

- 不要相信用户输入的数据，不要拼接数据到最终执行的SQL中
- 需要转义的字符

输入字符	解释
;	查询分隔符
'	字符串分隔符
--	备注
/* ... */	多行备注
xp_	扩展存储过程，例如： **xp_cmdshell**.

- 例如：

```
private string SafeSqlLiteral(string inputSQL)
{
  return inputSQL.Replace("'", "''");
}
```

- 并不稳妥，比如where password=1 or 1=1,没有特殊符号！

防范手段-用户输入过滤2

- 但有些业务可能需要特殊符号，比如出版社O’Reilly
- 按照业务范围对输入值范围进行过滤（年龄在0到100，登录名中是否包含空格，等等）
- 很多SQL注入攻击依赖于长SQL，对输入长度进行限制

一个漫画

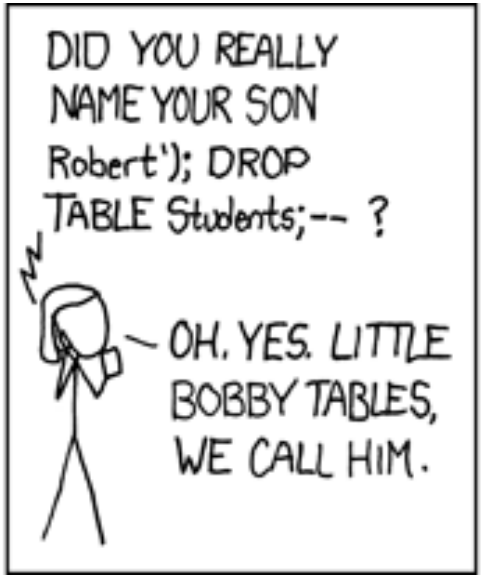
这是你儿子学校来的电话，我们的计算机有一点毛病

天呐，他又破坏什么了吗？

以某种方式吧

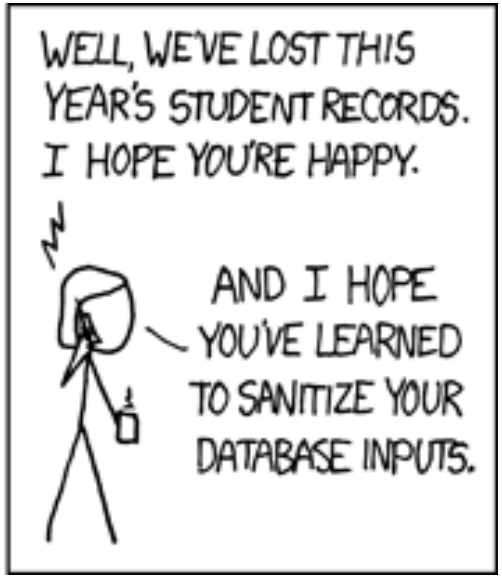
你真的给你儿子起名叫Robert’);drop table students;--

是的，我们叫他小名：bobby tables



好吧，我们丢了一年的学生记录

我希望你学会过滤一下你的数据库输入参数



防范手段-参数化查询

- 参数化查询可以防范SQL注入，C#中参数化

  command.Parameters.AddWithValue("@lastName", "John");

  对应SQL Server执行片段：

  EXEC sp_executesql N'SELECT * FROM dbo.member WHERE lastname LIKE @lastname', N'@lastname varchar(15)', ‘John‘

注：@lastname仅仅是一个容器和占位符，SQL Server知道该位置是数据，由于语句已经被预编译，只有在运行时才会将参数替换进去，所以无法将参数中的内容当作SQL执行。

- 如果不用这种方式，SQL Server则通过EXEC调用
- 参数化查询有可能对性能产生影响，但大多数情况利大于弊端

防范手段-权限

- 基本原则：不应赋予多于所需权限的权限
- 应用程序连接数据库帐号的正确姿势
  - 数据库的DataReader组
  - 数据库的DataWriter组
  - Grant Exec to [用户名]

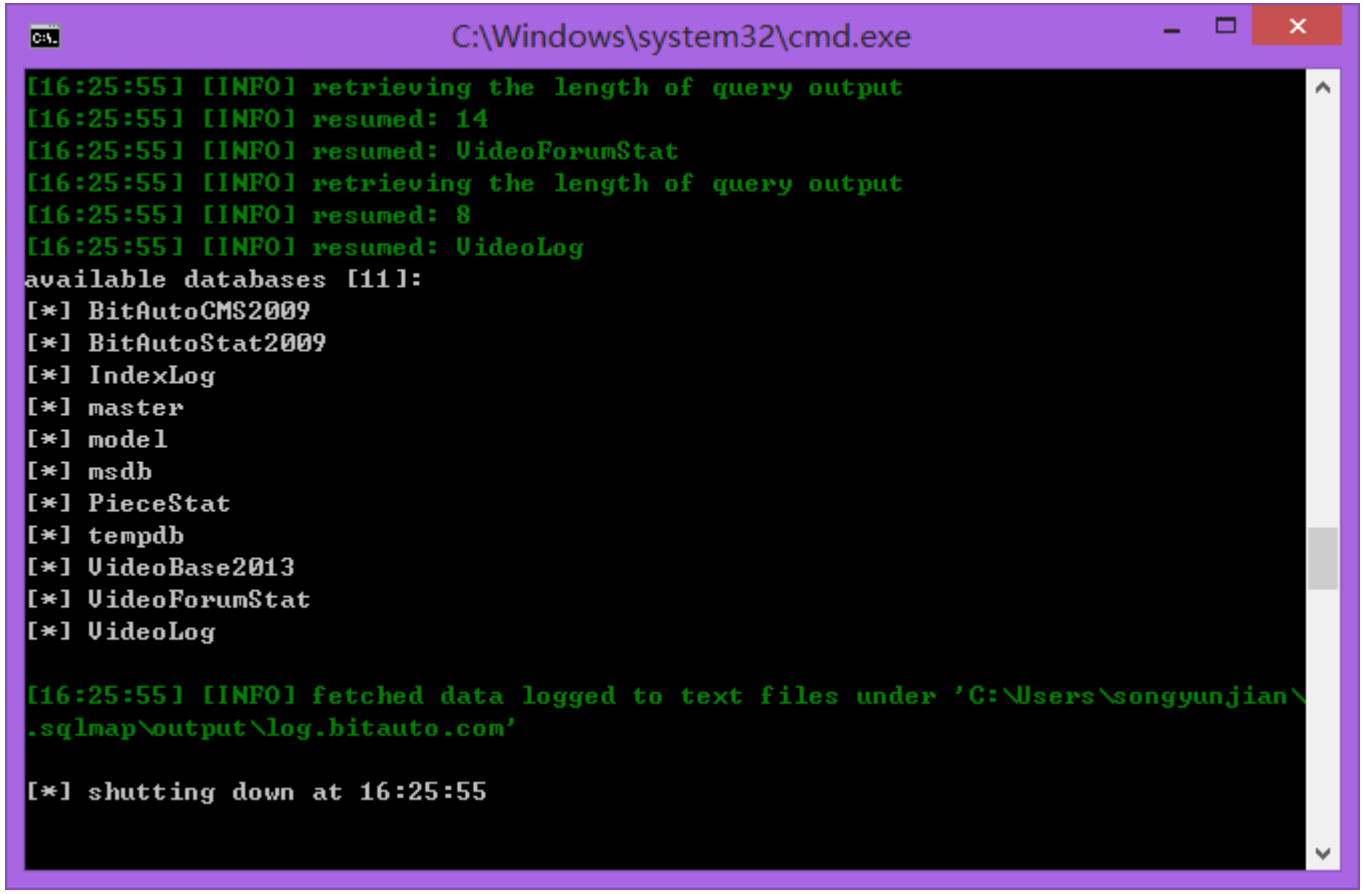
- 拒绝SA作为生产环境应用程序帐号
  - 不幸的是，90%以上的传统企业是这么做的。

防止SQL注入扫描表&数据库名称

- 所需在用户库禁用的应用程序用户权限
  - DENY SELECT ON sys.sysobjects TO [用户名]

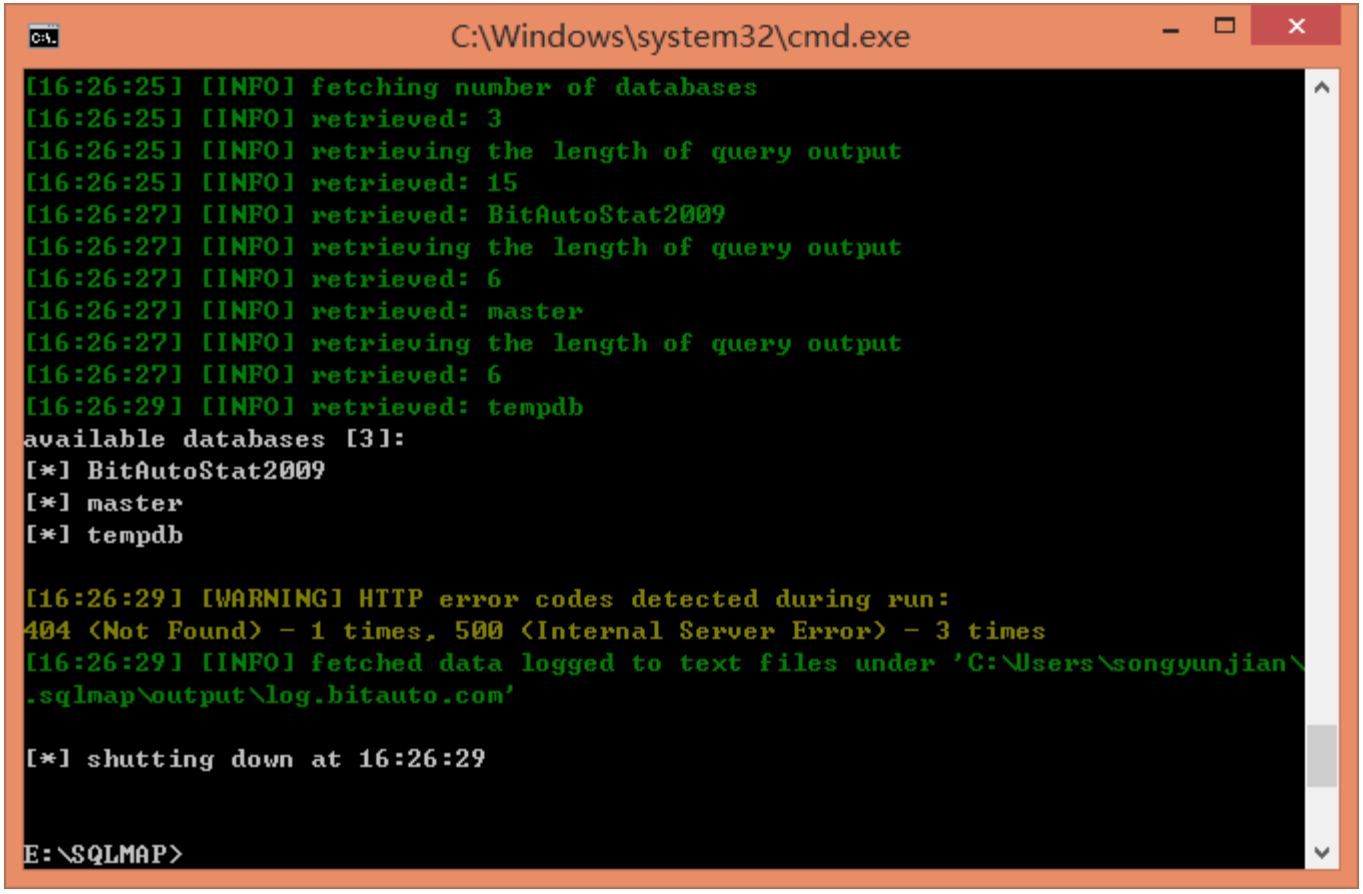
- 当前用户必须添加到Master库中，还需要在Master库中禁用的权限
  - DENY SELECT ON information_schema.tables TO [用户名]
  - DENY SELECT ON sys.sysobjects TO [用户名]

防止SQL注入扫描库名称效果（禁用前）

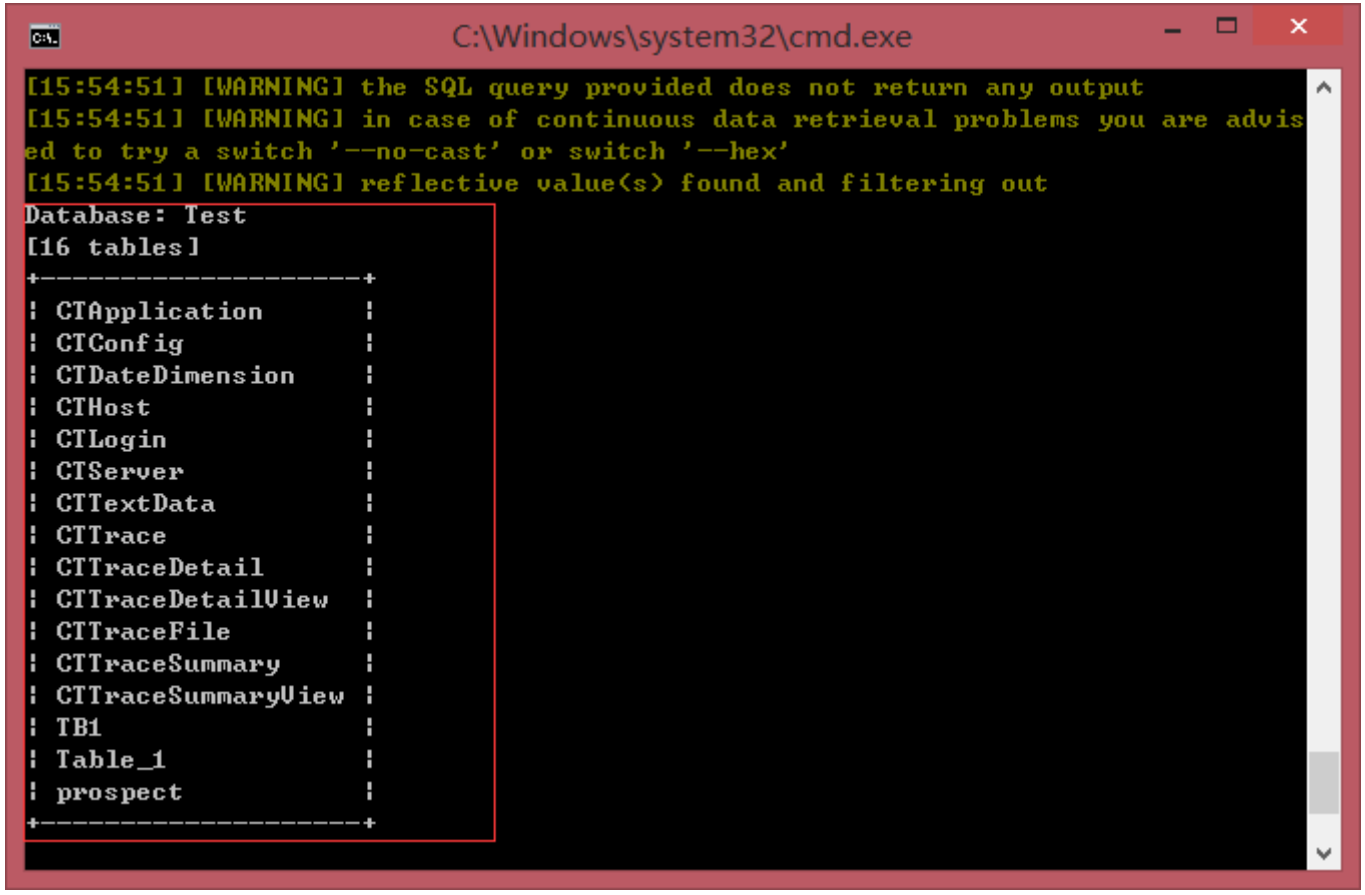


防止SQL注入扫描库名称效果（禁用后）

只能扫描到当前库，以及两个系统库

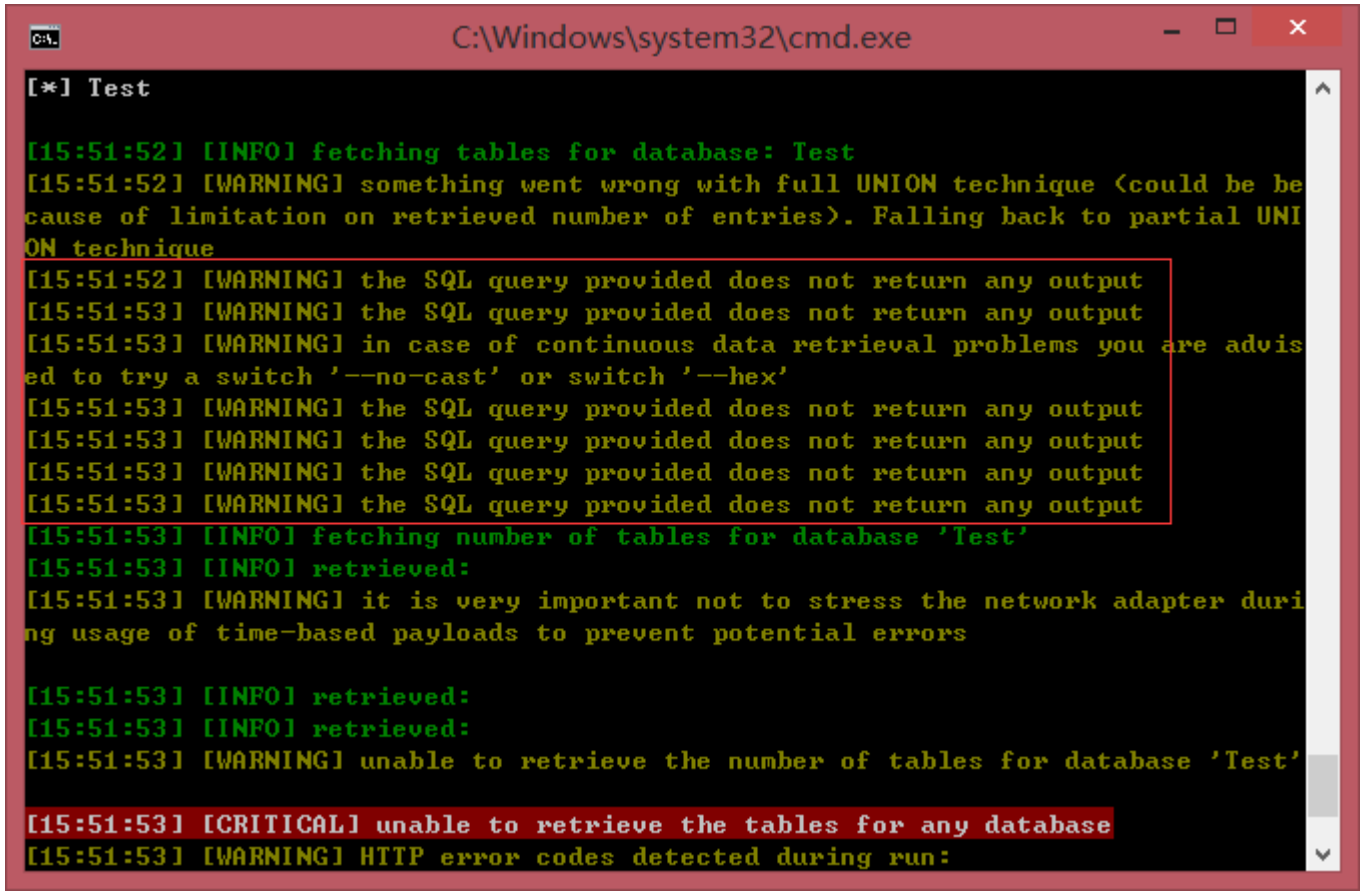


防止SQL注入扫描表名称效果（禁用前）



防止SQL注入扫描表名称效果（禁用后）

无法获取任何表信息



更激进的防范措施

- 具有成本！需要考虑Trade-Off

- 可以利用SQL TRACE或扩展事件对SQL报错进行记录(DEMO)
- 监控执行语句，对可疑语句进行捕捉
  - ;--

企业客服热线：4000-716-719

谢 谢！

易车网 部门
2013.1.16